KA680 CPU Module

Technical Manual

Order Number: EK-KA680-TM-001

First Edition, December 1991

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

This document was prepared using VAX DOCUMENT, Version 2.0.

Contents

4 KA680 Cache Memory Overview

5 KA680 Main Memory System

6 KA680 I/O Subsystem

7 The Console Line, TOY Clock

8 KA680 Boot and Diagnostic Facility

9 KA680 Q22–bus Interface

10 Network Interface

11 KA680 Mass Storage Interface

12 KA680 Firmware

E MOP Support

F Q22–bus Specification

G Specifications

Tables

Preface

The *KA680 CPU Module Technical Manual* documents the functional, physical, and environmental characteristics of the KA680 CPU module, and includes information on the MS690 memory expansion modules.

Organization

The manual is divided into three parts.

Overview and Installation

- Chapter 1, “Overview,” introduces the KA680 CPU module, the MS690 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,” describes the functions of the central processing unit.
- Chapter 4, “KA680 Cache Memory Overview,” describes the operation of the KA680 CPU module’s cache memory.
- Chapter 5, “KA680 Main Memory System,” describes the operation of the KA680 CPU module’s main memory.
- Chapter 6, “KA680 I/O Subsystem,” describes the I/O system configuration, along with the NCA chip architecture.
- Chapter 7, “The Console Line, TOY Clock,” describes the console serial line and the time-of-year clock. The chapter also provides an overview of the KA680 bus system.
- Chapter 8, “KA680 Boot and Diagnostic Facility,” describes the boot and diagnostic registers, EPROM memory, battery backed-up RAM and hardware initialization.
- Chapter 9, “KA680 Q22–bus Interface,” describes the interfaces the KA680 CPU module uses for the Q22–bus.
- Chapter 10, “ Network Interface,” describes the network interface of the KA680.
- Chapter 11, “ KA680 Mass Storage Interface,” describes the interfaces the KA680 CPU module uses for the mass storage bus.

Firmware

- Chapter 12, “KA680 Firmware,” describes the entry dispatch code, boot diagnostics, device booting sequence, console program, and console commands.

Appendices

- Appendix A, “NVRAM Partitioning,” describes how the KA680 firmware partitions the SSC 1 Kb battery backed-up (BBU) RAM.
- Appendix B, “Data Structures,” describes the global data structures used by the KA680 firmware.
- Appendix C, “ Error Messages ,” describes the error messages for the KA680, including machine check register dumps, halt codes, VMB error messages, and console error messages.
- Appendix D, “Machine State on Powerup,” describes the state of the KA680 after a power-up halt.
- Appendix E, “MOP Support,” describes the maintenance operation protocol (MOP) support features in the KA680 firmware.
- Appendix F, “Q22–bus Specification,” describes the specifications for the Q22–bus.
- Appendix G, “Specifications,” describes the physical, electrical, and environmental characteristics of the KA680 CPU module.
- Appendix H, “VAX Instruction Set,” is a list of the VAX instructions, provided for reference only.
- Appendix I, “Address Assignments,” provides a map of VAX memory space.
- Appendix J, “Configurable Machine State,” provides a list of all configurable bits in the CPU module that are left after the successful completion of power-up RAM diagnostics.

Conventions

The following table lists the conventions used in this manual.

Table 1 Conventions

Convention	Meaning
<x:y>	Represents a bit field, a set of lines, or signals, ranging from x through y. For example, R0 <7:4> Indicates bits 7 through 4 in a general-purpose register R0.
[x:y]	Represents a range of bits, from y through x.
Return	A label enclosed in a box represents a key (usually a control or a special character key) on the keyboard (in this case, the return key).
Note	Contains general information.
Caution	Contains information to prevent damage to equipment.
n	Indicates a variable.
[ ]	Represents a console command element that is optional.
{}	Represents a console command element.
...	Represents a list of command elements.
CPU	Refers to the NVAX central processor chip used in this design.

Related Documents

The following documents are related to the KA680 CPU.

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

You can order these documents 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

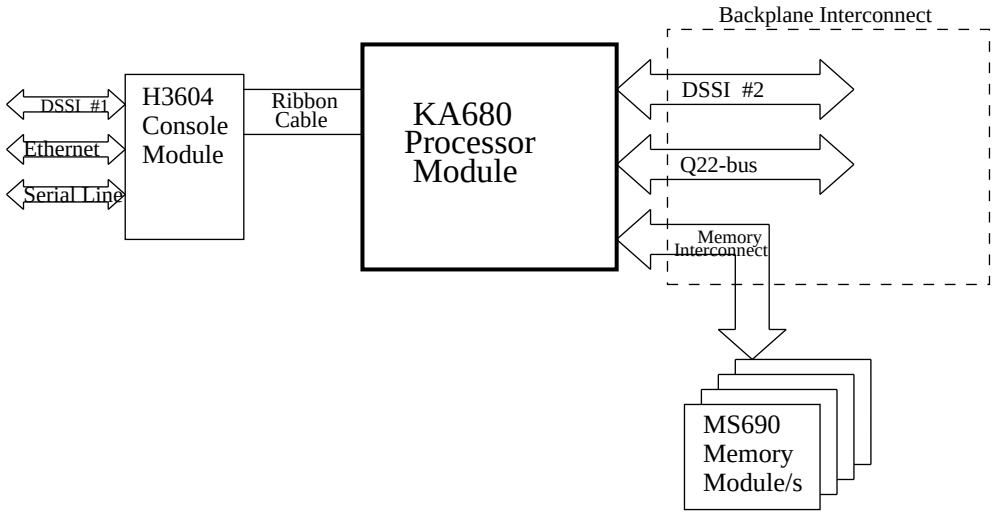
1
Overview

1.1 Introduction

This chapter briefly describes the KA680 CPU/memory subsystem.

The KA680 CPU module combines with the MS690 memory modules to form the CPU/memory subsystem for the VAX 4000-500 product. The subsystem is housed in the BA440 enclosure. The subsystem uses the DSSI bus to communicate with mass storage devices and the Q22–bus to communicate with I/O devices. A single KA680 CPU module can support up to four MS690 memory modules. Figure 1–1 is a block diagram of the CPU/memory subsystem's major functions.

Figure 1–1 KA680 Module in a System



The KA680 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 module uses multiple levels of cache memory to maximize performance. It is designed for use in high-speed, real-time applications and for multiuser, multitasking environments.

The KA680 and the MS690 designs are implemented in fingerless quad-height sized modules. Both use high-density, right angle connectors and mount in *dedicated* slots in the backplane. The CPU uses a 270-pin backplane connector while the memory module uses a 150-pin connector.

The CPU module communicates with the memory modules across a memory interconnect routed through the high-density connectors and the backplane. The backplane connector also connects the subsystem with the Q22–bus and one DSSI bus. There are no jumpers or switches to configure on the processor module. Fuses are located on the H3604 console module. The KA680 connects to the H3604 console module with a 100-pin ribbon cable. The console module contains configuration switches, Ethernet and DSSI connectors, and an LED display.

1.2 KA680 CPU Module

Figure 1–2 is a photograph of the KA680 CPU Module.

Figure 1–2 KA680 CPU Module

photograph of the MS690 memory module

The major hardware components of the KA680 CPU module are listed below. The chip identification numbers are shown in Figure 1–3.

• DC246	Central processor	(NVAX)
• Cache RAMs	A 128 KB backup cache	–
• DC243	NDAL to CDAL I/O bus interface chip	(NCA)
• DC244	Main memory controller, with ownership bit control	(NMC)
• DC527	Q22–bus interface	(CQBIC)
• DC541	Ethernet interface	(SGEC)
• DC542	DSSI interface chips (2)	(SHAC)
• DC511	System support chip	(SSC)
• DC509	Clock	(CCLK)
• Firmware ROMs (4)	512 KB; each 128 KB by 8, FLASH programmable	–
• Console Connection	100-pin ribbon cable to the H3604 console module	–
• Backplane Connection	270-pin ribbon cable to the backplane carrying signals for the Q22–bus, the DSSI bus, and the memory interconnect	–

Figure 1–3 shows the positions of the major chips on the KA680.

Figure 1–3 KA680 CPU Module Component Side

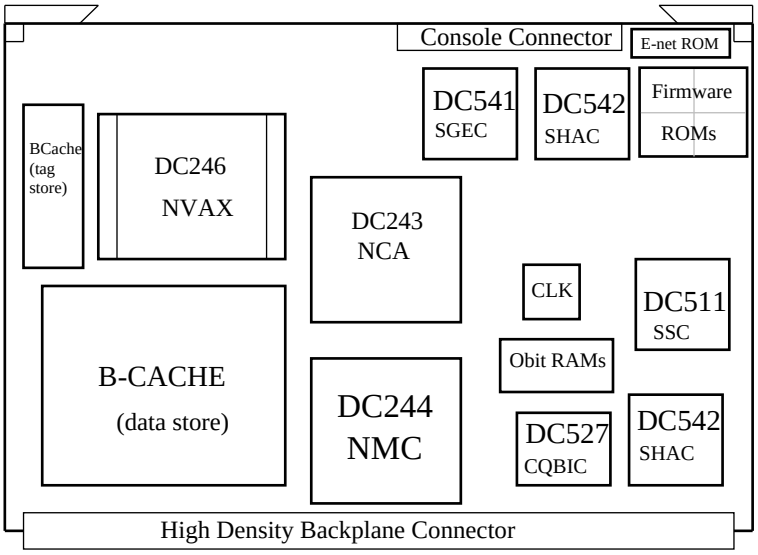
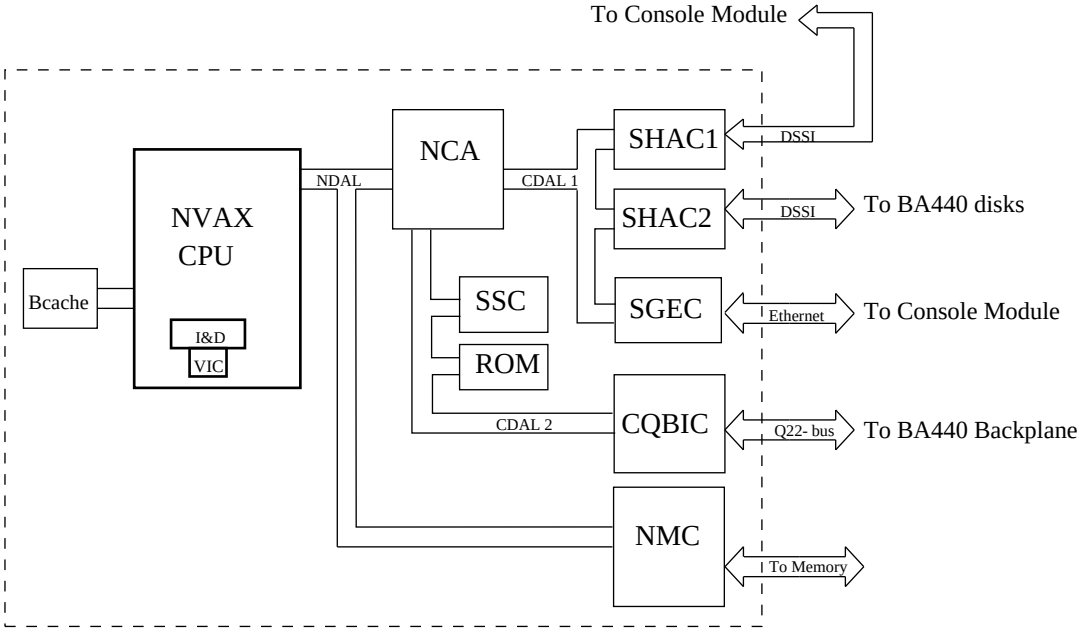


Figure 1–4 shows the major functional blocks of the KA680 CPU module.

Figure 1–4 KA680 CPU Module Block Diagram



Functionally, the KA680 CPU module is divided into four major areas:

- The central processing subsystem
- The system support subsystem
- The I/O subsystem
- The memory control subsystem

The H3604 is described in Section 1.4.

1.2.1 The Central Processing Subsystem

The central processing subsystem includes the NVAX CPU chip with an integral floating-point accelerator and a 3-level cache architecture. The RAM storage for the third level of cache (backup cache) is located on the CPU module.

1.2.1.1 The NVAX Central Processing Unit (DC246)

The NVAX CPU (DC246) chip is the heart of the KA680 module. It executes the VAX base instruction group as defined by the VAX Architecture Standard plus the optional VAX vector instructions and the virtual machine instructions. A complete listing of the instructions executed by the CPU is given in Appendix H. The NVAX processor also supports full VAX memory management with demand paging and a 4-gigabyte virtual address space.

For the rest of this document the NVAX CPU chip will be referred to simply as the "CPU," or "central processor," or "NVAX."

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

- CMPC3
- CMPC5
- LOCC
- MOVC3
- MOVC5
- 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
- F-floating
- G-floating
- D-floating

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

1.2.1.2 The Cache Memory

The KA680 processor module uses a 3-level cache architecture to maximize performance. The first level of cache, referred to as the *virtual instruction cache* (VIC), is 2 kilobytes (KB), and is located in the CPU chip. This cache handles instructions only (no data references), and deals only with virtual addresses. In this way, the CPU can obtain instruction information without the need for virtual to physical address translation, thereby decreasing latency and improving performance.

The second level of cache, referred to as the *primary cache* (Pcache), is 8 kilobytes in size and is located in the CPU chip. This cache implements a write-through instruction and data cache, and helps to reduce latency on access to instructions that are not found in the VIC. The Pcache uses physical addresses.

The third level of cache, referred to as the *backup cache* (Bcache), stores instruction and data, and is 128 KB in size. The Bcache is controlled by the Bcache controller located in the CPU chip. The data and tag store memory for this cache is located in SRAM chips on the KA680 module. The Bcache uses physical addresses.

1.2.2 The System Support Subsystem

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

1.2.2.1 The System Support Chip [SSC (DC511)]

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

- ROM address decoding
- 1 KB battery backed-up RAM
- Halt-arbitration logic
- Console serial line
- VAX standard time-of-year clock with battery backup
- IORESET register

- Programmable CDAL bus timeout
- Two programmable timers
- Register controlling the diagnostic LEDs

1.2.2.2 The Firmware ROMs

Resident firmware ROM is located on four chips, each 128K by 8 bits of FLASH* programmable EPROMs, for a total of 512 KB of ROM. The firmware gains control when the CPU halts, and contains programs that provide the following services:

- Board initialization
- Power-up self-testing of the KA680 and MS690 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 and DSSI devices
- Multilingual translation of key system messages

1.2.2.3 The Boot and Diagnostic Register

The boot and diagnostic register (BDR) allows the firmware and the operating system to read KA680 configuration bits.

1.2.2.4 The Station Address ROM

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

1.2.3 The I/O Subsystem

The I/O subsystem contains the following:

- CP-bus adapter
- DSSI mass storage interfaces
- An Ethernet interface
- A Q22–bus interface

1.2.3.1 NVAX CP-bus Bus Adapter [NCA (DC243)]

In order to provide buffering and connection to the I/O devices, the KA680 contains an NCA (DC243). The NCA provides an interface between the NVAX NDAL bus and two CP-buses where the the I/O device adapters reside. The CP-buses do not leave the KA680 module. As a bus adapter, the NCA controls transactions between the higher performance NDAL bus and the lower performance CP-buses. Each of the NCA's CP-bus ports provides a CVAX compatible peripheral bus for direct memory access (DMA) by peripheral devices. The NCA is in a 339-pin PGA package.

* A FLASH EPROM is a programmable read-only memory that uses electrical (bulk) erasure rather than ultraviolet erasure.

1.2.3.1.1 DSSI Mass Storage Interface [SHAC (DC542)] The two shared-host adapter chips (SHAC) implement the DSSI (Digital storage system interconnect) bus interfaces. One SHAC interfaces to the system console module while the other SHAC interfaces to the system backplane. The DSSI interface allows each DSSI bus on the KA680 to transmit packets of data to, and receive packets from, up to seven other DSSI devices. The SHAC facilitates scatter and gather mapping along with internal FIFO buffering, and features parity protection during data transfers to and from the CPU and main memory.

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. Controllers are built into the integrated storage elements (ISEs), enabling many functions to be handled without host or adapter intervention.

1.2.3.1.2 Ethernet Interface [SGEC (DC541)] The Ethernet interface handles communications between the CPU module and other nodes on the Ethernet. It is implemented with the second generation Ethernet controller chip (SGEC, DC541) on-board network interface. Used in connection with the H3604 console module, the SGEC allows the KA680 to connect to either a ThinWire or standard Ethernet. It supports the Ethernet Data Link Layer and provides CP-bus parity protection. The SGEC chip is in an 84-pin package. The chip facilitates scatter and gather mapping along with dual internal FIFO buffering.

1.2.3.1.3 Q22–bus Interface [CQBIC (DC527)] The KA680 includes a Q22–bus interface that allows communication between the KA680 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 KA680 does not provide Q22–bus termination; the backplane provides the termination resistors. The Q22–bus interface supports the following functions:

- Scatter/gather mapping functions
- Masked and unmasked longword reads and writes from CPU to the Q22–bus memory and I/O space and the CQBIC 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

1.2.4 The Memory Control Subsystem

This subsystem provides support for the KA680 memory subsystem. A key feature of the KA680 memory subsystem is the use of ownership bits to maintain a sense of ownership over each hexaword (32 bytes) of main memory. This ownership mechanism serves the dual function of maintaining coherency between main memory and the NVAX cache memory, as well as providing a secure interlock mechanism for synchronization between NVAX and the I/O devices.

1.2.4.1 NVAX Memory Controller [NMC (DC244)]

The memory controller is implemented by the NVAX memory controller chip (DC244). The NMC is an ECC[1] memory controller. The NMC controls transactions between the main memory and the NVAX, and between main memory and any of the I/O devices (CQBIC, SGEC, and the two SHAC chips). In addition, the NMC has a key role in maintaining main memory coherency with the NVAX primary and backup caches through the use of ownership bits.

The NMC interfaces the NVAX and I/O subsystem with up to four memory modules. The NMC controls access to shared memory locations through the use of the ownership bits, thereby providing a reliable interlock mechanism for memory that is shared between the NVAX and the I/O devices.

1.3 MS690 Memory Module

The MS690 is a double-sided memory board, in a 72-bitwide array (64-bit data and 8-bit error correction code). Two versions will be offered; one implemented with 1 Mb DRAM chips and another version using 4 Mb DRAM chips. The version using 1 Mb DRAMs will contain 32 MB per module, and the version using 4 Mb DRAMs will contain 64 MB or 128 MB per module.

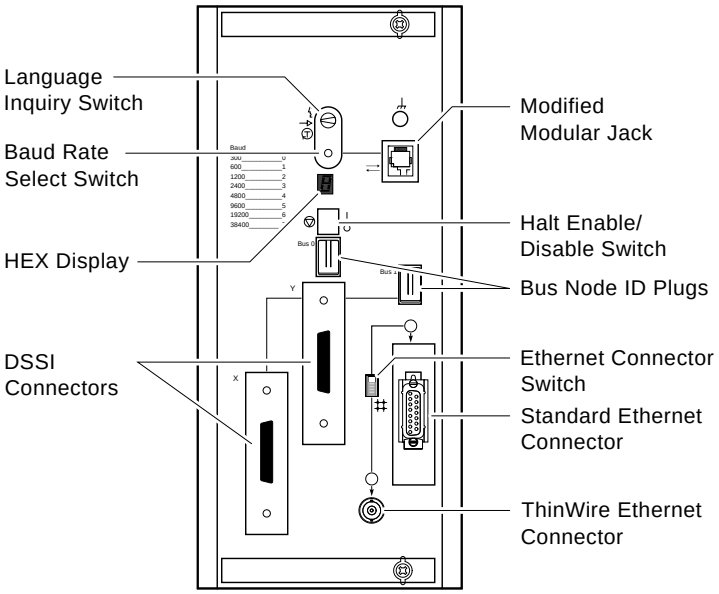
The module mounts in a dedicated memory backplane slot. It is fingerless and uses a 150-pin high density, right angle connector to connect to the backplane.

1.4 H3604 Console Module

The H3604 console module (Figure 1–5) allows the KA680 CPU module to interface to a serial line console device, a DSSI bus, and to the Ethernet. The H3604 module is wide enough to cover the five slots dedicated to the KA680 and its four MS690 modules. Five adhesive tags are included for the user to name the modules in the respective slots.

[1] ECC stands for error checking and correction.

Figure 1–5 H3604 Console Module Front



The H3604 contains the following connectors to allow CPU communication:

- A console serial line with baud rate switch
- Two Ethernet connectors, with a switch to choose between them
- Two DSSI connectors (50-pin high density) that allow daisy chaining of one DSSI bus; terminators for both DSSI connectors; and two DSSI BUS ID select plugs

The H3604 also has four feature selection switches:

- Baud rate select switch for the serial console line.
- Power-up mode switch.
- Break enable/disable switch. This determines whether the CPU may be "halted", causing execution to jump to the halt restart firmware. If halts are enabled, then the CPU may be halted from the console keyboard in one of two ways, depending on the setting of bit 15 of the SSC configuration register (SSCCR<15>). Refer to Section 12.2.2 for more information regarding halts. If this switch is set to the enable position (1), the system does not autoboot on powerup. It enters console I/O mode and displays the >>> prompt.
- Ethernet connector switch to select the following:
  - A 15-conductor connector for standard Thickwire 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 contains the following features:

- Console serial line drivers and receivers
- Hexadecimal display
- Battery charger and low-voltage detect
- 25.6 KHz TOY clock oscillator
- -9 V dc/dc converter
- Ethernet serial interface adapter chip (SIA)
- Fused current surge protection

The door of the H3604 contains a DSSI circuit fuse and two plugs. The fuse is to protect against shorts from the accidental grounding of the DSSI cable power pin. The external node ID plugs are used to select the host adapter numbers. Node number 7 is preset 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 to the internal BA440. One is a 4-pin power connection to a small printed circuit card that inserts into the backplane alongside the KA680. The other is the 100-pin connector to the KA680 CPU module.

2
Installation and Configuration

This chapter describes how to install the KA680 in a system. It discusses the following topics:

- Installing the KA680 and MS690 modules
- Configuring the KA680
- KA680 connectors

2.1 Installing the KA680 and MS690 Memory Modules

Note

You can use the KA680 and MS690 modules only in BA440 system enclosures that use high-density backplane connector slots.

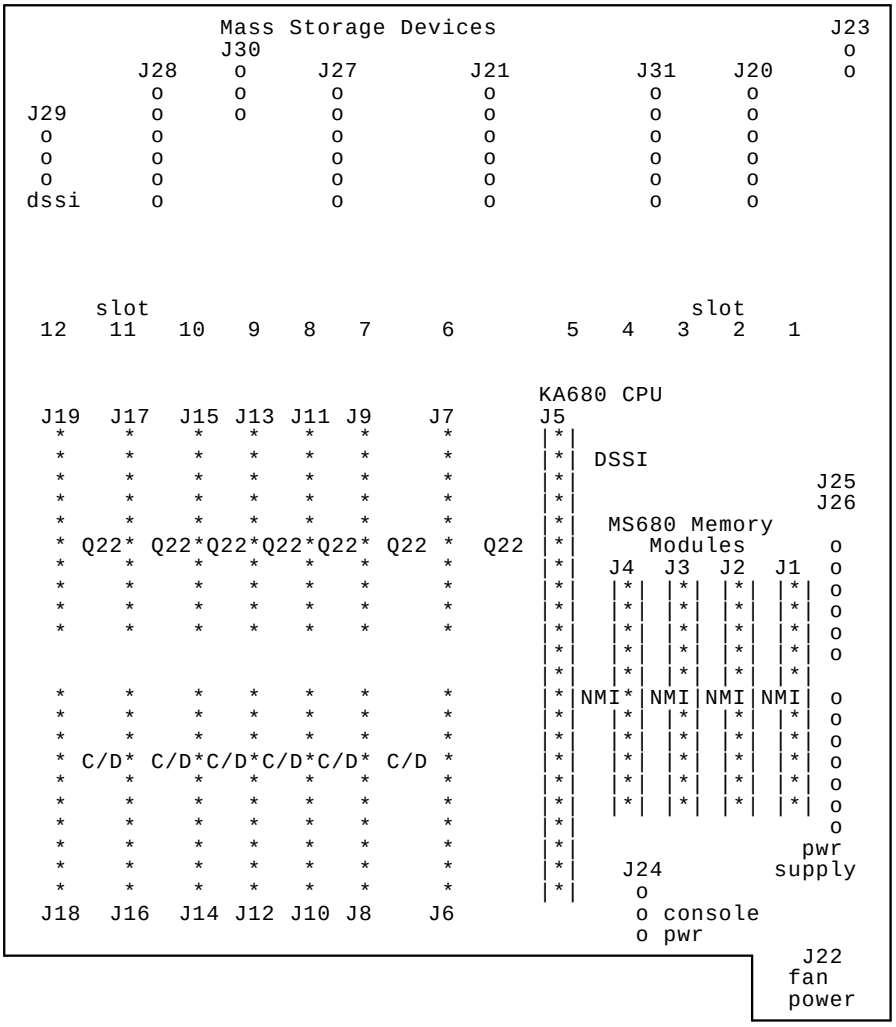
The KA680 CPU module and the MS690 memory modules must be installed in the five rightmost backplane slots. Note that the KA680 module installs in backplane slot J5, and the memory modules install in slots J4 through J1.

To install the KA680 and MS690 modules:

1. Install the KA680 CPU in slot J5 of the backplane.
2. Install MS690 memory modules in slots J4 through J1 next to to the KA680 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 KA680 CPU and the console module.

Figure 2–1 shows the positions of the KA680 CPU and the memory modules in the backplane.

Figure 2–1 Backplane



2.2 Module Configuration and Naming

Each Q22–bus 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 CONFIG 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 CONFIG 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 *KA680 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 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 part 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 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 on board the controller module. This node name is determined in manufacturing from an algorithm based on the drive serial number. You can change the node name of the DSSI device to something more meaningful by following the procedure in Example 2–1. In the example, the node name for the ISE at DSSI node address 1 is changed from R3YBNE to DATADISK.

Example 2–1 Changing a DSSI Node Name

```
>>> 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 (*)
```

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 1 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.

Example 2–2 Changing a DSSI Unit Number

```
>>> 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)
```

(continued on next page)

Example 2–2 (Cont.) Changing a DSSI Unit Number

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

DSSI Node 7 (*)
```

2.3.3 Accessing RF-series Firmware in VMS Through DUP

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

Use the VMS operating system to access the ISE firmware if you want to look up or view parameter settings, but not change them. To change ISE parameter settings, enter the ISE firmware through the console I/O mode SET HOST/DUP command.

Load the FYDRIVER using the following commands in SYSGEN:

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

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

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

2.3.3.1 Allocation Class

When a KA680 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 cable connects the DSSI bus to other devices, either hosts or expanders.

There are two DSSI ports in the KA680 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 KA680. The near-end termination is contained on the backplane for the internal DSSI bus, and is 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 node ID plugs (Figure 1–5). These ID plugs provide an identity for each DSSI bus. Because the DSSI controllers implemented by the SHAC chips on the KA680 CPU module are separate, the two ID plugs may be identical.

2.5 KA680 Connectors

The KA680 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 KA680 through the H3604 console module. Figure 1–3 shows the location of the connectors on the KA680 module.

3
Central Processor

The central processor of the KA680 supports the MicroVAX chip subset (plus six additional string instructions) of the VAX instruction set, and datatypes and full VAX memory management. It is implemented with a single VLSI chip called the NVAX (DC246).

3.1 Processor State

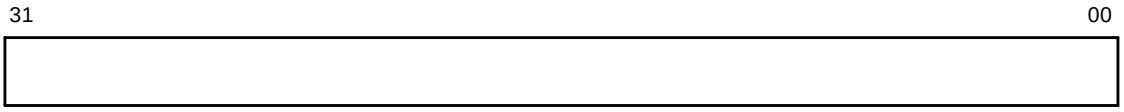
The **processor state** consists of that portion of the state of a process stored in processor registers rather than in memory. The processor state is composed of sixteen general-purpose registers (GPRs), the processor status longword (PSL), and the internal processor registers (IPRs).

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

3.1.1 General-Purpose Registers

The KA680 implements 16 general-purpose registers. as implemented per the *VAX Architecture Reference Manual* These registers are used for temporary storage, accumulators, and base and index registers for addressing. These registers are denoted R0—R15. The bits of a register are numbered from the right <0> through <31>. Figure 3–1 shows the general-purpose register format. Table 3–1 describes the registers.

Figure 3–1 General-Purpose Register



LJ-01323-TI0

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

Table 3–1 General-Purpose Register Description

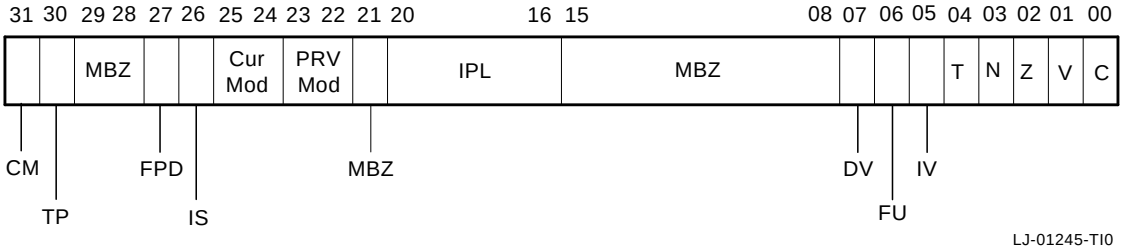
Register	Register Name	Mnemonic	Description
R15	Program Counter	PC	The PC contains the address of the next instruction byte of the program.
R14	Stack Pointer	SP	The SP contains the address of the top of the processor defined stack.
R13	Frame Pointer	FP	The VAX–11 procedure call convention builds a data 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 called 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.

3.1.2 Processor Status Longword

The KA680 processor status longword (PSL) is implemented per the *VAX Architecture Reference Manual*, which should be consulted for a detailed description of the operation of this register. The PSL is saved on the stack when an exception or interrupt occurs and is saved in the process control block (PCB) on a process context switch. Bits <15:00> may be accessed by nonprivileged software, while bits <31:16> may only be accessed by privileged software. Processor initialization sets the PSL to 041F 0000_{16}. Figure 3–2 shows the processor status longword format. 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 Definitions

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 KA680 hardware; it 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.

3.1.3 Internal Processor Registers

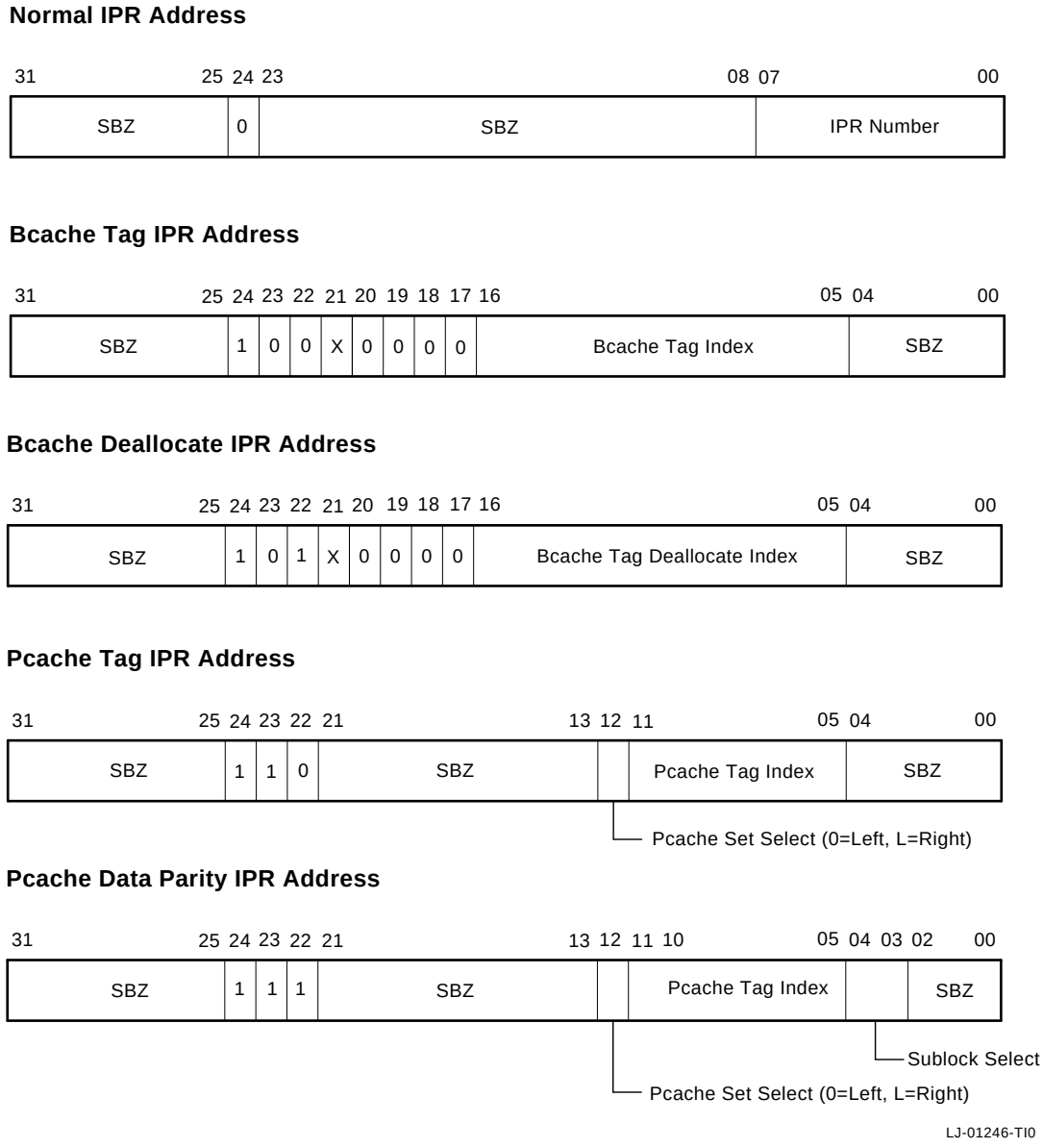
The processor registers that are implemented by the NVAX CPU chip, and those that are required of the system environment, are logically divided into five groups, as follows:

- Normal—Those IPRs that address individual registers in the NVAX CPU chip or system environment.
- Bcache tag IPRs—The read/write block of IPRs that allow direct access to the Bcache tags.
- Bcache deallocate IPRs—The write-only block of IPRs by which a Bcache block may be deallocated.
- Pcache tag IPRs—The read/write block of IPRs that allow direct access to the Pcache tags.

- Pcache data parity IPRs—The read/write block of IPRs that allow direct access to the Pcache data parity bits.

Each group of IPRs is distinguished by a particular pattern of bits in the IPR address, as shown in Figure 3–3.

Figure 3–3 IPR Address Space Decoding



The numeric range for each of the four groups is shown in Table 3–3.

Table 3–3 IPR Address Space Decoding

IPR Group	Mnemonic[1]	IPR Address Range (hex)	Contents
Normal	–	00000000..000000FF[2]	256 individual IPRs
Bcache Tag	BCTAG	01000000..0107FFE0[2]	16K Bcache tag IPRs, each separated by 20(hex) from the previous one
Bcache Deallocate	BCFLUSH	01400000..0147FFE0[2]	16K Bcache tag deallocate IPRs, each separated by 20(hex) from the previous one
Pcache Tag	PCTAG	01800000..01801FE0[2]	256 Pcache tag IPRs, 128 for each Pcache set, each separated by 20(hex) from the previous one
Pcache Data Parity	PCDAP	01C00000..01C01FF8[2]	1024 Pcache data parity IPRs, 512 for each Pcache set, each separated by 8(hex) from the previous one

[1]The mnemonic is for the first IPR in the block.

[2]Unused fields in the IPR addresses for these groups should be zero. Neither hardware nor microcode detects and faults on an address in which these bits are nonzero. Although noncontiguous address ranges are shown for these groups, the entire IPR address space maps into one of these groups. If these fields are nonzero, the operation of the CPU is UNDEFINED.

Note

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

Because of the sparse addressing used for IPRs in groups other than the normal group, valid IPR addresses are not separated by one. Rather, valid IPR addresses are separated by either 8 or 20(hex). For example, the IPR address for Bcache tag 0 is 01000000 (hex), and the IPR address for Bcache tag 1 is 01000020 (hex). In this group, bits <4:0> of the IPR address are ignored, so IPR numbers 01000001 through 0100001F all address Bcache tag 0. Similarly, the IPR address for the first subblock of Pcache data parity is 01C00000 (hex), and the IPR address for the second subblock of Pcache data parity is 01C00008 (hex).

Processor registers in all groups except the normal group are processed entirely by the NVAX CPU chip and will never appear on the NDAL (Section 3.11). This is also true for a number of the IPRs in the normal group. IPRs in the normal group that are not processed by the NVAX CPU chip are converted into I/O space references and passed to the system environment via a read or write command on the NDAL.

Each of the 256 possible IPRs in the normal group are of longword length, so a 1 KB block of I/O space is required to convert each possible IPR to a unique I/O space longword. This block starts at address E1000000 (hex). Conversion of an IPR address to an I/O space address in this block is done by shifting the IPR address left into bits <9:2>, filling bits <1:0> with zeros, and merging in the base address of the block. This can be expressed by the equation

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

The actual hardware implementation of this is different in that the IPR number is shifted left by 2 bits, and bits <31:29,24> are set. There is no multiplying or adding done as one might conclude from the equation.

Because many of the 256 possible IPRs in the normal group are processed entirely by the NVAX CPU chip, the corresponding I/O space location in the 1 KB block is never referenced as a result of an MTPR/MFPR to or from these IPRs. However, note that a programmer can indeed reference these locations via an explicit I/O space reference (for example, MOVL). References to this block of I/O space locations with instructions other than MTPR/MFPR may result in UNDEFINED behavior.

The processor registers implemented by the NVAX CPU are shown in Table 3–4.

Note

Many of the processor registers listed in Table 3–4 are used internally by the microcode during normal operation of the CPU, and are not intended to be referenced by software except during test or diagnosis of the system. These registers are flagged with the notation “Testability and diagnostic use only; not for software use in normal operation.” References by software to these registers during normal operation can cause UNDEFINED behavior of the CPU.

Table 3–4 Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Kernel Stack Pointer	KSP	0	0	RW	NVAX	1-1	–
Executive Stack Pointer	ESP	1	1	RW	NVAX	1-1	–
Supervisor Stack Pointer	SSP	2	2	RW	NVAX	1-1	–
User Stack Pointer	USP	3	3	RW	NVAX	1-1	–
Interrupt Stack Pointer	ISP	4	4	RW	NVAX	1-1	–
Reserved	–	5	5	–	–	3	E1000014
Reserved	–	6	6	–	–	3	E1000018
Reserved	–	7	7	–	–	3	E100001C
P0 Base Register	P0BR	8	8	RW	NVAX	1-2	–
P0 Length Register	P0LR	9	9	RW	NVAX	1-2	–
P1 Base Register	P1BR	10	A	RW	NVAX	1-2	–
P1 Length Register	P1LR	11	B	RW	NVAX	1-2	–
System Base Register	SBR	12	C	RW	NVAX	1-2	–
System Length Register	SLR	13	D	RW	NVAX	1-2	–
CPU Identification	CPUID	14	E	RW	NVAX	2-1	–
Reserved	–	15	F	–	–	3	E100003C
Process Control Block Base	PCBB	16	10	RW	NVAX	1-1	–
System Control Block Base	SCBB	17	11	RW	NVAX	1-1	–

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Interrupt Priority Level[1]	IPL	18	12	RW	NVAX	1-1	–
AST Level[1]	ASTLVL	19	13	RW	NVAX	1-1	–
Software Interrupt Request Register	SIRR	20	14	W	NVAX	1-1	–
Software Interrupt Summary Register[1]	SISR	21	15	RW	NVAX	1-1	–
Reserved	–	22	16	–	–	3	E1000058
Reserved	–	23	17	–	–	3	E100005C
Interval Counter Control/Status	ICCS	24	18	RW	NCA	2-7	E1000060
Next Interval Count	NICR	25	19	RW	NCA	3-7	E1000064
Interval Count	ICR	26	1A	RW	NCA	3-7	E1000068
Time of Year Register	TODR	27	1B	RW	SSC	2-3	E100006C
Console Storage Receiver Status	CSRS	28	1C	RW	SSC	2-3	E1000070
Console Storage Receiver Data	CSRD	29	1D	R	SSC	2-3	E1000074

[1]Initialized on reset

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Console Storage Transmitter Status	CSTS	30	1E	RW	SSC	2-3	E1000078
Console Storage Transmitter Data	CSTD	31	1F	W	SSC	2-3	E100007C
Console Receiver Control/Status	RXCS	32	20	RW	SSC	2-3	E1000080
Console Receiver Data Buffer	RXDB	33	21	R	SSC	2-3	E1000084
Console Transmitter Control/Status	TXCS	34	22	RW	SSC	2-3	E1000088
Console Transmitter Data Buffer	TXDB	35	23	W	SSC	2-3	E100008C
Reserved	–	36	24	–	–	3	E1000090
Reserved	–	37	25	–	–	3	E1000094
Machine Check Error Register	MCESR	38	26	W	NVAX	2-1	–
Reserved	–	39	27	–	–	3	E100009C
Reserved	–	40	28	–	–	3	E10000A0
Reserved	–	41	29	–	–	3	E10000A4
Console Saved PC	SAVPC	42	2A	R	NVAX	2-1	–
Console Saved PSL	SAVPSL	43	2B	R	NVAX	2-1	–

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved	–	44	2C	–	–	3	E10000B0
Reserved	–	45	2D	–	–	3	E10000B4
Reserved	–	46	2E	–	–	3	E10000B8
Reserved	–	47	2F	–	–	3	E10000BC
Reserved	–	48	30	–	–	3	E10000C0
Reserved	–	49	31	–	–	3	E10000C4
Reserved	–	50	32	–	–	3	E10000C8
Reserved	–	51	33	–	–	3	E10000CC
Reserved	–	52	34	–	–	3	E10000D0
Reserved	–	53	35	–	–	3	E10000D4
Reserved	–	54	36	–	–	3	E10000D8
I/O System Reset Register	IORESET	55	37	W	SSC	2-3	E10000DC
Memory Management Enable[1,3]	MAPEN	56	38	RW	NVAX	1-2	–
Translation Buffer Invalidate All[3]	TBIA	57	39	W	NVAX	1-1	–
Translation Buffer Invalidate Single[3]	TBIS	58	3A	W	NVAX	1-1	–
Reserved	–	59	3B	–	–	3	E10000EC
Reserved	–	60	3C	–	–	3	E10000F0

[1]Initialized on reset

[3]Change broadcast to vector unit if present

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
System Identification	SID	62	3E	R	NVAX	1-1	–
Translation Buffer Check	TBCHK	63	3F	W	NVAX	1-1	–
IPL 14 Interrupt ACK[5]	IAK14	64	40	R	SSC	2-3	E1000100
IPL 15 Interrupt ACK[5]	IAK15	65	41	R	SSC	2-3	E1000104
IPL 16 Interrupt ACK[5]	IAK16	66	42	R	SSC	2-3	E1000108
IPL 17 Interrupt ACK[5]	IAK17	67	43	R	SSC	2-3	E100010C
Clear Write Buffer[5]	CWB	68	44	RW	SSC	2-3	E1000110
Reserved	–	69	45	–	–	3	E1000114
Reserved	–	70	46	–	–	3	E1000118
Reserved	–	71	47	–	–	3	E100011C

[5]Testability and diagnostic use only; not for software use in normal operation

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved	–	72	48	–	–	3	E1000120
Reserved	–	73	49	–	–	3	E1000124
Reserved	–	74	4A	–	–	3	E1000128
Reserved	–	75	4B	–	–	3	E100012C
Reserved	–	76	4C	–	–	3	E1000130
Reserved	–	77	4D	–	–	3	E1000134
Reserved	–	78	4E	–	–	3	E1000138
Reserved	–	79	4F	–	–	3	E100013C
Reserved	–	80	50	–	–	3	E1000140
Reserved	–	81	51	–	–	3	E1000144
Reserved	–	82	52	–	–	3	E1000148
Reserved	–	83	53	–	–	3	E100014C
Reserved	–	84	54	–	–	3	E1000150
Reserved	–	85	55	–	–	3	E1000154

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved	–	86	56	–	–	3	E1000158
Reserved	–	87	57	–	–	3	E100015C
Reserved	–	88	58	–	–	3	E1000160
Reserved	–	89	59	–	–	3	E1000164
Reserved	–	90	5A	–	–	3	E1000168
Reserved	–	91	5B	–	–	3	E100016C
Reserved	–	92	5C	–	–	3	E1000170
Reserved	–	93	5D	–	–	3	E1000174
Reserved	–	94	5E	–	–	3	E1000178
Reserved	–	95	5F	–	–	3	E100017C
Reserved	–	96	60	–	–	3	E1000180
Reserved	–	97	61	–	–	3	E1000184
Reserved	–	98	62	–	–	3	E1000188
Reserved	–	99	63	–	–	3	E100018C

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved for VM	–	100	64	–	–	3	E1000190
Reserved for VM	–	101	65	–	–	3	E1000194
Reserved for VM	–	102	66	–	–	3	E1000198
Reserved	–	103	67	–	–	3	E100019C
Reserved	–	104	68	–	–	3	E10001A0
Reserved	–	105	69	–	–	3	E10001A4
Reserved	–	106	6A	–	–	3	E10001A8
Reserved	–	107	6B	–	–	3	E10001AC
Reserved	–	108	6C	–	–	3	E10001B0
Reserved	–	109	6D	–	–	3	E10001B4
Reserved	–	110	6E	–	–	3	E10001B8
Reserved	–	111	6F	–	–	3	E10001BC
Reserved	–	112	70	–	–	3	E10001C0
Reserved	–	113	71	–	–	3	E10001C4
Reserved	–	114	72	–	–	3	E10001C8
Reserved	–	115	73	–	–	3	E10001CC
Reserved	–	116	74	–	–	3	E10001D0
Reserved	–	117	75	–	–	3	E10001D4

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved	–	118	76	–	–	3	E10001D8
Reserved	–	119	77	–	–	3	E10001DC
Reserved	–	120	78	–	–	3	E10001E0
Reserved	–	121	79	–	–	3	E10001E4
Interrupt System Status Register	INTSYS	122	7A	RW	NVAX	2-1	–
Performance Monitoring Facility Count	PMFCNT	123	7B	RW	NVAX	2-1	–
Patchable Control Store Control Register	PCSCR	124	7C	WO	NVAX	2-1	–
Ebox Control Register	ECR	125	7D	RW	NVAX	2-1	–
Mbox TB Tag Fill[5]	MTBTAG	126	7E	W	NVAX	2-1	–
Mbox TB PTE Fill[5]	MTBPTE	127	7F	W	NVAX	2-1	–
Cbox Control Register	CCTL	160	A0	RW	NVAX	2-5	–
Reserved	–	161	A1	–	NVAX	2-6	–
Bcache Data ECC	BCDECC	162	A2	W	NVAX	2-5	–
Bcache Error Tag Status	BCETSTS	163	A3	RW	NVAX	2-5	–
Bcache Error Tag Index	BCETIDX	164	A4	R	NVAX	2-5	–

[5]Testability and diagnostic use only; not for software use in normal operation

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	(Hex)	Type	Impl	Cat	I/O Address
Bcache Error Tag	BCETAG	165	A5	R	NVAX	2-5	–
Bcache Error Data Status	BCEDSTS	166	A6	RW	NVAX	2-5	–
Bcache Error Data Index	BCEDIDX	167	A7	R	NVAX	2-5	–
Bcache Error ECC	BCEDECC	168	A8	R	NVAX	2-5	–
Reserved	–	169	A9	–	NVAX	2-6	–
Reserved	–	170	AA	–	NVAX	2-6	–
Fill Error Address	CEFADR	171	AB	R	NVAX	2-5	–
Fill Error Status	CEFSTS	172	AC	RW	NVAX	2-5	–
Reserved	–	173	AD	–	NVAX	2-6	–
NDAL Error Status	NESTS	174	AE	RW	NVAX	2-5	–
Reserved	–	175	AF	–	NVAX	2-6	–
NDAL Error Output Address	NEOADR	176	B0	R	NVAX	2-5	–
Reserved	–	177	B1	–	NVAX	2-6	–
NDAL Error Output Command	NEOCMD	178	B2	R	NVAX	2-5	–
Reserved	–	179	B3	–	NVAX	2-6	–

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
NDAL Error Data High	NEDATHI	180	B4	R	NVAX	2-5	–
Reserved	–	181	B5	–	NVAX	2-6	–
NDAL Error Data Low	NEDATLO	182	B6	R	NVAX	2-5	–
Reserved	–	183	B7	–	NVAX	2-6	–
NDAL Error Input Command	NEICMD	184	B8	R	NVAX	2-5	–
Reserved	–	185	B9	–	NVAX	2-6	–
Reserved	–	186	BA	–	NVAX	2-6	–
Reserved	–	187	BB	–	NVAX	2-6	–
Reserved	–	188	BC	–	NVAX	2-6	–
Reserved	–	189	BD	–	NVAX	2-6	–
Reserved	–	190	BE	–	NVAX	2-6	–
Reserved	–	191	BF	–	NVAX	2-6	–
Reserved	–	192	C0	–	–	3	E1000300
Reserved	–	193	C1	–	–	3	E1000304
Reserved	–	194	C2	–	–	3	E1000308
Reserved	–	195	C3	–	–	3	E100030C

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved	–	196	C4	–	–	3	E1000310
Reserved	–	197	C5	–	–	3	E1000314
Reserved	–	198	C6	–	–	3	E1000318
Reserved	–	199	C7	–	–	3	E100031C
Reserved	–	200	C8	–	–	3	E1000320
Reserved	–	201	C9	–	–	3	E1000324
Reserved	–	202	CA	–	–	3	E1000328
Reserved	–	203	CB	–	–	3	E100032C
Reserved	–	204	CC	–	–	3	E1000330
Reserved	–	205	CD	–	–	3	E1000334
Reserved	–	206	CE	–	–	3	E1000338
Reserved	–	207	CF	–	–	3	E100033C
VIC Memory Address Register	VMAR	208	D0	RW	NVAX	2-5	–
VIC Tag Register	VTAG	209	D1	RW	NVAX	2-5	–
VIC Data Register	VDATA	210	D2	RW	NVAX	2-5	–
Ibox Control and Status Register	ICSR	211	D3	RW	NVAX	2-5	–

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	(Hex)	Type	Impl	Cat	I/O Address
Ibox Branch Prediction Control Register[5]	BPCR	212	D4	RW	NVAX	2-5	–
Reserved	–	213	D5	–	NVAX	2-6	–
Ibox Backup PC[5]	BPC	214	D6	R	NVAX	2-5	–
Ibox Backup PC with RLOG Unwind[5]	BPCUNW	215	D7	R	NVAX	2-5	–
Reserved	–	216	D8	–	NVAX	2-6	–
Reserved	–	217	D9	–	NVAX	2-6	–
Reserved	–	218	DA	–	NVAX	2-6	–
Reserved	–	219	DB	–	NVAX	2-6	–
Reserved	–	220	DC	–	NVAX	2-6	–
Reserved	–	221	DD	–	NVAX	2-6	–
Reserved	–	222	DE	–	NVAX	2-6	–
Reserved	–	223	DF	–	NVAX	2-6	–

[5]Testability and diagnostic use only; not for software use in normal operation

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	(Hex)	Type	Impl	Cat	I/O Address
Mbox P0 Base Register[5]	MP0BR	224	E0	RW	NVAX	2-5	–
Mbox P0 Length Register[5]	MP0LR	225	E1	RW	NVAX	2-5	–
Mbox P1 Base Register[5]	MP1BR	226	E2	RW	NVAX	2-5	–
Mbox P1 Length Register[5]	MP1LR	227	E3	RW	NVAX	2-5	–
Mbox System Base Register[5]	MSBR	228	E4	RW	NVAX	2-5	–
Mbox System Length Register[5]	MSLR	229	E5	RW	NVAX	2-5	–
Mbox Memory Management Enable[5]	MMAPEN	230	E6	RW	NVAX	2-5	–
Mbox Physical Address Mode	PAMODE	231	E7	RW	NVAX	2-5	–
Mbox MME Address	MMEADR	232	E8	R	NVAX	2-5	–
Mbox MME PTE Address	MMEPTE	233	E9	R	NVAX	2-5	–
Mbox MME Status	MMESTS	234	EA	R	NVAX	2-5	–

[5]Testability and diagnostic use only; not for software use in normal operation

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	(Hex)	Type	Impl	Cat	I/O Address
Reserved	–	235	EB	–	NVAX	2-6	–
Mbox TB Parity Address	TBADR	236	EC	R	NVAX	2-5	–
Mbox TB Parity Status	TBSTS	237	ED	RW	NVAX	2-5	–
Reserved	–	238	EE	–	NVAX	2-6	–
Reserved	–	239	EF	–	NVAX	2-6	–
Reserved	–	240	F0	–	NVAX	2-6	–
Reserved	–	241	F1	–	NVAX	2-6	–
Mbox Pcache Parity Address	PCADR	242	F2	R	NVAX	2-5	–
Reserved	–	243	F3	–	NVAX	2-6	–
Mbox Pcache Status	PCSTS	244	F4	RW	NVAX	2-5	–
Reserved	–	245	F5	–	NVAX	2-6	–

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

(continued on next page)

Table 3–4 (Cont.) Processor Registers

Register Name	Mnemonic	Number (Dec)	Number (Hex)	Type	Impl	Cat	I/O Address
Reserved	–	246	F6	–	NVAX	2-6	–
Reserved	–	247	F7	–	NVAX	2-6	–
Mbox Pcache Control	PCCTL	248	F8	RW	NVAX	2-5	–
Reserved	–	249	F9	–	NVAX	2-6	–
Reserved	–	250	FA	–	NVAX	2-6	–
Reserved	–	251	FB	–	NVAX	2-6	–
Reserved	–	252	FC	–	NVAX	2-6	–
Reserved	–	253	FD	–	NVAX	2-6	–
Reserved	–	254	FE	–	NVAX	2-6	–
Reserved	–	255	FF	–	NVAX	2-6	–
Unimplemented	–	–	100–00FFFFFF	–	–	3	–
See Table 3–3	–	–	01000000–FFFFFFFF	–	–	2	–

Type:

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

Impl(emented):

NVAX = Implemented in the NVAX CPU chip
System = Implemented in the system environment
Vector = Implemented in the optional vector unit or its NDAL interface

Cat(egory), *class-subclass*, where:

class is one of:

1 = Implemented per DEC Standard 032
2 = NVAX-specific implementation that is unique or different from the DEC Standard 032 implementation
3 = Not implemented internally; converted to I/O space read or write and passed to system environment

subclass is one of:

1 = Processed as appropriate by Ebox microcode
2 = Converted to Mbox IPR number and processed via internal IPR command
3 = Processed by internal IPR command, then converted to I/O space read or write and passed to system environment
4 = If virtual machine option is implemented, processed as in 1; otherwise, as in 3
5 = Processed by internal IPR command
6 = May be block decoded; reference causes UNDEFINED behavior
7 = Full interval timer may be implemented in the system environment. Subset ICCS is implemented in NVAX CPU chip
8 = Converted to MFVP MSYNC

3.2 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 KA680 implements these structures as defined in *VAX Architecture Reference Manual*, which should be referenced for a description of the PCB and the PCBB.

3.3 Data Types

The KA680 CPU supports the following subset of the VAX data types: The central processor provides the following subset of the VAX data types:

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

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

3.4 Instruction Set

The KA680 CPU implements the following subset of the VAX instruction set types in microcode.

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

MOVC3
MOVC5
CMPC3 (See Note)
CMPC5 (See Note)
LOCC (See Note)
SCANC (See Note)
SKPC (See Note)
SPANC (See Note)

Note

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

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

- Decimal string
- CRC
- EDITPC

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

- Octaword
- Compatibility mode instructions

Appendix H lists the entire KA680 instruction set, indicating which instructions have microcode assists to speed up macrocode emulation.

3.5 Memory Management

The KA680 implements full VAX Memory Management as defined in the *VAX Architecture Reference Manual*. System space addresses are virtually mapped through single-level page tables, and process space addresses are virtually mapped through two-level page tables. See the *VAX Architecture Reference Manual* for descriptions of the virtual-to-physical address translation process, and the format for VAX page table entries (PTEs).

3.5.1 Translation Buffer

To reduce the overhead associated with translating virtual addresses to physical addresses, the NVAX CPU chip employs a 96-entry, fully associative, translation buffer (TB) 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 (MAPEN)].
- Any page table base or length registers are modified (for example, by writes to IPRs 8 to 13).
- Writing to IPR 57 (TBIA).

In addition, individual TB entries may be flushed by writing to IPR 58 (TBIS).

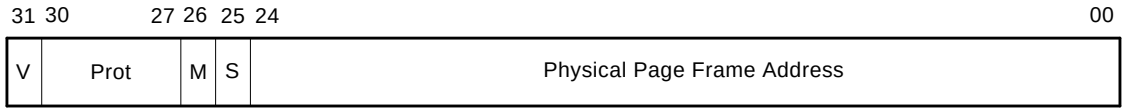
Each entry in the translation buffer is divided into two parts: a 24-bit tag register and a 27-bit PTE register. The tag register is used to store the virtual page number (VPN) of the virtual page that the corresponding PTE register maps, and a valid bit (TB.V) indicating that the tag contains a valid VPN. The PTE register stores the 21-bit PFN field, the PTE.V bit, the PTE.M bit, and the 4-bit PROT field from the corresponding VAX PTE.

When MAPEN bit is set in the MAPEN IPR (IPR 56), memory management is enabled and the CPU will perform VAX memory translation from virtual addresses to physical addresses. The translation buffer is the mechanism by which the NVAX performs quick virtual-to-physical address translations.

Figure 3–4 shows the format of a page table entry and a TB entry.

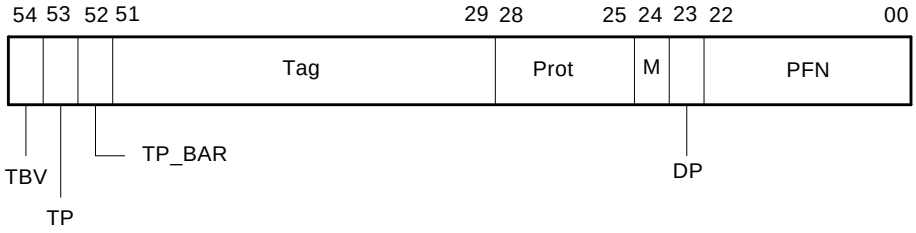
Figure 3–4 PTE and TB Format

Page Table Entry



Where: V = Valid Bit
Prot = Authorized Access Modes
M = Modify Bit
S = Reserved Bit

TB Entry



Where: = TB Entry Valid Bit

TP =	Even Tag Parity Bit
TP_BAR =	Compement of TP
Tag =	Virtual Address<31:9>
Prot =	Authorized Access Modes
M =	Modify Bit
DP =	Even Parity for Validated PTE Field
PFN =	Physical Page Frame Address

LJ-01247-TI0

Note that the TB entry stores all but three bits of the PTE field. The TB entry does not store the S bit because it is not used, and the TB entry does not store the upper two bits of the PTE PFN because these bits correspond to a larger physical address space than NVAX CPU uses. The tag field stores the virtual page frame address. The TBV bit indicates whether the corresponding entry is valid. If TBV is set, then PTE<31> is valid because the TB only caches PTEs whose valid bits are set.

During virtual-to-physical address translation, the contents of the 96 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, and the contents of the corresponding PTE register are used for the translation.

If there is no match, the translation buffer does not contain the necessary VAX PTE information to translate the address of the reference, and the PTE must be fetched from memory. Upon fetching the PTE, the translation buffer is updated so that subsequent references to the same page of memory will hit in the translation buffer.

TB entries are allocated using an NLU (not-last-used) TB allocation pointer. The update is achieved by replacing the entry that is selected by the replacement pointer. The allocation pointer increments in round robin fashion whenever a new buffer entry is loaded, such as after a TB miss. Because the allocation pointer is guaranteed not to point to the last entry referenced, this scheme implements a not-last-used allocation scheme.

The associativity of each TB entry is implemented by the use of comparators on the TBV and tag fields. When a virtual address is to be translated, each TB tag comparator, whose corresponding TBV bit is set, looks for a match between the virtual page frame address and its corresponding tag. If no comparator finds a match, a "TB miss" condition has occurred in that no TB entry contains a translation for the specified address.

If one of the entries detects a match ("TB hit"), the PFN, PROT, and M fields of the corresponding TB entry are read out of the TB and are used to complete the address translation.

The PROT, and M fields, which are accessed along with the PFN, are used by the memory management exception detection logic to determine ACV and M=0 conditions.

TB entries can be invalidated in the following ways:

- An entry can be invalidated by being displaced from the TB by allocation of another PTE to the same TB entry.
- An entry can be invalidated by writing to the TBIS IPR (IPR 58). If the specified TBIS virtual address matches a TB tag, the TBV bit corresponding to the matched tag is cleared. Clearing the TBV bit invalidates the TB entry.
- All entries can be invalidated by writing to the TBIA IPR (IPR 57). This command resets the TBV bit of every TB entry.

3.5.2 30-bit to 32-bit Physical Address Translations

When PAMODE=0, such as is mandatory with the KA680, the NVAX system is configured such that only 30-bit physical addresses are processed at the program level. This is done in two ways.

1. When the address translation logic receives a physical address from one of its reference sources, the mapping is implemented by an address sign extension scheme involving the upper three address bits. In this scheme, address bits <31:30> are forced to the state of address<29>.
2. When the virtual/physical address tranlation logic receives a virtual address, virtual address translation occurs normally without any sign extension of the resulting physical address. This is possible because the corresponding sign extension function is preprocessed on the upper three bits of page frame address, which is written into the TB during the TB tag fill operation.

The following two figures show the register format of the TBTAG and TBDATA IPRs. These two IPRs allow software to directly access the contents of the translation buffer for diagnostic purposes.

Figure 3–5 Translation Buffer Tag (TBTAG) - IPR 47

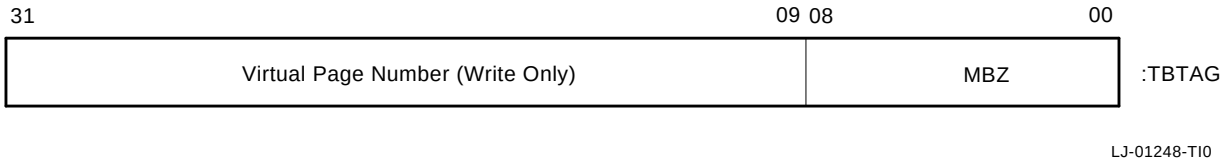
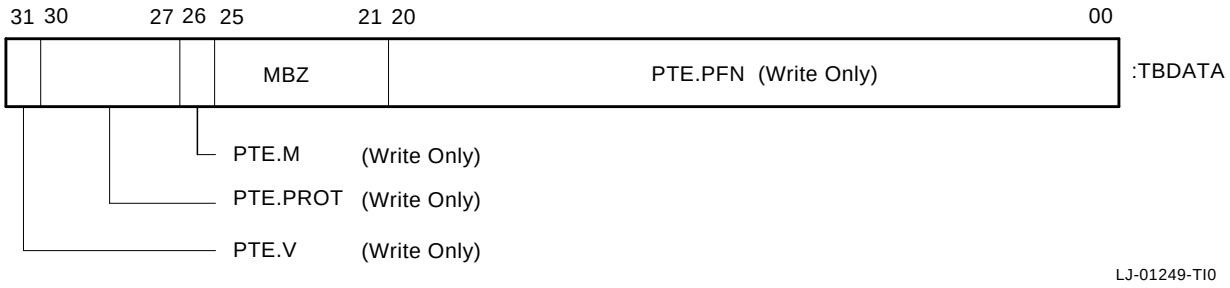


Figure 3–6 Translation Buffer Data (TBDATA) - IPR 59



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 and 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-stream 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 NVAX CPU to initiate a reserved operand fault.

3.5.3 Memory Management Control Registers

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

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

Memory management can be enabled/disabled via IPR 56 (MAPEN). Writing 0 to this register with an MTPR instruction disables memory management and writing a 1 to this register with an MTPR instruction enables memory management. Writes to this register flush the translation buffer. To determine whether or not memory management is enabled, IPR 56 is read using the MFPR instruction.

Translation buffer entries that map a particular virtual address can be invalidated by writing the virtual address to IPR 58 (TBIS) using the MTPR instruction.

Note

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

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

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

Note

The TBIS, TBIA, and TBCHK IPRs are write-only. The operation of an MFPR instruction from any of these registers is UNDEFINED.

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

- 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 NVAX CPU accomplishes this by flushing the TB on any change to SBR, SLR, or 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.

There are two IPRs that are used by diagnostic software to test the translation buffer:

IPR 47 (TBTAG) Format shown in Figure 3–5.
IPR 59 (TBDATA) Format shown in Figure 3–6.

For information regarding the use of the V, PROT, and M bit fields, consult the *VAX Architecture Reference Manual*.

3.6 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).

3.6.1 Interrupts

Interrupts can be divided into two classes: nonmaskable and maskable.

Nonmaskable interrupts cause a halt via 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 (except those 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 are saved.
- The processor IPL is raised to the priority level of the interrupt (except for Q22–bus, mass storage, and network interface interrupts in which 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).

The various interrupt conditions for the KA680 are listed in Table 3–5 along with their associated priority levels and SCB offsets. The reader should note that this table is intended as a quick reference, and may not include all possible causes of the various interrupts, specifically with regard to error conditions.

Table 3–5 Interrupt Priority Levels

Priority Level	Interrupt Condition	SCB Offset
Nonmaskable	BDCOK and BPOK negated then asserted on Q22–bus (Powerup)	*
–	BDCOK negated then asserted while BPOK asserted on Q22–bus (Powerup)	**
–	BHALT asserted on Q22–bus	**
–	BREAK generated by the console device	**
1F	Unused	–
1E	BPOK negated on Q22–bus	0C
1D	Backup cache addressing errors	60
–	Backup cache uncorrectable data ECC errors on Bcache read for a write that hits valid/owned	60
–	NVAX read timeout or Read Data Error on Oread for a write after the requested quadword has arrived	60
–	Illegal length write transaction to memory or I/O	60
–	Reserved command detected by memory or I/O during write transaction	60
–	Pending write times out waiting for disown write	60
–	Disown write to unowned memory location	60
–	Main memory NXM errors on writes	60
–	NDAL parity errors on writes	60
–	CP-bus NXM/TIMEOUT on a write	60
–	Q22–bus NXM/NOSACK on a write	60
–	Q22–bus NOGRANT on a write	60
–	Uncorrectable memory errors during map read for Q22–bus address translation	60
1C:1B	Unused	–
1A	Correctable main memory errors	54
–	Uncorrectable main memory errors	54

* 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).

(continued on next page)

Table 3–5 (Cont.) Interrupt Priority Levels

Priority Level	Interrupt Condition	SCB Offset
–	Correctable O-bit memory errors	54
–	Pending read times out waiting for disown write	54
–	No acknowledgement on returned read data from NMC	54
–	NDAL data parity errors	54
–	Primary cache tag or data parity errors	54
–	Virtual instruction cache tag or data parity errors	54
–	Backup cache addressing errors	54
–	Backup cache correctable data ECC errors	54
–	Backup cache uncorrectable data ECC errors	54
–	Backup cache correctable tag ECC errors	54
–	Backup cache uncorrectable data ECC errors	54
–	Illegal length transaction to memory or I/O space	54
–	Reserved command to memory or I/O space	54
–	CP-bus parity errors on I/O read transactions	54
–	CP-bus ERR_L signal asserted by I/O device during I/O read transaction	54
–	CP Bus NXM/TIMEOUTS errors on I/O reads	54
19:18	Unused	–
17	BR7 L asserted	Q22–bus vector plus 200_{16}
16	Interval timer interrupt	C0
–	BR6 L asserted	Q22–bus vector plus 200_{16}
15	BR5 L asserted	Q22–bus vector plus 200_{16}

* 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).

(continued on next page)

Table 3–5 (Cont.) Interrupt Priority Levels

Priority Level	Interrupt Condition	SCB Offset
14	Console Terminal	F8,FC
–	Programmable Timers	78,7C
–	Mass storage interface one (DSSI port 1)(External)	108
–	Mass storage interface two (DSSI port 2)(Internal)	104
–	Network interface	10C
–	Interprocessor doorbell	204
–	BR4 L asserted	Q22–bus vector plus 200_{16}
13:10	Unused	–
0F:01	Software interrupt requests	84-BC

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

Note

Because the Q22–bus does not allow differentiation between the four bus grant levels (for example, a level 7 device could respond to a level 4 bus grant), the KA680 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 KA680 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 3–7), is used for loading the processor priority field in the PSL (bits<20:16>).
- IPR 20, the software interrupt request register (SIRR) (Figure 3–8), is used for creating software interrupt requests.
- IPR 21, the software interrupt summary register (SISR) (Figure 3–9), records pending software interrupt requests at levels 1 through 15.

The format of these registers is presented in Figure 3–7, Figure 3–8, and Figure 3–9. Refer to the *VAX Architecture Reference Manual* for more information on these registers.

Figure 3–7 Interrupt Priority Level Register (IPLR) - IPR 18

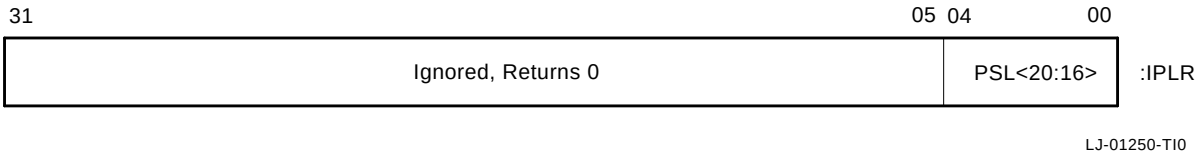


Figure 3–8 Software Interrupt Request Register (SIRR) - IPL 20

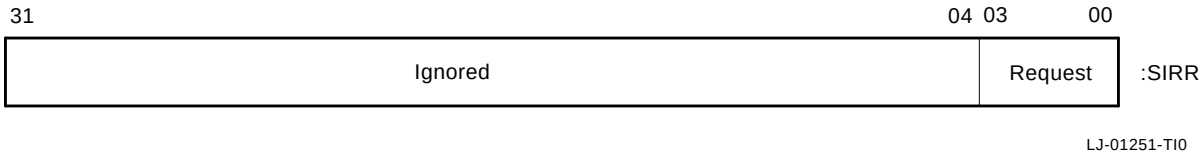
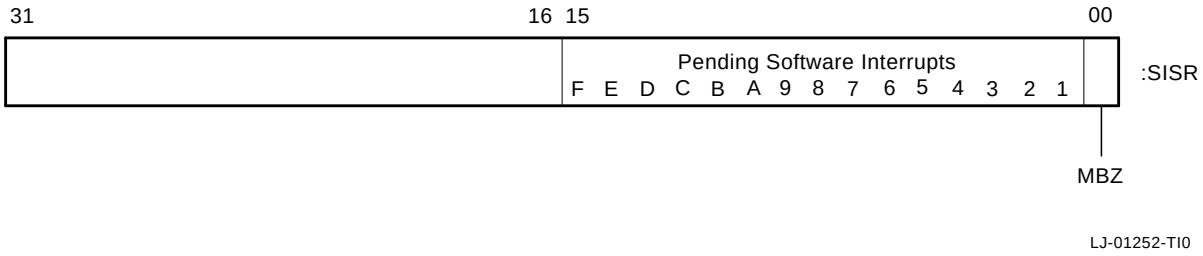


Figure 3–9 Software Interrupt Summary Register (SISR) - IPL 21

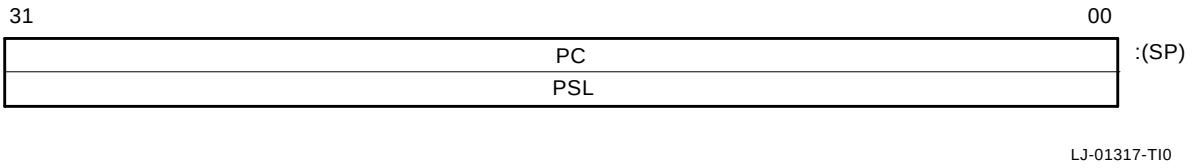


3.6.1.1 Power Fail Interrupt

Power fail interrupts are requested to report imminent loss of power to the CPU. Power fail interrupts are requested via the PWRFL_L pin at IPL 1E (hex) and are dispatched to the operating system through SCB vector 0C (hex).

The stack frame for a power fail interrupt is shown in Figure 3–10.

Figure 3–10 Power Fail Interrupt Stack Frame

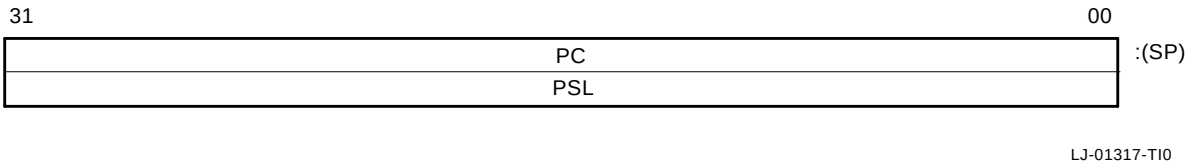


3.6.1.2 Hard Error Interrupts

Hard error interrupts are requested to report an error that was detected asynchronously with respect to instruction execution. This results in an interrupt at IPL 1D (hex) to be dispatched through SCB vector 60 (hex). Typically, these errors indicate that machine state has been corrupted and that retry is not possible.

The stack frame for a hard error interrupt is shown in Figure 3–11.

Figure 3–11 Hard Error Interrupt Stack Frame

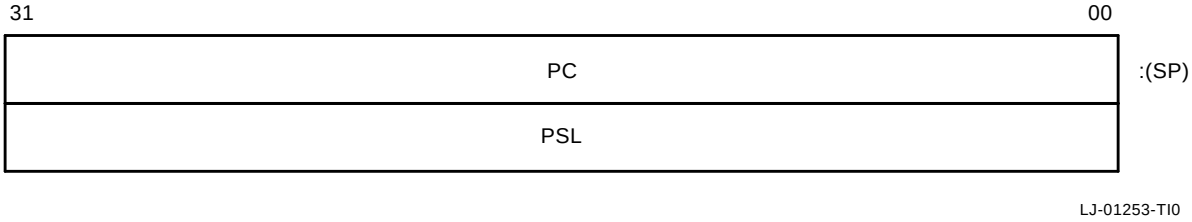


3.6.1.3 Soft Error Interrupts

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 (hex) to be dispatched through SCB vector 54 (hex).

The stack frame for a soft error interrupt is shown in Figure 3–12.

Figure 3–12 Soft Error Interrupt Stack Frame



3.7 Exceptions

The VAX architecture recognizes six classes of exceptions. Table 3–6 lists instances of exceptions in each class.

Table 3–6 Exception Classes

Exception Class	Instances
Arithmetic traps/faults	Integer overflow trap Integer divide-by-zero trap Subscript range trap Floating overflow fault Floating divide-by-zero fault Floating underflow fault

(continued on next page)

Table 3–6 (Cont.) Exception Classes

Exception Class	Instances
Memory management exceptions	Access control violation fault Translation not valid fault M=0 fault
Operand reference exceptions	Reserved addressing mode fault Reserved operand fault or abort
Instruction execution exceptions	Reserved/privileged instruction fault Emulated instruction faults XFC fault Change-mode trap Breakpoint fault Vector disabled fault
Tracing exceptions	Trace fault
System failure exceptions	Kernel-stack-not-valid abort Interrupt-stack-not-valid halt Console error halt Machine check abort

A trap is an exception occuring at the end of the instruction that caused the exception. Therefore, the PC saved on the stack is the address of the next instruction that would normally have been executed.

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

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

The following sections describe only those exceptions that are unique to the NVAX CPU, or where the *VAX Architecture Reference Manual* is not clear about the implementation.

3.7.1 Arithmetic Exceptions

Arithmetic exceptions are detected during the execution of instructions that perform integer or floating-point arithmetic manipulations. Whether the exception is reported as a trap or a fault is a function of the specific event. In any case, the exception is reported through SCB vector 34 (hex) with the stack frame shown in Figure 3–13. Table 3–7 lists the exceptions reported by this mechanism.

Figure 3–13 Arithmetic Exception Stack Frame

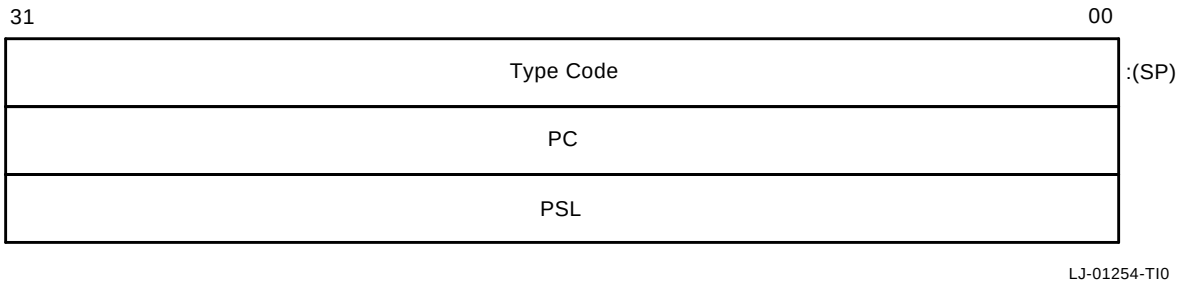


Table 3–7 Arithmetic Exceptions

Type Code			
Decimal	**Hex**	**Type**	**Exception**
1	1	Trap	Integer overflow
2	2	Trap	Integer divide-by-zero
7	7	Trap	Subscript range
8	8	Fault	Floating overflow
9	9	Fault	Floating divide-by-zero
10	A	Fault	Floating underflow

3.7.2 Memory Management Exceptions

Memory management exceptions are detected during a memory reference and are always reported as faults. The three memory management exceptions are listed in Table 3–8. All three exceptions push the same frame on the stack, as shown in Figure 3–14. The top longword of the stack frame contains a fault parameter whose bits are described in Table 3–9.

Table 3–8 Memory Management Exceptions

SCB Vector	Exception
20 (hex)	Access control violation
24 (hex)	Translation not valid
3C (hex)	Modify fault

Figure 3–14 Memory Management Exception Stack Frame

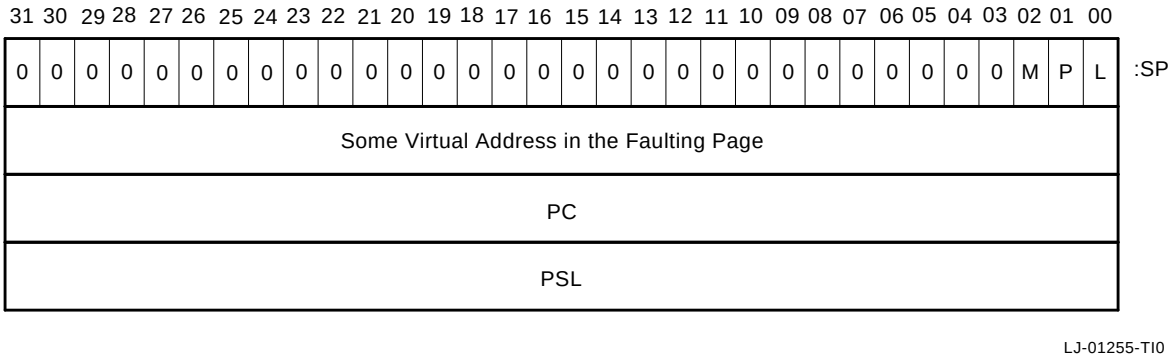


Table 3–9 Memory Management Exception Fault Parameter

Bit	Mnemonic	Meaning
0	L	Length violation
1	P	PTE reference
2	M	Modify or write intent
3	0	Always 0
4	0	Always 0

3.7.3 Emulated Instruction Exceptions

The NVAX CPU implements the VAX base instruction group. For certain instructions outside that group, the NVAX microcode provides support for the macrocode emulation of instructions. There are two types of emulation exceptions, depending on whether PSL<FPD> is set at the beginning of the instruction.

If PSL<FPD>=0 at the beginning of the instruction, the exception is reported through SCB vector C8 (hex) as a trap with the stack frame shown in Figure 3–15. The longwords in the stack frame are described in Table 3–10.

Figure 3–15 Instruction Emulation Trap Stack Frame

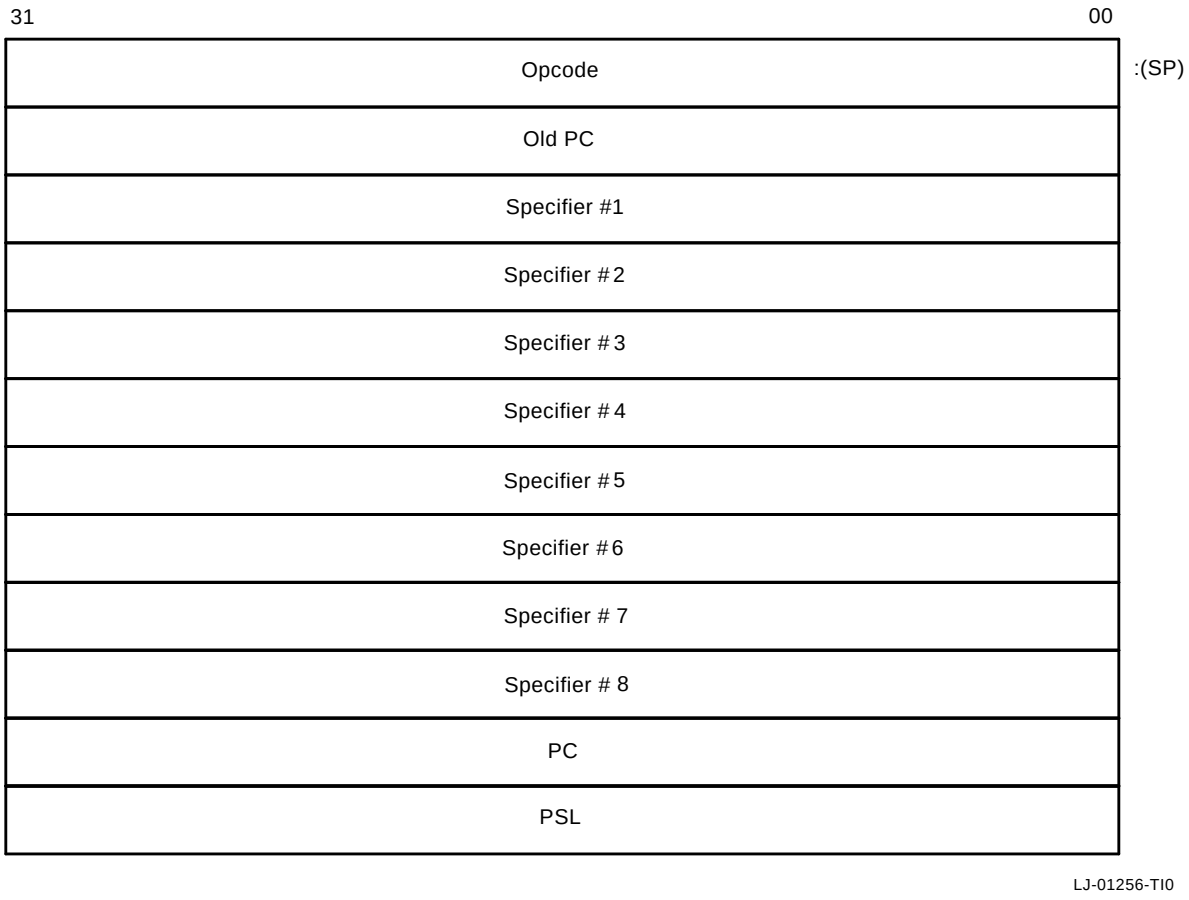
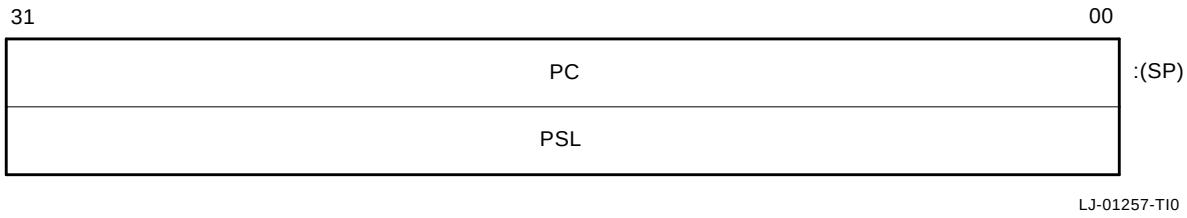


Table 3–10 Instruction Emulation Trap Stack Frame

Location	Use
Opcode	Zero-extended opcode of the emulated instruction.
Old PC	PC of the opcode of the emulated instruction.
Specifiers	Address of the specified operand for specifiers of access type write (.wx) or address (.ax). Operand value for specifiers of access type read (.rx). For read-type operands whose size is smaller than a longword, the remaining bits are UNPREDICTABLE. For those instructions that do not have 8 specifiers, the remaining specifier longwords contain UNPREDICTABLE values.
PC	PC of the instruction following the emulated instruction.
PSL	PSL saved at the time of the trap.

If PSL<FPD>=1 at the beginning of the instruction, the exception is reported through SCB vector CC (hex) as a fault with the stack frame shown in Figure 3–16. In this case, PC is that of the opcode of the emulated instruction.

Figure 3–16 Suspended Emulation Fault Stack Frame



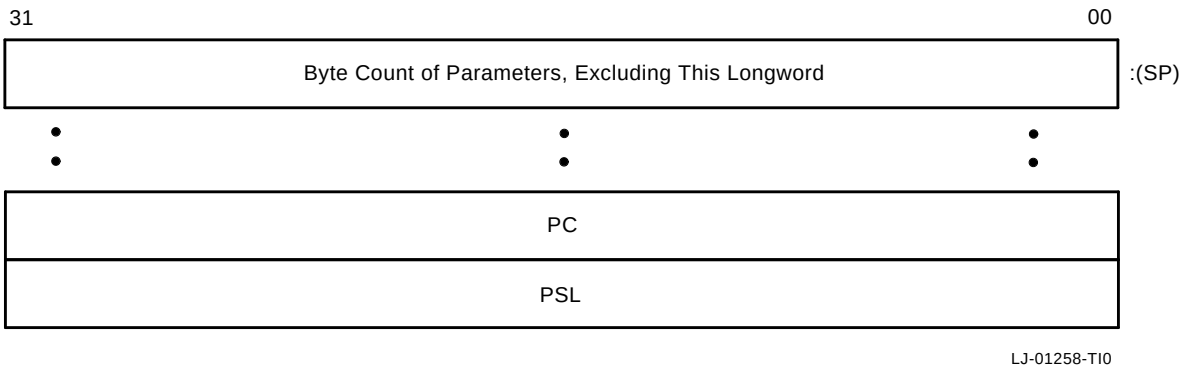
3.7.4 Vector Unit Disabled Fault

When the NVAX CPU attempts to issue a vector instruction to the optional vector processor, it will result in this fault because the KA680 does not contain a vector unit. There are no parameters for this exception (beside the usual PC/PSL pair).

3.7.5 Machine Check Exceptions

A machine check exception is reported through SCB vector 04 (hex) when the NVAX CPU detects an error condition. The frame pushed on the stack for a machine check indicates the type of error and provides internal state information that may help identify the cause of the error. The generic machine check stack frame is shown in Figure 3–17.

Figure 3–17 Generic Machine Check Stack Frame



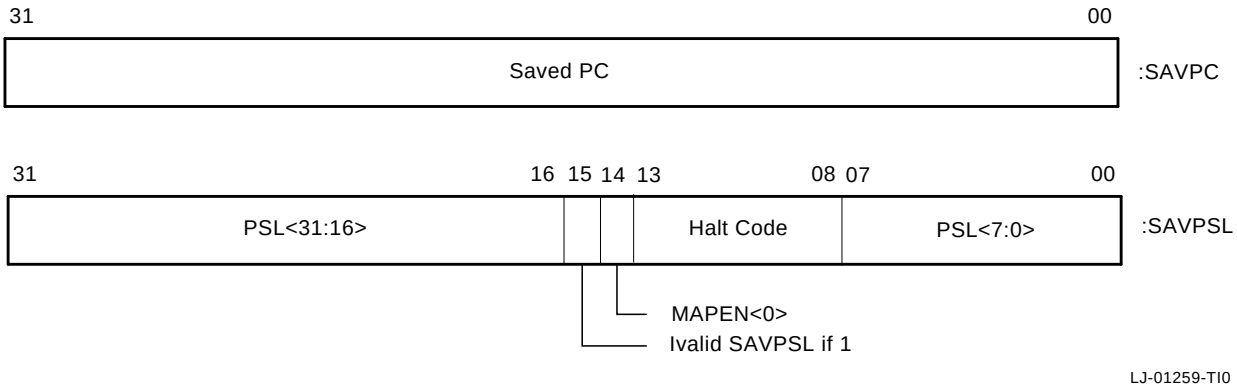
3.7.6 Console Halts

In certain microcode flows, the NVAX microcode may detect an inconsistency in internal state, a kernel-mode HALT, or a system reset. In these instances, the microcode initiates a hardware restart sequence, which passes control to the console program.

When a hardware restart sequence is initiated, the NVAX microcode saves the current CPU state, partially initializes the CPU, and passes control to the console program at physical address E0040000 (hex).

During a hardware restart sequence, the stack pointer is saved in the appropriate stack pointer IPR (0 through 4), the current PC is saved in IPR 42 (SAVPC), and the current PSL, halt code, and validity flag are saved in IPR 43 (SAVPSL). The format of SAVPC and SAVPSL are shown in Figure 3–18.

Figure 3–18 Console Saved PC and Saved PSL



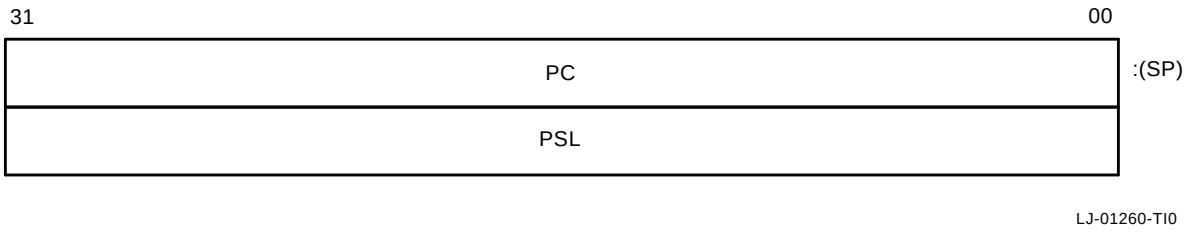
Console halts are discusssed in detail in Appendix D.

3.7.7 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. Note that a console halt with an error code of ERR_INTSTK is taken if a memory management exception is encountered while attempting to push information on the interrupt stack.

The kernel stack not valid exception is dispatched through SCB vector 08 (hex) with the stack frame shown in Figure 3–19.

Figure 3–19 Kernel Stack Not Valid Stack Frame



3.8 System Control Block (SCB)

The system control block (SCB) consists of two pages in main memory that contain the vectors by which interrupts and exceptions are dispatched to the appropriate service routines. The SCB is pointed to by IPR 17, the system control block base register (SCBB). The system control block base format is shown in Figure 3–20. The description of the format is in Table 3–11.

Figure 3–20 System Control Block Base Register (SCBB) - IPR 17

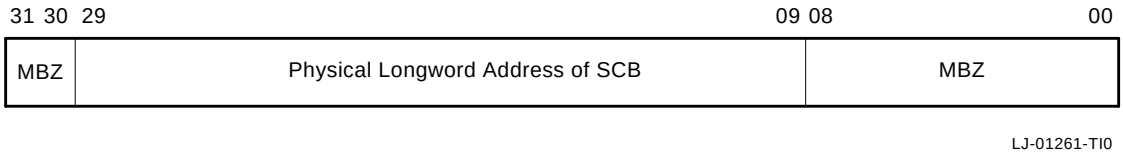


Table 3–11 The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	# Params	Notes
00	Passive Release	Interrupt	0	IPL is raised to request IPL
04	Machine Check	Abort	6	Parameters reflect machine state
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 systems
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	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

(continued on next page)

Table 3–11 (Cont.) The System Control Block Format

SCB Offset	Interrupt/Exception Name	Type	# Params	Notes
6C-74	Unused	–	–	
78	Programmable Timer 0	Interrupt	0	IPL is 14
7C	Programmable Timer 1	Interrupt	0	IPL is 14
80	Unused	–	–	
84	Software Level 1	Interrupt	0	
88	Software Level 2	Interrupt	0	Ordinarily used for AST delivery
8C	Software Level 3	Interrupt	0	Ordinarily used for process scheduling
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 One (DSSI PORT 1)	Interrupt	0	IPL is 14
104	Mass Storage Interface Two (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-FFFC are used to directly vector interrupts from the external bus. The SCB 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, which cannot guarantee that the device generating the SCB 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.

3.9 System Identification

The KA680 firmware and operating system software references two registers to determine the processor on which they are running. The first, the system identification register (SID), is an internal processor register. The second, the system identification extension register (SIE), is a firmware register located in the KA680 EPROM.

3.9.1 System Identification Register

The system identification register (SID), IPR 62, is a read-only register implemented in the NVAX CPU. This 32-bit, read-only register is used to identify the processor type and its microcode revision level. The SID longword is read from IPR 62 using the MFPR instruction. This longword value is processor-specific. The format is shown in Figure 3–21. Bit definitions are listed in Table 3–12.

Figure 3–21 System Identification Register (SID) - IPR 62

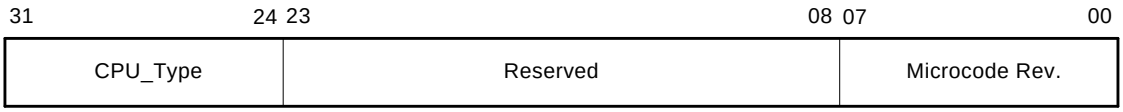


Table 3–12 System Identification Register

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

3.9.2 System Identification Extension (SIE) Register (20040004)

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

By convention, all MicroVAX systems implement a longword at physical location 20040004 in the firmware EPROM for the SIE register. This 32-bit read-only register is implemented in the KA680 ROM. Figure 3–22 shows the format of this register. Table 3–13 lists the definitions of the register bits.

Figure 3–22 System Identification Extension Register (SIE)

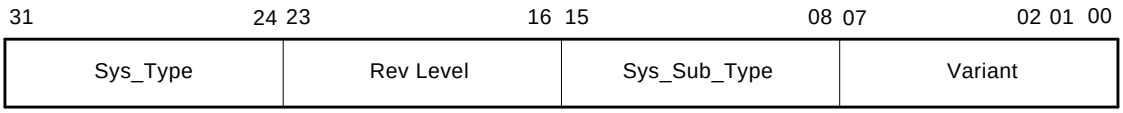


Table 3–13 System Identification Extension Register Bits

Field	Name	RW	Description
31:24	SYS_TYPE	ro	This field identifies the type of system for a specific processor. *01 : Q22–bus single processor system.*
23:16	VERSION	ro	This field identifies 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 (hex).
15:8	SYS_SUB_ TYPE	ro	This field identifies the particular system subtype. *01 : KA650* *02 : KA640* *03 : KA655* *04 : KA670* *05 : KA660* *06 : KA680*
7:0	RESERVED		This field is reserved

3.10 CPU References

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

- Instruction-stream (I-stream) read references.
- Data-stream (D-stream) read references.
- Ownership read (OREAD) references.
- Disown write (WDISOWN) references.
- Write references.
- Bad data write cycles (BADWDATA). See Section 4.4 for more information regarding the use of BADWDATA.

3.10.1 Instruction-Stream Read References

An instruction stream (I-stream) reference is defined as a read reference to acquire a VAX instruction. Furthermore, a VAX instruction consists of its opcode, all operand specifiers, and any operands that are necessarily contiguous with the rest of the instruction stream. Thus, the instruction stream includes short literals, immediate-mode operands, absolute addresses in absolute mode addressing, branch displacements, CALLx entry masks, and CASEx tables. Except for immediate-mode operands and branch displacements, the instruction stream does not include operands, even ones addressed relative to the PC. Except for absolute addresses, the instruction stream does not include indirect addresses. It also does not include EDITPC patterns. All instruction operands and indirect addresses not considered to be within the instruction stream are considered data, and are therefore accessed by the NVAX using D-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 will generate an aligned quadword I-stream read reference.

3.10.2 Ownership Read References

The NVAX uses ownership read (OREAD) references to gain ownership of a hexaword (32 bytes) block of memory. OREADs are defined only from memory space; they are not used in I/O space.

The primary purpose for ownership reads on the KA680 CPU module is to facilitate the use of a write-back caching scheme. The backup cache on the KA680 CPU module uses a "write-back" scheme, whereby write transactions to cached memory locations result in the modification of the cached copy only (that is, the write transaction does not modify the actual memory location). This substantially reduces the time required for the write transaction, resulting in a corresponding improvement in system performance.

This presents a potential problem for memory locations, which may be shared between the NVAX CPU and the I/O DMA devices, since the Bcache may contain the only up-to-date copy of a location referenced by a DMA device. Because of this, the KA680 memory subsystem utilizes the concept of memory ownership. In this scheme, each hexaword (32 bytes) of main memory has associated with it an ownership bit. These ownership bits reside on the KA680 CPU module and are controlled by the NVAX memory controller chip (NMC). Whenever a device on the module requests a transaction to main memory, the memory controller checks to see if the hexaword containing the referenced location is owned by another device (for example, the NVAX CPU or an I/O device). If it is owned, then the requested transaction is pended until the owner of the hexaword relinquishes ownership of that hexaword. In this way, the system can maintain memory consistency in the presence of the NVAX CPU write-back cache.

The NVAX CPU will request ownership of memory locations in two general cases. The first is when software attempts to write to a location that is not currently cached in the Bcache AND owned by the CPU. In this case, the NVAX CPU will issue an OREAD to the memory subsystem to acquire ownership of the referenced hexaword. In this way, subsequent writes to the owned (and cached) location will occur only in the cache, and will not update the contents of the actual memory location. If another device wishes to access the owned hexaword, or if the CPU's Bcache determines that it must deallocate the cache block containing the owned hexaword, then the NVAX CPU will perform a disown write (described below) of the owned hexaword to memory. The disown write updates memory with the potentially modified data, and signals the memory subsystem that the NVAX CPU is relinquishing ownership of that hexaword.

The other case when the NVAX CPU will perform an OREAD to request ownership of a hexaword is for VAX instructions that perform interlocked read/modify/write operations on memory. The ownership mechanism is used in this case to prevent I/O or Q22–bus devices from accessing the relevant memory location in the middle of the read/modify/write operation.

When memory receives an ownership read, an "owned" bit corresponding to the hexaword containing the referenced location is set in memory and the read data is returned. Each hexaword in memory has an ownership bit. The NVAX backup cache is organized by hexawords also, with an owned bit for each hexaword. Memory clears the owned bit when a disown write of any length is received for an owned block.

If the ownership bit is already set in memory when the OREAD arrives, data is not returned immediately to the commander. Once the node that owns the data performs a disown write to the owned block, the ownership bit is set in memory and the data is returned to the commander.

For more information regarding the use of ownership bits in the KA680 system, refer to Chapter 5 and Section 4.4.2.

3.10.3 Disown Write References

The disown write (WDISOWN) transaction is the complement to the ownership read. After the NVAX CPU successfully gains ownership of a block in memory, it must relinquish ownership when another device (for example, a Q22–bus DMA device) wants to access the block or when the Bcache needs to do a deallocate. The NVAX CPU accomplishes this in a way transparent to software by deallocating the owned hexaword from the Bcache and performing a disown write to the memory with the latest copy of the data. The memory, which has been monitoring the bus traffic, notices the disown write from the NVAX CPU and clears the ownership bit in memory and writes the new data.

The NVAX CPU uses the disown write transaction of hexaword (32 bytes) length to perform writebacks from the backup cache. This is transparent to software except in the case of errors, when the logged information could indicate that an error occured during a disown write transaction.

3.10.4 Data-Stream Read References

Data-stream (D-stream) references are defined as all read references that do not fall under either OREAD or I-stream categories. Generally, D-stream references are used by the NVAX CPU to read instruction operands and data from memory. A more complete list of read references that qualify as D-stream is given below.

- 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, an OREAD/WDISOWN transaction pair is used to prevent I/O devices from accessing the referenced location in the middle of the instruction.

3.10.5 Write References

Whenever data is stored or moved, and a WDISOWN transaction is not in order, a normal WRITE reference is generated.

3.11 NVAX Data/Address Lines (NDAL)

3.11.1 NDAL Transactions

The following sections describe the set of NDAL transactions.

In order to maximize the bandwidth of the bus connecting the CPU to the memory and I/O controllers, the NVAX chip set (NVAX, NMC, NCA) communicate over a "pended" bus, the NDAL. The main feature of this bus is that devices requesting read data do not tie up the bus while waiting for the return data. Rather, a device will issue one of the "read" commands on the NDAL and then relinquish control of the bus to other devices so that other transactions may be performed. At the same time, the responder to the first device prepares to send back the data associated with the read request. Because of the pended nature of the bus, the

NDAL bus command set includes separate transactions for returning data from an earlier read cycle. Table 3–14 shows the entire set of NDAL commands and how they are used by NVAX.

In memory space, NVAX issues all reads with hexaword length. Normal writes to memory space are always quadword length, and disown writes are quadword or hexaword. When the cache is operating normally, disown writes are only issued in hexaword length. When the cache is in error transition mode, NVAX issues disown writes of both hexaword and quadword length. For a discussion of error transition mode, refer to the section on the backup cache.

When the cache is off, NVAX issues only quadword disown writes. NVAX issues quadword disown writes only as the result of a VAX interlocked instruction.

In I/O space, the ownership commands (OREAD and Disown Write) are not defined at all. NVAX issues only quadword operations in I/O space. NVAX never uses the BADWDATA command in I/O space.

Table 3–14 NDAL Command Usage by NVAX

Address Space	Command	Used by NVAX	Length QW	Length OW	Length HW
N/A	Nop	Yes	–	–	–
N/A	Reserved	No	–	–	–
Memory	WRITE	Yes	Yes	No	No
Memory	WDISOWN	Yes	Yes	No	Yes
Memory	IREAD	Yes	No	No	Yes
Memory	DREAD	Yes	No	No	Yes
Memory	OREAD	Yes	No	No	Yes
Memory	RDE	No	–	–	–
Memory	WDATA	Yes	–	–	–
Memory	BADWDATA	Yes	–	–	–
Memory	RDR0	No	–	–	–
Memory	RDR1	No	–	–	–
Memory	RDR2	No	–	–	–
Memory	RDR3	No	–	–	–
I/O	WRITE	Yes	Yes	No	No
I/O	WDISOWN	No	No	No	No
I/O	IREAD	Yes	Yes	No	No
I/O	DREAD	Yes	Yes	No	No
I/O	OREAD	No	No	No	No
I/O	RDE	No	–	–	–
I/O	WDATA	Yes	–	–	–
I/O	BADWDATA	No	–	–	–
I/O	RDR0	No	–	–	–
I/O	RDR1	No	–	–	–
I/O	RDR2	No	–	–	–
I/O	RDR3	No	–	–	–

3.11.1.1 Reads and Fills

The read address cycle, which is recognized by one of the three read commands (DREAD, IREAD, or OREAD), is decoded by the NDAL devices that are receiving NDAL commands (only one of the NDAL chips may drive the bus at a time). The one recognizing the address latches that address and command. This device is the responder. The responder uses read data return (RDR) or read data error (RDE) cycles to return the data. Reads and fills are described in the following sections. The NVAX CPU never issues the RDR or RDE cycles, since it is never a direct responder to a read cycle. For examples of when the NVAX CPU may indirectly respond to a read transaction from the NCA, see the section on disown writes.

3.11.1.1.1 D-stream Read Requests (DREAD) The NVAX CPU and the NCA use the DREAD command to request data-stream data from a responder, either memory or an I/O device. The NVAX CPU may issue DREAD cycles to the NMC or NCA. The NCA may issue DREAD cycles only to the NMC.

3.11.1.1.2 I-stream Read Requests (IREAD) The IREAD command is used by the NVAX CPU and the NCA to request instruction-stream data. The NVAX CPU can request I-stream data from the NMC or the NCA. The NCA can request I-stream data only from the NMC. For more information regarding the difference between I-stream and D-stream references, see Section 3.10.

3.11.1.1.3 Ownership Read Requests (OREAD) The NVAX CPU and the NCA use the OREAD command to gain ownership of a hexaword block of memory. In addition to facilitating the write-back cache of the NVAX CPU, the memory ownership concept is used to implement memory interlocks for VAX interlocked instructions.

OREADs are only defined for memory space; they are not used in I/O space.

When memory receives an ownership read, an "owned" bit is set in memory and the read data is returned. Each hexaword in memory has an owned bit. The NVAX backup cache is organized by hexawords also, with an owned bit for each hexaword. The memory controller clears the owned bit when a disown write of any length is received at the same block.

If the ownership bit is already set in memory when the OREAD arrives, data is not returned immediately to the NDAL commander. An example of this is if an I/O device on one of the CP-buses attempts to access a buffer pointer in main memory and the referenced location is owned by the NVAX CPU. In this case, the NMC will notice that the ownership bit for the referenced hexaword is set and will pend the transaction. Simultaneously, the NVAX CPU, which constantly monitors the NDAL, has seen the reference to a location it owns, and will therefore write back the owned hexaword with a WDISOWN cycle. The NMC will then allow the original transaction from the NCA to complete. The opposite situation is also possible where the NCA may own a location that the NVAX CPU needs to access. The NCA does not have a write-back cache like the NVAX CPU, but the NCA will use the OREAD/WDISOWN transaction pair when performing interlocked references from I/O devices.

Through this ownership mechanism, the OREAD/WDISOWN transactions prevent access by the NCA (on behalf of I/O devices) to memory locations that may have become stale. This occurs because the NVAX CPU has modified the locations contents in the backup cache, but has not yet updated the actual memory location.

During normal operation, the NVAX will aquire ownership, via an OREAD, of any memory space location that it needs to modify. Since this process also allocates a block in the backup cache for the referenced location, the NVAX CPU will virtually never need to perform writes directly to memory. Rather, memory will be modified with the updated values only when the associated Bcache block must be written back to memory to make room in the Bcache, or because of an I/O reference to an owned location. Note that I/O space references are never cached, so writes to I/O space always appear on the NDAL.

3.11.1.1.4 Read Data Return Cycles (RDR0, RDR1, RDR2, RDR3) The Read Data Return command is used in response to any read request, whether IREAD, DREAD, or OREAD. Multiple cycles are necessary to transfer all the quadwords in a given hexaword transaction, and the cycles are not required to be consecutive. The commander, which has been monitoring the bus traffic waiting for its return data, latches the information. The responder returns the commander ID with the returned read data so the commander can recognize the returned read data it requested.

Because the NDAL is a pended bus, multiple reads may be outstanding at a time. Because read data return cycles do not have to occur contiguously, it is possible for read data return cycles resulting from different read requests to take place in an interleaved fashion.

3.11.1.1.5 Read Data Error Cycles (RDE) RDE is used to notify a commander of a problem with read data that is being returned. For example, the NMC uses this command when it encounters an uncorrectable read error while processing a read request from the NVAX CPU or the NCA.

3.11.1.2 Writes

3.11.1.2.1 Normal Write Transactions (WRITE) These transactions are used to move a pattern of bytes from an NDAL commander to one of the responders.

Parity must be correct for all bytes sent from any node because all three NVAX chips check parity across the entire NDAL during every cycle.

If NVAX sees a write on the NDAL, it treats it as an invalidate request. In this case, the NVAX will perform a lookup of the backup cache tag store to see if the referenced location is contained in the backup cache. If it is, then the referenced cache block is invalidated. If the cache block is marked as owned, then the NVAX CPU also performs a write-back of the block to memory, since the data may be the only copy of valid data for that memory location.

3.11.1.2.2 Disown Write Transactions (WDISOWN) The disown write transaction is the complement to the ownership read. After NVAX successfully gains ownership of a hexaword block in memory, it must relinquish ownership if an I/O device wants to access the block (through the NCA) or when the Bcache needs to do a deallocate. The NVAX CPU accomplishes this by performing a disown write to the memory with the latest copy of the backup cache data. The memory, which has already pended the I/O device's transaction, notices that the CPU's transaction is a disown write. This condition allows it to clear the ownership bit in memory and to write the data as requested. Immediately following this, the memory controller allows the pended I/O device's transaction to complete, since the referenced hexaword is no longer owned by the NVAX CPU.

NVAX uses the Disown Write command of hexaword length to perform write-backs from the backup cache. When the cache is off, it uses quadword disown writes to achieve the effect of a write unlock for VAX interlocked instructions. As mentioned previously, the NCA also uses the OREAD/WDISOWN transaction pair when accessing main memory on behalf of I/O devices that use interlocked bus cycles.

3.11.1.2.3 Write Data and Bad Write Data (WDATA,BADWDATA) The Write Data command is used during the data cycles of a write if the data is good. If the data has been corrupted in some way, the command used is Bad Write Data. An example of the use of BADWDATA is when the backup cache must deallocate an owned block to make room for new data or because of an I/O reference to that hexaword, and in the process of performing write-back, the NVAX CPU encounters uncorrectable errors in the cached data. In this case, the CPU will use the WDISOWN command, specifying the hexaword being disowned, followed by four data cycles to transfer each of the four quadwords of data. The bad quadword(s) will be marked through the use of the BADWDATA command instead of the WDATA command.

When one quadword of a hexaword write disown is bad, the Bad Write Data command is only used for that quadword. The Write Data command is used for the good quadwords. The memory can use this information to distinguish which quadword of a hexaword block is bad. In addition, a soft error interrupt may be generated when a BADWDATA cycle is driven on the NDAL by the NCA.

3.11.2 Cache Coherency

Ownership reads and disown writes on the NDAL are intended to support the NVAX CPU's writeback cache by attaching an owner status to each block in physical memory. A block in memory is defined as a hexaword, or 32 bytes. Once the NVAX CPU owns a block, it may write it repeatedly without accessing memory. A memory block may be owned either by the memory subsystem or by the NVAX CPU, but not both at the same time. Ownership is passed from memory to the NVAX CPU through an Ownership Read command. Ownership is passed back to the memory through a Disown Write command from the CPU.

The ownership bits in the Bcache and in memory indicate which device owns the block: the Bcache or the NMC. The ownership bit in the Bcache is set when it owns the block and is clear when memory (NMC) owns the block. The ownership bit in memory is set when the Bcache owns the block and clear when memory owns the block. The ownership bit corresponding to a hexaword of memory may also be set to indicate that the NCA owns the block. This is possible when the NCA is accessing memory on behalf of an I/O device that is using interlocked bus cycles on the CP-bus. In this case, the NCA will always follow the OREAD transaction immediately with the WDISOWN transaction.

Shared read-only access to a block is permitted only when memory owns it. Otherwise, the block can only be read by the node that owns the block.

The NVAX can gain ownership and retain it for a very long time. The NVAX CPU monitors the bus continuously for memory space read-type and write commands to memory space by the NCA. When the CPU detects a request for a block that it owns, it will perform the disown write to memory, allowing the original command to complete successfully. Such NDAL transactions, which take place solely for the purpose of maintaining the ownership protocol, are referred to as "cache coherency transactions."

Table 3–15 shows what action is performed in the backup cache based upon the state of the block in the cache when a particular command is driven onto the NDAL by the NCA.

Table 3–15 NVAX Backup Cache Invalidates and Write-backs

NDAL Command	Invalid Block	Valid & Unowned	Valid & Owned
IREAD,DREAD	–	–	Write-back, set Bcache to valid-unowned state
OREAD	–	Invalidate	Write-back, Invalidate
WRITE	–	Invalidate	Write-back, Invalidate
WDISOWN	–	–	–

The I/O devices connected to the NCA through the CP-buses may cause the NCA to access memory on their behalf. As these transactions go to memory, the NVAX CPU recognizes them and performs the appropriate cache coherency action. The NVAX CPU does not acknowledge the commands, since the memory interface is the receiver for the transaction. The NVAX CPU distinguishes cycles driven by devices other than itself by decoding the commander's ID. The ID is driven onto the NDAL along with the command, and recognizes those NCA initiated cycles as cache coherency transactions.

3.11.3 VAX Architecturally-defined Interlocks

A VAX interlocked instruction causes the generation of a read-lock and a write-unlock, which are guaranteed to happen back-to-back. The NDAL does not explicitly define interlocked transactions. Instead, the Ownership Read command is used in place of Read Lock and the Disown Write command is used in place of Unlock Write.

If the interlocked location is already owned in the backup cache, then there is no need to issue an OREAD on the NDAL, and it is serviced directly by the Bcache.

It is also possible for I/O devices to cause the NCA to issue OREAD/WDISOWN pairs on the NDAL for the purpose of performing interlocked access to VAX memory.

3.11.3.1 Ownership and Interlock Transactions

The NVAX CPU does not support interlocks to I/O space. If software attempts to perform an interlocked instruction on an I/O space address, the NVAX CPU will use normal DREAD/WRITE accesses to complete the operation. No interlock is provided.

3.11.4 Errors

The NDAL supports the detection of all single-bit and some multiple-bit transmission-related error conditions on the NDAL_H, CMD_H, ID_H, and PARITY_H lines by implementing parity across those lines. Additionally, the NDAL allows commanders to recover from some memory and I/O-space read/write class errors.

3.11.4.1 Transaction Timeout

The NVAX CPU and NCA implement timeout counters for each read that they may have outstanding. The NVAX implements two timeout counters, one for each possible outstanding read. If a read request times out, it is aborted by the NVAX and a soft error interrupt is generated. Any missing read data return cycles will eventually cause that read to timeout in the NVAX CPU. See Section 4.4.10 for details on how timeout is handled.

Transaction timeout is not a normal occurrence and is expected to happen only on serious system failures.

3.11.4.2 Nonexistent Memory and I/O

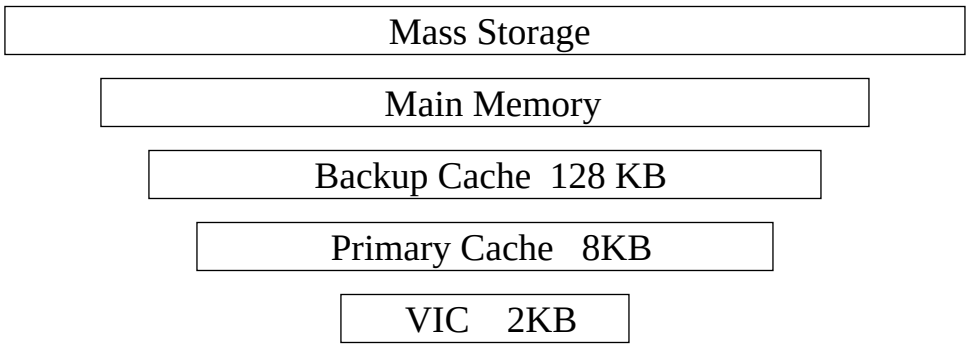
An address that is not implemented in memory on a particular system is known as nonexistent memory. An I/O address of a device that is not present on a particular system is known as nonexistent I/O. When software attempts to access addresses that are nonexistent, an error interrupt will be generated.

4

KA680 Cache Memory Overview

The NVAX memory subsystem has a hierarchical structure. The VIC, Pcache, Bcache, Main Memory, and finally the Mass Storage form the hierarchical memory subsystem of the KA680. The hierarchical order of the levels of KA680 memory is shown in Figure 4–1.

Figure 4–1 KA680 Cache/Memory Hierarchy



For I-stream references, the memory hierarchy starts with the VIC, whereas for D-stream references, the memory hierarchy starts with the Pcache.

References generated by the NVAX CPU are issued to the memory subsystem at the first hierarchical level, as determined by the reference type (I-stream or D-stream). The reference will then pass up through the hierarchy until it is serviced by one of the layers. References serviced at lower layers take less time than references that must pass to higher layers. For this reason, it is the intent of the memory subsystem to service most references within the lower layers, thus maximizing system performance.

By creating successively faster layers of memory hierarchy below the main memory, the KA680 decreases the average amount of time required to access information. Because each layer in the hierarchy tends to be smaller in size than the next higher (slower) layer, there is the problem of allocating space at each layer for storing references. Furthermore, care must be taken to ensure that the state of the system is singularly and accurately represented by the combined contents of the caches and main memory.

In the KA680, this issue is most critical between main memory and the backup and primary caches, because main memory can be accessed by DMA devices as well as the NVAX CPU. Furthermore, this problem is complicated by the write-back nature of the backup cache. This write-back mechanism, while significantly decreasing the latency of write operations, complicates the problem of maintaining a coherent and consistent representation of main memory in DMA traffic.

For example, during normal operation, the NVAX CPU and DMA devices will share access to certain memory locations. In order to guarantee proper operation, the KA680 provides hardware mechanisms to ensure that DMA updates to main memory will be invalidated in the primary and backup caches. Furthermore, the KA680 provides mechanisms to ensure that DMA traffic is presented with correct data in the presence of the write-back cache. The following sections discuss each of the caches and how they are used.

4.1 Cacheable References

Any reference that can be stored by the virtual instruction cache, or the primary or backup caches, 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 = 0) only. They do not store references to the VAX I/O space.

Whenever the CPU generates a noncacheable reference, or a cacheable reference not stored in any of the three caches, a single hexaword reference of the same type is generated on the NDAL bus.

Whenever the CPU generates a cacheable reference that is stored in one of the caches, no reference is generated on the NDAL bus.

4.2 Virtual Instruction Cache

Before any instruction can be executed, it must first be fetched from memory. The NVAX CPU contains an instruction prefetcher, which fetches sequential instructions ahead of the instruction currently being executed. This is done in an attempt to reduce the effective access time of the instruction fetch by pipelining it with decode and instruction execution. The instruction prefetcher maintains an instruction prefetch queue (IPQ) of up to 16 bytes (4 longwords) of I-stream data. In order to fill the IPQ, the prefetcher sends I-stream read requests to the virtual instruction cache.

The virtual instruction cache (VIC) is a 2 KB, direct-mapped cache for caching I-stream data. The VIC is located within the NVAX CPU chip. In order to reduce the overhead associated with virtual-to-physical address translation, the VIC caches references based on virtual addresses. In the event that the virtual references made by the instruction prefetcher hit in the VIC, the I-stream data is loaded from the VIC directly to the IPQ.

If the references made by the instruction prefetcher miss in the VIC, then the VIC will issue an I-stream read request on behalf of the instruction prefetcher to the next level of memory hierarchy: the primary cache.

4.2.1 Virtual Instruction Cache Organization

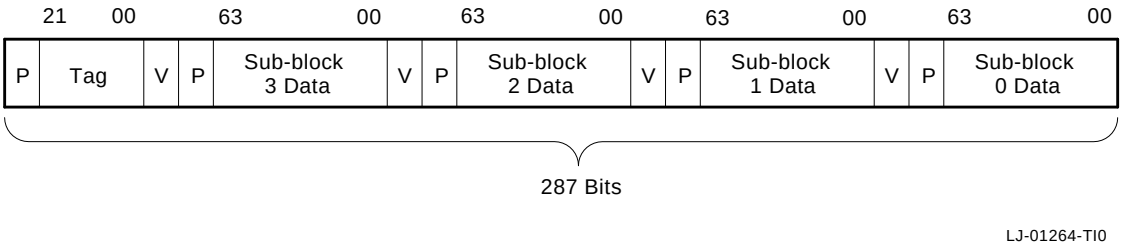
The VIC attributes are summarized in Table 4–1.

Table 4–1 VIC Attributes

Attribute	Description
Cache Size	2 KB
Access Type	Direct-mapped
Block Size	32 bytes
Subblock Size	8 bytes
Valid Bits	4 valid bits/cache block = 1 per subblock
Data Parity Bits	4 even parity bits/cache block = 1 per subblock
No. of Tags	64 tags
Tag Parity Bit	1 even parity bit per tag
Fill Algorithm	Fill forward, random cycle allocate if no tag hit or data subblock valid
Access Size	8 bytes
Bus Size	8 bytes
Prefetching	None
Data Stored	I-stream only
Virtual/Physical	Virtual

The format of each cache row is shown in Figure 4–2. Each cache row stores a 22-bit tag with even parity for the tag, and four quadword subblocks, each with a valid bit and an even parity bit that covers the data only. During a cache read, the data, tags, valid and parity bits of the direct-mapped cache block are read. The tag is compared to bits <31:10> of the virtual address. If the tag matches, then the data is returned to the instruction prefetch queue. Otherwise, the request has "missed" in the VIC, and the read request is forwarded to the Pcache.

Figure 4–2 VIC Cache Row Format



Macrocode Restriction

To avoid parity errors, the VIC tag arrays, including parity, must be written with valid data and parity before enabling the VIC. The tag values in bank 0 must be written with different values than the tag values in bank 1 so that the tags from both banks do not simultaneously produce a tag hit.

4.2.2 Virtual Instruction Cache Internal Processor Registers

The VIC contains four internal processor registers, which provide VIC control and read/write access to the data and tag arrays.

Macrocode Restriction

The VIC_ENABLE bit of the ICSR IPR must be cleared before writing to the other VIC IPRs VMAR, VDATA, or VTAG. Similarly, the VIC_ ENABLE bit of the ICSR IPR must be cleared before reading from the VIC IPRs VDATA and VTAG.

4.2.2.1 VIC Virtual Memory Address Register (VMAR) - IPR 208

Figure 4–3 VMAR Register

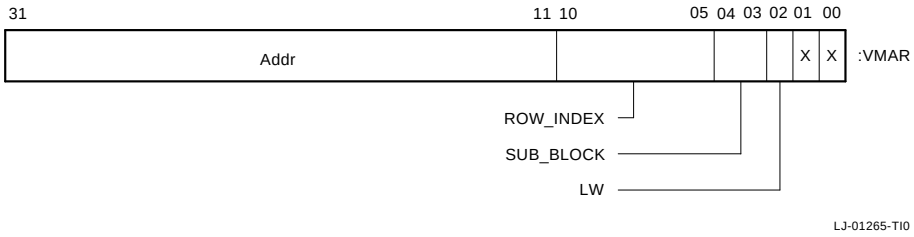


Table 4–2 VMAR Register

Name	Extent	Type	Description
LW	2	WO	Longword select bit. Selects longword of subblock for access to cache array.
SUB_BLOCK	4:3	RW	Subblock select. Selects data subblock for access to cache array; also latches bits <4:3> of the virtual address on VIC parity errors.
ROW_INDEX	10:5	RW	Row select. Row index for read and write access to cache array; also latches bits <9:5> of a virtual address, which resulted in a VIC parity error.

(continued on next page)

Table 4–2 (Cont.) VMAR Register

Name	Extent	Type	Description
ADDR	31:11	RO	Error address field. Latches tag portion of the virtual address, which resulted in a VIC parity error.

When the VIC is disabled, the VIC memory address register (VMAR) may be used as an index for direct IPR access to the cache arrays. Bits <9:5> of this register supply the cache row index, bit <10> selects the bank, bits <4:3> supply the cache subblock, and bit <2> indicates the longword within a quadword address.

The VMAR IPR also latches and holds bits <31:3> of the virtual address of the reference that resulted in a parity error on VIC array parity errors.

4.2.2.2 VIC TAG Register (VTAG) - IPR 209

Figure 4–4 VTAG Register

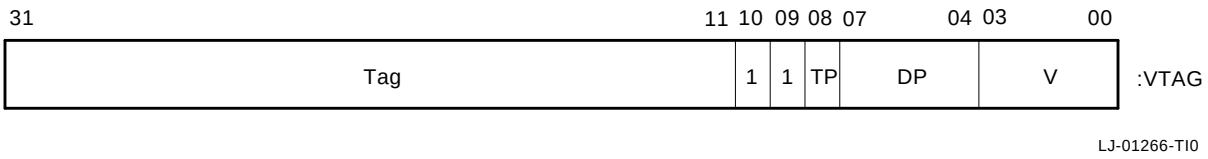


Table 4–3 VTAG Register

Name	Extent	Type	Description
V	3:0	RW	Data valid bits. Supply data valid bits on array read/writes.
DP	7:4	RW	Data parity bits. Supply data parity on array read/writes.
TP	8	RW	Tag parity bit. Supplies tag parity on tag array read/writes.
TAG	31:11	RW	Tag. Supplies tag on tag array read/writes.

The VTAG IPR provides read and write access to the cache tag array. An IPR write to VTAG will write the tag, parity, and valid bits for the row indexed by VMAR<9:5> and the bank selected by VMAR<10>. VTAG<31:10> are written to the cache tag. VTAG<8> is written to the associated tag parity bit. VTAG<7:4> are used to write the four data parity bits associated with the indexed cache row. Similarly, VTAG<3:0> write the four data valid bits associated with the cache row. VTAG<7:4> and VTAG<3:0> are the data parity and data valid bits, respectively, for the four quadwords of data in the same row. VTAG<4> and VTAG<0> correspond to the quadword of data addressed when address bits 4:3 = 00; VTAG<5> and VTAG<1> correspond to the quadword of data addressed when address bits 4:3 = 01, and so forth.

4.2.2.3 VIC Data Register (VDATA) - IPR 210

Figure 4–5 VDATA Register

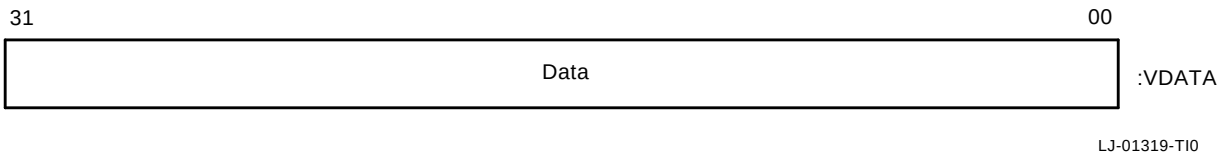


Table 4–4 VDATA Register

Name	Extent	Type	Description
DATA	31:0	RW	Data for data array reads and writes

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

Writes to VDATA with VMAR<2> = 0 simply accumulate the IPR data destined for the low longword of a subblock in a scratch register internal to the NVAX CPU. A subsequent write to VDATA with VMAR<2> = 1 triggers a cache write to the subblock indexed by VMAR.

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

4.2.2.4 VIC Control and Status Register (ICSR) - IPR 211

Figure 4–6 ICSR Register

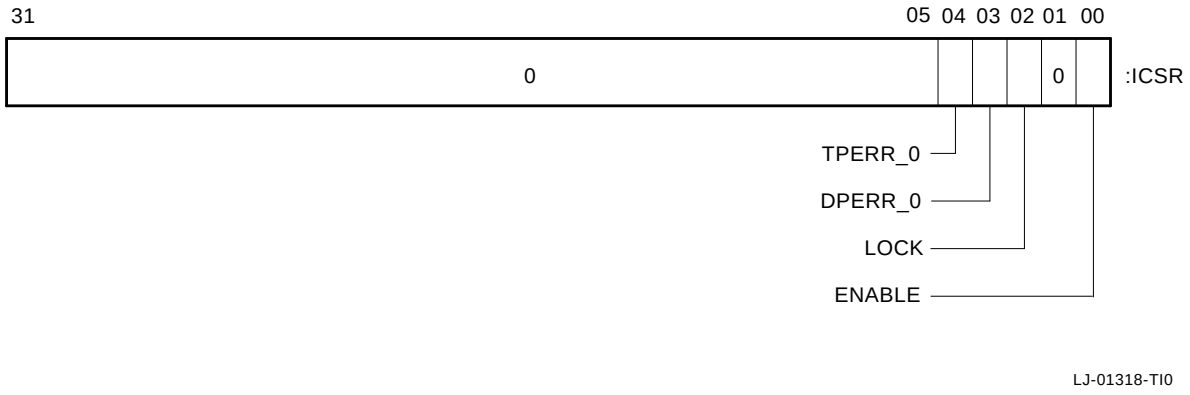


Table 4–5 ICSR Register

Name	Extent	Type	Description
ENABLE	0	RW,0	Enable bit. When set, allows cache access to the VIC. Initializes to 0 on system reset.
LOCK	2	WC	Lock bit. When set, validates and prevents further modification of the error status bits in the ICSR and the error address in the VMAR register. When clear, indicates no VIC parity error has been recorded and allows ICSR and VMAR to be updated.
DPERR	3	RO	Data error bit. When set, indicates data parity error occurred in data array.
TPERR	4	RO	Tag error bit. When set, indicates tag parity error occurred in tag array.

The ICSR IPR provides control and status functions for the VIC. VIC tag and data parity errors are latched in the read-only bits ICSR<4:3>, respectively. ICSR<2> is set when a tag or data parity error occurs, and keeps the error status bits and the VMAR IPR from being modified further. Writing a logic one to ICSR<2> clears the lock bit and allows the error status to be updated. When ICSR<2> is clear, the values in ICSR<4:3> are meaningless. When ICSR<2> is set, a VIC parity error has occurred, and either ICSR<4> or ICSR<3> will be set. This indicates that the parity error was either a tag parity error or a data parity error, respectively. ICSR<4:3> cannot be cleared from software. ICSR<0> provides IPR control of the VIC enable. It is cleared on system reset.

4.3 Primary Cache

The Pcache is a 2-way set associative, read allocate, no-write allocate, write-through, physical address cache of I-stream and D-stream data. It stores 8192 bytes (8K) of data and 256 tags corresponding to 256 hexaword blocks (1 hexaword = 32 bytes). Each tag is 20 bits wide, corresponding to bits <31:12> of the physical address.

There are four quadword subblocks per block with a valid bit associated with each subblock. The access size for both Pcache reads and writes is one quadword. Byte parity is maintained for each byte of data (32 bits per block). One bit of parity is maintained for every tag. The Pcache has a 1-cycle access and a 1-cycle repetition rate for both reads and writes.

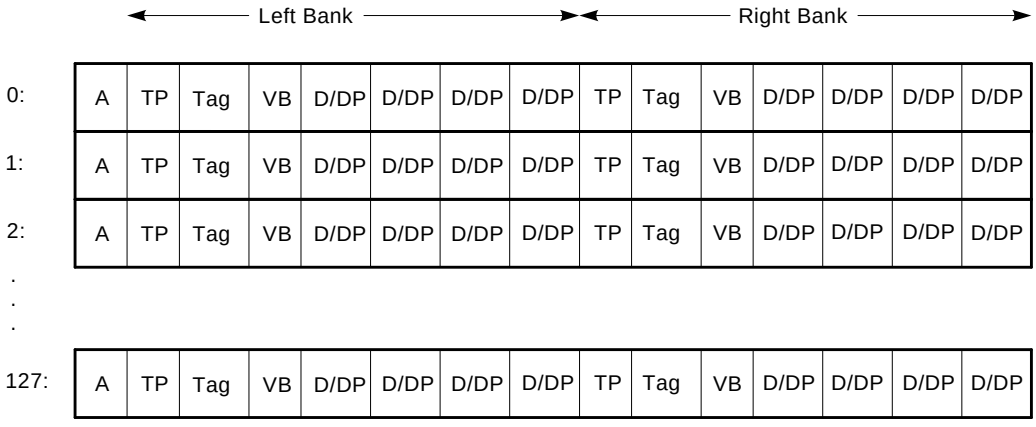
The Pcache represents the first level of D-stream memory hierarchy and the second level of I-stream memory hierarchy in all NVAX computer systems. Pcache entries must be invalidated in order to maintain cache coherency with higher levels of the memory hierarchy. See Section 4.3.2 for more information on the Pcache.

The Pcache is located within the NVAX CPU chip. Unlike the VIC, the Pcache is based on physical addresses rather than virtual addresses. The Pcache handles I-stream requests from the VIC, as well as D-stream requests for instruction operands. The Pcache uses a write-through scheme for handling writes to memory locations that are contained in the Pcache. In this scheme, the write operation updates the contents of the Pcache, and the write operation is propagated to the next level of memory hierarchy: the backup cache. The Pcache is maintained as a strict subset of the backup cache.

4.3.1 Primary Cache Organization

Figure 4–7 shows the logical organization of the Pcache.

Figure 4–7 Logical Pcache Organization



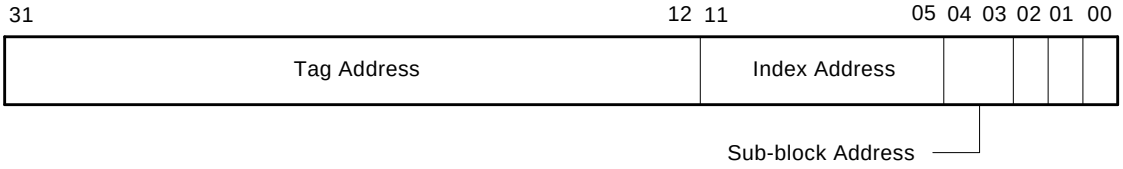
Where: A = Allocation bit. Indicates whether the left of right bank was last allocated.
TP = 1 bit of even tag parity.
Tag = 20 bits of tag address.
VB = 4 valid bits. each bit corresponds to 8 bytes of data.
D/DP = 8 bytes of data with 8 bits of even byte parity (72 total bits).

LJ-01267-TI0

The Pcache is logically organized into 128 direct-mapped indexes, where each index consists of two blocks, and each block consists of: 20-bit tag, 1-bit tag parity, 4 valid bits, 256 bits of data, and 32 bits of data parity. In addition, each index also contains a 1-bit allocation pointer.

The breakdown of address bits for Pcache decoding is shown below:

Figure 4–8 Pcache Address Breakdown



Where: Tag Address = Bits loaded into or compared with tag.
Index Address = Address 1 of 128 indexes.
Sub-block Address = Addresses 1 of 4 aligned quadwords within the hexaword data block.

LJ-01268-TI0

4.3.2 Pcache Control

The Pcache is controlled through the PCCTL internal processor register. This IPR controls whether the Pcache is enabled or disabled, as well as controlling the general mode of operation. As with other internal processor registers, it is accessible through the MTPR and MFPR instructions. The following diagram shows the organization of the PCCTL internal processor register.

Figure 4–9 PCCTL Register

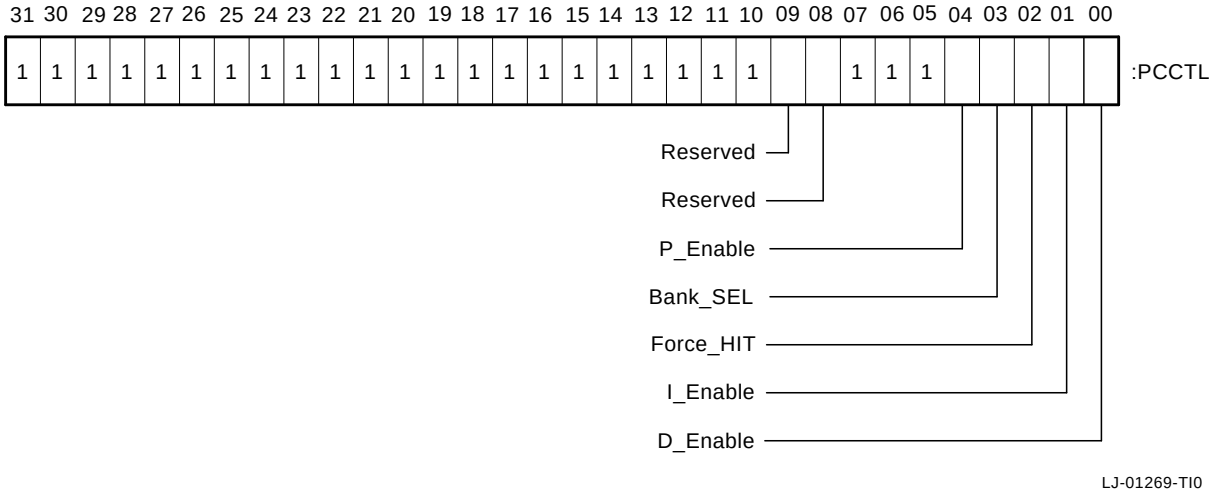


Table 4–6 PCCTL Definition

Name	Extent	Type	Description
D_ENABLE	0	RW,0	When set, enables Pcache for all invalidate operations and for all D-stream read/write/fill operations. Qualified by other control bits. When clear, forces a Pcache miss on all Pcache D-stream read/write/fill operations. Note, however, that an ACV/TNV/M=0 condition overrides a deasserted D_ENABLE because it will force a Pcache hit condition with D_ ENABLE=0.
I_ENABLE	1	RW,0	When set, enables Pcache processing of invalidate, I-stream read and I-stream cache fills. When clear, forces a Pcache miss on I-stream read operations and prevents state modification due to an I-stream cache fill operation. Note, however, that an ACV/TNV/M=0 condition overrides a deasserted I_ ENABLE because it will force a Pcache hit condition with I_ENABLE=0.
FORCE_HIT	2	RW,0	When set, forces a Pcache hit on all reads and writes when Pcache is enabled for I- or D-stream operation. This is used for diagnostic purposes so that the cache data store can be directly accessed.
BANK_SEL	3	RW,0	When set with FORCE_HIT=1, selects the "right bank" of the addressed Pcache index.

(continued on next page)

Table 4–6 (Cont.) PCCTL Definition

Name	Extent	Type	Description
			When clear with FORCE_HIT=1, selects the "left bank" of the addressed Pcache index. This bit is a don't care when FORCE_ HIT=0.
P_ENABLE	4	RW,0	When set, enables detection of Pcache tag and data parity errors. When deasserted, disables Pcache parity error detection.
Reserved	7:5	R,1	Unused. Read as ones.
Reserved	8	R,0	–
Reserved	9	R,0	Pcache redundancy bit. When set, this bit indicates that one or more Pcache redundant elements have been enabled.

Note that Pcache operation is further qualified by the state of PCSTS<0>. If this bit is nonzero, Pcache operation is automatically forced to behave as if I_ ENABLE=0 and D_ENABLE=0, regardless of the actual state of I_ENABLE and D_ENABLE. Effectively, this shuts down normal Pcache operation due to the presence of a previous Pcache parity error.

Based on the bit definitions above, note that Pcache invalidate operations are only disabled if both D_ENABLE=0 and I_ENABLE=0, or if PCSTS<0> is set. Also note that Pcache IPR read and write operations are always unconditionally enabled, regardless of the state of I_ENABLE or D_ENABLE, or PCSTS<0>.

If either D_ENABLE or I_ENABLE are to be toggled to the on state, the Pcache array must be initialized prior to such action.

When the FORCE_HIT (force hit) bit is set and I-stream or D-stream operation is enabled, all enabled memory space read and write references are forced to hit in the Pcache. The BANK_SEL bit specifies which tag of the pair of tags addressed is forced to hit. Thus when FORCE_HIT=1, the Pcache becomes a 4K direct-mapped cache with all reads and writes forced to hit in the Pcache. Toggling BANK_SEL causes the other half of the 8K Pcache to become accessible in this direct mapped mode. Note that the FORCE_HIT bit only affects memory space references. I/O space references still miss in the Pcache regardless of the state of the FORCE_HIT bit.

The FORCE_HIT feature is designed to facilitate testing the Pcache data array and to make diagnostic tests easily loadable within the Pcache by simple memory write operations. When FORCE_HIT=0, the Pcache is configured as an 8K 2-way set associative cache, no reads or writes are forced to hit, and the BANK_SEL bit is a don't care.

The P_ENABLE (parity enable) bit allows the detection of Pcache tag and data parity errors to be enabled or disabled. If P_ENABLE=0, Pcache parity errors will not be detected. Thus when P_ENABLE=0, no Pcache error will be recorded in PCSTS, nor will they cause an error interrupt or machine check.

Note however, that when FORCE_HIT=1, Pcache tag parity is never checked regardless of the state of P_ENABLE.

4.3.3 Pcache Hit/Miss Determination

4.3.3.1 Hit/Miss Determination by Tag Comparison

For I-stream or D-stream reads or writes, the Pcache must determine if the referenced data is present in its array. To do this, physical address bits <11:5> are input to the Pcache row decoders in order to determine which one of the 128 direct-mapped indexes is being addressed. Subsequently, all 627 bits within the addressed index are accessed by the assertion of the corresponding word line. The two accessed tag values are simultaneously compared to physical address bits<31:12>. A Pcache hit condition occurs when all of the following conditions are simultaneously true:

- The contents of one of the two addressed tags matches the data on physical address bits <31:12>.
- The valid bit corresponding to both the matched tag and to the addressed quadword subblock (specified by physical address bits<4:3>) is set.
- The stored tag parity corresponding to the matched tag is the same as the value calculated from bits <31:12>.

If an address match is detected on one of the tags and the valid bit that corresponds to both the matched tag and the addressed subblock (specified by physical address bits <4:3>) is set, then a Pcache hit condition has been detected on the corresponding Pcache tag. The absence of the Pcache hit condition causes a Pcache miss condition.

4.3.3.2 Conditions That Force Pcache Miss

The Pcache miss condition is forced to override the tag determination of hit/miss described above when any one of the following conditions is satisfied:

- If PCSTS<0> is set, the Pcache miss condition is forced due to a previous Pcache parity error.
- If an I-stream read or cache fill operation is accessing the Pcache and I_ ENABLE=0, the Pcache miss condition is forced.
- If a D-stream read or cache fill operation is accessing the Pcache and D_ ENABLE=0, the Pcache miss condition is forced.
- If a D-stream read lock operation is executing (such as for an interlocked instruction), the Pcache miss condition is forced. This guarantees that the read will propagate to the Bcache where memory ownership can be obtained for synchronization purposes.
- If an I-stream cache fill operation is executing, but the reference is noncacheable, the Pcache miss condition is forced.
- If a D-stream cache fill operation is executing, but the reference is noncacheable, the Pcache miss condition is forced.

4.3.3.3 Conditions That Force Pcache Hit

The Pcache hit condition is forced to override the tag determination of hit/miss described above when any one of the following conditions is satisfied:

- If a read or write reference has a memory management fault or hard error associated with it, a Pcache hit condition is forced. **NOTE: This force hit condition takes precedence over any force miss condition described above.**
- If the operation is a D-stream read, write or WRITE_UNLOCK, and D_ ENABLE=1 and FORCE_HIT=1, the Pcache hit condition is forced on the tag corresponding to both the addressed Pcache index and the bank specified by the BANK_SEL bit.
- If the operation is an I-stream read and I_ENABLE=1 and FORCE_HIT=1, the Pcache hit condition is forced on the tag corresponding to both the addressed Pcache index and the bank specified by the BANK_SEL bit.

4.3.4 Pcache Behavior on Write Operations

A Pcache write operation is initiated by a write or WRITE_UNLOCK reference. A Pcache write begins by determining the Pcache hit or miss condition described above. If a Pcache hit is detected, the data is selectively written into the quadword corresponding to both the tag in which the hit occurred and to physical address bits <4:3>. The data is selectively written according to the length specified in the instruction causing the write. The corresponding data parity is also written in the same manner for each corresponding byte that is written.

If a Pcache miss condition occurs, no Pcache write operation takes place. However, the write reference is forwarded to the Bcache for processing regardless of the hit/miss condition in the Pcache.

4.3.5 Pcache Replacement Algorithm

When a Pcache miss occurs during a read operation, it must be decided which one of two blocks will be allocated for the subsequent Pcache fill sequence. When the Pcache miss occurred because no validated tag field matched the read address, the state of the corresponding allocation bit indicates which bank (left or right) should be used for the resulting fill sequence. The value of each allocation bit changes according to the "not-last-used" algorithm. That is, the allocation bit always points to the bank within the index that was not last accessed.

When a read miss occurs because no validated tag field matched the read address, the value of the allocation bit will be used as the bank select input during the subsequent fill sequence. As each fill operation takes place (that is, as each quadword comes back from memory), the allocation bit is written to point to the other bank. This ensures that the next fill operation to this cache entry will be written to the other bank. Also, during Pcache read or write operations, the value of the allocation bit is set to point to the opposite bank that was just referenced because this is now the new "not-last-used" bank.

The one exception to this algorithm occurs during an invalidate. Pcache invalidates occur because of I/O activity in the system and the need to maintain the Pcache as a strict subset of the Bcache. When a Pcache invalidate clears the valid bits of a particular tag within an index, set the allocation bit to point to the bank select used during the invalidate regardless of which bank was last allocated. By doing so, it is guaranteed that the next allocated block within the index will not displace any valid tag because the allocation bit points to the tag that was just invalidated.

4.3.6 Pcache Fill Operation

A Pcache fill operation is initiated by an I-stream or D-stream read operation that missed in the Pcache. A fill is functionally identical to a Pcache write operation from write reference except for the following differences:

- The bank within the addressed Pcache index is selected by the following algorithm. If a validated tag field within the addressed index matches the cache fill address, then the block corresponding to this tag is used for the fill operation. If this is not true, then the value of the corresponding allocation bit selects which block will be used for the fill.
- The first fill operation to a block causes all four valid bits of the selected bank to be written so that the valid bit of the corresponding fill data is set and the other three are cleared. All subsequent fills cause only the valid bit of the corresponding fill data to be set.
- Any fill operation causes the fill address bits <31:12> to be written into the tag field of the selected bank. Tag parity is also written in an analogous fashion.
- A fill operation causes the allocation bit to be written with the complement of the value it had at the time of the reference that missed and caused the fill sequence.
- A fill operation forces every bit of the corresponding byte mask field to be set. Thus, all eight bytes of fill data are always written into the Pcache array on a fill operation.

4.3.7 Pcache Invalidate Operation

A Pcache invalidate operation is initiated by I/O activity to main memory and the need to maintain the Pcache as a strict subset of the Bcache. During I/O activity, the I/O devices may update main memory, which is cached in the Bcache and possibly the Pcache. When such references occur, the caches must invalidate their copies of the referenced memory location, because they are now stale. The invalidate operation is interpreted as a NOP by the Pcache if the address does not match either tag field in the addressed Pcache index. If a match is detected on either tag, an invalidate will occur on that tag. Note that this determination is made based only on a match of the tag field bits rather than on satisfying all criteria for the Pcache hit condition (Pcache hit factors in valid bits and verified tag parity into the equation).

When an invalidate is to occur, the four valid bits of the matched tag are written with zeros and the allocation bit is written with the value of the bank select used during the current invalidate operation.

Also note that an uncorrectable error, either in the Bcache or in main memory during a cache fill sequence, causes the cache fill operation to be processed as if it were an INVAL operation.

4.3.8 Pcache IPR Summary

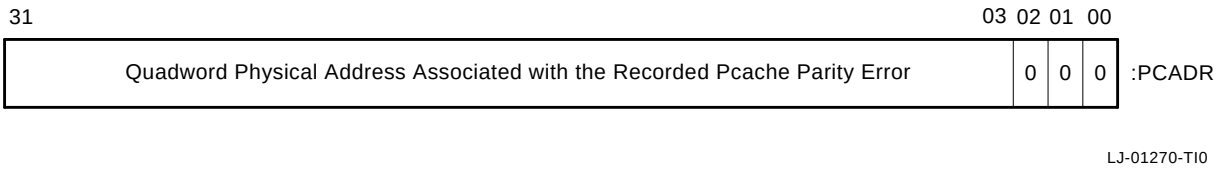
The following table summaries all IPRs associated with the Pcache:

Table 4–7 Pcache IPRs

Register Name	IPR Address (in hex)
PCADR (quadword address of reference causing Pcache parity error)	F0
PCSTS (status of Pcache parity error)	F1
PCCTL (control state of Pcache operation)	F2
PCTAG	01800000..01801FE0
PCDAP	01C00000..01C01FF8

4.3.8.1 PCADR - IPR 240

Figure 4–10 PCADR Register



4.3.8.2 PCSTS - IPR 241

Figure 4–11 PCSTS Register

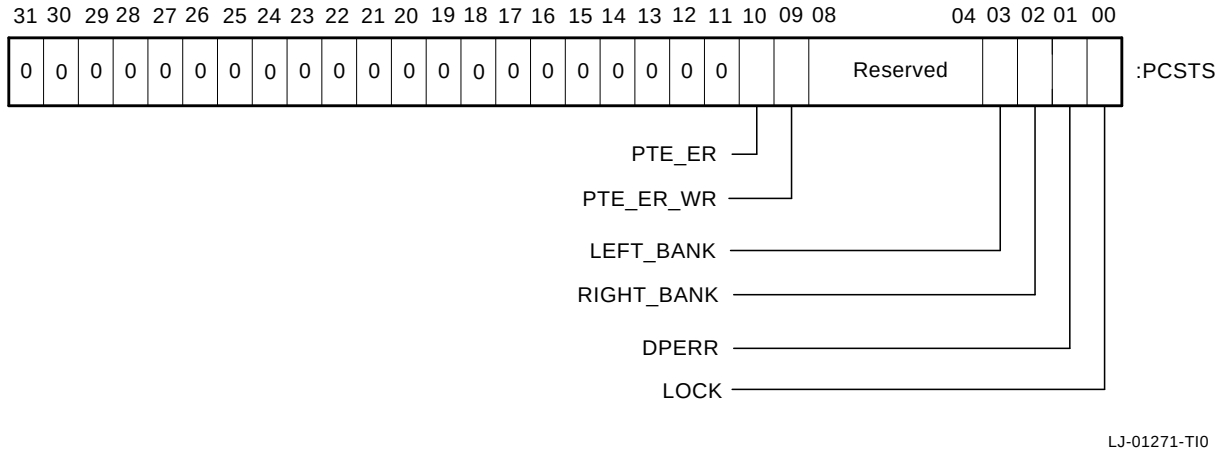


Table 4–8 PCSTS Description

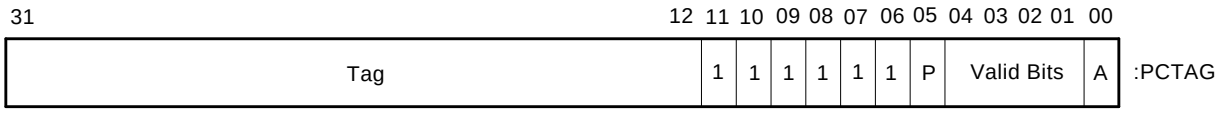
Name	Extent	Type	Description
LOCK	0	WC,0	Lock bit. When set, validates PCSTS<8:1> contents and prevents modification of these fields. When clear, invalidates PCSTS<8:1> and allows these fields and PCADR to be modified.
DPERR	1	RO	Data error bit. When set, indicates a Pcache data parity error.
RIGHT_BANK	2	RO	Right bank tag error bit. When set, indicates a Pcache tag parity error on the right bank.
LEFT_BANK	3	RO	Left bank tag error bit. When set, indicates a Pcache tag parity error on the left bank.
RESERVED	8:4	RO	–
PTE_ER_WR	9	WC,0	Indicates a hard error on a data read from a page table entry, which resulted from a TB miss on a write or WRITE_ UNLOCK.
PTE_ER	10	WC,0	Indicates a hard error on a data read from a page table entry.

4.3.8.3 PCCTL - IPR 242

See Figure 4–9 in Section 4.3.2 for information regarding the format and use of this register.

4.3.8.4 PCTAG - IPRs 01800000_{16} to $01801FE0_{16}$

Figure 4–12 PCTAG Register



LJ-01272-TI0

Table 4–9 Pcache Tag IPR Format

Name	Extent	Type	Description
A	0	RW	Allocation bit corresponding to index of this tag
Valid bits	4:1	RW	Valid bits corresponding to the four data subblocks PCTAG<4> corresponds to uppermost quadword in block PCTAG<1> corresponds to lowermost quadword in block
P	5	RW	Even tag parity
Tag	31:12	RW	Tag data

4.3.8.5 PCDAP - IPR $01C00000_{16}$ to $01C01FF8_{16}$

Figure 4–13 PCDAP Register

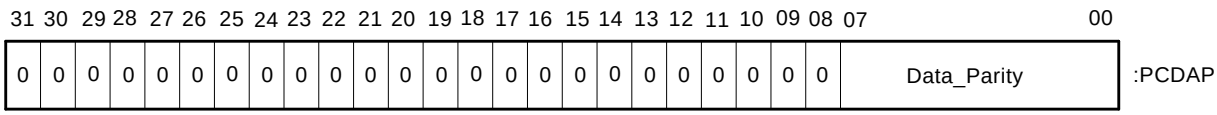


Table 4–10 Pcache Data Parity IPR Format

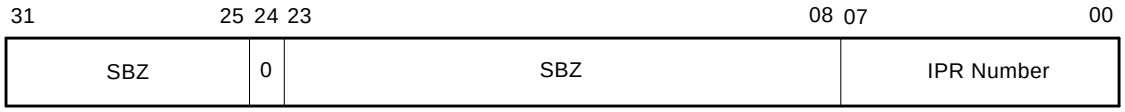
Name	Extent	Type	Description
DATA_PARITY	7:0	RW	Even byte parity corresponding to addressed quadword of data. Bit n represents parity for byte n of addressed quadword.

4.3.9 Pcache IPR Access

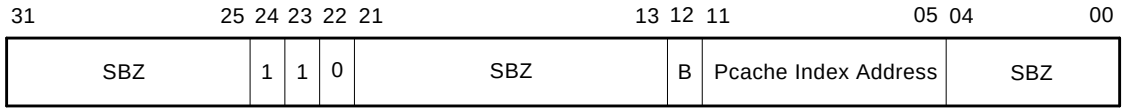
For testability reasons, it is important to verify that every Pcache storage bit can be read and written in both "0" and "1" states. The easiest way to do this is to directly read and write every bit in the Pcache array. The data field is accessible through reads and writes to addresses that hit in the Pcache. (The hit condition can be forced through setting the FORCE_HIT bit in the PCCTL IPR.) The tag field, tag parity, valid bits, and data parity are directly accessible through MTPR and MFPR instructions to the Pcache IPRs defined below.

Figure 4–14 IPR Address Space Mapping

Normal IPR Address

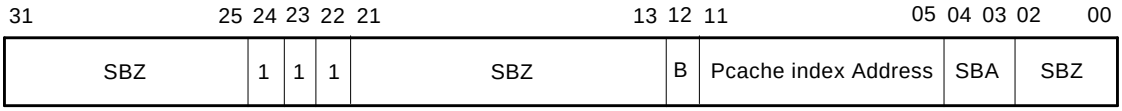


Pcache TAG IPR Address



Where B = 0 → Select the Left Bank of the Specified Index
1 → Select the Right Bank of the Specified Index

Pcache Date Parity IPR Address



Where B = 0 → Select the Left Bank of the Specified Index
1 → Select the Right Bank of the Specified Index

SBA = Sublock Address Selection

LJ-01274-TI0

The tag parity bit is included in the Pcache tag IPR format to allow the user to write bad tag parity into the array in order to verify the tag parity logic. Further, the valid bits and allocation bit are also included so that the Pcache can be initialized to a known state.

The Pcache data parity allows the Pcache data parity to be directly read and written for testability purposes.

4.4 Backup Cache

The following sections describe the organization and operation of the backup cache.

4.4.1 Write-back Cache and Ownership Concepts

There is one fundamental difference between a write-back cache, such as the Bcache, and a write-through cache, such as the Pcache. When a write is received by a write-through cache, the data may or may not be written into the cache, but is always written to memory. When a write is received by a write-back cache, the write is not necessarily forwarded to memory; the write may be done only into the cache. The data is written back to memory only if another element in the system needs that data, or if the block is displaced (deallocated) from the cache.

The NVAX backup cache is a write-back design in which a cache block may exist in one of three states: invalid, valid-unowned, and valid-owned. A block that is valid-unowned is a read-only copy of memory data. A block that is valid-owned may be written by NVAX, and if it has been written since being put into the cache, is the only up-to-date copy of the data in the system. The NVAX cache makes no distinction between valid-owned blocks it has written and those it has not written.

The KA680 system supports the concept of memory ownership by implementing an ownership bit for every hexaword of main memory. When this memory receives an ownership read (OREAD) for a hexaword (due to a Bcache write miss or for VAX interlocked instructions), ownership is passed to the NVAX as the data is returned from the memory subsystem. If another read request arrives for that hexaword from a DMA device, memory does not return the data since the hexaword is not owned by memory but by the NVAX. The NVAX CPU recognizes the second read request as a cache coherence transaction and writes back the data from its cache, using the Disown Write (WDISOWN) command. The memory subsystem will then forward the data to the requesting DMA device.

During write operations, the NVAX CPU issues an OREAD to the memory subsystem and receives ownership for blocks that miss in the Bcache. After gaining ownership of the hexaword and allocating it into the Bcache, the NVAX CPU performs the desired write to that block in the backup cache. The NVAX CPU relinquishes ownership of the data by performing a WDISOWN write-back of the block when it sees an access to that hexaword on the NDAL bus from a DMA device.

4.4.2 Backup Cache Overview

The backup cache (Bcache) is direct-mapped, with quadword access size and a hexaword (32 bytes) block size. The Bcache allocates on reads and writes, and uses a write-back protocol. Bcache tags and cache data are stored in static RAMs that reside on the CPU module. The NVAX CPU implements the control for the Bcache tags and data.

The Bcache and Pcache communicate internally to the NVAX CPU in such a way as to maintain the Pcache as a strict subset of the Bcache. This is done through the use of "invalidate" commands sent automatically from the Bcache to the Pcache whenever the Bcache must invalidate a block. The Bcache will invalidate a block in response to DMA activity to the corresponding memory location, or to make room in the cache for new data. In the case of Bcache blocks that contain data for NVAX-owned memory locations, the process of invalidating the block involves a write-back of the data contained in the cache to the corresponding memory location. The write-back operation simultaneously relinquishes ownership of the hexaword.

The flow of memory transactions within the NVAX CPU is as follows:

I-stream read requests generated within the NVAX are first issued to the VIC. If the data is found there, then the request is satisfied. If the request misses in the VIC, then the request is forwarded to the Pcache. Since the VIC stores only I-stream data, all D-stream requests bypass the VIC and start at the Pcache. If the request hits in the Pcache, then the NVAX uses this data. Otherwise, the request is forwarded to the Bcache. If the read request hits in the Bcache, then the NVAX uses this data and simultaneously copies it into the Pcache to speed up future references to this data. If the request misses in the Bcache as well, then an IREAD or DREAD cycle is issued to the memory subsystem on the NDAL bus. When the data comes back from the memory subsystem, the Pcache, Bcache, and possibly the VIC each allocate a block and fill it with this data. Note that the VIC caches only I-stream data. Likewise, reads that hit in the Bcache will also cause fills in the Pcache, and if I-stream, the VIC as well.

Normal (that is, not disown writes) write transactions generated within the NVAX CPU cause cache lookups in the Pcache and Bcache. If the location to be written is found in the Bcache **and is marked in the Bcache as owned by the NVAX CPU**, then the write transaction will update only the Bcache and Pcache entries and will not be sent to the memory subsystem. Note that on writes, the Pcache is updated only if the referenced location was already cached in the Pcache.

If the location to be written is cacheable but is not currently cached in the Bcache, **or is cached but not marked as owned by the NVAX CPU**, then the NVAX CPU will issue an OREAD on the NDAL bus to gain ownership of the location. When the memory subsystem returns the data corresponding to the OREAD, the NVAX CPU will allocate a block in the Bcache if necessary, and will set the corresponding ownership bit in the Bcache. The setting of this ownership bit in the Bcache indicates to the NVAX CPU that it may freely update the contents of the cache without compromising the consistency of memory. The write transaction that initially caused the OREAD then completes, updating only the Bcache and Pcache entries. Since the data is now owned by the NVAX CPU, subsequent writes to that location will not result in NDAL cycles, since the data is only updated in the cache. The resulting speedup of write transactions and the conservation of NDAL bus bandwidth are the primary reasons for using the write-back algorithm in the Bcache. Note that under normal conditions, the memory ownership mechanism and the write-back nature of the Bcache is totally transparent to software.

4.4.3 Backup Cache Operating Modes

The backup cache has four distinct modes of operation.

- Cache on. Normal operation.
- Cache off. Reset puts the backup cache into the OFF state. The backup cache may be enabled/disabled (turned ON/OFF) by software through the Backup Cache Control IPR. Cache off mode is described in Section 4.4.13.1.
- Force hit. This forces all memory space reads and writes forwarded to the Bcache to hit in the Bcache. This mode is used for testing and initialization purposes. Force hit mode is described in Section 4.4.13.2.
- Error transition mode. The Bcache enters error transition mode (ETM) upon recognition of some error conditions or when put into ETM explicitly by an IPR write. Error transition mode is described in Section 4.4.13.3.

4.4.4 NVAX Backup Cache Organization and Interface

The NVAX backup cache is configurable based on the size and speed of the cache RAMs used to implement the cache on the board. Because of this, the configuration register must be written with the appropriate information based on the size and speed of the Bcache RAM chips used on the KA680. This function is performed by the console firmware.

The KA680 has a 128 kilobyte Bcache. The NVAX CPU provides a control register by which it can be configured with the size of the Bcache. This is controlled by the SIZE field in the CCTL register, as described in Section 4.4.8.1. Firmware configures the NVAX CPU to operate with the amount of Bcache memory provided on the module.

Table 4–11 Backup Cache Size and RAMs Used

Cache Size	Tag RAM Size	Data RAM Size	Number of Tags	Valid Bits Per Tag
128 kilobytes	4K x 4	16K x 4	4K	1

The cache has a block size of 32 bytes and has no subblocks. The data bus to the cache is 8 bytes wide, so in order to read out an entire block, 4 accesses are done. Each block contains 32 bytes of data and has associated with it a tag, a valid bit, and an owned bit. ECC protection is provided on each quadword in the cache. ECC protection is also provided on the tag store.

Each address bit serves either as an index bit or as a tag bit. Table 4–12 shows how the bits are used.

Table 4–12 Tag and Index Interpretation Based on Cache Size

Cache Size	Tag Bits Used	Index Bits Used
128 kilobytes	Tag<28:17>	Index<16:5>

Note that bits <31:29> are "don't cares" with respect to the mapping of tags and index bits because the KA680 is operated with the NVAX CPU in 30-bit address mode.

Bit <29> is a "don't care" because cacheable references always have bit <29>=0. For example, they are always in VAX memory space. With the NVAX CPU in 30-bit mode, Bit <29>=1 indicates I/O space references, which are not cacheable.

The NVAX CPU also requires software to program the backup cache speed as dictated by the speed of the RAM chips used on the board. The TAG_SPEED and DATA_SPEED fields of the Bcache control register, CCTL, are used to control the number of NVAX cycles used by the Cbox to access the RAMs. These bit fields are written by firmware with the values appropriate to the module configuration.

4.4.5 Backup Cache Block Diagrams

Figure 4–15 and Figure 4–16 show the connections to the tag store and data RAMs, and the way the address is used for the 128-kilobyte cache used on the KA680.

Figure 4–15 Tags and Data for 128-Kilobyte Cache

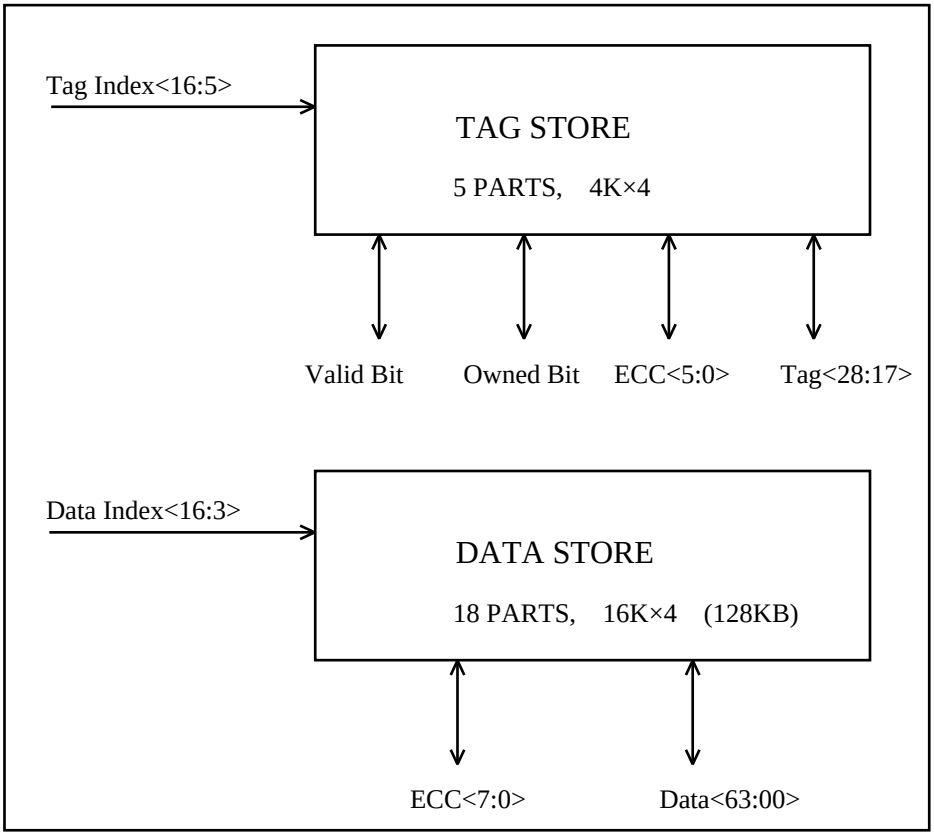
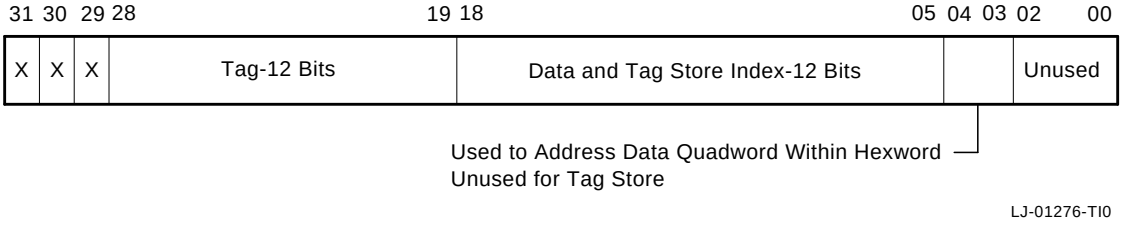


Figure 4–16 Address Used for 128-Kilobyte Cache



4.4.6 Backup Cache Data Block Allocation

4.4.6.1 Read References

On cacheable read references that are not in the Pcache, the request is forwarded to the Bcache internally to the NVAX CPU. If the requested quadword cannot be found in the Bcache, a read request is sent from the NVAX CPU to the memory subsystem. The length of the request is a hexaword, which is the same size as the Bcache and Pcache block size. Before the request is sent, however, the Bcache and Pcache each allocate a block for the new data. When the data returns from memory, copies are made in both the Bcache and Pcache to reduce latency on future references. Note that if the Bcache block being displaced by the new data is valid **and has its owned bit set**, then the displaced block will be written back to memory via a disown write transaction on the NDAL bus **before** the read for the cache fill is issued. This updates the displaced hexaword in memory with the possibly modified cached copy, and relinquishes ownership of that hexaword location. Since the Bcache is direct-mapped, that is, it has only one bank, there is no need to decide which bank to allocate for the fill.

4.4.6.2 Write References

The Bcache performs allocates on writes. Write references to cacheable memory locations do not propagate directly out to memory unless the Bcache has been disabled by clearing CCTL<ENABLE>. If the hexaword being written is contained in the Bcache and is owned by the NVAX CPU, then the write is done only to the Bcache entry. If the referenced hexaword is not in the Bcache, or if it is in the Bcache but is not owned by the NVAX CPU, then ownership of the hexaword will be obtained by issuing an ownership read on the NDAL bus to the memory subsystem. Once ownership has been obtained, the write will proceed as described above, updating only the Bcache.

Write references will update the Pcache only if the reference is already cached there. The Pcache does not perform allocations on writes.

4.4.7 Effects of I/O Traffic on the Backup Cache

In the following paragraphs, the term "DMA device" or "DMA traffic" refer to the Q22–bus, DSSI bus, or Ethernet interfaces.

The NVAX CPU monitors all DMA traffic to main memory. For each reference, the NVAX CPU will check the contents of the backup cache to see if it contains the hexaword being referenced. If the referenced hexaword is not in the Bcache, then the NVAX CPU takes no further action. If, however, the referenced hexaword is currently cached, the specific actions taken thereafter depend on the type of DMA reference and whether the referenced hexaword is owned by the NVAX CPU.

If the Bcache contains the hexaword referenced by the I/O device, but the Bcache block is not marked as "owned," then the NVAX CPU will invalidate the Bcache entry only if the DMA reference was a write. Note that in this case the Pcache is also checked for the referenced hexaword, and the appropriate Pcache block is invalidated if found. No cache entries are invalidated due to reads from DMA devices.

If the Bcache contains the hexaword referenced by the I/O device, and the Bcache block is marked as "owned," then two things happen. First, when the reference from the DMA device is received by the memory subsystem, the reference will be pended in the NMC memory controller because the referenced hexaword is

not currently owned by the memory subsystem. Simultaneously, the NVAX CPU determines that it owns the hexaword referenced by the DMA device, and will write back the Bcache block to memory using an ndal disown write transaction (WDISOWN), relinquishing ownership of the hexaword in the process. The memory controller will respond to the disown write by updating the contents of main memory with the value from the Bcache, then allowing the DMA reference to complete.

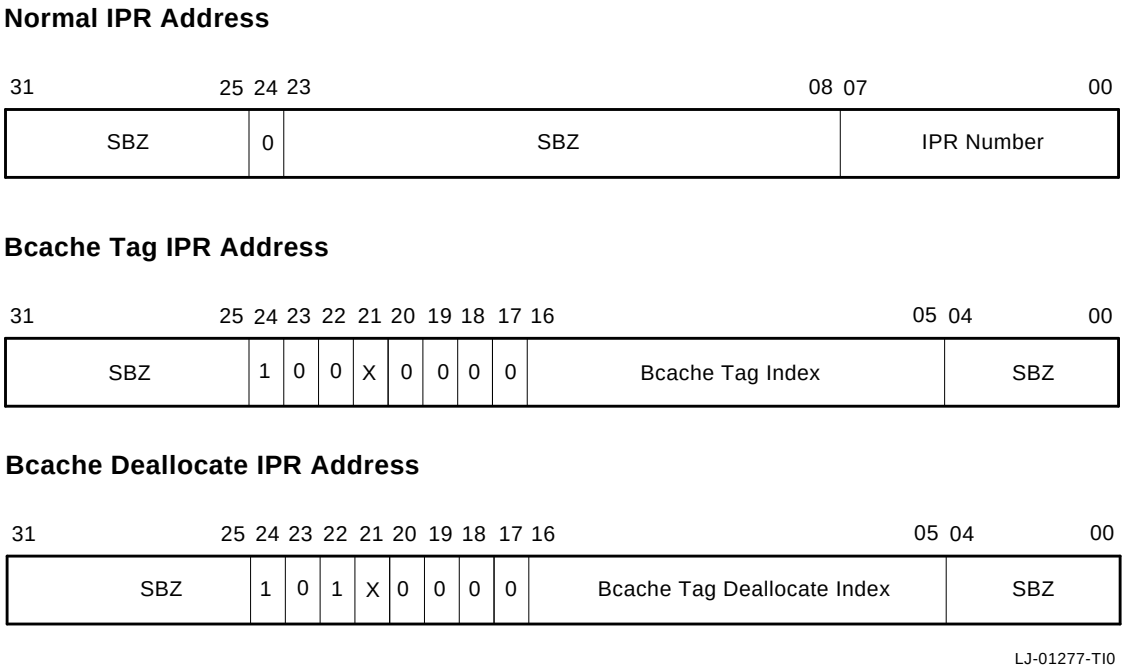
4.4.8 Backup Cache Internal Processor Registers

The Bcache processor registers that are implemented by the NVAX CPU are logically divided into three groups, as follows:

- Normal—Those IPRs that address individual registers in the NVAX CPU chip or system environment.
- Bcache tag IPRs—The read-write block of IPRs that allow direct access to the Bcache tags.
- Bcache deallocate IPRs—The write-only block of IPRs by which a Bcache block may be deallocated.

Each group of IPRs is distinguished by a particular pattern of bits in the IPR address, as shown in Figure 4–17.

Figure 4–17 IPR Address Space Decoding as Seen by Software



The numeric range for each of the three groups is shown in Table 4–13.

Table 4–13 IPR Address Space Decoding - KA680

IPR Group	Mnemonic[1]	IPR Address Range (hex)	Contents
Normal	–	00000000..000000FF	256 individual IPRs
Bcache Tag	BCTAG	01000000..0101FFE0[2]	4K Bcache tag IPRs, each separated by 20(hex) from the previous one
Bcache Deallocate	BCFLUSH	01400000..0141FFE0[2]	4K Bcache tag deallocate IPRs, each separated by 20(hex) from the previous one

[1]The mnemonic is for the first IPR in the block.

[2]Unused fields in the IPR addresses for these groups should be zero. Neither hardware nor microcode detects and faults on an address in which these bits are nonzero, and they are ignored with respect to the tag or data location that is accessed.

Because of the sparse addressing used for IPRs in groups other than the normal group, valid IPR addresses are not separated by one. Rather, valid IPR addresses are separated by 20(hex). For example, the IPR address for Bcache tag 0 is 01000000 (hex), and the IPR address for Bcache tag 1 is 01000020 (hex). In this group, bits <4:0> of the IPR address are ignored, so IPR numbers 01000001 through 0100001F all address Bcache tag 0.

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

The Bcache processor registers implemented by the NVAX CPU are are shown in Table 4–15.

Table 4–15 Bcache/NDAL Processor Registers

Register Name	Mnemonic	Number (Dec)	(Hex)	Type	Impl	Cat
Bcache Control Register	CCTL	160	A0	RW	NVAX	2-5
Reserved for NVAX	–	161	A1	–	NVAX	2-6
Bcache Data ECC	BCDECC	162	A2	W	NVAX	2-5
Bcache Error Tag Status	BCETSTS	163	A3	RW	NVAX	2-5
Bcache Error Tag Index	BCETIDX	164	A4	R	NVAX	2-5
Bcache Error Tag	BCETAG	165	A5	R	NVAX	2-5
Bcache Error Data Status	BCEDSTS	166	A6	RW	NVAX	2-5
Bcache Error Data Index	BCEDIDX	167	A7	R	NVAX	2-5
Bcache Error Data ECC	BCEDECC	168	A8	R	NVAX	2-5
Reserved for NVAX	–	169	A9	–	NVAX	2-6
Reserved for NVAX	–	170	AA	–	NVAX	2-6
Fill Error Address	CEFADR	171	AB	R	NVAX	2-5
Fill Error Status	CEFSTS	172	AC	RW	NVAX	2-5
Reserved for NVAX	–	173	AD	–	NVAX	2-6
NDAL Error Status	NESTS	174	AE	RW	NVAX	2-5
Reserved for NVAX	–	175	AF	–	NVAX	2-6
NDAL Error Output Address	NEOADR	176	B0	R	NVAX	2-5
Reserved for NVAX	–	177	B1	–	NVAX	2-6
NDAL Error Output Command	NEOCMD	178	B2	R	NVAX	2-5
Reserved for NVAX	–	179	B3	–	NVAX	2-6
NDAL Error Data High	NEDATHI	180	B4	R	NVAX	2-5
Reserved for NVAX	–	181	B5	–	NVAX	2-6
NDAL Error Data Low	NEDATLO	182	B6	R	NVAX	2-5

Table 4–15 Bcache/NDAL Processor Registers

Register Name	Mnemonic	Number (Dec)	(Hex)	Type	Impl	Cat
Reserved for NVAX	–	183	B7	–	NVAX	2-6
NDAL Error Input Command	NEICMD	184	B8	R	NVAX	2-5
Reserved for NVAX	–	185	B9	–	NVAX	2-6
Reserved for NVAX	–	186	BA	–	NVAX	2-6
Reserved for NVAX	–	187	BB	–	NVAX	2-6
Reserved for NVAX	–	188	BC	–	NVAX	2-6
Reserved for NVAX	–	189	BD	–	NVAX	2-6
Reserved for NVAX	–	190	BE	–	NVAX	2-6
Reserved for NVAX	–	191	BF	–	NVAX	2-6
Bcache Tag-KA680 (01000000 - 0101FFE0 HEX)	BCTAG	–	–	RW	NVAX	2-5
Bcache Deallocate-KA680 (01400000 - 0141FFE0 HEX)	BCFLUSH	–	–	W	NVAX	2-5

If a write-only NVAX processor register is read, the Cbox returns UNPREDICTABLE data. Reading an unimplemented NVAX processor register returns UNPREDICTABLE data; if an unimplemented register is written, the write is discarded and normal operation continues.

If software attempts to access an IPR, specifying an address that is not within the NVAX block of IPR addresses, the reference will be converted to an I/O space read or write. In this case, the NVAX merges the IPR address with E1000000 hex, effectively adding the base I/O space address of the IPR block to the IPR address. This is done in hardware by forcing bits <31:29> and bit <24> to 1s. (The other upper bits are expected to be received as 0s.)

From this point on, the transaction is treated as an I/O space transaction by the NVAX. It sends the request off-chip to the NDAL. When the data returns, it is not cached by the NVAX. I/O space reads and writes are never cached in the primary cache or the backup cache.

4.4.8.1 Bcache Control IPR (CCTL)

CCTL is a read/write register that contains bits controlling the behavior of the Bcache and related portions of the NVAX CPU. The bits are detailed in Figure 4–18 and Table 4–16.

Figure 4–18 IPR Format of CCTL

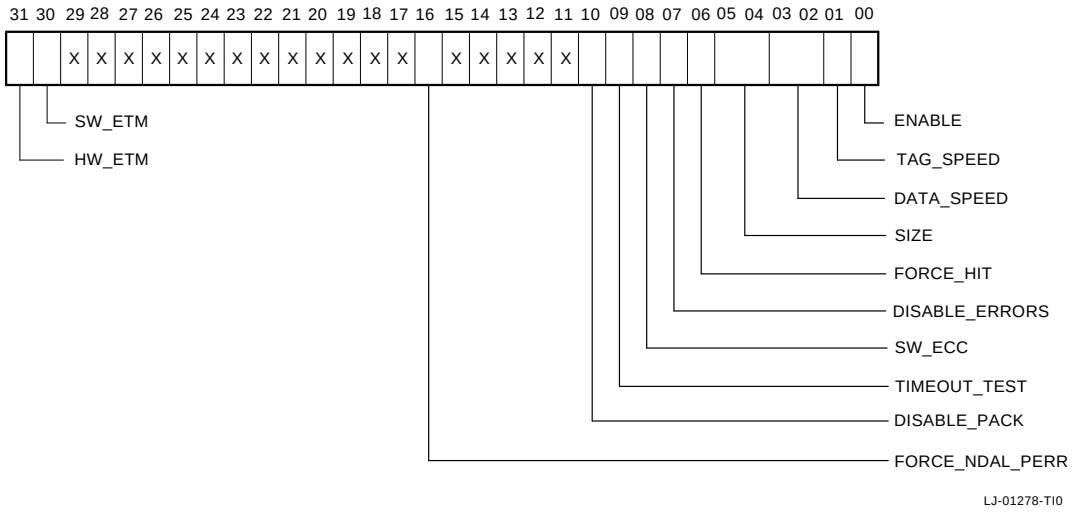


Table 4–16 CCTL

Name	Extent	Type	Description
ENABLE	0	RW,0	Turns the Bcache on and off.
TAG_SPEED	1	RW,0	Controls time NVAX allows to access the tag RAMs.
DATA_SPEED	3:2	RW,0	Controls time NVAX allows to access the data RAMs.
SIZE	5:4	RW,0	Selects between backup cache sizes.
FORCE_HIT	6	RW,0	Forces memory reads and writes to hit in the backup cache.
DISABLE_ERRORS	7	RW,0	Disables all backup cache ECC errors.
SW_ECC	8	RW,0	Enables use of ECC check bits as given by software for the tag and data.
TIMEOUT_TEST	9	RW,0	Puts the NDAL read timeout timers into test mode.
DISABLE_PACK	10	RW,0	Disables the write packer.
FORCE_NDAL_PERR	16	RW,0	Forces a parity error in the command field of the next outgoing NDAL transaction.
SW_ETM	30	RW	Used by software to put the backup cache into ETM.
HW_ETM	31	WC	Used by hardware to put the backup cache into ETM.

ENABLE

When ENABLE=1, the backup cache is enabled for operation. When ENABLE=0, the backup cache is off and all references are treated as misses and are not looked up in the backup cache. When the backup cache is off, FORCE_HIT, SW_ETM, and HW_ETM are ignored. System reset clears this bit so that the Bcache is off when the chip is reset.

TAG_SPEED

The NVAX provides this bit to configure the NVAX to function properly with Bcache tag RAM chips of various speeds. This bit will select the mode of Bcache operation that corresponds to the speed of the RAM chips used on the module.

Note

Improper setting of these bits can prevent the NVAX CPU from functioning properly.

This bit is cleared on system reset and should not be set to a value other than that recommended in Table 4–17. Table 4–17 shows the relationship of the value of TAG_SPEED and the access time of the tag RAMs, given in NVAX cycles. This is the total RAM access time including internal NVAX processing time. Reset clears this bit so that the tag access repetition rate is 3 cycles when the chip is reset. See Appendix J.

Table 4–17 TAG_SPEED

TAG_SPEED	Tag Read (rep rate)	Tag Write (rep rate)	Comments
0	3 Cycles	3 Cycles	–
1	4 Cycles	4 Cycles	May not be used when DATA_ SPEED=00

DATA_SPEED

The NVAX provides this bit to configure the NVAX to function properly with Bcache data RAM chips of various speeds. This bit will select the mode of Bcache operation that corresponds to the speed of the Bcache data RAM chips used on the module.

Note

Improper setting of this bit can prevent the NVAX CPU from functioning properly.

This bit is cleared on system reset and should not be set to a value other than that recommended in Table 4–17. Table 4–18 shows the relationship of the value of DATA_SPEED and the access time of the DATA RAMs, given in NVAX cycles. This is the total RAM access time including internal NVAX processing time. Reset clears this bit so that the Bcache data access repetition rate is 2 cycles when the chip is reset.

Table 4–18 DATA_SPEED

DATA_SPEED<1:0>	Data Read (rep rate)	Data Write (rep rate)	Comments
00	2 Cycles	3 Cycles	May not be used when TAG_SPEED=1
01	3 Cycles	4 Cycles	–
10	4 Cycles	5 Cycles	–
11	Unused	Unused	–

SIZE

These two bits are used to program the size of the Bcache. Backup cache size is programmed by using the SIZE bits, as shown in Table 4–19. These bits are cleared on reset so that when the chip is reset, the correct setting is selected by default.

Note

On the KA680, specifying any value other than that for the 128-kilobyte cache is strictly forbidden, because it will prevent operation.

Table 4–19 SIZE

SIZE<1:0>	Backup Cache Size
00	128 kilobytes

FORCE_HIT

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

The backup cache must be enabled when the cache is used in FORCE_HIT mode.

This mode provides the capability for directly accessing the Bcache data store, and is expected to be used for testing purposes only.

DISABLE_ERRORS

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

SW_ECC

When SW_ECC is clear, the NVAX CPU generates correct ECC check bits for all writes to the tag store and data RAMs. When SW_ECC is set, the NVAX does not generate the check bits when the backup cache is written with data, but uses the check bit values as specified by software in the BCDECC register.

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

Reset clears this bit.

TIMEOUT_TEST

When TIMEOUT_TEST is set, the NVAX uses the internal clock to clock its read timeout counter. When TIMEOUT_TEST is clear, the NVAX uses an internal time base clock its timeout counters. Reset clears this bit. The effect of setting this bit is to cause reads on the NDAL to timeout much sooner than they would if this bit were clear. This is useful primarily for testing purposes. This bit should not be set during normal operation.

DISABLE_PACK

The NVAX normally packs consecutive memory space writes to the same quadword into one write, thereby saving Bcache or NDAL bus bandwidth, depending on whether the referenced hexaword is cached. When DISABLE_ PACK is set, the NVAX does not pack quadword writes together. Instead, the write packing logic inside the NVAX CPU passes every write directly to its destination: either the Bcache or the memory subsystem. When the bit is clear, the NVAX packs writes together to maximize performance. DISABLE_PACK is intended for testing purposes only. Reset clears this bit.

FORCE_NDAL_PERR

When a 1 is written to FORCE_NDAL_PERR, a parity error is caused in the command field of the next outgoing NDAL transaction. Setting this bit causes only one parity error. Another parity error will be produced with the bit is cleared and set again by software.

Reset clears this bit.

SW_ETM

This is a software-writable bit to put the backup cache into error transition mode. When the cache is on and software ascertains that the cache is producing errors, it can set this bit in order to turn off the cache while ensuring cache coherency. Software can then flush owned data through use of the Bcache deallocate IPR, BCFLUSH. In this manner, the unique data can be extracted from the cache before it is turned off completely.

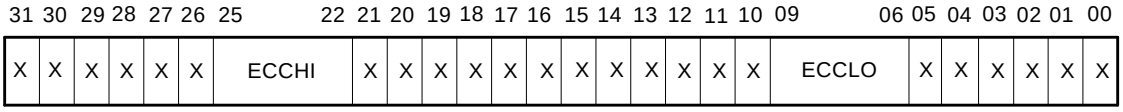
HW_ETM

Hardware sets this bit when an uncorrectable error is detected in the backup cache tag store or data RAMs, unless DISABLE_ERRORS is set. Hardware sets the bit to put the backup cache into error transition mode.

Software clears HW_ETM by writing a 1 to it.

4.4.8.2 Backup Cache Data ECC IPR (BCDECC)

Figure 4–19 Format of the BCDECC



The ECCHI field corresponds to data check bits <7:4>. The ECCLO field corresponds to data check bits <3:0>.

This register is written by software. It is a write-only register.

Software writes BCDECC using an MTPR instruction. The value in the register is then used to explicitly write ECC into the data RAMs during any write of the data RAMs, but only if SW_ECC is set in the control register. If SW_ECC is not set, hardware ignores the value in BCDECC and generates the check bits to be written using the ECC syndrome generator.

One use of BCDECC is to allow software to explicitly write bad ECC into the data RAMs in order to test the Bcache error detection logic.

Reset does not affect this register.

4.4.8.3 Backup Cache Tag Store Error Registers (BCETSTS, BCETIDX, BCETAG)

On some tag store errors, hardware overwrites the corrupted values so that they cannot be diagnosed by reading the tag store directly. For this reason, there are tag store error registers that hold the relevant data so that software can understand the problem.

The tag store error registers are loaded when any tag store error occurs. The status bits in BCETSTS indicate what sort of error happened. Correctable errors are indicated by the CORR bit; the UNCORR and BAD_ADDR errors are both uncorrectable errors.

If no error is yet logged in the registers, the registers are loaded when either a correctable or an uncorrectable error occurs. Once the registers are loaded with information from a correctable error, they are locked against further correctable errors, and are only loaded again if an uncorrectable error happens. At this time, either UNCORR or BAD_ADDR is set. The LOCK bit in BCETSTS is set as well. In this way, information from the first correctable error is held in the registers, and is only overwritten if an uncorrectable error happens later.

The error registers are cleared and unlocked by software. If the error registers hold data from a noncorrectable error and yet another noncorrectable error happens before the error registers are unlocked, the LOST_ERR bit is set. This indicates to software that it does not have sufficient information in the error registers to recover from all uncorrectable errors that have occurred.

4.4.8.3.1 Bcache Error Tag Status (BCETSTS) The BCETSTS register gives the general status of an error in the tag store, indicating the transaction taking place at the time and the type of error. The register is written by hardware and read by software. Hardware does not clear the error bits in this register; this must be done by software using write-one-to-clear to the bottom five bits of the register.

Figure 4–20 IPR Format of BCETSTS

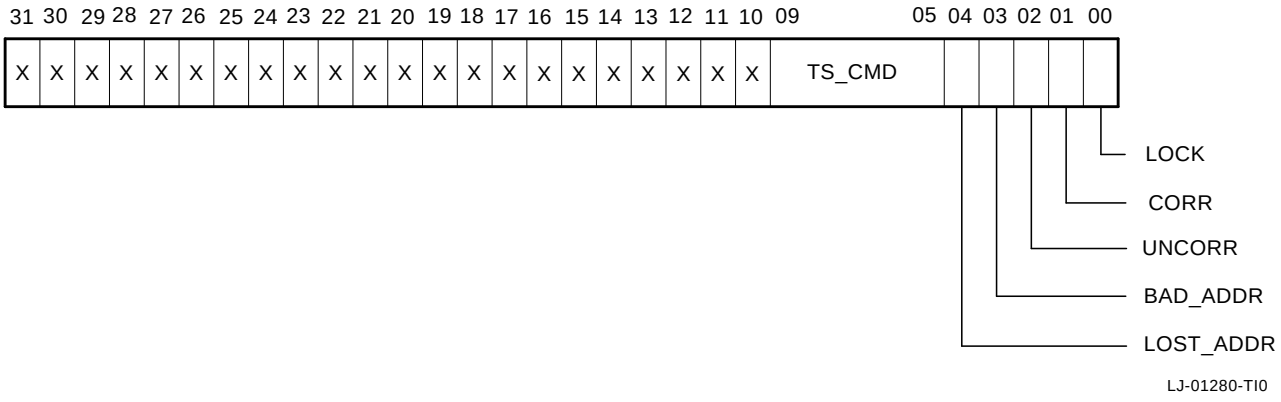


Table 4–20 Bcache Tag Store Status IPR Format

Name	Extent	Type	Description
LOCK	0	WC	Indicates that BCETSTS, BCETIDX, and BCETAG are locked.
CORR	1	WC	Indicates that a correctable ECC error was encountered.
UNCORR	2	WC	Indicates that an uncorrectable ECC error was encountered.
BAD_ADDR	3	WC	Indicates that an addressing error was detected. This is an uncorrectable error.
LOST_ERR	4	WC	Indicates that more than one uncorrectable error occurred, which was not recorded in the error registers.
TS_CMD	9:5	R	Indicates what tag store command was being processed at the time the error occurred.

LOCK

Whenever the tag store error registers are locked due to an uncorrectable error, the LOCK bit is set. At this time, either UNCORR or BAD_ADDR is also set to indicate the type of uncorrectable error. When the LOCK bit is set, the BCETSTS, BCETIDX, and BCETAG registers are all locked. Clearing the lock bit unlocks all three registers. The LOCK bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

CORR

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

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

The CORR bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

UNCORR

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

The UNCORR bit and the BAD_ADDR bit are exclusive: only one of them is set for a given error that sets the LOCK bit. If the other type of error occurs later, the related bit is not set since the register is already locked. In this case, LOST_ ERR is set instead.

The UNCORR bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

BAD_ADDR

BAD_ADDR is set when the tag store ECC decoder detects an error in the address bit, indicating some problem with the address lines going to the tag RAMs. This is an uncorrectable error; thus, when it occurs, the Bcache Tag Store Error registers are loaded and locked.

The UNCORR bit and the BAD_ADDR bit are exclusive: only one of them is set for a given error that sets the LOCK bit. If the other type of error occurs later, the related bit is not set since the register is already locked. In this case, LOST_ ERR is set instead.

The BAD_ADDR bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

LOST_ERR

LOST_ERR indicates that after the first uncorrectable error was recorded in the tag store error registers, an additional uncorrectable error occurred for which state was not saved. LOST_ERR is set by hardware and is cleared by software. It is a write-one-to-clear bit.

TS_CMD

The five bit field, TS_CMD, indicates what the tag store was doing when the error was detected. Its values are listed in Table 4–21.

Table 4–21 Interpretation of TS_CMD

TS_CMD	Name	Tag Store Operation
00111	DREAD	Data-stream tag lookup
00011	IREAD	Instruction-stream tag lookup
00010	OREAD	Ownership-read tag lookup for a write or a READ_ LOCK
01000	WUNLOCK	Ownership-read tag lookup for a WRITE_UNLOCK
01101	R_INVAL	Cache coherency tag lookup as the result of NDAL DREAD or IREAD
01001	O_INVAL	Cache coherency tag lookup as the result of NDAL OREAD or WRITE
01010	IPR_DEALLOC	Tag lookup for an explicit IPR deallocate operation

There are three tag store operations that do not cause any sort of errors: tag store update after a fill, IPR write of the tag store, and IPR read of the tag store. Thus, these commands will not appear in BCETSTS.

4.4.8.3.2 Bcache Error Tag Index (BCETIDX) This register is loaded and locked when a tag store error occurs. If a correctable error is followed by a second error that is not correctable, the register is loaded with information from the second, more serious error. Except for this case, once it is locked, it is not changed until software explicitly unlocks the register. This register is written by hardware and read by software.

Figure 4–21 Backup Tag Store Error Address IPR

BCETIDX contains the complete hexaword address corresponding to a tag store request that resulted in an error. Since the full address is saved, both the cache index and the cache tag of the request are known. Thus, this register shows what index was being accessed when the error occurred as well as showing what the tag of the request was. Software can compare this tag with the actual tag read from the RAMs, which is saved in BCETAG.

4.4.8.3.3 Bcache Error Tag (BCETAG) This register is loaded when a tag store error occurs. It is locked when an uncorrectable error occurs on a tag store access. Once the register is locked, it is not overwritten until it is unlocked by software. BCETAG is written by hardware and read by software. It is a read-only register from the software point of view.

The register holds the data that was read from the tag store and produced the error, as shown in Figure 4–22.

Figure 4–22 IPR Format of BCETAG

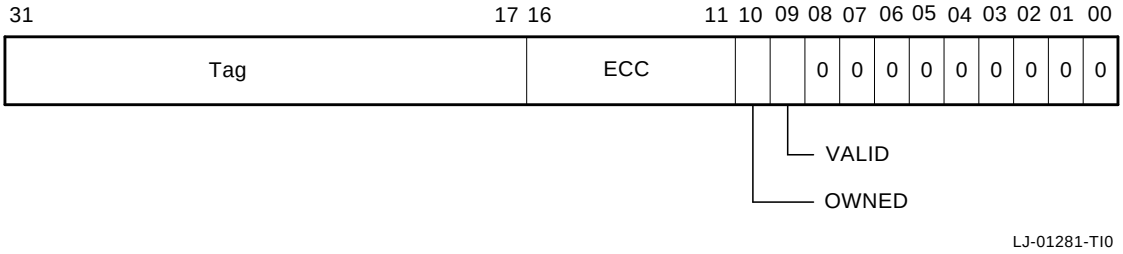


Table 4–22 BCETAG IPR Format

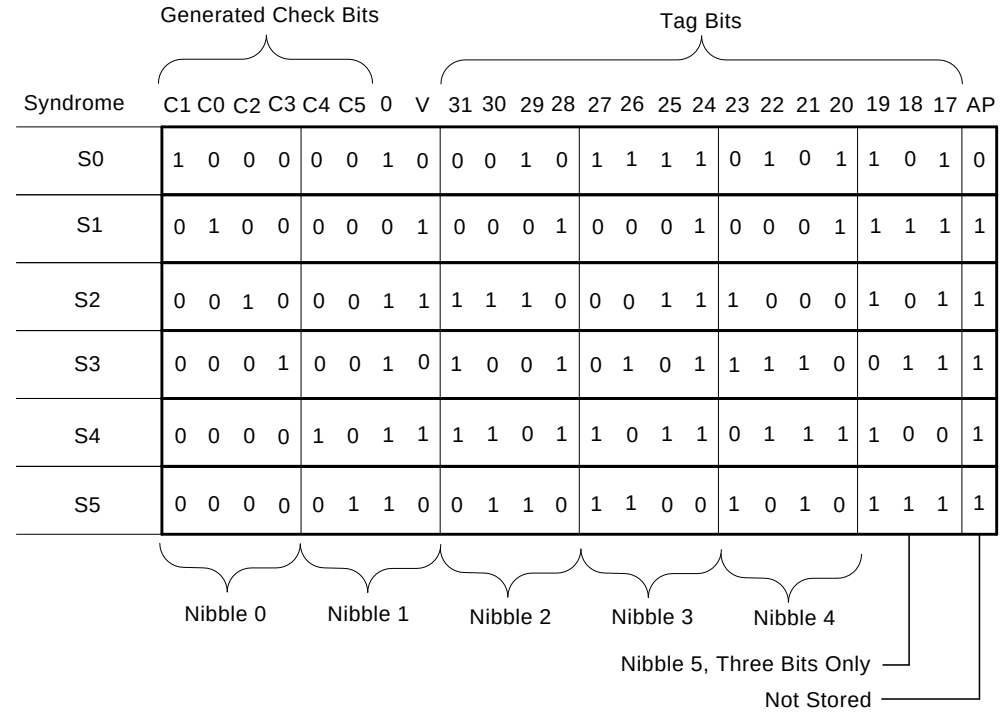
Name	Extent	Type	Description
VALID	9	RO	Valid bit
OWNED	10	RO	Ownership bit
ECC	16:11	RO	ECC check bits
TAG	31:17	RO	Backup cache tag

VALID
VALID is the bit read from the tag RAMs, which indicates whether the block is valid in the Bcache.

OWNED
OWNED is the bit read from the tag RAMs, which indicates whether the Bcache (NVAX) owns the memory hexaword contained in this Bcache block.

ECC
The ECC field contains the check bits as read from the tag RAMs during the tag access that produced the error. The code used for tag ECC is shown in Figure 4–23. The check bit marked with a "1" in each row is generated by a parity tree whose inputs are the Tag, Valid, Owned, and AP (address parity) bits, which are marked with a "1" in that row.

Figure 4–23 Tag Store Error Correcting Code Matrix



Even Parity - C0, C2, C3, C5
Odd Parity - C1, C4
Sn = (Generated Cn) XOR (Stored Cn)

LJ-01282-TI0

In a tag store read operation, a nonzero syndrome indicates an error. If the syndrome generated matches one of the columns in the matrix, the error is correctable and the matching column indicates the bit to be corrected. Any syndrome value that is nonzero and does not match a column in the matrix indicates an uncorrectable error.

Odd parity is used for check bits 1 and 4 to protect against the all-zeros failure mode. Otherwise, all-zeros would be a valid code word. The choice of odd and even parity bits prevents all-ones from being a valid code word as well.

TAG

The TAG field of BCETAG is the cache tag as read from the tag RAMs. It must be interpreted based on the cache size being used, as shown in Table 4–23. When certain address bits are not used as tag bits for the cache size given, their value in BCETAG is 0.

Table 4–23 TAG Interpretation

Cache Size	Tag Bits Used	Unused Tag Bits
128 KB (KA680)	TAG<28:17>	TAG<31:29>

4.4.8.4 Backup Cache Data RAM Error Registers (BCEDSTS, BCEDIDX, BCEDECC)

The data RAM error registers hold data relevant to errors in the backup cache data RAMs so that software can understand the problem.

BCEDSTS holds the general status of the problem. BCEDIDX holds the data RAM index being used when the problem occurred. BCEDECC holds the syndrome bits as calculated on the data, which was read from the RAMs when the problem occurred.

If no error is yet logged in the data RAM error registers, the registers are loaded when either a correctable or an uncorrectable error occurs. Once the registers are loaded with information from a correctable error, they are locked against further correctable errors, and are only loaded again if an uncorrectable error happens. If an uncorrectable error happens, the LOCK bit in BCEDSTS is set and the registers are not overwritten until software clears the error bits. In this way, information from the first correctable error is held in the registers, and is only overwritten if an uncorrectable error happens later.

If the registers are locked, any subsequent noncorrectable error causes the LOST_ERR bit to be set, but does not modify any other information in the registers. LOST_ERR indicates to software that it does not have sufficient information in the error registers to recover from all uncorrectable errors that have occurred.

Of the backup cache data RAM error registers, only BCEDSTS is writable by software. Software clears the error and LOCK bits, which re-enables all the data RAM error registers to record the next error that occurs.

4.4.8.5 Bcache Error Data Status (BCEDSTS)

Figure 4–24 IPR Format of BCEDSTS

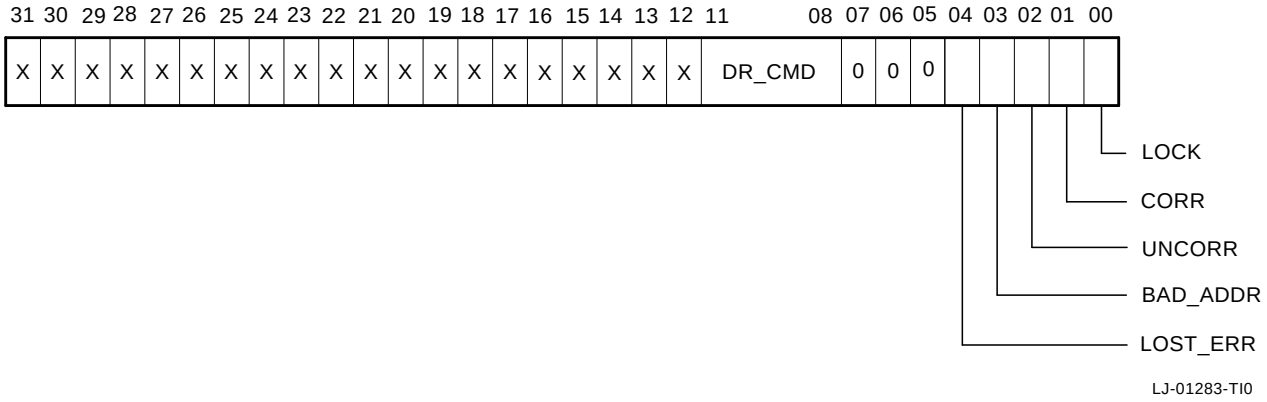


Table 4–24 Bcache Data RAM Status IPR Format

Name	Extent	Type	Description
LOCK	0	WC	Lock bit. Indicates that the BCEDSTS, BCEDIDX, and BCEDECC registers are locked.
CORR	1	WC	Indicates that a correctable ECC error was encountered.
UNCORR	2	WC	Indicates that an uncorrectable ECC error was encountered.
BAD_ADDR	3	WC	Indicates that an addressing error was detected.
LOST_ERR	4	WC	Indicates that a second, uncorrectable error occurred; it was not recorded in the error registers.
DR_CMD	11:8	R	Indicates what command was being processed at the time the error occurred.

The LOCK bit is set when an error that was not correctable has occurred. If the CORR bit is set, the data ram error registers are locked unless an uncorrectable error occurs. On an uncorrectable error, the LOCK bit is set and the registers are permanently locked until unlocked by software.

LOCK

Whenever the data RAM error registers are loaded with an uncorrectable error, the LOCK bit is set. At this time either UNCORR or BAD_ADDR is also set to indicate the type of uncorrectable error. When the LOCK bit is set, the BCEDSTS, BCEDIDX, and BCEDECC registers are all locked. Clearing the lock bit unlocks all three registers. The LOCK bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

CORR

CORR is set when the data ECC decoder detects a correctable error. When this occurs, the Bcache data error registers are loaded and locked against further correctable errors. The CORR bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

UNCORR

UNCORR is set when the data ECC decoder detects an uncorrectable error. When this occurs, the Bcache data error registers are loaded and locked. The UNCORR bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

BAD_ADDR

BAD_ADDR is set when the data ECC decoder detects an error in the address bit, indicating some problem with the address lines going to the data RAMs. This is an uncorrectable error; thus, when it occurs, the Bcache data error registers are loaded and locked. The BAD_ADDR bit is set by hardware and it is cleared by software. It is a write-one-to-clear bit.

LOST_ERR

LOST_ERR indicates that after the first uncorrectable error was recorded in the data error registers, an additional uncorrectable error occurred for which state was not saved. LOST_ERR is set by hardware and is cleared by software. It is a write-one-to-clear bit.

DR_CMD

The DR_CMD field indicates what the data RAMs were doing when the error was detected. Its values are listed in Table 4–25.

Table 4–25 Interpretation of DR_CMD

DR_CMD<11:7>	Name	Data RAM Operation
0111	DREAD	Data lookup for a D-stream read.
0011	IREAD	Data lookup for an I-stream read.
0100	WBACK	Data lookup for a write-back.
0010	RMW	Data lookup for a read-modify-write. Done for normal writes and WRITE_UNLOCKs.

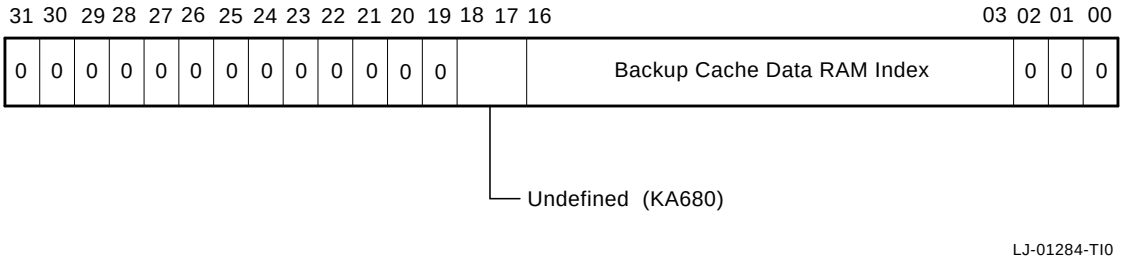
There are two data RAM operations that do not cause any sort of errors: full quadword writes and fills. Thus, these commands will not appear in BCEDSTS.

DR_CMD is only written by hardware. It is read-only for software.

4.4.8.5.1 Bcache Error Data Index (BCEDIDX) This register holds the index of a data RAM transaction; it is loaded when an error is detected on a data RAM access. The index loaded due to a correctable error is not overwritten unless an uncorrectable error occurs afterwards. If an uncorrectable error occurs, BCEDIDX is loaded and locked. BCEDIDX is unlocked by software; the LOCK bit is in the BCEDSTS register.

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

Figure 4–25 BCEDIDX



BCEDIDX must be interpreted based on the cache size being used, as shown in Table 4–26. When certain address bits are not used as index bits for the cache size given, their value in BCEDIDX is undefined.

Table 4–26 BCEDIDX Interpretation

Cache Size	Index Bits Used	Undefined Index Bits
128 KB (KA680)	BCEDIDX<16:3>	BCEDIDX<20:17>

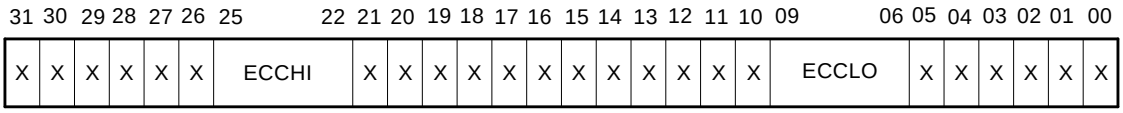
4.4.8.6 Bcache Error Data ECC (BCEDECC)

This register holds the syndrome as calculated on the backup cache data and check bits, and is loaded when an error occurs on a data RAM access. Once loaded, it follows the same lock rules that the other Bcache data error registers follow. It is unlocked by software. The lock bit is in the BCEDSTS register.

When DISABLE_ERRORS is set, BCEDECC is loaded on every quadword read from the cache. This provides a way of testing the ECC logic by reading the results of the syndrome calculation.

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

Figure 4–26 Format of the BCEDECC Register



The ECCHI field corresponds to syndrome bits <7:4>. The ECCLO field corresponds to syndrome bits <3:0>.

The code used for data ECC is shown in Figure 4–27. The check bit (C), which is marked with a "1" in each row, is generated by a parity tree whose inputs are the data bits marked with a "1" in that row.

Figure 4–27 Backup Cache Data Store Error Correcting Code Matrix

```
DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD D          DDD
0123 4567 8911 1111 1111 2222 2222 2233 3333 3333 4444 4444 4455 5555 5555 6CCC CCCC C666 A
             01 2345 6789 0123 4567 8901 2345 6789 0123 4567 8901 2345 6789 0012 3456 7123 P


 1101 0001 0001 0011 1110 1101 1101 1100 0011 0010 0010 1101 1110 1010 1110 0100 0000 0000 1  S0
 1010 0010 0010 0100 1011 1010 1010 1011 0100 0100 0100 1011 1011 1101 1101 1010 0000 0000 1  S1
 0111 1100 1100 1000 0101 0111 0111 0111 1000 1001 1001 0110 0101 0111 0011 0001 0000 0000 1  S2
 0100 1111 1111 0001 0010 0001 0001 0001 0001 0001 0001 0001 1111 0001 1111 1000 1000 0100 0  S3
 1111 0101 1001 0111 0001 0100 0100 1000 0111 1101 1101 0010 0110 1000 0110 0000 0100 0010 1  S4
 1111 1010 0110 1101 0100 1000 1000 0010 1101 1011 1011 0100 1001 0100 1001 0000 0010 0001 1  S5
 1011 1111 0000 0100 1000 1101 0010 0100 1011 1000 0111 0111 1111 1101 0000 1000 0001 0111 0  S6
 0000 1111 0000 1111 0000 1111 0000 0000 0000 1111 0000 1111 0000 1111 1111 0000 0000 1111 0  S7

AP is not stored in the RAMS.
Even parity - C0, C1, C2, C4, C5, C6
Odd parity - C3, C7
Sn = (Generated Cn) XOR (Stored Cn)
```

LJ-01286-TI0

As in tag store ECC, any syndrome value not matching a column in the table indicates an uncorrectable error. Odd parity is used in check bits 3 and 7 to prevent all-ones and all-zeros from being valid code words.

4.4.9 Fill Error Registers (CEFADR, CEFSTS)

Some errors are related to outstanding reads to memory. These errors may be diagnosed using the CEFSTS and CEFADR registers. CEFSTS holds general information about the type of read outstanding; CEFADR holds the address of the outstanding read.

4.4.9.1 Bcache Error Fill Status (CEFSTS)

The CEFSTS register holds information related to a problem on a read that was sent to memory. If a read request to memory times out or is terminated with an error, the CEFSTS register and the CEFADR register are loaded and locked.

The register is read-write. Only the lowest five bits may be written, and then only to clear them after an error. The lowest five bits are write-one-to-clear.

Figure 4–28 IPR Format of CEFSTS

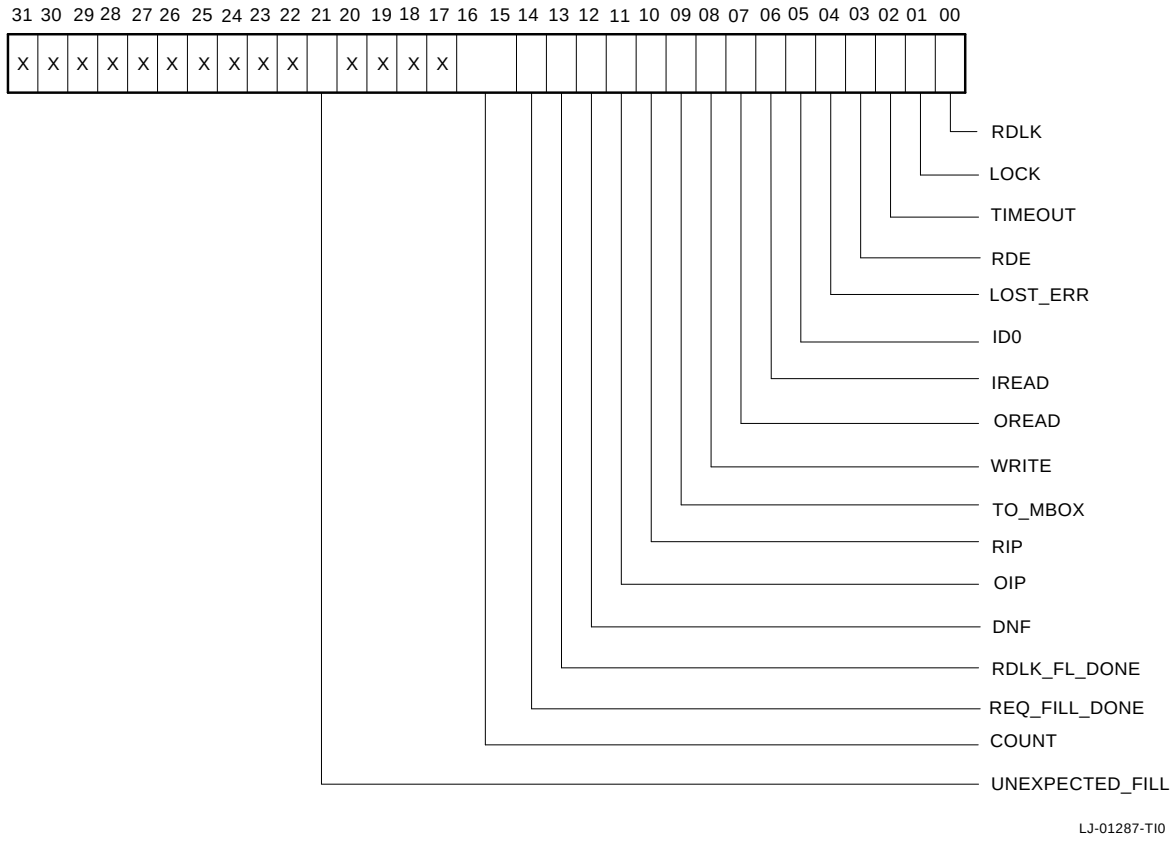


Table 4–27 Fill Error Status IPR Format

Name	Extent	Type	Description
RDLK	0	WC	Indicates that a READ_LOCK was in progress.
LOCK	1	WC	Indicates that an error occurred and the register is locked.
TIMEOUT	2	WC	Fill failed due to transaction timeout.
RDE	3	WC	Fill failed due to read data error.
LOST_ERR	4	WC	Indicates that more than one error related to fills occurred.
ID0	5	RO	NDAL identification bit for the read request.
IREAD	6	RO	This is an I-stream read from the Mbox, which may be aborted.
OREAD	7	RO	This is an outstanding OREAD.
WRITE	8	RO	This read was done for a write.
TO_MBOX	9	RO	Data is to be returned to the Mbox.
RIP	10	RO	READ invalidate pending.
OIP	11	RO	OREAD invalidate pending.
DNF	12	RO	Do not fill - data not to be written into the cache or validated when the fill returns.
RDLK_FL_DONE	13	RO	Indicates that the last fill for a READ_LOCK arrived.
REQ_FILL_DONE	14	RO	Indicates that the requested quadword was successfully returned from the NDAL.
COUNT	16:15	RO	For a memory space transaction, indicates how many of the fill quadwords have been successfully returned. For I/O space, is set to 11 when the transaction starts since only one quadword will be received.
UNEXPECTED_FILL	21	RO	Set to indicate that an unexpected fill was received on the NDAL.

RDLK

RDLK is set to show that a READ_LOCK is in progress. This bit is write-one-to-clear.

The effect of performing a write-one-to-clear to this bit is to clear the VALID bit for an entry that had its RDLK bit set; this has the effect of clearing out the FILL_CAM entry. This is the same action taken when a WRITE_UNLOCK is received.

This bit is normally not read as a one by software, because the NVAX microcode ensures that the READ_LOCK-WRITE_UNLOCK sequence is an indivisible operation. If, however, the first quadword of a READ_LOCK is returned successfully and then the transaction either times out or is terminated in read data error (RDE), CEFSTS is loaded with the RDLK bit set.

LOCK

The LOCK bit is set when a read transaction that has been sent to memory terminates in read data error or in timeout. At the same time, all information corresponding to the read is loaded from the FILL_CAM into the CEFSTS register. When the LOCK bit is set, one of TIMEOUT, RDE, or UNEXPECTED_ FILL is also set to indicate the type of error. Once the LOCK bit is set, none of the information in CEFSTS or CEFADR changes, with the possible exception of LOST_ERR, until the LOCK bit is cleared.

Hardware sets the LOCK bit and software clears it by writing a one to that location.

TIMEOUT

TIMEOUT is set when a read transaction that was sent to the NDAL times out for some reason. When TIMEOUT is set, the LOCK bit is also set.

Hardware sets the TIMEOUT bit and software clears it by writing a one to that location.

RDE

RDE (read data error) is set when a read transaction that was sent to the NDAL terminates in RDE. This could happen because of an uncorrectable main memory error, or in the case of I/O addresses, it could mean some type of I/O error. When the RDE bit is set, the LOCK bit is also set.

Hardware sets the RDE bit and software clears it by writing a one to that location.

LOST_ERR

The LOST_ERR bit is set when CEFSTS is already locked and another RDE, TIMEOUT, or UNEXPECTED_FILL error occurs. This indicates to software that multiple errors have happened and state has not been saved for every error.

Hardware sets the LOST_ERR bit and software clears it by writing a one to that location.

ID0

ID0 corresponds to the NDAL signal, ID_H, which was issued with the read that failed. According to NDAL protocol, the NVAX, as well as other NDAL devices, may have up to two outstanding transactions. Since memory reads are pended, the NVAX could have issued up to two read requests before the error occurred. This bit tells software which of the two NDAL transactions is associated with the error.

IREAD

IREAD indicates that the transaction in error was an IREAD.

OREAD

OREAD indicates that the transaction in error was an OREAD (ownership read); the OREAD may have been done for a write, a READ_LOCK, or a read modify.

WRITE

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

TO_MBOX

TO_MBOX indicates that data returning for the read was to be sent to the Mbox. The Mbox is the part of the NVAX CPU that contains the virtual/physical address translation logic as well as the Pcache.

RIP

RIP (read invalidate pending) is set when the NVAX observes a DMA read transaction on the NDAL to a memory location for which the NVAX is currently acquiring ownership (that is, the OREAD transaction has already been sent to the memory subsystem). This triggers a write-back of the block when the OREAD fill data arrives; a valid copy of the data is kept in the cache.

OIP

OIP (OREAD invalidate pending) is set when a cache coherency transaction due to an OREAD or a WRITE on the NDAL is requested for a block that has OREAD fills outstanding at the time. This triggers a write-back and invalidate of the block when the fill data arrives.

DNF

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

RDLK_FL_DONE

This bit is set in the fill cam when a READ_LOCK hits in the Bcache or the last fill arrives from the BIU for a READ_LOCK. Once this is set, the corresponding WRITE_UNLOCK is allowed to proceed. This overrides the FILL_CAM block conflict on the WRITE_UNLOCK, which is inevitable since the READ_LOCK is held in the FILL_CAM until the WRITE_UNLOCK is done.

REQ_FILL_DONE

This bit is set when the requested quadword of data was successfully received from the NDAL. This information is provided to facilitate error handling.

COUNT

These two bits indicate how many of the expected four quadwords have been returned successfully from memory for this read. If they are 00(BIN), no quadwords have returned, if they are 01(BIN), one quadword has returned, and so forth. If the entry was for a quadword read, the count bits are set to 11(BIN) when the reference is sent out.

UNEXPECTED_FILL

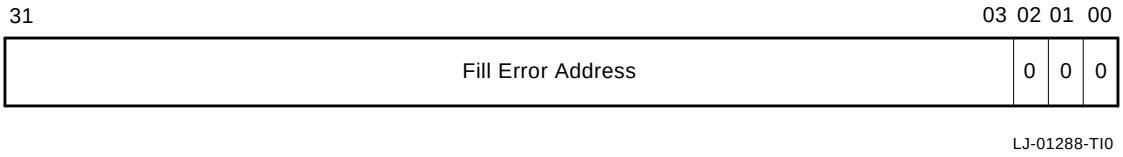
This bit is set to indicate that an RDE or RDR cycle was received on the NDAL with an ID for which the FILL_CAM entry was not valid. When UNEXPECTED_ FILL is set, CEFSTS and CEFADR are loaded and locked.

4.4.9.2 Fill Error Address (CEFADR)

The CEFADR register holds the address of a fill that ended in an error condition. It is loaded when an error is detected on a fill. It is a read-only register.

CEFADR is locked when CEFSTS is locked.

Figure 4–29 IPR Format of CEFADR



4.4.10 NDAL Error Registers (NESTS, NEOADR, NEOCMD, NEDATHI, NEDATLO, NEICMD)

The NDAL error registers hold information related to NDAL errors. NESTS, NDAL error status, holds error bits relating to any problems encountered.

NEOADR, NDAL error output address, holds the address corresponding to the cycle that was in error. NEOCMD, NDAL error output command, holds the command bits corresponding to the cycle in error.

NEDATHI, NDAL error data high longword, and NEDATLO, NDAL error data low longword, hold the data from an NDAL cycle where NVAX detected a parity error on the bus. NEICMD, NDAL error input command, holds the command bits corresponding to a cycle with a parity error.

4.4.10.1 NDAL Error Status IPR (NESTS)

The NESTS register holds information about any errors that happened on the NDAL. All six bits in this register are write-one-to-clear. Reset does not affect this register. Powerup does not initialize the register.

Figure 4–30 IPR Format of NESTS

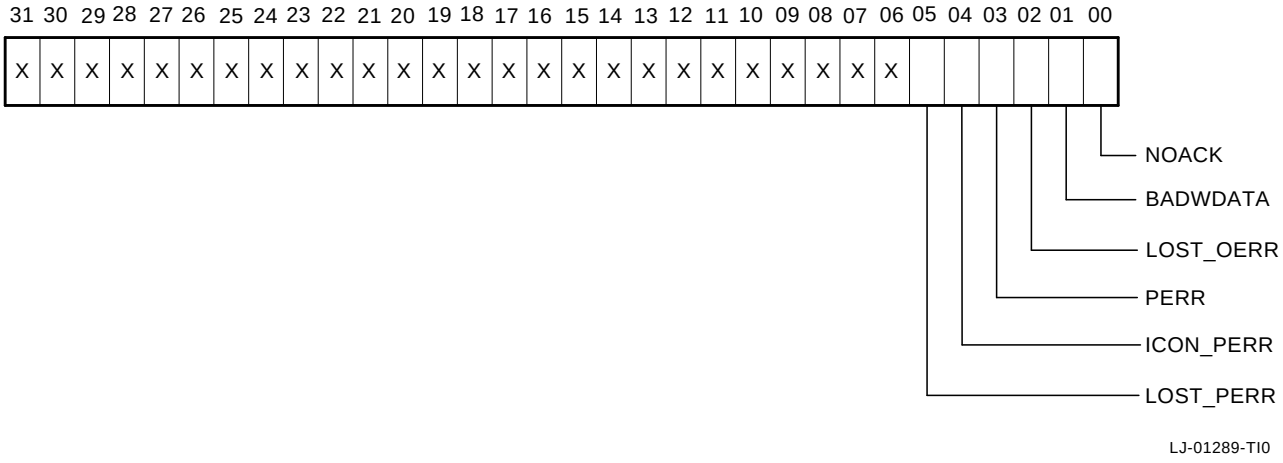


Table 4–28 NESTS IPR Format

Name	Extent	Type	Description
NOACK	0	WC	Indicates that an outgoing NVAX NDAL cycle was not acknowledged by any of the other NDAL devices. This bit locks NEOADR and NEOCMD.
BADWDATA	1	WC	Indicates that an outgoing NDAL data cycle was accompanied by the BADWDATA command. This bit locks NEOADR and NEOCMD.
LOST_OERR	2	WC	Indicates that multiple outgoing errors, either NOACK or BADWDATA, were detected.
PERR	3	WC	Indicates that a parity error was detected on the NDAL. This bit locks NEDATHI, NEDATLO, AND NEICMD.
INCON_PERR	4	WC	Inconsistent parity error. This means that although NVAX detected a parity error, some other device apparently did not and acknowledged the NDAL transaction.
LOST_PERR	5	WC	Indicates that multiple NDAL parity errors were detected.

NOACK

NOACK is set when NVAX detects that the NDAL signal "ACK_L" was not asserted by any receiving device on the NDAL for an outgoing NVAX cycle. When NOACK is set, NEOADR and NEOCMD are locked so that software can read them to see what transaction was being attempted when the error occurred.

NOACK is set on any outgoing NVAX cycle that is not acknowledged, whether it was an address cycle or a data cycle. The information that is locked in NEOADR and NEOCMD corresponds to the address cycle of the transaction. For example, if an outgoing write data cycle is not acknowledged, the address cycle for that write operation is saved in NEOADR and NEOCMD.

NOACK is not set if there was a previous BADWDATA. If a BADWDATA cycle is NOACK'd, both BADWDATA and NOACK are set.

NOACK is cleared by write-one-to-clear.

BADWDATA

BADWDATA is set when the BIU receives data for a write-back from the cache that had an uncorrectable ECC error, and thus is being issued on the NDAL with the BADWDATA command. When BADWDATA is set, NEOADR and NEOCMD are locked so that software can read them to retrieve the information about the failure.

The address for the write operation is captured in NEOADR, and the command information for the cycle is captured in NEOCMD.

NOACK is not set if there was a previous BADWDATA. If a BADWDATA cycle is NOACK'd, both BADWDATA and NOACK are set.

LOST_OERR

LOST_OERR is set when NOACK or BADWDATA is already set and another one of those errors occurred. It notifies software that state was saved only for the first outgoing error.

LOST_OERR is cleared by a write-one-to-clear.

PERR

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

Since NVAX calculates parity on every cycle, PERR will be set on both its own transfers and the transfers of other devices that fail the parity check.

PERR is cleared by a write-one-to-clear.

INCON_PERR

INCON_PERR (inconsistent parity error) is set when an NDAL parity error is detected on a cycle, which is also acknowledged with the NDAL signal "ACK_L." This means that NVAX detected a parity error but some other device acknowledged the transfer.

INCON_PERR is only set in conjunction with PERR. It is not set unless PERR is set. If one NDAL parity error has already occurred, setting PERR, but INCON_ PERR was not set for that cycle, a subsequent cycle with an inconsistent parity error will not cause INCON_PERR to be set.

INCON_PERR is cleared by a write-one-to-clear.

LOST_PERR

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

LOST_PERR is cleared by a write-one-to-clear.

4.4.10.2 NDAL Error Output Address IPR (NEOADR)

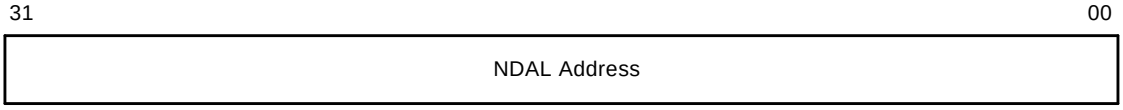
The NEOADR register is loaded for every address cycle that the NVAX drives onto the NDAL unless it is locked. It is loaded during the cycle when the corresponding ACK_L should be asserted on the NDAL. It is locked when the NOACK bit in the NESTS register is set.

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

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

NEOADR is read-only to software.

Figure 4–31 IPR Format of NEOADR



4.4.10.3 NDAL Error Output Command (NEOCMD)

The NEOCMD register is loaded and locked exactly as NEOADR is loaded and locked. The format of NEOCMD is similar to that of the high longword of the NDAL during an address cycle. The high quadword byte enable positions are NOT included, since NVAX only uses quadword byte-enabled transactions. The NDAL ID and command are added in the lower seven bits of the longword.

Figure 4–32 IPR Format of NEOCMD

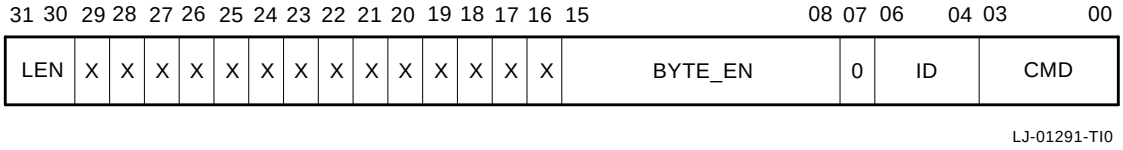


Table 4–29 NEOCMD IPR Format

Name	Extent	Type	Description
CMD	3:0	RO	NDAL command as driven by NVAX during the transaction.
ID	6:4	RO	Commander ID as driven by NVAX during the transaction.
BYTE_EN	15:8	RO	Byte enable as driven by NVAX during the transaction.
LEN	31:30	RO	Length of the NDAL transaction.

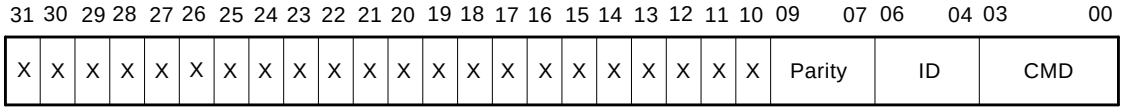
4.4.10.4 NDAL Error Input Command (NEICMD)

NEICMD, NEDATHI, and NEDATLO are loaded at the same time and they are locked at the same time. They are all loaded when a parity error occurs; at this time the PERR bit is set in NESTS, which locks the three registers. If a second NDAL parity error happens, the registers are not loaded again. They are not loaded again until after they are unlocked when software clears PERR.

NEICMD contains the NDAL signals CMD_H<3:0>, ID_H<2:0>, and PARITY_H<2:0> from the failed transfer.

NEICMD is a read-only register.

Figure 4–33 IPR Format of NEICMD



LJ-01292-TI0

PARITY
The PARITY field corresponds to the NDAL lines PARITY_H<2:0>.

ID
The ID field corresponds to the NDAL lines ID_H<2:0>.

CMD
The CMD field corresponds to the NDAL lines CMD_H<3:0>.

4.4.10.5 NDAL Error Data High and NDAL Error Data Low (NEDATHI and NEDATLO)

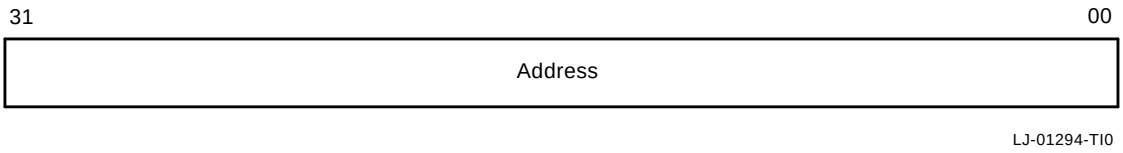
NEDATHI and NEDATLO behave similarly to NEICMD. They capture NDAL_H<63:0> during a cycle with a parity error. NEDATHI contains the high longword of data from the NDAL (NDAL_H<63:32>); NEDATLO contains the low longword of data from the NDAL (NDAL_H<31:0>).

The format of NEDATHI and NEDATLO must be interpreted based on the CMD found in NEICMD. If the CMD field shows that the cycle was a data cycle, the registers contain two longwords of data. If the CMD field shows that the cycle was an address cycle, the registers are in the format of an NDAL address cycle, as shown in Figure 4–34 and Figure 4–35.

Figure 4–34 NEDATHI, Address Cycle Format

LJ-01293-TI0

Figure 4–35 NEDATLO, Address Cycle Format



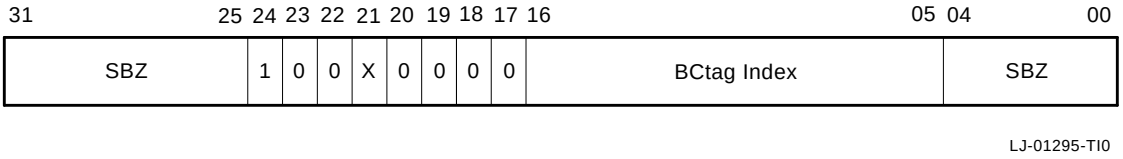
4.4.11 Backup Cache Tag Store Access Through IPR Reads and Writes (BCTAG)

Direct access to the backup cache tag store is provided to aid in error recovery and diagnosis and to assist testing. These accesses work whether the cache is on or off, in ETM or in force hit mode.

If there is a valid FILL_CAM entry for the same cache block that is being accessed through an IPR read or write, the IPR read or write is stalled until the fills return and the FILL_CAM entry is no longer valid.

When the backup cache tag store is being accessed through IPR reads and writes, address bits <24:22> = 100 (BINARY). Address bits <18:5> (KA680) or <18:5> (KA680) are used as the index into the tag store RAMs; these indicate which backup cache location is to be written or read.

Figure 4–36 Backup Cache Tag Store IPR Addressing Format



The format for reading and writing the backup cache tag store as an IPR is described in Figure 4–37 and Table 4–30.

Figure 4–37 IPR Format of the Backup Cache Tag Store

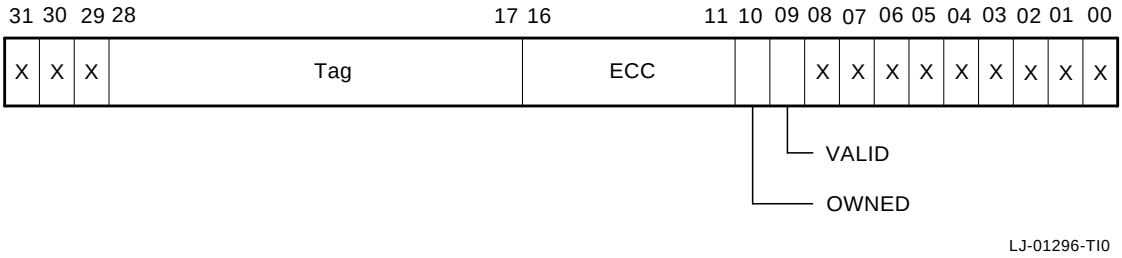


Table 4–30 Bcache Tag IPR Format

Name	Extent	Type	Description
VALID	9	RW	Valid Bit
OWNED	10	RW	Ownership Bit
ECC	16:11	RW[1]	ECC Check Bits
TAG	28:17	RW	Tag Data

[1]The ECC bits are written from the value given in the MTPR instruction only if the SW_ECC bit of the CCTL IPR is set. Otherwise, the Cbox generates and writes correct ECC for the tag, owned and valid values.

Table 4–31 Tag and Index Interpretation for BCTAG IPR

Cache Size	Tag Bits Used	Index Bits Used
128 KB (KA680)	TAG<28:17>	Index<16:5>

The tag store must be initialized to a known state when the chip is powered up. This is done through the MTPR instruction to BCTAG.

When the tag store is read, the ECC check bits are read out directly from the tag store in the format shown. ECC is not checked on IPR accesses to the tag store; no errors can occur during these accesses.

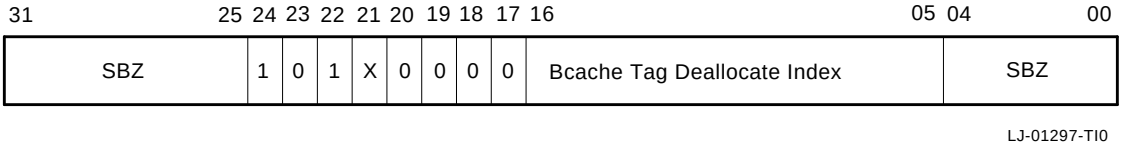
Some care must be taken if IPR reads of the tag store are done while other transactions are in progress. The tag information read out may not be what the programmer expects if cache misses or cache coherency transactions are in progress on the block being read. For example, if a cache miss is in progress, the new tag will be in the tag store but the valid and owned bits will be clear.

4.4.12 Backup Cache Deallocates Through IPR Access (BCFLUSH)

The backup cache deallocate IPR is a write-only register that software uses to explicitly request the deallocation of a cache block. For example, this register may be used when hardware has put the cache into ETM and software wants to request write-back of the owned blocks to memory.

If there is a FILL_CAM entry for the same cache block that is being flushed, the flush is stalled until the fills return and the FILL_CAM entry is no longer valid.

Figure 4–38 Backup Cache Deallocate IPR Addressing Format



When BCFLUSH is written, the NVAX accesses the Bcache tag store. If the block is invalid, no further action is taken. If the block is valid but not owned, the NVAX invalidates the entry in the Bcache tag store, as well as the corresponding Pcache entry if it exists. If the block is valid and owned, the NVAX invalidates

the Pcache entry if it exists, performs a write-back of the Bcache data, and invalidates the Bcache entry in the tag store.

This behavior takes place whether the cache is on, off, in ETM, or in FORCE_HIT mode. In FORCE_HIT mode, BCFLUSH does a real lookup of the tag store and does not force the access to hit. Software must take care not to force deallocates when cache state is not consistent with the state of memory. For example, when the cache is off, valid and owned bits may be set for blocks that are no longer up-to-date with respect to memory.

When a deallocate is done, the VALID and OWNED bits will be cleared as necessary, and the value of the stored TAG is modified. Its value is UNPREDICTABLE. Correct ECC is stored on the tag store entry.

A BCFLUSH operation never changes the data stored in the data RAMs.

Errors are detected and reported during BCFLUSH operations.

The index given is interpreted as in Table 4–31, based on the size of the cache.

BCFLUSH may be used when the Bcache is on, because the Pcache is kept a subset of the Bcache during these operations. However, new blocks may be allocated due to memory reads and writes as the cache is being flushed.

4.4.13 Bcache Abnormal Conditions

This section describes the various modes of Bcache behavior as well as Cbox response when it detects an error.

The Bcache has four operating states that are controlled by the following bits in the CCTL register: ENABLE, FORCE_HIT, SW_ETM, and HW_ETM. The four states are ON, OFF, ETM, and FORCE_HIT. The four states are determined and prioritized as follows:

1. OFF. If the ENABLE bit is cleared in CCTL, the Bcache is OFF and those conditions take precedence.
2. FORCE_HIT. If the ENABLE bit is set and FORCE_HIT is set, the Bcache is in FORCE_HIT mode and those conditions take precedence.
3. ETM. If the ENABLE bit is set, FORCE_HIT is cleared, and either SW_ETM or HW_ETM is set. The cache is in ETM mode and those conditions take precedence.
4. ON. If the ENABLE bit is set and FORCE_HIT, SW_ETM, and HW_ETM are cleared, the cache is ON.

The ON state is the normal operating condition of the cache. OFF, FORCE_HIT, and ETM modes are described in the following sections.

4.4.13.1 NVAX Behavior When the Backup Cache is OFF

The backup cache may be off for two reasons: the chip has just powered up, or software has disabled the cache by clearing the ENABLE bit in the Bcache control register.

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

When the backup cache is off, all cache lookups due to DMA cycles on the NDAL are forwarded as invalidates to the Pcache, since the data may be valid in the Pcache. All reads that miss in the Pcache go directly to the NDAL. Cache fills returning from the memory subsystem are sent directly to the Pcache without incurring the overhead of Bcache access. All writes go directly to the NDAL.

When the cache is off, VAX interlocked instructions that generate atomic read/write pairs become hexaword ownership read/quadword disown write on the NDAL.

All writes issued from NVAX when it is operating without a backup cache are of quadword length. Memory reads are of hexaword length since the Pcache block size is a hexaword. Even if the Pcache is off, a hexaword of data is returned to the Mbox.

A VAX instruction that generates read references with modify intent normally generates an OREAD on the NDAL if it misses in the Bcache. However, when the Bcache is off, a normal DREAD is used on the NDAL.

4.4.13.2 NVAX Behavior When the Backup Cache is in FORCE_HIT Mode

FORCE_HIT mode is intended to be used for testing purposes only. It is used when the cache is enabled.

When FORCE_HIT is set, all memory space reads and writes to the Bcache, both I-stream and D-stream, are forced to hit. Tag store state is not changed at all; the data RAMs are accessed as if the tag store access produced an owned-valid hit. DMA I/O references on the NDAL are treated as they are when the Bcache is off. They are not looked up in the backup cache; they are all forwarded to the Pcache as invalidates, and Bcache state is not changed as the result of the DMA references.

When the Bcache is in FORCE_HIT mode, deallocates are not done. Even if the tag matches and the VALID and OWNED bits are set, the block is not written back. The implication of this is that if FORCE_HIT mode is being used, the Bcache must be flushed of all owned blocks beforehand.

Tag store and data RAM ECC errors are detected in FORCE_HIT mode if DISABLE_ERRORS in the CCTL register is not set, resulting in the usual error handling.

As an example of the use of FORCE_HIT mode, suppose the ECC logic for the data RAMs is to be tested. Put the cache in FORCE_HIT mode. Set SW_ECC in the Bcache control register. Write the desired ECC into BCDECC. Do a write to a memory location that maps to the desired Bcache block, and the location will be written using ECC from BCDECC rather than from NVAX-generated ECC. Suppose the ECC written is such that when the data is read, an ECC error will be flagged.

Now perform a read of the same memory while FORCE_HIT is still set. The read will result in a Bcache data ECC error, showing that the logic is working correctly. The data RAM error registers may be read and will correspond to the induced error.

4.4.13.3 NVAX Behavior When the Backup Cache is in Error Transition Mode

When the NVAX detects certain errors, it puts itself into error transition mode (ETM).

The goals of the Bcache design during ETM are the following:

1. Preserve the state of the Bcache as much as possible for diagnostic software.
2. Honor references that hit owned blocks in the backup cache since this is the only source of data in the system.
3. Respond to NDAL DMA references normally (that is, write-back owned blocks that are referenced by DMA devices), and perform invalidates on DMA-referenced cached unowned blocks.

Once the NVAX enters error transition mode, it remains in ETM until software explicitly disables or enables the Bcache. To ensure Bcache coherency with main memory, the Bcache must be completely flushed of valid blocks before it is re-enabled because some data can become stale while the cache is in ETM.

Table 4–32 describes how the backup cache behaves while it is in ETM.

Table 4–32 Backup Cache Behavior During ETM

Cache Transaction	Cache Response — Miss	Valid Hit	Owned Hit
CPU IREAD,DREAD	Read from memory	Read from memory	Read from cache
CPU READ_LOCK	Read from memory	Read from memory	Force block write-back, read from memory[1]
CPU Write	Write to memory	Write to memory	Force block write-back, write to memory[1]
CPU WRITE_ UNLOCK	Write to memory	Write to memory	Write to cache[1]
Fill (from read started before ETM)	—-Normal cache behavior—-		
Fill (from read started during ETM)	—-Do not update backup cache; return data to Mbox—-		
NDAL DMA reference	—-Normal cache behavior—-		

[1]Done to preserve write ordering

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

The cache supplies data for IREADs, DREADs, and dread modifies that hit valid-owned; this is normal cache behavior.

If a write hits a valid-owned block in the cache, the block is written back to memory and the write is also sent to memory.

If a READ_LOCK hits valid-owned in the cache, a write-back of the block is forced and the READ_LOCK is sent to memory (as an OREAD on the NDAL).

Data returning from the memory subsystem as the result of any type of read originated before the Bcache entered ETM are processed in the usual fashion. If the returning cache fill is a result of a write miss, the write data is merged, as usual, as the requested fill returns. Fills caused by any type of read originated during ETM are not written into the Bcache or validated in the tag store.

During ETM, the state of the cache is modified as little as possible. Table 4–33 shows how each transaction modifies the state of the cache.

Table 4–33 Backup Cache State Changes During ETM

Cache Transaction	Cache State: Miss	Modified: Valid Hit	Modified: Owned Hit
CPU IREAD,DREAD	None	None	None.
CPU READ_LOCK	None	None	Clear VALID and OWNED; change TS_ECC accordingly.
CPU Write	None	None	Clear VALID and OWNED; change TS_ECC accordingly.
CPU WRITE_ UNLOCK	None	None	Write new data; change DR_ECC accordingly.
Fill (from read started before ETM)	Write new TS_TAG, TS_VALID, TS_OWNED, TS_ECC, DR_DATA, DR_ECC		
Fill (from read started during ETM)	None		
NDAL DMA transaction	Clear VALID and OWNED; change TS_ECC accordingly		

4.4.14 How to Turn the Bcache Off

Because the Bcache is a write-back cache, care must be taken to maintain cache coherency when turning it off.

If the cache is running normally and software wishes to turn it off, it must do the following:

1. Write to CCTL register to set SW_ETM. In this mode, the Bcache will not allocate any new blocks and will send all DMA-caused Bcache lookups to the Pcache as invalidates.
2. Use the BCFLUSH register to flush all owned blocks out of the cache.
3. Turn off the Bcache by writing the CCTL register to clear ENABLE and SW_ETM simultaneously. If an error was encountered during the deallocate process, HW_ETM may be set. If so, it should be cleared as well.

If the Bcache encounters an uncorrectable ECC error, the NVAX sets HW_ETM in the CCTL register. If software wishes to turn off the cache, it must do the following:

1. Use the BCFLUSH register to flush all owned blocks out of the cache.
2. Write CCTL to clear ENABLE and clear HW_ETM simultaneously. This turns off the Bcache.

If Bcache errors are occurring only in part of the cache, software may be able to avoid the portion of the cache that is in error by disabling it through the use of the SIZE field in CCTL. If part of the cache is failing, a smaller cache size may be selected so that only part of the cache RAMs is being used. The cache must be flushed before changing the cache size so that the tags are correct.

This works only if the smallest cache size is not being used, and if the failing areas of cache do not fall within the range of the smaller cache size selected.

4.4.15 How to Turn the Bcache On

When NVAX powers up, garbage data is stored in the Bcache tags and data. This would result in ECC errors if the cache were turned on immediately.

Through IPR writes, every Bcache tag store entry must be written with cleared OWNED and VALID bits. The value written to the tag is irrelevant, as long as correct ECC is written to the tag store.

Once the tag store has been initialized, the cache may be enabled by setting ENABLE in the CCTL register.

It is not strictly necessary to initialize the Bcache data RAMs with correct ECC on powerup. ECC errors in the data RAMs are ignored if the corresponding tag store entry is invalid. Since data RAM ECC errors are not detected on fills, the cache data is self-initializing.

FORCE_HIT mode may be used to initialize the Bcache data RAMs with correct ECC. If full quadword writes are used, no data RAM errors will be detected during this process, since the RAMs are written without being read first. If partial quadword writes are used, errors will be detected because of the read-modify-write that is necessary. If the programmer sets the DISABLE_ERRORS bit in the CCTL register, the NVAX will ignore these errors.

If the Bcache is in ETM, it may be incoherent with respect to memory because of how it treats writes that hit valid but not owned in the cache (Table 4–32). In addition, the Pcache, if enabled, is no longer a subset of the backup cache. The procedure for turning on the Pcache and the Bcache as described in this section must be followed.

If the Bcache is operating normally and is turned off for some reason, the programmer must ensure that when it is re-enabled, all the OWNED and VALID bits are cleared.

4.4.16 Backup Cache Errors

In general, the NVAX logs as much state as possible concerning errors and notifies the Ebox and/or Mbox that an error has occurred. For every error, the NVAX does at least one of the following to notify software of the error: hard error interrupt, soft error interrupt, or machine check exception. The backup cache goes into error transition mode when it detects any uncorrectable error from the cache RAMs.

Table 4–34 Backup Cache ECC Errors and NVAX CPU Error Responses

General Problem	Specific Situation and Action Taken by NVAX CPU	
Correctable ECC error in the data RAMs	Read hit for write-back or read hit for deallocate IPR	Soft error interrupt. The data for the write-back is corrected and the write-back continues normally.
	Read hit for Pcache miss	Soft error interrupt.
	Read for write hit	Soft error interrupt. The corrected data is merged with the write data and written into the RAMs.
	Miss	No error is reported.
Uncorrectable ECC error in the data RAMs (includes addressing errors)	Read for write-back or deallocate IPR	Soft error interrupt, puts backup cache into ETM. The data cycle command for the NDAL is changed to BADWDATA and the write-back continues normally.
	VALID-OWNED or VALID-UNOWNED read for Pcache miss	Soft error interrupt, puts backup cache into ETM.
	VALID-OWNED DREAD_LOCK, first quadword fails	Soft error interrupt, puts backup cache into ETM.
	VALID-OWNED DREAD_LOCK for Pcache miss, quadword other than the first one fails	Soft error interrupt, puts backup cache into ETM.
	Read for write, valid-owned hit, or write unlock	Hard error interrupt, puts backup cache into ETM. When the error is detected, write data has already been merged with the corrupted data. The NVAX inverts three of the ECC check bits (bits 3,6,7), which gives a high probability that when the data is read again, an uncorrectable error will be detected.
	Miss	No error is reported.
Correctable ECC error in the tag store	Any read or write except WUNLOCK; hit or miss	NVAX takes a soft error interrupt, assumes the transaction missed, and sends a READ or an OREAD to memory. If the location was owned, making a deallocate necessary, the outgoing address is corrected for the write-back. Note that if the transaction actually hit-owned, the READ or OREAD is sent to the NDAL followed by a write-back of the same block. The errored location is corrected by hardware when the tag and valid bit are written for the fill.

(continued on next page)

Table 4–34 (Cont.) Backup Cache ECC Errors and NVAX CPU Error Responses

General Problem	Specific Situation	and Action Taken by NVAX CPU
	DMA cache coherence transaction miss	Soft error interrupt. Hardware does not correct the bad location; it may be done by software.
	DMA cache coherence transaction hit	Soft error interrupt. Writes the corrected tag, valid, and owned bits back into the tag store when invalidating the entry. Uses corrected address for the write-back if necessary.
Uncorrectable ECC error in the tag store (includes addressing errors)	Read for Pcache miss	Soft error interrupt. Backup cache put into ETM. The read is sent to memory; if the backup cache actually owned the block, the read will time out.
	Write	Soft error interrupt. Backup cache put into ETM. The OREAD for the write is sent to memory. If the cache actually owned the block, the read will time out and the write will then be sent to memory. The write will then time out as well unless error handling software cleans up the problem. If the cache did not own the block, the OREAD will complete, the write will be merged with it, and the merged data will be written to the cache.
	WRITE_UNLOCK	No tag store lookup is done, so this case does not occur.
	DMA cache coherence transaction	Soft error interrupt. Backup cache put into ETM. Transaction is treated as a miss with regard to the backup cache; the invalidate is forwarded to the Pcache if the cache coherence transaction were due to an OREAD or a WRITE.

4.4.16.1 Backup Cache Errors Incurred While in Error Transition Mode

Table 4–35 describes error handling when the backup cache is already in ETM.

Note

The table below only describes ETM error cases that differ from error handling when the cache is in normal mode.

Table 4–35 Backup Cache ECC Error Handling During ETM

General Problem	Specific Situation and Action Taken by NVAX CPU	
Uncorrectable ECC error in the data RAMs (includes addressing errors)	Read for WRITE_ UNLOCK, VALID-OWNED hit	Hard error interrupt. When the error is detected, write data has already been merged with the corrupted data. The NVAX inverts three of the ECC check bits (bits 3,6,7), which gives a high probability that when the data is read again, an uncorrectable error will be detected.
Uncorrectable ECC error in the tag store (includes addressing errors)	Write	Soft error interrupt. The write is sent to memory. If the cache actually owned the block, the write will time out in the memory interface unless software forces the Cbox to disown the block. If the cache did not own the block, the system handles the write as it normally does for a cache that is off.
	WRITE_UNLOCK	Soft error interrupt. The write is sent to memory as a quadword length WDISOWN. Since the READ_LOCK was done just previously, memory always believes that the Bcache owns the block. In most cases, the cache itself does not have a record of owning the block since a READ_ LOCK to an owned block during ETM forces a write-back of the block. In these cases, the WRITE_UNLOCK handling is very consistent. There is only one case where the cache does own the block: if we entered ETM on or after the READ_LOCK and before the WRITE_UNLOCK. In this case, the cache may contain previously written data that is not now reflected into memory. This may be handled by software.

5

KA680 Main Memory System

The main memory system is implemented in the NVAX memory controller chip (NMC). The NMC communicates with the MS690 memory boards over the MS690 memory interconnect. Up to four MS690 memory boards are supported, for a maximum of 512 MB of main memory.

The NMC serves as an interface between the NDAL and the NVAX memory interconnect. The NMI is comprised of the set of signals leading from the NMC to the memory modules, and provides a 64-bit path to the memory modules. The arbiter for the NDAL is also built into the NMC.

5.1 Overview of the NVAX Memory Subsystem Support Functions

There are two chips that support the memory subsystem:

- The NVAX memory controller chip (NMC).
- The GMI memory interface chip (GMX).

5.1.1 The NMC Chip

The NMC controls and passes data to or from one, two, three, or four buffered memory modules using a bank interleaved memory access. It responds to commands from the CPU and the I/O adaptor (NCA). The NMC is never a commander on the NDAL.

The memory interconnect to the NMC supports the MS690 64-bit memory modules.

The MS690 memory module can have one or two banks of 1 Mb or 4 Mb DRAMs. Each bank consists of 72-1 Mb or 72-4 Mb Fast page mode 100 ns RAS access time DRAMs. Of the 72 RAMs, 64 are used to store data. The remaining 8 RAMs are used for storing ECC bits.

A given memory module is always populated with only one size of DRAM chips. The size of the memory bank is determined by reading the configuration from the memory modules. A single memory module has two banks of memory in which each bank is 16 MB or 64 MB for 1 Mb and 4 Mb DRAMs, respectively.

There is one ownership bit (O-bit) for each hexaword of data in memory. These ownership bits are implemented on the CPU module and are distinct from the MS690 memory boards. The CPU uses ownership reads (OREADs) to obtain ownership of a hexaword of data that it wishes to modify. Interlocked reads are also done as OREADs. I/O devices use OREADs to perform interlock read transactions. The ownership bits are set as a result of OREADs issued on the NDAL and are cleared by a disown write. The control signals for the O-bits are provided by a separate port on the NMC.

The CPU uses I-stream or D-stream reads to read locations that it will not modify, and ownership reads to access locations it wishes to modify (when the backup cache is on). The I/O devices use D-stream reads to do reads without locks, and ownership read/disown write pairs to implement locked transactions.

5.1.2 The GMX Chip

The GMX is a 20-bit data multiplexer and transceiver that buffers DRAMs on a memory module. It also includes support for a high-speed memory diagnostic function. It is the interface between the CMOS NMI and the TTL DRAM array.

5.2 Overview of NMC-supported NDAL Transactions

The NDAL is a 64-bit pended, synchronous bus with multiplexed data and address lines and centralized arbitration. All the components that interface to it use four clock signals from the NVAX CPU chip. The NMC is the arbiter and the default bus master. The following transactions are supported on the NDAL:

- IREAD, I-stream read.
- DREAD, D-stream read (no lock or ownership).
- OREAD, D-stream read ownership.
- RDE, read data error.
- WRITE, write masked (no disown or unlock).
- WDISOWN, disown write. These transactions use WDATA cycles to transmit write data. If the data is in error, a BADWDATA cycle is used instead.
- RDR0,RDR1,RDR2,RDR3, read data return (0 to 3).
- NOP.

5.3 Overview of NMI Transactions

The NMC performs the following transactions on the NMI:

- Quadword, octaword, and hexword OREADs, IREADs, and DREADs
- Longword, quadword, octaword, and hexword WRITEs (masked, no disown or unlock)
- Quadword and hexword disown writes
- Signature read
- Memory refresh
- Fast diagnostic test mode read or write

5.4 NMC Architectural Overview

The NVAX memory controller (NMC) serves as an interface between the NDAL (NVAX data-address lines) and the memory subsystem over a private interconnect (NMI), which is 64 bits. The NMC also serves as the arbiter for the nodes on the NDAL. The NMC is made up of five major sections - the NDAL interface, the memory interface, the control and status registers, the transactions handler, and the NDAL arbiter. This section describes these five sections and their interaction.

5.4.1 NDAL Bus Interface Architecture

The system has three nodes on the NDAL - the NMC, the NVAX CPU, and the NCA (CP-bus adapter). The NVAX CPU and the NMC can do memory transactions on the NDAL, but only the CPU can do I/O transactions. The NMC services all memory transactions (address<31:29>= 000..110) and I/O transactions written within its I/O space (address 2101 0000 - 2101 804C, 2100 0110). The NMC supports the following transactions on the NDAL:

- OREAD - ownership read
- IREAD - I-stream read
- DREAD - D-stream read
- WDISOWN - disown write
- WRITE - masked write

Every transaction on the NDAL is decoded and loaded into an appropriate input queue in the NMC. The NMC has four input queues; CPU_QUE, IO1_QUE, IO2_QUE, and WB_QUE. The CPU_QUE, IO1_QUE, and IO2_QUE queues (these will be collectively referred to as non-writeback queues, or NWB_QUEs) buffer up non-writeback transactions - OREADs, IREADs, DREADs, WRITEs. All disown writes (WDISOWN) are buffered in the writeback queue - WB_QUE. All four queues have one entry each.

5.4.1.1 The Non-Writeback Queues

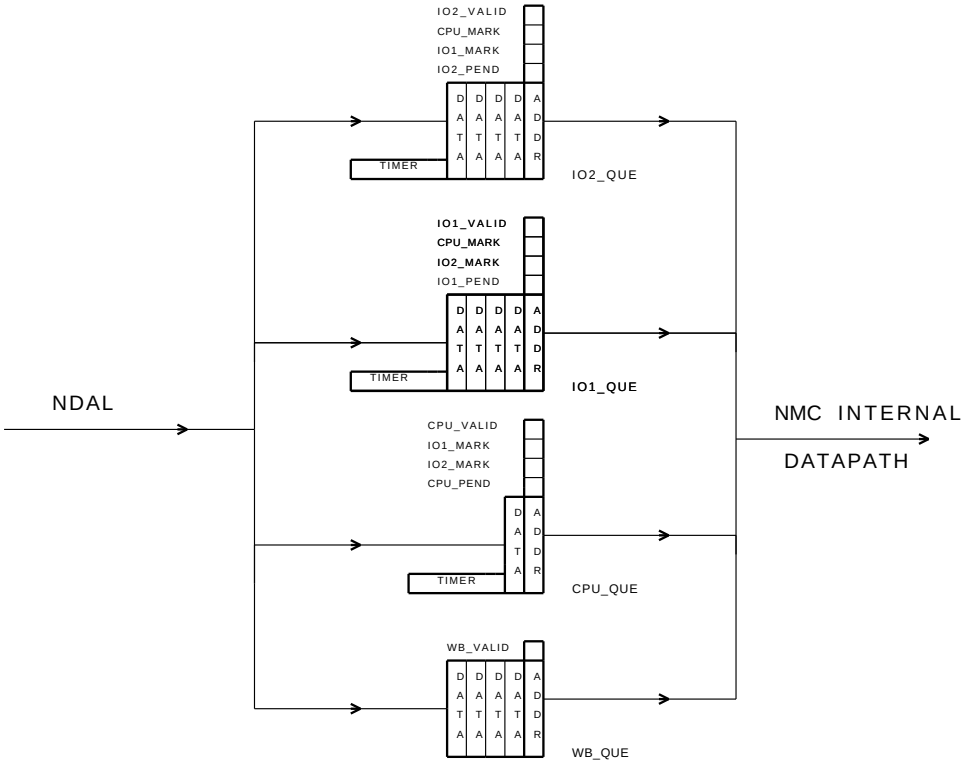
Every NWB_QUE entry contains an address packet, a data packet, a valid bit, a pending bit, and two mark bits. Figure 5–1 illustrates the organization of the NDAL IN_QUEs in the NMC.

The address packet consists of the following:

- Address - 32 bits
- Command - 4 bits
- Commander ID - 3 bits
- Byte mask - 16 bits
- Transfer length - 2 bits

The data packet in the IO1_QUE and IO2_QUE consists of 4 quadwords of data and the corresponding parity information. The data packet in the CPU_QUE consists of one quadword and the corresponding parity. The CPU_QUE data packet is only one quadword wide because during normal operation, the only writes coming from the NVAX CPU and going to memory will be write-backs, which will go into the write-back queue. The CPU_QUE will only be used when the Bcache is off or in error transition mode. In this case, the NVAX CPU writes are all of quadword length.

Figure 5–1 NDAL IN_QUEs in the NMC



The valid bit is set whenever a transaction is loaded into the queue. This also asserts a request to an internal arbiter, SEL_TRANS, which selects the transactions to be serviced by the NMC according to their priority. The valid bit is set until the transaction can be completed in memory. Refer to Section 5.4.9.1 for details.

When a memory transaction is serviced, and the corresponding hexaword is owned by another NDAL node, a pending bit is set in the NWB_QUE corresponding to that memory transaction. This bit is set until the corresponding disown write is received. Every NWB_QUE is also associated with a pending timer that counts the number of cycles a transaction has been waiting for a disown write.

Each NWB_QUE entry has two mark bits corresponding to the other two NWB_ QUEs. For instance, the CPU_QUE has two mark bits - IO1_MARK and IO2_ MARK; the IO1_QUE has mark bits CPU_MARK and IO2_MARK; and the IO2_ QUE has mark bits CPU_MARK and IO1_MARK. These mark bits are used to preserve ordering of the NDAL transactions. Ordering of transactions on the NDAL has to be preserved because not all devices use interlocked transactions for synchronization; some devices use memory reads, IO reads/writes, or interrupts. The NMC does not monitor all these synchronization events on the NDAL and therefore has to maintain order of transactions to the same hexaword address. This is done in the following way.

When the NMC sees a non-writeback memory transaction on the NDAL from a node, it performs a hexaword address compare between the incoming address and the addresses stored in valid entries of the NWB_QUEs corresponding to the other two commander nodes on the NDAL. If there is a match, the mark bit corresponding to the commander ID of the incoming transaction is set in the NWB_QUE that matched. The incoming transaction will not be serviced by the NMC until all transactions marked for its ID have been serviced. For example, the CPU does a read to hexaword H1. H1 is compared with the addresses in IO1_QUE and IO2_QUE. Suppose address H1 corresponds to a valid address in IO1_QUE. The CPU_MARK bit is set in IO1_QUE. The CPU read from H1 is not serviced by the NMC until the transaction from IO1_QUE is done. In this way, the ordering of NDAL transactions is maintained by the NMC.

5.4.1.2 The Write-back Queue

WB_QUE contains an address packet, a hexaword data packet, and a valid bit similar to the NWB_QUEs.

When a memory space disown write (WDISOWN) happens on the NDAL, the NMC compares the hexaword address of the disown write to valid addresses in the NWB_QUEs. If there is a match, and the corresponding transaction is pending, the transaction is selected for completion along with the disown write. This is described in greater detail in Section 5.4.9. The NMC allows disown writes to bypass non-writeback transactions.

WB_QUE is given the highest priority by the NMC, except when Q22–bus devices are accessing main memory. In this case, the CP-bus connected to the CQBIC is given highest priority. This is done to reduce the latency on Q22–bus transactions to main memory.

5.4.1.3 The OUT_QUE

When a read is serviced by the NMC, data to be returned on the NDAL is loaded into an "outgoing" data queue, the OUT_QUE. The OUT_QUE is unloaded when the NDAL is granted to the NMC. It can store up to 6 entries; each entry consists of 64 bits of data, the commander ID, error information, and the quadword number. Parity is generated as the information is sent from the OUT_QUE into the output pad latch. Data from the OUT_QUE is returned in the order in which it was loaded; the requested data is always returned first. Data is not necessarily returned in consecutive cycles.

5.4.2 Memory Interface Architecture

The NMC supports up to 128 MB of memory using 1 Mb DRAMs only and up to 512 MB of memory using 4 Mb DRAMs only. The minimum memory increment is 16 MB with 1 Mb DRAMs and 64 MB with 4 Mb DRAMs. The maximum available physical memory is divided into 8 sets. Each set is 16 MB if memory is made up of 1 Mb DRAMs and 64 MB if memory is made up of 4 Mb DRAMs. The NMC supports the use of 1 Mb DRAM-based memory modules with 4 Mb-based memory modules in the same system.

The NMC supports a single error correcting/double error detecting/single symbol detecting (SEC/DED/SSD) code on memory data. On the 64-bit MS690 memory modules, 72 bits are implemented - 64 bits for data and 8 ECC check bits.

Every hexaword in memory has a corresponding ownership bit (O-bit) associated with it. Single error correction on the O-bits is done by generating 4 ECC check bits across 8 O-bits. Since the system can support up to 512 MB of main memory, corresponding to 16M hexawords, there are 16M O-bits implemented on the CPU module. The NMC provides a separate O-bit port to access the O-bit memory.

The NMI supports 2-way memory interleaving. Pagemode operation of the DRAMs is used within a transaction but not across transactions. The NMC provides the basic control and timing for data memory and ownership bit memory. For details about the organization of the memory modules and the NMI interface, refer to Section 5.6.

5.4.2.1 Data Memory Addressing

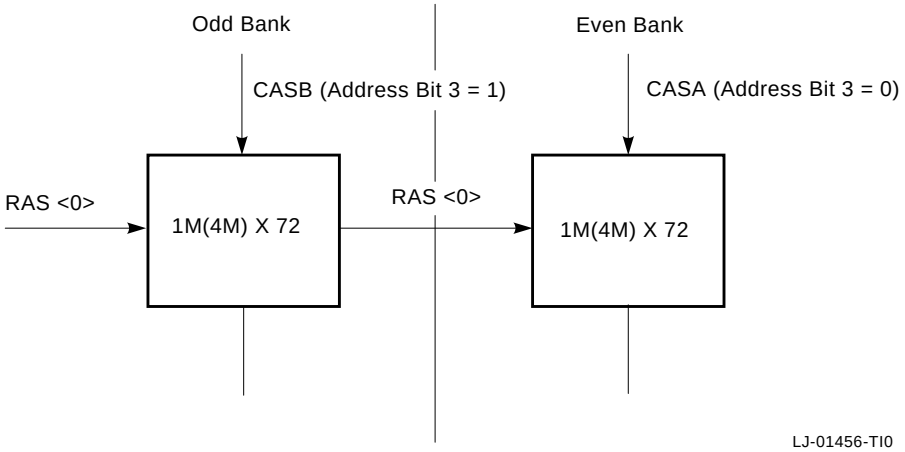
The NVAX memory space ranges from address<31:29> = 000 to address<31:29> = 110. The NVAX CPU is configured in 30-bit address mode; therefore, the maximum memory space it can support is 512 MB (bits <31:30> are ignored). Each set of memory can be mapped anywhere in this 512 MB of NVAX memory space (see the following paragraph). If a set is made up of 1 Mb DRAMs (16 MB set), 24 bits of the address are required to reference a byte within the set. Address bits <23:0> are used for this purpose. Address bits <28:24> are used to select the appropriate set. These bits contain the base address of this set. If a set is made up of 4 Mb DRAMs (64 MB set), 26 bits of the address are required to reference a byte within the set. Address bits <25:0> are used for this purpose. Address bits <28:26> are used to select the appropriate set. These bits contain the base address of this set.

A set can therefore be configured to map to any value of bits 28:24 if it is made up of 1 Mb DRAMs (bits 28:26 if it is made up of 4 Mb DRAMs). This base address mapping is stored in a corresponding configuration register. When a validated base address value matches the incoming address, the corresponding set is selected for reading or writing. Refer to Section 5.4.2.3 and Table 5–1 for further details.

5.4.2.2 Memory Set Organization

A memory set can be 16 MB or 64 MB. Each set on an MS690 memory module is made up of two banks of 72 DRAMs each. These banks are quadword interleaved; bit <3> of the address is used to select between even and odd banks. When bit <3> is 0, the even bank is selected; when bit <3> is 1, the odd bank is selected. Figure 5–2 illustrates this.

Figure 5–2 Data Memory Addressing



The following table shows the NDAL to memory address mapping for 1 Mb and 4 Mb DRAMs.

Table 5–1 Memory Address Mapping for Data

DRAM Size	Base Address	Row Address<10:0>	Column Address<10:0>
1M	<31:24>	<23:14>	<13:4>
4M	<31:26>	<25, 23:14>	<24, 13:4>

5.4.2.3 Memory Configuration

The NMC supports eight memory sets. Corresponding to each set is a configuration register and a signature register (Section 5.4.8.1.1 and Section 5.4.8.1.2). MEMCON0 and MEMSIG0 correspond to memory set 0, MEMCON1 and MEMSIG1 correspond to memory set 1, and so on. By loading the appropriate address in the corresponding configuration register, a set can be mapped to any 16 MB (for 1 Mb DRAMs) or 64 MB (for 4 Mb DRAMs) segment of the NVAX memory space. For instance, a base address of 0 in MEMCON7 would map bank set 7 to the lowest 16 MB (for 1 Mb DRAMs) or 64 MB (for 4 Mb DRAMs) of NVAX memory space.

Memory banks can be made up of 1 Mb DRAMs or 4 Mb DRAMs. The size of each memory bank can be obtained by using the following 2-step procedure:

1. Software reads the corresponding memory signature register (one of MEMSIG0 - MEMSIG7). This would cause the NMC to read the signature of the appropriate memory set and return it on the NDAL.
2. Software should then determine from this signature the type of memory set on the selected module, and then program the memory configuration register appropriately. Also stored in each memory configuration register is the base address to which each memory set responds, and a valid bit to indicate that the contents of the associated memory configuration register (MEMCON0 - MEMCON7) is valid.

A signature of 11 (binary) indicates that the corresponding set is not present in the system. The total number of sets in the system can be determined by keeping count of the number of signatures with no sets. Having determined the number of sets and their sizes, software can program the base addresses in the base address registers.

The signatures that will be returned for each of the four possible memory configurations a memory set may have are listed in Table 5–2.

Table 5–2 Memory Signature Configurations

Signature with 1 Mb DRAMs	Signature with 4 Mb DRAMs
5248AA92	2D875565

WARNING

The NMC requires that all sets with 4 Mb DRAMs be mapped on aligned 64 MB boundaries. This is a requirement because of the way the row and column addresses are generated from the incoming address. Refer to Section 5.4.2.1 for details. To enable this, all bank sets of 4 Mb DRAMs should be mapped to lower addresses than sets of 1 Mb DRAMs.

For example, consider a system with two bank sets of memory, where the first set is made up of 1 Mb DRAMs and the second is made up of 4 Mb DRAMs. MEMCON2, which corresponds to set 2 (the 4M bank is on the second memory module), should be mapped to base address <31:26> = 000000 and MEMCON0, which corresponds to set 0 (the 1M bank is on the first memory module), should be mapped to base address <31:24> = 00000100. If the base addresses were programmed the other way, the 4M set would start on an unaligned 4 MB boundary.

5.4.2.4 Ownership Bit Memory Organization and Addressing

During normal operation, the function of O-bit memory is totally transparent to software. The O-bit memory serves to maintain a working record of ownership of each hexaword of main memory to prevent simultaneous write access by both the NVAX CPU and the NCA (acting on behalf of an I/O device). This is necessary because of the write-back protocol used by the CPU's Bcache.

Every hexaword in memory has one ownership bit associated with it. 16M O-bits are implemented, thus supporting a maximum main memory of 512 MB.

O-bit memory is implemented on the CPU module. The NMC provides a separate port for this O-bit memory. The organization and addressing of this memory is different from that of the data memory. Eight O-bits are stored in every O-bit memory location. The NMC accesses eight O-bits at a time, and based on address<7:5>, decides which of the eight O-bits corresponds to the current hexaword. This scheme requires two 1M banks of 8 O-bits each to support a total of 16M O-bits; thus requiring six 1 Mb x 4 DRAMs. Each bank of O-bits has a separate CAS. The O-bit memory addresses are independent of memory address mapping. All O-bits are addressed using physical addresses from the NDAL.

Address bits <7:5> are used to select a particular O-bit in the 8-bit field. Address bit<26> is used to select the O-bit bank. Address bits <28,27,25:8> are used to address locations in the DRAMs.

Table 5–3 and Figure 5–3 illustrate the mapping scheme.

Figure 5–3 O-bit Port Addressing

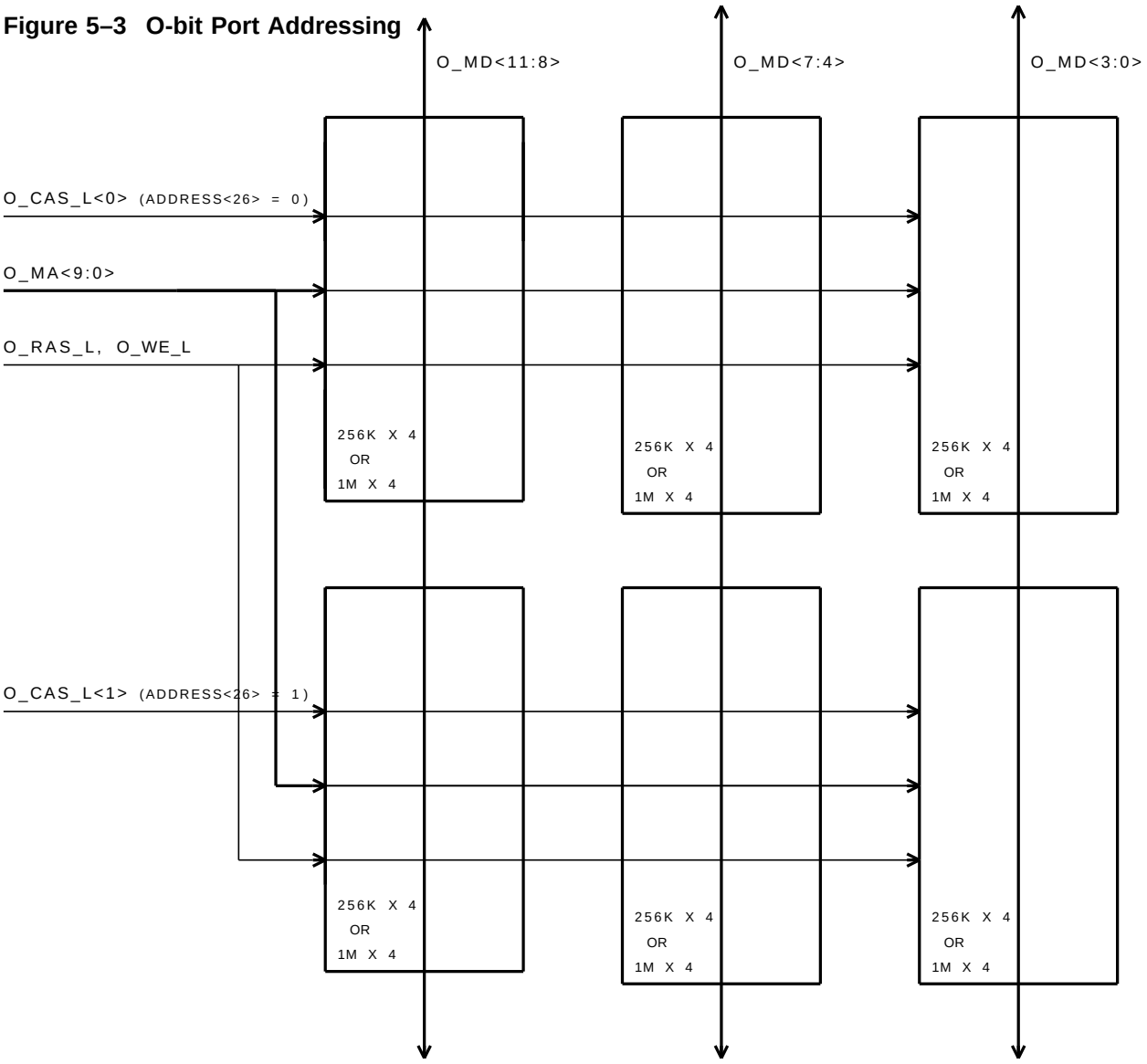


Table 5–3 O-bit Port Address Mapping

Maximum Main Memory	O-bit Row Address<9:0> [1]	O-bit Column Address<9:0>[2]
512 MB	<28,25:17>	<27,16:8>

[1]Address<26> determines the O-bit bank to be selected. If address<26> is 0, O_CAS_L<0> is asserted; if address<26> is 1, O_CAS_L<1> is asserted.

[2]Address<7:5> are used to select one of the eight O-bits in the 8-bit field.

5.4.3 NMI Transactions

The NMC supports the following transactions on the NMI:

- Refresh
- Signature read
- Data read
- Unmasked data write
- Masked data write
- Nonexistent memory access

5.4.3.1 Refresh

The NMC provides the mechanism to refresh the DRAMs for the data and ownership memories. If 1 Mb DRAMs are used, every row has to be refreshed once every 8 ms. If 4 Mb DRAMs are used, every row has to be refreshed once every 16 ms. This means that the interval between refreshes of two consecutive rows has to be 15.62 µs. When idle, the NMC generates a new refresh transaction every 13.44 µs. If a refresh request happens during a transaction, that transaction is completed before the refresh transaction is done on the NMI. The NMC allows a margin of 2.18 µs over the DRAM specification for this reason. The NMC has an internal refresh interval timer that initiates a refresh transaction every 320 NDAL cycles. The NMC also provides the refresh address for the DRAMs on the MA<10:0> lines. It contains a 10-bit binary counter, the refresh address counter, which generates consecutive addresses for every refresh. 1 Mb DRAMs need a 9-bit refresh address; 4 Mb DRAMs need a 10-bit refresh address. The 1 Mb DRAMs are thus refreshed at twice the rate of the 4 Mb DRAMs.

An O-bit memory refresh is done along with a data memory refresh. Refresh address <9:0> is mapped onto O_MA<9:0>.

5.4.3.2 Signature Read

A signature read transaction is initiated by doing a read to one of the memory signature registers, MEMSIG0-7. On these transactions, there is no O-bit access. Data received from the memory module is returned unchanged on the NDAL.

5.4.3.3 Read/Write Transactions

The NMC does quadword, octaword, or hexaword read transactions, and longword, quadword, octaword, or hexaword write transactions on the NMI. The NMI memory is quadword interleaved. Consecutive quadwords in 64-bit mode are read/written from/to the even and odd banks of each memory set. A transfer on the NMI is defined as one access from/to the DRAMs. A quadword transaction is one transfer, an octaword two, and a hexaword four; a longword write transaction is implemented as a read-modify-write. The requested data is always accessed first. The NMC uses page mode of the DRAMs to do a multitransfer transaction but it does not use page mode between two transactions.

- When the NMC starts a memory read transaction on the NMI, it simultaneously issues an O-bit read of the ownership RAMs on the CPU module. If the transaction is a memory read and the O-bit is set, the transaction is kept pending. The read data is discarded and subsequent transfers are aborted. (Refer to Section 5.4.9 for details.)

- If the NDAL transaction being serviced is an unmasked write, the write is done in parallel with an O-bit read. If the hexaword being written is found to be owned, the write is aborted after the first transfer, and is kept pending. (Refer to Section 5.4.9.) The first transfer can be done before it is known whether the location is owned because even if it is owned, since the NDAL transaction was an UNMASKED write, the entire hexaword will be overwritten as soon as the disown write comes from the owner. Therefore, it is all right to allow the first data transfer to take place to the memory before it is known whether the location is owned.
- If the NDAL transaction is a masked write, a single transfer read is done and the O-bit is read in parallel. If the write is not owned, it is completed by merging the write data with the data from memory and writing it back. If the write data is owned, it is aborted after the read and kept pending.
- If the NDAL transaction that is being serviced is a disown write, and it is found to be unowned when the O-bit is read, an error is flagged but the write is completed.
- If a write transaction has a completely masked transfer (all corresponding byte masks are equal to 0; no bytes will be written), then the CAS corresponding to that transfer is not asserted (that is, no write is done to the DRAMs).

5.4.3.4 Nonexistent Memory Access

If the incoming address bits <28:24> (<28:26> for 4 Mb DRAMs) do not match any of the programmed base addresses in the memory configuration register, the NMC memory interface flags an error by returning read data error NDAL cycle on a read or requesting a hard error interrupt (SCB offset=60_{16}) on writes. In every case, the NMC responds to the NDAL transaction regardless of the fact that it is to nonexistent memory. This prevents errors in the NVAX CPU resulting from NDAL timeouts on read transactions.

5.4.4 Error Checking for Data Memory

The NMC implements a single bit error correcting, double bit error detecting, single symbol (nibble) error detecting (SEC/DED/SSD) code across 64 bits of memory data (Figure 5–4). On memory write transactions, the NMC generates ECC on the outgoing data and writes it into memory along with the data. On memory read transactions, the NMC reads the memory data, generates the corresponding ECC, compares the generated check bits with the incoming check bits, and generates a syndrome. If the syndrome is a 0, there is no error in the incoming data. If the syndrome matches a column in the code of Figure 5–4, there is a correctable error in the corresponding data bit. The NMC returns the corrected data. If the syndrome is not 0 and does not match any of the columns, then the error is uncorrectable.

If, during a write transaction, there is a parity error in the NDAL data, or if an illegal command is received on an NDAL write data cycle, or a BADWDATA NDAL command (Section 3.11) is received on the NDAL, then the NMC forces incorrect check bits in memory. This is done by inverting the three least significant check bits. When, at a later stage, this data is read from memory, an uncorrectable syndrome of 07_{16} will be received.

Figure 5–4 SEC/DED/SSD Code Used in the NMC

```
DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DDDD DCCC CCCC CDDD
0000 0000 0011 1111 1111 2222 2222 2233 3333 3333 4444 4444 4455 5555 5555 6BBB BBBB B666
0123 4567 8901 2345 6789 0123 4567 8901 2345 6789 0123 4567 8901 2345 6789 0012 3456 7123

 1101 0001 1101 1101 1100 1110 0011 0010 0001 1101 1101 1100 1110 0011 0010 0100 0000 0000 S0
 1010 0010 1011 1010 1011 1101 0100 0100 0010 1011 1010 1011 1101 0100 0100 1010 0000 0000 S1
 0111 1100 0110 0111 0111 0011 1000 1001 1100 0110 0111 0111 0011 1000 1001 0001 0000 0000 S2
 0001 1111 0001 0001 0001 1111 0001 0001 1111 0001 0001 0001 1111 0001 0001 1000 1000 0100 S3
 1111 1001 0010 0100 1000 0110 0111 1101 1001 0010 0100 1000 0110 0111 1101 0000 0100 0010 S4
 1111 0110 0100 1000 0010 1001 1101 1011 0110 0100 1000 0010 1001 1101 1011 0000 0010 0001 S5
 1110 0000 1000 0010 0100 1111 1011 0111 1111 0111 1101 1011 0000 0100 1000 1000 0001 0111 S6
 0000 0000 0000 0000 0000 0000 0000 0000 1111 1111 1111 1111 1111 1111 1111 0000 0000 1111 S7*

S0, S1, S2, S4, S5, S6 ---- Even Parity
S3, S7                 ---- Odd Parity

*S7 Depends on Data Bits <63:32> Only.
```

LJ-01321-TI0

5.4.5 Error Checking for Ownership Bit Memory

The NMC supports single error correction across eight O-bits in memory. An O-bit memory read/write accesses eight O-bits and four ECC check bits. During an O-bit read transaction, the NMC generates the check bits on the incoming O_MD<7:0> (the eight O-bits being read) and XORs them with the received check bits, O_MD<11:8>. If the resulting syndrome is 0, there is no error in the data. If the syndrome matches any of the columns in Figure 5–5, then the error is correctable. This code does not detect all multiple bit errors. Other than single bit errors the NMC detects the all 0s and all 1s failures on O_MD<11:0>.

Figure 5–5 Single Error Correcting Code for O-bit Memory

```
                C C C C
DDDDDDDD        B B B B
01234567        0 1 2 3

11101001        1 0 0 0   S0
11010110        0 1 0 0   S1
10110101        0 0 1 0   S2
01111010        0 0 0 1   S3
```

LJ-01322-TI0

5.4.6 Memory Diagnostic Support

The NMC facilitates data memory testing in two ways - fast diagnostic mode, and diagnostic check bit mode. These modes can be used separately or simultaneously.

The NMC allows memory error detection to be disabled to allow testing the check bit memory DRAMs. The NMC does not have to be in either of the two diagnostic modes for memory error dectection to be disabled. It can be disabled by setting MMCDSR<DIS_MEM_ERR_DETECT>.

WARNING

Both of these modes are for diagnostic purposes only; they should not be set during normal system operation.

5.4.6.1 Fast Diagnostic Mode

Fast diagnostic mode allows main memory to be tested at a fast rate by reading, writing, and comparing locations from more than one memory bank at a time. The NMC enters fast diagnostic mode when MMCDSR bit<FAST_DIAG_MODE> is set. (Refer to Table 5–8.) The implementation of fast diagnostic mode is the same as that in the PELE system. (Refer to the GMX specification.) When the NMC is in fast diagnostic mode, it sets MODE_SEL<1> to "1" on every read or write transaction. If at the same time the FDM second pass bit is set in MMCDSR (Table 5–8), MODE_SEL<0> is also set to "1".

5.4.6.2 Diagnostic Check Bit Mode

Diagnostic check bit mode enables software to force an arbitrary value on the data memory check bits. Diagnostic check bit mode can be entered by setting MMCDSR bit<DIAG_CKB_MODE>. (Refer to Table 5–8.)

When diagnostic check bit mode is set, on a memory write transaction, the check bit field in MMCDSR<MEM_DIAG_CKBS> is written to memory instead of the check bits generated by the ECC logic. The check bits received on a memory read transaction can be obtained by reading MMCDSR<MEM_DIAG_CKBS> or MMCDSR<MEM_CHECK_BITS>, regardless of the value of MMCDSR<DIAG_CKB_MODE>.

Memory tests using diagnostic check bit mode should be run from the ROM with both the Pcache and Bcache off. This forces all memory reads to be of quadword length and is required when using diagnostic check bit mode. The NMC only logs the check bits corresponding to the first two transfers of a memory read transaction. If software wants to read the memory check bits, it should read the corresponding memory location and immediately follow it up with an MMCDSR read. This is necessary to ensure that the read check bits loaded in MMCDSR correspond to the correct read data. The NMC logs the diagnostic check bits on any memory read transactions, including the read part of a masked write.

5.4.7 Ownership Bit Memory Diagnostic Support

The NMC provides 4K quadword aligned registers (O-bit data registers) to access the O-bit memory in I/O space. These registers range from 2101 0000 - 2101 7FFF. The NMC supports a maximum of 2M O-bit locations. These locations can be made up of 512 segments, where each segment consists of 4K O-bit locations. One of the 4K NMC O-bit data registers corresponds to 512 possible O-bit locations, one for each segment. The segment number can be specified in an O-bit address and mode register. Bits <14:3> of the addresses of MODRs along with the segment number in the O-bit address and mode register are used to specify an O-bit location.

The address and mode register also contains a 3-bit mask field to specify a particular O-bit within an O-bit memory location.

O-bit memory can be accessed in one of 5 modes - reconstruction mode, memory test mode with ECC, memory test mode with forced check bits, fast memory test mode with ECC, and fast memory test mode with forced check bits.

To perform a diagnostic operation on O-bit memory, software should load the appropriate address and mode in the O-bit address and mode register (MOAMR), and then do a read or a write to the appropriate O-bit data register. The different O-bit diagnostic modes are described in the following sections.

5.4.7.1 Reconstruction Mode

This mode allows software to change the value of a particular O-bit. The O-bit address register should be loaded with the segment address corresponding to that O-bit, the mask should be written, and the mode should be set to reconstruction. Then, software should do a write transaction to the corresponding O-bit data register. This would cause the NMC to do a read-modify-write transaction on O-bit memory location and update the O-bit.

A read in this mode results in the entire O-bit field being returned on bits <11:0>.

5.4.7.2 Memory Test Mode with ECC

This mode allows software to overwrite the data part of an O-bit memory location without the check bits. The check bits are generated by the ECC logic. A write to an O-bit data register in this mode forces the data bits <7:0> to be written onto the corresponding O-bit memory location.

A read in this mode results in the entire O-bit field being returned.

5.4.7.3 Fast Memory Test Mode with ECC

The fast memory test mode with ECC allows software to overwrite O-bit memory data as in memory test mode with ECC. In this mode, two locations are written and read. When a write is done to an O-bit I/O address in this mode, data from bits <7:0> and <19:12> is written to two consecutive O-bit memory locations, respectively.

A read of the O-bit data register in this mode returns data from two consecutive O-bit locations in bits <11:0> and <23:12>, respectively.

5.4.7.4 Memory Test Mode with Forced Check Bits

This mode allows software to overwrite one entire O-bit memory location (eight O-bits plus four check bits) with any desired pattern. A write to an O-bit data register in this mode forces the data bits <11:0> to be written onto the corresponding O-bit memory location.

A read of the O-bit data register in this mode results in a read of the corresponding O-bit memory location. The data that is read from O-bit memory is returned on bits <11:0>.

5.4.7.5 Fast Memory Test Mode with Forced Check Bits

The fast memory test mode with forced check bits allows software to overwrite O-bit memory locations as in memory test mode with forced check bits. In this mode, two locations are written and read. When a write is done to an O-bit data register in this mode, data from bits <11:0> and <23:12> is written to two consecutive O-bit memory locations, respectively.

A read of the O-bit register in this mode returns data from two consecutive O-bit locations on bits <11:0> and <23:12>, respectively.

5.4.8 I/O Section

This section consists of the control and status registers (NMC_CSRs) and the decoders for I/O and memory addresses. It does the following operations:

- CSR reads
- CSR writes

- Clear write buffer

The I/O space of the NMC is defined as 2101 0000 - 2101 804C, 2100 0110 (hex); the NMC acknowledges these I/O addresses only.

5.4.8.1 Registers

The NMC has 21 control and status registers (NMC_CSRs) that can be read or written by the CPU. These CSRs are addressed as aligned longwords only. Quadword reads or writes to NMC_CSRs are not supported.

In addition, it has 4K O-bit data registers, which are quadword-aligned.

All the NMC_CSRs are cleared on power-up reset, unless otherwise specified in the following description. Write operations to read-only registers do not cause any errors and are responded to as a normal operation; however, the operation does not alter any NMC register contents. Table 5–4 lists all the registers and their addresses. Included in this list is the clear write buffer register.

Table 5–4 NMC Registers

Number	Name	Address	No. of Registers
MEMCON0-7	Memory Configuration Registers	2101 8000 - 801C	8
MEMSIG0-7	Memory Signature Registers	2101 8020 - 803C	8
MEAR	Error Address Information Registers	2101 8040	1
MESR	Error Status Register	2101 8044	1
MMCDSR	Mode Control and Diagnostic Register	2101 8048	1
MOAMR	O-bit Address and Mode Register	2101 804C	1
MEMCON0-19	NMC Registers	2101 8000 - 804C	20
MCWB	Clear Write Buffer Register	2100 0110	1
MODRs	O-bit Data Registers	2101 0000 - 7FFF	4K

5.4.8.1.1 Memory Configuration Registers (MEMCON0 - MEMCON7) The NMC supports up to 8 memory sets of 16 MB with 1 Mb DRAMs and 64 MB with 4 Mb DRAMs. Each set is associated with one software programmable register. Each register stores set specific information consisting of a set base address, a set enable, and the set signature. These registers are written following a signature read transaction from MEMSIG0-7 and the computation of the base addresses. The following is a description of all the bit fields in these registers.

Figure 5–6 Memory Configuration Registers

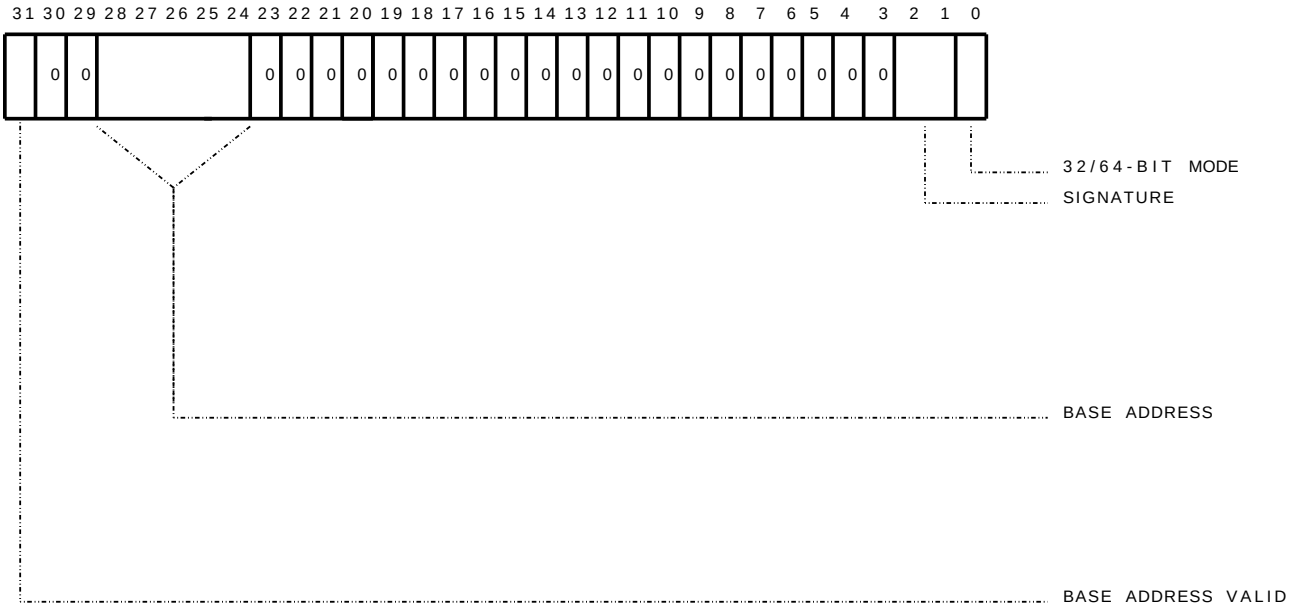


Table 5–5 Memory Configuration Registers, MEMCON0-7

Register Field	Bits	Type, Reset State	Description
Base Address Valid / Set Enable	31	RW, 0	This read/write bit indicates that the base address programmed in bits <28:24> and the signature in bits <2:1> are valid. This bit is cleared on powerup. It should be set when the base address and the signature are written to enable addressing of the set. In addition, this bit may be used by diagnostics to selectively disable sets.
Unused	30:29	MBZ, 0	This field reads as 0.
Base Address	28:24	RW, 0	Specifies the memory base address of the related set. If the RAM size of the set is 1 Mb, then all 5 bits are used in the address comparison. If the RAM size of the set is 4 Mb, then only bits <28:26> are used in the address comparison. In either case, all 5 bits can be read and written. See Section 5.4.2.1 and Section 5.4.2.3 for details on the use of the base address.
Unused	23:3	MBZ, 0	This field reads as 0.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 5–5 (Cont.) Memory Configuration Registers, MEMCON0-7

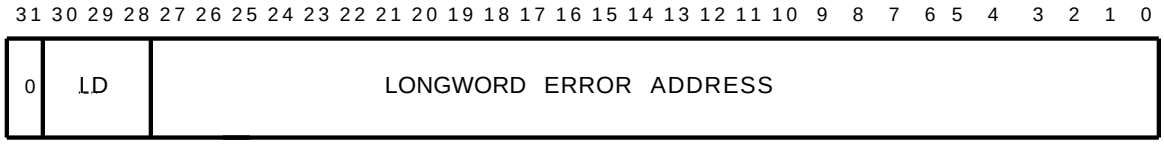
Register Field	Bits	Type, Reset State	Description
Signature	2:1	RW, 0	Indicates the RAM size of the corresponding memory set. It has to be written to by software after doing a signature read of the corresponding set.
			Value — Configuration
			00 — Unassigned
			01 — RAM size 1 Mb
			10 — RAM size 4 Mb
			11 — Nonexistent bank
Must be 1	0	RW, 0	This bit must be set to a 1 by software. This bit, when clear, specifies memory controller operation to be 32-bit mode. when set, it indicates 64-bit mode. Since the KA680 only supports 64-bit memory, this bit must be set to a 1 by ROM software during power-up initialization to enable operations with memory.

5.4.8.1.2 Memory Signature Registers (MEMSIG0 - MEMSIG7) Each set of physical memory banks has a signature register associated with it. MEMSIG0 corresponds to set 0, MEMSIG1 to set 1, and so on. A read from any of these registers causes the NMC to do a signature read transaction on the NMI to the corresponding set. The information returned on a read to these registers should be written by software to the appropriate bits in the corresponding memory configuration register (MEMCON0-7). For instance, a read from MEMSIG0 would return the signature of memory set 0; this should be written by software to MEMCON0.

The signature information includes the size of the DRAMs used to make up that memory set and the width of memory data, if the set is present in memory. If the set is not present, a value of FFFFFFFF (hexadecimal) is returned as read data. During system power-up configuration and self-test, the ROM software reads the signature values. Based on the signature that is returned read from each MEMSIG register, the corresponding MEMCON register should be programmed accordingly. Table 5–2 shows the two possible values that will be returned when a MEMSIG register is read for an existing memory set.

5.4.8.1.3 Error Address Information Register (MEAR) When an error is logged by the NMC in MESR, the corresponding address and commander ID are logged in MEAR. This information is loaded by the first error and is not changed until all the error bits are cleared. These registers are read-only and have valid information only when an error bit is set in MESR.

Figure 5–7 Error Address Information Register



MEAR : 2101 8040

Table 5–6 Error Address Information Register, MEAR

Register Field	Bits	Type, Reset State	Description
Unused	31	MBZ, 0	This bit is unused and is read as 0.
ID	30:28	RO, 0	This field contains the ID of the commander corresponding to the transaction in error. The ID is logged on NDAL errors only; memory errors do not log the ID.
Error Address	27:3	RO, 0	This field contains the hexaword address at which the error occurred. This field is always logged.
Error Address	2:1	RO, 0	This field indicates the quadword at which the error occurred. This field is logged on memory errors. This field is logged on NDAL write data parity errors, and on NDAL illegal write command errors. This field is not valid on unowned disown write errors and disown write timeout errors. The address is valid up to the hexaword only. This field is not valid on nonexistent memory errors.
Error Address	0	RO, 0	This bit indicates the longword address of a memory transaction with an error. This bit is not valid on NDAL errors.

RO—Read-only
RW—Read/write
WC—Write one-to-clear

5.4.8.1.4 Error Status Register (MESR) The NMC reports error information in MESR.

Figure 5–8 Error Status Register

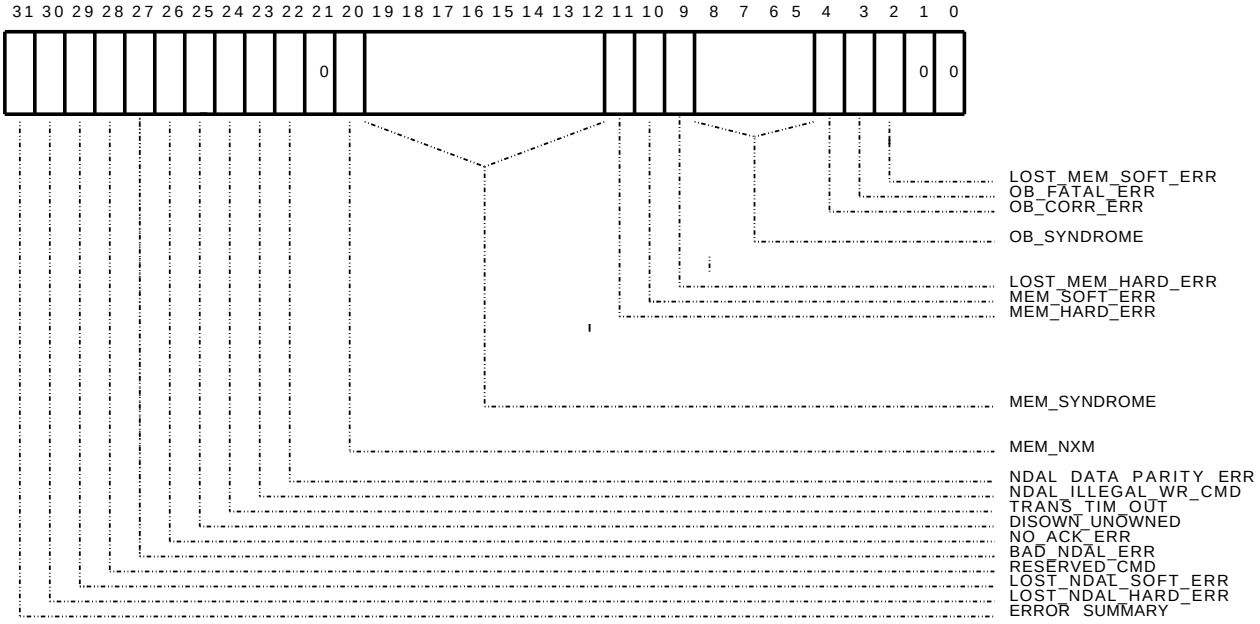


Table 5–7 Error Status Register, MESR

Register Field	Bits	Type, Reset State	Description
Error Summary	31	RO, 0	This bit is "1" when any error is logged by the NMC in this register. It is clear when all the error bits are cleared.
LOST_NDAL_HARD_ ERR	30	WC, 0	This bit is set if a pending transaction times out, a disown write is found to be unowned, or a nonexistent address is received on the NDAL, and MEAR cannot be loaded because of a previous error that has not been cleared.
LOST_NDAL_SOFT_ERR	29	WC, 0	This bit is set if a an NDAL data parity error or an illegal write command is detected on the NDAL, and MEAR cannot be loaded because of a previous error that has not been cleared and soft error logging is enabled (MMCDSR<EN_SOFT_ERR_LOG> is 1). This bit is also logged if an NDAL data parity error or an illegal write data command is detected on the NDAL and soft error logging is disabled (MMCDSR<EN_SOFT_ERR_LOG> is 0).

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 5–7 (Cont.) Error Status Register, MESR

Register Field	Bits	Type, Reset State	Description
NDAL_RESERVED_CMD	28	WC, 0	This bit is set when the NMC receives a reserved command on the NDAL. The address is not logged in MEAR.
BAD_NDAL_ERR	27	WC, 0	This bit is set when the NMC receives a parity error on an NDAL address cycle or if it receives an illegal length code on an otherwise valid transaction. The address is not logged in MEAR.
NO_ACK_ERR	26	WC, 0	This bit is set when the NMC does not receive ACK_L when it returns read data on the NDAL. The address is not logged in MEAR.
DISOWN_UNOWNED	25	WC, 0	This bit is set if the NMC receives a disown write to an unowned location and MEAR can be loaded with the address information.
PEND_TRANS_TIM_OUT	24	WC, 0	This bit is set when a pending transaction times out waiting for the corresponding disown write, and the address can be saved in MEAR.
NDAL_ILLEGAL_WR_ CMD	23	WC, 0	This bit is set by the NMC when it receives a command other than WDATA or BADWDATA on the data part of an NDAL write, and MEAR is free to load address information.
NDAL_DATA_PAR_ERR	22	WC, 0	This bit is set if a parity error is detected on the NDAL during a data cycle of a write transaction, and if MEAR is free to load the address information.
Unused	21	MBZ, 0	This bit reads as 0.
MEM_NXM	20	WC, 0	This bit is set if the address received on the NDAL is in memory space, but does not match any of the programmed banks in the NMC memory configuration registers. It is set only if MEAR can be loaded with new address information.
MEM_SYNDROME	19:12	RO, 0	This field stores the memory error syndrome and is loaded when an ECC data memory error is detected on the NMI. The syndrome is logged only when the corresponding memory error is not logged as a lost error.
MEM_HARD_ERR	11	WC, 0	This bit is set if an uncorrectable ECC error occurred during a memory read transaction, and if MEAR can be loaded with address information.
MEM_SOFT_ERR	10	WC, 0	This bit is set if a correctable ECC error occurred during a memory read or a masked write transaction, or if an uncorrectable error occurred during a masked write transaction, and if MEAR can be loaded with address information.

(continued on next page)

Table 5–7 (Cont.) Error Status Register, MESR

Register Field	Bits	Type, Reset State	Description
LOST_MEM_HARD_ERR	9	WC, 0	This bit is set if an uncorrectable ECC error occurred during a memory read transaction, and if MEAR could not be loaded with address information because a previous error bit had been logged in MESR.
OB_SYNDROME	8:5	RO, 0	This field stores the O-bit error syndrome when an error is detected in O-bit memory reads. The O-bit syndrome is logged only when the corresponding O-bit error is not logged as a lost error.
OB_CORR_ERR	4	WC, 0	This bit is set if a correctable O-bit memory error occurred during an O-bit read transaction, and if MEAR can be loaded with address information.
OB_FATAL_ERR	3	WC, 0	This bit is set if the O-bit ECC logic detects a syndrome of 1111(binary), indicating that the incoming 12-bit O-bit data has an all 1s or all 0s failure. No address information is logged.
LOST_MEM_SOFT_ERR	2	WC, 0	This bit is set when: a correctable error occurs during a memory read or a masked write transaction, a correctable error occurs on an O-bit read transaction, or an uncorrectable data memory error occurs on a masked write transaction, and a previous error has been logged in MEAR and soft error logging is enabled (MMCDSR<EN_SOFT_ERR_LOG> is 1). This bit is also set when: a correctable error occurs during a memory read or a masked write transaction, a correctable error occurs on an O-bit read transaction, or an uncorrectable data memory error occurs on a masked write transaction and soft error logging is disabled (MMCDSR<EN_SOFT_ERR_LOG> is 0), irrespective of the state of other error bits in MESR.
Unused	1:0	MBZ, 0	This field is unused.

5.4.8.1.5 Mode Control and Diagnostic Status Register (MMCDSR) NMC operating modes are controlled by bits in this register. In addition, this register is used for storing diagnostic status information. The following is a description of all the bit fields in this register.

Figure 5–9 Mode Control and Diagnostic Status Register

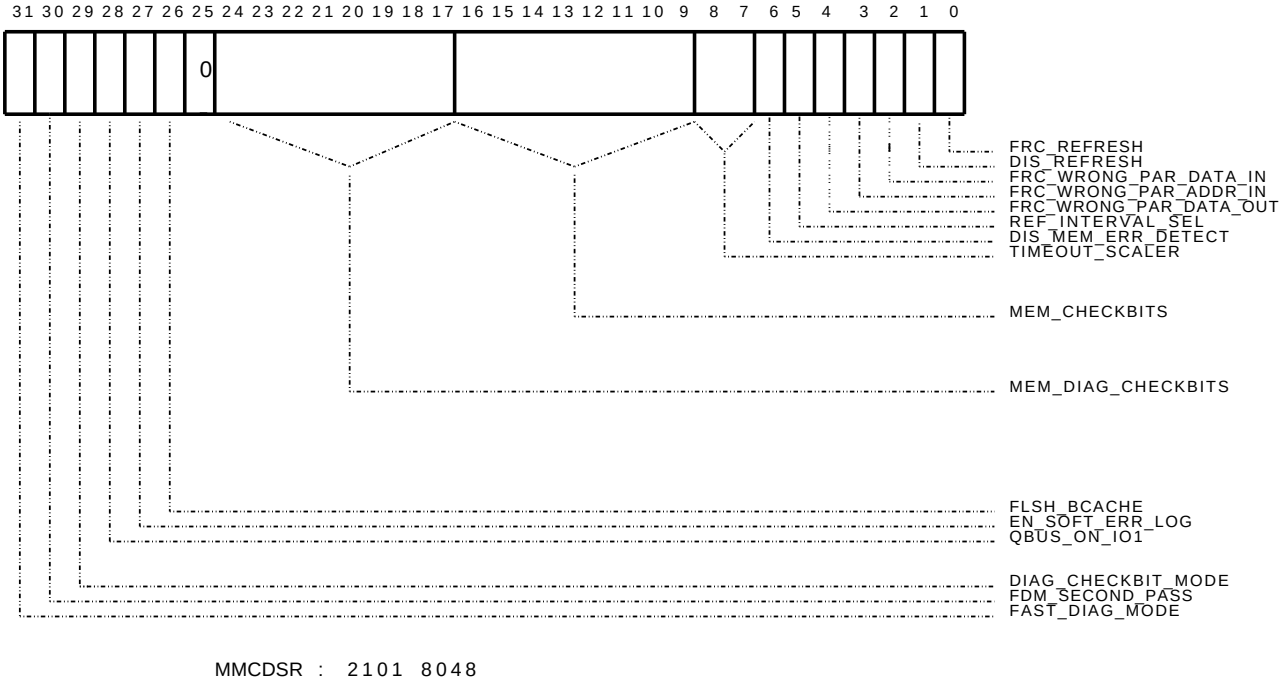


Table 5–8 Mode Control and Diagnostic Status Register, MMCDSR

Register Field	Bits	Type, Reset State	Description
FAST_DIAG_ MODE	31	RW, 0	This bit provides the mechanism for speeding up initial diagnostic testing of memory. This bit indicates to the NMC that it is in fast diagnostic mode.
FDM_SECOND_ PASS	30	RW, 0	In a system with two sets of memory banks, fast diagnostic mode has to be done in two passes. This bit has to be set when the second pass of the test has to be run. This bit is valid only when MMCDSR<FAST_DIAG_MODE> is set.
DIAG_CKB_MODE	29	RW, 0	When set to 1, this bit enables the contents of MMCDSR <MEM_DIAG_CKBS> to be driven onto the check bit field of the memory data, instead of the normal ECC check bits. When this bit is a 0, the contents of MMCDSR<MEM_DIAG_CKBS> are ignored during memory write transactions. MMCDSR<MEM_DIAG_CKBS> should be valid when this bit is set.

RO—Read-only
RW—Read/write
WC—Write one-to-clear

(continued on next page)

Table 5–8 (Cont.) Mode Control and Diagnostic Status Register, MMCDSR

Register Field	Bits	Type, Reset State	Description
QBUS_ON_IO1	28	RW, 0	This bit is used by the NMC to indicate on which CP-bus the Q22–bus interface (CQBIC) resides. THIS BIT IS CLEARED ON POWERUP AND MUST NOT BE CHANGED.
EN_SOFT_ERR_ LOG	27	RW, 0	When this bit is a 0, NDAL and memory-related soft errors are detected but no information is logged in MEAR and MESR, and S_ERR_L is not asserted. If the error happened on the NDAL, MESR<LOST_ NDAL_SOFT_ERR> is logged; if the error happened in memory, the MESR<LOST_MEM_SOFT_ERR> is logged. When this bit is 1, all the soft errors are logged normally as described in the description of MESR, and S_ERR_L is asserted whenever a correctable error occurs. This bit does not affect hard error logging.
FLUSH_BCACHE	26	RO, 0	When this bit is set by software, the NMC asserts the CPU_WB_ONLY signal on the NDAL, which informs the NVAX CPU to refrain from performing non-writeback transactions on the NDAL. The purpose of this bit is to allow the Bcache to be flushed without creating excessive NDAL traffic that might otherwise adversely affect the latency of devices on CP1 and CP2 (namely, the Q22–bus).
Unused	25	RW, 0	–
MEM_DIAG_CKB	24:17	RW, 0	This field is ignored on memory write transactions when diagnostic check mode (MMCDSR<DIAG_ CKB_MODE>) is cleared. During diagnostic check bit mode, the bits in this field are written to memory instead of the check bits generated by the ECC logic. A read of MMCDSR returns the check bits corresponding to the first transfer of a memory read transaction that happened prior to this NMC_ CSR read. This field is loaded on memory reads or masked write transactions.
MEM_CHECKBITS	16:9	RO, 0	This field contains the memory check bits corresponding to the second transfer of a memory read transaction that happened prior to this NMC_ CSR read. This field is loaded on memory reads or masked write transactions.
TIMEOUT_ SCALER	8:7	RW, 0	This 2-bit field is used as a prescaler to the disown write timeout counter.

Value	Timer Count	µs (36 ns)	µs (42 ns)
00	2600	93.6	109.2
01	1600	57.6	67.2
10	800	28.8	33.6
11	400	14.4	16.8

(continued on next page)

Table 5–8 (Cont.) Mode Control and Diagnostic Status Register, MMCDSR

Register Field	Bits	Type, Reset State	Description
DIS_MEM_ERR	6	RW, 0	When this bit is 1, memory error detection and correction is disabled. All memory error related logging in MEAR and MESR is disabled. S_ERR_L does not assert on correctable errors in memory data; S_ERR_L does not assert on uncorrectable errors in the read data of a masked write. A read data error response is not returned on the NDAL when uncorrectable errors occur in memory read data.
REF_INT_SEL	5	RW, 0	When this bit is 1, the NMC uses an alternate refresh interval for use in conjunction with NDAL cycle times longer (slower) than 42 ns. Because the KA680 and KA680 CPU modules operate at 42 ns NDAL cycle times or faster, this bit should not be set by software because it will cause excessive memory refresh cycles and subsequently reduce system performance.
FRC_WRONG_ PAR_DATA_OUT	4	RW, 0	When set to 1, this bit forces the NDAL parity generator to generate incorrect parity on the command and ID field of a read data return cycle. This will cause the commanders to not ACK the response and timeout, waiting for the appropriate data. The wrong parity is forced for one NMC responder transaction only. This bit is self-clearing; it is always read as 0. This bit is for test purposes only and should not be set during normal system operation.
FRC_WRONG_ PAR_ADDR_IN	3	RW, 0	When set to 1, this bit forces the NDAL parity generator in the NMC to generate incorrect parity on the command and ID field of an address cycle addressed to it. This results in the NMC detecting incorrect parity on incoming transactions, thus causing it to not ACK the commander and assert S_ERR_L. The wrong parity is forced for one NDAL transaction (not a NOP) only. This bit is self-clearing. It is always read as 0. This bit is for test purposes only and should not be set during normal system operation.
FRC_WRONG_ PAR_DATA_IN	2	RW, 0	When set to 1, this bit forces the NDAL parity generator to generate incorrect parity on the command and ID field of a write data. This causes the NMC to respond to the cycle with ACK_L, but assert S_ERR_L and force incorrect check bits in memory data. The wrong parity is forced for one NDAL data cycle only. This bit is self-clearing; it is always read as 0. This bit is for test purposes only and should not be set during normal system operation.

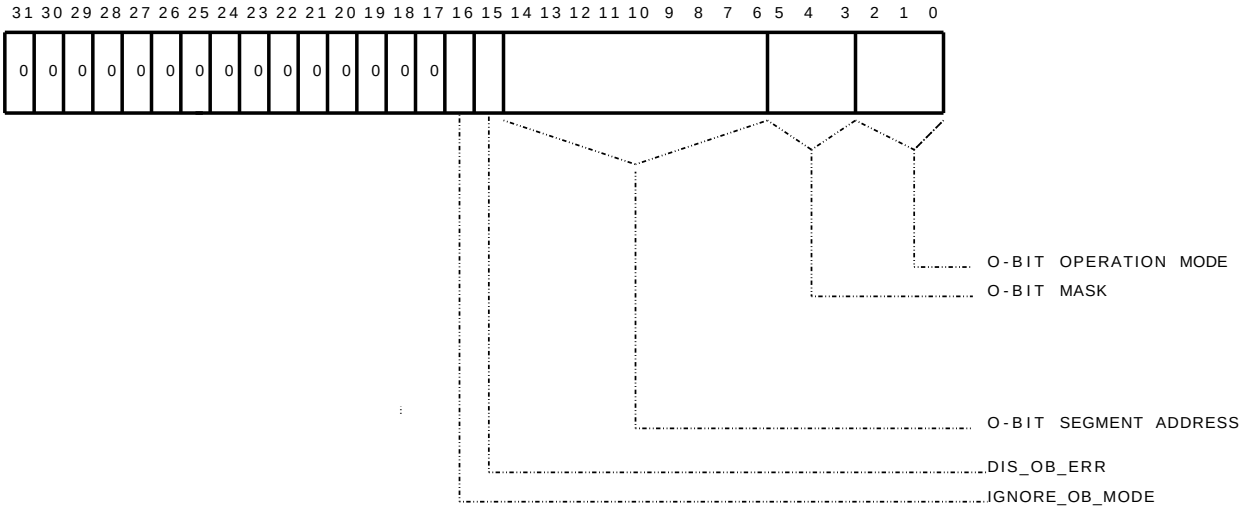
(continued on next page)

Table 5–8 (Cont.) Mode Control and Diagnostic Status Register, MMCDSR

Register Field	Bits	Type, Reset State	Description
DIS_REFRESH	1	RW, 0	This bit, when set to 1, disables memory refresh transactions, irrespective of the state of MMCDSR <FRC_REFRESH>, the force refresh bit. It also clears the refresh interval and address counters to 0. This functionality has been added for test purposes only and should not be used in normal system operation.
FRC_REFRESH	0	RW, 0	When this bit is cleared to 0, the refresh control logic behaves normally and refreshes are done on the NMI when the refresh interval timer overflows. When this bit is set to 1, it forces the memory interface of the NMC to do continuous memory refreshes as long as there are no requests for memory transactions. If the memory is requested to do a read, write, or signature read transaction, the stream of memory refreshes is interrupted.

5.4.8.1.6 O-bit Address and Mode Register (MOAMR) This register, along with the O-bit data register, facilitates initialization and testing of O-bit memory. This register stores the address and mode of operation.

Figure 5–10 O-bit Address and Mode Register



MOAMR : 2101 804C

Table 5–9 O-bit Address and Mode Register, MOAMR

Register Field	Bits	Type, Reset State	Description
Unused	31:17	MBZ, 0	This field is read as 0.
Ignore O-bit Mode	16	RW,0	Setting this bit causes the memory controller to ignore the state of the ownership bits when processing memory transactions. This bit is for diagnostic purposes only and should not be set under normal conditions.
Disable O-bit Errors	15	RW,0	Setting this bit causes the memory controller to ignore any ECC errors in the O-bits when processing memory transactions. This bit is for diagnostic purposes only and should not be set under normal conditions.
O-bit Segment Address	14:6	RW, 0	This field contains the segment address corresponding to the O-bit to be accessed.
O-bit Mask	5:3	RW, 0	This field contains the O-bit mask. It is used in reconstruction mode to specify the O-bit to be accessed.
O-bit Operation Mode	2:0	RW, 0	This field indicates the mode of operation on a read or write of the O-bit data registers. The following is the assignment of bits<2:0> in this field.

Value	O-bit Mode
000	Reconstruction mode
001	Unassigned
010	Memory test mode
011	Fast memory test mode
100	Unassigned
101	Unassigned
110	Memory test mode with forced check bits
111	Fast memory test mode with forced check bits

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

5.4.8.1.7 O-bit Data Registers (MODRs) There are 4K O-bit data registers; each register corresponds to one of 512 O-bit memory locations. The NMC uses the segment number provided in the O-bit address and mode register to determine the appropriate O-bit location addressed. A read or write from/to an O-bit data register causes the NMC to do a read/write from/to the O-bit memory location whose segment number corresponds to the value in the O-bit address and mode register.

Figure 5–11 O-bit Data Registers

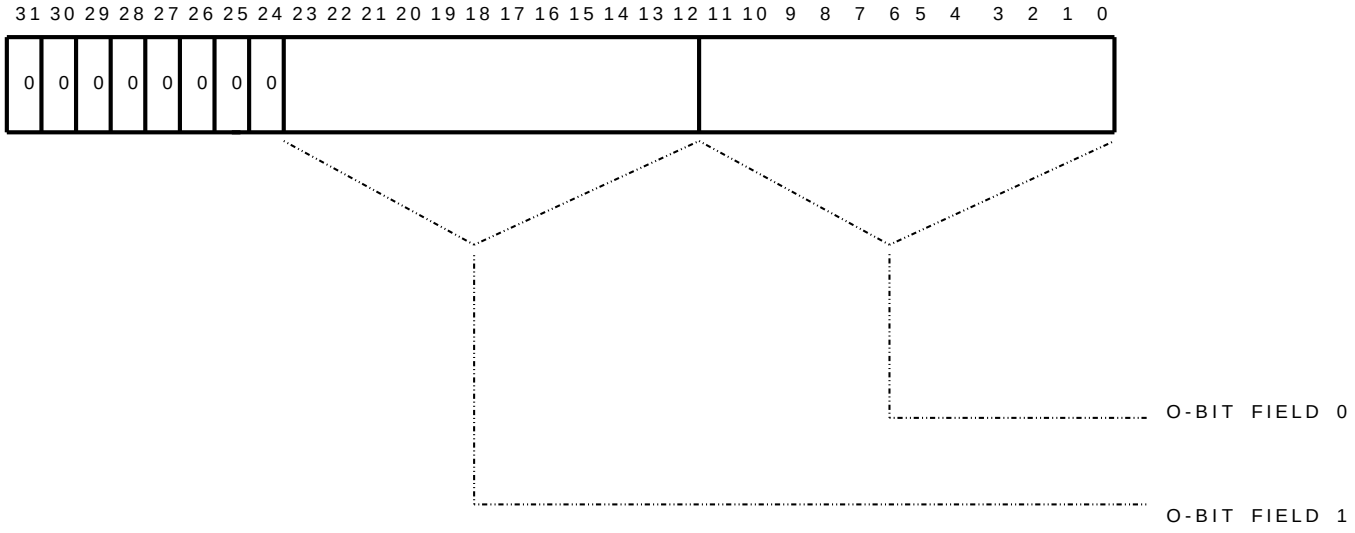


Table 5–10 O-bit Data Registers, MODR

Register Field	Bits	Type, Reset State	Description
Unused	31:24	MBZ, 0	This field is read as 0.
O-bit Field 1	23:12	RW, 0	This field is used in fast memory test only. It contains the O-bit field (O-bits + check bits) for the second O-bit location to be read/written in fast memory test mode. When the force check bits mode is disabled in the O-bit address and mode register, only bits <19:12> of this field are valid.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 5–10 (Cont.) O-bit Data Registers, MODR

Register Field	Bits	Type, Reset State	Description
O-bit Field 0	11:0	RW, 0	This field contains the value for the O-bit field to be used in the first part of a fast O-bit test mode transaction, memory test mode, and reconstruction mode. A read of MODRs returns the entire O-bit location in fast memory test, memory test, and reconstruction mode. A write to this field in reconstruction mode requires valid data in the appropriate O-bit only. A write to this field in O-bit memory test or fast O-bit memory test mode with the force check bits mode disabled requires valid data in bits <7:0> only. A write to this field in O-bit memory test or fast O-bit memory test mode requires valid data in all 12 bits.

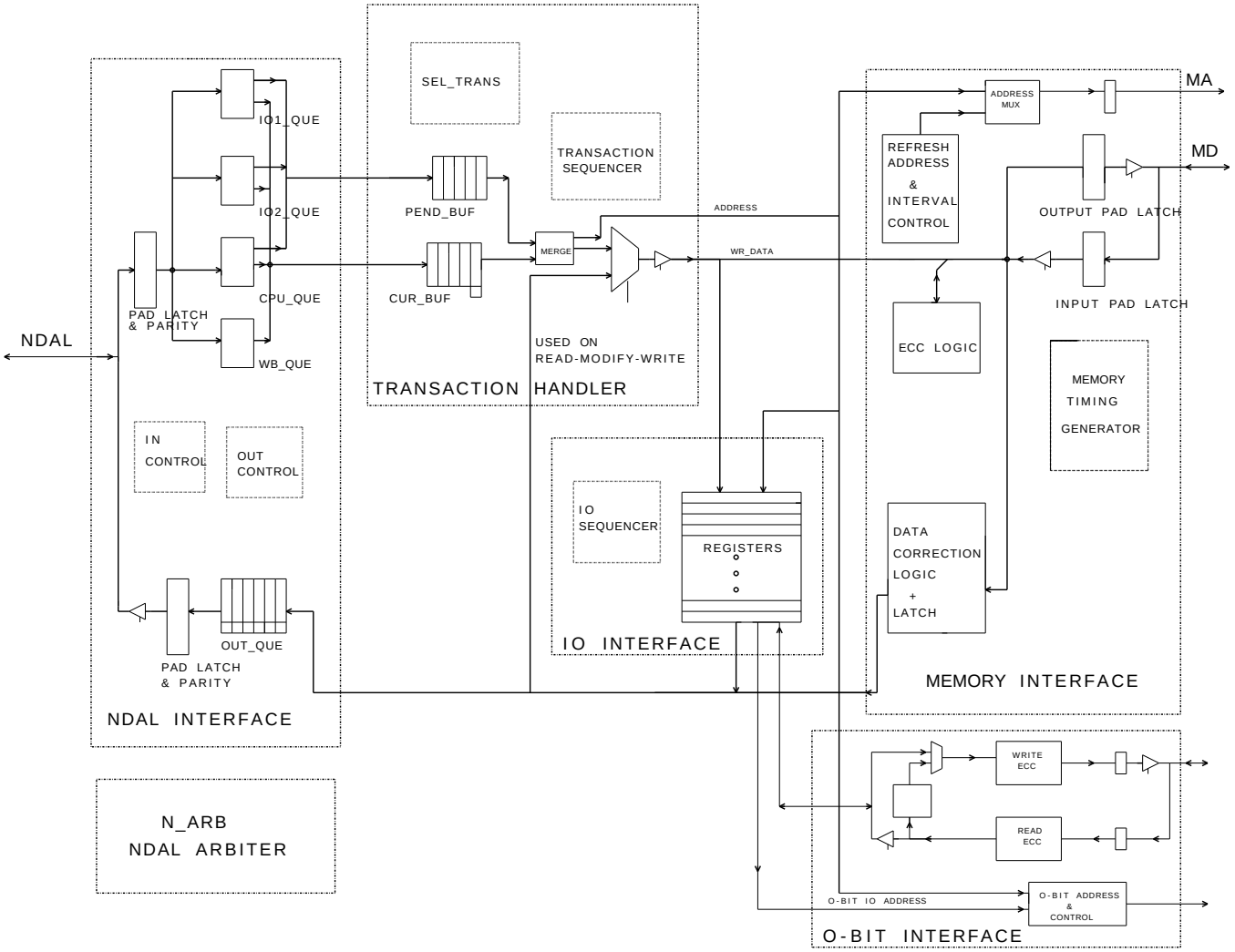
5.4.8.1.8 Clear Write Buffer Register (NMC_CSR20)

The clear write buffer register address, 2100 0110, is in the NVAX CPU IPR address range. It is longword-aligned. A read or write to this register is required to flush any CPU write buffers in the NMC. The NMC buffers up CPU writes in its CPU_QUE and CPU disown writes in WB_QUE. Since the CPU_QUE of the NMC is one deep and WB_QUE is given the highest priority by the NMC, all writes and write-backs from the CPU that happened before a clear write buffer transaction on the NDAL are completed in memory before the clear write buffer is serviced by the NMC. Thus, the NMC does not need to take any special action on a clear write buffer transaction. A read from NMC_CSR20 returns all zeros as the data; a write does nothing.

5.4.9 NMC Transaction Handling

The previous sections in this chapter described the architecture and organization of four major blocks in the NMC: the NDAL interface, the memory interface, the O-bit interface, and the I/O interface. This section describes the fifth block in the NMC, the transaction handler. Figure 5–12 illustrates the organization of these five blocks.

Figure 5–12 NMC Block Diagram



The transaction handler consists of an internal arbiter. The data path of the transaction handler includes a current transaction buffer and one pending transaction buffer. These blocks are described in greater detail in the following sections.

5.4.9.1 NMC Internal Arbitration

Transactions loaded into the IN_QUEs of the NDAL interface of the NMC are selected for servicing by an internal arbiter. This internal arbiter can get up to four requests, one from each of the three NWB_QUEs, and one from WB_QUE. In the absence of memory accesses from the Q22–bus, the WB_QUE is given the highest priority, and the three NWB_QUEs are given equal priority; they are serviced in round-robin. When Q22–bus devices are accessing main memory, IO2_QUE is given a higher priority over WB_QUE (which helps reduce Q22–bus latency), and transactions from IO1_QUE and CPU_QUE are not selected unless they are marked.

5.4.9.2 Transaction Handler Datapath

When a transaction is selected for processing by the NMC, the information from the corresponding NMC's IN_QUE entry is loaded into the current transaction buffer, CUR_BUF. CUR_BUF contains all the information that an IN_QUE entry contains.

When a disown write transaction is received on the NDAL, and there is a transaction pending to the same hexaword in one of the NWB_QUEs, the address information and data (writes only) for that transaction are loaded into a "pending buffer"- PEND_BUF, and the corresponding pending, valid, and mark bits for that entry are cleared.

5.4.10 NMC Transactions

This section describes how the various transactions are handled by the NMC.

5.4.10.1 Ownership Read

When an ownership read transaction is received in CUR_BUF, the data read and the O-bit read are started in parallel.

If the corresponding O-bit is not set, indicating that the hexaword can be owned by the requesting commander, the hexaword of data is loaded into the OUT_QUE as it is received from memory with the requested quadword loaded first. The O-bit is set by the memory interface after it checks the memory data for errors. The O-bit is only set when there are no uncorrectable errors in the requested memory data or the O-bit field. As a performance enhancement, the O-bit interface unconditionally sets the O-bit after every OREAD transaction; if later a memory data uncorrectable error is found, the original state of the O-bit is restored. The corresponding IN_QUE entry is cleared as soon as the transaction handler determines that the hexaword is not owned.

If the corresponding O-bit is set, indicating that the hexaword is owned by another device, the "pending bit" is set in the appropriate NWB_QUE, the pending timer for that queue is started, and the read is aborted on the NMI.

5.4.10.2 Memory Read

The flow for the memory read is the same as that of the OREAD, the only difference being that the O-bit is not set at the end of the read.

5.4.10.3 Memory Write

On a memory write transaction, the data write and the O-bit read are done in parallel, instead of reading the O-bit first and doing the write only if the O-bit is cleared. Starting the write and the O-bit read in parallel saves time on writes that are not owned.

If the hexaword is not owned (the corresponding O-bit is clear), the write is completed. The corresponding IN_QUE entry is cleared as soon as the transaction handler finds out that the transaction is not owned.

If the hexaword is owned (the corresponding O-bit is set), the write is aborted (after one transfer on an unmasked write, or after the read part of a masked write), the corresponding pending bit is set, and the pending timer is started. At this point, some of the data corresponding to the aborted write is written to memory. This results in an apparent memory incoherency. However, since the corresponding hexaword is owned by some other node, the data in memory is stale anyway and has to be updated. The disown write flow ensures the coherence of memory.

5.4.10.4 Disown Write

Disown writes can be quadword or hexaword writes. A quadword disown write may be a write unlock from the NCA on behalf of an I/O device, or it could be from the CPU (when the Bcache is off or in error transition mode). The CPU may also do a quadword disown write instead of a hexaword disown if it has seen one of the other commanders do a hexaword write to the same location; in this case, it masks out all the data. The reason for this optimization is as follows: if the CPU owns a hexaword of memory, and it sees on the NDAL a hexaword write transaction (from the NCA) to that location, then according to NDAL protocol it must relinquish ownership of the hexword via a disown write transaction (WDISOWN). Recall, however, that the hexaword write from the NCA will be queued up in the NMC, waiting for the WDISOWN from the CPU. Since whatever value the CPU specifies in the WDISOWN would be immediately overwritten by the NCA's pending hexaword write, the CPU saves three NDAL cycles by only writing a quadword of data with the WDISOWN transaction.

When the NMC's transaction sequencer receives a disown write, it checks the state of the pending bits in the NWB_QUEs to see if there are any pending reads or writes to that hexaword.

If there are no pending transactions, the memory write and an O-bit read are started in parallel. If the hexaword is not owned, an error is flagged but the data write is completed. If the O-bit is set, the write is completed and the O-bit is cleared. The corresponding IN_QUE entry is cleared as soon as the O-bit read is complete.

If a write is pending to the same hexaword, the IN_QUE entry corresponding to the pending write is cleared by the transaction handler. Merging of data occurs according to the byte masks of the pending write, and the data is written to memory. The data in memory is overwritten by the merged data, and memory coherence is preserved.

If the pending transaction is a read, and the disown write is a hexaword, read data is returned as the disown write is written to memory. If the disown write is a quadword, the read is done from memory, and the write data is merged with the appropriate quadword and returned on the NDAL. If the pending transaction is a normal memory read, the O-bit is cleared at the end of the disown write. If the pending transaction is a ownership read, the O-bit is set at the end of the disown write if no uncorrectable memory errors have been found in the requested quadword.

5.4.11 I/O Transactions

The NMC supports I/O reads or writes to its internal registers only. No ownership protocol is supported on I/O addresses. Ownership reads or disown writes to I/O space are treated as normal reads and writes.

5.4.12 NMC Error Handling

Errors in the NMC can be of two types—NDAL-related errors and memory-related errors. These are described in the following tables.

Table 5–11 NDAL-related Errors and NMC Responses

Description of Error	Specific Situation	Action Taken by NMC
NMC detects a parity error on an NDAL cycle	Presumed address cycle, not an NMC write data cycle	Soft error interrupt on reads, both hard and soft error interrupt on writes; MESR<BAD_NDAL_ERR> logged. No address information is logged.
	Presumed data cycle on a write; address has been received previously	Soft error interrupt; logs the error in MESR<NDAL_DATA_PAR_ERR>. Memory write is done and incorrect check bits are forced in memory data. CSR writes are aborted, no write is done, soft error interrupt. Quadword address and commander ID are logged in MEAR.
NMC detects an illegal length on an NDAL address cycle to its address space	Presumed address cycle	ACK_L not asserted; S_ERR_L asserted; MESR<BAD_NDAL_ERR> logged. No address information is logged.
NMC detects a reserved command on an NDAL cycle	Presumed address cycle	Soft error interrupt on reads to NMC, hard error interrupt on writes to NMC; MESR<RESERVED_CMD> is logged. No address information is logged.
Illegal write data CMD	Data cycles on writes	Write for the specified length is done; incorrect check bits are loaded in memory for the quadword with the incorrect command; soft error interrupt. Error logged in MESR<ILLEGAL_WR_CMD>. On CSR write transactions, a soft error interrupt is requested, and the write is aborted. Quadword error address and commander ID are logged.
No acknowledgement when requested read data is returned by the NMC	IREAD, DREAD	Soft error interrupt is requested; logs the error in MESR<NO_ACK_ERR>. No address logged; the corresponding commander should log the address information. All read data is returned.
	OREAD	Soft error interrupt is requested; logs the error in MESR<NO_ACK_ERR>. No address information is logged; the corresponding commander should log address information. All read data is returned. Does not affect the setting of the O-bit.
Pending transaction times out waiting for disown write	Write	Hard error interrupt is requested; logs the error information in MESR<PEND_TRANS_TIM_OUT>. Hexaword address and ID are logged. Transaction is aborted.
	Read	Read data error transaction returned on requested quadword. Log the error information in MESR<PEND_TIM_OUT>. Hexaword address and commander ID are logged.

(continued on next page)

Table 5–11 (Cont.) NDAL-related Errors and NMC Responses

Description of Error	Specific Situation	Action Taken by NMC
Disown write to an unowned location	WDISOWN	Hard error interrupt requested. Do the write. Pending transactions will be completed. Log the error in MESR<DISOWN_UNOWNED>. Hexaword address and commander ID are logged.
Nonexistent memory address	Read	Read data error returned on requested data. Error logged in MESR<MEM_NXM>. Hexaword address and commander ID are logged.
	Write	Hard error interrupt is requested. Error logged in MESR<MEM_NXM>. Hexaword address and commander ID are logged.

Table 5–12 Memory-related Errors and NMC Responses

Description of Error	Specific Situation	Action Taken by NMC
NMC detects an uncorrectable data memory error	Owned reads, Owned masked writes	The NMC does not flag any errors and does not log any bits in this case. A memory transaction to a location that is found to be owned is not completed until the corresponding disown write happens. The data in memory is stale, and the disown write may overwrite it, thus writing correct data over the stale data with errors. No error will be logged in this case. If the disown write does not overwrite this location (quadword disown), then the uncorrectable data memory error is found again, and will be flagged and logged this time.
	Memory reads (not owned)	Read data error is returned on that transfer; subsequent transfers are aborted. Error is logged in MESR<MEM_HARD_ERR>. Longword address is logged.
	OREADs (not owned)	Read data error is returned on that transfer; subsequent transfers are aborted. Error is logged in MESR<MEM_HARD_ERR>. Longword address is logged. O-bit not set if error occurred on the requested quadword.
	Masked memory disown writes	S_ERR_L is asserted; the write corresponding to that transfer does not happen. If a read is pending to that location, RDE is returned on the read. In this case, no error interrupt is requested. Error is logged in MESR<MEM_SOFT_ERR>. Longword address is logged.
	Masked memory writes, no disown (not owned)	Soft error interrupt is requested; the write corresponding to that transfer does not happen. Error is logged in MESR<MEM_SOFT_ERR>. Longword address is logged.

(continued on next page)

Table 5–12 (Cont.) Memory-related Errors and NMC Responses

Description of Error	Specific Situation	Action Taken by NMC
NMC detects a fatal O-bit error	All transactions	An O-bit fatal error is flagged when the NMC does an O-bit read transaction that has an error syndrome of 1111 (binary). This syndrome is received if the O-bit read returns all 0s or all 1s in the 12-bit field. When this error happens, the NMC requests a hard error interrupt, and logs MESR<OB_FATAL_ERR> bit. No address is logged, and the transactions are completed as if there were no error.
NMC detects a correctable data memory error	Memory reads	Data is corrected and returned on the NDAL. Soft error interrupt requested. Data is not changed in memory. Error is logged in MESR<MEM_CORR_ ERR>. Longword address is logged.
	Masked memory writes	Soft error interrupt is requested; read data is corrected and merged with write data and retired to memory. Error is logged in MESR<MEM_CORR_ ERR>. Longword address is logged.
NMC detects a correctable error in O-bit field	Read, writes	Soft error interrupt is requested. Error is logged in MESR<OB_CORR_ERR>. Hexaword address is logged in MEAR.

5.4.13 NDAL Arbitration

The three nodes on the NDAL request the bus whenever they have a transaction to perform on the NDAL. A commander could initiate a transaction or a responder could be responding to a pending transaction. The NMC is a responder only, the NVAX CPU is a commander only, and the NCA can be both a commander and responder, although not during the same transaction. The NMC contains the arbiter (N_ARB) for the NDAL. Arbitration is done in parallel with data transfer cycles using a set of lines dedicated specifically for arbitration. They are detailed in this section.

The NMC request is asserted a cycle before quadword read data is loaded into the OUT_QUE. The N_ARB priority scheme is discussed in this section.

When WB_QUE is full, the N_ARB does not grant the bus to any node except the NMC.

When a non-writeback transaction is recieved by the NMC, the WB_ONLY signal corresponding to that commander is asserted. N_ARB does not grant the bus to that node for the next cycle.

The NMC is given the highest priority by N_ARB. The two I/O ports of the NCA are given the second priority; they are serviced with a round-robin scheme. The NVAX CPU has the lowest priority. Table 5–13 indicates the priority of the NDAL arbiter.

Table 5–13 NDAL Arbitration Priority

Priority	Node
1	NMC
2	IO1, IO2
3	CPU

The NMC request is never ignored by N_ARB. If no node requests the NDAL, the NMC is given the grant (it is the default bus master of the NDAL).

5.5 NMC Initialization

5.5.1 Internal Register States

The following list describes the state of each NMC_CSR when NDAL_RESET_L asserts.

- Memory configuration registers (MEMCON0 - MEMCON7) :
  These registers are cleared to 0.
- Memory signature registers (MEMSIG0 - MEMSIG7) :
  These registers are virtual registers in the NMC and have no power-up value.
- Error address information register (MEAR) :
  This register is cleared to 0.
- Error status register (MESR) :
  This register is cleared to 0.
- Mode control and diagnostic status register (MMCDSR):
  This register is cleared to 0.
- O-bit address and mode register (MOAMR):
  This register is cleared to 0.
- O-bit data registers (MODRs):
  These registers are virtual registers in the NMC and have no value on reset.

5.5.2 Counter States

The following list describes the state of all the counters in the NMC when NDAL_ RESET_L asserts.

- Pending transaction timers :
  These timers are cleared to the ZERO state: the prescaler is also cleared to 0 so that the timeout interval is 2600 cycles.
- Refresh interval timer:
  This timer is cleared to the ZERO state.
- Refresh address counter:
  This counter is cleared to 0 so that the first refresh address after NDAL_ RESET_L deasserts is 0.

5.6 Memory Subsystem Organization

The NMC supports a 64-bit memory interconnect.

5.6.1 64-bit Interconnect

The memory subsystem consists of up to four memory modules having up to four 72-bit banks each (64 bits data and 8 bits ECC). There can be a total of up to 16 banks of memory. The memory banks are 2-way interleaved. This requires that each module has an even number of banks. The following discussion refers to bank pairs. The module organization is shown in Figure 5–13.

Up to 512 MB of ECC memory can be supported by the KA680 when using memory modules populated with 4 Mb DRAMs. KA680 memory systems can also contain a mixture of 4 Mb and 1 Mb based memory modules, although the maximum memory will be lower than 512 MB when one or more 1 Mb based modules are used.

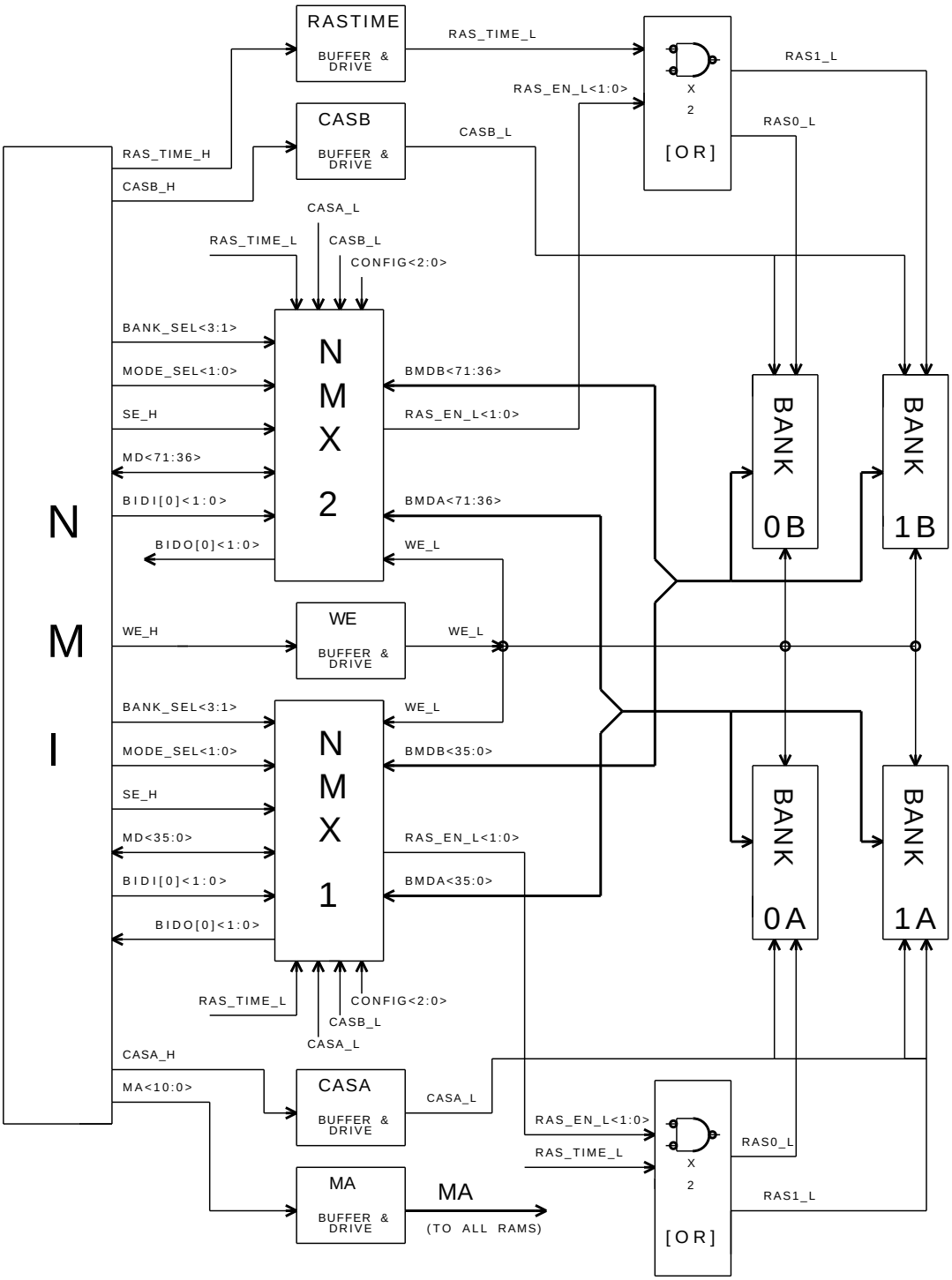
The NVAX memory space is mapped to the physical memory using the memory configuration registers in the NMC. The NMC requires that all bank pairs with 4 Mb DRAMs be mapped on aligned 64 MB boundaries. To enable this, all bank pairs with 4 Mb DRAMs should be mapped to addresses lower than the 1 Mb DRAMs.

5.6.2 GMX Chip

Each memory module has four transceiver chips (GMX) that perform data multiplexing between the NMI bus and the DRAM array. The GMX chips also perform other functions such as memory signature read and fast memory diagnostics.

Each GMX has four data bus ports—BMD0A<19:0>, BMD0B<19:0>, BMD1A<19:0>, and BMD1B<19:0>. On the MS690 memory modules, all the GMX's BMD lines are connected to one bank of DRAMs each. Thus, 64-bit MS690 memory boards have four GMX chips to handle the 72 data and ECC bits.

Figure 5–13 Memory Organization with 64-bit Interconnect



6

KA680 I/O Subsystem

The I/O subsystem is controlled by the NVAX I/O adapter chip (NCA). The NCA is a 339-pin custom VLSI chip packaged in a PGA package. The NCA functions as a bidirectional bus adapter between the NDAL and two CVAX-compatible peripheral buses : the CP1-bus on port 1 and the CP2-bus on port 2. On the NDAL, the NCA supports the NVAX CPU chip and the NVAX memory controller chip (NMC). On the CP1-bus and CP2-bus, the NCA supports the shared host adapter chip (SHAC), the second generation Ethernet controller (SGEC), the system support chip (SSC), and the CMOS Q22–bus adapter chip (CQBIC).

6.1 NCA Overview

The NCA provides a bidirectional interface between the NDAL and two CVAX-compatible peripheral (CP) buses: the CP1-bus on NCA port 1 and the CP2-bus on NCA port 2. The NVAX CPU, the NVAX memory controller (NMC), and the NCA are connected to the NDAL bus.

The CP1 and CP2 are CVAX-compatible peripheral (CP) buses for the system's DMA I/O devices and program-controlled I/O devices.

The NCA supports the CQBIC, the SSC, the SGEC, and two SHACs on the CP buses.

6.2 I/O System Configuration

The I/O bus configuration used on the KA680 is shown in Chapter 1. In this configuration, the Q22–bus, SSC, and the console firmware ROMs reside on the CP2-bus of the NCA. They are isolated in this manner in order to minimize the latency incurred when the KA680 is responding to Q22–bus transactions as a slave.

6.3 NCA Chip Architecture

The NCA serves as a bidirectional adapter between the NVAX DAL (NDAL) and two CVAX peripheral buses (CP1 on port 1 and CP2 on port 2). The four major functional blocks of the NCA chip are listed below:

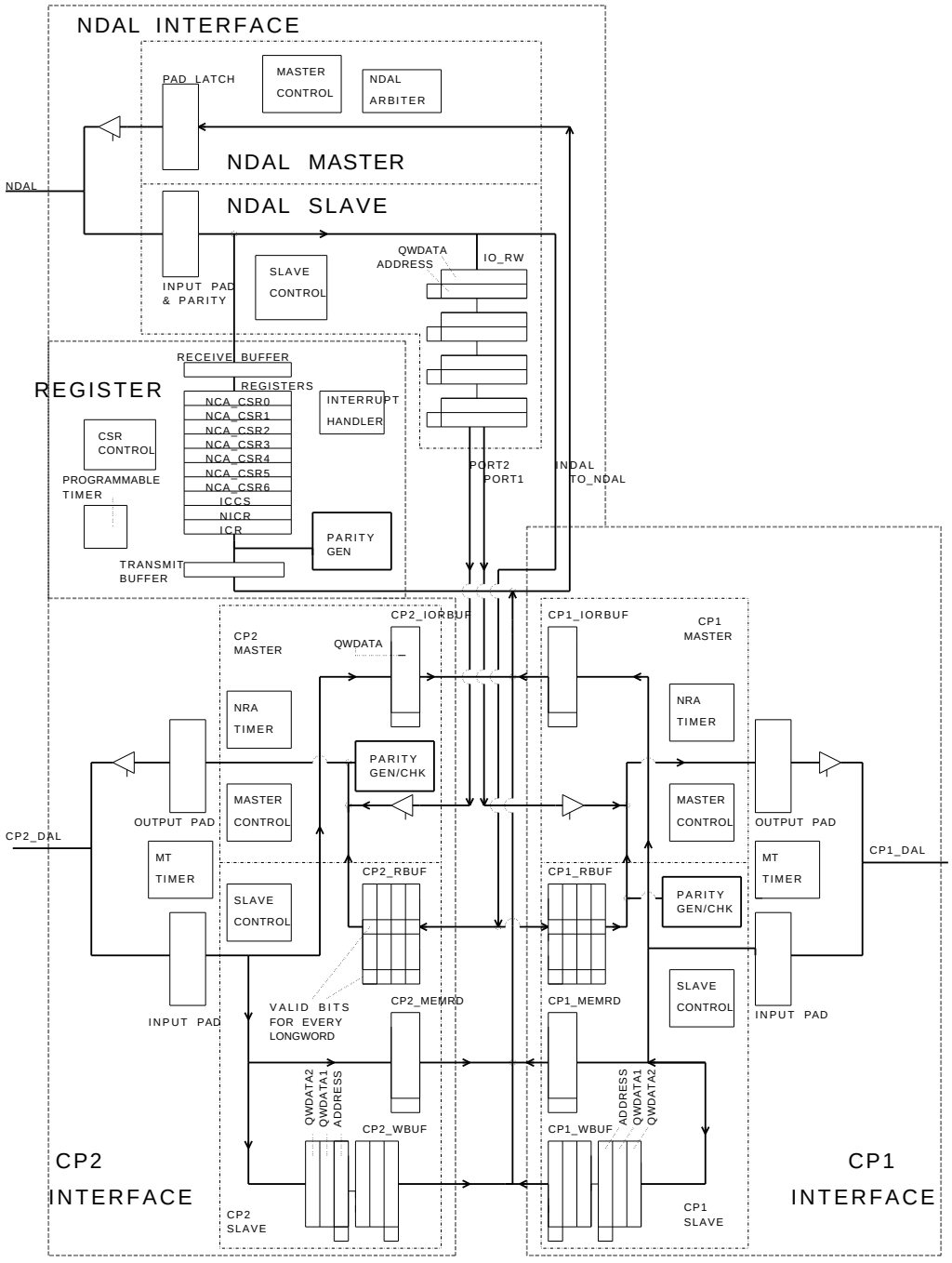
- NDAL interface (Section 6.3.2)
- CP1 interface (Section 6.3.3)
- CP2 interface (Section 6.3.3)
- Registers (Section 6.3.4)

Figure 6–1 shows the organization of the NCA.

The following sections describe each functional block and its interactions, and the NCA addressing. A discussion on error handling is also included at the end of this section.

The NDAL and CP1 and CP2 interface sections are further divided into master and slave subsections. Information transfer between the different sections of the NCA occurs via two buses—TO_NDAL and TO_CDAL. TO_NDAL transfers information to the NDAL interface from both CP-bus interfaces. The TO_CDAL transfers information from the NDAL interface to other sections of the chip. In addition, there are two more buses : TO_CP1 and TO_CP2, which transfer NVAX initiated I/O transactions information to the respective CP-bus interface.

Figure 6–1 NCA Block Diagram



TERMINOLOGY

For the rest of this manual, the following terms are commonly used to refer to NCA ports.

- CP1—means port 1 of the NCA where CP1-bus is connected
- CP2—means port 2 of the NCA where CP2-bus is connected
- CP—means both CP1 and CP2

6.3.1 NCA Addressing

The NCA responds to addresses on the NDAL that access devices on the CP1-bus, CP2-bus, or internal registers in the NCA. These addresses and their destinations are summarized in Table 6–1.

Table 6–1 NCA Addresses

Address Range (hex)	Destination
2000 0000 - 2000 3FFF	CP2 device addresses
2000 4000 - 2000 FFFF	CP1 device addresses
2001 0000 - 20FF FFFF	CP2 device addresses
2100 0000 - 2100 005F	Not Acknowledged
2100 0060 - 2100 0063	SRM ICCS Register
2100 0064 - 2100 0067	SRM NICR Register
2100 0068 - 2100 006B	SRM ICR Register
2100 006C - 2100 00FF	Not Acknowledged
2100 0100 - 2100 010F	Interrupt Vector Read addresses
2100 0110 - 2101 FFFF	Not Acknowledged
2102 0000 - 2102 001F	NCA internal registers
2102 0020 - 2102 FFFF	Not Acknowledged
2103 0000 - 2103 FFFF	CP2 device addresses if IO2_ID_EN is set, else Not Acknowledged
2104 0000 - 27FF FFFF	CP1 device addresses
2800 0000 - 2FFF FFFF	CP2 device addresses
3000 0000 - 3FFF FFFF	CP2 device addresses if IO2_ID_EN is set, else Not Acknowledged

6.3.2 NDAL Interface

The NDAL interface section is made up of the NDAL master and slave interface subsections.

6.3.2.1 NDAL Slave Interface

The NCA's NDAL slave interface continuously monitors the NDAL for transactions addressed to the NCA. Parity is checked on the NDAL<63:0>, CMD_H<3:0>, and ID_H<2:0> for valid parity. If there is no parity error and the transaction is addressed to the NCA, the NCA acknowledges with the assertion of ACK_L. The information is then placed into one of the following queues : IO_RW, CP1_RBUF, or CP2_RBUF. IO_RW queue is described below and the other two buffers are described in their respective sections.

- **IO_RW -** IO_RW is a 4-entry deep queue that holds the transactions addressed to the NCA I/O space ports 1 and 2. Each entry contains quadword address, quadword data (for I/O write only), a valid bit, a port ID (CP1 or CP2), and a transaction token. The IO_RW is a common pool of I/O transactions for both port 1 and port 2. An I/O transaction is stored into the IO_RW queue whenever an entry becomes "empty." The CP1 and CP2 interfaces examine the IO_RW entries to determine if there is a pending transaction for their ports. This is accomplished by the valid bit, the port ID, and the transaction token. Since the IO_RW is 4-entry deep, two bits are maintained as the token. The token of each entry is compared with the expected token of each CP port controllor. When there is a match in any one of the four entries, the port controllor initiates the transaction on the bus. By doing so, CP1 and CP2 are operated independently and the utilization of the queue is optimized. The expected token of CP1 (CP2) is incremented (mod 4) after each transaction.

  When the IO_RW queue has four valid entries (queue is full), the IO1_SUPPRESS_L signal is asserted to the NMC, which asserts the CPU_WB_ONLY to the NVAX CPU. The NVAX CPU is then expected to perform WDISOWN operations only on the NDAL (that is, the NVAX CPU will not drive any transactions on the NDAL other than WDISOWNs). This prevents the NCA IO_RW queue from overflowing.

  Note

  The NCA supports only longword and quadword transactions to the I/O space.

6.3.2.2 NDAL Master Interface

The NDAL master interface is responsible for initiating DMA read and write operations on the NDAL, as well as returning I/O read data to the CPU. The information for all NDAL master transactions comes from the CP1, CP2, and register sections. The following is the list of transactions and the buffers from which the transactions are initiated:

- CP1 DMA write (CP1_WBUF)
- CP1 DMA read (CP1_MEMRD)
- CP1 I/O read data return (CP1_IORBUF)
- CP2 DMA write (CP2_WBUF)
- CP2 DMA read (CP2_MEMRD)
- CP2 I/O read data return (CP2_IORBUF)
- CSR read data return (TRANSMIT BUFFER)

The master control block in the NDAL master interface contains the bus interface and arbitration units. The bus interface unit requests NDAL bus ownership when any of the NDAL master buffers contains valid information, and sequences the transaction when the NCA is granted the ownership of the NDAL.

The arbitration unit follows a fixed priority for initiating transactions on the NDAL when more than one transaction is pending. The CP1 and CSR transactions are arbitrated in the following priority, from highest to lowest :

1. CP1 memory read
2. CP1 memory write
3. CP1 I/O read data return
4. CSR and SRM interval timer read data return

The CP2 transactions are arbitrated in the following priority, from highest to lowest :

1. CP2 memory read
2. CP2 memory write
3. CP2 I/O read data return

Note

The NCA only initiates one transaction for each NDAL bus grant; however, it may retain bus ownership for additional NDAL cycles to return I/O, CSR, or VAX interval timer read data after completing the original transaction. This is to prevent the lower priority transaction from starving.

6.3.3 CP1 and CP2 Interface

Because both the CP1 and CP2 interfaces are copies of each other, the two interfaces will be discussed as CP.

The CP interface section is made up of the master and slave interface subsections.

6.3.3.1 CP Master Interface

The CP master interface is responsible for initiating I/O read and write operations on the CP buses. Address and/or data information for these transactions comes from the IO_RW queue of the NDAL slave section.

CP_IORBUF - CP_IORBUF is a single entry buffer that holds I/O read data returned from the CP bus. It holds up to a quadword of data and has a valid bit. When the buffer is valid, CP_IORBUF sends a request to the NDAL master arbitration unit. When the transaction is granted to the CP_IORBUF, the quadword data, the command bits, the ID bits, and the parity bits are driven onto the NCA's internal TO_NDAL bus and then onto the NDAL when IO1_ GRANT_L is asserted by the NMC.

6.3.3.2 MT and NRA Timers

The CP master interface also contains the "no response abort" (NRA) and master transaction (MT) timers. The NRA timer times out 2 µs (assuming 70 ns CP-bus cycle) after the NCA initiates an I/O transaction on the CP-bus. If the NRA timer times out before the NCA receives a response from an I/O device, the NCA aborts the transaction on the CP-bus (no response abort). The MT timer has 4 settings : 144, 1440, 14400, and 144000 cycles, which translate to 10.08 µs, 100.8 µs, 1.008 ms, and 10.08 ms for 70 ns CP-bus cycle time. During an I/O transaction initiated by the CP master interface to the CP-bus, the timer times out if the master interface does not receive any response.

The purpose of the MT and NRA timers is to prevent either of the CP-buses from becoming "hung" due to hardware or software errors. The NRA timer prevents a CP-bus from hanging due to a software error in which an access to a nonexistent I/O address is attempted. Each of the devices on the CP-bus with the exception of the SSC asserts a "not_me" signal during such accesses. When all the "not_me" signals on the CP-bus are asserted, external logic generates an NRA input to the NCA, to inform it that none of the devices on the corresponding CP-bus have claimed ownership of the issued address. Since the SSC does not implement a "not_me" signal, the NCA will wait 2 µs to give the SSC a chance to respond to the transaction before the NRA timer expires. If the SSC does not complete the transaction within this time, then a no response abort takes place and the I/O transaction is terminated.

The purpose of the MT timer is to prevent either of the CP-buses from becoming "hung" in the situation when a hardware error has occurred. In such cases, it is possible that one of the chips connected to the CP-bus in question could fail to assert its "not_me" signal, indicating that it intends to respond to the transaction. If the chip in question fails to respond to the transaction, presumably due to a hardware failure, the MT timer will expire, thus aborting the I/O transaction and preventing the associated CP-bus from becoming "hung."

6.3.3.3 CP Slave Interface

The CP slave interface responds to DMA transactions on the CP-bus. This subsection features three queues:

- **CP_WBUF -** CP_WBUF is a 2-entry deep queue that holds the DMA write transaction information. Each entry contains one quadword address and two quadword data elements, together with a valid bit. The address element contains information for setting up the NVAX address cycle while the data element contains the write data and a BDATA bit, which is set if bad parity is found in the corresponding quadword of write data on the CP bus. The entry is validated when the DMA address and all the DMA write data have been loaded into CP_WBUF. When the contents of one or more entries is valid, a request is sent to the NDAL arbitration unit. The CP_WBUF entry is invalidated when the write transaction corresponding to this entry is initiated on the NDAL.
- **CP_MEMRD -** On DMA reads, the slave interface places the read address in CP_MEMRD, a quadword address buffer that holds the information for setting up the NDAL address cycle. The valid bit for this buffer is set after ensuring that the DMA read address does not hit in CP_WBUF. If the DMA read address hits in CP_WBUF, the validation of the CP_MEMRD entry is held off until CP_WBUF is flushed. When the valid bit is set, a request is sent to the NDAL arbitration unit.

- **CP_RBUF -** This buffer holds the DMA memory read data that it receives from the NDAL slave interface. This is a hexword data buffer that is organized as longwords (that is, every longword contains a valid bit). The NCA prefetches (programmable) DMA memory read data up to an octaword. As the requested read data is returned to the DMA device on the CP-bus, the associated valid bit is cleared.

As a CP-bus slave, the NCA accepts all transactions addressed in the memory space. The NCA will signal an error to the CP-bus master on DMA transactions addressed in VAX I/O space.

6.3.4 Registers

This section describes the NCA's internal registers. The NCA has seven control and status registers (NCA_CSR) and three interval clock registers. These ten registers are longword-aligned. The NCA supports only I/O write to one register per transaction by the NVAX CPU. However, quadword read by the NVAX CPU of two NCA registers per transaction is allowed.

All the registers are cleared on power-up reset, unless otherwise specified in the following description. Write operations to read-only registers do not cause any errors and are responded to as normal operations; however, the operations do not alter any of the register contents.

The register block also contains a receive buffer and a transmit buffer. The receive buffer is used to store the register address in register read (and data, if register write). The transmit buffer is used to store the read data of a register read transaction while the register block is waiting to return the data to the NDAL. If the I/O address happens to be NCA's CSR or interval timer address, the NCA's CSR control logic sequences the read or write to the appropriate register. On reads, the read data together with the CMD and ID is placed in the transmit buffer and the valid bit is set in the transmit buffer. Once this occurs, the transmit buffer sends a request to the NDAL arbitration unit in the NDAL master interface. When the NCA is granted the mastership of the NDAL, the data, CMD, and ID are driven onto NDAL. On writes, the write data is written into the appropriate register.

Table 6–2 NCA CSR and Interval Timer Registers

Number	Name	Address	No. of Registers
CESR	Error status register	2102 0000	1
CMCDSR	Mode control and diagnostic register	2102 0004	1
CSEAR1	CP1 slave error address register	2102 0008	1
CSEAR2	CP2 slave error address register	2102 000C	1
CIOEAR1	CP1 I/O error address register	2102 0010	1
CIOEAR2	CP2 I/O error address register	2102 0014	1
CNEAR	NDAL error address register	2102 0018	1
ICCS	Interval clock control status register	2100 0060	1
NICR	Next interval count register	2100 0064	1
ICR	Interval count register	2100 0068	1
[1] NCA_ CSR7-10	Interrupt acknowledge registers	2100 0100 - 010F	4

[1]These are virtual registers and they are read-only.

6.3.4.1 Control and Status Registers

The NCA has seven CSRs, the functions of which are decribed below.

6.3.4.2 Error Status Register (CESR)

The NCA reports error information in CESR.

Figure 6–2 Error Status Register

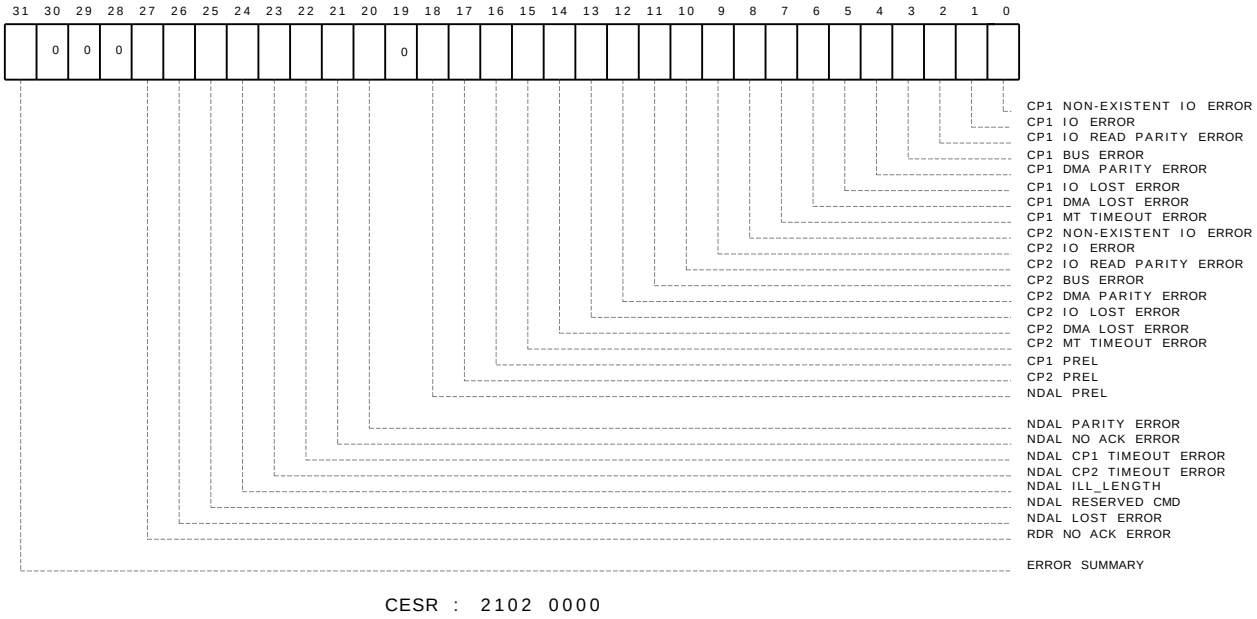


Table 6–3 Error Status Register, CESR

Register Field	Bit	Type, Reset State	Description
Error Summary	31	RO, 0	This bit is "1" when any error is logged by the NCA in this register, with the exception of CESR<25>, which does not affect this bit. It is cleared when all the error bits are cleared.
Unused	30:28	MBZ, 0	These bits are read as 0.
RDR NO_ACK Error	27	WC, 0	This bit is set if NCA does not receive ACK_L when it is initiating an RDRx to the NDAL. No address is logged when this bit is set.
NDAL Lost Error	26	WC, 0	This bit is set if an error condition described in CESR<23:21> occurs and CESR<21> is already set. When this bit is set, CNEAR contains the address of the previous error condition that has not been cleared at the time the current error condition happens. It is possible that this bit is set but all the error bits in CESR<23:21> are cleared because of the asynchronous events. When this happens, the CNEAR may contain an invalid address.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 6–3 (Cont.) Error Status Register, CESR

Register Field	Bit	Type, Reset State	Description
NDAL Reserved cmd	25	WC, 0	This bit is set when the NDAL interface detects a reserved command during any NDAL cycle and this bit is not already set. No address is logged when this bit is set.
NDAL Ill_length	24	WC, 0	This bit is set when the NDAL interface detects an illegal length during an NDAL transaction addressed to the NCA and when this bit is not already set. No address is logged when this bit is set.
NDAL CP2 Timeout Error	23	WC, 0	This bit is set if not all requested read data is returned from the memory within the NDAL cycles specified by the NDAL timer prescaler (CMCDSR<11:10>) for the DMA read transactions initiated on the CP2-bus, and this bit is not already set.
NDAL CP1 Timeout Error	22	WC, 0	This bit is set if not all requested read data is returned from the memory within the NDAL cycles specified by the NDAL timer prescaler (CMCDSR<11:10>) for the DMA read transactions initiated on the CP1-bus, and this bit is not already set.
NDAL NO_ACK Error	21	WC, 0	This bit is set if no ACK_L is received during an NCA-initiated memory transaction and this bit is not already set. CNEAR contains the error address when this bit is set and CMCDSR<23:22> are not set.
NDAL Parity Error	20	WC, 0	This bit is set if NCA detects an NDAL parity error on any NDAL cycles and this bit is not already set. No address is logged when this bit is set.
Unused	19	MBZ, 0	This bit is read as 0.
NDAL PREL	18	WC, 0	This bit is set when there is no XA (that is, CMCDSR<8> is set) present and there is no pending interrupts at both the CP1 and CP2 port at the priority level indicated by the interrupt vector read on the NDAL. No address is logged when this bit is set.
CP2 PREL	17	WC, 0	This bit is set when an interrupt vector read is initiated on the CP2-bus and either CP2_ERR_L is asserted or CP2 master transaction (MT) timer times out. In either case, the NCA returns to the NDAL with RDR and NDAL bit<33,01> set to 1. No address is logged when this bit is set.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 6–3 (Cont.) Error Status Register, CESR

Register Field	Bit	Type, Reset State	Description
CP1 PREL	16	WC, 0	This bit is set when an interrupt vector read is performed on the CP1-bus and either CP1_ERR_L is asserted or CP1 master transaction (MT) timer times out. In either case, the NCA returns to the NDAL with RDR and NDAL bit<33,01> set to 1. No address is logged when this bit is set.
CP2 MT Timeout Error	15	WC, 0	This bit is set if the I/O read or write transaction initiated by the NCA to the CP2-bus causes the CP2 master transaction (MT) timer to time out and this bit is not already set. CIOEAR2 contains the I/O error address when this bit is set and CESR<10:8> are not set.
CP2 DMA Lost Error	14	WC, 0	This bit is set if CESR<12> is already set and either a DMA device initiates an I/O transaction on the CP2-bus or the CP2 interface detects a parity error on a DMA write data. When this bit is set, CSEAR2 contains an address for a previous error that has not been cleared at the time the current error condition happens. It is possible that this bit is set but the error bit in CESR<12> is cleared because of the asynchronous events. When this happens, the CSEAR2 may contain an invalid address.
CP2 IO Lost Error	13	WC, 0	This bit is set if any bit in CESR<15,10:8> is already set and any of the error conditions described in CESR<15,10:8> occurs. When this bit is set, CIOEAR2 contains an address for a previous error that has not been cleared at the time the current error condition happens. It is possible that this bit is set but all the error bits in CESR<15,10:8> are cleared because of the asynchronous events. When this happens, the CIOEAR2 may contain an invalid address.
CP2 DMA Parity Error	12	WC, 0	This bit is set if the NCA detects a parity error in the DMA write data on the CP2-bus. CSEAR2 contains the DMA write address when this bit is set.
CP2 Bus Error	11	WC, 0	This bit is set if a DMA device on the CP2-bus initiates an I/O transaction or a transaction with one of the reserved commands. No address is logged when this bit is set.
CP2 IO Read Parity Error	10	WC, 0	This bit is set if the NCA detects a parity error in the I/O read data on the CP2-bus. CIOEAR2 contains the I/O error address when this bit is set and CESR<15,9:8> are not set.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 6–3 (Cont.) Error Status Register, CESR

Register Field	Bit	Type, Reset State	Description
CP2 IO Error	9	WC, 0	This bit is set if an NCA-initiated transaction, read or write, on the CP2-bus results in CP2_ ERR_L assertion. CIOEAR2 contains the I/O error address when this bit is set and CESR<15,10,8> are not set.
CP2 Nonexistent IO Error	8	WC, 0	This bit is set if CP2_NRA timer times out and CP2_NRA pin is asserted on an NCA-initiated I/O transaction on the CP2-bus. CIOEAR2 contains the I/O error address when this bit is set and CESR<15,10:9> are not set.
CP1 MT Timeout Error	7	WC, 0	This bit is set if the I/O read or write transaction initiated by the NCA to the CP1-bus causes the CP1 master transaction (MT) timer to times out and this bit is not already set. CIOEAR1 contains the I/O error address when this bit is set and CESR<2:0> are not set.
CP1 DMA Lost Error	6	WC, 0	This bit is set if CESR<4> is already set and either a DMA device initiates an I/O transaction on the CP1-bus or the CP1 interface detects a parity error on a DMA write data. When this bit is set, CSEAR1 contains an address for a previous error that has not been cleared at the time the current error condition happens. It is possible that this bit is set but the error bit in CESR<4> is cleared because of the asynchronous events. When this happens, the CSEAR1 may contain an invalid address.
CP1 IO Lost Error	5	WC, 0	This bit is set if any bit in CESR<7,2:0> is already set and any of the error conditions described in CESR<7,2:0> occurs. When this bit is set, CIOEAR1 contains an address for a previous error that has not been cleared at the time the current error condition happens. It is possible that this bit is set but all the error bits in CESR<7,2:0> are cleared because of the asynchronous events. When this happens, the CIOEAR1 may contain an invalid address.
CP1 DMA Parity Error	4	WC, 0	This bit is set if the NCA detects a parity error in the DMA write data on the CP1-bus. CSEAR1 contains the DMA write address when this bit is set.
CP1 Bus Error	3	WC, 0	This bit is set if a DMA device on the CP1-bus initiates an I/O transaction or a transaction with one of the reserved commands. No address is logged when this bit is set.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 6–3 (Cont.) Error Status Register, CESR

Register Field	Bit	Type, Reset State	Description
CP1 IO Read Parity Error	2	WC, 0	This bit is set if the NCA detects a parity error in the I/O read data on the CP1-bus. CIOEAR1 contains the I/O error address when this bit is set and CESR<7,1:0> are not set.
CP1 IO Error	1	WC, 0	This bit is set if an NCA-initiated transaction, read or write, on the CP1-bus results in CP1_ ERR_L assertion. CIOEAR1 contains the I/O error address when this bit is set and CESR<7,2,0> are not set.
CP1 Nonexistent IO Error	0	WC, 0	This bit is set if CP1_NRA timer times out and CP1_NRA pin is asserted on an NCA-initiated I/O transaction on the CP1-bus. CIOEAR1 contains the I/O error address when this bit is set and CESR<7,2:1> are not set.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

6.3.4.3 Mode Control and Diagnostic Register (CMCDSR)

The NCA operating modes are controlled by information in this register. In addition, this register is used for storing diagnostic status information. The following is a description of all the bit fields in the register.

Figure 6–3 Mode Control and Diagnostic Status Register

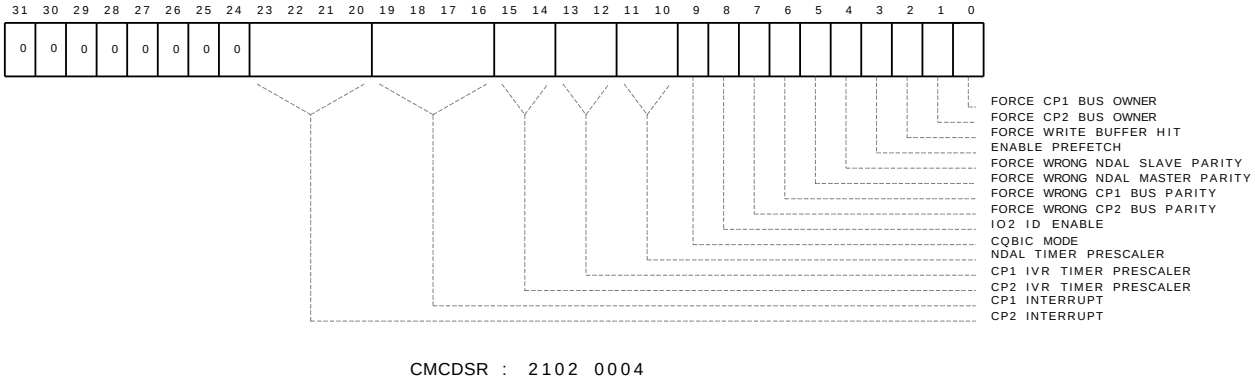


Table 6–4 Mode Control and Diagnostic Status Register, CMCDSR

Register Field	Bit	Type, Reset State	Description
Unused	31:24	MBZ, 0	This field reads as 0.
CP2 Pending Interrupt	23:20	RO, 0	This 4-bit field shows the presence of interrupts at IPL 17, 16, 15, and 14, respectively, on the CP2-bus.
CP1 Pending Interrupt	19:16	RO, 0	This 4-bit field shows the presence of interrupts at IPL 17, 16, 15, and 14, respectively, on the CP1-bus.
CP2 MT Timer Prescaler	15:14	RW, 11	This 2-bit field is used as a prescaler to the CP2 master transaction timer. The CP2 MT timer has 4 settings: 00 (binary) = 144 cycles, 01 (binary) = 1440 cycles, 10 (binary) = 14400 cycles, and 11 (binary) = 144000 cycles.
CP1 MT Timer Prescaler	13:12	RW, 11	This 2-bit field is used as a prescaler to the CP1 master transaction timer. The CP2 MT timer has 4 settings: 00 (binary) = 144 cycles, 01 (binary) = 1440 cycles, 10 (binary) = 14400 cycles, and 11 (binary) = 144000 cycles.
NDAL Timeout Prescaler	11:10	RW, 0	This 2-bit field is used as a prescaler for the NDAL transaction pending timers. These timers should always time out after the NMC pending timers time out. The NDAL timer prescaler has 4 settings: 00 (binary) = 3200 cycles, 01 (binary) = 2000 cycles, 10 (binary) = 1000 cycles, and 11 (binary) = 500 cycles.
CQBIC Mode	9	RW, 0	When set, this bit indicates that CQBIC (Q–bus adapter) is present on the CP2-bus.
IO2 ID Enable	8	RW, 0	When set, this bit enables the NCA to use IO2 ID when initiating CP2 DMA transactions on the NDAL. This bit should be set if there is no XA in the NVAX system.
Force Wrong CP2 Bus Parity	7	RW, 0	When set to 1, this bit causes the CP2 port to generate wrong data parity on the incoming and outgoing data. The NCA clears this bit after it initiates or responds to one CP2-bus transaction. This bit is for test purposes only and should not be set during normal operation.
Force Wrong CP1 Bus Parity	6	RW, 0	When set to 1, this bit causes the CP1 port to generate wrong data parity on the incoming and outgoing data. The NCA clears this bit after it initiates or responds to one CP-bus transaction. This bit is for test purposes only and should not be set during normal operation.

RO—Read-only
RW—Read/write
WO—Write-only, read as 0
WC—Write one-to-clear
MBZ—Read-only, read as 0

(continued on next page)

Table 6–4 (Cont.) Mode Control and Diagnostic Status Register, CMCDSR

Register Field	Bit	Type, Reset State	Description
Force Wrong NDAL Master Parity	5	RW, 0	When set to 1, this bit forces the NCA NDAL master parity generator to generate incorrect parity in the command field in the address cycle (or data cycle if read data return) of the next NCA-initiated NDAL transaction. This bit is self-cleared by the NCA after it has initiated one NDAL transaction. This bit is for test purposes only and should not be set during normal system operation.
Force Wrong NDAL Slave Parity	4	RW, 0	When set to 1, this bit forces the NDAL slave parity generator to generate incorrect parity on any NDAL cycle. This emulates parity errors in all PARITY_H<2:0>. The NCA asserts S_ERR_L and sets CESR<NDAL_PARITY_ERROR> every cycle as long as this bit is set. This bit is self-cleared by the NCA after it has responded to one valid NDAL transaction addressed to the NCA other than a read data return. This bit is for test purposes only and should not be set during normal system operation.
Enable Prefetch	3	RW, 0	When set to 1, this bit enables the CP1 and CP2 ports to perform data prefetching on DMA read transactions. This is set to 1 during powerup when CP_RESET_L is asserted.
Force Write Buffer Hit	2	RW, 0	When set to 1, this bit forces all DMA read addresses from CP1 and CP2 ports to hit any valid element in the CP1_WBUF and CP2_WBUF queues, respectively. This bit should be set for diagnostic purposes only.
Force CP2 Bus Owner	1	RW, 1	When set to 1, the NCA asserts CP2_DMR_L to become the CP2-bus master regardless of whether there is an I/O transaction pending in the NCA or not. This bit is set to 1 during powerup when CP_RESET_L is asserted, and must be cleared by software to allow normal I/O transactions to take place.
Force CP1 Bus Owner	0	RW, 1	When set to 1, the NCA asserts CP1_DMR_L to become the CP1-bus master regardless of whether there is an I/O transaction pending in the NCA or not. This bit is set to 1 during powerup when CP_RESET_L is asserted, and must be cleared by software to allow normal I/O transactions to take place.

RO—Read-only
RW—Read/write
WO—Write-only, read as 0
WC—Write one-to-clear
MBZ—Read-only, read as 0

6.3.4.4 CP1 Slave Error Address Register (CSEAR1)

When either CESR<3> or CESR<4> is set because of a CP1-bus slave error condition, the corresponding octaword address is logged in this register. The content remains valid until both error bits are cleared. This register is read-only and contains valid information only when either of CESR<4:3> is set.

Figure 6–4 CP1 Slave Error Address Register

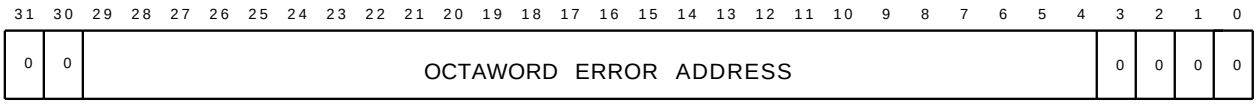


Table 6–5 CP1 Slave Error Address Register, CSEAR1

Register Field	Bit	Type, Reset State	Description
Unused	31:30	MBZ, 0	These bits read as 0.
Error Address	29:4	RO, 0	These bits contain the CP1 slave octaword address.
Unused	3:0	MBZ, 0	These bits read as 0.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

6.3.4.5 CP2 Slave Error Address Register (CSEAR2)

When either CESR<10> or CESR<11> is set because of a CP2-bus slave error condition, the corresponding octaword address is logged in this register. The content remains valid until both error bits are cleared. This register is read-only and contains valid information only when either of CESR<11:10> is set.

Figure 6–5 CP2 Slave Error Address Register

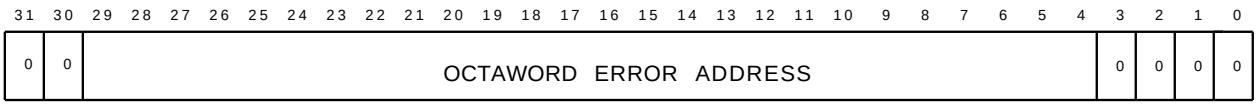


Table 6–6 CP2 Slave Error Address Register, CSEAR2

Register Field	Bit	Type, Reset State	Description
Unused	31:30	MBZ, 0	These bits read as 0.
Error Address	29:4	RO, 0	These bits contain the CP2 slave octaword address.
Unused	3:0	MBZ, 0	These bits read as 0.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

6.3.4.6 CP1 IO Error Address Register (CIOEAR1)

When any of CESR<7, 2:0> is set because of an error condition in an NCA-initiated I/O transaction on the CP-bus, the corresponding address and ID are logged in this register. The content remains valid until all CESR<7, 2:0> bits are cleared. This register is read-only and contains valid information only when any of the error bits is set.

Figure 6–6 CP1 IO Error Address Registers

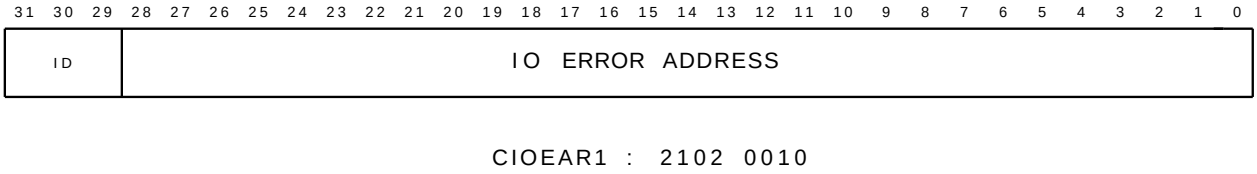


Table 6–7 CP1 IO Error Address Register, CIOEAR1

Register Field	Bit	Type, Reset State	Description
Commander ID	31:29	RO, 0	These bits contain the commander ID corresponding to the I/O transaction that resulted in the error condition.
Error Address	28:0	RO, 0	These bits contain the address corresponding to the I/O transaction that resulted in the error condition.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0
MBO—Read-only, read as 1

6.3.4.7 CP2 IO Error Address Register (CIOEAR2)

When any of CESR<7, 10:8> is set because of an error condition in an NCA-initiated I/O transaction on the CP2-bus, the corresponding address and ID are logged in this register. The content remains valid until all CESR<7, 10:8> bits are cleared. This register is read-only and contains valid information only when any of the error bits is set.

Figure 6–7 CP2 IO Error Address Registers

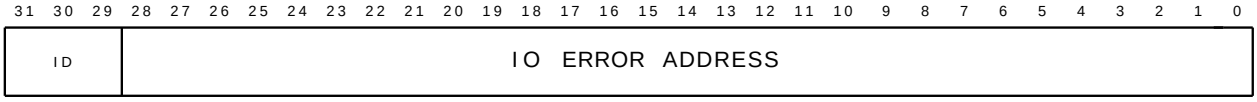


Table 6–8 CP2 IO Error Address Register, CIOEAR2

Register Field	Bit	Type, Reset State	Description
Commander ID	31:29	RO, 0	These bits contain the commander ID corresponding to the I/O transaction that resulted in the error condition.
Error Address	28:0	RO, 0	These bits contain the address corresponding to the I/O transaction that resulted in the error condition.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0
MBO—Read-only, read as 1

6.3.4.8 NDAL Error Address Register (CNEAR)

When any of CESR<23:21> is set because of an error condition in an NCA-initiated NDAL transaction, the corresponding address and ID are logged in this register. The content remains valid until all CESR<23:21> bits are cleared. This register is read-only and contains valid information only when any of the error bits is set.

Figure 6–8 NDAL Error Address Registers

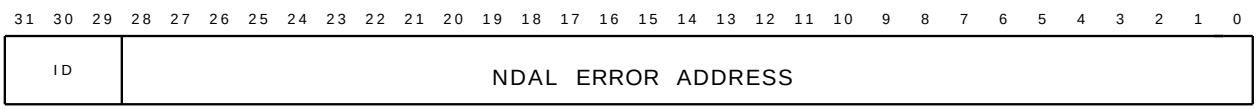


Table 6–9 NDAL Error Address Register, CNEAR

Register Field	Bit	Type, Reset State	Description
Commander ID	31:29	RO, 0	These bits contain the commander ID corresponding to NCA-initiated NDAL transaction that resulted in the error condition.
Error Address	28:0	RO, 0	These bits contain the address corresponding to NCA-initiated NDAL transaction that resulted in the error condition.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

6.4 Interval Clock Registers

The interval clock is used for accounting, for time-dependent events, and to maintain the software date and time. The NVAX CPU implements the interrupt at IPL 22 programmed interval only (that is, the interval timer increments at 1 µs intervals). The interval timer consists of three registers and a counter. For information on the interval clock, refer to the *VAX Architecture Reference Manual*.

6.4.1 Interval Clock Control and Status Register (ICCS)

The ICCS is a 32-bit register containing the control and status information for the interval timer. Figure 6–9 shows the format of the ICCS register, and Table 6–10 lists the bit decriptions.

Figure 6–9 Interval Clock Control and Status Register

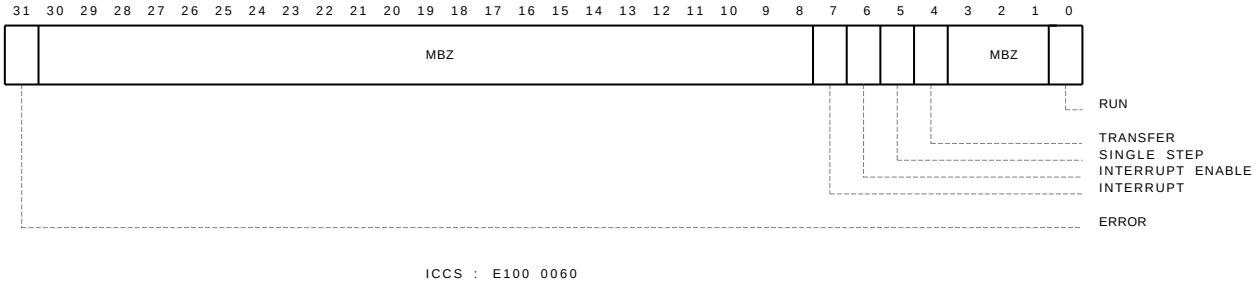


Table 6–10 Interval Clock Control and Status Register, ICCS

Register Field	Bit	Type, Reset State	Description
Error	31	RW, 0	This bit is set when ICR overflows and if interrupt is already set. This bit indicates an unacknowledgement from the CPU. This bit is write one-to-clear.
Unused	30:8	RO, 0	These bits read as 0.
Interrupt	7	RW, 0	This bit is set when the ICR (Section 6.4.3) overflows. This bit is write one-to-clear.
Interrupt Enable	6	RW, 0	When set, an interrupt request is generated every time interrupt is set. When clear, no interrupt is requested. Similarly, if interrupt is already set and interrupt enable is set, an interrupt request is generated. This bit is cleared on powerup.
Single Step	5	RW, 0	When run is cleared, each time this bit is set, the ICR is incremented by 1. This bit should not be set when run is set, otherwise the consequence is unpredictable. This bit is cleared at powerup.
Transfer	4	RW, 0	When set, the content of NICR (Section 6.4.2) is transferred to ICR. This bit is write-only. This bit should not be set when run is set; otherwise, the consequence is unpredictable.
Unused	3:1	RO, 0	These bits read as 0.
Run	0	RW, 0	When set, ICR increments every 1 µs. This bit is cleared on powerup.

RO—Read-only
RW—Read/write
WC—Write one-to-clear
MBZ—Read-only, read as 0

6.4.2 Next Interval Count Register (NICR)

The NICR is a 32-bit register holding the initial count value for the ICR register. When ICR overflows, the contents of NICR are loaded into ICR. This is a read/write register. This register is set to the value FFFFD8F0 (hex) (10 ms) upon powerup.

6.4.3 Interval Count Register (ICR)

The ICR is a 32-bit register holding the current count of the interval timer. The content of ICR is incremented by 1 every 1 µs when ICCS<0> is set, or every time ICCS<5> is set. When ICR overflows or when ICCS<4> is set , the content of NICR is loaded into this register. This register is read-only. This register is set to the value FFFFD8F0 (hex) (10 ms) upon powerup.

6.5 NCA Transaction Handling

This section describes the NCA behavior by tracing the flow of transactions through the functional units of the chip. There are seven types of transactions supported by the NCA:

- IO write
- IO read

- Interrupt vector read
- Register read
- Register write
- DMA read
- DMA write

6.5.1 IO Write

IO write transactions are initiated by the NVAX CPU on the NDAL. IO write can be WRITE or WDISOWN operations to addresses assigned to the CP1 or CP2 ports (Table 6–1). WDISOWN is treated as WRITEs. The data length for IO writes is always quadword, but could be masked on longword alignment. A quadword write is performed as two longword writes on the CP-bus in two separate CP-bus grants.

The address on the NDAL is latched and decoded by the NDAL slave interface to determine if the NVAX CPU is addressing the CP1 or CP2 ports. If it is an address to one of the ports, and there is no parity error, the NCA acknowledges the NDAL transaction and begins to process it. The address and the port ID are latched in the IO_RW queue. The data is latched from the NDAL during the data cycle that follows. Again, parity is calculated and checked. If there is no parity error on the data, then the data is latched in the IO_RW queue and the valid bit for the entry is set. If a parity error is detected, then a soft error interrupt will be requested, the transaction is ignored, and the valid bit of the entry of the IO_RW queue is not set.

If the NDAL address is not within the NCA address space, it is ignored by the NCA. If it is within the NCA address space but a parity error is detected, or if the address is outside the NCA address space and a parity error is detected, then ACK_L is not asserted. Instead, a soft error interrupt is requested to notify the NVAX CPU of the error. In addition, the NDAL parity error bit is set in CMCDSR.

If the IO write is for port 1 (port 2), then CP1_RBUF (CP2_RBUF) is invalidated. Depending on the byte masks of the IO write, the transaction will require either one or two longword writes on the associated CP-bus. The CP1_IORBUF (CP2_IORBUF) is flushed before the IO write can proceed on CP1 (CP2) port; that is, any previous IO read must be forwarded to the NDAL interface before the IO write can proceed. After CP1_IORBUF (CP2_IORBUF) has been flushed, the NCA initiates the transaction on the CP1 (CP2) bus.

The lower longword of the quadword is initiated first on the CP-bus, unless the byte masks corresponding to the lower longword are all 0s. Then if the byte masks corresponding to the upper longword are not all 0s, the IO write for the upper longword is initiated on the CP-bus. If the byte masks corresponding to the lower longword are all 0s, then the IO write for the upper longword is initiated on the CP-bus regardless of the value of the byte masks for the upper longword.

6.5.2 IO Read

IO read transactions are initiated by the NVAX CPU on the NDAL. IO reads can be caused by NDAL IREAD, DREAD, or OREAD operations to addresses assigned to the CP1 and CP2 ports (Table 6–1). As in IO writes, the data length is always a quadword, and may be masked. OREADs are treated as DREAD operations.

IO reads are handled the same way as IO writes on the NDAL slave interface, except that no data is latched in the IO_RW queue. If the address is within the NCA address space, then the valid bit for the entry is set. A quadword read is performed as two longword reads on the CP-bus in two separate CP bus grants.

If the IO read is for port 1 (port 2), then CP1_RBUF (CP2_RBUF) is invalidated. Depending on the byte masks of the IO read, the transaction can either be one or two longword reads. The CP1_IORBUF (CP2_IORBUF) is flushed before the IO read can proceed on CP1 (CP2) port; that is, any previous IO read must be forwarded to the NDAL interface before the next IO read can proceed. When CP1_IORBUF (CP2_IORBUF) has been flushed, the NCA initiates the transaction on the CP1 (CP2) bus. An IREAD will result in an I-stream read, and an OREAD or DREAD will result in a D-stream read on the CP1 (CP2) bus.

The lower longword of the quadword is initiated first on the CP-bus, unless the byte masks for this longword are all 0s. In this case, the upper longword read happens first. If the byte masks for the lower longword are all 0s, then the upper longword is initiated on the CP-bus regardless of the value of the upper longword byte masks.

As the read data is returned on the CP1 (CP2) bus, it is stored in the CP1_ IORBUF (CP2_IORBUF) queue in the CP1 (CP2) port. When all the read data has been returned, a request is sent to the NCA's arbitration unit of the NDAL master interface.

The NCA NDAL master interface arbitrates with the other NDAL devices for mastership of the NDAL. When NDAL bus ownership has been granted, the NCA initiates a read data return cycle (RDR) and drives the data from the CP1_ IORBUF (CP2_IORBUF) onto the NDAL to complete the read.

The RDR number is determined by the NDAL address of the IO read. Table 6–11 shows how the RDR number is calculated.

Table 6–11 IO Read RDR number

NDAL<4:3>	RDR Number	Comments
00 (binary)	RDR0	Quadword 0 of a hexaword
01 (binary)	RDR1	Quadword 1 of a hexaword
10 (binary)	RDR2	Quadword 2 of a hexaword
11 (binary)	RDR3	Quadword 3 of a hexaword

6.5.3 Interrupt Vector Read

An interrupt vector read appears on the NDAL as a DREAD, IREAD, or OREAD to one of four longwords within the range E100 0100 to E100 010F (hex). The address must be longword-aligned. Interrupt vector reads are initiated by the NVAX CPU in response to an assertion of one of the NVAX CPU's interrupt lines (IRQ<3:0>). The read address identifies the level of the interrupt being serviced. The read data length is always a quadword, which is masked to a single longword. The NCA expects the NVAX CPU to provide the proper byte mask information; that is, the byte mask should either be 03 (hex) or 30 (hex).

IRQ<3:0>can be asserted only by the NCA, which asserts these signals on behalf of devices on either of the two CP-buses.

The interrupt vector read cycle starts in the NCA when the NDAL slave interface detects a read address cycle on the NDAL. The address is latched and decoded to determine if it is an interrupt vector address. If it is an interrupt vector address, and there is no parity error, the NCA acknowledges the NDAL transaction. The address is then latched in the IO_RW queue.

The address decoder determines the destination of the read. If one of the CP ports has an interrupt of the proper level pending, then the read is directed to that port. If both ports have interrupts pending at that level, then the CP2 port will receive the interrupt vector read. If neither port has an interrupt, then the NCA will abort the vector read transaction.

If the interrupt vector read is directed to either of the two CP ports, the ID of that port is latched in the IO_RW queue together with the address, and the valid bit of the entry is set.

If the IO read is for port 1 (port 2), then CP1_RBUF (CP2_RBUF) is invalidated. The CP1_IORBUF (CP2_IORBUF) is also flushed before the read can proceed on CP1 (CP2) port. This is done because any previous IO read must be forwarded to the NDAL interface before the interrupt vector read can proceed. When CP1_ IORBUF (CP2_IORBUF) is flushed, the NCA initiates the transaction on the CP1 (CP2) bus.

As the read data is returned on the CP1 (CP2) bus, it is stored in the CPx_ IORBUF queue in the CP1 (CP2) port. A request is sent to the arbitration unit of the NDAL master interface.

The NCA NDAL master interface arbitrates for the NDAL mastership after the request is received. When NDAL bus ownership is granted, the NCA initiates a read data return (RDR) cycle and drives the data from the CP1_IORBUF (CP2_ IORBUF) onto the NDAL to complete the read.

The RDR number for interrupt vector read is determined in Table 6–12.

Table 6–12 Interrupt Vector Read RDR Number

NDAL<31:0>	RDR Number
2100 0000 (IPL 14)	RDR0
2100 0100 (IPL 15)	RDR0
2100 1000 (IPL 16)	RDR1
2100 1100 (IPL 17)	RDR1

NOTE

If the NCA initiates an I/O transaction (IO reads, IO writes, or interrupt vector reads) and the CP master transaction timer times out, the NCA will terminate the transaction. No error will be logged.

6.5.4 Register Read

Register read can be a read to NCA's CSR registers or to the VAX standard interval timer registers. Register read transactions are initiated by the NVAX CPU on the NDAL. They appear as DREAD, IREAD, or OREAD operations, and the data length is always a quadword (1 data transfer).

The read transaction starts in the NCA when the NDAL slave interface detects a read address cycle on the NDAL. The address is latched on the NCA, and a decoder determines if it is the NCA's register address. If there is a match, and there is no parity error, the NCA acknowledges the NDAL cycle.

The NCA address decoder determines that the destination of the read is the register block. The address is then latched in the receive buffer. The NCA always returns the register data corresponding to the quadword address regardless of the byte mask value. The quadword data is stored in the transmit buffer. When the read data is ready, a request is forwarded to the arbitration unit of the NCA's NDAL master interface.

The NCA then arbitrates with the other NDAL devices for control of the NDAL. When it becomes bus master, it initiates a read data return cycle and drives the data onto the NDAL to complete the read.

6.5.5 Register Write

Register write transactions are initiated by the NVAX CPU on the NDAL. They appear as either WRITE or WDISOWN operations and the write data length is always a quadword, but only one longword data is written in the appropriate register at a time.

The register write transaction starts in the NCA when the NDAL slave interface detects a write address cycle on the NDAL. The address is latched by the NCA, and a decoder determines if it is the NCA's register address. If there is a match, and there is no parity error, the NCA acknowledges the NDAL transaction.

The NCA address decoder determines if the destination of the write is the register block. The address and the write data are then latched in the receive buffer. The address indicates the quadword boundary while the mask field of the lower longword selects the proper longword. If the mask field is not all 0s, then the write data on NDAL<31:0> is written into the NCA register on the lower

longword address. If the mask field is all 0s, then the data on NDAL<63:32> is written into the NCA register on the upper longword address.

6.5.6 CP1 DMA Read

CP1 DMA reads are initiated by I/O devices that reside on the CP1-bus. DMA reads are supported by the NCA to memory address space only. All DMA reads to VAX I/O address space are terminated by the NCA by asserting the CP1_ERR_L signal on the CP1-bus. DMA reads can be longword (2 words), quadword (4 words), hexword (6 words), or octaword (8 words) on the CP1-bus.

When the NCA's prefetch enable bit is set, the NCA will respond to DMA ready cycles to main memory by prefetching sequential locations in anticipation of future requests to these memory locations by DMA devices. Table 6–13 shows the prefetch scheme the NCA uses.

Table 6–13 CP1 DMA Memory Read Prefetching

CP1 Read Data Length	Data Length Requested
Longword	Quadword
Quadword	Octaword
Hexword	Octaword
Octaword	Hexaword

When a DMA read happens on the CP1-bus, the address is latched in the CP1_MEMRD buffer. The NCA compares the address with the previous DMA read address. If they are within the same hexaword boundary, then the current requested read data might be in the CP1_RBUF. The NCA then checks if all the requested longwords are in the CP1_RBUF. If this is also true, then the NCA will not forward the DMA read request to the NDAL master interface. Instead, the CP1 slave interface returns the read data onto the CP1-bus directly from the CP1_RBUF. As each longword is driven onto the CP1-bus, the valid bit of the corresponding longword is cleared.

If the DMA read address is not within the same hexaword boundary of the previous DMA read, or if not all the requested read data is in the CP1_RBUF, then the CP1 slave interface will forward the read request to the NDAL master interface and all entries in the CP1_RBUF will be invalidated.

It is possible for the NCA to receive a new DMA read transaction on the CP1-bus before all the prefetch data of the previous DMA read has been received. If the current DMA read is within the same hexaword as the previous read but not all the requested longword data is in the CP1_RBUF, then the CP1 slave interface will wait until all the prefetch is completed. Once all the prefetched data has returned, the read data will return from the CP1_RBUF, thus avoiding the need to forward the DMA read request to the NDAL. If the current hexaword DMA read address does not match the previous one, the DMA read request will be forwarded to the NDAL master interface, since the data in CP1_RBUF is for a different hexaword address.

It is possible that a DMA read address matches a DMA write transaction waiting in the CP1_WBUF. If the addresses are within the same hexaword boundary, then the DMA read request is stalled until the DMA write has completed on the NDAL.

If the DMA read is a lock operation, then any outstanding DMA writes waiting in CP1_WBUF are flushed before the read request is sent to the NDAL master interface.

The NCA gives higher priority to DMA reads than writes if both are pending in the CP1 slave interface so that the DMA read operations will not hold up the CP1-bus (DMA writes are dump-and-run).

When the NDAL master interface receives a DMA request from the CP1 slave interface, the NCA arbitrates for the NDAL on behalf of the CP1-bus. When NDAL mastership is granted to the NCA for this transaction, the DMA read address is forwarded to the NDAL interface and driven onto the NDAL. Some time later, data is returned from the memory. Each quadword of data is latched in the NDAL interface and in CP1_RBUF. According to NDAL protocol, the first quadword returned on the NDAL is guaranteed to be the requested quadword.

As the data becomes available in CP1_RBUF, it is driven onto the CP1-bus. As each longword of data is returned on the CP1-bus, the corresponding entry is invalidated in CP1_RBUF. When all the requested data has been returned, the transaction completes on the CP1-bus. The NCA is now ready to receive new DMA transactions on the CP1-bus. The nonrequested data is kept in CP1_RBUF as prefetch data.

The prefetch data in the CP1_RBUF is invalidated under the following conditions:

- DMA memory write or write-unlock on the CP1-bus to any memory address
- DMA memory read with different hexaword address or the prefetch data does not contain all the requested longword data
- DMA memory read-lock on the CP1-bus to any memory address
- I/O transaction on the CP1-bus initiated by the NCA

6.5.7 CP1 DMA Write

CP1 DMA writes are initiated by I/O devices that reside on the CP1-bus. DMA writes are supported by the NCA only to VAX memory space; that is, NOT to VAX I/O space. All DMA writes to VAX I/O space are terminated by the NCA with the assertion of the CP1-bus signal CP1_ERR_L. DMA write can be longword, quadword, hexword, or octaword on the CP1-bus. Longword and quadword writes are performed as quadword writes on the NDAL, and hexword and octaword writes are performed as octaword writes on the NDAL. Writes can be masked or unmasked.

When a DMA write happens on the CP1-bus, the address and data are latched in the CP1_WBUF. A DMA write on the CP1-bus causes the CP1_RBUF to be invalidated. When the address and write data are ready, the NCA will arbitrate for the NDAL on behalf of the CP1-bus. Once the NDAL has been granted to the NCA, the write is initiated on the NDAL and the transaction completes.

6.5.8 CP2 DMA Read

CP2 DMA reads are initiated by I/O devices that reside on the CP2-bus. DMA reads are supported by the NCA to VAX memory space only. All DMA reads to VAX I/O space are terminated by the NCA by asserting the CP2-bus signal CP2_ERR_L. The NCA supports DMA reads originating on the CP2-bus of longword, quadword, hexword, or octaword length.

When NCA prefetching is enabled, the NCA will prefetch read data by requesting more data from the system main memory than originally requested on the CP2-bus. Table 6–14 shows the prefetch scheme the NCA uses.

Table 6–14 CP2 DMA Memory Read Prefetching

CP2 Read Data Length	Data Length Requested on NDAL
Longword	Quadword
Quadword	Octaword
Hexword	Octaword
Octaword	Hexaword

When a DMA read happens on the CP2-bus, the address is latched in the CP2_MEMRD buffer. The NCA compares the address with the previous DMA read address. If they are within the same hexaword boundary, then the current requested read data might be in the CP2_RBUF. The NCA then checks whether all the requested longwords are in the CP2_RBUF. If this is also true, then the NCA will not forward the DMA read request to the NDAL master interface. Instead, the CP2 slave interface returns the read data onto the CP2-bus directly from the CP2_RBUF. As each longword is driven onto the CP2-bus, the valid bit of the corresponding longword is cleared.

If the DMA read address is not within the same hexaword boundary of the previous DMA read or not all the requested read data is in the CP2_RBUF, then the CP2 slave interface forwards the read request to the NDAL master interface, and all entries in the CP2_RBUF are invalidated.

It is possible for the NCA to receive a new DMA read transaction on the CP2-bus when not all the prefetch data of the previous DMA read has been received. If the current DMA read is within the same hexaword as the previous read but not all the requested longword data is in the CP2_RBUF, then the CP2 slave interface will wait until all the prefetch is completed. Once all the prefetched data has returned, the read data will return from the CP2_RBUF, thus avoiding the need to forward the DMA read request to the NDAL. If the current hexaword DMA read address does not match the previous one, the DMA read request will be forwarded to the NDAL master interface, since the data in CP2_RBUF is for a different hexaword address.

It is possible that a DMA read address matches a DMA write transaction waiting in the CP2_WBUF. If the addresses are within the same hexaword boundary, then the DMA read request is stalled until the DMA write has completed on the NDAL.

If the DMA read is a lock operation, then any outstanding DMA writes waiting in CP2_WBUF are flushed before the read request is sent to the NDAL master interface.

The NCA gives higher priority to DMA reads than writes if both are pending in the CP2 slave interface so that the DMA read operations will not hold up the CP2-bus (DMA writes are dump-and-run).

The CQBIC is the only DMA device on the CP2-bus. Therefore, when a DMA longword read occurs on the CP2-bus as the transaction is forwarded to the NDAL, the NCA asserts the QBUS_TRANS_L signal to the NDAL arbiter in the NMC. The assertion of this signal causes a modification of the NDAL arbitration

priority such that the CP2-bus (CQBIC) has the highest priority on the NDAL. This is done to reduce the latency on memory reads by Q22–bus devices.

Once asserted, the NCA will continue to assert QBUS_TRANS_L for TBD cycles after the read data is returned from the NMC on the NDAL, or until any of the following conditions happen on the CP2-bus:

1. The longword memory read is followed by a memory write.
2. The longword memory read is followed by a longword memory read.
3. The longword memory read is followed by a quadword memory read.

In case 1, QBUS_TRANS_L is deasserted after the memory write completes on the CP2-bus. In case 2, QBUS_TRANS_L is kept asserted for another TBD cycle after the read data for the second longword memory read is returned from the NMC. In case 3, QBUS_TRANS_L is deasserted when the quadword memory read address and command are pushed onto the NDAL.

When the NDAL master interface receives a DMA request from the CP2 slave interface, a bus request is asserted on the NDAL. When the bus grant is received, the DMA read address is forwarded to the NDAL interface and driven onto the NDAL. Later, data is returned from the memory. Each quadword of data is latched in the NDAL interface and then CP2_RBUF. According to NDAL protocol, the first quadword returned on the NDAL is guaranteed to be the requested quadword.

As the data becomes available in CP2_RBUF, it is driven onto the CP2-bus. As each requested longword is driven onto the CP2-bus, the corresponding CP2_ RBUF entry is invalidated. When all the requested data has been returned, the transaction completes on the CP2-bus. The NCA is now ready to receive new DMA transactions on the CP2-bus. The nonrequested data is kept in CP2_RBUF as prefetch data.

The prefetch data in the CP2_RBUF is invalidated under the following conditions:

- DMA memory write or write-unlock on the CP2-bus to any memory address
- DMA memory read with different hexaword address or the prefetch data does not contain all the requested longword data
- DMA memory read-lock on the CP2-bus to any memory address
- I/O transaction on the CP2-bus initiated by the NCA

6.5.9 CP2 DMA Write

CP2 DMA writes are initiated by I/O devices that reside on the CP2-bus. DMA writes are supported by the NCA to VAX memory space only. All DMA writes to VAX I/O space are terminated by the NCA by asserting the CP2-bus CP2_ERR_L signal. The NCA supports DMA writes on the CP2-bus of longword, quadword, hexword, or octaword length. Longword and quadword writes are performed as quadword writes on the NDAL, and hexword and octaword writes are performed as octaword writes on the NDAL. Writes can be masked or unmasked.

When a DMA write happens on the CP2-bus, the address and data are latched in the CP2_WBUF. A DMA write on the CP2-bus causes the CP2_RBUF to be invalidated. When the address and all write data have been latched by the NCA, a DMA write request is sent to the NCA's NDAL master interface. The NCA will then arbitrate for the NDAL on behalf of the CP2-bus DMA device (CQBIC). When the NDAL is granted to the NCA, the write is initiated on the NDAL, thereby completing the DMA write transaction.

6.6 NCA Error Handling

Errors in the NCA can be of two types - NDAL-related errors and CP-bus related errors. These errors are described in the following tables.

Table 6–15 NDAL-Related Errors and NCA Responses

Description of Error	Specific Situation	Action Taken by NCA
NCA detects a parity error on any NDAL cycle	Error in CMD<3:0> or ID<2:0>	No ACK_L; set CESR<NDAL_PARITY_ERROR>; asserts S_ERR_L.
	Address cycle, error on NDAL<63:0>	No ACK_L; set CESR<NDAL_PARITY_ERROR>; asserts S_ERR_L.
	Data cycle, error on NDAL<63:0>	No ACK_L; set CESR<NDAL_PARITY_ERROR>; asserts S_ERR_L. If DMA read, then NDAL CP timer will eventually time out and CP-bus interface assert CP_ERR_L if not all requested data is received.
ACK_L is not received when NCA is master	Address cycle	Assert H_ERR_L if DMA write; set CESR<NACK>. Log address in CNEAR. If DMA read, then asserts CP_ERR_L to abort.
	Data cycle	If DMA write, then asserts H_ERR_L; set CESR<NACK>. Log address in CNEAR. If read data return, then asserts S_ERR_L; set CESR<RDR_ NACK>.
NCA detects illegal length to NCA addressing space	Address cycle	Asserts S_ERR_L; no ACK_L; set CESR<ILL_ LENGTH>; no address log.
NCA detects reserved command on any NDAL cycle	Address or data cycle	No ACK_L; set CESR<RESERVE_CMD>; no address log.
Read data return error	NDAL CP timer times out before all data is returned	Set CESR<CP_TIMEOUT_ERROR>; log address in CNEAR. If CP-bus is still waiting for data, assert CP_ ERR_L to abort.
	Receive RDE	If data is requested on CP-bus, assert CP_ERR_L; otherwise, ignore data returned.
Interrupt vector read error	No interrupts pending at that level and XA is not present	Set CESR<NCA_PREL>; return with RDR and NDAL bit<33,1> set to 1.

Table 6–16 CP-Bus (CP1 and CP2 Buses) Related Errors and NCA Responses

Description of Error	Specific Situation	Action Taken by NCA
NCA detects a parity error on CP-bus	DMA write data cycle	Set CESR<CP_DMA_PAR_ERR>; log address in CSEAR1 or CSEAR2. Perform BADWDATA operation on NDAL; asserts S_ERR_L.
	IO read data cycle	Set CESR<CP_IO_READ_PAR_ERR>; return RDE on NDAL. Log address in CIOEAR1 or CIOEAR2.
	Interrupt vector read data cycle	Set CESR<CP_IO_READ_PAR_ERR>, return RDE on NDAL. Log address in CIOEAR1 or CIOEAR2.
Invalid DMA address	NCA receives IO space address	Asserts CP_ERR_L; set CESR<CP_BUS_ERR>; no address logging.
Reserved command	NCA receives reserved command	Asserts CP_ERR_L; set CESR<CP_BUS_ERR>; no address logging.
CP_ERR_L is asserted when NCA is master	IO read	Set CESR<CP_IO_ERR>; return RDE to NDAL. Log address in CIOEAR1 or CIOEAR2.
	IO write	Assert H_ERR_L; set CESR<CP_IO_ERR>. Log address in CIOEAR1 or CIOEAR2.
	Interrupt vector read	Set CESR<CP_PREL>; return RDR with NDAL bit<33,1> set to 1.
No response abort	IO read	Set CESR<CP_NXIO>; return RDE on NDAL. Log address in CIOEAR1 or CIOEAR2.
	IO write	Assert H_ERR_L; set CESR<CP_NXIO>. Log address CIOEAR1 or CIOEAR2.
	Interrupt vector read	No action; the NCA will continue to wait for the read data or until either the CP MT timer times out or the assertion of CP_ERR_L before terminating the transaction.
CP MT timer times out	Interrupt vector read	Terminates CP-bus transaction by deasserting CP_AS_L and CP_DS_L. Return RDR with NDAL bit<33,1> set to 1; set CESR<CP_PREL>.
	IO read	Terminates CP-bus transaction by deasserting CP_AS_L and CP_DS_L. Set CESR<CP_MT_TIMEOUT>; log address in CIOEAR1 or CIOEAR2.
	IO write	Asserts H_ERR_L; terminates CP bus transaction by deasserting CP_AS_L and CP_DS_L. Set CESR<CP_MT_TIMEOUT>; log address in CIOEAR1 or CIOEAR2.

7

The Console Line, TOY Clock

7.1 KA680 Console Serial Line

The console serial line provides the KA680 processor with a full-duplex, RS-423 EIA, serial line interface, which is also RS-232-C 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*.

7.1.1 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 32-35. Refer to Table 7–1.

Table 7–1 Console Registers

IPR Number		Register Name	Mnemonic
Dec	Hex		
32	20	Console Receiver Control/Status	RXCS
33	21	Console Receiver Data Buffer	RXDB
34	22	Console Transmit Control/Status	TXCS
35	23	Console Transmit Data Buffer	TXDB

7.1.1.1 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. The format is shown in Figure 7–1. Table 7–2 lists the bit descriptions.

Figure 7–1 Console Receiver Control/Status Register (IPR 32_{10} 20_{16})

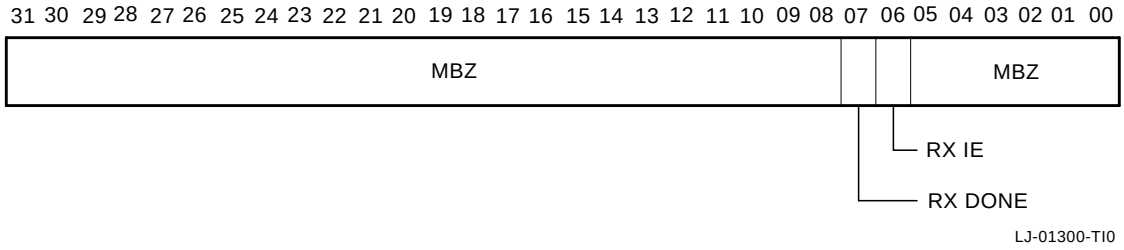


Table 7–2 Console Receiver Control/Status Register

Data Bit	Name	Description
<31:8>	MBZ	These bits read as zeros; writes have no effect.
<7>	RX DONE	Receiver done. Read-only. Writes have no effect. This bit is set when an entire character has been received and is ready to be read from the RXDB register. This bit is automatically cleared when the RXDB register is read. It is also cleared on powerup and on 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 powerup and on the negation of DCOK.
<5:0>	Unused	These bits read as zeros. Writes have no effect.

7.1.1.2 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. The format is shown in Figure 7–2. Bit descriptions are listed in Table 7–3.

Figure 7–2 Console Receiver Data Buffer (IPR 33_{10} 21_{16})

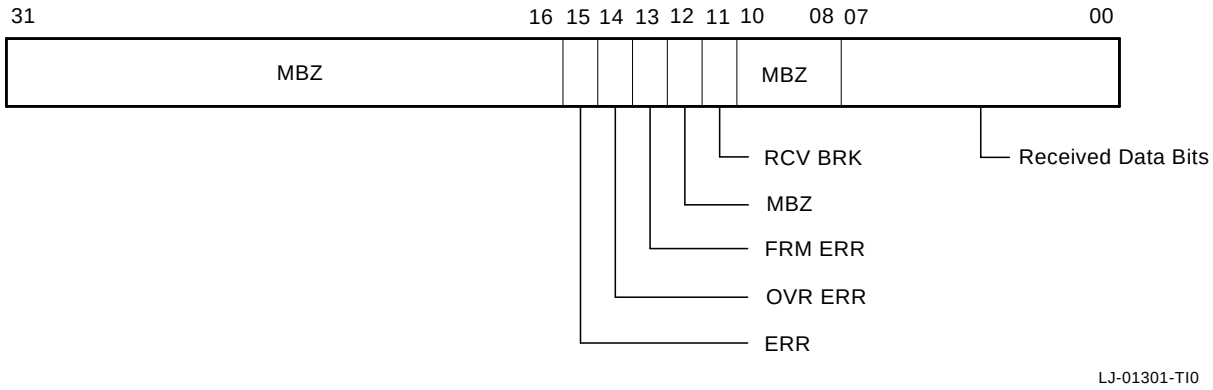


Table 7–3 Console Receiver Data Buffer

Data Bit	Name	Description
<31:16>	MBZ	These bits always read as zero. Writes have no effect.
<15>	ERR	Error. Read-only. Writes have no effect. This bit is set if RBUF <14> or <13> is set. It is clear if these two bits are clear. This bit cannot generate a program interrupt. Cleared on powerup and on 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. Cleared by reading the RXDB, on powerup, and on the negation of DCOK.
<13>	FRM ERR	Framing error. Read-only. Writes have no effect. This bit is set if the present character did not have a valid stop bit. Cleared by reading the RXDB, on powerup, and on the negation of DCOK. **Error conditions are updated when the character is received. Error conditions remain present until the character is read, at which point the error bits are cleared.**
<12>	MBZ	This bit always reads as zero. Writes have no effect.

(continued on next page)

Table 7–3 (Cont.) Console Receiver Data Buffer

Data Bit	Name	Description
<11>	RCV BRK	Received break. Read-only. Writes have no effect. This bit is set at the end of a received character for which the serial data input remained in the space condition for 20 bit times. Cleared by reading the RXDB, on powerup, and on the negation of DCOK.
<10:8>	MBZ	These bits always read as zero. Writes have no effect.
<7:0>	Received Data Bits	Read-only. Writes have no effect. These bits contain the last received character.

7.1.1.3 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. The format is shown in Figure 7–3. Bit descriptions are listed in Table 7–4.

Figure 7–3 Console Transmitter Control/Status Register (IPR 34_{10} 22_{16})

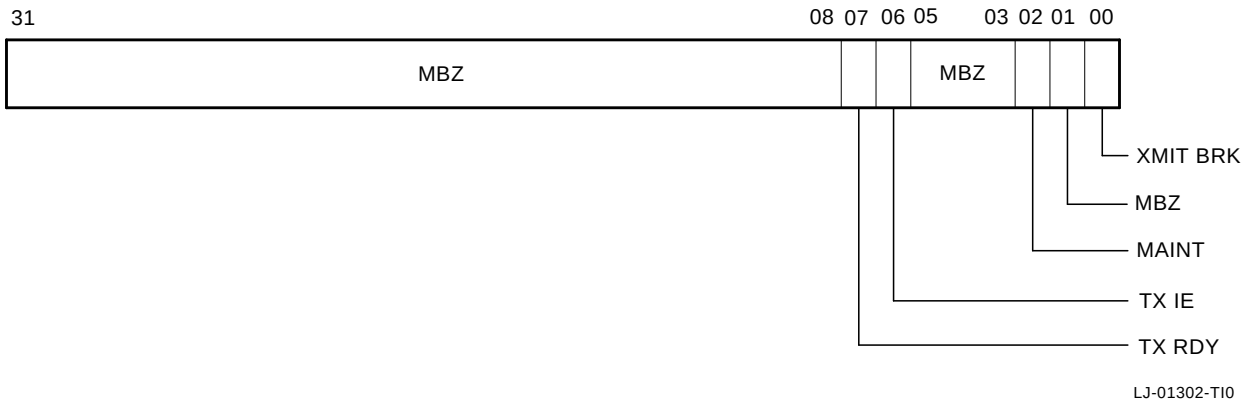


Table 7–4 Console Transmitter Control/Status Register

Data Bit	Name	Description
<31:8>	MBZ	These bits read as zeros. 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 powerup and on the negation of DCOK.
<6>	TX IE	Transmitter interrupt enable. Read/write. When set, this bit causes an interrupt to be requested 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 powerup and on the negation of DCOK.
<5:3>	MBZ	These bits read as zeros. Writes have no effect.
<2>	MAINT	Maintenance. Read/write. This bit is used to facilitate a maintenance self-test. When MAINT is set, the external serial output is set to MARK and the serial output is used as the serial input. This bit is cleared on powerup and on the negation of DCOK.
<1>	Unused	This bit reads as zero. Writes have no effect.
<0>	XMIT BRK	Transmit break. Read/write. When this bit is set, the serial output is forced to the space condition after the character in TXDB<7:0> is sent. While XMIT BRK is set, the transmitter will operate normally, but the output line will remain low. Thus, software can transmit dummy characters to time the break. This bit is cleared on powerup.

7.1.1.4 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. The format is shown in Figure 7–4. Table 7–5 lists the bit descriptions.

Figure 7–4 Console Transmitter Data Buffer (IPR 35_{10} 23_{16})

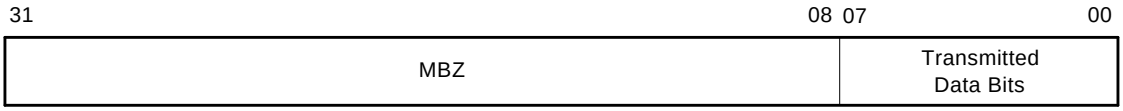


Table 7–5 Console Transmitter Data Buffer

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

7.1.2 Break Response

The console serial line unit recognizes a break condition, which consists of 20 consecutively received space bits. If the console detects a valid break condition, the RCV BRK bit is set in the RXDB register. If the break was the result of 20 consecutively received space bits, the FRM ERR bit is also set. If halts are enabled, the KA680 will halt and transfer program control to the console firmware ROM location E004 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.

7.1.3 Baud Rate

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

The user selects the desired baud rate through the baud rate select signals, which are received from an external 8-position switch mounted on the console module (H3604). The KA680 firmware must read this code from boot and diagnostic register bits <6:4>, compliment and then load it into SSC configuration register bits <14:12>.

Table 7–6 shows the baud rate selection, the corresponding code as read in the boot and diagnostic register bits <6:4>, and the inverted code that should be loaded into SSC configuration register bits <14:12>.

Table 7–6 Baud Rate Selection

Baud Rate	BDR<6:4>	SSC<14:12>
300	111	000
600	110	001

(continued on next page)

Table 7–6 (Cont.) Baud Rate Selection

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

7.1.4 Console Interrupt Specifications

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

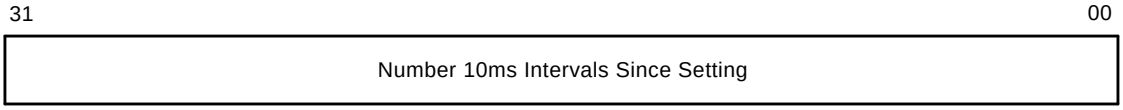
7.2 KA680 TOY Clock and Timers

The KA680 clocks include 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.

7.2.1 Time-of-Year Clock (TODR) - EPR 27

The time-of-year clock (TODR) forms an unsigned 32-bit binary counter that is driven from a 100 Hz oscillator, so that the least significant bit of the clock represents a resolution of 10 milliseconds, with less than .0025% error. The register counts only when it contains a nonzero value. This register is implemented in the SSC chip. The format is shown in Figure 7–5.

Figure 7–5 Time-of-Year Clock (TODR) - (EPR 27_{10} $1B_{16}$)



The time-of-year clock is maintained during power failure by battery backup circuitry that interfaces, via the external connector, to a set of batteries mounted on the CPU console module. The TODR will remain valid for more 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 which, if set after initialization, the TODR is cleared, and will remain at zero 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 one to it.

7.2.2 Programmable Timers

The KA680 features two programmable timers. Although they are modeled after the VAX standard interval clock, they are accessed as I/O space registers (rather than as internal processor registers) and a control bit has been added that 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. **The KA680 firmware uses these timers. They are not preserved across console activity.**

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

The timer number (0 or 1) is represented by **n**.

7.2.2.1 Timer Control Registers (TCR0 and TCR1)

The KA680 has two timer control registers: one for controlling timer 0 (TCR0), and one for controlling timer 1 (TCR1). TCR0 is accessible at address 2014 0100_{16}, and TCR1 is accessible at 2014 0110_{16}. These registers are implemented in the SSC chip. Figure 7–6 shows the format. Table 7–7 lists the bit descriptions.

Figure 7–6 Timer Control Registers (TCR0 and TCR1)

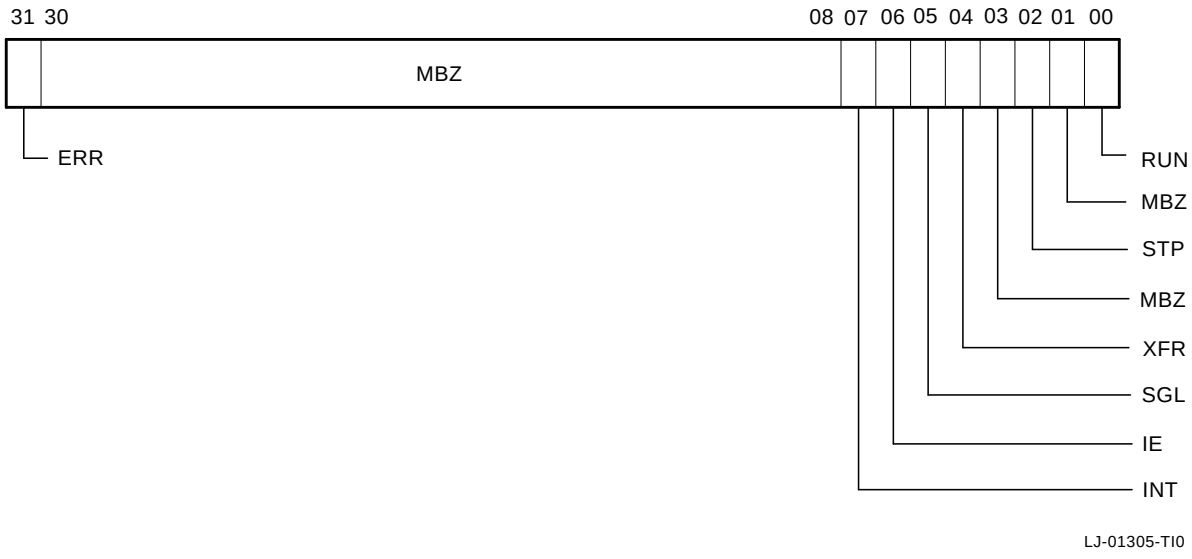


Table 7–7 Timer Control Register Bit Descriptions

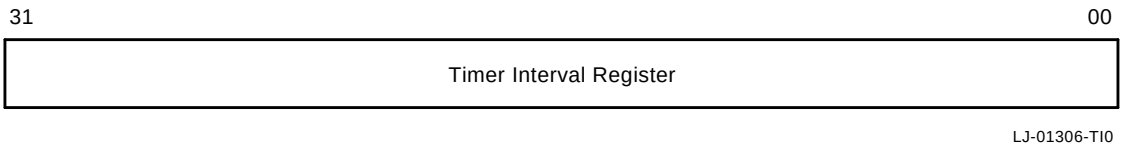
Date Bit	Name	Description
<31>	ERR	Error. Read/write to clear. This bit is set whenever the timer interval register overflows and the INT bit is already set. Thus, the ERR bit indicates a missed overflow. Writing a one to this bit clears it. Cleared on powerup.
<30:8>	MBZ	Read as zeros, must be written as zeros.
<7>	INT	Read/write to clear. This bit is set whenever the timer interval register overflows. If IE is set when INT is set, an interrupt is posted at IPL 14. Writing a one to this bit clears it. Cleared on powerup.
<6>	IE	Read/write. When this bit is set, the timer will interrupt at IPL 14 when the INT bit is set. Cleared on powerup.
<5>	SGL	Read/write. Setting this bit causes the timer interval register to be incremented by one if the RUN bit is cleared. If the RUN bit is set, then writes to the SGL bit are ignored. This bit is always read as zero. Cleared on powerup.
<4>	XFR	Read/write. Setting this bit causes the timer next interval register to be copied into the timer interval register. This bit is always read as zero. Cleared on powerup.
<3>	MBZ	Read as zeros, must be written as zeros.
<2>	STP	Read/write. This bit determines whether the timer stops after an overflow when the RUN bit is set. If the STP bit is set at overflow, the RUN bit is cleared by the hardware at overflow and counting stops. Cleared on powerup.
<1>	MBZ	Read as zeros, must be written as zeros.
<0>	RUN	Read/write. When set, the timer interval register is incremented once every microsecond. The INT bit is set when the timer overflows. If the STP bit is set at overflow, the RUN bit is cleared by the hardware at overflow and counting stops. When the RUN bit is clear, the timer interval register is not incremented automatically. Cleared on powerup.

7.2.2.2 Timer Interval Registers (TIR0 and TIR1)

The KA680 has two timer interval registers: one for timer 0 (TIR0), and one for timer 1 (TIR1). TIR0 is accessible at address 2014 0104_{16}, and TIR1 is accessible at 2014 0114_{16}.

The timer interval register is a read-only register containing the interval count. When the RUN bit is 0, writing a 1 increments the register. When the RUN bit is 1, the register is incremented once every microsecond. When the counter overflows, the INT bit is set, and an interrupt is posted at 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 powerup. Figure 7–7 shows the format.

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

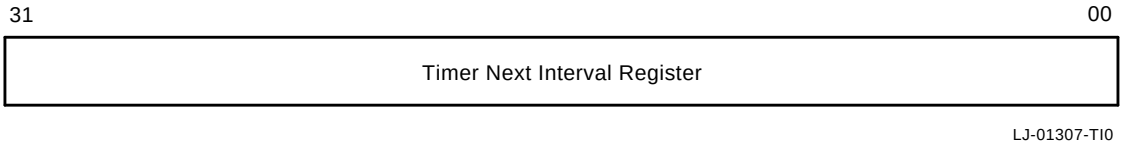


7.2.2.3 Timer Next Interval Registers (TNIR0 and TNIR1)

The KA680 has two timer next interval registers: one for timer zero (TNIR0), and one for timer one (TNIR1). TNIR0 is accessible at address 2014 0108_{16}, and TNIR1 is accessible at 2014 0118_{16}. These registers are implemented in the SSC chip. The format is shown in Figure 7–8.

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

Figure 7–8 Timer Next Interval Registers (TNIR0 and TNIR1)

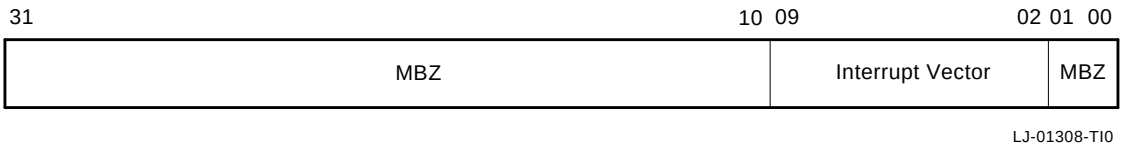


7.2.2.4 Timer Interrupt Vector Registers (TIVR0 and TIVR1)

The KA680 has two timer interrupt vector registers: one for timer zero (TIVR0), and one for timer one (TIVR1). TIVR0 is accessible at address 2014 $010C_{16}$, and TIVR1 is accessible at 2014 $011C_{16}$. These registers are implemented in the SSC chip and are set to 78_{16} and $7C_{16}$, respectively, by the resident firmware. The format is shown in Figure 7–9.

This read/write register contains the timer's interrupt vector. Bits <31:10> and <1:0> are read as zeros and must be written as zeros. When TCRn<6> (IE) and TCRn<7> (INT) transition to one, an interrupt is posted at IPL14. When a timer's interrupt is acknowledged, the content of the interrupt vector register is passed to the CPU, and the INT bit is cleared. Interrupt requests can also be cleared by clearing either the IE or INT bit. This register is cleared on powerup.

Figure 7–9 Timer Interrupt Vector Registers (TIVR0 and TIVR1)



Note

Note that both timers interrupt at the same IPL (IPL14) as the console serial line unit. When multiple interrupts are pending, the console serial line has priority over the timers, and timer 0 has priority over timer 1.

8

KA680 Boot and Diagnostic Facility

The KA680 boot and diagnostic facility features two registers, 512 KB of flash programmable read-only memory (FEPROM), and 1 KB of battery backed up RAM. The EPROM and battery backed up RAM may be accessed with longword, word, or byte references.

8.1 Boot and Diagnostic Register (BDR)

The boot and diagnostic register is a longword-wide register located in the VAX I/O page at physical addresses 2008 4000 - 2008 $407C_{16}$. It can be accessed by KA680 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 KA680 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 eight bytes contain the station address. The next 24 bytes are provided for testing purposes. Figure 8–1 shows the format for the boot and diagnostic register. Table 8–1 describes the bits in the register.

Figure 8–1 Boot and Diagnostic Register (BDR)

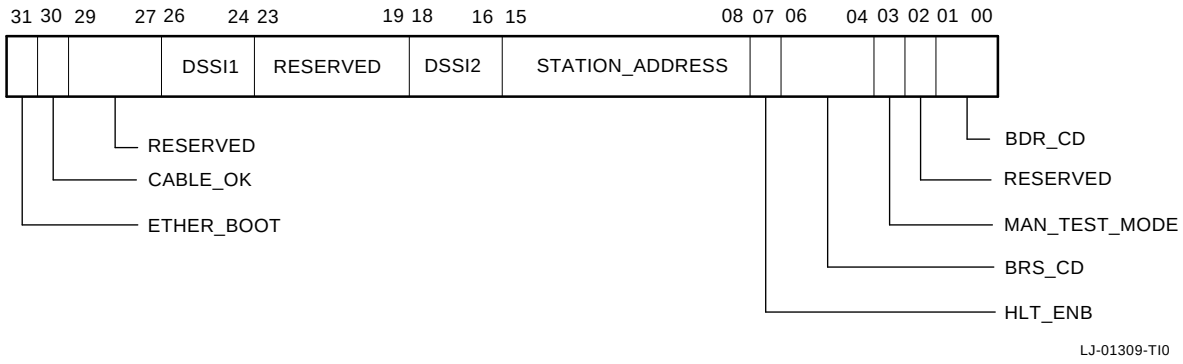


Table 8–1 Boot and Diagnostic Register Bit Description

Data Bit	Name	Description
<31>	ETHER_BOOT	Enable Ethernet remote boot. This bit reflects the current setting of the enable Ethernet remote boot jumper found on the console module (H3604). If this bit is zero, remote Ethernet boots are enabled. If this bit is one, remote Ethernet boots requests are ignored.
<30>	CABLE_OK	Console module cable OK. When this bit is zero, there is a high probability that the console module cable is functioning correctly. If this bit is one, the console module cable is either malfunctioning or plugged in the wrong orientation. This bit is determined by sending a signal out to the console module over one path and reading it back down another on the cable.
<29:27>	Reserved	Reserved.
<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>	Reserved	Reserved.
<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 KA680'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 provided for testing purposes.

(continued on next page)

Table 8–1 (Cont.) Boot and Diagnostic Register Bit Description

Data Bit	Name	Description
<7>	HLT ENB	Halt enable, read-only. Writes have no effect. This bit reflects the state of break enable switch on the console module (H3604). The assertion of this signal enables the halting of the CPU upon detection of a console break condition. On a powerup, the KA680 resident firmware reads the HLT ENB bit to decide whether to enter the console emulation program (HLT ENB set) or to boot the operating system (HLT ENB clear). On the execution of of a HALT instruction while in 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).
<6:4>	BRS CD	Baud rate select—read-only. Writes have no effect. These three bits originate from the console module (H3604) baud rate select switch. They reflect the setting of the the baud rate as shown 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 this bit is set, the KA680 is in normal run mode. When cleared (by grounding a test point on the backplane), the KA680 is in manufacturing test mode. In this mode, special diagnostic test scripts can be run on the console.
<2>	RESERVED	Reserved.
<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 KA680 firmware programs use BDG_CD <1:0> to determine the power-up mode as shown in the following table:

(continued on next page)

Table 8–1 (Cont.) Boot and Diagnostic Register Bit Description

Data Bit	Name	Description

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

8.2 Diagnostic LED Register (DLEDR)

The diagnostic LED register (DLEDR), address 2014 0030_{16}, is implemented in the SSC chip and contains four read/write bits that control the external LED display. A zero in a bit lights the corresponding LED; all four bits are cleared on powerup and on the negation of DCOK to provide a power-up lamp test. Figure 8–2 shows the register format. Table 8–2 lists the bit descriptions.

Figure 8–2 Diagnostic LED Register (DLEDR)

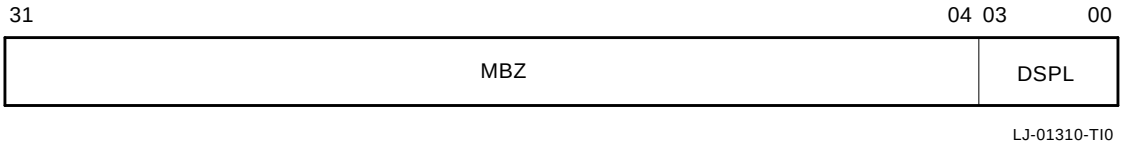


Table 8–2 Diagnostic LED Register Bit Descriptions

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

8.3 EPROM Memory

The KA680 has 512 KB of flash EPROM memory for storing code for the following functions:

Board initialization
Board self-tests
Boot code
VAX standard console emulation

EPROM memory may be accessed via byte, word, and longword references. The EPROM is organized as a 512K x 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 KB block of the local EPROM space (E004 0000 - E004 $1FFF_{16}$).

8.3.1 EPROM Address Space

Only the first 256 KB of the 512 KB of ROM can be read in the region E004 0000 - E007 $FFFF_{16}$. By appropriate programming of the SSC's programmable address strobe 0 match and mask registers, all 512 KB of ROM can be read at the locations programmed by the user. Care must be taken, however, to ensure that the I/O address space chosen for the EPROM's alternate address space is allowed by the NCA's address map, and does not conflict with any other I/O devices on the module. When this is done, the lower 256 KB of the ROM's alternate address space is a copy of the 256 KB that appears at E004 0000 - E007 FFFF.

Note

There is no concept of halt unprotect space (as used on previous Q22-based MicroVAX systems) on the KA680.

Any I-stream read from the EPROM space places the KA680 in halt mode. The Q22–bus SRUN signal is deasserted, causing the front panel RUN light to extinguish and the CPU is protected from further halts.

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

Note

The logic that controls halt mode/run mode can only detect I-stream references to addresses mapped to CP2.

8.3.2 KA680 Resident Firmware Operation

The KA680 CPU module's 512 KB of EPROM contain the resident firmware, which can be entered by transferring program control to location E004 0000_{16}.

Appendix C lists the various halt conditions that cause the KA680 to transfer program control to location E004 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 powerup.
- An interactive command language allowing the user to examine and alter the state of the processor.
- Diagnostic tests executed on powerup that check out the CPU, the memory system, the Q22–bus map, the SHAC, and the SGEC.
- Support of video or hardcopy terminals as the console terminal.

8.3.2.1 Power-Up Modes

The boot and diagnostic EPROM programs use boot and diagnostic code <1:0> to determine the power-up modes shown in Table 8–3.

Table 8–3 Power-Up Modes

Code	Power-up Mode
11	Run (factory setting). If the console terminal supports the multinational character set (MCS), the user will be prompted for language if the time-of-year clock battery backup has failed, or SSC RAM is corrupted or uninitialized (1st powerup). Full startup diagnostics are run.
01	Language inquiry. If the console terminal supports MCS, the user will be prompted for language on every powerup and restart. Full startup diagnostics are run.
10	Test. EPROM programs run wraparound serial line unit (SLU) tests.
00	Unused.

8.4 Battery Backed-up RAM

The KA680 contains 1 KB of battery backed-up static RAM, located in the SSC, for use as a console "scratchpad." This RAM supports byte, word, and longword references. Read operations take 700 ns to complete, while write operations require 600 ns. The RAM is organized as a 256 X 32-bit (one longword) array. The array appears in a 1 KB block of the VAX I/O page at addresses 2014 0400 - 2014 $07FF_{16}$. This array is not protected by parity, and CP-bus parity is neither checked nor generated on reads or writes to this RAM.

8.5 KA680 Initialization

The VAX architecture defines three kinds of hardware initialization:

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

8.5.1 Power-up Initialization

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

8.5.2 Hardware Reset

A hardware reset occurs on powerup or the negation of DCOK. A hardware reset causes the hardware halt procedure to be initiated with a halt code of 03. It also initializes some IPRs and most I/O page registers to a known state. Those IPRs affected by a hardware reset are noted in Section 3.1.3. The effect that a hardware reset has on I/O space registers is documented in the description of the register.

8.5.3 I/O Bus Initialization

An I/O bus initialization occurs on powerup, 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 IPCR and DSER, and 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 has no effect on the KA680.

8.5.3.1 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 IORESET causes an I/O bus initialization. Note that the SGEC and SHACs are not reset by MTPRs to IPR 55.

8.5.4 Processor Initialization

A processor initialization occurs on powerup, the negation of DCOK, as the result of a console INITIALIZE command, and after a halt caused by an error condition.

In addition to initializing those registers defined in the *VAX Architecture Reference Manual,* the KA680 firmware must also configure main memory, the local I/O page, and the Q22–bus map during a processor initialization.

8.5.4.1 Configuring the Local I/O Page

The following registers control the configuration of the KA680 local I/O page. They are unique to CPU designs that use the 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

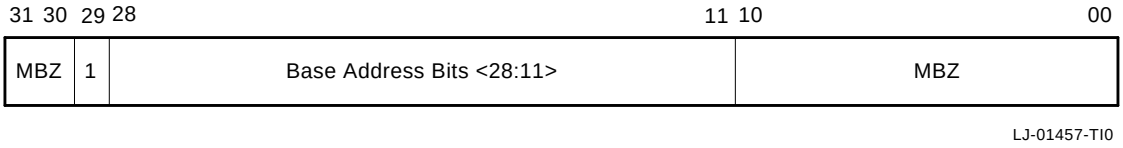
8.5.5 SSC Base Address Register (SSCBR)

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

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

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

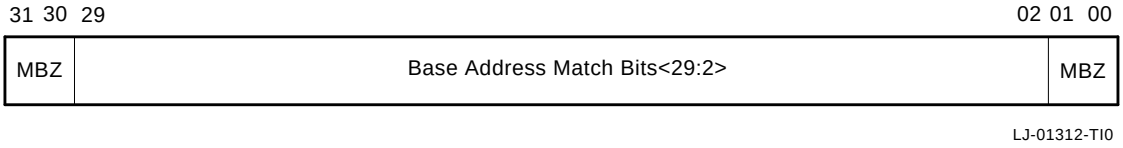
Figure 8–3 SSC Base Address Register (SSCBR)



8.5.6 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 powerup and on the negation of DCOK. BDMTR<31:30,1:0> are unused. They read as zeros, and must be written as zeros. This register should be set to 2008 4000_{16} by firmware during processor initialization. The BDMTR has the format shown in Figure 8–4.

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



8.5.7 BDR Address Decode Mask Register (BDMKR)

The BDR address decode mask register, address 2014 0144_{16}, controls the range of addresses to which the BDR responds. (An example is the number of copies of the BDR that appear in the physical address space.) This read/write register is cleared on powerup and on the negation of DCOK. Bits BDMKR<31:30,1:0> are unused. They read as zeros, and must be written as zeros. 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 KA680's LAN station address ROM are read using the BDR. The BDMKR has the format shown in Figure 8–5.

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

8.5.8 SSC Configuration Register (SSCCR)

The SSC configuration register, address 2014 0010_{16}, controls the set-up parameters for the console serial line, programmable timers, EPROM, TOY clock, and BDR. The format is shown in Figure 8–6. Table 8–4 contains a list of the bit descriptions.

Figure 8–6 SSC Configuration Register (SSCCR)

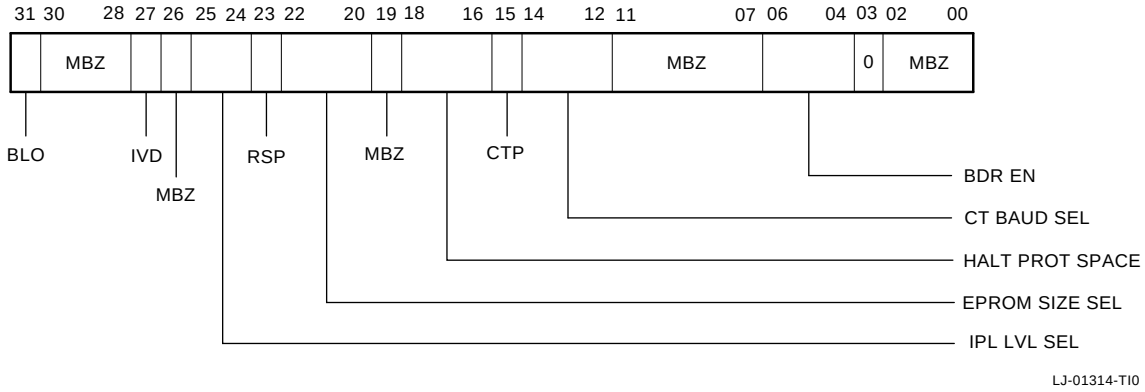


Table 8–4 SSC Configuration Register Bit Descriptions

Data Bit	Name	Description
<31>	BLO	Battery low. Read/write. If the battery voltage goes below threshold while the module is powered down, this bit is set on powerup, after the assertion of DCOK after the assertion of POK. Once set, this bit can only be cleared by software writing it as one. If this bit is set, then the TOY clock will be cleared by powerup and and by the negation of DCOK.
<30:28>	MBZ	Read as zeros, must be written as zeros.
<27>	IVD	Interrupt vector disable. Read/write. When set, the console serial line and programmable timers will not respond to interrupt acknowledge cycles. Cleared on powerup, by the negation of DCOK, and by a processor initialization.
<26>	MBZ	Read as zeros, must be written as zeros.
<25:24>	IPL_LVL_ SEL	IPL level select read/write. These bits are used to specify the IPL level of interrupt acknowledge cycle to which the console serial line and programmable timers respond. These bits must be cleared [programmed to 00 (binary)] for the console serial line and programmable timers to respond to interrupt acknowledge cycles that they generated (IPL 14). These bits are cleared on powerup, by the negation of DCOK, and by a processor initialization.
<23>	RSP	ROM speed. Read/write. This bit is used to select the EPROM access time. This bit must be set for the KA680 EPROMs to run at maximum speed. This bit is cleared on powerup and by the negation of DCOK. It must be set to one 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 to which the EPROM responds. These bits must be set to 101 (binary) because the KA680 contains 256 KB of EPROM, yielding an address range of 256 KB (E004 0000 - E007 $FFFF_{16}$). These bits are cleared on powerup and by the negation of DCOK, yielding an address range of 8 KB (E004 0000 - E004 $1FFF_{16}$). These bits must be set to the proper value during processor initialization.
<18:16>	HALT PROT SPACE	EPROM halt protect address space size select. Read/write. These bits control the size of the halt mode address range. These bits must be set to 101 (binary) because the KA680 contains 256 KB of EPROM, yielding a halt mode address range of 256 KB (E004 0000 - E007 $FFFF_{16}$). These bits are cleared on powerup and by the negation of DCOK, yielding a halt mode address range of 8 KB (E004 0000 - E004 $1FFF_{16}$). These bits must be set to the proper value by a processor initialization. Note that any instruction fetch from the EPROM puts the KA680 in halt protect mode.

(continued on next page)

Table 8–4 (Cont.) SSC Configuration Register Bit Descriptions

Data Bit	Name	Description
<15>	CTP	Control P enable. Read/write. When this bit is set, a CTRL/P typed at the console causes the CPU to be halted, if halts are enabled (BDR<7> set). When this bit is cleared, a BREAK typed at the console causes the CPU to be halted, if halts are enabled (BDR<7> set). This bit is cleared on powerup and by the negation of DCOK.
<14:12>	CT BAUD SELECT	Console terminal baud rate select. Read/write. These bits are used to select the baud rate of the console terminal serial line. They are cleared on powerup and by the negation of DCOK. They should be loaded from compliment of BDR<6:4> by the processor initialization code. The bit encodings correspond to selected baud rates as shown in the following table:

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

<11:7>	MBZ	Read as zero, must be written as zero.
<6:4>	BDR EN	Read/write. These bits are used to enable the BDR. They are cleared on powerup and by the negation of DCOK. These bits must be set to 111 (binary) by a processor initialization to enable the BDR.
<3>	MBZ	Read as zero, must be written as zero.
<2:0>	MBZ	Read as zero, must be written as zero.

9

KA680 Q22–bus Interface

The KA680 includes a Q22–bus interface implemented via a single VLSI chip called the CQBIC. It contains a CP 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 that allows any page in the Q22–bus memory space to be mapped to any page in main memory.
- A direct mapping function for translating 29-bit 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 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 using two asynchronous DMA octaword transfers. For block mode writes of fewer than sixteen words, the words are buffered and transferred to main memory using the most efficient combination of octaword, quadword, and longword asynchronous DMA transfers. The maximum write bandwidth for block mode references is 3.3 MB/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. Therefore, 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 MB/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.

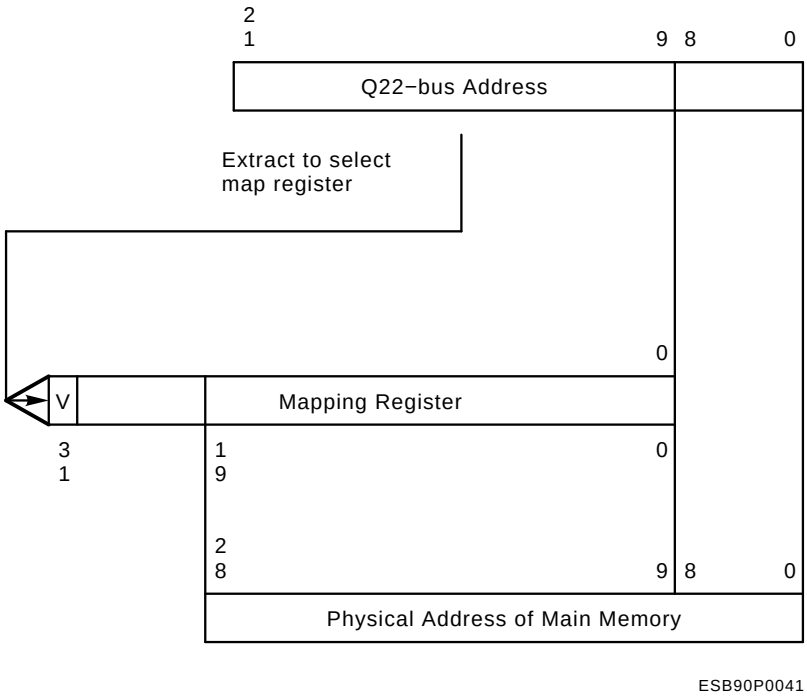
The interface also contains Q22–bus interrupt arbitration logic that recognizes Q22–bus interrupt requests BR7-BR4, and translates them into CPU interrupts at levels 17-14.

The Q22–bus interface detects Q22–bus "no sack" timeouts, Q22–bus interrupt acknowledge timeouts, Q22–bus nonexistent memory timeouts, and main memory errors on DMA accesses from the Q22–bus and Q22–bus device parity errors.

9.1 Q22–bus to Main Memory Address Translation

On DMA references to main memory, the 22-bit, Q22–bus address must be translated into a 29-bit main memory address (Figure 9–1). This translation process is performed by the Q22–bus interface using the Q22–bus map. This map contains 8192 mapping registers, (one for each page in the Q22–bus memory space), each of which can map a page (512 bytes) of the Q22–bus memory address space into any of the 1024K pages in main memory. Since local I/O space addresses cannot be mapped to Q22–bus pages, the local I/O page is inaccessible to devices on the Q22–bus. Figure 9–1 shows how Q22–bus addresses are translated into main memory addresses.

Figure 9–1 Q22–bus Address Translation



ESB90P0041

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

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

Note

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

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

9.1.1 Q22–bus Map Registers (QMRs)

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

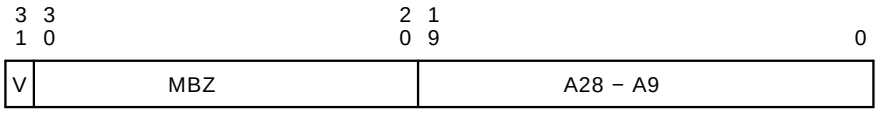
The local I/O space address of each register was chosen so that register address bits <14:2> are identical to Q22–bus address bits <21:9> of the Q22–bus page that the register maps. Table 9–1 lists the register addresses.

Table 9–1 Q22–bus Map Register Addresses

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

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

Figure 9–2 Q22–bus Map Register Format



ESB90P0043

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

Table 9–2 Q22–bus Map Register Bit Description

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

9.1.2 Accessing the Q22–bus Map Registers

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

Note

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

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

On a Q22–bus map read by the CPU, the Q22–bus interface decodes the local I/O space address (2008 8000 - 2008 $FFFC_{16}$). If the register is in the Q22–bus map cache, the Q22–bus interface will internally resolve any conflicts between CPU and Q22–bus transactions (if both are attempting to access the Q22–bus map cache entries at the same time), then return the data. If the map register is not in the map cache, the Q22–bus interface will force the CPU to retry, acquire the CP- bus, and perform an asynchronous DMA read of the map register. On completion of the read, the CPU is provided with the data when its read operation is retried. A map read by the CPU does not cause the register that was read to be stored in the map cache.

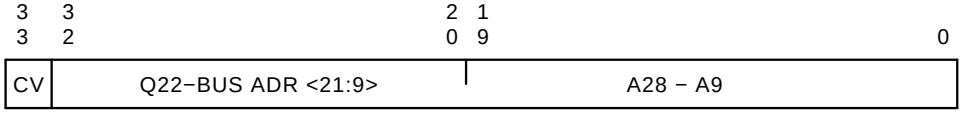
On a Q22–bus map write by the CPU, the Q22–bus interface latches the data. On completion of the CPU write, it acquires the CP- bus and performs an asynchronous DMA write to the map register. If the map register is in the Q22–bus map cache, then the CAMValid bit for that entry will be cleared to prevent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

9.1.3 The Q22–bus Map Cache

To speed up the process of translating Q22–bus address to main memory addresses, the Q22–bus interface utilizes a fully associative, 16-entry, Q22–bus map cache that is 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 9–3 shows the format. Table 9–3 contains a description of the Q22–bus map cache entry bits.

Figure 9–3 Q22–bus Map Cache Entry Format



ESB90P0044

Table 9–3 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 powerup, the negation of DCOK, setting the QMCIA (Q22–bus map cache invalidate all) bit in the interprocessor communication register, writes to IPR 55 (IORESET), by a write to the Q22–bus map base register, or by writing to the QMR that is being cached.
<32:20>	QBUS ADR	These bits contain the Q22–bus address bits <21:9> of the page that this entry maps. This is the content addressable field of the 16-entry cache for determining if the map register for a particular Q22–bus address is in the map cache. These bits are undefined on powerup.
<19:0>	Address bits A28-A9	When a mapping register is selected by a Q22–bus address, and if that register's CAMValid bit is set, then these 20 bits are used as main memory address bits 28 through 9. Q22–bus address bits 8 through 0 are used as local memory address bits 8 through 0. These bits are undefined on powerup.

9.2 CP to Q22–bus Address Translation

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

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

9.3 Interprocessor Communications Facility

The KA680 can only be configured as a Q22–bus arbiter.

The KA680 interprocessor communication facility allows other processors on the Q22–bus to request program interrupts from the KA680 without using the Q22–bus interrupt request lines. It also controls external access to local memory (via the Q22–bus map).

9.3.1 Interprocessor Communication Register (IPCR)

The interprocessor communication register is a 16-bit register residing in the Q22–bus I/O page address space, and can be accessed by any device that can become Q22–bus master (including the KA680 itself). The IPCR is implemented in the CQBIC chip and is byte accessible, meaning that a write byte instruction can write to either the low or high byte without affecting the other byte. Figure 9–4 shows the format. Table 9–4 describes the bits.

Figure 9–4 Interprocessor Communication Register (IPCR)

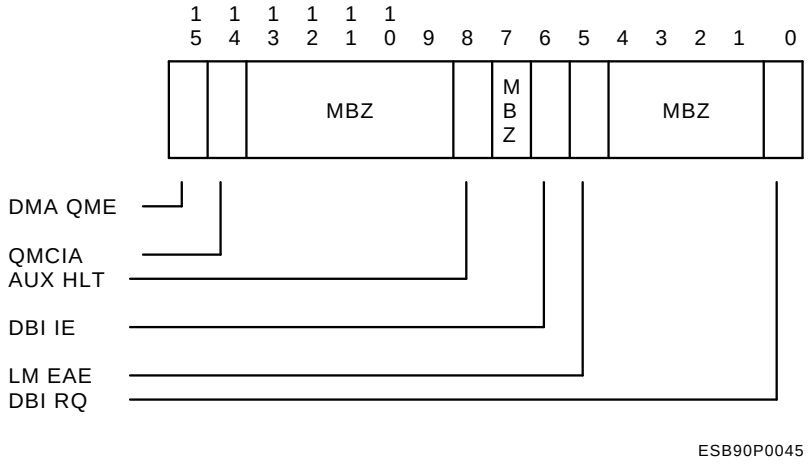


Table 9–4 Interprocessor Communication Register Bit Description

Data Bit	Name	Description
<15>	DMA QME	DMA Q22–bus address space memory error. Read/write to clear. This bit indicates that an error occurred when a Q22–bus device was attempting to read main memory. It is set if DMA system error register bit DSER<4> (MAIN MEMORY ERROR) is set, or the CP timer expires. The MAIN MEMORY ERROR bit indicates that an uncorrectable error occurred when an external device (or CPU) was accessing the KA680 local memory.

(continued on next page)

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

Data Bit	Name	Description
		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 one to it, on powerup, by the negation of DCOK, by writes to IPR 55 (IORESET), and whenever DSER<4> is cleared.
<14>	QMCIA	Q22–bus map cache invalidate all. Write-only. Writing a one to this bit clears the CAMValid bits in the cached copy of the MAP. This bit always reads as zero. Writing a zero has no effect.
<13:09>	Unused	Read as zeros. Must be written as zeros.
<8>	AUX HLT	Auxiliary halt. Read-only. When this bit is set, it has no effect on the operation of the on-board CPU. This bit is cleared on powerup, 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 zero. Must be written as zero.
<6>	DBI IE	Doorbell interrupt enable. Read/write when the KA680 is Q22–bus master. Read-only when another device is Q22–bus master. When set, this bit enables interprocessor doorbell interrupt requests via IPCR<0>. Cleared on powerup, by the negation of DCOK, and writes to IPR 55 (IORESET).
<5>	LM EAE	Local memory external access enable. Read/write when the KA680 is Q22–bus master. Read-only when another device is Q22–bus master. When set, this bit enables external access to local memory (via the Q22–bus map). Cleared on powerup and by the negation of DCOK.
<4:1>	Unused	Read as zeros. Must be written as zeros.
<0>	DBI RQ	Doorbell interrupt request. Read/write. If IPCR<6> (DBI IE) is set, setting this bit generates a doorbell interrupt request. If IPCR<6> is clear, setting this bit has no effect. Clearing this bit has no effect. DBI RQ is cleared when the CPU grants the doorbell interrupt request. DBI RQ is held clear whenever DBI IE is clear. This bit is cleared on powerup and the negation of DCOK.

9.3.2 Interprocessor Doorbell Interrupts

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

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

Note

Following an interprocessor doorbell interrupt, the KA680 CPU sets the IPL to 14. The IPL is set to 17 for external Q22–bus BR4 interrupts.

9.4 Q22–bus Interrupt Handling

The KA680 responds to interrupt requests BR7-4 with the standard Q22–bus interrupt acknowledge protocol (DIN followed by IAK). The console serial line unit, the programmable timers, and the interprocessor doorbell request interrupt at IPL 14 and have priority over all Q22–bus BR4 interrupt requests. After responding to any interrupt request BR7-4, the CPU sets the processor priority to IPL 17. All BR7-4 interrupt requests are disabled unless software lowers the interrupt priority level.

Interrupt requests from the KA680 interval timer are handled directly by the CPU. Interval timer interrupt requests have a higher priority than BR6 interrupt requests. After responding to an interval timer interrupt request, the CPU sets the processor priority to IPL 16. Thus, BR7 interrupt requests remain enabled.

9.5 Configuring the Q22–bus Map

The KA680 implements the Q22–bus map in an 8K longword (32 KB) block of main memory. This map must be configured by the KA680 firmware during a processor initialization by writing the base address of the uppermost 32 KB block of good main memory into the Q22–bus map base register. The base of this map must be located on a 32 KB boundary.

Note

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

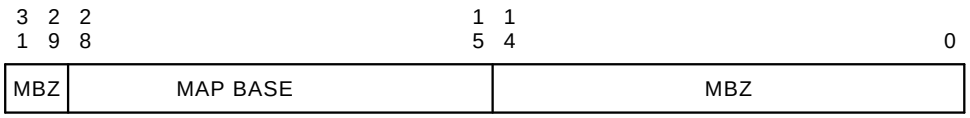
9.5.1 Q22–bus Map Base Address Register (QBMBR)

The Q22–bus map base address register, address 2008 0010_{16}, controls the main memory location of the 32 KB 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 zeros, and will return zeros when read. Figure 9–5 shows the format.

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

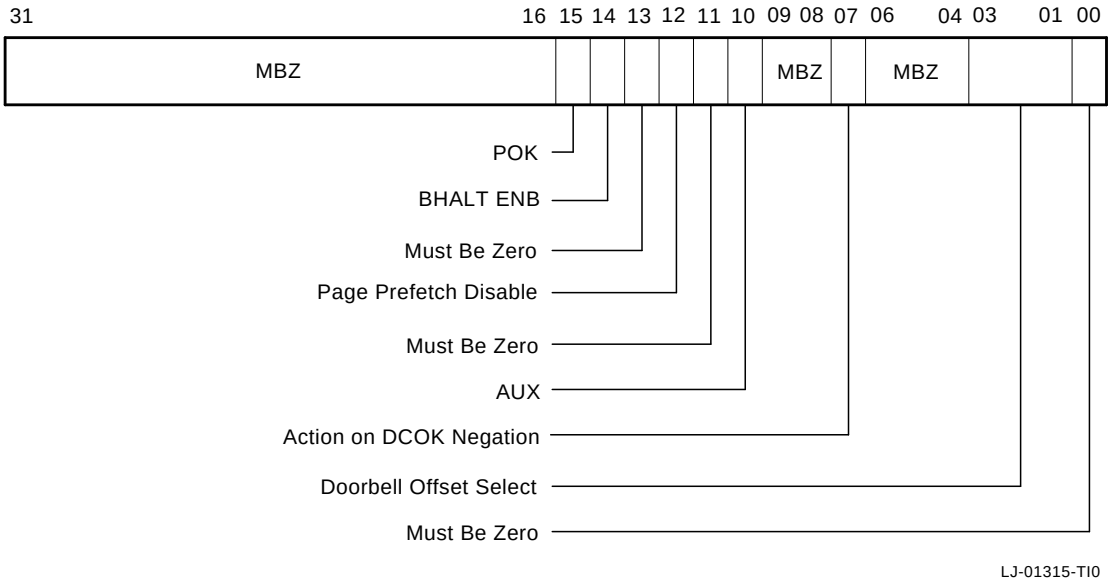
The contents of this register are undefined on powerup and the negation of DCOK, and are not affected by BINIT being asserted on the Q22–bus.

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



ESB90P0046

Figure 9–6 System Configuration Register (SCR)



9.6 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 OK flag, and an AUX flag. Figure 9–6 shows the format. Table 9–5 describes the bits in the system configuration register.

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

Table 9–5 System Configuration Register Bit Description

Data Bit	Name	Description
<31:16>	Unused	Read as zeros. Must be written as zeros.
<15>	POK	Power OK. Read-only. Writes have no effect. This bit is set if the Q22–bus BPOK signal is asserted and clear if it is negated. This bit is cleared on powerup and by the negation of DCOK.
<14>	BHALT EN	BHALT enable. Read/write. This bit controls the effect the Q22–bus BHALT signal has on the CPU. When set, asserting the Q22–bus BHALT signal will halt the CPU and assert DSER<15>. When cleared, the Q22–bus BHALT signal will have no effect. This bit is cleared on powerup and by the negation of DCOK.

(continued on next page)

Table 9–5 (Cont.) System Configuration Register Bit Description

Data Bit	Name	Description
<10>	AUX	Auxiliary. Read-only. Writes have no effect. This bit defines auxiliary and arbiter mode of operation of the KA680. When read as a zero, arbiter mode is selected, and when read as a one, auxiliary mode is selected. Because the KA680 can only be configured as an arbiter, this bit should always read as zero.
<9:8>	Unused	Read as zeros. Must be written as zeros.
<7>	ACTION ON DCOK NEGATION	Read/write. When cleared, the Q22–bus interface asserts SYSRESET (causing a hardware reset of the board and control to be passed to the resident firmware via the hardware halt procedure with a halt code of 3) when DCOK is negated on the Q22–bus. When set, the Q22–bus interface asserts HALCYON (causing control to be passed to the resident firmware via the hardware halt procedure with a halt code of 2) when DCOK is negated on the Q22–bus. Cleared on powerup and the negation of DCOK.
<6:4>	Unused	Read as zeros. Must be written as zeros.
<3:1>	Unused	Read as zeros. Must be written as zeros.
<0>	Unused	Read as zero. Must be written as zero.

9.7 Error Reporting Registers

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

- The DMA system error register (DSER)
- The Q22–bus error address register (QBEAR)
- The DMA error address register (DEAR)

These registers are located in the local VAX I/O address space and can only be accessed by the local processor. The **DSER** is implemented in the CQBIC chip and it logs main memory errors on DMA transfers, Q22–bus parity errors, Q22–bus nonexistent memory errors, and Q22–bus no-grant. 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 **DEAR** contains the address of the page in local memory, which caused a memory error during an access by an external device or the processor during a local-miss, global-hit transaction. An access by the local processor that the Q22–bus interface maps into main memory will provide error status to the processor when the processor does a retry for a read local-miss, global-hit, or by an interrupt in the case of a local-miss, global-hit write.

9.7.1 DMA System Error Register (DSER)

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

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

Figure 9–7 DMA System Error Register (DSER)

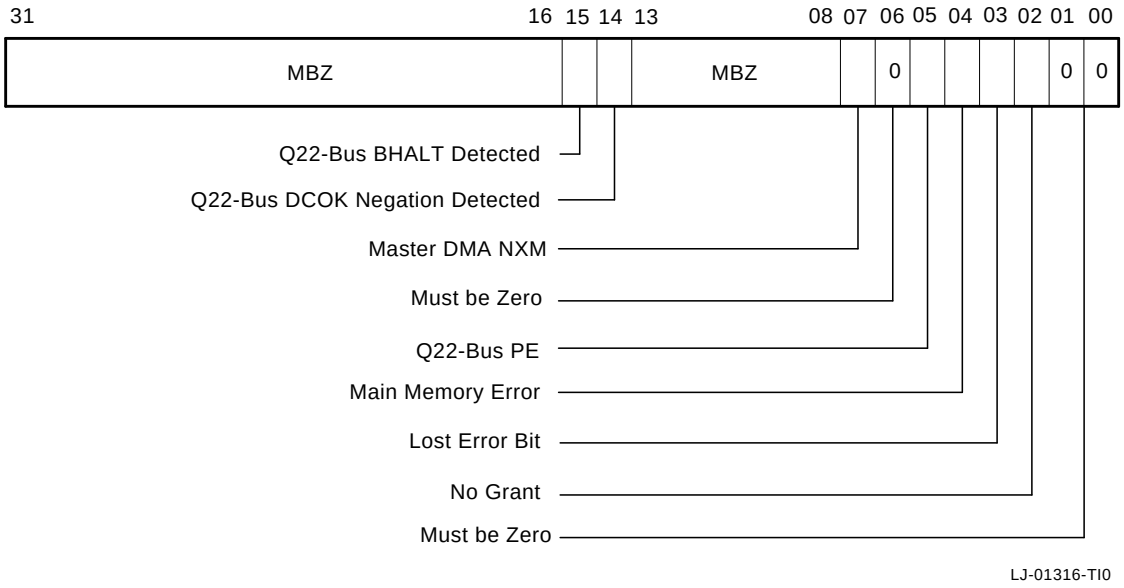


Table 9–6 DMA System Error Register Bit Description

Data Bit	Name	Description
<31:16>	Unused	Read as zeros. Must be written as zeros.
<15>	Q22-BUS BHALT DETECTED	Read/write to clear. This bit is set when the Q22–bus interface detects that the Q22–bus BHALT line was asserted and SCR<14> (BHALT ENABLE) is set. Cleared by writing a one, writes to IPR 55 (IORESET), on powerup, and the negation of DCOK.
<14>	Q22-BUS DCOK NEGATION DETECTED	Read/write to clear. This bit is set when the Q22–bus interface detects the negation of DCOK on the Q22–bus and SCR<7> (ACTION ON DCOK NEGATION) is set. Cleared by writing a one, writes to IPR 55 (IORESET), on powerup, and the negation of DCOK.
<13:8>	Unused	Read as zeros. Must be written as zeros.
<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. Cleared by writing a one, on powerup, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<6>	Unused	Read as zero. Must be written as zero.
<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. During interrupt acknowledge cycles or request read cycles, this bit is not set. Cleared by writing a one, on powerup, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<4>	MAIN MEMORY ERROR	Read/write to clear. This bit is set if an external Q22–bus device or local-miss, global-hit receives a memory error while reading local memory. The IPCR<15> reports the memory error to the external Q22–bus device. Cleared by writing a one, on powerup, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<3>	LOST ERROR	Read/write to clear. This bit indicates that an error address has been lost because of DSER<7,5,4,0> having been previously set and a subsequent error of either type occurs, which would have normally captured an address and set either DSER<7,5,4,0> flag. Cleared by writing a one, on powerup, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<2>	NO GRANT TIMEOUT	Read/write to clear. This bit is set if the Q22–bus does not return a bus grant within 10 ms of the bus request from a CPU demand read cycle or write cycle. During interrupt acknowledge or request read cycles, this bit is not set. Cleared by writing a one, on powerup, by the negation of DCOK, and by writes to IPR 55 (IORESET).
<1:0>	Unused	Read as zeros. Must be written as zeros.

9.7.2 Q22–bus Error Address Register (QBEAR)

The Q22–bus error address register, address 2008 0008_{16}, is a read-only, longword-accessible register that is implemented in the CQBIC chip. Its contents are valid only if DSER<5> (Q22-BUS PARITY ERROR) is set, or if DSER<7> (MASTER DMA NXM) is set.

Reading this register when DSER<5> and DSER<7> are clear 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 on-board CPU, which set DSER<5>, or a master timeout, which set DSER<7>.

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

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

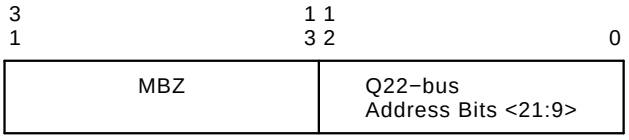
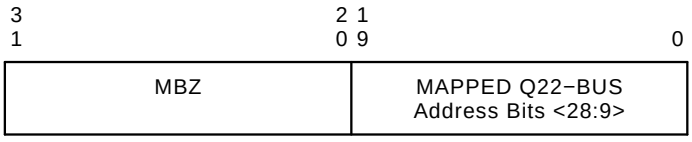


Figure 9–9 DMA Error Address Register (DEAR)



Note

This is a read-only register. If a write is attempted, a hard error (IPL 1D) will be generated.

9.7.3 DMA Error Address Register (DEAR)

The DMA error address register, address 2008 $000C_{16}$, is a read-only, longword-accessible register that is implemented in the CQBIC chip. It contains valid information only when DSER<4> (MAIN MEMORY ERROR) is set .

The DEAR contains the map translated address of the page in local memory that caused a memory error or nonexistent memory error. This occurred during an access by an external device or the Q22–bus interface for the CPU during a local-miss, global-hit transaction or Q22–bus map access.

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

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

Note

This is a read-only register. If a write is attempted, a hard error (IPL 1D) will be generated.

9.8 Q22–bus Interface Error Handling

The Q22–bus interface does not generate or check parity.

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

The Q22–bus interface maintains several timers to prevent incomplete accesses from hanging the system indefinitely. They include: a 10 µs nonexistent memory timer for accesses to the Q22–bus memory and I/O spaces, a 10 µs "no sack" timer for acknowledgement of Q22–bus DMA grants, and a 10 ms "no grant" timer for acquiring the Q22–bus.

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

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

If the Q22–bus interface does not receive an acknowledgement 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 to obtain Q22–bus mastership on a CPU demand read reference, and does not obtain it within 10 ms, DSER<2> is set, and a machine check abort is initiated.

The Q22–bus interface also monitors Q22–bus signals BDAL<17:16> while reading information over the Q22–bus so that parity errors detected by the device being read 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>, but no machine check is generated.

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

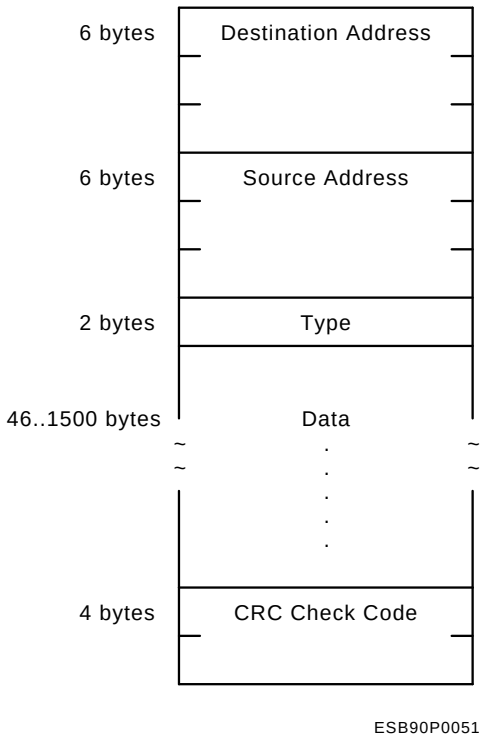
10
Network Interface

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

10.1 Ethernet Overview

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

Figure 10–1 Ethernet Packet Format



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

All nodes on the Ethernet have equal priority. The technique used to control access to the bus is carrier sense, multiple access, with collision detection (CSMA /CD). To access the bus, devices must first wait for the bus to clear (no carrier sensed). Once the bus is clear, all devices that want to access the bus have equal priority (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 and are described below.

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

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 to a group of logically related stations.

Broadcast address: A predefined multicast address that denotes the set of all the stations on the Ethernet.

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

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

10.2 NI Station Address ROM (NISA ROM)

The network interface includes a bytewide, 32-byte, socketted 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 - 2008 $407C_{16}$. Bytes one, three, and four of each longword are defined in the boot diagnostic register (Section 9.1). The second byte of the first six longwords contain the 48-bit network physical address (NPA) of the . The low-order byte in the remaining 26 longwords are used for testing. This address range is read-only. Writes to this address range will

10.3 Programming the SGEC

The operation of the SGEC is controlled by a program in host memory called the port driver. The SGEC and the port driver communicate through two data structures: **network interface command and status registers (NICSRs)** located in the SGEC and mapped in the host I/O address space, and through **descriptor lists and data buffers** (collectively called **host communication area** in host memory.

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

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

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

1. After power on (or reset), verifying the self test completed successfully.
2. Writing NICSRs to set major parameters such as system base register, interrupt vector, address filtering mode, and so on.
3. Creating the transmit and receive lists in memory and writing the NICSRs to identify them to the SGEC.

4. Placing a setup frame in the transmit list, to load the internal reception address filtering table.
5. Starting the reception and transmission processes, placing them in the RUNNING state.
6. Waiting for SGEC interrupts. NICSR5 contains all the global interrupt status bits.
7. Remedying the suspension cause, if either of the reception or transmission processes enter the SUSPENDED state.
8. Issuing a **Tx Poll Demand** command, to return the transmission process to the RUNNING state. In addition, to remedy the reception process suspension cause, an Rx Poll Demand could be issued to return the reception process to the RUNNING state.

   If the Rx Poll Demand is not issued, the reception process will return to the RUNNING state when the SGEC receives the next recognized incoming frame.

The following sections contain detailed programming and state transition information.

10.3.1 Command and Status Registers

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

10.3.2 Host Access to NICSRs

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

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

Table 10–1 Bit Access Modes

Bit Marked	Meaning
0	Reserved for future expansion - ignored on write, read as "0"
1	Reserved for future expansion - ignored on write, read as "1"
R	Read-only, ignored on write
R/W	Read or write
W	Write-only, unpredictable on read
R/W1	Read, or clear by writing a "1" - writing with a "0" has no effect

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

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

The group in which the NICSR is part, determines the way the host will access it.

10.3.2.1 Physical NICSRs

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

10.3.2.2 Virtual NICSRs

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

NICSR5<DN> is used to synchronize access to the virtual NICSRs: after the first virtual NICSR access, the SGEC deasserts NICSR5<DN> until it will complete the action.

Note

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

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

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

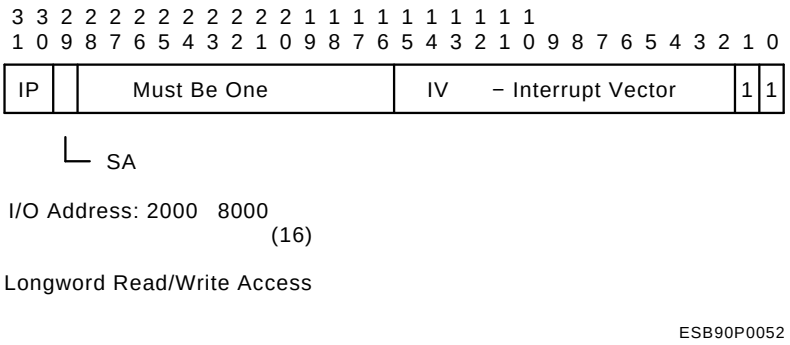
10.3.2.2.2 NICSR Read To read a virtual NICSR, the host takes the following actions:

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

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

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

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



Note

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

1. Write NICSR0.
2. Read NICSR0.
3. Compare value read to value written. If values do not match, repeat the procedure starting with step 1.
4. Read NICSR5 and examine NICSR5<ME> for pending parity interrupt. Should an interrupt be pending, write NICSR5 to clear it.

Table 10–2 NICSR0 Bits

Bit	Name	Access	Description
<31:30>	IP	R/W	Interrupt priority - the VAX interrupt priority level to which the SGEC will respond. IP: 00 → IPL (hex): 14 IP: 01 → IPL (hex): 15 IP: 10 → IPL (hex): 16 IP: 11 → IPL (hex): 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 be 14_{16} for the KA690.
<29>	SA	R/W	Sync/Async - This bit determines the SGEC operating mode when it is the bus master. When set, the SGEC will operate as a synchronous device and when clear, the SGEC will operate as an asynchronous device.
<15:00>	IV	R/W	Interrupt vector - During an interrupt acknowledge cycle for an SGEC interrupt, this is 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.

Table 10–3 NICSR0 Access

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

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

Figure 10–3 Polling Demand (NICSR1)

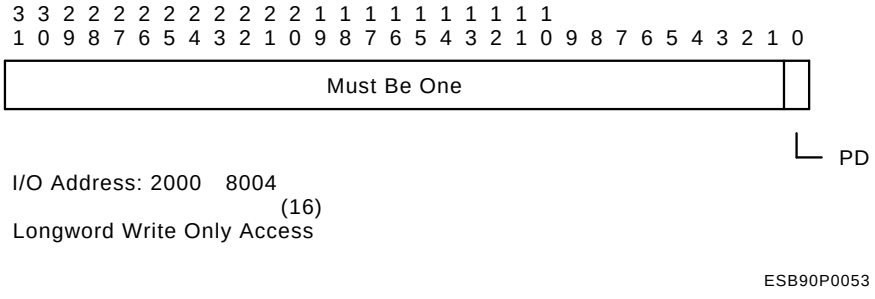


Table 10–4 NICSR1 Bits

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

Table 10–5 NICSR1 Access

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

10.3.4 Receive Polling Demand (NICSR2)

Figure 10–4 NICSR2 Format

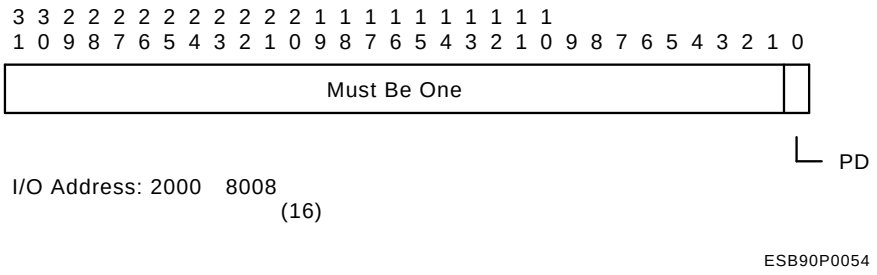


Table 10–6 NICSR2 Bits

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

Table 10–7 NICSR2 Access

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

10.3.5 Descriptor List Addresses (NICSR3, NICSR4)

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

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

Note

For best performance, 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** (see Section 10.3.7), or the respective process will remain in the STOPPED state. New list head addresses are only acceptable while the respective process is in the STOPPED or SUSPENDED state. Addresses written while the respective process is in the RUNNING state are *ignored* and *discarded*.

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

Figure 10–5 Descriptor List Addresses Format

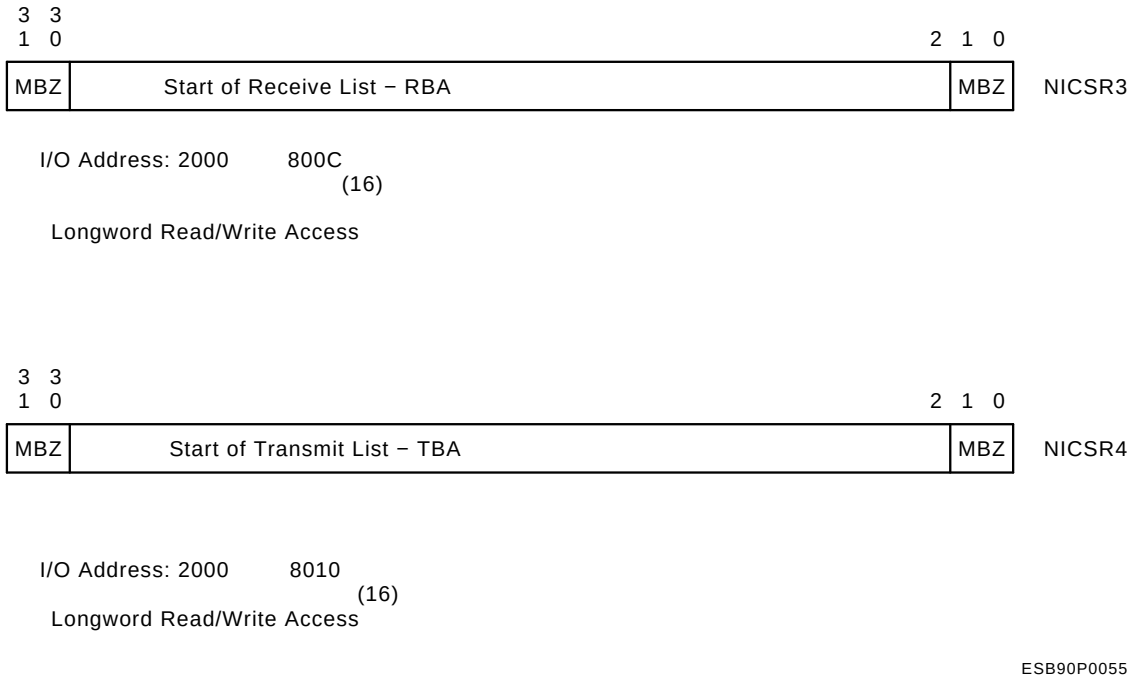


Table 10–8 Descriptor List Addresses Bits

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

Note

The descriptor lists must be longword-aligned.

Table 10–9 NICSR3 Access

Value after **RESET**:	Unpredictable
Read access rules:	None
Write access rules:	Rx process STOPPED or SUSPENDED

Table 10–10 NICSR4 Access

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

After 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**.

10.3.6 Status Register (NICSR5)

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

Figure 10–6 NICSR5 Bits

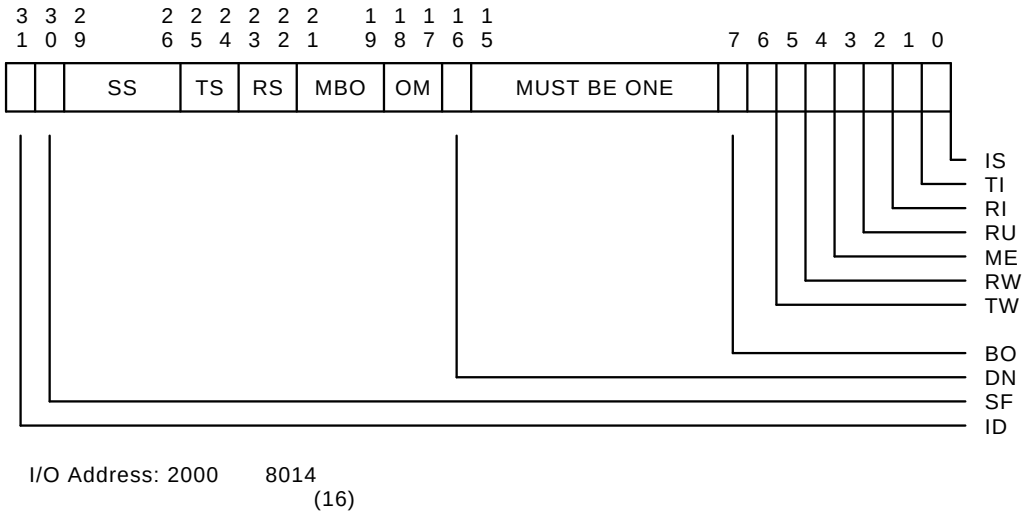


Table 10–11 NICSR5 Bits

Bit	Name	Access	Description
<31>	ID	R	Initialization done - When set, indicates the SGEC has completed the initialization (reset and self-test) sequences, and is ready for further commands. When clear, indicates the SGEC is performing the initialization sequence and ignoring all commands. After the initialization sequence completes, the transmission and reception processes are in the STOPPED state.

(continued on next page)

Table 10–11 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<30>	SF	R	Self-test failed - When set, indicates the SGEC self-test has failed. The self-test completion code bits indicate the failure type.
<29:26>	SS	R	Self-test status - The self-test completion code, according to the following table, 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

Information

Self-test takes 25 ms to complete after hardware or software RESET.

<25:24>	TS	R	Transmission process state - Indicates the current state of the transmission process, as follows:

Value	Meaning
00	STOPPED
01	RUNNING
10	SUSPENDED

Section 10.3.18.5 explains the transmission process operation and state transitions.

<23:22>	RS	R	Reception process state - Indicates the current state of the reception process, as follows:

Value	Meaning
00	STOPPED
01	RUNNING
10	SUSPENDED

Section 10.3.18.4 explains the reception process operation and state transitions.

(continued on next page)

Table 10–11 (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 10.3.18.6 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 also looped back to the receive list, subject to address filtering. Section 10.3.18.6 explains this mode of operation.
11	Reserved for diagnostics

<16>	DN	R	Done - When set, indicates the SGEC has completed a requested virtual NICSR access. After a reset, this bit is set.
<15:8>	MBO	–	Must be one - This field is reserved. Writes are ignored, read as one.
<7>	BO	R/W1	Boot_Message - When set, indicates that the SGEC has detected a boot_message on the serial line and has set the external pin BOOT_L.
<6>	TW	R/W1	Transmit watchdog timer interrupt - When set, indicates the transmit watchdog timer has timed out, indicating the SGEC transmitter was babbling. The transmission process is *aborted* and placed in the STOPPED state. (Also reported into the Tx descriptor status TDES0<TO> flag).
<5>	RW	R/W1	Receive watchdog timer interrupt - When set, indicates the receive watchdog timer has timed out, indicating that some other node is babbling on the network. Current frame reception is aborted, and RDES0<LE> and RDES0<LS> will be set. Bit NICSR5<RI> will also set. The reception process remains in the RUNNING state.

(continued on next page)

Table 10–11 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<4>	ME	R/W1	Memory error - Is set when any of the followings occur: • SGEC is the CP-bus master and the ERR_L pin is asserted by external logic (generally indicative of a memory problem). • Parity error detected on 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, it is mandatory that the port driver issue a Reset command and rewrite all NICSRs.
<3>	RU	R/W1	Receive buffer unavailable - When set, indicates that the next descriptor on the receive list is owned by the host and could not be acquired by the SGEC. The reception process is placed in the SUSPENDED state. Section 10.3.18.4 explains the reception process state transitions. Once set by the SGEC, this bit will not be set again until the SGEC encounters a descriptor it cannot acquire. To resume processing receive descriptors, the host must flip the ownership bit of the descriptor and can issue the Rx Poll Demand command. If no Rx Poll Demand is issued, the reception process resumes when the next recognized incoming frame is received.
<2>	RI	R/W1	Receive interrupt - When set, indicates that a frame has been placed on the receive list. Frame-specific status information was posted in the descriptor. The reception process remains in the RUNNING state.

(continued on next page)

Table 10–11 (Cont.) NICSR5 Bits

Bit	Name	Access	Description
<1>	TI	R/W1	Transmit interrupt - When set, indicates one of the following: • Either all the frames in the transmit list have been transmitted (next descriptor owned by the host), or a frame transmission was aborted due to a locally induced error. The port driver must scan the list of descriptors to determine the exact cause. The transmission process is placed in the SUSPENDED state. Section 10.3.18.5 explains the transmission process state transitions. To resume processing transmit descriptors, the port driver must issue the Poll Demand command. • A frame transmission completed, and TDES1<IC> was set. The transmission process remains in the RUNNING state, unless the next descriptor is owned by the host or the frame transmission aborted due to an error. In the latter cases, the transmission process is placed in the SUSPENDED state.
<0>	IS	R/W1	Interrupt summary - The logical "OR" of NICSR5 bits 1 through 6.

Table 10–13 NICSR5 Access

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

10.3.6.1 NICSR5 Status Report

The status register NICSR5 is split into two words:

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

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

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

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

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

The SGEC 16-bit internal processor updates the 32-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 first the NICSR5 low word to get the interrupt status, then the NICSR high word to get the related process state. For example, <TI> interrupt with <TS> = SUSPENDED reports an end of transmission due to a Tx descriptor unavailable.

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

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

10.3.7 Command and Mode Register (NICSR6)

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

Figure 10–7 NICSR6 Format

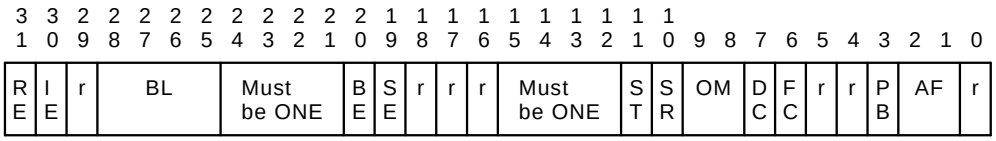


Table 10–14 NICSR6 Bits

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

(continued on next page)

Table 10–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<29>	r	–	Reserved
<28:25>	BL	R/W	Burst limit mode - Specifies the maximum number of longwords to be transferred in a single DMA burst on the host bus. When NICSR6<SE> is cleared, permissible values are 1,2,4,8. When SE is set, the only permissible values are 1 and 4; a value of 2 or 8 is respectively forced to 1 or 4. After initialization, the burst limit is set to 1.
<24:21>	MBO	–	This field is reserved. Writes are ignored, read as one.
<20>	BE	R/W	Boot_message enable mode - When set, enables the boot_message recognition. When the SGEC recognizes an incoming boot message on the serial line, NICSR5<BO> is set and the external pin BOOT_L is asserted for a duration of 6*Tcycles (of the host clock).
<19>	SE	R/W	Single_cycle enable mode - When set, the SGEC transfers only a single longword or an octaword in a single DMA burst on the host bus.
<18:12>	MBO	–	Must be one. This field is reserved. Writes are ignored, read as one.
<11>	ST	R/W	Start/Stop Transmission command - When set, the transmission process is placed in the RUNNING state, the SGEC checks the transmit list at the *current* position for a frame to transmit (the address set by *NICSR4* or the position retained when the Tx process was previously stopped). If it does not find a frame to transmit, the transmission process enters the SUSPENDED state. The Start Transmission command is honored only when the transmission process is in the STOPPED state. *The first time this command is issued, an additional requirement is that NICSR4 has already been written to, or the transmission process will remain in the STOPPED state.* When cleared, the transmission process is placed in the STOPPED state after completing transmission of the current frame. The next descriptor position in the transmit list is saved, and becomes the current position after transmission is restarted. The Stop Transmission command is honored only when the transmission process is in the RUNNING or SUSPENDED states. Refer to Section 10.3.18.5 for more information.

(continued on next page)

Table 10–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<10>	SR	R/W	Start/Stop Reception command - When set, the reception process is placed in the RUNNING state, and the SGEC attempts 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 Rx process was previously stopped). If no descriptor can be acquired, the reception process enters the SUSPENDED state. The Start Reception command is honored only when the reception process is in the STOPPED state. *The first time this command is issued, an additional requirement is that NICSR3 has already been written to, or the reception process will remain in the STOPPED state.* When cleared, the reception process is placed in the STOPPED state after completing reception of the current frame. The next descriptor position in the receive list is saved, and becomes the *current* position after reception is restarted. The Stop Reception command is honored only when the reception process is in the RUNNING or SUSPENDED states. Refer to Section 10.3.18.4 for more information.

(continued on next page)

Table 10–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<9:8>	OM	R/W	Operating mode - These bits determine the SGEC main operating mode.

Value	Meaning
00	Normal operating mode.
01	Internal loopback - The SGEC will loop back buffers from the transmit list. The data will be passed from the transmit logic back to the receive logic. The receive logic will treat the looped frame as it would any other frame, and subject it to the address filtering and validity check process.
10	External loopback - The SGEC transmits normally and in addition, will enable its receive logic to its own transmissions. The receive logic will treat the looped frame as it would any other frame, and subject it to the address filtering and validity check process.
11	Reserved for diagnostic

(continued on next page)

Table 10–14 (Cont.) NICSR6 Bits

Bit	Name	Access	Description
<7>	DC	R/W	Disable data chaining mode - When set, no data chaining will occur in reception; frames, longer than the current receive buffer, will be 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 clear, frames too long for the current receive buffer will be 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 FC is set, a collision will be forced during the next transmission attempt. This will result in 16 transmission attempts with excessive collision reported in the transmit descriptor.
<5:4>	MBO		Must be one. This field is reserved. Writes are ignored, read as one.
<3>	PB	R/W	Pass bad frames mode - When this bit is set, the SGEC will pass frames that have been damaged by collisions or are too short due to premature reception termination. Both events should have occurred within the collision window (64 bytes), or other errors will be reported. When clear, these frames will be discarded and never show up in the host receive buffers. **Note** Pass bad frames is subject the address filtering mode. For example, to monitor the network, this mode must be set together with the promiscuous 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 one. This field is reserved. Writes are ignored, read as one.

Value	Meaning
00	Normal - Incoming frames will be filtered according to the values of the <HP> and <IF> bits of the setup frame descriptor.
01	Promiscuous - All incoming frames will be passed to the host, regardless of the <HP> bit value.
10	All multicast - All incoming frames with multicast address destinations will be passed to the host. Incoming frames with physical address destinations will be filtered according to the <HP> bit value.
11	Unused - Reserved.

Table 10–15 NICSR6 Access

Value after **RESET**:	83E0F000 hex or 03E0F000 hex
Read access rules:	None
Write access rules:	
<RE, IE, BE>	Unconditional
<BL, SE, OM>	Rx and Tx processes STOPPED
<FC>	Rx and Tx processes STOPPED, Internal_Loopback mode
<DC, PB, AF>	Rx STOPPED
Start_Receive <SR>=1	Rx STOPPED and NICSR3 initialized
Start_Transmit <ST>=1	Tx STOPPED and NICSR4 initialized
Stop_Receive <SR>=0	Rx RUNNING or SUSPENDED
Stop_Transmit <ST>=0	Tx RUNNING or SUSPENDED

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

10.3.8 System Base Register (NICSR7)

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

Figure 10–8 NICSR7 Format

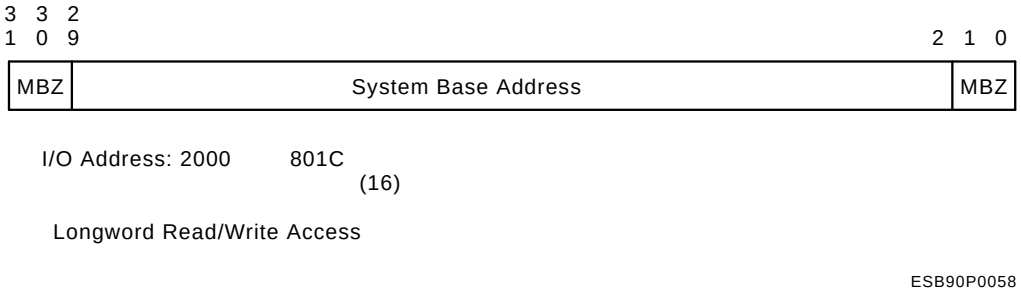


Table 10–16 NICSR7 Bits

Bit	Name	Access	Description
<31:30>	MBZ	–	Must be zero. Read as zero. Writes are ignored.

(continued on next page)

Table 10–16 (Cont.) NICSR7 Bits

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

Table 10–17 NICSR7 Access

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

10.3.9 Reserved Register (NICSR8)

This entire register is reserved.

10.3.10 Watchdog Timers (NICSR9)

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

Figure 10–9 NICSR9 Format

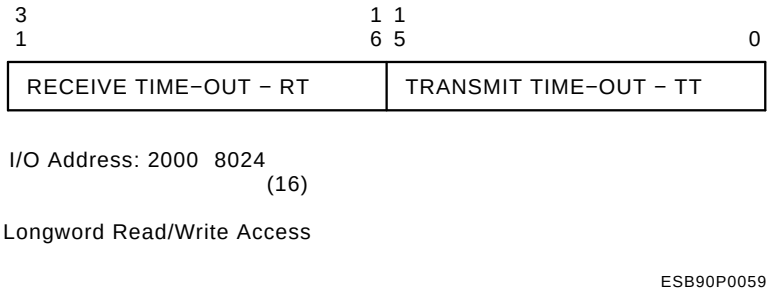


Table 10–18 NICSR9 Bits

Bit	Name	Access	Description
<31:16>	RT	R/W	Receive watchdog 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 zero, it will never time out. The value of RT is an unsigned integer. With a 10 MHz serial clock, this provides a range of 72 µs to 100 ms. The default value is 1250 corresponding to 2 ms. The Rx watchdog timer is programmed only while the reception process is in the STOPPED state. **Note** An Rx watchdog value between 1 and 44 is forced to the minimum time_out value of 45 (72 µs).
<15:00>	TT	R/W	Transmit watchdog timeout - The transmit watchdog timer protects the network against babbling SGEC transmissions, on top of any such circuitry present in tranceivers. If the transmitter stays on for $TT * 16$ cycles of the serial clock, the SGEC will cut off the transmitter and set the NICSR5<TW> bit. If the timer is set to zero, it will never time out. The value of TT is an unsigned integer. With a 10 MHz serial clock, this provides a range of 72 µs to 100 ms. The default value is 1250 corresponding to 2 ms. The Tx watchdog timer is programmed only while the transmission process is in the STOPPED state. **Note** A Tx watchdog value between 1 and 44 is forced to the minimum time_out value of 45 (72 µs).

Table 10–19 NICSR9 Access

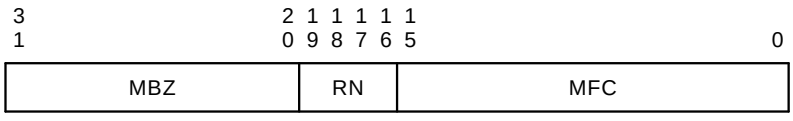
Value after **RESET**:	00000000 hex
Read access rules:	None
Write access rules:	
Rx watchdog timer	Rx process STOPPED
Tx watchdog timer	Tx process STOPPED

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

10.3.11 Revision Number and Missed Frame Count (NICSR10)

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

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



I/O Address: 2000 802C
(16)

Longword Read Only Access

ESB90P0060

Table 10–20 NICSR10 Bits

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

Table 10–21 NICSR10 Access

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

10.3.12 Boot Message (NICSR11, 12, 13)

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

Figure 10–11 Boot Message

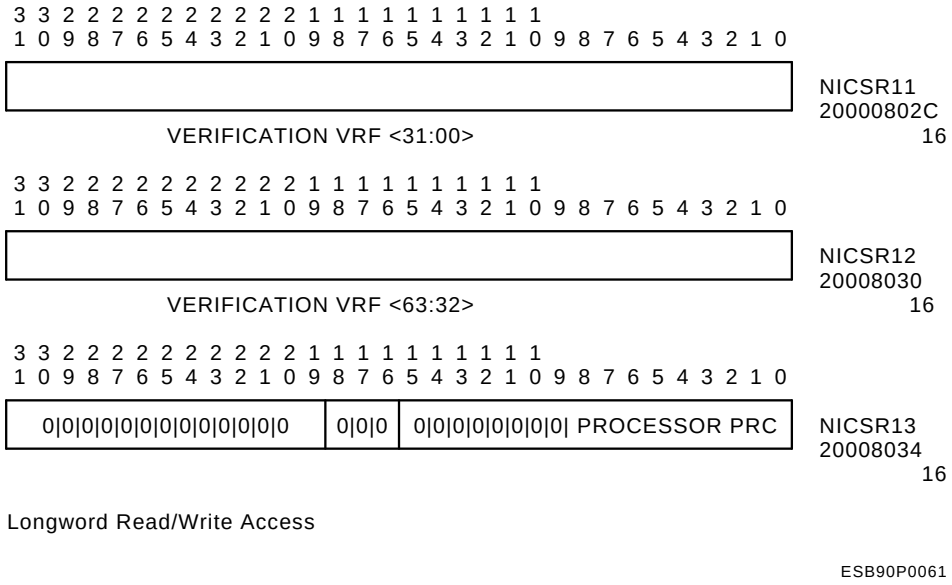


Table 10–22 NICSR11,12,13 Bits

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.

Table 10–23 NICSR11,12,13 Access

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

10.3.13 Diagnostic Registers (NICSR14, 15)

These registers are reserved for diagnostic features.

10.3.13.1 Diagnostic Breakpoint Address Register (NICSR14)

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

Figure 10–12 NICSR14 Format

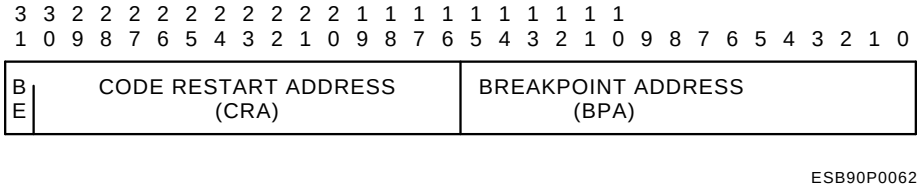


Table 10–24 NICSR14 Bits

Bit	Name	Type	Description
<31>	BE	R/W	When set, breakpoint enabled.
<30:16>	CRA	R/W	Code restart address - The first address in the internal RAM where the internal processor will jump after a breakpoint occurred.
<15:0>	BPA	R/W	Breakpoint address - The internal processor address at which the program will halt and jump to the RAM loaded code.

Note

This register, in conjunction with the diagnostic descriptors, allows software patches.

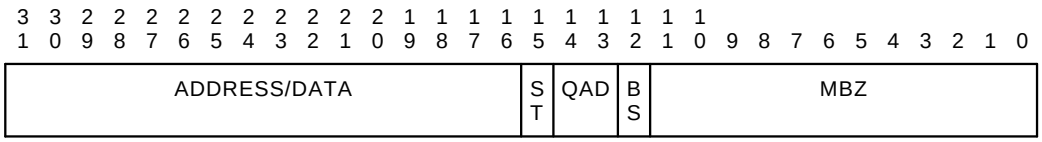
Table 10–25 NICSR14 Access

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

10.3.13.2 Monitor Command Register (NICSR15)

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

Figure 10–13 NICSR15 Format



ESB90P0063

Table 10–26 NICSR15 Bits

Bit	Name	Type	Description
<31:16>	ADDR/DATA	R/W	Before the "Examine" cycle, it points to the location to be read. Three cycles after the assertion of <ST>, it contains the read data.
<15>	ST	W	Start read - When set, starts the "Examine" cycle: the data addressed by CSR<31:16> is fetched and stored into the same register field. Reset by hardware at the end of operation.
<14:13>	QAD	W	Quad selects bits - These bits define the specific four bits of the internal Data_bus or Address_ bus, which are monitored on the external test pins BM_L/TEST<3:0>. 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. Meaningful only in test mode (TSM=1).
<11:0>	MBZ	–	Must be zero.

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>

Table 10–27 NICSR15 Access

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

10.3.14 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, which are also resident in host memory.

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

Note

The SGEC first reads the descriptors, ignoring all unused bits regardless of their state. The only word the SGEC writes back is the first word (xDES0) of each descriptor. Unused bits in xDES0 will be written as "0." Unused bits in xDES1 - xDES3 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, through NICSR6<DC>. Data buffers reside in VAX memory space, either physical or virtual.

Note

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

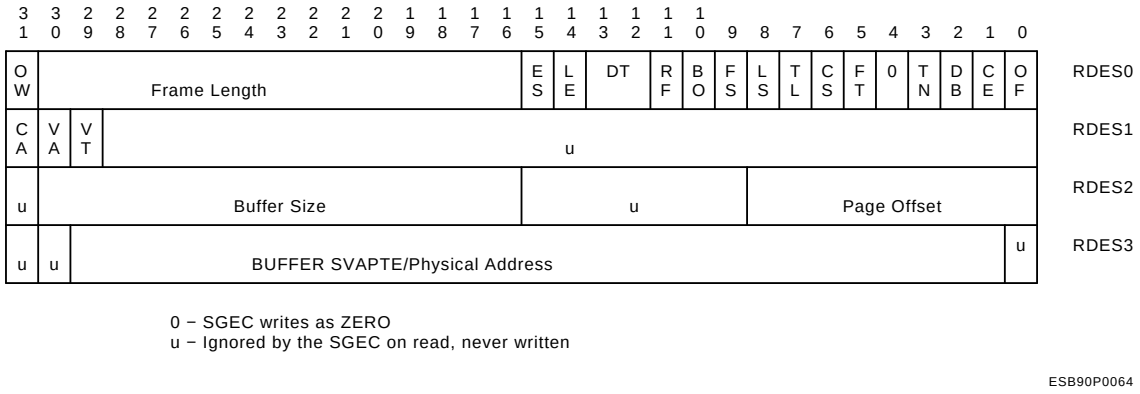
Note

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

10.3.15 Receive Descriptors

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

Figure 10–14 Receive Descriptor Format



10.3.15.1 RDES0 Word

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

Table 10–28 RDES0 Bits

Bit	Name	Description
<31>	OW	Own bit - When set, indicates the descriptor is owned by the SGEC. When cleared, indicates the descriptor is owned by the host. The SGEC clears this bit upon completing processing of the descriptor and its associated buffer.
<30:16>	FL	Frame length - The length in bytes of the received frame. Meaningless if RDES0<LE> is set.
<15>	ES	Error summary - The logical “OR” of RDES0 bits OF, CE, TN, CS, TL, LE, and RF.
<14>	LE	Length error - When set, indicates a frame truncation caused by one of the following: • The frame segment does not fit within the current buffer and the SGEC does not own the next descriptor. The frame is truncated. • The receive watchdog timer expired. NICSR5<RW> is also set.

(continued on next page)

Table 10–28 (Cont.) RDES0 Bits

Bit	Name	Description
<13:12>	DT	Data type - Indicates the type of frame the buffer contains, according to the following table:

Value	Meaning
00	Serial received frame
01	Internally looped back frame
10	Externally looped back frame, serial received frame [1]

<11>	RF	Runt frame - When set, indicates this frame was damaged by a collision or premature termination before the collision window had passed. Runt frames will be passed on to the host only if (NICSR6<PB>) is set. Meaningless if RDES0<OF> is set.
<10>	BO	Buffer overflow - When set, indicates that the frame has been truncated due to a buffer too small to fit the frame size. This bit may be set only if data chaining is disabled (NICSR6<DC> = 1).
<09>	FS	First segment - When set, indicates this buffer contains the first segment of a frame.
<08>	LS	Last segment - When set, indicates this buffer contains the last segment of a frame and status information is valid.
<07>	TL	Frame too long - When set, indicates the frame length exceeds the maximum Ethernet specified size of 1518 bytes.

Note

Frame too long is only a frame length indication and does not cause any frame truncation.

<06>	CS	Collision seen - When set, indicates the frame was damaged by a collision that occurred after the 64 bytes following the SFD.
<05>	FT	Frame type - When set, indicates the frame is an Ethernet type frame (Frame Length_Field > 1500). When clear, indicates the frame is an IEEE 802.3 type frame. Meaningless for runt frames < 14 bytes.
<04>	0	Zero. SGEC writes as zero.
<03>	TN	Translation not valid - When set, indicates that a translation error occurred when the SGEC was translating a VAX virtual buffer address. It will only set if RDES1<VA> was set. The reception process remains in the RUNNING state and attempts to acquire the next descriptor.

[1]The SGEC does not differentiate between the loopback and the serial received frames. Therefore, this information is global and reflects only NICSR6<OM>.

(continued on next page)

Table 10–28 (Cont.) RDES0 Bits

Bit	Name	Description
<02>	DB	Dribbling bits - When set, indicates the frame contained a noninteger multiple of eight bits. This error will be reported only if the number of dribbling bits in the last byte is greater than two. Meaningless if RDES0<CS> or RDES0<RF> is 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 six dribbling bits will have this bit set, but if <CE> (or another error indicator) is not set, these frames should be considered valid:**

CE	DB	Error
0	0	None
0	1	None
1	0	CRC error
1	1	Alignment error

<01>	CE	CRC Error - When set, indicates that a CRC error has occurred on the received frame.
<00>	OF	Overflow - When set, indicates received data in this descriptor's buffer was corrupted due to internal FIFO overflow. This will generally occur if SGEC DMA requests are not granted before the internal receive FIFO fills up.

10.3.15.2 RDES1 Word

Table 10–29 RDES1 Bits

Bit	Name	Descriptor
<31>	CA	Chain address - When set, RDES3 is interpreted as another descriptor's VAX physical address. This allows the SGEC to process multiple, non-contiguous descriptor lists and explicitly "chain" the lists. Note that contiguous descriptors are implicitly chained. In contrast to what is done for an Rx 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 infinite loop, a chain descriptor pointing back to itself, is seen as **owned by the host**, regardless of the ownership bit state.

(continued on next page)

Table 10–29 (Cont.) RDES1 Bits

Bit	Name	Descriptor
<30>	VA	Virtual addressing - When set, RDES3 is interpreted as a virtual address. The type of virtual address translation is determined by the RDES1<VT> bit. The SGEC uses RDES3 and RDES2<Page Offset> to perform a VAX virtual address translation process to obtain the physical address of the buffer. When clear, RDES3 is interpreted as the actual physical address of the buffer:

VA	VT	Addressing Mode
0	x	Physical
1	0	Virtual - SVAPTE
1	1	Virtual - PAPTE

<29>	VT	Virtual type - In case of virtual addressing (RDES1<VA> = 1), indicates the type of virtual address translation. When set, the buffer address RDES3 is interpreted as an SVAPTE (system virtual address of the page table entry). When clear, the buffer address is interpreted as a PAPTE (physical address of the page table entry). Meaningful only if RDES1<VA> is set.
<28:0>	u	Unused. Ignored by the SGEC on reads. Never written.

10.3.15.3 RDES2 Word

Table 10–30 RDES2 Bits

Bit	Name	Descriptor
<31>	u	Unused. Ignored by the SGEC on reads. Never written.
<30:16>	BS	Buffer size - The size, in bytes, of the data buffer.

Note

Receive buffer size must be an even number of bytes.

<15:9>	u	Unused. Ignored by the SGEC on reads. Never written.
<08:00>	PO	Page offset - The byte offset of the buffer within the page. Only meaningful if RDES1<VA> is set.

Note

Receive buffers must be word-aligned.

10.3.15.4 RDES3 Word

Table 10–31 RDES3 Bits

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

10.3.15.5 Receive Descriptor Status Validity

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

Table 10–32 Receive Descriptor Status Validity

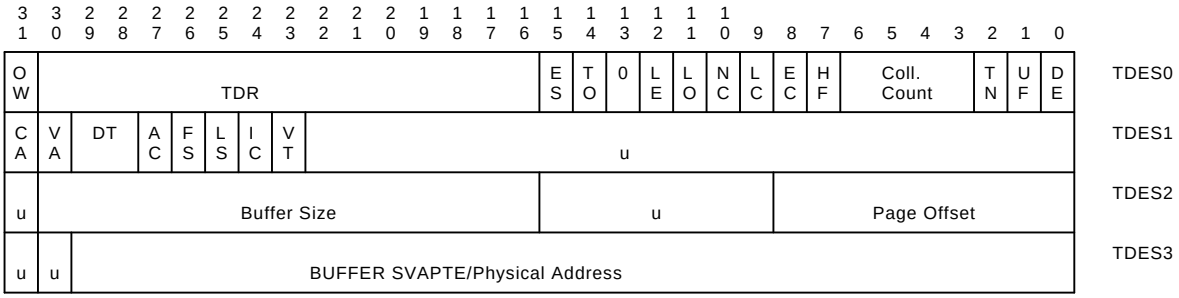
Reception Status	RF	TL	CS	FT	DB	CE	Rx Status Report (ES,LE,BO,DT,FS,LS,FL,TN,OF)
Overflow	X	V	X	V	X	X	V
Collision after 512 bits	V	V	V	V	X	X	V
Runt frame	V	V	V	V	X	X	V
Runt frame < 14 bytes	V	V	V	X	X	X	V
Watchdog timeout	V	V	X	V	X	X	V

V - Valid
X - Meaningless

10.3.16 Transmit Descriptors

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

Figure 10–15 Transmit Descriptor Format



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

ESB90P0065

10.3.16.1 TDES0 Word

TDES0 word contains transmitted frame status and descriptor ownership information.

Table 10–33 TDES0 Bits

Bit	Name	Description
<31>	OW	Own bit - When set, indicates the descriptor is owned by the SGEC. When cleared, indicates the descriptor is owned by the host. The SGEC clears this bit upon completing processing of the descriptor and its associated buffer.
<29:16>	TDR	Time domain reflectometer - This is a count of bit time and is useful for locating a fault on the cable using the velocity of propagation on the cable. Only valid if TDES0<EC> is also set. Two excessive collisions in a row and with the same or similar (within 20) TDR values indicate a possible cable open.
<15>	ES	Error summary - The logical “OR” of UF, TN, EC, LC, NC, LO, LE, and TO.
<14>	TO	Transmit watchdog timeout - When set, indicates the transmit watchdog timer has timed out, indicating the SGEC transmitter was babbling. The interrupt NICSR5<TW> is set and the transmission process is *aborted* and placed in the STOPPED state.
<13>	MBZ	Must be zero. SGEC writes as zero.
<12>	LE	Length error - When set, indicates one of the following: • Descriptor unavailable (owned by the host) in the middle of data chained descriptors. • Zero length buffer in the middle of data chained descriptors. • Setup or diagnostic descriptors (Data type TDES1<DT> <> 0) in the middle of data chained descriptors. • Incorrect order of first_segment TDES1<FS> and last_segment TDES1<LS> descriptors in the descriptor list. • The Transmission process enters the SUSPENDED state and sets NICSR5<TI>.

(continued on next page)

Table 10–33 (Cont.) TDES0 Bits

Bit	Name	Description
<11>	LO	Loss of carrier - When set, indicates loss of carrier during transmission (possible short circuit in the Ethernet cable). Meaningless in internal loopback mode (NICSR5<OM>=1).
<10>	NC	No carrier - When set, indicates the carrier signal from the transceiver was not present during transmission (possible problem in the transceiver or transceiver cable). Meaningless in internal loopback mode (NICSR5<OM>=1).
<09>	LC	Late collision - When set, indicates frame transmission was aborted due to a late collision. Meaningless if TDES0<UF>.
<08>	EC	Excessive collisions - When set, indicates that the transmission was aborted because 16 successive collisions occurred while attempting to transmit the current frame.
<07>	HF	Heartbeat fail - When set, indicates heartbeat collision check failure (the transceiver failed to return a collision pulse as a check after the transmission). Some tranceivers do not generate heartbeat, and will always have this bit set. If the transceiver does support it, it indicates transceiver failure. Meaningless if TDES0<UF>.
<06:03>	CC	Collision count - A 4-bit counter indicating the number of collisions that occurred before the transmission attempt succeeded or failed. Meaningless when TDES0<EC> is also set.
<02>	TN	Translation not valid - When set, indicates that a translation error occurred when the SGEC was translating a VAX virtual buffer address. It may only set if TDES1<VA> was set. The transmission process enters the SUSPENDED state and sets NICSR5<TI>.
<01>	UF	Underflow error - When set, indicates that the transmitter has truncated a message due to data late from memory. UF indicates that the SGEC encountered an empty transmit FIFO while in the midst of transmitting a frame. The transmission process enters the SUSPENDED state and sets NICSR5<TI>.
<00>	DE	Deferred - When set, indicates that the SGEC had to defer while trying to transmit a frame. This condition occurs if the channel is busy when the SGEC is ready to transmit.

10.3.16.2 TDES1 Word

Table 10–34 TDES1 Bits

Bit	Name	Descriptor
<31>	CA	Chain address - When set, TDES3 is interpreted as another descriptor's VAX physical address. This allows the SGEC to process multiple, non-contiguous descriptor lists and explicitly "chain" the lists. Note that contiguous descriptors are implicitly chained. In contrast to what is done for an Rx buffer descriptor, the SGEC clears neither the ownership bit TDES0<OW> or one of the other bits of TDES0 of the chain descriptor after processing. To protect against infinite loop, a chain descriptor pointing back to itself, is seen as **owned by the host**, regardless of the ownership bit state.

(continued on next page)

Table 10–34 (Cont.) TDES1 Bits

Bit	Name	Descriptor
<30>	VA	Virtual addressing - When set, TDES3 is interpreted as a virtual address. The type of virtual address translation is determined by the TDES1<VT> bit. The SGEC uses TDES3 and TDES2<Page Offset> to perform a VAX virtual address translation process to obtain the physical address of the buffer. When clear, TDES3 is interpreted as the actual physical address of the buffer:

VA	VT	Addressing Mode
0	x	Physical
1	0	Virtual - SVAPTE
1	1	Virtual - PAPTE

Bit	Name	Descriptor
<29:28>	DT	Data type - 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 10.3.17.
11	Diagnostic frame

Bit	Name	Descriptor
<27>	AC	Add CRC disable - When set, the SGEC will not append the CRC to the end of the transmitted frame. To take effect, this bit must be set in the descriptor where FS is set.

Note

If the transmitted frame is shorter than 64 bytes, the SGEC will add the padding field and the CRC regardless of the <AC> flag.

Bit	Name	Descriptor
<26>	FS	First segment - When set, indicates the buffer contains the first segment of a frame.
<25>	LS	Last segment - When set, indicates the buffer contains the last segment of a frame.
<24>	IC	Interrupt on completion - When set, the SGEC will set NICSR5<TI> after this frame has been transmitted. To take effect, this bit must be set in the descriptor where LS is set.
<23>	VT	Virtual type - In case of virtual addressing (TDES1<VA> = 1), indicates the type of virtual address translation. When set, the buffer address TDES3 is interpreted as an SVAPTE (system virtual address of the page table entry). When clear, the buffer address is interpreted as a PAPTE (physical address of the page table entry). Meaningful only if TDES1<VA> is set.
<22:0>	u	Unused. Ignored by the SGEC on reads. Never written.

10.3.16.3 TDES2 Word

Table 10–35 TDES2 Bits

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

10.3.16.4 TDES3 Word

Table 10–36 TDES3 Bits

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

10.3.16.5 Transmit Descriptor Status Validity

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

Table 10–37 Transmit Descriptor Status Validity

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

V - Valid
X - Meaningless

10.3.17 Setup Frame

A setup frame defines SGEC Ethernet destination addresses. These addresses will be used to filter all incoming frames. The setup frame is *never* transmitted over the Ethernet, nor looped back to the receive list. While the setup frame is being processed, the receiver logic will temporarily disengage 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

10.3.17.1 First Setup Frame

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

Note

The self-test completes with the SGEC address filtering table fully set to "0." A reception process started without loading a setup frame will reject all the incoming frames except those with a destination physical address = 000000h.

10.3.17.2 Subsequent Setup Frame

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

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

10.3.17.3 Setup Frame Descriptor

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

Figure 10–16 Setup Frame Descriptor Format

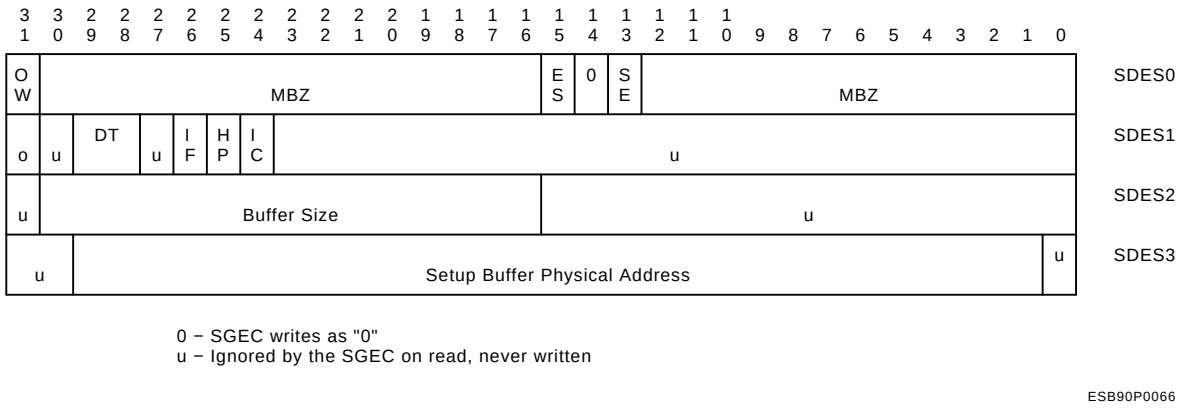


Table 10–38 Setup Frame Descriptor Bits

Word	Bit	Name	Description
SDES0	<13>	SE	Setup error - When set, indicates the setup frame buffer size is not 128 bytes.
	<15>	ES	Error summary - Set when SE is set.
	<31>	OW	Own bit - When set, indicates the descriptor is owned by the SGEC. When cleared, indicates the descriptor is owned by the host. The SGEC clears this bit upon completing processing of the descriptor and its associated buffer.
SDES1	<24>	IC	Interrupt on completion - When set, the SGEC will set NICSR5<TI> after this setup frame has been processed.
	<25>	HP	Hash/perfect filtering mode - When set, the SGEC will interpret the setup frame as a hash table, and do an imperfect address filtering. The imperfect mode is useful when there are more than 16 multicast addresses to listen to. When clear, the SGEC will do a perfect address filter of incoming frames according to the addresses specified in the setup frame.

(continued on next page)

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

Word	Bit	Name	Description
	<26>	IF	Inverse filtering - When set, the SGEC will do an inverse filtering; the SGEC will receive the incoming frames with destination address not matching the perfect addresses, and will reject the frames with destination address matching one of the perfect addresses. Meaningful only for Perfect_filtering (SDES1<HP>=0), while Promiscuous and All_Multicast modes are not selected (NICSR6<AF>=0).
	<29:28>	DT	Data type - Must be 2 to indicate setup frame.
SDES2	<30:16>	BS	Buffer size - Must be 128.
SDES3	<29:1>	PA	Physical address - Physical address of setup buffer. **Note** Setup buffer must be word-aligned.

10.3.17.4 Perfect Filtering Setup Frame Buffer

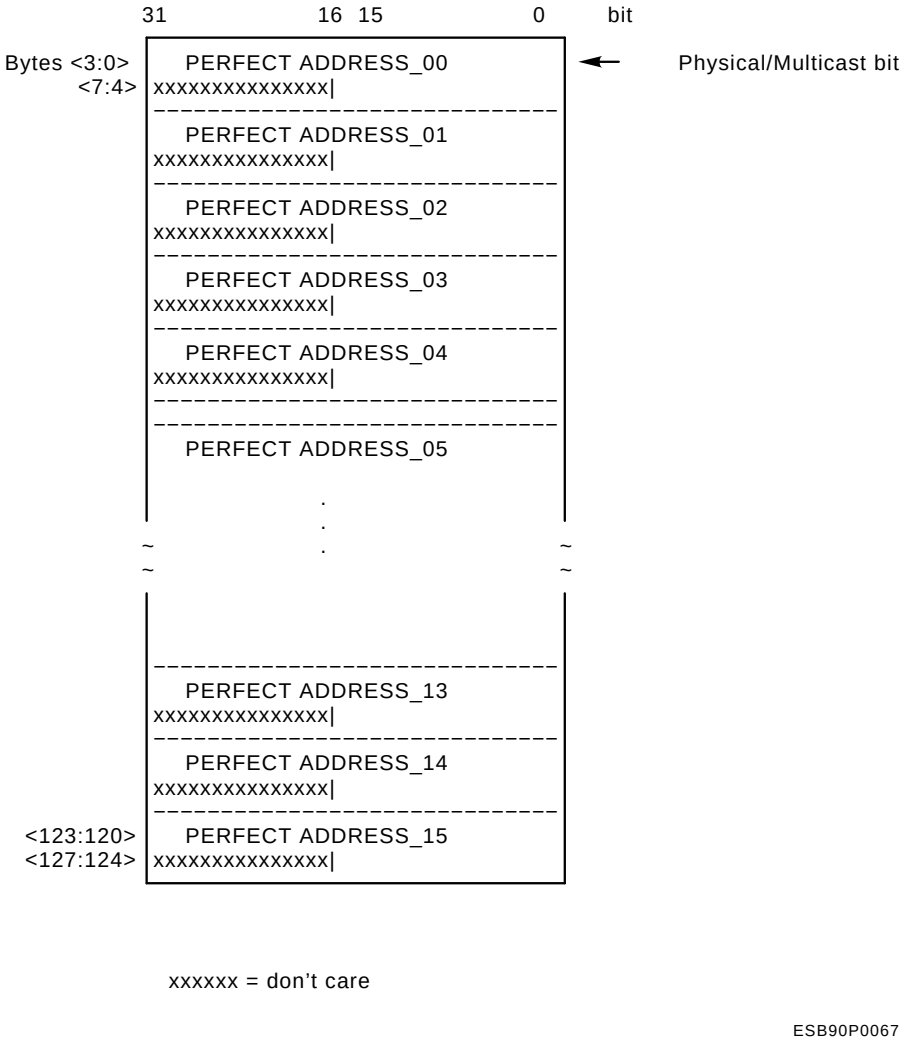
This section describes how the SGEC interprets a setup frame buffer when SDES1<HP> is clear.

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

- Those that do not match, if SDES1<IF> = 0
- Those that match, if SDES1<IF> = 1

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

Figure 10–17 Perfect Filtering Setup Frame Buffer Format



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

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

Example 10–1 Perfect Filtering 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 according to the DEC STD 134 specification for address display.

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

10.3.17.5 Imperfect Filtering Setup Frame Buffer

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

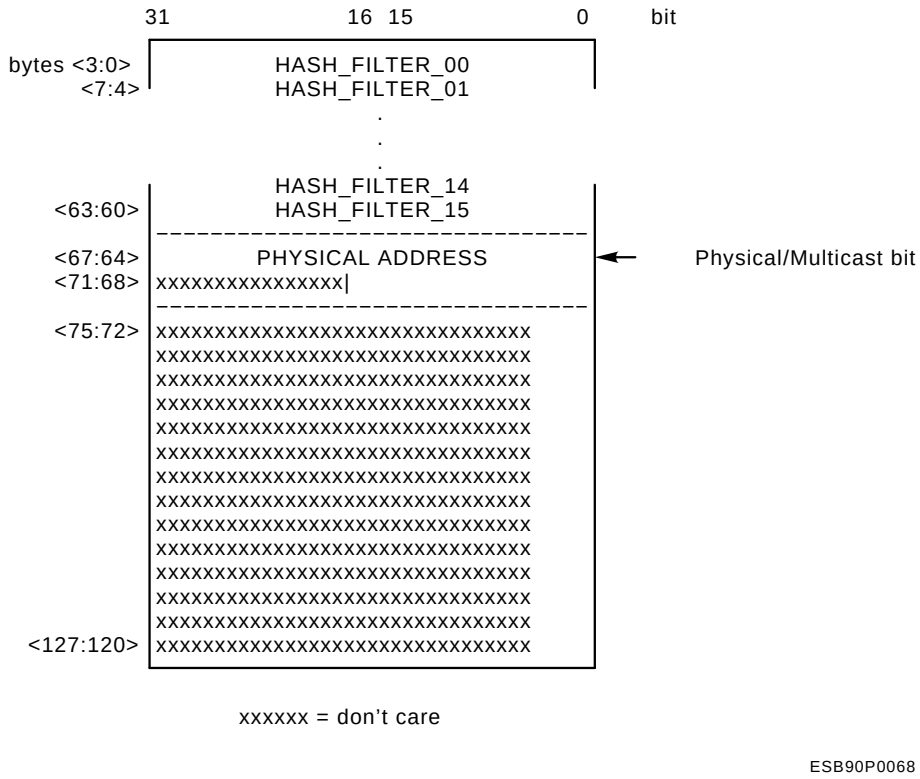
The SGEC can store 512 bits, serving as hash bucket heads, and one *physical* 48-bit Ethernet address. Incoming frames with multicast destination addresses will be subjected to the imperfect filtering. Frames with physical destination addresses will be checked against the single physical address.

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

This filtering mode is called imperfect because multicast frames not addressed to this station may slip through; however, it will still cut down the number of frames with which the host will be presented.

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

Figure 10–18 Imperfect Filtering Setup Frame Format



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

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

Example 10–2 Imperfect Filtering 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
```

(continued on next page)

Example 10–2 (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 10–3 shows a C program to compute the hash bucket heads and create the resulting setup frame buffer.

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

```
#include <stdio>

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

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

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

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

(continued on next page)

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

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

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

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

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

    goto main_loop;
}

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

/* Set CRC to all 1's           */

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

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

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

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

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

    return hash;
}

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

10.3.18 SGEC Operation

10.3.18.1 Hardware and Software Reset

The SGEC responds to two types of reset commands: a hardware reset through the RESET_L pin, and a software reset command triggered by setting NICSR6<RE>. In **both cases**, the SGEC **aborts** all ongoing processing and starts the reset sequence. The SGEC restarts and reinitializes all internal states and registers. *No internal states are retained, no descriptors are owned, and all the host visible registers are set to "0," except where otherwise noted.*

Note

The SGEC does not explicitly disown any owned descriptor; therefore, descriptors' owned bits might be left in a state indicating SGEC ownership.

Table 10–39 indicates the NICSR fields that are not set to "0" after reset:

Table 10–39 NICSR Fields Not Reset to Zero

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

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

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

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

Information

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

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

Successive reset commands (either hardware or software) may be issued. The only restriction is that SGEC NICSRs should not be accessed during a 1 µs period following the reset. Access during this period will result in a CP-bus timeout

error. Access to SGEC NICSRs during the self-test are permitted; however, only NICSR5 reads should be performed.

10.3.18.2 Interrupts

Interrupts are generated as a result of various events. NICSR5 contains all the status bits that may 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*, and if the interrupting event reoccurs *before* the port driver has responded to it, no additional interrupts will be 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 to the first SGEC owned one.

An interrupt will be generated only *once* for simultaneous, multiple interrupting events. It is the port driver responsibility to scan NICSR5 for the interrupt cause(s). The interrupt will not be *regenerated*, unless a *new* interrupting event occurs *after* the host acknowledged the previous one, and provided the port driver *cleared* the appropriate 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 will be delivered with the NICSR5<RU> bit set.

Should the port driver clear *all* NICSR5 set interrupt bits before the interrupt has been acknowledged, the interrupt will be suppressed.

10.3.18.3 Startup Procedure

A sequence of checks and commands must be performed by the port driver to prepare the SGEC for operation.

1. Wait for the SGEC to complete its initialization sequence by polling on NICSR5<ID> and NICSR5<SF>. (Refer to Section 10.3.6 for details.)
2. Examine NICSR5<SF> to find out whether the SGEC passed its self-test. If it did not, it should be replaced. (Refer to Section 10.3.6 for details.)
3. Write NICSR0 to establish system configuration dependent parameters. (Refer to Section 10.3.3 for details.)
4. If the port driver intends to use VAX virtual addresses, NICSR7 must be written to identify the system page table to the SGEC. (Refer to Section 10.3.8 for details.)
5. If the port driver wishes to change the default settings of the watchdog timers, it must write to NICSR9. (Refer to Section 10.3.10 for details.)
6. The port driver must create the transmit and receive descriptor lists, then write to NICSR3 and NICSR4 to provide the SGEC with the starting address of each list. The first descriptor on the transmit list will usually contain a setup frame. (Refer to Section 10.3.5 for details.)
7. Write NICSR6 to set global operating parameters and start the transmission and reception processes. The reception and transmission processes enter the RUNNING state and attempt to acquire descriptors from the respective descriptor lists, and begin processing incoming and outgoing frames. (Refer to Section 10.3.7 for details.) The reception and transmission processes are independent of each other and can be started and stopped separately.

Caution

If address filtering (either perfect or imperfect) is desired, the reception process should be started 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.

10.3.18.4 Reception Process

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

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

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

After 64 bytes have been received, the SGEC begins transferring the frame data to the buffer pointed to by the current descriptor. If data chaining is enabled (NICSR6<DC> clear), the SGEC will write 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 the last segment of a frame has been transferred to a buffer.

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

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

Any incoming frames while in this state will cause the SGEC to fetch the current descriptor in the host memory. If the descriptor is now owned by the SGEC, the reception reenters the RUNNING state and starts the frame reception.

If the descriptor is still owned by the host, the SGEC increments the missed frames counter (NICSR10<MFC>) and discards the frame.

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

Table 10–40 Reception Process State Transitions

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

10.3.18.5 Transmission Process

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

NICSR5<TI> will also be 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. While in the RUNNING state, as the transmission process starts, it first expects a descriptor with TDES1<FS> set. Frame data transfer from the host buffer to the internal FIFO is initiated. Concurrently, if the current frame has TDES1<LS> clear, the transmission process attempts to acquire the next descriptor. It expects TDES1<FS> and TDES1<LS> to be clear, indicating an intermediary buffer, or TDES1<LS> to be set, indicating the end of the frame. After the last buffer of the frame has been transmitted, the SGEC writes back final status information to the TDES0 word of the descriptor having TDES1<LS> set, optionally sets NICSR5<TI> if TDES1<IC> were set, and repeats the process

with the next descriptor(s). Actual frame transmission begins *after at least 72 bytes* have been transferred to the internal FIFO, or *a full frame is contained* in the FIFO. Descriptors are released (TDES0<OW> bit cleared) as soon as the SGEC finishes processing them.

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. The position in the transmit list, in all the above cases, is retained. The retained position is that of the *descriptor following the last descriptor closed (set to host ownership) by the SGEC*.

Note

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

The following table summarizes the transmission process state transitions:

Table 10–41 Transmission Process State Transitions

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

(continued on next page)

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

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

10.3.18.6 Loopback Operations

The SGEC supports two loopback modes:

- Internal loopback

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

- External loopback

  This mode generally is used to verify correct operations up to the Ethernet cable. While in this mode, the SGEC will take frames from the transmit list and transmit 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 as transmitted frames are placed on the Ethernet wire. Furthermore, the SGEC does not check the origin of any incoming frames; consequently, frames not necessarily originating from the SGEC might make it to the receive buffers.

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

1. Place the reception and transmission processes in the STOPPED state. The port driver must wait for any previously scheduled frame activity to cease. This is done by polling the TS and RS fields in NICSR5.

2. Prepare appropriate transmit and receive descriptor lists in host memory. These may follow the existing lists at the point of suspension, or may be new lists that will have to be identified to the SGEC by appropriately writing NICSR3 and NICSR4.
3. Write to NICSR6<OM> according to the desired loopback mode, and place the transmission and reception processes in the RUNNING state through Start commands.
4. Respond and process any SGEC interrupts, as in normal processing.

To restore normal operations, the port driver must execute step 1, then write the OM field in NICSR6 with "00."

10.3.18.7 DNA CSMA/CD Counters and Events Support

This section describes the SGEC features that support the port driver in implementing and reporting the specified counters and events [1].

Table 10–42 CSMA/CD Counters

Counter	SGEC Feature
Time since counter creation	Supported by the host driver.
Bytes received	Port driver must add up the RDES0<FL> fields of all successfully received frames.
Bytes sent	Port driver must add up the TDES2<BS> fields of all successfully transmitted buffers.
Frames received	Port driver must count the successfully received frames in the receive descriptor list.
Frames sent	Port driver must count the successfully transmitted frames in the transmit descriptor list.
Multicast bytes received	Port driver must add up the RDES0<FL> fields of all successfully received frames with multicast address destinations.
Multicast frames received	Port driver must count the successfully received frames with multicast address destinations.
Frames sent, initially deferred	Port driver must count the successfully transmitted frames with TDES0<DE> set.
Frames sent, single collision	Port driver must count the successfully transmitted frames with TDES0<CC> equal to 1.
Frames sent, multiple collisions	Port driver must count the successfully transmitted frames with TDES0<CC> greater than 1.
Send failure- Excessive collisions	Port driver must count the transmit descriptors having TDES0<EC> set.
Send failure- Carrier check failed	Port driver must count the transmit descriptors having TDES0<LC> set.
Send failure- Short circuit	Two successive transmit descriptors with the No_ carrier flag TDES0<NC> set, indicates a short circuit.

(continued on next page)

[1] Specified in the *DNA Maintenance Operations (MOP) Functional Specification*, Version T.4.0.0, 28 January 1988.

Table 10–42 (Cont.) CSMA/CD Counters

Counter	SGEC Feature
Send failure- Open circuit	Two successive transmit descriptors with the Excessive_collisions flag TDES0<EC> set with the same Time domain reflectometer value TDES0<TDR>, indicate an open circuit.
Send failure- Remote Failure to Defer	Flagged as a late collision TDES0<LC> in the transmit descriptors.
Receive failure- Block Check Error	Port driver must count the receive descriptors having RDES0<CE> set with RDES0<DB> cleared.
Receive failure- Framing Error	Port driver must count the receive descriptors having both RDES0<CE> and RDES0<DB> set.
Receive failure- Frame too long	Port driver must count the receive descriptors having RDES0<TL> set.
Unrecognized frame destination	Not applicable.
Data overrun	Port driver must count the receive descriptors having RDES0<OF> set.
System buffer unavailable	Reported in the Missed_frame counter NICSR10<MFC>. (Refer to Table 10–20.)
User buffer unavailable	Not applicable.
Collision detect check failed	Port driver must count the transmit descriptors having TDES0<HF> set.

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

11

KA680 Mass Storage Interface

The KA680 contains a DSSI bus interface, which is implemented with the single host adapter chip (SHAC). This interface allows the KA680 to transmit packets of data to, and receive packets of data from, up to seven other DSSI devices (typically RF-type disk drives and TF-type streaming tape drives). It should also be noted that the SHAC supports CP-bus parity protection.

11.1 SHAC Introduction

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

- The problem of interfacing a group of mass-storage device controllers (MSDCs) to a VAX system.
- The problem of 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 VAX systems communicate at high speed, with a 4 to 5 MB/s burst transfer rate. The SHAC handles the problem of providing effective, efficient, and reliable interfacing between this DSSI bus and the CPU , having direct host memory access (DMA) over the host's 32-bit wide, 16 MB/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.

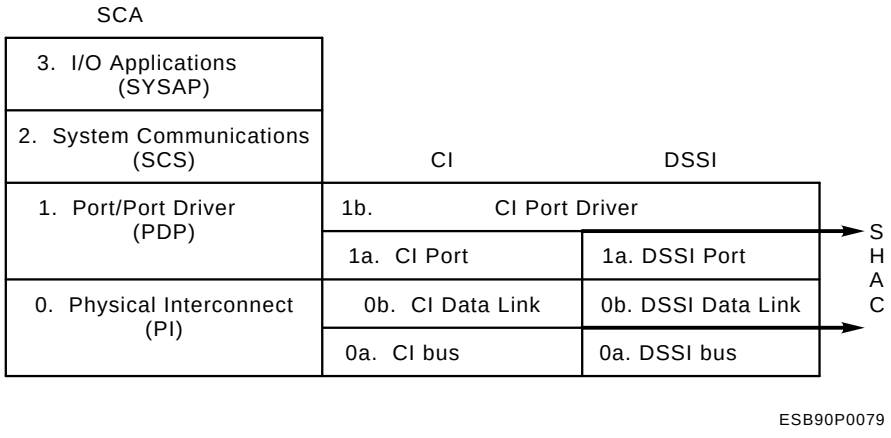
- A single DSSI bus has room for eight nodes, which may be partitioned among host adapters (for example, SHACs) and MSDCs.
- Up to four SHACs can be installed on a single host bus.
- Because there must be a host, there can be up to seven MSDCs on a single DSSI.

The SHAC provides a small amount of buffering (1.2 KB) on chip to improve bus utilization on both sides, but the SHAC is designed to pass data from one bus to the other as rapidly as the two 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 RISC that obtains its code and internal data from on-chip RAM and ROM. The RAM will be loaded from main memory during both initialization and as circumstances require during normal run time. With this capability, it can read in new code and data from the main memory and adapt its behavior to new circumstances. This will permit inexpensive upgrades of SHACs after they are installed in the field. Furthermore, it will allow the SHAC to store infrequently accessed code in main memory, providing more capability than could be included in on-chip ROM.

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

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



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

These "envelopes" are simply standard-sized "queuable" blocks of host memory. All commands and responses are copied into one of these standard-sized blocks. Included in the header on each block are a pair of queue pointers (for a doubly linked queue) and various standard identifiers that specify what is contained in the block, and how much of the block represents the actual command or response. To be visible, a block must be on a queue where pointers from other elements or the queue header show its presence. Once a block is removed from a queue, it is visible only to the entity 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 rest of this SHAC overview section describes a typical set of steps that the SHAC covers in serving its role as the CI port with "mail" in both directions.

11.2 CI-DSSI Overview

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

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

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

A **message** is a "two-sided" communication used when a virtual circuit (an established formal relationship) between members of the bus exist. Once such a virtual circuit is established, the host(s) understand how to make requests of the other side. Such a request could be an order for a data transfer in either direction. The message itself (*move data*) is contained in a command (*deliver this message to* ...). All messages are of lengths guaranteed to fit in a message envelope. Messages are always delivered sequentially to a given node—that is, in the order in which they were enqueued on a particular queue. While the SHAC supports retries if a message fails to get through, once the retry limit is reached

without successful delivery, SHAC returns the command to the host, marking it as *undeliverable*, and then breaks the virtual circuit to that node.

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 KB from host memory, starting at location X and to be stored in location Y on disk. The queues are doubly linked, so at the top of every envelope there is a forward link **FLINK** and a backward link **BLINK**. Enqueuing involves putting link values into the new element's FLINK and BLINK, and making the previous last element's FLINK and the queue header's BLINK point to the new element.

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

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

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

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

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

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

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

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

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

11.3 SHAC Registers

The CPU communicates directly with the SHAC chip through a set of device registers in each SHAC. These registers occupy a 1-page (512-byte) region in I/O address space, aligned on a page boundary.

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

In addition to the access restrictions listed for specific registers, no register other than SHAC software chip reset (SSWCR) may be read or written while certain chip intialization functions are being executed. The results of such an access during the 100 milliseconds following a reset (powerup 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 defined in the CI Port Architecture specification
- The SHAC specific registers

11.3.1 CI Port Registers

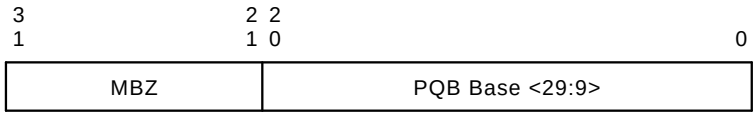
11.3.1.1 Port Queue Block Base Register (PQBBR)

SHAC I/O Address: 2000 4048$_{16}$

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

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

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



Longword Read/Write Access.

ESB90P0069

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

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

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

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

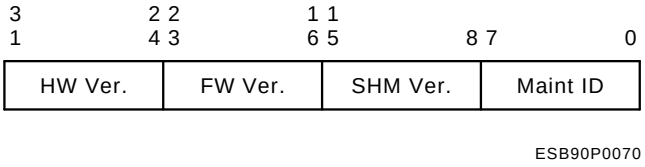


Table 11–2 Port Queue Block Base Address Register Bits After Reset

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

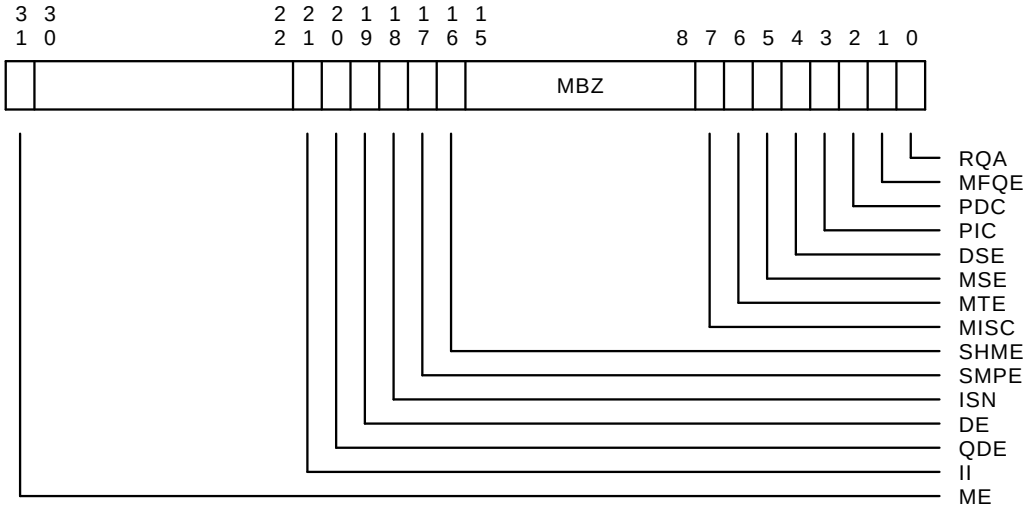
11.3.1.2 Port Status Register (PSR)

SHAC I/O Address: 2000 404C$_{16}$

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. Once an interrupt is requested by the port, the value of PSR is fixed and is not changed until the CPU releases it by writing the port status release control register (PSRCR). The port status register format is shown in Figure 11–4 and the bit descriptions are in Table 11–3.

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

Figure 11–4 Port Status Register (PSR)



Longword Read Only Access.

ESB90P0071

Table 11–3 Port Status Register Bit Descriptions

Data Bit	Name	Description
<31>	MTE	Maintenance error. When set, the port has detected an implementation specific error (or hardware status condition). The source of the error may be more accurately determined from the other bits in the upper word of this register (PSR) and the contents of other registers. Also, when set, the port is in the *uninitialized* state (port is nonfunctional). Maintenance errors normally indicate a severe SHAC hardware or software failure.
<30:22>	MBZ	Read as zero, writes have no effect.

(continued on next page)

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

Data Bit	Name	Description
<21>	II	Illegal interrupt. When set, this bit indicates a SHAC internal error, detected when the SHAC's microprocessor received an interrupt from an invalid source. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is nonfunctional).
<20>	QDE	QUIP detected error. When set, this bit indicates a SHAC internal error detected when the SHAC's microprocessor (QUIP) was given an invalid instruction. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is nonfunctional).
<19>	DE	Diagnostic error. When set, an error was detected while the SHAC was running its internal self-test. This causes ME (PSR<31>) to set and the port to enter the *uninitialized* state (port is nonfunctional).
<18>	ISN	Illegal segment number. When set, this indicates a SHAC internal error in which it 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 (port is nonfunctional).
<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 (port is nonfunctional).
<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 (port is nonfunctional).
<15:8>	MBZ	Read as zero, 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 set, the maintenance timer has expired. The port is in the *uninitialized*/*maintenance* state.
<5>	MSE	Memory system error. When set, the port has encountered an uncorrectable data or nonexistent memory error in referencing memory. Port is in the *disabled* or *disabled*/*maintenance* state. See the port failing address register (PFAR) for further information.
<4>	DSE	Data structure error. When set, the port has encountered an error in a port data structure (for example, queue entry, PQB, BDT, or page table). Port is in the *disabled* or *disabled* /*maintenance* state. See the port error status register (PESR) and the port failing address register (PFAR) for further information. Note that errors in queue structures leave the queues locked.
<3>	PIC	Port initialization complete. When set, the port has completed internal initialization. The port is in the *disabled* or *disabled* /*maintenance* state.

(continued on next page)

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

Data Bit	Name	Description
<2>	PDC	Port disable complete. When set, the port is in the *disabled* or *disabled/maintenance* state.
<1>	MFQE	Message free queue empty. When set, the port attempted to remove an entry from the message free queue (MFREEQ) and found it empty. Port processing of commands continues, and the MFREEQ may not be empty at the time the port driver gets control.
<0>	RQA	Response queue available. When set, this bit indicates port has inserted an entry on an empty response queue.

11.3.1.3 Port Error Status Register (PESR)

SHAC I/O Address: 2000 4050_{16}

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

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

Figure 11–5 Port Error Status Register (PESR)

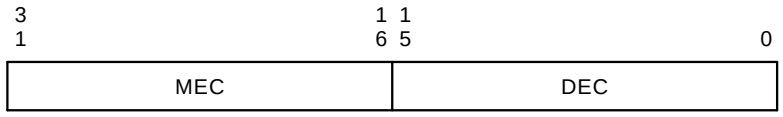


Table 11–4 Port Error Status Register Bit Definitions

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

11.3.1.4 Port Failing Address Register (PFAR)

SHAC I/O Address: 2000 4054_{16}

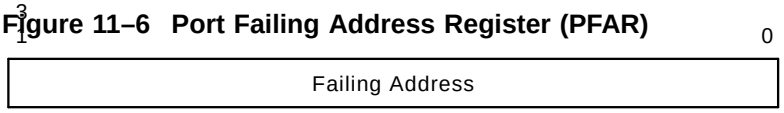
The format for the port failing address register is shown in Figure 11–6.

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

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

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

Figure 11–6 Port Failing Address Register (PFAR)



Longword Read Only Access

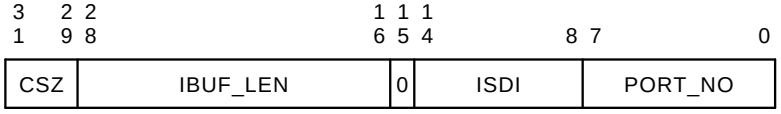
ESB90P0073

11.3.1.5 Port Parameter Register (PPR)

SHAC I/O Address: 2000 4058_{16}

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

Figure 11–7 Port Parameter Register (PPR)



Longword Read Only Access

ESB90P0074

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

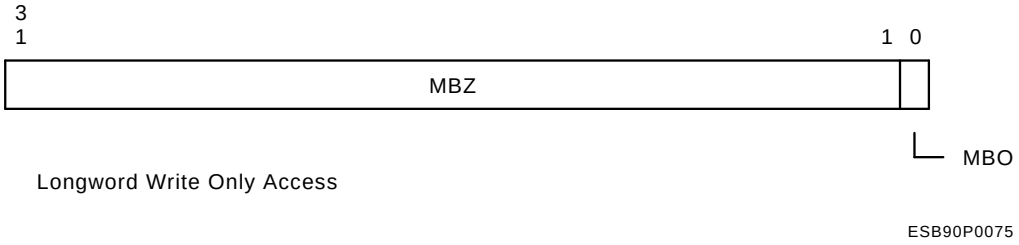
Data Bit	Name	Description
<31:29>	CSZ	Cluster size. For SHAC, this value always is zero, indicating a maximum of 16 ports on the DSSI bus. (Note that the DSSI architecture only allows up to 8 ports on the bus, but 16 is the smallest size defined for the CSZ field.)
<28:16>	IBUF_LEN	Internal buffer length.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

11.3.1.6 Port Control Registers

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

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

Figure 11–8 Port Control Registers



11.3.1.6.1 Port Command Queue 0 Control Register (PCQ0CR) SHAC I/O Address: 2000 4080_{16}

When the port driver inserts an entry in an empty CMDQ0, the port driver writes PCQ0CR to initiate port execution of the command queue. PCQ0CR can be written only when the port is in the *enabled* or *enabled/maintenance* state. Writing to PCQ0CR when the port is in any other state has no effect.

11.3.1.6.2 Port Command Queue 1 Control Register (PCQ1CR) SHAC I/O Address: 2000 4084_{16}

Same as PCQ0CR, except refers to CMDQ1.

11.3.1.6.3 Port Command Queue 2 Control Register (PCQ2CR) SHAC I/O Address: 2000 4088_{16}

Same as PCQ0CR, except refers to CMDQ2.

11.3.1.6.4 Port Command Queue 3 Control Register (PCQ3CR) SHAC I/O Address: 2000 $408C_{16}$

Same as PCQ0CR, except refers to CMDQ3.

11.3.1.6.5 Port Datagram Free Queue Control Register (PDFQCR) SHAC I/O Address: 2000 4090_{16}

When the port driver inserts an entry on the DFREEQ and the latter was previously empty, the port driver writes PDFQCR to indicate the availability of DFREEQ entries. PDFQCR can be written only if the port is in the *enabled* or *enabled/maintenance* state. Writing to PDFQCR when the port is in any other state has no effect.

11.3.1.6.6 Port Message Free Queue Control Register (PMFQCR) SHAC I/O Address: 2000 4094_{16}

Same as PDFQCR, except refers to MFREEQ.

11.3.1.6.7 Port Status Release Control Register (PSRCR) SHAC I/O Address: 2000 4098_{16}

After the port driver has received an interrupt and read the PSR, it returns the PSR to the port by writing PSRCR.

11.3.1.6.8 Port Enable Control Register (PECR) SHAC I/O Address: 2000 $409C_{16}$

The port driver enables the port by writing PECR. PECR is ignored if the port is in the *uninitialized*, *uninitialized/maintenance*, *enabled*, or *enabled/maintenance* state.

11.3.1.6.9 Port Disable Control Register (PDCR) SHAC I/O Address: 2000 $40A0_{16}$ The port driver disables the port by writing PDCR. When the port is disabled, the port sets PDC (PSR <2>), and if interrupts are enabled, requests an interrupt. PDCR is ignored if the port is in the *uninitialized*, *uninitialized /maintenance*, *disabled*, or *disabled/maintenance* state.

11.3.1.6.10 Port Initialize Control Register (PICR) SHAC I/O Address: 2000 $40A4_{16}$ The port driver initializes the port by writing PICR. When the initialization is complete, the port sets PDC (PSR <2>) and requests an interrupt if interrupts are enabled. As part of the initialization, the maintenance timer is set to expire in 100 seconds.

11.3.1.6.11 Port Maintenance Timer Control Register (PMTCR) SHAC I/O Address: 2000 $40A8_{16}$ The port driver forces the maintenance timer to reset its expiration time by writing the PMTCR. If the PMTCR is not written again before the expiration time, the port will enter the *uninitialized/maintenance* state setting MTE (PSR <6>), and request an interrupt if interrupts are enabled. PMTCR is ignored if the maintenance timer is not running.

11.3.1.6.12 Port Maintenance Timer Expiration Control Register (PMTECR) SHAC I/O Address: 2000 $40AC_{16}$ The port driver forces a maintenance-timer-expiration interrupt by writing the PMTECR. This register may be written only when the port is in the *enabled*, *enabled/maintenance*, *disabled*, and *disabled /maintenance* states, and only while the maintenance timer is not disabled.

11.3.1.6.13 Port Maintenance Control and Status Register (PMCSR) SHAC I/O Address: 2000 $405C_{16}$ The port maintenance control and status register (PMCSR) is used for maintenance level control and status reporting. The CI Port specification defines all but the two least significant bits. The format is shown in Figure 11–9 and the bit descriptions are listed in Table 11–6.

The bits can be divided into the following categories:

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

  These bits are divided into two classes:

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

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

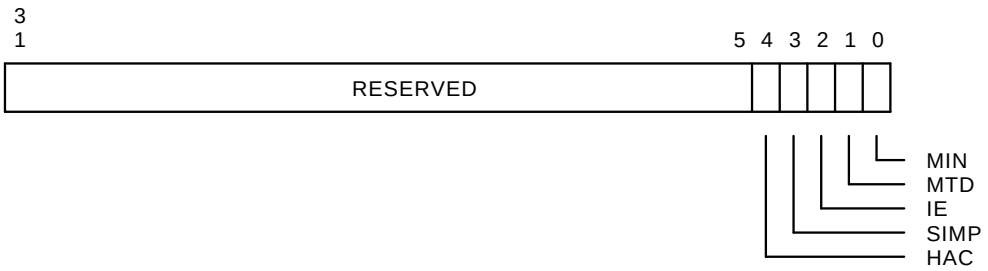


Table 11–6 Port Maintenance Control and Status Register (PMCSR) Bits

Data Bit	Name	Description
<31:5>	RESERVED	These bits are reserved. They should not be written; reads return unpredictable results.
<4>	HAC	Host access feature. This bit must be zero, except for diagnostic purposes. This is an enable/disable class control bit.
<3>	SIMP	Simple SHAC mode. Must be zero, except for diagnostic purposes. This is an enable/disable class control bit.
<2>	IE	Interrupt enable. When set, interrupts from the port to the CPU are enabled. Power-up state is clear (interrupts disabled). This is an enable/disable class control bit.
<1>	MTD	Maintenance timer disable. Read/write by CPU . If set, the maintenance timer is turned off. Timer is set to the initial value and suspended. If clear, timer functions normally. Power-up state is clear (timer enabled). This is an enable/disable class control bit.
<0>	MIN	Maintenance init. Writing a one to this bit resets the port. Upon completion, the port is in the *uninitialized* state and MIN is clear. Writing a zero to this bit has no effect. It always reads as zero, except while the reset function is active. Although maintenance init resets the port, it is not equivalent to a write to the SHAC software chip reset register. In particular, the SHAC shared host memory address is not reset by maintenance init.

11.3.2 SHAC Specific Registers

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

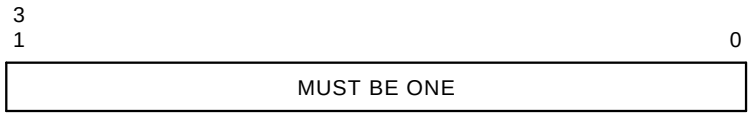
11.3.2.1 SHAC Software Chip Reset Register (SSWCR)

SHAC I/O Address: 2000 4030_{16}

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

SSWCR is write-only by the CPU and may be written to at any time. Its value when read is unpredictable. The result, if other than FFFF $FFFF_{16}$, is written to SSWCR as undefined.

Figure 11–10 SHAC Software Chip Reset (SSWCR)



Longword Write Only Access

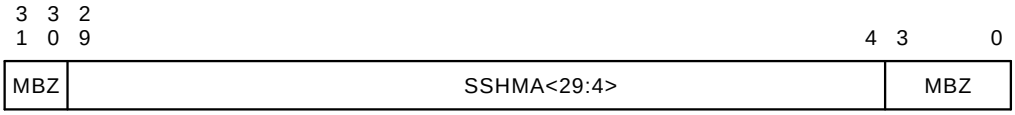
ESB90P0077

11.3.2.2 SHAC Shared Host Memory Address (SSHMA)

SHAC I/O Address: 2000 4044_{16}

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

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



Longword Read/Write Access

ESB90P0078

Following 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/write 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.

12

KA680 Firmware

12.1 KA680 Firmware Overview

This chapter describes the KA680 functional firmware. The firmware is VAX–11 code, which resides in FEPROM on the KA680 module. Typically KA680 firmware gains control whenever the on-board 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 KA680 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 powerup, on reset, or conditionally after processor halts
- Diagnostic tests executed both at powerup 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 KA680 is provided in Section 12.2, and a structural diagram of the KA680 firmware is shown in Figure 12–1.

Throughout this chapter, "firmware" is a generic term describing all program code located in the KA680 FEPROM. Sometimes it is referred to as either the "boot ROM," "diagnostics ROM," or "console ROM," depending on context. Each major element of the firmware is referred to by other terms (for instance, the boot program as "VMB" or "primary bootstrap," the ROM-based diagnostic program as the "diagnostic" or "self-test," and the operator interface as the "console" or "console program").

Certain terminology and conventions are used throughout this chapter. With one exception, numbers (unless otherwise indicated or implied) are decimal. Eight-digit numbers throughout this document are hexadecimal longwords, typically representing VAX 32-bit addresses or data. Where there is ambiguity, the radix is explicitly stated. For instance, 72 is assumed to be decimal and for clarity can be written as 72 (dec). However, alternate representations for 72 are 1001000 (bin) for binary, 110 (oct) for octal, or 48 (hex) for hexadecimal. On the other hand, E0040000 is the hexadecimal address or the base of the firmware FEPROM.

Ranges of integers are expressed as a pair of numbers separated by a colon and are always inclusive. For example, 7:4 specifies the range of integers from 7 to 4 (namely, 7, 6, 5, and 4).

A bit field or position within a register or data structure follows the structure name and is enclosed in angle brackets. The associated field name (if defined) typically follows the field definition and appears in parentheses. For instance, PSL<20:16> (IPL) represents the 5-bit field for the interrupt priority level in the processor status longword.

12.2 Firmware Capabilities

The KA680 firmware provides the following services:

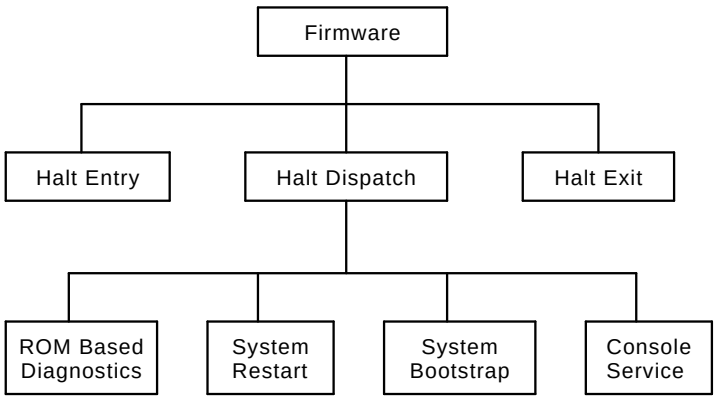
- Diagnostics that test all components on the board and verify the module is working correctly
- Automatic/manual bootstrap of an operating system following processor halts.
- Automatic/manual restart of an operating system following processor halts.
- An interactive command language that allows the 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 remainder of this section describes in detail the functions and external characteristics of the KA680 firmware.

12.2.1 General Description

The KA680 firmware is comprised of several major functional blocks of code. The **halt entry code** is invoked following system halts, resets, or severe errors. This code is responsible for saving the machine state and transferring control to the halt dispatcher code. The **halt dispatcher code** determines the nature of the halt, then transfers control to the appropriate subcode. The **halt exit code** is invoked whenever a transition is desired from a halted state to the running state. This code performs a restoration of the saved context prior to the transition. Figure 12–1 illustrates this, and these functions are discussed in detail in Section 12.2.2.

Figure 12–1 KA680 Firmware Structural Components



The *ROM-based diagnostics* consist of an initial power-up test and a series of functional component diagnostics invoked by a diagnostic executive. These functions are described in Section 12.2.5 on powerup, and in Section 12.3 on diagnostics.

Depending on the nature of the halt and the hardware context, the firmware attempts either an *operating system restart* (discussed in Section 12.2.7), *a bootstrap operation* (described in Section 12.2.6), or transitions to *console I/O mode* (covered in Section 12.2.8).

12.2.2 Halt Code

The main purpose of the halt code is to save the state of the machine on halt entry, invoke the dispatcher, and restore the state of the machine on exit to program I/O mode. It is comprised of halt entry, halt dispatch, and halt exit codes.

12.2.3 Halt Entry - Saving Processor State

The entry code, residing at physical address E0040000, is executed whenever the KA680 halts. The value that the program counter contained when the processor was halted is saved in IPR 42 (PR$_SAVPC). On a powerup, the PR$_SAVPC register value is undefined.

The processor will halt for a variety of reasons. The reason for the halt is stored in PR$_SAVPSL<13:8>(RESTART_CODE), IPR 43. A complete list of the halt reasons and the associated console messages can be found in Table C–1 in Appendix C.

After a halt, the firmware first saves the current LED code, then writes an "E" to the diagnostic LEDs. This action occurs within several instructions after the firmware has been invoked. The intent of saving the LED code is to let the user know that at least some instructions have been successfully executed.

The KA680 firmware unconditionally saves the contents of the following registers on any halt:

- R0 through R15, the general-purpose registers
- PR$_SAVPSL, the saved PSL register
- PR$_SCBB, the system control block base register
- DLEDR, the diagnostic LED register

Note

The SSC programmable timer registers are not saved. In some cases, such as bootstrap, the timers are used by the firmware and previous "time" context is lost.

Several registers are unconditionally set to predetermined values by the firmware on any halt, processor init, or bootstrap. This action ensures that the firmware itself can run and protects the board from physical damage.

The following is a list of registers that fall into this category:

- The SSC configuration register (SSCCR)
- The SSC address match and mask registers (ADxMCH & ADxMSK)
- The CDAL bus timeout control register (CBTCR)
- The SSC timer interrupt vector registers (TIVRx)

Whenever the halt entry code is invoked, the firmware sets the console serial line baud rate based on the value read from the BDR and extends the halt protection from 8 KB to 512KB to include all of the FEPROM.

12.2.4 Halt Dispatch

The action taken by the firmware on a halt is dependent primarily on the following information:

- The state of BREAK enable switch BDR<7>(HALT_ENABLE)
- The state of the console program mailbox, CPMBX<1:0>(HALT_ACTION)
- The user-defined halt action (SET HALT)
- The halt code, PR$_SAVPSL<13:8>(RESTART_CODE)

In general, the BREAK enable switch governs whether or not a BREAK condition from the console serial line is recognized by the KA680. This switch also determines the default action taken on a powerup or other internal halt condition. By default, if BREAKs are enabled, the firmware invokes the console emulation code. If BREAKs are disabled, the firmware attempts a recovery operation.

The console program mailbox, CPMBX<1:0>(HALT_ACTION), is used by operating systems to override the BREAK enable switch. It is used to instruct the firmware to invoke the console service, attempt to restart the operating system, or reboot the system following a halt, regardless of the setting of the BREAK enable switch. (See Figure A–2.)

The user-defined halt action invoked by using the SET HALT console command (refer to the description of the SET command in Section 12.2.9 is an alternative way to specify a default halt action. This feature allows users to specify autobooting on powerups, even when BREAKs are enabled. For HALT instructions and error halt conditions, it is similar in function to the console program mailbox but has lower precedence and is only used when the console program mailbox is 0. This provides the user with a mechanism to specify what action should be taken in the event that the operating system or user application does not set the console program mailbox.

The halt (or restart) code is automatically deposited in PR$_ SAVPSL<13:8>(RESTART_CODE) on any halt condition. This field indicates the cause of the halt, and for the purpose of dispatching, collapses into three categories.

02: External halts
03: Reset/powerup
xx: All other values—(HALT instruction and all error halts)

Table 12–1 summarizes the action taken on all halt conditons, except external halts, which are described in Section 12.2.4.0.1. The actual halt dispatch state machine is described in detail in Section B.1 of Appendix B.

Table 12–1 Halt Action Summary

Halt Code= 3	BREAK Enable Switch	User-Defined Halt Action	Console Program Mailbox	Action(s)
T	1	0,1,3	x	Diagnostics, console
T	1	2,4	x	Diagnostics, if success boot, if either fail console
T	0	x	x	Diagnostics, if success boot, if either fail console
F	1	0	0	Console
F	0	0	0	Restart, if this fails boot, if that fails console
F	x	1	0	Restart, if it fails console
F	x	2	0	Boot, if it fails console
F	x	3	0	Console
F	x	4	0	Restart, if this fails boot, if that fails console
F	x	x	1	Restart, if it fails console
F	x	x	2	Boot, if it fails console
F	x	x	3	Console

"T" TRUE—indicates a reset or power-up condition
"F" FALSE—indicates a HALT instruction or error halt condition
"x" DON'T CARE—indicates that the condition is "don't care"

Because the KA680 does not support battery backed-up main memory, an operating system restart operation is not attempted on a powerup.

12.2.4.0.1 External Halts Several conditions can trigger an external halt and different actions are taken, depending on the condition.

An external halt can be caused by one of the following conditions.

1. A BREAK condition on the system console serial line, if the BREAK enable switch is set to "enabled." In this case, BDR<7>(HALT_ENABLE) = 1 and the console code is invoked. **Control-P may be established as the "BREAK" condition by using the SET CONTROLP ENABLE console command.**
2. The assertion of the BHALT line on the Q22–bus causes an external halt if the SCR<14>(BHALT_ENABLE) bit in the CQBIC is set. As a result, the console code is invoked.
3. The negation of DCOK on the Q22–bus, if the SCR<7>(DCOK_ACTION) bit is set, causes an external halt (by default, this bit is clear). As a result, the console code is invoked.
4. Recognition of a valid MOP BOOT message by an appropriately initialized SGEC, if the REMOTE_BOOT_ENABLE jumper is in place [BDR<31>(REMOTE_BOOT_ENABLE) = 1]. As a result, a bootstrap is attempted. If that fails, the console is entered.

Note

The firmware does not initialize the SGEC for this operation. The operating system must set up the SGEC to support this feature.

Note

The switch labeled "RESTART" negates DCOK. The DCOK bit may also be negated by the DEQNA sanity timer, or any other Q22–bus module that chooses to implement the Q22–bus restart/reboot protocol. Because the SCR<7>(DCOK_ACTION) bit is cleared on powerup, the default consequence to deasserting DCOK is to generate a processor restart. Hence, pushing the "RESTART" button typically initiates a power-up sequence and destroys system state.

12.2.4.1 Halt Exit - Restoring Processor State

When the firmware exits, it uses the currently defined saved context. This context is initially determined by what was saved when the firmware code was invoked. However, this context may be modified by console commands, or automatic operations such as an automatic bootstrap on powerup.

When restoring the context, the firmware will flush the CPU internal cache if enabled, and invalidate all translation buffer entries via the internal processor register PR$_TBIA, IPR 57.

In restoring the context, the console pushes the user's PSL and PC onto the user's interrupt stack, then executes a *return from exception or interrupt* instruction (REI) from that stack. This implies that the user's interrupt stack pointer (ISP) is valid before the firmware can exit. This is done automatically on a bootstrap. However, it is suggested that the stack pointer (SP) be set to a valid memory location before issuing the START or CONTINUE command. Furthermore, the user should validate the system control block base register (SCBB or PR$_SCBB) prior to executing a NEXT command, because the firmware uses the trace trap vector for this function. At powerup, the user ISP is set to 200 (hex) and the system control block base register is undefined.

12.2.5 Power-up

This section describes the sequence of events that occurs on power-up.

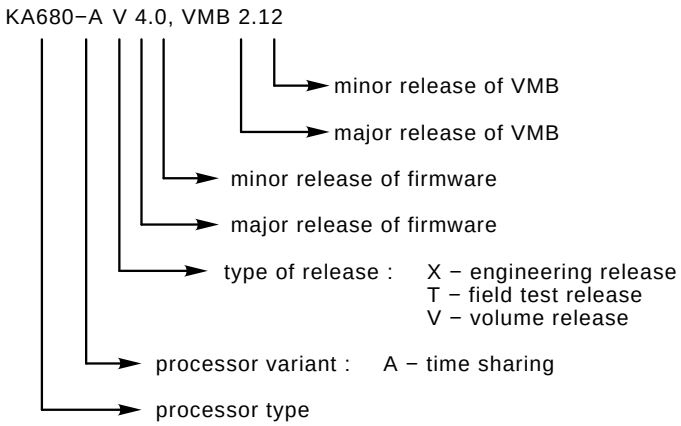
At power-up, the KA680 firmware performs actions that are unique to the power-up condition. Among these actions are the following: locating and identifying a console device, language query, and the diagnostic countdown. Certain actions are dependent on the state of the "mode" switch on the H3604-SA. The mode switch panel which has three settings:

12.2.5.1 Identifying the Console Device

After powerup, the firmware attempts to determine what type of console device is present so that the device may be used to display further diagnostic progress. Normally, this is the device attached to the console serial line cable, and the firmware sends the "device attributes escape sequence" (<ESC>[c) across the cable. This action determines the type of terminal attached and the functions it supports. Terminals that do not respond to the device attributes request correctly are assumed to be hard copy devices.

Once a console device has been identified, the firmware displays the KA680 banner message, which contains the processor name, the version of the firmware, and the version of the VMB code as explained in Figure 12–2.

Figure 12–2 Console Banner



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 if the firmware is engineering release, field test release, or volume release. 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 shown in Figure 12–3 in Section 12.2.5.1.2.

After the language query, the firmware invokes the ROM-based diagnostics, and eventually displays the console prompt.

12.2.5.1.1 Mode Switch Set to "Test" If the mode switch is set to "test," the console serial line external loopback test is executed at the end of the IPT. **An external loopback connector should be inserted in the serial line connector on the H3604-SA panel prior to cycling power to invoke this test.** The purpose of this test is to verify that the console serial line connections from the KA680 through the H3604-SA panel are intact.

During this test, the firmware toggles between two states, active and passive, each a few seconds long and each displaying a different number on the LEDs.

During the active state (about 3 seconds long), the LEDs are set to "6." In this state, the firmware reads the baud rate and mode switch, then transmits and receives a character sequence. If the mode switch has been moved from the "test" position, the firmware exits the test and continues as if on a normal powerup.

During the passive state (about 7 seconds long), the LEDs are set to "3."

If the firmware detects an error (parity, framing, overflow, or no characters), the firmware "hangs" with a "6" on the LEDs.

12.2.5.1.2 Mode Switch Set to "Query" If the mode switch is set to "query" (or the firmware detects that the battery failed during a power loss), the firmware queries the user for a language that is used for displaying critical system messages.

The language selection menu is shown in Figure 12–3.

Figure 12–3 Language Selection Menu

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

The user may select from one of the eleven supported languages. If no response is received within 30 seconds, the language defaults to English KEEP>((United States/Canada).) For those languages that do not have a unique keyboard, Figure 12–3 displays supported country-specific keyboard variants in parentheses. **Language inquiry is performed only if the console device supports DEC MCS. Any console device that does not support DEC MCS, such as a VT100, defaults to English (United States/Canada).**

After completing language inquiry, the firmware proceeds as if the mode switch were set to "normal," as described in Section 12.2.5.1.3.

12.2.5.1.3 Mode Switch Set to "Normal" If the mode selected is "normal," then the next step in the power-up sequence is to execute the bulk of ROM-based diagnostics. In addition to the message text, a "countdown" is displayed to indicate diagnostic test progress. A successful diagnostic countdown is shown in Figure 12–4.

Figure 12–4 Normal Diagnostic Countdown

```
Performing normal system tests.
66..65..64..63..62..61..60..59..58..57..56..55..54..53..52..51..
50..49..48..47..46..45..44..43..42..41..40..39..38..37..36..35..
34..33..32..31..30..29..28..27..26..25..24..23..22..21..20..19..
18..17..16..15..14..13..12..11..10..09..08..07..06..05..04..03..
Tests completed.
```

In the case of diagnostic failures, a diagnostic register dump is performed similar to the one shown in Figure 12–5. The remaining diagnostics execute and the countdown continues. For a detailed description of the register dump, refer to Section 12.3.

Figure 12–5 Abnormal Diagnostic Countdown

```
Performing normal system tests.
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..

?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
10..09..08..07..06..05..04..03..
Normal operation not possible.
```

If the diagnostics have successfully completed and halts are enabled, the firmware displays the console prompt and enters "console I/O" mode.

Figure 12–6 Console Prompt

```
>>>
```

If the diagnostics have successfully completed and halts are disabled, the firmware attempts to boot an operating system.

Figure 12–7 Console Boot Display with No Default Boot Device

```
Loading system software.
No default boot device has been specified.
Devices:
-DIA0 (RF70)
-DIA1 (RF70)
-MUA0 (TK70)
-EZA0 (08-00-2B-03-82-78)
Device? [EZA0]:

(BOOT/R5:0 EZA0)

  2..
-EZA0
```

12.2.5.2 LED Codes

In addition to the console diagnostic countdown, a hexadecimal value is displayed on the diagnostic LEDs on the module and the H3604-SA panel. The purpose of the LED display is to improve fault isolation when there is no console terminal, or when the hardware is incapable of communicating with the console terminal. Table 12–2 lists all LED codes and the associated actions performed at powerup. The LED code is changed before the corresponding test or action is performed.

Table 12–2 LED Codes

LED Value	Actions
F	Initial state on powerup, no code has executed
E	Entered ROM space, some instructions have executed
D	Waiting for power to stabilize (POK)
C	SSC RAM, SSC registers, and ROM checksum tests
B	O-bit memory, interval timer, and virtual mode tests
A	FPA tests
9	Backup cache, primary cache, and memory tests
8	NMC, NCA, 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 VMB
1	Control passed to secondary bootstrap
0	"Program I/O" mode, control passed to operating system

12.2.6 Operating System Bootstrap

Bootstrapping is the process by which an operating system loads and assumes control of the system. The KA680 supports bootstrap of the following operating systems: VAX/VMS and VAXELN. Additionally, the KA680 will boot MDM diagnostics and any user application image that conforms to the boot formats described in this section.

On the KA680 a bootstrap occurs whenever a BOOT command is issued at the console or whenever the processor halts and the conditions specified in the Table 12–1 for automatic bootstrap are satisfied.

12.2.6.1 Preparing for the Bootstrap

Prior to dispatching to the primary bootstrap (VMB), the firmware initializes the system to a known state. The initialization sequence follows:

1. Check the console program mailbox "bootstrap in progress" bit [CPMBX<2>(BIP)]. If it is set, bootstrap fails.
2. If this is an automatic bootstrap, print the message "Loading system software." on the console terminal.
3. Set CPMBX<2>(BIP).
4. Validate the page frame number (PFN) bitmap. If PFN bitmap checksum is invalid, then:
   a. Perform an UNJAM .
   b. Perform an INIT .
   c. Retest memory and rebuild PFN bitmap.
5. Validate the boot device name. If none exists, supply a list of available devices and prompt user for a device. If no device is entered within 30 seconds, use EZA0.
6. Write a form of this BOOT request including the active boot flags and boot device on the console: for example, "(BOOT/R5:0 DUA0)".
7. Initialize the Q22–bus scatter/gather map.
   a. Set IPCR<8>(AUX_HLT).
   b. Clear IPCR<5>(LMEAE).
   c. Perform an UNJAM .
   d. Perform an INIT .
   e. If an arbiter, map all vacant Q22–bus pages to the corresponding page in local memory and validate each entry if that page is "good."
   f. Set IPCR<5>(LMEAE).
8. Search for a 128 KB contiguous block of good memory as defined by the PFN bitmap. If 128 KB cannot be found, the bootstrap fails.
9. Initialize the general purpose registers.

   R0 = Address of descriptor of the boot device name, or 0 if none specified
   R2 = Length of PFN bitmap in bytes
   R3 = Address of PFN bitmap
   R4 = Time of day from PR$_TODR at powerup
   R5 = Boot flags

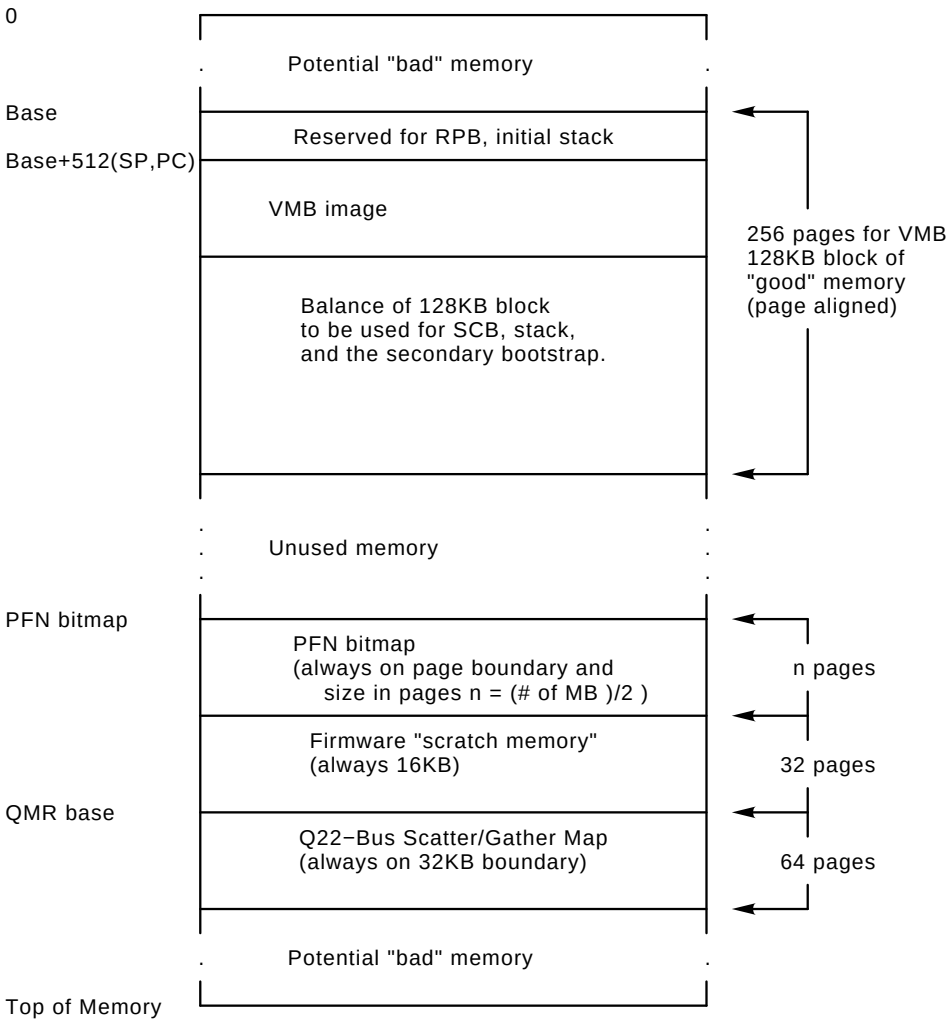
R10 = Halt PC value
R11 = Halt PSL value (without halt code and map enable)
AP = Halt code
SP = Base of 128 KB good memory block + 512
PC = Base of 128 KB good memory block + 512
R1, R6, R7, R8, R9, FP = 0

10. Copy the VMB image from FEPROM to local memory beginning at the base of the 128 KB 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 12–8.

Figure 12–8 Memory Layout Prior to VMB Entry



12.2.6.1.1 Boot Devices The KA680 firmware passes the address of a descriptor of the boot device name to VMB through R0. This device name used for the bootstrap operation is one of the following:

- The local Ethernet device, if no default boot device has been specified
- The default boot device specified at initial powerup or via a SET BOOT command
- The boot device name explicitly specified in a BOOT command line

The device name may be any arbitrary character string, with a maximum length of 17 characters. Longer strings cause an error message to be issued to the console. Otherwise, the console makes no attempt at interpreting or validating the device name. The console converts the string to uppercase letters, and passes to VMB the address of a string descriptor for the device name in R0.

Table 12–3 correlates the boot device names expected in a BOOT command with the corresponding supported devices.

Table 12–3 KA680 Supported Boot Devices

	Boot Name[1]	Controller Type	Device Type(s)
Disk:			
	[node$]DIAn	On-board DSSI	RF31, RF35, RF71, RF72
	DUcn	RQDX3 MSCP	RD52, RD53, RD54, RX33, RX50
		KDA50 MSCP	RA70, RA80, RA81, RA82, RA90, RA92
		KFQSA MSCP	RF31, RF35, RF71, RF72
		KLESI	RC25
	DLcn	RLV12	RL01, RL02
	DKcnnn	KZQSA	RRD42
Tape:			
	[node$]MIAn	On-board DSSI	TF85, TF857
	MUcn	TQK50 MSCP	TK50
		TQK70 MSCP	TK70
		KFQSA MSCP	TF70
		KLESI	TU81E
	MKcnnn	KZQSA	TLZ04
Network:			
	EZA0	On-board Ethernet	—
	XQcn	DELQA	—
		DESQA	—

[1] Boot device names consist of a 2-letter device code (minimum), followed by a single-character controller letter (A...Z), and terminating in a device unit number (0...65535). DSSI device names may optionally include a node prefix, consisting of either a node number (0...7) or a node name (a string of up to 8 characters), terminating in a "$".

(continued on next page)

Table 12–3 (Cont.) KA680 Supported Boot Devices

	Boot Name[1]	Controller Type	Device Type(s)
PROM:			
	PRA0	MRV11	——
	PRB0	On-board EPROM	——

[1] Boot device names consist of a 2-letter device code (minimum), followed by a single-character controller letter (A...Z), and terminating in a device unit number (0...65535). DSSI device names may optionally include a node prefix, consisting of either a node number (0...7) or a node name (a string of up to 8 characters), terminating in a "$".

Table 12–3 presents a definitive list of boot devices that the KA680 supports. However, the KA680 will likely boot other devices that adhere to the MSCP standards.

12.2.6.1.2 Boot Flags The action of VMB is qualified by the value passed to it in R5. R5 contains boot flags that specify conditions of the bootstrap. The firmware passes to VMB either the R5 value specified in the BOOT command or the default boot flag value specified with a SET BFLAG command.

Figure 12–9 shows the location of the boot flags used by VMB in the boot flag longword and describes each flag's function.

Figure 12–9 VMB Boot Flags (/R5:)

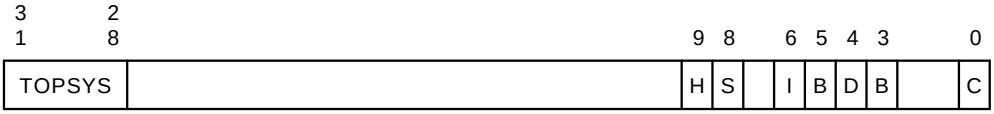


Table 12–4 VMB Boot Flags

Field	Name	Description
<31:28>	RPB$V_TOPSYS	This field can be any value from 0 through F. This flag changes the top-level directory name for the system disks with multiple operating systems. For example, if TOPSYS is 1, the top-level directory name is [SYS1...]. **This does not apply to network bootstraps.**
<9>	RPB$V_HALT	Halt during bootstrap. When this bit is set, VMB halts on entry to VMB before transferring control to the loaded image, and potentially in the loaded image.
<8>	RPB$V_SOLICT	File name solicit. When this bit is set, VMB prompts the operator for the name of the application image file. A maximum of a 39-character file specification is allocated at RPB$T_FILE. **Only 16 characters are utilized in both tape boot and network MOP V3 booting.**
<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.

(continued on next page)

Table 12–4 (Cont.) VMB Boot Flags

Field	Name	Description
<5>	RPB$V_BOOBPT	Bootstrap breakpoint. If this flag is set, a breakpoint instruction is executed in VMB and control is transferred to XDELTA prior to boot.
<4>	RPB$V_DIAG	Diagnostic bootstrap. When this bit is set, the load image requested over the network is [SYS0.SYSMAINT]DIAGBOOT.EXE.
<3>	RPB$V_BBLOCK	Secondary bootstrap from bootblock. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the boot block format. If in conformance, the block is executed to continue the bootstrap. No attempt to perform a Files–11 bootstrap is made.
<0>	RPB$V_CONV	Conversational bootstrap.

12.2.6.2 Primary Bootstrap, VMB

Virtual memory boot (VMB) is the primary bootstrap for booting VAX processors. On the KA680, VMB is resident in the firmware and is copied into main memory before control is transferred to it. VMB then loads the secondary bootstrap image and transfers control to it.

In certain cases, such as VAXELN, VMB actually loads the operating system directly. However, in this chapter, "secondary bootstrap" refers to any VMB loadable image.

VMB inherits a well-defined environment and is responsible for further initialization. The following summarizes the operation of VMB:

1. Initialize a 2-page SCB on the first page boundary above VMB.
2. Allocate a 3-page stack above the SCB.
3. Initialize the restart parameter block (RPB). Refer to Table B–2.
4. Initialize the secondary bootstrap argument list. Refer to Table B–3 in Appendix D.
5. If not a PROM boot, locate a minimum of 3 consecutive valid QMRs.
6. Write "2" to the diagnostic LEDs and display "2.." on the console to indicate that VMB is searching for the device.
7. Optionally, solicit from the console a "Bootfile: " name.
8. Write the name of the boot device from which VMB will attempt to boot on the console (for example, "-DUA0").
9. Copy the secondary bootstrap from the boot device into local memory above the stack. If this fails, the bootstrap fails.
10. Write "1" to the diagnostic LEDs and display "1.." on the console to indicate that VMB has found the secondary bootstrap image on the boot device and has loaded the image into local memory.
11. Clear CPMBX<2>(BIP) and CPMBX<3>(RIP).
12. Write "0" to the diagnostic LEDs and display "0.." on the console to indicate that VMB is now transferring control to the loaded image.

13. Transfer control to the loaded image with the following register usage:

    R5 = Transfer address in secondary bootstrap image
    R10 = Base address of secondary bootstrap memory
    R11 = Base address of RPB
    AP = Base address of secondary boot parameter block
    SP = Current stack pointer

If the bootstrap operation fails, VMB relinquishes control to the console by halting with a HALT instruction. **VMB makes no assumptions about the location of Q22–bus memory. However, VMB searches through the Q22–bus map registers (QMRs) for the first QMR marked "valid." VMB requires a minimum of 3 and a maximum of 129 contiguous "valid" maps to complete a bootstrap operation. If the search exhausts all map registers or there are fewer than the required number of "valid" maps, a bootstrap cannot be performed. It is recommended that a suitable block of Q22–bus memory address space be available (unmapped to other devices) for proper operation.**

Figure 12–10 shows a sample console display of a successful automatic bootstrap.

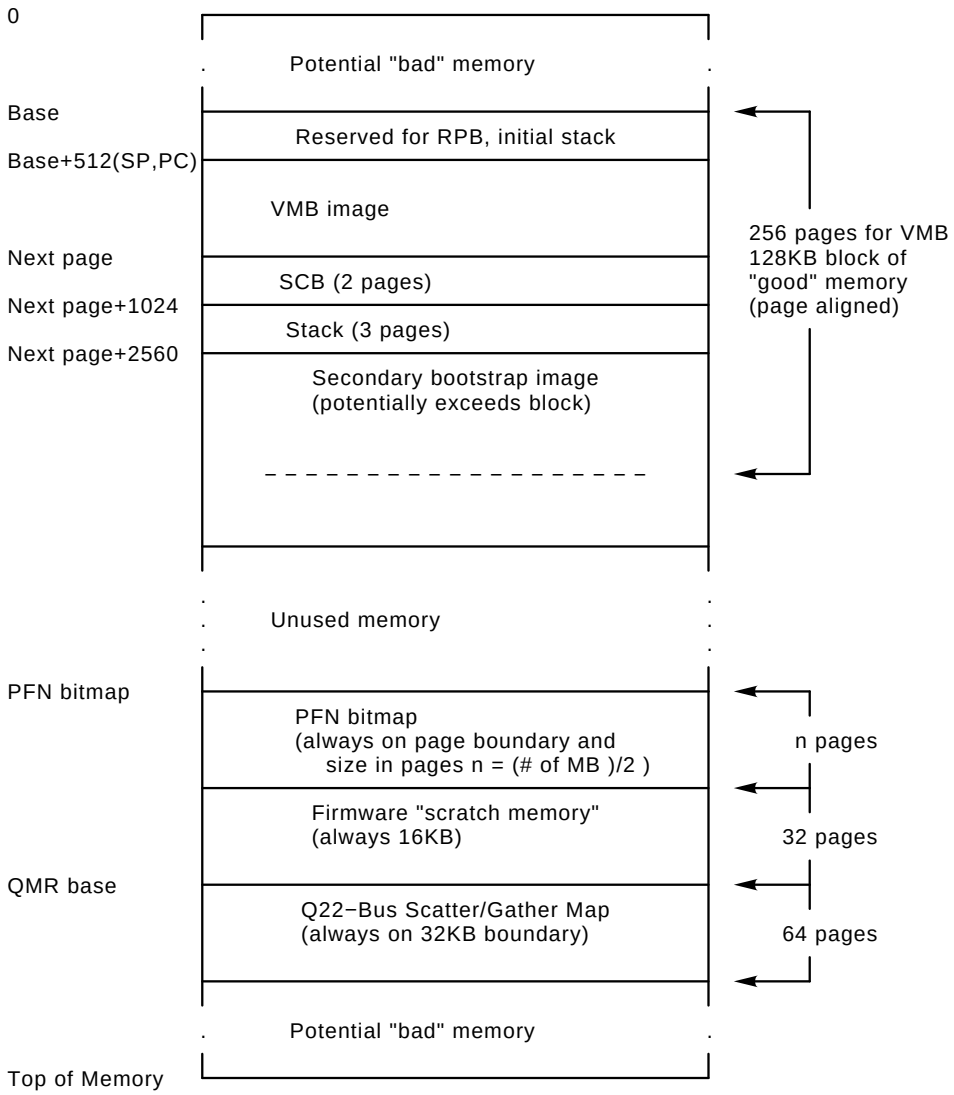
Figure 12–10 Successful Automatic Bootstrap

```
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 12–11.

Figure 12–11 Memory Layout at VMB Exit



In the event an operating system has an extraordinarily large secondary bootstrap that overflows the 128 KB of "good" memory, VMB loads the remainder of the image in memory above the "good" block. However, if there are not enough contiguous "good" pages above the block to load the remainder of the image, the bootstrap fails.

12.2.6.3 Device-Dependent Bootstrap Procedures

As mentioned earlier, the KA680 supports bootstrapping from a variety of boot devices. The following sections describe the various device-dependent boot procedures.

12.2.6.3.1 Disk and Tape Bootstrap Procedure The disk and tape bootstrap supports Files–11 lookup (supporting only the ODS level 2 file structure) or the boot block mechanism (used in PROM boot, also). Of the standard Digital operating systems, VMS and ELN use the Files–11 bootstrap procedure and ULTRIX-32 uses the boot block mechanism.

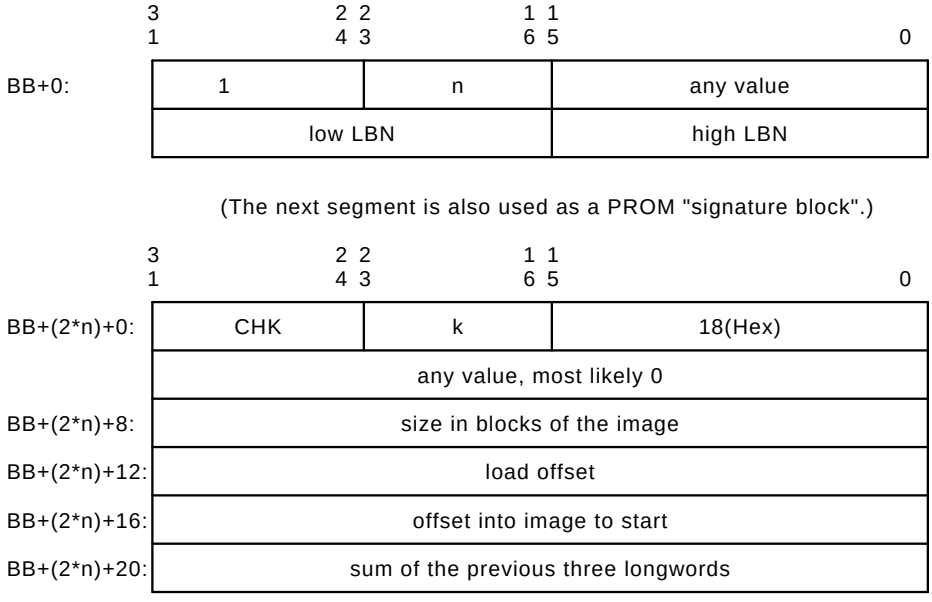
VMB attempts a Files–11 lookup unless the RPB$V_BBLOCK boot flag is set. If VMB determines that the designated boot disk is a Files–11 volume, it searches the volume for the designated boot program, usually [SYS0.SYSEXE]SYSBOOT.EXE. However, VMB can request a diagnostic image or prompt the user for an alternate file specification (Section 12.2.6.1.2). If the boot image cannot be found, VMB fails.

If the volume is not a Files–11 volume or the RPB$V_BBLOCK boot flag was set, the boot block mechanism proceeds as follows:

1. Read logical block 0 of the selected boot device (this is the boot block).
2. Validate that the contents of the boot block conform to the boot block format (Figure 12–12).
3. Use the boot block to find and read in the secondary bootstrap.
4. Transfer control to the secondary bootstrap image (the same as for a Files–11 boot).

The format of the boot block must conform to that shown in Figure 12–12.

Figure 12–12 Boot Block Format



Where:

1) the 18(hex) indicates this is a VAX instruction set
2) 18(hex) + "k" = the one's complement of "CHK"

12.2.6.3.2 PROM Bootstrap Procedure The PROM bootstrap uses a variant of the boot block mechanism. VMB searches for a valid PROM "signature block," the second segment of the boot block defined in Figure 12–12. If PRA0 is the selected "device," then VMB searches through Q22–bus memory on 16 KB boundaries. If the selected "device" is PRB0, VMB checks the top 4096-byte block of the FEPROM.

At each boundary, VMB :

1. Validates the readability of that Q22–bus memory page
2. If readable, checks to see if it contains a valid PROM signature block

If verification passes, the PROM image will be copied into main memory and VMB will transfer control to that image at the offset specified in the PROM boot block. If not, the next page will be tested.

Note that it is not necessary that the boot image actually reside in PROM. Any boot image in Q22–bus memory space with a valid signature block on a 16 KB boundary is a candidate. Indeed, auxiliary bootstrap assumes that the image is in shared memory.

The PROM image is copied into main memory in 127 page "chunks" until the entire PROM is moved. All destination pages beyond the primary 128 KB block are verified to make sure they are marked good in the PFN bitmap. The PROM must be copied contiguously, and if all required pages cannot fit into the memory immediately following the VMB image, the boot fails.

12.2.6.3.3 Network Bootstrap Procedure Whenever a network bootstrap is selected on a KA680, the VMB code makes continuous attempts to boot from the network. VMB uses the DNA maintenance operations protocol (MOP) as the transport protocol for network bootstraps and other network operations. (Refer to Appendix E for a complete description of supported MOP functions during bootstrap.) Once a network boot has been invoked, VMB turns on the designated network link and repeats load attempts until either a successful boot occurs, a fatal controller error occurs, or VMB is halted from the operator console.

The KA680 supports the load of a standard operating system, a diagnostic image, or a user-designated program via network bootstraps. The default image is the a standard operating system; however, a user may select an alternate image by setting either the RPB$V_DIAG bit or the RPB$V_SOLICT bit in the boot flag longword R5. Note that the RPB$V_SOLICT bit has precedence over the RPB$V_ DIAG bit. If both bits are set, then the solicited file is requested. (Refer to Figure 12–9 for the usage of these bits.)

VMB accepts a maximum of 39 characters for a file specification for solicited boots. However, MOP V3 only supports a 16-character file name. If the network server is running VMS, the following defaults apply to the file specification: the directory MOM$LOAD: and an extension .SYS. Therefore, the file specification need only consist of the file name if the default directory and extension attributes are used.

The KA680 VMB uses the MOP program load sequence for bootstrapping the module and the MOP "dump/load" protocol type for load-related message exchanges. The types of MOP message used in the exchange are listed in Table E–1 and Table E–2 in Appendix E.

- VMB, the requester, starts by sending a REQ_PROGRAM message in the appropriate envelope (Table E–3 in Appendix E) to the MOP "dump/load" multicast address (Table E–4 in Appendix E). It then waits for a response in the form of a VOLUNTEER message from another node on the network, the MOP load server. If a response is received, then the destination address is changed from the multicast address to the node address of the server. The same REQ_PROGRAM message is retransmitted to the server as an acknowledge, which initiates the load.
- Next, VMB begins sending REQ_MEM_LOAD messages in response to any of the following:
  - A MEM_LOAD message, while there is still more to load
  - A MEM_LOAD_w_XFER, if it is the end of the image
  - A PARAM_LOAD_w_XFER, if it is the end of the image and operating system parameters are required
- The "load number" field in the load messages is used to synchronize the load sequence. At the beginning of the exchange, both the requester and server initialize the load number. The requester only increments the load number if a load packet has been successfully received and loaded. This forms the acknowledge to each exchange. The server will resend a packet with a specific load number until it sees a request with the load number incremented. The final acknowledge is sent by the requester and has a load number equivalent to the load number of the appropriate LOAD_w_XFER message + 1.
- Because the request for load assistance is a MOP "must transact" operation, the network bootstrap continues indefinitely until a volunteeer is found. The REQ_PROGRAM message is sent out in bursts of eight at 4-second intervals, the first four in MOP Version 4 IEEE 802.3 format and the last four in MOP Version 3 Ethernet format. The backoff period between bursts doubles each cycle from an initial value of four seconds, to eight seconds... up to a maximum of five minutes. However, to reduce the likelihood of many nodes posting requests in lock-step, a random "jitter" is applied to the backoff period. The actual backoff time is computed as (.75+(.5*RND(x)))*BACKOFF, where 0<=x<1.

12.2.7 Operating System Restart

An operating system restart is the process of bringing up the operating system from a known initialization state following a processor halt. This procedure is often called *restart* or *warmstart*, and should not be confused with a processor restart, which results in firmware entry.

On the KA680, a restart occurs if the conditions specified in Table 12–1 are satisfied.

To restart a halted operating system, the firmware searches system memory for the restart parameter block (RPB), a data structure constructed for this purpose by VMB. (Refer to Table B–2 in Appendix B for a detailed description of this data structure.) If a valid RPB is found, the firmware passes control to the operating system at an address specified in the RPB.

The firmware keeps a "restart in progress" (RIP) flag in CPMBX, which it uses to avoid repeated attempts to restart a failing operating system. An additional "restart in progress" flag is maintained by the operating system in the RPB.

The firmware uses the following algorithm to restart the operating system:

1. Check CPMBX<3>(RIP). If it is set, restart fails.
2. Print the message "Restarting system software." on the console terminal.
3. Set CPMBX<3>(RIP).
4. Search for a valid RPB. If none is found, restart fails.
5. Check the operating system RPB$L_RSTRTFLG<0>(RIP) flag. If it is set, restart fails.
6. Write "0" on the diagnostic LEDs.
7. Dispatch to the restart address, RPB$L_RESTART, with :

   SP = The physical address of the RPB plus 512
   AP = The halt code
   PSL = 041F0000
   PR$_MAPEN = 0

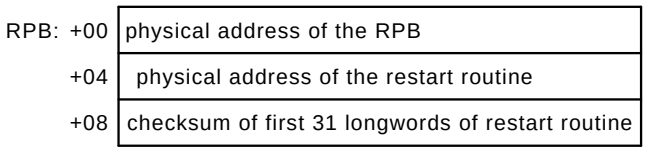
If the restart is successful, the operating system must clear CPMBX<3>(RIP).

If restart fails, the firmware prints "Restart failure." on the system console.

12.2.7.1 Locating the RPB

The RPB is a page-aligned control block, which can be identified by the first three longwords. The format of the RPB "signature" is shown in Figure Figure 12–13. (Refer to Table B–2 in Appendix B for a complete description of the RPB.)

Figure 12–13 Locating the Restart Parameter Block



The firmware uses the following algorithm to find a valid RPB:

1. Search for a page of memory that contains its address in the first longword. If none is found, the search for a valid RPB has failed.
2. Read the second longword in the page (the physical address of the restart routine). If it is not a valid physical address, or if it is zero, return to step 1. The check for zero is necessary to ensure that a page of zeros does not pass the test for a valid RPB.
3. Calculate the 32-bit twos-complement sum (ignoring overflows) of the first 31 longwords of the restart routine. If the sum does not match the third longword of the RPB, return to step 1.
4. A valid RPB has been found.

12.2.8 Console Service

By definition, the KA680 is "halted" whenever the console program is running and the triple angle prompt ">>>" is displayed on the console terminal. When the processor is halted, the firmware provides most of the services of a standard VAX console through the device that is designated as the system console. The firmware also implements several commands not defined in the *VAX Architecture Reference Manual.* For a summary of the console commands, see Table 12–16.

12.2.8.1 Console Control Characters

In console I/O mode, several characters have special meanings.

Table 12–5 Console Control Characters

Keyboard Key	Control Character	Meaning
RETURN	Carriage Return	Ends a command line. No action is taken on a command until after it is terminated by a carriage return. A null line terminated by a carriage return is treated as a valid, null command. No action is taken, and the console reprompts for input. Carriage return is echoed as carriage return, line feed.
X	Delete Character	When the operator types rubout, the console deletes the character that the operator previously typed. What appears on the console terminal depends on whether the terminal is a video terminal or a hard copy terminal. For hard copy terminals, when a rubout is typed, the console echoes with a backslash ("<backslash>"), followed by the character being deleted. If the operator types additional rubouts, the additional characters deleted are echoed. When the operator types a nonrubout character, the console echoes another backslash, followed by the character typed. The result is to echo the characters deleted, surrounding them with backslashes. For video terminals, when RUBOUT is typed, the previous character is erased from the screen, and the cursor is restored to its previous position. The console does not delete characters past the beginning of a command line. If the operator types more rubouts than there are characters on the line, the extra rubouts are ignored. If a RUBOUT is typed on a blank line, it is ignored.
Ctrl/A	Control-A or F14	Toggle insertion/overstrike mode for command line editing. By default, the console powers up to overstrike mode.

(continued on next page)

Table 12–5 (Cont.) Console Control Characters

Keyboard Key	Control Character	Meaning
Ctrl/B	Control-B or up_arrow (or down_arrow)	Recall previous command(s). Command recall is only operable if sufficient memory is available. This function may then be enabled and disabled using the SET RECALL command.
Ctrl/C	Control-C	Causes the console to echo ^C and to abort processing of a command. Control-C has no effect as part of a binary load data stream. Control-C reenables output stopped by Control-O.
Ctrl/D	Control-D or left_ arrow	Moves the cursor left one position.
Ctrl/E	Control-E	Moves the cursor to the end of the line.
Ctrl/F	Control-F or right_ arrow	Moves the cursor right one position.
Ctrl/H	Control-H, BACKSPACE or F12	Moves the cursor to the beginning of the line.
Ctrl/O	Control-O	Causes the console to throw away transmissions to the console terminal until the next Control-O is entered. Control-O is echoed as ^O<CR> when it disables output, but is not echoed when it reenables output. Output is reenabled if the console prints an error message, or if it prompts for a command from the terminal. Displaying a REPEAT command does not reenable output. When output is reenabled for reading a command, the console prompt is displayed. Output is also enabled by Control-S.
Ctrl/Q	Control-Q	Causes the output to the console terminal to resume. Additional control-Qs are ignored. Control-S and control-Q are not echoed.
Ctrl/S	Control-S	Stops output to the console terminal until control-Q is typed. Control-S and Control-Q are not echoed.
Ctrl/U	Control-U	The console echoes ^U<CR>, and deletes the entire line. If Control-U is typed on an empty line, it is echoed, and the console prompts for another command.
Ctrl/R	Control-R	Causes the console to echo <CR><LF> followed by the current command line. This function can be used to improve the readability of a command line that has been heavily edited. When Control-C is typed as part of a command line, the console deletes the line as it does with Control-U.
BREAK	BREAK	If the console is in console I/O mode, BREAK is equivalent to Control-C and is echoed as "^C".

Note

If the local console is in program I/O mode and halts are disabled, BREAK is ignored. If the console is in program I/O mode and halts are enabled, BREAK causes the processor to halt and enter console I/O mode.

Control characters are typed by pressing the character key while holding down the control key.

If an unrecognized control character (ASCII code less than 32 decimal or between 128 and 159 decimal) is typed, it is echoed as up arrow followed by the character with ASCII code 64 greater. For example, BEL (ASCII code 7) is echoed as "^G", because capital G is ASCII code 7+64=71. When a control character is deleted 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 will respond with an error message.

12.2.8.2 Console Command Syntax

The console accepts commands of lengths up to 80 characters. It responds to longer commands with an error message. The count does not include rubouts, rubbed out characters, or the terminating carriage return.

Commands may be abbreviated. Abbreviations are formed by dropping characters from the end of a keyword, as long as the resulting keyword is still unique. Most commands can be uniquely expressed with their first characters.

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, and is started with a slash (ASCII code 47 decimal).

All numbers (addresses, data, counts) are in hexadecimal. Note, though, that symbolic register names number the registers in decimal. The console does not distinguish between upper- and lowercase either in numbers or in commands; both are accepted.

12.2.8.3 Console Command Keywords

The KA680 firmware implements a variant of the VAX console command set. The only commands defined in the VAX SRM and not supported by the KA680 are MICROSTEP, LOAD, and @. The CONFIGURE, HELP, MOVE, SEARCH and SHOW commands have been added to the command set to facilitate system debugging and access to system parameters. In general, however, the KA680 console is similar to other VAX consoles.

Table 12–6 lists command and qualifier keywords.

Table 12–6 Command, Parameter, and Qualifier Keywords

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

(continued on next page)

Table 12–6 (Cont.) Command, Parameter, and Qualifier Keywords

Qualifier Keywords		
Data Control	**Address Space Control**	**Command Specific**
		/SE*RVICE

"*" indicates the minimal number of characters that are required to uniquely identify the keyword.

A complete summary of the console commands is provided in Table 12–16 following the command descriptions.

12.2.8.4 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 12–17 is a summary of the qualifers recognized by the KA680 console.

12.2.8.5 Console Numeric Expression Radix Specifiers

By default, the console treats any numeric expression used as an address or a datum as a hexadecimal integer. The user may override the default radix by using one of the specifiers listed in Table 12–7.

Table 12–7 Console Radix Specifiers

Form 1	Form 2	Radix
%b	^b	Binary
%o	^o	Octal
%d	^d	Decimal
%x	^x	Hexadecimal, default

For instance, the value 19 is by default hexadecimal, but it may also be represented as %b11001, %o31, %d25, and %x19 (or in the alternate form as ^b11001, ^o31, ^d25, and ^x19).

12.2.8.6 Command Address Specifiers

Several commands take an address or addresses as arguments. In the context of the console, an address has two components, the address space, and the offset into that space. The console supports 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), and the PSL (/M qualifier).

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

12.2.8.7 Console Symbolic Addressing

The KA680 console supports symbolic references to addresses. A symbolic reference simultaneously defines the address space for a given symbol. Table 12–8 lists the symbols that use the last referenced address and address space to compute the effective address. Table 12–9, Table 12–10, and Table 12–11 list the symbolic addresses supported by the console grouped according to address space.

Table 12–8 Console Symbols Using Last Referenced Address

"*"	The last location successfully referenced in an EXAMINE or DEPOSIT command.
"+"	The location immediately following the last location successfully referenced in an EXAMINE or DEPOSIT command. For references to physical or virtual memory spaces, the location referenced is the last address, plus the size of the last reference (1 for byte, 2 for word, 4 for longword, 8 for quadword). For other address spaces, the address is the last address referenced plus one.
"-"	The location immediately preceding the last location successfully referenced in an EXAMINE or DEPOSIT command. For references to physical or virtual memory spaces, the location referenced is the last address minus the size of this reference (1 for byte, 2 for word, 4 for longword, 8 for quadword). For other address spaces, the address is the last address referenced minus one.
"@"	The location addressed by the last location successfully referenced in an EXAMINE or DEPOSIT command.

Table 12–9 Console Symbols for General-Purpose Registers - /G

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
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	—	—	—	—	—		

Table 12–10 Console Symbols for Internal/External Processor Registers - /I

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
pr$_ksp	00	pr$_pcbb	10	pr$_rxcs	20	—-	30
pr$_esp	01	pr$_scbb	11	pr$_rxdb	21	—-	31
pr$_ssp	02	pr$_ipl	12	pr$_txcs	22	—-	32
pr$_usp	03	pr$_astlv	13	pr$_txdb	23	—-	33
pr$_isp	04	pr$_sirr	14	—-	24	—-	34
—-	05	pr$_sisr	15	—-	25	—-	35
—-	06	—-	16	pr$_mcesr	26	—-	36
—-	07	—-	17	—-	27	pr$_ioreset	37
pr$_p0br	08	pr$_iccs	18	—-	28	pr$_mapen	38
pr$_p0lr	09	pr$_nicr	19	—-	29	pr$_tbia	39
pr$_p1br	0A	pr$_icr	1A	pr$_savpc	2A	pr$_tbis	3A
pr$_p1lr	0B	pr$_todr	1B	pr$_savpsl	2B	—-	3B
pr$_sbr	0C	—-	1C	—-	2C	—-	3C
pr$_slr	0D	—-	1D	—-	2D	—-	3D
—-	0E	—-	1E	—-	2E	pr$_sid	3E
—-	0F	—-	1F	—-	2F	pr$_tbchk	3F
pr$_ecr	7D	—-	—-	—-	—-	—-	—-
pr$_cctl	A0	pr$_neoadr	B0	pr$_vmar	D0	—-	F0
—-	A1	—-	B1	pr$_vtag	D1	—-	F1

(continued on next page)

Table 12–10 (Cont.) Console Symbols for Internal/External Processor Registers - /I

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
pr$_bcdecc	A2	pr$_neocmd	B2	pr$_vdata	D2	pr$_pcadr	F2
pr$_bcetsts	A3	—-	B3	pr$_icsr	D3	—-	F3
pr$_bcetidx	A4	pr$_nedathi	B4	—-	D4	pr$_pcsts	F4
pr$_bcetag	A5	—-	B5	—-	D5	—-	F5
pr$_bcedsts	A6	pr$_nedatlo	B6	—-	D6	—-	F6
pr$_bcedidx	A7	—-	B7	pr$_pamode	E7	—-	F7
pr$_bcedecc	A8	pr$_neicmd	B8	—-	E8	pr$_pcctl	F8
pr$_cefadr	AB	—-	B9	—-	E9	—-	F9
pr$_cefsts	AC	—-	BA	pr$_tbadr	EC	—-	FA
pr$_nests	AE	—-	BB	pr$_tbsts	ED	—-	FB
pr$_bctag	01000000	pr$_bcflush	01400000	pr$_pctag	01800000	pr$_pcdap	01C00000

Table 12–11 Console Symbols for VAX Physical I/O Space Registers - /P

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
qbio	20000000	qbmem	30000000	qbmbr	20 080010	—-	—-
rom	20040000	—-	—-	bdr	20084004	—-	—-
scr	20080000	dser	20080004	qbear	20 080008	dear	2008000C
ipcr0	20001f40	ipcr1	20001f42	ipcr2	20001f44	ipcr3	20001f46
ssc_ram	20140400	ssc_cr	20140010	ssc_cbtcr	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
nicsr0	20008000	nicsr1	20008004	nicsr2	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_txpoll	20008004	sgec_rxpoll	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

(continued on next page)

Table 12–11 (Cont.) Console Symbols for VAX Physical I/O Space Registers - /P

Symbol	Address	Symbol	Address	Symbol	Address	Symbol	Address
shac1_pesr	20004050	shac1_pfar	20004054	shac1_ppr	20004058	shac1_pmcsr	2000405C
shac1_pcq0cr	20004080	shac1_pcq1cr	20004084	shac1_pcq2cr	20004088	shac1_pcq3cr	2000408C
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
nmccwb	21000110	—-	—-	—-	—-	—-	—-
memcon0	21018000	memcon1	21018004	memcon2	21018008	memcon3	2101800c
memcon4	21018010	memcon5	21018014	memcon6	21018018	memcon7	2101801c
memsig8	21018020	memsig9	21018024	memsig10	21018028	memsig11	2101802c
memsig12	21018030	memsig13	21018034	memsig14	21018038	memsig15	2101803c
mear	21018040	mser	21018044	nmcdsr	21018048	moamr	2101804C
cear	21020000	ncadsr	21020004	csear1	21020008	csear2	2102000c
cpioea1	21020010	cpioar2	21020014	ndear	21020018	—-	—-

12.2.8.8 References to Processor Registers and Memory

The KA680 console is implemented in VAX macrocode executing from FEPROM. Actual processor registers cannot be modified by the console command interpreter. When the console is entered, the console saves the processor registers in console memory and all command references to them are directed to the corresponding saved values, not to the registers themselves.

When the console reenters program I/O mode, the saved registers are restored and 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, and any direct reference to these registers will be intercepted by the console and the access will be to the saved copies:

- R0...R15 - The general-purpose registers (GPRs)
- PR$_IPL - The interrupt priority level register (IPL)
- PR$_SCBB - The system control block base register (SCBB)
- PR$_ISP - The interrupt stack pointer (ISP)
- PR$_MAPEN - The memory management enable register (MAPEN)
- PR$_ECR - Ebox control register (ECR)

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

- PR$_SAVPC - The halt PC (SAVPC)
- PR$_SAVPSL - The halt PSL (PSL)
- ADxMCH/ADxMSK - The SSC address decode and match registers (BDMTR, BDMKR)
- SSCCR - The SSC configuration register
- DLEDR - The SSC diagnostic LED register

12.2.9 Console Commands

The following sections define the commands accepted by the console when the KA680 is in console I/O mode. The following conventions are used to describe command syntax:

Syntax Conventions

The following conventions are used to describe command syntax:

Table 12–12 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:

Table 12–13 Default Radix

%d	Decimal (for example, %d1234)
%x	Hexadecimal (for example, %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}
Boot flags is a 32-bit hex value that is passed to VMB in R5. No interpretation of this value is performed by the console. Refer to Figure 12–9 for the bit assignments of R5. A default boot flags longword may be specified using the SET BFLAG command and displayed with the SHOW BFLAG command.

/{boot_flags}
Equivalent to the form above.

Arguments

[{boot_device}]
The boot device name may be any arbitrary character string, with a maximum length of 17 characters. Longer strings cause a "VAL TOO BIG" error message to be issued from the console. Otherwise, the console makes no attempt at interpreting or validating the device name. The console converts the string to all uppercase, and passes to VMB a string descriptor to this device name in R0. A default boot device may be specified using the SET BOOT command and displayed with the SHOW BOOT command. The factory default is the Ethernet device, EZA0.

Description

The console initializes the processor and transfers execution to VMB. VMB attempts to boot the operating system from the specified device or the default boot device if none is specified.

If a list of devices is specified, VMB attempts to boot from each device, in turn, and then transfers control to the first successfully booted image. In a list, network devices should always be placed last because network bootstraps only terminate if a fatal hardware error occurs or an image is successfully loaded.

The console qualifies the bootstrap operation by passing a boot flags to VMB in R5. A more detailed description of the bootstrap process and how the default bootstrap device is determined is described in Section 12.2.6.

In case either the qualifiers or the device name is absent, the corresponding default value is used. Explicitly stating the boot flags or the boot device overrides the current default value for the current boot request, but does not change the corresponding default value in NVRAM.

There are three mechanisms by which the default boot device and and boot flags may be set.

1. The operating system may write a default boot device and flags into the appropriate locations in NVRAM (Appendix A).
2. The user may explicitly set the default boot device and boot flags with the console SET BOOT and SET BFLAG commands, respectively.

3. The console will prompt the user for the default boot device if any of the following conditions are met:
   - The power-up mode switch is set to "query" mode.
   - The console detects that the battery failed; therefore, the contents of NVRAM are no longer valid.
   - The console detects that the default boot device has not been explicitly set by the user. Either a previous device query timed out and defaulted to EZA0 or neither (1) nor (2) has been performed. Simply stated, the console will prompt the user on every powerup for a default boot device, until such a request has been satisfied.

On powerup, if no default boot device is specified in NVRAM, the console issues a list of potential bootable devices and then queries the user for a device name. If no device name is entered within 30 seconds, EZA0 is used. However, EZA0 does not become the "default" boot device.

Examples

```
>>>show boot
DUA0
>>>show bflag
0
>>>b                    ! Boot using default boot flags and device.
(BOOT/R5:0 DUA0)

  2..
-DUA0

>>>bo EZA0              ! Boot using default boot flags and specified device.
(BOOT/R5:0 EZA0)

  2..
-EZA0

>>>boot/10              ! Boot using specified boot flags and default device.
(BOOT/R5:10 DUA0)

  2..
-DUA0

>>>boot /r5:220 EZA0    ! Boot using specified boot flags and device.
(BOOT/R5:220 EZA0)

  2..
-EZA0

>>>boot dia0,mua0,eza0
(BOOT/R5:0 DIA0,MUA0,EZA0)

  2..
```

CONFIGURE

Format

CONFIGURE

Qualifiers

None.

Arguments

None.

Description

CONFIGURE is similar to the VMS SYSGEN CONFIG utility. This feature simplifies system configuration by providing information that is typically available only with a running operating system.

The CONFIGURE command invokes an interactive mode that permits the user to enter Q22–bus device names, then generates a table of Q22–bus I/O page device CSR addresses and device vectors.

Examples

```
>>>configure
Enter device configuration, HELP, or EXIT
Device,Number? help
Devices:
 LPV11        KXJ11        DLV11J       DZQ11        DZV11        DFA01
 RLV12        TSV05        RXV21        DRV11W       DRV11B       DPV11
 DMV11        DELQA        DEQNA        DESQA        RQDX3        KDA50
 RRD50        RQC25        KFQSA-DISK   TQK50        TQK70        TU81E
 RV20         KFQSA-TAPE   KMV11        IEQ11        DHQ11        DHV11
 CXA16        CXB16        CXY08        VCB01        QVSS         LNV11
 LNV21        QPSS         DSV11        ADV11C       AAV11C       AXV11C
 KWV11C       ADV11D       AAV11D       VCB02        QDSS         DRV11J
 DRQ3B        VSV21        IBQ01        IDV11A       IDV11B       IDV11C
 IDV11D       IAV11A       IAV11B       MIRA         ADQ32        DTC04
 DESNA        IGQ11        DIV32        KIV32        DTCN5        DTC05
 KWV32        KZQSA        M7577        LNV24        M7576        DEQRA
Numbers:
 1 to 255, default is 1
Device,Number? kda50
Device,Number? kfqsa
Device is ambiguous
Device,Number? kfqsa-disk
Device,Number? kfqsa-tape
Device,Number? cxy08
Device,Number? cxa16
Device,Number? exit

Address/Vector Assignments
-772150/154 KDA50
-760334/300 KFQSA-DISK
-774500/260 KFQSA-TAPE
-760500/310 CXY08
-760520/320 CXA16
>>>
```

CONTINUE

Format

CONTINUE

Qualifiers

None.

Arguments

None.

Description

The processor begins instruction execution at the address currently contained in the program counter. Processor initialization is not performed. The console enters program I/O mode. Internally, the CONTINUE command pushes the user's PC and PSL onto the user's ISP, and then executes an REI instruction. This implies that the user's ISP is pointing to some valid memory.

Examples

```
>>>continue
>>>
```

DEPOSIT

Format

DEPOSIT [qualifier_list] {address} {data} [{data}...]

Qualifiers

/B
The data size is byte.

/W
The data size is word.

/L
The data size is longword.

/Q
The data size is quadword.

/G
The address space is the general-purpose register set, R0 through R15. The data size is always long.

/I
The address space is internal processor registers (IPRs). These are the registers only accessible 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 occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. Virtual space DEPOSITs cause the PTE<M> bit to be set. If memory mapping is not enabled, virtual addresses are equal to physical addresses.

/U
Access to console private memory is allowed. This qualifier also disables virtual address protection checks. On virtual address writes, the PTE<M> bit will not be set if the /U qualifier is present. This qualifier is not inherited, and must be respecified on each command.

/N:{count}
The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address "-", the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession. For repeated references to preceding addresses, use REPEAT DEPOSIT - <DATA>.

/STEP:{size}
The number to add to the current address. Normally this defaults to the data size, but is overridden by the presence of this qualifier. This qualifier is not inherited.

/WRONG
The ECC bits for this data are forced to the value of 3. (ECC bits of 3 will always generate a double bit error.)

Arguments

{address}
A longword address that specifies the first location into which data is deposited. The address can be any legal address specifier as defined in Section 12.2.8.6.

{data}
The data to be deposited. If the specified data is larger than the deposit data size, the console ignores the command and issues an error response. If the specified data is smaller than the deposit data size, it is extended on the left with zeros.

[{data}]
Additional data to be deposited (up to a maximum of six values).

Description

Deposits the data into the address specified. If no address space or data size qualifiers are specified, the defaults are the last address space and data size used in a DEPOSIT, EXAMINE, MOVE, or SEARCH command. After processor initialization, the default address space is physical memory, the default data size is a longword, and the default address is zero. If conflicting address space or data sizes are specified, the console ignores the command and issues an error response.

Examples

```
>>>d/p/b/n:1FF 0 0              ! Clear first 512 bytes of physical memory.
>>>d/v/l/n:3 1234 5             ! Deposit 5 into four longwords starting at
                                  virtual memory address 1234.
>>>d/n:8 R0 FFFFFFFF            ! Loads GPRs R0 through R8 with -1.
>>>d/n:200 - 0   ! Starting at previous address, clear 513 bytes.
>>>d/l/p/n:10/s:200 0 8         ! Deposit 8 in the first longword of
                                  the first 17 pages in physical memory.
>>>
```

EXAMINE

Format

EXAMINE [qualifier_list] [{address}]

Qualifiers

/B
The data size is byte.

/W
The data size is word.

/L
The data size is longword.

/Q
The data size is quadword.

/G
The address space is the general-purpose register set, R0 through R15. The data size is always long.

/I
The address space is internal processor registers (IPRs). These are the registers only accessible 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, and must be respecified with each command.

/N:{count}
The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address "-", the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction

of succession. For repeated references to preceding addresses, use "REPEAT EXAMINE - <DATA>".

/STEP:{size}
The number to add to the current address. Normally this defaults to the data size, but is overridden by the presence of this qualifier. This qualifier is not inherited.

/WRONG
ECC errors on this read access to main memory are ignored. Also, if specified, the ECC bits actually read are displayed in parentheses following the datum. In the case of quadword and octaword data, the ECC bits shown apply to the most significant longword only.

/INSTRUCTION
Disassemble and display the VAX Macro-32 instruction at the specified address.

Arguments

[{address}]
A longword address that specifies the first location to be examined. The address can be any legal address specifier as defined in Section 12.2.8.6. If no address is specified, "+" is assumed.

Description

Examines the contents of the memory location or register specified by the address. If no address is specified, "+" is assumed. The display line consists of a single character address specifier, the hexadecimal physical address to be examined, and the examined data also in hexadecimal.

EXAMINE uses the same qualifiers as DEPOSIT. However, the /WRONG qualifier will cause EXAMINEs to ignore ECC errors on reads from physical memory. Additionally, the EXAMINE command supports an /INSTRUCTION qualifier, which will disassemble the instructions at the current address.

Examples

```
>>>ex pc                                          ! Examine the PC.
  G 0000000F FFFFFFFC
>>>ex sp                                          ! Examine the SP.
  G 0000000E 00000200
>>>ex psl                                         ! Examine the PSL.
  M 00000000 041F0000
>>>e/m                                            ! Examine PSL another way.
  M 00000000 041F0000
>>>e r4/n:5                                       ! Examine R4 through R9.
  G 00000004 00000000
  G 00000005 00000000
  G 00000006 00000000
  G 00000007 00000000
  G 00000008 00000000
  G 00000009 801D9000
>>>ex pr$_scbb                                    ! Examine the SCBB, IPR 17.
  I 00000011 2004A000
>>>e/p 0                                          ! Examine local memory 0.
  P 00000000 00000000
>>>ex /ins 20040000                               ! Examine 1st byte of EPROM.
  P 20040000   11 BRB     20040019
>>>ex /ins/n:5 20040019                           ! Disassemble from branch.
  P 20040019   D0 MOVL    I^#20140000,@#20140000
  P 20040024   D2 MCOML   @#20140030,@#20140502
  P 2004002F   D2 MCOML   S^#0E,@#20140030
  P 20040036   7D MOVQ    R0,@#201404B2
  P 2004003D   D0 MOVL    I^#201404B2,R1
  P 20040044   DB MFPR    S^#2A,B^44(R1)
>>>e/ins                                         ! Look at next instruction.
  P 20040048   DB MFPR    S^#2B,B^48(R1)
>>>
```

FIND

Format

FIND [qualifier-list]

Qualifiers

/MEMORY
Search memory for a page-aligned block of good memory, 128 KB 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 physical memory for a restart parameter block. The search does not use the bitmap to qualifiy which pages are looked at. The command leaves the contents of memory unchanged.

Arguments

None.

Description

The console searches main memory starting at address zero for a page-aligned 128 KB segment of good memory, or a restart parameter block (RPB). If the segment or block is found, its address plus 512 is left in SP (R14). If the segment or block is not found, an error message is issued, and the contents of SP are preserved. If no qualifier is specified, /RPB is assumed.

Examples

```
>>>ex sp                                          ! Check the SP.
  G 0000000E 00000000
>>>find /mem                                      ! Look for a valid 128KB.
>>>ex sp                                          ! Note where it was found.
  G 0000000E 00000200
>>>find /rpb                                      ! Check for valid RPB.
?2C FND ERR 00C00004                              ! None to be found here.
>>>
```

HALT

Format

HALT

Arguments

None.

Description

This command has no effect and is included for compatibility with other consoles.

Examples

```
>>>halt                                  ! Pretend to halt.
>>>
```

HELP

Format

HELP

Qualifiers

None.

Arguments

None.

Description

This command has been included to help the console operator answer simple questions about command syntax and usage.

Examples

```
>>>help
Following is a brief summary of all the commands supported by the console:

        UPPERCASE  denotes a keyword that you must type in
        |          denotes an OR condition
        []         denotes optional parameters
        <>         denotes a field specifying a syntactically correct value
        ..         denotes one of an inclusive range of integers
        ...        denotes that the previous item may be repeated

Valid qualifiers:
    /B /W /L /Q /INSTRUCTION
    /G /I /V /P /M
    /STEP: /N: /NOT
    /WRONG /U
```

```
Valid commands:
    BOOT [/R5:<boot_flags> | /<boot_flags>] [<boot_device>[:]]
    CONFIGURE
    CONTINUE
    DEPOSIT [<qualifiers>] <address> [<datum> [<datum>]]
    EXAMINE [<qualifiers>] [<address>]
    FIND [/MEMORY | /RPB]
    HALT
    HELP
    INITIALIZE
    MOVE [<qualifiers>] <address> <address>
    NEXT [count]
    REPEAT <command>
    SEARCH [<qualifiers>] <address> <pattern> [<mask>]
    SET BFL(A)G <boot_flags>
    SET BOOT <boot_device>
    SET CONTROLP <0..1 | DISABLED | ENABLED>
    SET HALT <0..4 | DEFAULT | RESTART | REBOOT | HALT | RESTART_REBOOT>
    SET HOST/DUP/DSSI/BUS<0..1> <node_number> [<task>]
    SET HOST/DUP/UQSSP </DISK | /TAPE> <controller_number> [<task>]
    SET HOST/DUP/UQSSP <physical_CSR_address> [<task>]
    SET HOST/MAINTENANCE/UQSSP/SERVICE <controller_number>
    SET HOST/MAINTENANCE/UQSSP <physical_CSR_address>
    SET LANGUAGE <1..15>
    SET RECALL <0..1 | DISABLED | ENABLED>
    SHOW BFL(A)G
    SHOW BOOT
    SHOW DEVICE
    SHOW DSSI
    SHOW ETHERNET
    SHOW HALT
    SHOW LANGUAGE
    SHOW MEMORY [/FULL]
    SHOW RECALL
    SHOW RLV12
    SHOW QBUS
    SHOW UQSSP
    SHOW SCSI
    SHOW TRANSLATION <physical_address>
    SHOW VERSION
    START <address>
    TEST [<test_code> [<parameters>]]
    UNJAM
    X <address> <count>
>>>
```

Examples

```
>>>help

Following is a brief summary of all the commands supported by the console:

        UPPERCASE  denotes a keyword that you must type in
        |          denotes an OR condition
        []         denotes optional parameters
        <>         denotes a field specifying a syntactically correct value
        ..         denotes one of an inclusive range of integers
        ...        denotes that the previous item may be repeated

Valid qualifiers:
    /B /W /L /Q /INSTRUCTION
    /G /I /V /P /M
    /STEP: /N: /NOT
    /WRONG /U
```

```
Valid commands:
    BOOT [[/R5:]<boot_flags>] [<boot_device>]
    CONFIGURE
    CONTINUE
    DEPOSIT [<qualifiers>] <address> <datum>
 [<datum>...]
    EXAMINE [<qualifiers>] [<address>]
    FIND [/MEMORY | /RPB]
    HALT
    HELP
    INITIALIZE
    MOVE [<qualifiers>] <address> <address>
    NEXT [<count>]
    REPEAT <command>
    SEARCH [<qualifiers>] <address> <pattern>
  [<mask>]
    SET BFLG <boot_flags>
    SET BOOT <boot_device>
    SET CONTROLP <0..1 | DISABLED | ENABLED>
    SET HALT <0..4 | DEFAULT | RESTART | REBOOT | HALT | RESTART_REBOOT>
    SET HOST/DUP/DSSI/BUS:<0..1> <node_number>
 [<task>]
    SET HOST/DUP/UQSSP </DISK | /TAPE> <controller_number>
  [<task>]
    SET HOST/DUP/UQSSP <physical_CSR_address> [<task>]
    SET HOST/MAINTENANCE/UQSSP/SERVICE <controller_number>
    SET HOST/MAINTENANCE/UQSSP <physical_CSR_address>
    SET LANGUAGE <1..15>
    SET RECALL <0..1 | DISABLED | ENABLED>
    SHOW BFLG
    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 <physical_address>
    SHOW UQSSP
    SHOW VERSION
    START <address>
    TEST [<test_code> [<parameters>]]
    UNJAM
    X <address> <count>
```

INITIALIZE

Format

INITIALIZE

Qualifiers

None.

Arguments

None.

Description

A processor initialization is performed. The following registers are initialized, as specified in the *VAX Architecture Standard*.

PSL — 041F0000
IPL — 1F
ASTLVL — 4
SISR — 0
ICCS — bits <6> and <0> are clear, the rest are unpredictable
RXCS — 0
TXCS — 80
MAPEN — 0
CPU cache — flushed
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 KA680 firmware performs the following additional initialization:

The CDAL bus timer is initialized.
The address decode and match registers are initialized.
The programmable timer interrupt vectors are initialized.
The BDR registers are read to determine the baud rate, and then the SSCCR is configured accordingly.
All error status bits are cleared.

Examples

```
>>>init
>>>
```

MOVE

Format

MOVE [qualifier-list] {src_address} {dest_address}

Qualifiers

/B
The data size is byte.

/W
The data size is word.

/L
The data size is longword.

/Q
The data size is quadword.

/P
The address space is physical memory.

/V
The address space is virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. Virtual space MOVEs cause the destination PTE<M> bit to be set. If memory mapping is not enabled, virtual addresses are equal to physical addresses.

/U
Access to console private memory is allowed. This qualifier also disables virtual address protection checks. On virtual address writes, the PTE<M> bit will not be set if the /U qualifier is present. This qualifier is not inherited, and must be respecified on each command.

/N:{count}
The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address "-", the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession.

/STEP:{size}
The number to add to the current address. Normally this defaults to the data size, but is overriden by the presence of this qualifier. This qualifier is not inherited.

/WRONG
On reads, ECC errors on the access of data in main memory are ignored. On writes, the ECC bits for this data are forced to the value of 3. (ECC bits of 3 will always generate a double bit error.)

Arguments

{src_address}
A longword address that specifies the first location of the source data to be copied.

{dest_address}
A longword address that specifies the destination of the first byte of data. These addresses may be any legal address specifier as defined in Section 12.2.8.6. If no address is specified, "+" is assumed.

Description

The console copies the block of memory starting at the source address to a block beginning at the destination address. Typically, this command is used with the /N: qualifier to transfer large blocks of data. The destination will correctly reflect the contents of the source, regardless of the overlap between the source and the data.

The MOVE command actually performs byte, word, longword, and quadword reads and writes as needed in the process of moving the data. Moves are supported for the physical and virtual address spaces only.

Examples

```
>>>ex /n:4 0                              ! Observe destination.
  P 00000000 00000000
  P 00000004 00000000
  P 00000008 00000000
  P 0000000C 00000000
  P 00000010 00000000
>>>ex /n:4 200                            ! Observe source data.
  P 00000200 58DD0520
  P 00000204 585E04C1
  P 00000208 00FF8FBB
  P 0000020C 5208A8D0
  P 00000210 540CA8DE
>>>move /n:4 200 0                        ! Move the data.
>>>ex /n:4 0                              ! Observe the destination.
  P 00000000 58DD0520
  P 00000004 585E04C1
  P 00000008 00FF8FBB
  P 0000000C 5208A8D0
  P 00000010 540CA8DE
>>>
```

NEXT

Format

NEXT {count}

Qualifiers

None.

Arguments

{count}
A value representing the number of macroinstructions to execute.

Description

The NEXT command causes the processor to "step" the specified number of macroinstructions. If no count is specified, "single-step" is assumed. The console then enters "Spacebar Step Mode". In this mode, subsequent spacebar strokes initiate single steps and 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 function. This creates the following restrictions on the usage of the NEXT command:

- If memory management is enabled, the NEXT command works if and only if the first page in SSC RAM is mapped somewhere in S0 (system) space.
- The NEXT command, due to the instructions executed in implementation, does not work where time-critical code is being executed.
- The NEXT command elevates the IPL to 31 for long periods of time (milliseconds) while single-stepping over several commands.
- Unpredictable results occur if the macroinstruction being stepped over modifies the SCBB, or the trace trap entry. This means that the NEXT command cannot be used in conjunction with other debuggers. This also implies that the user should validate PR$_SCCB before using the NEXT command.

Examples

```
>>>dep 1000 50D650D4                        ! Create a simple program.
>>>dep 1004 125005D1
>>>dep 1008 00FE11F9
>>>ex /instruction /n:5 1000                ! List it.
  P 00001000   D4 CLRL    R0
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001009   11 BRB     00001009
  P 0000100B   00 HALT
>>>dep pr$_scbb 200                        ! Set up a user SCBB...
>>>dep pc 1000                              ! ...and the PC.
>>>
>>>n                                        ! Single step...
  P 00001002   D6 INCL    R0                ! SPACEBAR
  P 00001004   D1 CMPL    S^#05,R0          ! SPACEBAR
  P 00001007   12 BNEQ    00001002          ! SPACEBAR
  P 00001002   D6 INCL    R0                ! CR
>>>n 5                                       ! ...or multiple step the program.
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
>>>n 7
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001002   D6 INCL    R0
  P 00001004   D1 CMPL    S^#05,R0
  P 00001007   12 BNEQ    00001002
  P 00001009   11 BRB     00001009
>>>n
  P 00001009   11 BRB     00001009
>>>
```

REPEAT

Format

REPEAT {command}

Qualifiers

None.

Arguments

{command}

Description

The console repeatedly displays and executes the specified command. The repeating is stopped by the operator typing Control-C. Any valid console command can be specified for the command with the exception of the REPEAT command.

Examples

```
>>>repeat ex pr$_todr                          ! Watch the clock.
  I 0000001B 5AFE78CE
  I 0000001B 5AFE78D1
  I 0000001B 5AFE78FD
  I 0000001B 5AFE7900
  I 0000001B 5AFE7903
  I 0000001B 5AFE7907
  I 0000001B 5AFE790A
  I 0000001B 5AFE790D
  I 0000001B 5AFE7910
  I 0000001B 5AFE793C
  I 0000001B 5AFE793F
  I 0000001B 5AFE7942
  I 0000001B 5AFE7946
  I 0000001B 5AFE7949
  I 0000001B 5AFE794C
  I 0000001B 5AFE794F
  I 0000001B 5^C
>>>
```

SEARCH

Format

SEARCH [qualifier_list] {address} {pattern} [{mask}]

Qualifiers

/B
The data size is byte.

/W
The data size is word.

/L
The data size is longword.

/Q
The data size is quadword.

/P
The address space is physical memory. Note that when virtual memory is examined, the address space and address in the response are the translated physical address.

/V
The address space is virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. If memory mapping is not enabled, virtual addresses are equal to physical addresses.

/U
Access to console private memory is allowed. This qualifier also disables virtual address protection checks. This qualifier is not inherited, and must be respecified with each command.

/N:{count}
The address is the first of a range. The first access is to the address specified, then subsequent accesses are made to succeeding addresses. Even if the address is the symbolic address "-", the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession.

/STEP:{size}
The number to add to the current address. Normally this defaults to the data size, but is overridden by the presence of this qualifier. This qualifier is not inherited.

/WRONG
ECC errors on read accesses to main memory are ignored.

/NOT
Inverts the sense of the match.

Arguments

{start_address}
A longword address that specifies the first location subject to the search. This address can be any legal address specifier as defined in Section 12.2.8.6. If no address is specified, "+" is assumed.

{pattern}
The target data.

[{mask}]
A longword containing the bits in the target that are to be "masked" out.

Description

The SEARCH command finds all occurrences of a pattern, and reports the addresses where the pattern was found. If the /NOT qualifier is present, all addresses where the pattern did not match are reported.

The command accepts an optional mask that indicates don't care bits. For example, to ignore bit 0 in the comparison, specify a mask of 1. The mask, if not present, defaults to 0.

Conceptually, a match condition occurs if the following condition is true:

```
(pattern AND NOT mask)  EQUALS  (data AND NOT mask)

where:  pattern -- is the target data.
        mask    -- is the optional don't care bitmask (which defaults to 0).
        data    -- is the data (byte, word, long, quad) at the current address.
```

The command reports the address if the match condition is true, and there is no /NOT qualifier, or if the match condition is false and there is a /NOT qualifier. Stating this in a tabular form:

/NOT Qualifier	Match Condition	Action
absent	true	report address
absent	false	no report
present	true	no report
present	false	report address

The address is advanced by the size of the pattern (byte, word, long, or quad), unless overridden by the /STEP qualifier.

Examples

```
>>>dep /p/l/n:1000 0 0    ! Clear some memory.
>>>
>>>dep 300 12345678    ! Deposit some "search" data.
>>>dep 401 12345678
>>>dep 502 87654321
>>>
>>>search /n:1000 /st:1 0 12345678  ! Search for all occurrences...
  P 00000300 12345678    ! ...of 12345678 on any byte...
  P 00000401 12345678    ! ...boundary.
>>>search /n:1000 0 12345678   ! Then try on longword...
  P 00000300 12345678    ! ...boundaries.
>>>search /n:1000 /not 0 0   ! Search for all 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 interpreted as...
>>>                                          ! ...a "spin" (10$: brb 10$).
>>>      ! Note, none found.
```

SET

Format

SET {parameter} {value}

Qualifiers

-
Depends on the parameters used

Arguments

None.

Description

Sets the indicated console parameter to the indicated value. The following are console parameters and their acceptable values:

Parameters

BFL(A)G
Sets the default R5 boot flags. The value must be a hexadecimal number of up to 8 hex digits.

BOOT
Sets the default boot device. The value must be a valid device name or device list as specified in Section 12.2.9, Console Commands (on the BOOT command).

CONTROLP
Sets Control-P as the console halt condition, instead of a BREAK. Values of 1 or ENABLED set Control-P recognition. Values of 0 or DISABLED set BREAK recognition. In either case, the setting of the BREAK Enable switch will determine whether or not a halt will occur.

HALT
Sets the user-defined halt action. Acceptable values are 0 through 4 or the ordinal keywords DEFAULT, RESTART, REBOOT, HALT, and RESTART_REBOOT. Refer to Table 12–1 for usage.

HOST
Invokes the DUP or MAINTENANCE driver on the selected node. Only SET HOST /DUP accepts a value parameter. The hierarchy of the SET HOST qualifiers listed below suggests the appropriate usage. Each qualifier only supports the additional qualifiers at levels below it.

/DUP
Uses the DUP protocol to examine/modify parameters of a device on either of the DSSI buses or the Q22–bus. The optional value for SET HOST /DUP is a "task" name for the selected DUP driver to execute.

Note

The KA680 DUP driver only supports "SEND DATA IMMEDIATE" messages, and those devices that also support them.

/BUS
Selects the desired KA680 DSSI bus. A value of 0 selects DSSI bus 0 (internal backplane bus). A value of 1 selects DSSI bus 1 (external console module bus).

/DSSI Node
Selects the DSSI node, where "node" is a number from 0 to 7.

/UQSSP
Selects the Q22–bus device using one of the following three methods.

/DISK n — Specifies the disk controller number, where "n" is from 0 to 255. (The resulting fixed address for n=0 is 20001468 and the floating rank for n>0 is 26.)
/TAPE n — Specifies the tape controller number, where "n" is from 0 to 255. (The resulting fixed address for n=0 is 20001940 and the floating rank for n>0 is 30.)
csr_address — Specifies the Q22–bus I/O page CSR address for the device.

/MAINTENANCE
Uses the MAINTENANCE protocol to examine/modify KFQSA EEPROM configuration parameters. Note that SET HOST /MAINTENANCE does not accept a "task" value.

/UQSSP —

/SERVICE n — Specifies the KFQSA controller number "n" of a KFQSA in service mode, where "n" is from 0 to 3. (The resulting fixed address of a KFQSA in service mode is 20001910+4*n.)
csr_address — Specifies the Q22–bus I/O page CSR address for the KFQSA.

LANGUAGE
Sets console language and keyboard type. If the current console terminal does not support the Digital Multinational Character Set (MCS), then this command has no effect and the console remains in English message mode. Acceptable values are 1 through 15 and have the following meaning:

1) Dansk
2) Deutsch (Deutschland/Österreich)
3) Deutsch (Schweiz)
4) English (United Kingdom)
5) English (United States/Canada)
6) Español
7) Français (Canada)
8) Français (France/Belgique)
9) Français (Suisse)
10) Italiano
11) Nederlands
12) Norsk
13) Português

14) Suomi
15) Svenska

RECALL
Sets command recall state to either ENABLED (1) or DISABLED (0).

Arguments

None.

Examples

```
>>>
>>>set bflag 220
>>>
>>>set boot EZA0
>>>
>>>set controlp disabled
>>>
>>>set halt reboot
>>>
>>>set host /dup/dssi/bus:1  0
Starting DUP server...

DSSI Bus 1 Node 0 (SUSAN)
Copyright © 1990  Digital Equipment Corporation
DRVEXR V1.0  D  5-JUL-1990 15:33:06
DRVTST V1.0  D  5-JUL-1990 15:33:06
HISTRY V1.0  D  5-JUL-1990 15:33:06
ERASE  V1.0  D  5-JUL-1990 15:33:06
PARAMS V1.0  D  5-JUL-1990 15:33:06
DIRECT V1.0  D  5-JUL-1990 15:33:06
End of directory

Task Name? params
Copyright © 1990  Digital Equipment Corporation

PARAMS> stat path

   ID  Path Block     Remote Node      DGS_S      DGS_R      MSGS_S     MSGS_R
   -- ------------  ---------------  ---------- ---------- ---------- ------
----
    0 PB  FF811ECC   Internal Path           0          0          0          0
    6 PB  FF811FD0  KFQSA  KFX V1.0          0          0          0          0
    1 PB  FF8120D4  KAREN  RFX V101          0          0          0          0
    4 PB  FF8121D8  WILMA  RFX V101          0          0          0          0
    5 PB  FF8122DC  BETTY  RFX V101          0          0          0          0
    2 PB  FF8123E0  DSSI1  VMS V5.0          0          0      14328      14328
    3 PB  FF8124E4  3      VMB BOOT          0          0         61         61

PARAMS> exit
Exiting...

Task Name?

Stopping DUP server...
>>>
>>>set host /dup/dssi/bus:1 0 params
Starting DUP server...

DSSI Bus 1 Node 0 (SUSAN)
Copyright © 1990  Digital Equipment Corporation

PARAMS> show node
```

```
Parameter     Current           Default        Type     Radix
--------- ---------------  ---------------  --------  -----
NODENAME          SUSAN             RF30        String  Ascii   B

PARAMS> show allclass

Parameter     Current           Default        Type     Radix
--------- ---------------  ---------------  --------  -----
ALLCLASS                1                 0      Byte    Dec   B

PARAMS> exit
Exiting...

Stopping DUP server...
>>>
>>>set host /maint/uqssp 20001468
UQSSP Controller (772150)

Enter SET, CLEAR, SHOW, HELP, EXIT, or QUIT
Node   CSR Address   Model
 0       772150       21
 1       760334       21
 4       760340       21
 5       760344       21
 7     ------ KFQSA ------
? help
Commands:
    SET <node> /KFQSA                   set KFQSA DSSI node number
    SET <node> <CSR_address> <model>    enable a DSSI device
    CLEAR <node>                        disable a DSSI device
    SHOW                                show current configuration
    HELP                                print this text
    EXIT                                program the KFQSA
    QUIT                                don't program the KFQSA
Parameters:
    <node>                              0 to 7
    <CSR_address>                       760010 to 777774
    <model>                             21 (disk) or 22 (tape)
? set 6 /kfqsa
? show
Node   CSR Address   Model
 0       772150       21
 1       760334       21
 4       760340       21
 5       760344       21
 6     ------ KFQSA ------
? exit
Programming the KFQSA...
>>>
>>>set language 5
>>>
>>>set recall 1
>>>
```

SHOW

Format

SHOW {parameter}

Qualifiers

-
Depends on the specific parameter.

Parameters

BFL(A)G
Shows the default R5 boot flags.

BOOT
Shows the default boot device.

CONTROLP
Shows the current state of Control-P halt recognition, either ENABLED or DISABLED.

DEVICE
Shows a list of all devices in the system.

HALT
Shows the user-defined halt action. One of the following keywords are displayed: DEFAULT, RESTART, REBOOT, HALT, or RESTART_REBOOT. Refer to Table 12–1 for usage.

DSSI
Shows the status of all nodes that can be found on the DSSI busses. For each node on the DSSI bus, the console displays the node number, the node name, and the boot name and type of the device, if available. The command does not indicate the "bootability" of the device.

The node that issues the command reports a node name of "*".

The device information is obtained from the "media type" field of the MSCP command GET UNIT STATUS. In case the node is not running or is not capable of running an MSCP server, then no device information is displayed.

ETHERNET
Shows the hardware Ethernet address for all Ethernet adapters that can be found. Displays as blank if no Ethernet adapter is present.

LANGUAGE
Shows the console language and keyboard type. Refer to the corresponding SET LANGUAGE command for the meaning.

MEMORY
Shows main memory configuration on a board-by-board basis. Also reports the addresses of bad pages, as defined by the bitmap.

/FULL

Shows the normally inaccessible areas of memory, such as the PFN bitmap pages, the console scratch memory pages, and the Q22–bus scatter/gather map pages.

QBUS
Show all Q22–bus I/O addresses that respond to an aligned word read. For each address, the console displays the address in the VAX I/O space in hex, the address as it would appear in the Q22–bus I/O space in octal, and the word data that was read in hex.

This command may take several minutes to complete, so the user may want to issue a Control-C to terminate the command. The command disables the scatter /gather map for the duration of the command.

RECALL
Shows the current state of command recall, either ENABLED or DISABLED.

RLV12
Shows all RL01 and RL02 disks that appear on the Q22–bus.

SCSI
Shows any SCSI devices in the system.

TRANSLATION
Shows any virtual addresses that map to the specified physical address. The firmware uses the current values of page table base and length registers to perform its search (it is assumed that page tables have been properly built).

UQSSP
Shows the status of all disks and tapes that can be found on the Q22–bus that support the UQSSP protocol. For each such disk or tape on the Q22–bus, the console displays the controller number, the controller CSR address, and the boot name and type of each device connected to the controller. The command does not indicate the "bootability" of the device.

The device information is obtained from the "media type" field of the MSCP command GET UNIT STATUS. In case the node is not running or is not capable of running an MSCP server, then no device information is displayed.

VERSION
Show the current version of the firmware.

Qualifiers

-
On a per parameter basis.

Arguments

None.

Description

Displays the console parameter indicated.

Examples

```
>>>
>>>show bflag
00000220
>>>
>>>show boot
 EZA0
>>>
>>>show device
DSSI Bus 0 Node 7 (*)

DSSI Bus 1 Node 0 (SUSAN)
-DIA0 (RF30)

DSSI Bus 1 Node 1 (KAREN)
-DIA1 (RF30)

DSSI Bus 1 Node 4 (WILMA)
-DIA4 (RF30)

DSSI Bus 1 Node 5 (BETTY)
-DIA5 (RF30)

DSSI Bus 1 Node 6 (*)

DSSI Node 6 (KFQSA)

SCSI Adapter 0 (761300), SCSI ID 7
-DKA100 (DEC RZ31     (C) DEC)
-DKA300 (MAXTOR XT-8000S)

UQSSP Disk Controller 0 (772150)
-DUA0 (RF30)

UQSSP Disk Controller 1 (760334)
-DUB1 (RF30)

UQSSP Disk Controller 2 (760340)
-DUC3 (RF30)

UQSSP Disk Controller 3 (760344)
-DUD4 (RF30)

Ethernet Adapter
-EZA0 (08-00-2B-03-82-78)
>>>
DSSI Bus 0 Node 7 (*)

DSSI Bus 1 Node 0 (SUSAN)
-DIA0 (RF30)

DSSI Bus 1 Node 1 (KAREN)
-DIA1 (RF30)

DSSI Bus 1 Node 4 (WILMA)
-DIA4 (RF30)
```

```
DSSI Bus 1 Node 5 (BETTY)
-DIA5 (RF30)

DSSI Bus 1 Node 6 (*)

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
KA680 V4.0, VMB 2.13
>>>
```

START

Format

START [{address}]

Qualifiers

None.

Arguments

[{address}]
The address at which to begin execution. This is loaded in the user's PC.

Description

The console starts instruction execution at the specified address. If no address is given, the current PC is used. If memory mapping is enabled, macroinstructions are executed from virtual memory, and the address is treated as a virtual address. The START command is equivalent to a DEPOSIT to PC followed by a CONTINUE. No INITIALIZE is performed.

Examples

```
>>>start  1000
```

TEST

Format

TEST [{test_number} [{test_arguments}]]

Qualifiers

None.

Arguments

{test_number}
A 2-digit hexadecimal number specifying the test to be executed.

{test_arguments}
Up to five additional test arguments. These arguments are accepted but no meaning is attached to them by the console. For the interpretation of these arguments, consult the test specification for the individual test.

Description

The console invokes a diagnostic test program specified by the test number. If a test number of 0 is specified, the power-up script is executed. The console accepts an optional list of up to five additional hexadecimal arguments.

A more detailed explanation of the diagnostics may be found in Section 12.3.

Examples

```
>>>
>>>unjam         ! Use UNJAM and INIT to ensure a consistent state.
>>>init          ! Warning...this has the same effect as a powerup!
>>>              ! Execute the power-up diagnostic script.
>>>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..
>>>
>>>             ! Show the diagnostic state.
>>>
>>>t fe

 Bitmap=01FF2000, Length=00002000, Checksum=FFFF, Busmap=01FF8000
 Test_number=00, Subtest=00, Loop_Subtest=00, Error_type=00
 Error_vector=0000, Severity=02, Last_exception_PC=00000000
 Total_error_count=0005, Led_display=0C, Console_display=03, save_mchk_code=00
 parameter_1=00000000  2=00000000  3=00000000  4=00000000  5=00000000
 parameter_6=00000001  7=0001EA0F  8=0001EEE5  9=0001EC72 10=00000000
 previous_error=00C04200, FE014100, FE014100, FE014100
 Flags=FFFF C200 443E   BCache_Disable=06  KA680, 128KB BC,  14.0 ns
 Return_stack=201406A4, Subtest_pc=20056514, Timeout=00030D40
>>>
>>>             ! Display the CPU registers.
>>>
>>>t 9c
```

```
  SBR=01FB8000    SLR=00002021  SAVPC=20047ECC SAVPSL=20047ECC BCETSTS=00000000
 SCBB=20053E00   P0BR=80000000   P0LR=00100A80   P1BR=00000000 BCETIDX=00000000
 P1LR=00000000    SID=13001401   TODR=9614FD28   ICCS=00000000 BCEDSTS=00000700
  ECR=000000CA  MAPEN=00000000  BDMTR=20084000  BDMKR=0000007C BCEDIDX=00000008
 TCR0=00000000   TIR0=00000000  TNIR0=00000000  TIVR0=00000078 BCEDECC=00000000
 TCR1=00000001   TIR1=200775AD  TNIR1=0000000F  TIVR1=0000007C NEDATHI=00000000
 RXCS=00000000   RXDB=0000000D   TXCS=00000000   TXDB=00000030 NEDATLO=00000000
  SCR=0000D000   DSER=00000000  QBEAR=0000000F   DEAR=00000000    CESR=00000000
QBMBR=01FF8000    BDR=3CF808AB  DLEDR=0000000C  SSCCR=00D55570  CMCDSR=0000C108
CBTCR=00004000  IPCR0=0000     CSEAR1=00000000 CSEAR2=00000000 CIOEAR1=010FC000
PCSTS=FFFFF800  PCADR=FFFFFFF8  PCCTL=FFFFFC13   ICSR=00000001 CIOEAR2=000002C0
 CCTL=00000007 BCETAG=00000000                   VMAR=000007E0   CNEAR=00000000
NESTS=00000000 CEFSTS=00019200 NEOADR=E005BFC0 NEOCMD=8000FF04  NEICMD=00000000
DSSI_1=03  (BUS_1)     PQBBR_1=03060022    PMCSR_1=00000000    SSHMA_1=00008A20
     PSR_1=00000000     PESR_1=00000000     PFAR_1=00000000      PPR_1=00000000
DSSI_2=07  (BUS_0)     PQBBR_2=03060022    PMCSR_2=00000000    SSHMA_2=0000CA20
     PSR_2=00000000     PESR_2=00000000     PFAR_2=00000000      PPR_2=00000000
NICSR0=1FFF0003   3=00004030   4=00004050   5=8039FF00   6=83E0F000  7=00000000
NICSR9=04E204E2  10=00040000  11=00000000  12=00000000  13=00000000 15=0000FFFF
NISA=08-00-2B-16-01-
EA            MEAR=08406010_ADD=21018040      MESR=000FF000
     MEMCON_0:3; 0=80000003,  1=81000003,  2=00000007,  3=00000007   MMCDSR=01111000
     MEMCON_4:7; 4=00000007,  5=00000007,  6=00000007,  7=00000007    MOAMR=00000000
     >>>
     >>>            ! List all of the diagnostic tests.
     >>>
     >>>t 9e

     Test
     #   Address   Name               Parameters
     ____________________________________________________________________________
         20053E00  SCB
         20054E14  De_executive
     30  20063AD0  Memory_Init_Bitmap *** mark_Hard_SBEs ******
     31  20064264  Memory_Setup_CSRs  **********
     32  20064D60  NMC_registers      **********
     33  20064EFC  NMC_powerup        **
     34  2005B714  SSC_ROM            *
     35  20067AF8  B_Cache_diag_mode  bypass_test_mask *********
     37  2006849C  Cache_w_Memory     bypass_test_mask *********
     3F  200644EC  Mem_FDM_Addr_shorts *** cont_on_err ******
     40  200626B8  Memory_count_pages First_board Last_bd Soft_errs_allowed *******
     41  200564FC  Board_Reset        *
     42  2005A3B0  Chk_for_Interrupts *****
     46  2006782C  P_Cache_diag_mode  bypass_test_mask *********
     47  20063FF0  Memory_Refresh     start_a end incr cont_on_err time_seconds *****
     48  2006185C  Memory_Addr_shorts start_add end_add * cont_on_err pat2 pat3 ****
     49  200634D4  Memory_FDM         *** cont_on_err ******
     4A  200631E0  Memory_ECC_SBEs    start_add end_add add_incr cont_on_err ******
     4B  20061F8C  Memory_Byte_Errors start_add end_add add_incr cont_on_err ******
     4C  20062B70  Memory_ECC_Logic   start_add end_add add_incr cont_on_err ******
     4D  200616DC  Memory_Address     start_add end_add add_incr cont_on_err ******
     4E  20061D90  Memory_Byte        start_add end_add add_incr cont_on_err ******
     4F  200628BC  Memory_Data        start_add end_add add_incr cont_on_err ******
     51  2005A870  FPA                *******
     52  2005ABB0  SSC_Prog_timers    which_timer wait_time_us ***
     53  2005AE80  SSC_TOY_Clock      repeat_test_250ms_ea Tolerance ***
     54  2005A486  Virtual_Mode       *********
     55  2005B036  Interval_Timer     *****
     56  2005FF1C  SHAC_LPBCK         ********
     58  20060798  SHAC_RESET         dssi_bus port_number time_secs
     59  2005F064  SGEC_LPBCK_ASSIST  time_secs **
     5C  2005F5CC  SHAC                shac_number *******
     5F  2005E350  SGEC                loopback_type no_ram_tests ******
     60  2005DD4B  SSC_Console_SLU    start_BAUD end_BAUD ******
     63  2005B5B8  QDSS_any           input_csr selftest_r0 selftest_r1 ******
```

```
80  20065330  CQBIC_memory       bypass_test_mask *********
81  2005B21A  Qbus_MSCP          IP_csr ******
82  2005B3DF  Qbus_DELQA         device_num_addr ****
83  200577DE  QZA_Intlpbck1      controller_number ********
84  20058E98  QZA_Intlpbck2      controller_number *********
85  20056A24  QZA_memory         incr test_pattern controller_number *******
86  20056EE0  QZA_DMA            Controller_number main_mem_buf ********
87  2005A0DC  QZA_EXTLPBCK       controller_number ****
90  2005AB2E  CQBIC_registers    *
91  2005AAC4  CQBIC_powerup      **
99  200650F8  Flush_Ena_Caches   dis_flush_virtual dis_flush_backup dis_flush_primary
9A  2005D064  INTERACTION        pass_count disable_device ****
9B  20064F7C  Init_memory_16MB   *
9C  2005B7DE  List_CPU_registers *
9D  2005E11C  Utility            Expnd_err_msg get_mode init_LEDs clr_ps_cnt
9E  2005B1EC  List_diagnostics   *
9F  20060D30  Create_A0_Script   **********
C1  200566D0  SSC_RAM_Data       *
C2  200568A6  SSC_RAM_Data_Addr  *
C5  2005E23E  SSC_registers      *
C6  20056614  SSC_powerup        *********
```

```
Test
#   Address   Name               Parameters
_____________________________________________________________________________
D0  20067400  V_Cache_diag_mode  bypass_test_mask *********
D2  20065A5C  O_Bit_diag_mode    bypass_test_mask *********
DA  200682C4  PB_Flush_Cache     **********
DB  200661F0  Speed              print_speed *********
DC  20064490  NO_Memory_present  ********
DD  200669A0  B_Cache_Data_debug start_add end_add add_incr *******
DE  20066558  B_Cache_Tag_Debug  start_add end_add add_incr *******
DF  20065E30  O_BIT_DEBUG        start_add end_add add_incr seg_incr ******

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
>>>
```

UNJAM

Format

UNJAM

Qualifiers

None.

Arguments

None.

Description

An I/O bus reset is performed. This is implemented by writing 1 to IPR 55. Additionally, the SGEC and SHAC chips are explicitly software reset because PR$_IORESET has no effect on them.

Examples

```
>>>unjam
>>>
```

X - Binary Load and Unload

Format

X {address} {count} <CR> {line_checksum} {data} {data_checksum}

Qualifiers

None.

Arguments

None.

Description

The X command is for use by automatic systems communicating with the console. It is not intended for use by operators.

The console loads or unloads (that is, writes to memory, or reads from memory) the specified number of data bytes, starting at the specified address through the console serial line, regardless of which device is serving as the system console.

If bit 31 of the count is clear, data is to be received by the console, and deposited into memory. If bit 31 of the count is set, data is to be read from memory and sent by the console. The remaining bits in the count are a positive number indicating the number of bytes to load or unload.

The console accepts the command upon receiving the carriage return. The next byte the console receives is the command checksum, which is not echoed. The command checksum is verified by adding all command characters, including the checksum and separating whitespace (but not including the terminating carriage return or rubouts or characters deleted by rubout), into an 8-bit register initially set to zero. If no errors occur, the result is zero. If the command checksum is correct, the console responds with the input prompt and either sends data to the requester or prepares to receive data. If the command checksum is in error, the console responds with an error message. The intent is to prevent inadvertent operator entry into a mode where the console is accepting characters from the keyboard as data, with no escape mechanism possible.

If the command is a load (bit 31 of the count is clear), the console responds with the input prompt, then accepts the specified number of bytes of data for depositing to memory, and an additional byte of received data checksum. The data is verified by adding all data characters and the checksum character into an 8-bit register initially set to zero. If the final contents of the register is non-zero, the data or checksum are in error, and the console responds with an error message.

If the command is a binary unload (bit 31 of the count is set), the console responds with the input prompt, followed by the specified number of bytes of binary data. As each byte is sent, it is added to a checksum register initially set to zero. At the end of the transmission, the 2's complement of the low byte of the register is sent.

If the data checksum is incorrect on a load, or if memory errors or line errors occur during the transmission of data, the entire transmission is completed and 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 (Control-C, Control-S, Control-O, etc.) during a binary unload. It is not possible during a binary load because all received characters are valid binary data.

Data being loaded with a binary load command must be received by the console at a rate of at least one byte every 60 seconds. The command checksum that precedes the data must be received by the console within 60 seconds of the carriage return that terminates the command line. The data checksum must be received within 60 seconds of the last data byte. If any of these timing requirements are not met, the console aborts the transmission by issuing an error message and prompting for input.

The entire command, including the checksum, can be sent to the console as a single burst of characters at the console serial line's specified character rate. The console is able to receive at least 4 KB of data with a single X command.

XDELTA

Format

XDELTA

Qualifiers

/CONTINUE
Enter XDELTA in "step" mode.

Arguments

None.

Description

The KA680 XDELTA debugger is a subset of the VMS XDELTA debug utility. Although a command summary appears in Table 12–14, consult the *VMS DELTA/XDELTA Utility Manual* for a complete description of supported commands.

Table 12–14 XDELTA Command Summary

Command	Description
[m	Set Display Mode
[x][,y]/	Open Location and Display Contents in Prevailing Width Mode
[x][,y]**!**	Open Location and Display Contents in Instruction Mode
LINEFEED	Close Current Location, Open Next Location
ESCAPE	Close Location, Open and Display Previous Location
TAB	Close Location, Open and Display Indirect Location
[x][,y]**"**	Open Location and Display Contents in ASCII Mode
RETURN	Close Current Location
[z][,n][,d][,c]**;B**	Breakpoint
;P	Proceed from Breakpoint
[g]**;G**	Go
S	Step Instruction
O	Step Instruction Over Subroutine
'string**'**	Deposit ASCII String
c**;E**	Execute Command String
l,b**;X**	Load Base Register

(continued on next page)

Table 12–14 (Cont.) XDELTA Command Summary

Command	Description
expression**=**	Display Value of Expression using +, -, *, %, and @

The following list describes XDELTA command parameters represented in the table by lowercase letters:

m : Display mode, either B(byte), W(word), or L(longword)
x : Address of location to be displayed
y : Last address of a range (beginning with address x) to be displayed
z : Breakpoint address
n : Number of the breakpoint (2..8)
c : Address of an XDELTA command string to be executed (on breakpoint)
d : Address of location to be displayed on breakpoint
g : Address at which to begin execution
l : Address to deposit in base register
b : Number of base register (0..F)

Square brackets [ ] around a parameter indicate that it is optional.

Table 12–15 XDELTA Symbols

Symbol	Description
.	Current address, the address of the current location.
Q	The last value displayed.
Xn	Base registers n, 0 to F, used to reference data structures.
Rn	General-purpose register n, 0 to F, RF+4 is the PSL.
Pn	Internal processor register n.
G	System space address prefix; that is, G2E is equivalent to 8000002E.
H	Process control region address prefix; that is, H2E is equivalent to 7FFE002E.

When the console XDELTA command is executed, control is passed to the ROM based XDELTA utility. XDELTA diplays the current user PC address (if defined, else 20040000) and the instruction at that location and then awaits a command. In this mode of operation, XDELTA instruction stepping and program executions are disabled. However, it is possible to examine machine state and set breakpoints in a user program.

Using ;P in this context returns control to the console. The CONTINUE command may then be used run the user program. If an XDELTA breakpoint is encountered, the address and instruction are displayed and XDELTA awaits further commands. At this point normal XDELTA debugging may proceed, including single-stepping and running the user program.

Alternatively, XDELTA/CONTINUE can be used to trace trap into a user program at the address specified by the PC. Using this option enables single-stepping program execution, and the complete services of the firmware XDELTA utility.

Users should keep in mind that the XDELTA facility utilizes both the trace trap and breakpoint vectors of the active SCB. If XDELTA is to be used to debug a program that establishes its own SCB, the trace trap (SCB+28) and breakpoint (SCB+2C) vectors should be copied from the firmware SCB to the user's SCB.

Examples

```
>>>ex/p/ins/n:9 1200
  P 00001200   9E MOVAB   L^0000121D,R0
  P 00001207   DB MFPR    S^#22,R1
  P 0000120A   EF EXTZV   S^#07,S^#01,R1,R1
  P 0000120F   13 BEQL    00001207
  P 00001211   9A MOVZBL  (R0)+,R1
  P 00001214   13 BEQL    0000121B
  P 00001216   DA MTPR    R1,S^#23
  P 00001219   11 BRB     00001207
  P 0000121B   01 NOP
  P 0000121C   00 HALT
>>>ex pc
  G 0000000F 00001200
>>>xdelta
Stepping is disabled...

00001200/MOVAB   L^0000121D,R0   S
EH?
.;B;P
>>>continue

1 BRK AT 00001200
00001200/MOVAB   L^0000121D,R0   S
00001207/MFPR    S^#22,R1   S
0000120A/EXTZV   S^#07,S^#01,R1,R1   ;P
XLOAD succeeded loading XTEST.EXE!

?06 HLT INST
        PC = 0000121D
>>>dep pc 1200
>>>xdelta/continue

00001200/MOVAB   L^0000121D,R0   S
00001207/MFPR    S^#22,R1   S
0000120A/EXTZV   S^#07,S^#01,R1,R1   ;P
XLOAD succeeded loading XTEST.EXE!

?06 HLT INST
        PC = 0000121D
>>>
```

! - Comment

Format

!

Qualifiers

None.

Arguments

None.

Description

The comment command character is used to include optional text, which you can use to identify a command line or add descriptions. It can appear anywhere on the command line. All characters following the comment character are ignored.

Examples

```
>>>! The console ignores this line.
>>>
```

Table 12–16 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}	—
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	—	—

(continued on next page)

Table 12–16 (Cont.) Console Command Summary

Command	Qualifiers	Argument	Other(s)
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 12–17 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}	Specifies number of additional operations.
/STEP:{size}	Overrides the default step incrementing size with the value specified for the current reference.
/WRONG	On writes, uses the value of 3, which always generates double bit errors. Ignores 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, etc.)
/M	Machine state (PSL)

(continued on next page)

Table 12–17 (Cont.) Console Qualifier Summary
Command-Specific

/INSTRUCTION	EXAMINE command only. Disassembles the instruction at address specified.
/NOT	SEARCH command only. Inverts the sense of the match.
/R5:{boot_flags}, /{boot_flags}	BOOT command only. Specifies a function bitmap to pass to VMB through R5. Refer to Figure 12–9 for a bit description of R5. Either form of the command is acceptable.
/RPB, /MEM	FIND command only. Searches for valid RPB or good block of memory.
/DUP, /DSSI, /UQSSP, /DISK, /TAPE, /MAINTENANCE, /SERVICE	SET HOST command only. Refers to command description for usage.
/CONTINUE	XDELTA command only. Enters XDELTA step mode at current PC.

Nomenclature for Table 12–16 and Table 12–17 :

UPPERCASE denotes the command or qualifier keyword.
{} denotes a mandatory item that must be syntactically correct.
[ ] denotes an optional item.
! denotes an "or" condition.

And

boot_flags, count, size, address, and parameters denote hex longword values.
boot_device denotes a legal boot device name.
csr_address denotes a Q22–bus I/O page CSR address.
controller_number denotes a controller number from 0 to 255.
halt_action denotes the value of the user-defined halt action from 0 to 4.
language_type denotes the language value, from 1 to 15.
command denotes a console command other than REPEAT.
data, pattern, and mask denote hex values of the current size.
test_number denotes hex byte test number.

12.3 Diagnostics

The ROM-based diagnostics constitute the bulk of the firmware on the KA680. These diagnostics run automatically on powerup and can be executed interactively as a whole, or as individual tests using the TEST command. (See Section Section 12.2.9, Console Commands.) This section summarizes the operation of the ROM-based diagnostics.

The purpose of the ROM-based diagnostics is multifaceted:

1. During powerup, they determine if enough of the KA680 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 able to report all detected errors.
4. They allow sophisticated users and field service technicians to run individual diagnostics interactively, with the intent of isolating errors to the FRU (field replaceable unit).

To accommodate these requirements, the diagnostics are designed as a collection of individual parameterized tests. A data structure, called a *script*, and a program, called the *diagnostic executive*, orchestrate the running of these tests in the right order with the right parameters.

A script is a data structure that points to various tests. There are several scripts, one for the field and several for manufacturing, depending where on the manufacturing line the board is. Sophisticated users may also create their own scripts interactively. Additionally, the script contains other information:

- What parameters need to be passed to the test.
- What is to be displayed, if anything, on the console.
- What is to be displayed, if anything, on the LED.
- What to do on errors (halt, loop, or continue).
- Where the tests may be run from. For example, there are certain tests that can only be run from the FEPROM. Other tests are PIC (position independent code), and may be run from FEPROM 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 KA680 that are used for manufacturing, power-up, and Digital Services personnel. The diagnostic executive automatically invokes the correct script based on the current environment of the KA680. Any script can be explicitly run with the TEST command from the console terminal.

The diagnostic executive is also responsible for controlling the tests so that when errors occur, they can be caught and reported to the user. The executive also ensures that when the tests are run, the machine is left in a consistent and well-defined state.

12.3.1 Error Reporting

Before a console is established, the only error reporting is via the KA680 diagnostic LEDs (and any LEDs on other boards). Once a console has been established, all errors detected by the diagnostics are also reported by the console. When possible, the diagnostics issue an error summary on the console.

For example, Figure 12–14 shows a typical error display.

Figure 12–14 Diagnostic Register Dump

```
    ?9A 2 02 FF 0000 0000 01    ; SUBTEST_9A_02, DE_INTERACTION.LIS     (1)

    P1=00000002  P2=00000000  P3=00004000  P4=00008000  P5=0000C000     (2)
    P6=00000000  P7=00000002  P8=00000002  P9=84004000 P10=00001FFF     (3)
    r0=00000054  r1=00000040  r2=00000000  r3=0000C524  r4=00000014     (4)
    r5=30002800  r6=0000C4E0  r7=20008000  r8=00004000 EPC=20057BBD     (5)

    Normal operation not possible.


?42 2 C0 FF 00D4 0000 00       ; SUBTEST_42_C0, DE_Chk_for_Interrupts.LIS

P1=00000003  P2=00FC00C0  P3=000000D4  P4=00000000  P5=00000000
P6=21020020  P7=00000000  P8=00000000  P9=FFFFFFFF P10=00000002
r0=000000C0  r1=0000002E  r2=00000042  r3=20140778  r4=20059FA8
r5=20059FCB  r6=200680C7  r7=00000000  r8=00000008 EPC=2005A047
   SCBB=20053A00     TODR=0446A76E     ECR=000000CA  LIS_ADD=009F
    SCR=0000D000     DSER=00000000   QBEAR=0000000A     DEAR=00000000
  QBMBR=01FF8000      BDR=38FB08AB   SSCCR=00D55570    IPCR0=0020
 NCAERR=00000000  NCAMODE=0000C108  CP1SEA=00000000   CP2SEA=00000000
CP1IOEA=010FC000  CP2IOEA=1015C400  NDALEA=00000000    MAPEN=00000000
  PCSTS=FFFFF800    PCADR=FFFFFFF8   PCCTL=FFFFFE13
   ICSR=00000001     VMAR=000007E0    VTAG=20041200    VDATA=FFC13101
   CCTL=00000007  BCETSTS=00000000 BCETIDX=00000000   BCETAG=00000000
BCEDSTS=00000700  BCEDIDX=00000008  CEFSTS=00019200  BCEDECC=00000000
 CEFADR=F015C400    NESTS=00000000  NEOADR=E005C7E8   NEOCMD=8000FF04
 NEICMD=00000000  NEDATHI=00000000 NEDATLO=00000000 OBITMODE=00000340
NMCMODE=09115C20    NMCEA=08406010_____ADD=21020040   NMCERR=000FF000
MEMCON_0:7;  0=80000003,  1=81000003
2=00000007,  3=00000007,  4=00000007,  5=00000007,  6=00000007,  7=00000007
```

In Figure 12–14, the numbers in parentheses on the right side refer to lines of the display and are not a part of the diagnostic dump. The information on these lines is summarized below.

1. Test summary containing six hexadecimal fields.
   a. ?9A, **test** identifies the diagnostic test.
   b. 2, **severity** is the level of a test failure, as dictated by the script. A severity level 2 error causes the display of this 5-line error printout, and halts an autoboot to console I/O mode. A severity level 1 error displays the first line of the error printout, but does not interrupt an autoboot. Most tests have a severity level of 2.
   c. 02, **subtestlog** is a number, that in conjunction with listing files, isolates to within a few instructions where the diagnostic detected the error.
   d. FF, **de_error** is a code with which the diagnostic executive signals the diagnostic's state and any illegal behavior. This field indicates a condition that the diagnostic expects on detecting a failure. The possible codes are:

      FF - Normal error exit from diagnostic
      FE - Unanticipated interrupt

FD - Interrupt in cleanup routine
FC - Interrupt in interrupt handler
FB - Script requirements not met
FA - No such diagnostic
EF - Unanticipated exception in executive

e. 0000, **vector** 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).

f. 0000, **count** is the number of previous errors that have occurred.

g. 01, **loop_subtest** is an additional subtestlog generated out of the context of the current test as specified by the current test number and subtestlog. Usually these logs occur in common subroutines called from a diagnostic test.

h. SUBTEST_9A_02, **subtest_symbol** is a unique symbol that identifies the most recent subtestlog entry in the listing file.

i. DE_INTERACTION.LIS, **listing_file** is the name of the listing file that contains the failed diagnostic.

2. P1...P5 are the first five parameters containing diagnostic state.
3. P6...P10 are the last five parameters containing diagnostic state.
4. R0...R4 are the first five GPRs at the moment the error was detected.
5. R5...R8 are additional GPRs and EPC is PC at the time of the error.

The use of parameters and registers varies with each test. The appropriate listing file should be consulted for interpretation of these parameters and registers in determining diagnostic state.

12.3.2 Diagnostic Interdependencies

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

12.3.3 Areas Not Covered

The goal has been to achieve the highest possible coverage on the KA680 and the memory boards. However, the testing of the KA680 while running with memory management turned on is minimal. Also, due to the way the firmware is implemented (a polled environment running at IPL 31), the testing of interrupts is not extensive.

These diagnostics are not intended to be used as system level tests. There are no tests to completely verify that access to the Q22–bus will work. Thus, a disk, a controller, the backplane, or portions of the CQBIC may be faulty, and the diagnostics may not detect the fault. Such a fault may result later as an inability to boot.

12.4 Environment

The following is a description of the intended environment in which KA680 firmware will be used. This environment includes not only the surrounding hardware, but also the field of use.

12.4.1 Users

Engineering, Manufacturing, Digital Services, and customers will use this program to test, configure, and boot their KA680 modules. This firmware will be used to provide both an easy means to bootstrap a KA680 based system and to detect and isolate system malfunctions.

Of these users, all but customers are assumed to be computer "sophisticates." While this will often be true of customers as well, no such assumption is made. Target customers include both sophisticated users who can use and understand all the features provided by the firmware, as well as unsophisticated users who know little about computers.

Users are not assumed to speak English. The console program can be configured to output critical console messages in either English, French, German, Spanish, Italian, Norwegian, Dutch, Swedish, Finnish, Danish, or Portuguese. When the module powers up without a specified language, KA680 prompts the user for the language to be used to display critical system messages. The selected language is recorded in battery backed-up RAM in order to retain the preferred language when the system is powered down.

12.4.2 Hardware

The firmware described in this document resides on the KA680 module. The KA680 is an NVAX based Q22–bus CPU module with off-board expansion memory. Additionally, the KA680 is specifically designed to consolidate other "system" components on a single module. In particular, the KA680 has an on-board Ethernet adapter for network communications and integral DSSI ports for connection of mass storage devices. The KA680 provides a local serial I/O port for support of a standard VAX console.

The KA680 continues to support communications with other Q22–bus modules through its Q22–bus interface consisting of a scatter/gather map, a direct map of the Q22–bus I/O page and memory through VAX I/O space, and interprocessor communication registers. Because the KA680 processor is intended to be the Q22–bus arbiter, the use of the KA680 as an auxiliary processor is unsupported.

The KA680 provides a maximum of 512KB of FEPROM for the firmware. The firmware resides on the KA680 module in four 128KB FEPROM, located at the VAX restart location in I/O space at physical address E0040000. Unlike its predecessors, the KA680 firmware image appears only once in I/O space and the entire image runs halt-protected.

Note

All public call-in routines in the ROMs also run halt-protected and exit through SSC RAM so that halt protection is turned off when returning to the caller. Unsupported call-ins to ROM utilities may result in halt protection forced outside the FEPROM extent.

For the firmware to operate, the processor must be functioning to the point of executing instructions from the firmware FEPROM. This assumes that the NVAX core set of chips is operating correctly.

The firmware uses the KA680 module LEDs and a console terminal to display diagnostic progress, display error conditions, and indicate the current mode of operation. Additionally, the console terminal is used as the primary operator interface when in console I/O mode.

Note

A console terminal is not required for operation because the module can be configured to bootstrap automatically. However, in most scenarios, a console terminal is a recommended part of a standard configuration.

12.4.3 Software

The KA680 firmware runs standalone, and does not require other software products for normal operation in "console I/O mode." However, in most situations, an operating system (or other image) is loaded in KA680 local memory and control is transferred to it. When the operating system is running, the processor is in "program I/O mode." (The terms *console I/O mode* and *program I/O mode* refer to the context and usage of the console terminal.)

The KA680 will support the Digital standard operating systems: VMS and VAXELN. Additionally, the firmware will support bootstrap of MDM and other diagnostics images. Furthermore, the console will support communication with an APT host in manufacturing environments via the console serial line.

12.4.4 Services

The KA680 firmware is an integral part of the module and does not require installation or support services. However, if an ECO is required, it may be applied in the field by the customer.

12.5 Internationalization

Most firmware message text is either multilingual or language-independent. The messages displayed on powerup and system failures are multilingual. Depending on user preference, these messages are output in either English, French, German, Italian, Portuguese, Spanish, Dutch, Danish, Finnish, Norwegian, or Swedish. All other messages are language-independent; they are constructed of short, language-insensitive abbreviations rather than readable sentences.

Numeric status displays on the processor LEDs facilitate the diagnosis of failing processors in a language-independent manner.

The operation of the console is independent of the user's line voltage or frequency.

The console supports the Digital Multinational Character Set (MCS). This support extends to: displaying foreign language messages with MCS, accepting and echoing MCS characters, and accepting a device attributes report (console queries the terminal to determine if it is a CRT or not) using the C1 control characters of MCS. However, all console commands must be entered using the ANSI subset.

If the terminal does not support MCS, the console uses English message texts.

The console program uses four characters that are National Replacement Characters (NRCs): the caret "^", the backslash "\", and the left and right square brackets "[", "]". The caret is used by the console to denote control characters. The backslash is used to delimit text deletions when editing console input. The square brackets are used to denote directory specifications when the user directs the bootstrap to solicit a secondary bootstrap file name. No provision is made for terminals that replace any of these characters on output; however, use of angle brackets "<" and ">" for directory specifications on input is acceptable.

A

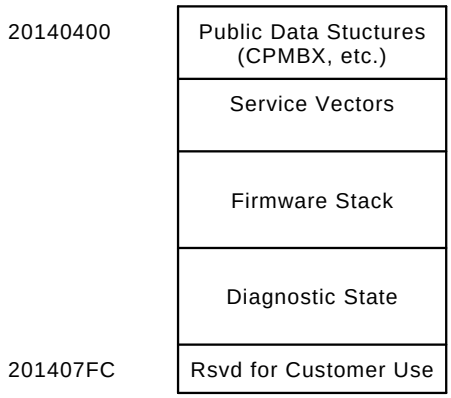
NVRAM Partitioning

This appendix describes how the KA680 firmware partitions the SSC 1 KB battery backed-up (BBU) RAM.

A.1 SSC RAM Layout

The KA680 firmware uses the 1 KB of NVRAM on the SSC for storage of firmware-specific data structures and other information that must be preserved across power cycles. This NVRAM resides in the SSC chip starting at address 20140400. The NVRAM should not be used by the operating systems except as documented below. This NVRAM is not reflected in the bitmap built by the firmware.

Figure A–1 KA680 SSC NVRAM Layout



A.1.1 Public Data Structures

The following is a list of the public data structures in NVRAM used by the console.

Fields that are desginated as reserved and/or internal use should not be written because there is no protection against such corruption.

A.1.2 Console Program MailBoX (CPMBX)

The Console Program MailBoX (CPMBX) is a software data structure located at the beginning of NVRAM (20140400). The CPMBX is used to pass information between the KA680 firmware and diagnostics, VMB, or an operating system. It consists of three bytes (referred to here as NVR0, NVR1, and NVR2).

Figure A–2 NVR0 (20140400) : Console Program MailBoX (CPMBX)

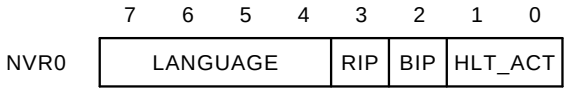


Table A–1 CPMBX NVR0

Field	Name	Description
7:4	LANGUAGE	This field specifies the current selected language for displaying halt and error messages on terminals that support MCS.
3	RIP	If set, a restart attempt is in progress. This flag must be cleared by the operating system if the restart succeeds.
2	BIP	If set, a bootstrap attempt is in progress. This flag must be cleared by the operating system if the bootstrap succeeds.
1:0	HLT_ACT	Processor halt action - This field, in conjunction with the conditions specified in Table 12–1, is used to control the automatic restart/bootstrap procedure. HLT_ACT is normally written by the operating system. 0 : Restart; if that fails, reboot; if that fails, halt. 1 : Restart; if that fails, halt. 2 : Reboot; if that fails, halt. 3 : Halt.

Figure A–3 NVR1 (20140401)

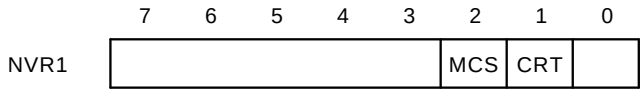


Table A–2 CPMBX NVR1

Field	Name	Description
2	MCS	If set, indicates that the attached terminal supports 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 hard copy.

Figure A–4 NVR2 (20140402)

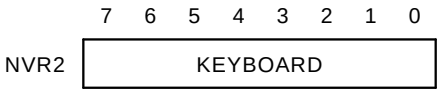


Table A–3 CPMBX NVR0

Field	Name	Description
7:0	KEYBOARD	This field indicates the national keyboard variant in use.

A.1.3 Firmware Stack

This area contains the stack that is used by all the firmware (except VMB, which has its own built-in stack).

A.1.4 Diagnostic State

This area is used by the firmware-resident diagnostics. It is not documented here.

A.1.5 USER Area

The KA680 console reserves the last longword (address 201407FC) of the NVRAM for customer use. This location is not tested by the console firmware. Its value is undefined.

B

Data Structures

This appendix contains definitions of key global data structures that are used by the KA680 firmware.

B.1 Halt Dispatch State Machine

The KA680 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 (Table B–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 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 B–1 Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbox Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
		Perform conditional initialization.[1]				
ENTRY	–>RESET INIT	xxx	01	xx	xxx	x - x - x - x - x - x - x
ENTRY	–>BREAK INIT	011	00	xx	xxx	x - 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 - x
		Perform common initialization. [2]				
RESET INIT	–>INIT	xxx	xx	xx	xxx	x - x - x - x - x - x - x
BREAK INIT	–>INIT	xxx	xx	xx	xxx	x - x - x - x - x - x - x
TRACE INIT	–>INIT	xxx	xx	xx	xxx	x - x - x - x - x - x - x
OTHER INIT	–>INIT	xxx	xx	xx	xxx	x - x - x - x - x - x - x
		Check for external halts. [3]				
INIT	–>BOOTSTRAP	010	00	xx	xxx	0 - x - x - x - x - x - x
INIT	–>BOOTSTRAP	101	00	xx	xxx	x - x - x - x - x - x - x
INIT	–>HALT	xxx	00	xx	xxx	x - 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
TRACE	–>HALT	xxx	xx	xx	xxx	x - x - x - x - x - x - x
		Check for bootstrap conditions. [5]				
INIT	–>BOOTSTRAP	xxx	01	xx	xxx	0 - 0 - 0 - 0 - 0 - 0

[1] Perform a unique initialization routine on entry. In particular, powerups, 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.

[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, mailbox is 0, and user action is 2.
If not powerup, restart failed, mailbox is 0, and user action is 0 or 4.

(continued on next page)

Table B–1 (Cont.) Firmware State Transition Table

Current State	Next State	Halt Type	Halt Code	Mailbox Action	User Action	HEN-ERR-TIP-DIP-BIP-RIP
INIT	–>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
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

[6] Restart the operating system if not powerup and:

If mailbox is 1.
If mailbox is 0 and user action is 1 or 4.
If mailbox is 0, user action is 0, and halts are disabled.

[7] Exit after halts, bootstrap, or restart. The exit state transitions to program I/O mode.

[8] Guard block that catches all exception conditions. In all cases, just halt.

"x" is used in this table to indicate a "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 B–1.

B.2 RPB

VMB typically utilizes the low portion of memory unless there are bad pages in the first 128 KB. The first page in its block is used for the RPB (restart parameter block) through which it communicates to the operating system. Usually, this is page 0.

VMB will initialize the restart parameter block (RPB) as follows:

Table B–2 Restart Parameter Block Fields

(R11)+	Field Name	Description
00:	RPB$L_BASE	Physical address of base of RPB.
04:	RPB$L_ RESTART	Cleared.
08:	RPB$L_ CHKSUM	-1
0C:	RPB$L_ RSTRTFLG	Cleared.
10:	RPB$L_HALTPC	R10 on entry to VMB (HALT PC).
10:	RPB$L_ HALTPSL	PR$_SAVPSL on entry to VMB (HALT PSL).
18:	RPB$L_ HALTCODE	AP on entry to VMB (HALT CODE).
1C:	RPB$L_BOOTR0	R0 on entry to VMB.

Note

The field RPB$W_R0UBVEC, which overlaps the high order word of RPB$L_BOOTR0, is set by the boot device drivers to the SCB offset (in the second page of the SCB) of the interrupt vector for the boot device.

20:	RPB$L_BOOTR1	VMB version number. The high-order word of the version is the major ID and the low-order word is the minor ID.
24:	RPB$L_BOOTR2	R2 on entry to VMB.
28:	RPB$L_BOOTR3	R3 on entry to VMB.

(continued on next page)

Table B–2 (Cont.) Restart Parameter Block Fields

(R11)+	Field Name	Description
2C:	RPB$L_BOOTR4	R4 on entry to VMB. **Note** The 48-bit booting node address is stored in RPB$L_BOOTR3 and RPB$L_BOOTR4 for compatibility with VAXELN V1.1. (This field is initialized this way only when performing a network boot.)
30:	RPB$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$Q_PFNMAP	The PFN bitmap is a array of bits where each bit has the value "1" if the corresponding page of memory is valid, or the value "0" if the corresponding page of memory contains a memory error. Through use of the PFNMAP, the operating system can avoid memory errors by avoiding known bad pages altogether. The memory bitmap is always page-aligned, and describes all the pages of memory from physical page #0 to the high-end of memory, but excluding the PFN bitmap itself and the Q–bus map registers. If the high byte of the bitmap spans some pages available to the operating system and some pages of the PFN bitmap itself, the pages corresponding to the bitmap itself will be marked as bad pages. The first longword of the PFNMAP descriptor contains the number of bytes in the PFNMAP; the second longword contains the physical address of the bitmap.
4C:	RPB$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.
66:	RPB$B_DEVTYP	Device type code of boot device.

(continued on next page)

Table B–2 (Cont.) Restart Parameter Block Fields

(R11)+	Field Name	Description
67:	RPB$B_SLAVE	Slave number of boot device.
68:	RPB$T_FILE	Name of secondary bootstrap image (defaults to [SYS0.SYSEXE]SYSBOOT.EXE). This field (up to 40 bytes) is overwritten with the input string on a "solicit" boot.
		Note 1 : For VAX VMS, the RPB$T_ FILE must contain the root directory string "SYSn." on a non-network bootstrap. This string is parsed by SYSBOOT (that is, SYSBOOT does not use the high nibble of BOOTR5). 2 : The RPB$T_FILE is overwritten to contain the boot node name for compatibility with VAXELN V1.1. (This field is initialized this way only 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 eight high bits of this longword contain the TR number, which is always zero for KA680.
C0:	RPB$L_ MEMDSC+4	PFN of the first page of memory. This field is always zero for KA680, even if page #0 is a bad page.
		Note No other memory descriptors are used.
104:	RPB$L_BADPGS	Count of "bad" pages of physical memory.

(continued on next page)

Table B–2 (Cont.) Restart Parameter Block Fields

(R11)+	Field Name	Description
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.

B.3 VMB Argument List

The VMB code will also initialize an argument list as follows (the address of the argument list is passed in the AP):

Table B–3 VMB Argument List

(AP)+	Field Name	Description
04:	VMB$L_ FILECACHE	Quadword file name.
0C:	VMB$L_LO_PFN	PFN of first page of physical memory (always zero, regardless of where 128 KB 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$B_ SYSTEMID	48-bit (actually, a quadword is allocated) booting node address initialized when performing a network boot. This field is copied from the target system address parameter of the parameters message. (The DECnet HIORD value is added if the field were two bytes.)
2C:	VMB$L_FLAGS	Set as needed.
30:	VMB$L_CI_ HIPFN	Cluster interface high PFN.
34:	VMB$Q_ NODENAME	Boot node name that is initialized when performing a network boot. This field is copied from the target system name parameter of the parameters message.
3C:	VMB$Q_ HOSTADDR	Host node address (this value is initialized only when booting over the network). This field is copied from the host system address parameter of the parameters message.
44:	VMB$Q_ HOSTNAME	Host node name (this value is initialized only when performing a network boot). This field is copied from the host system name parameter of the parameters message.
4C:	VMB$Q_TOD	Time of day (this value is initialized only when performing a network boot). The time of day is copied from the first eight bytes of the host system time parameter of the parameters message. (The time differential values are NOT copied.)
54:	VMB$L_XPARAM	Pointer to data retrieved from request of the parameter file.
58:	—	The rest of the argument list is zeroed.

C
Error Messages

The error messages issued by the KA680 firmware fall into three categories: halt code messages, VMB error messages, and console messages.

C.1 Machine Check Register Dump

Some error conditions, such as machine check, generate an error summary register dump preceding the error message. For example, examining a nonexistent memory location results in the following display:

```
>>>ex /p/l 7fffff0                ! Examine non-existent memory.

MESR=801FF000    MEAR=11FFFFF9     MMCDSR=01111000   MOAMR=00000000
CESR=00000000    CMCDSR=0000C108   CSEAR1=00000000   CSEAR2=00000000
CIOEAR1=010FC000 CIOEAR2=000002C0  CNEAR=00000000    ICSR=00000001
PCSTS=FFFFF800   PCADR=FFFFFFF8    TBSTS=C00000E0    TBADR=00000000
NESTS=00000000   NEOADR=E014066C   NEOCMD=8000F005   NEICMD=00000000
NEDATHI=00000000 NEDATLO=00000000  CEFSTS=0000022A   CEFADR=07FFFFF0
BCETSTS=00000000 BCETIDX=00000000  BCETAG=00000000   BCEDSTS=00000700
BCEDIDX=00000008 BCEDECC=00000000  CBTCR=00004000    DSER=00000000
QBEAR=0000000F   DEAR=00000000     IPCR0=0000        ECR=000000CA
?7D MACHINE CHECK  80060000 00000000 20047ECC 20047EBD 20047EB9 B0110080

>>>
```

C.2 Halt Code Messages

Except on powerup, which is not treated as an error condition, the following halt messages are issued by the firmware whenever the processor halts.

For example, if the processor encounters a HALT instruction while in kernel mode, the processor halts and the firmware displays the following before entering console I/O mode:

```
?06 HLT INST
PC = 800050D3
```

The number preceding the halt message is the "halt code." This number is obtained from SAVPSL<13:8>(RESTART_CODE), IPR 43, which is saved on any processor restart operation.

Table C–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 the reason for this halt.
?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 invalid translation occurred during processing of a kernel stack not valid exception.
?12	DBL ERR2	Double machine check error. A machine check occurred while trying to service a machine check.
?13	DBL ERR3	Double machine check error. A machine check occurred while trying to service a kernel stack not valid exception.
?19	PSL EXC5[1]	PSL<26:24> = 5 on interrupt or exception.
?1A	PSL EXC6[1]	PSL<26:24> = 6 on interrupt or exception.
?1B	PSL EXC7[1]	PSL<26:24> = 7 on interrupt or exception.
?1D	PSL REI5[1]	PSL<26:24> = 5 on an REI instruction.
?1E	PSL REI6[1]	PSL<26:24> = 6 on an REI instruction.
?1F	PSL REI7[1]	PSL<26:24> = 7 on an REI instruction.
?3F	MICROVERIFY FAILURE	Microcode power-up self-test failed.

[1]For the last six cases, the VAX architecture does not allow execution on the interrupt stack while in a mode other than kernel. In the first three cases, an interrupt is attempting to run on the interrupt stack while not in kernel mode. In the last three cases, an REI instruction is attempting to return to a mode other than kernel and still run on the interrupt stack.

C.3 VMB Error Messages

The following errors are issued by VMB.

Table C–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 off line.
?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	The Q22–bus map initialization failed.
?54	RETRY	No devices bootable, retrying.
?55	IVDEVNAM	Invalid device name.
?56	DRVERR	Drive error.

C.4 Console Error Messages

These error messages are issued in response to a console command that has error(s).

Table C–3 Console Error Messages

Code	Message	Description
?61	CORRUPTION	The console program database has been corrupted.
?62	ILLEGAL REFERENCE	Illegal reference. Either 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 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, and is not abbreviated to "Illegal command."
?6C	HALTED	The operator entered a HALT command.
?6D	FIND ERROR	A FIND command failed either to find the RPB or 128 KB of good memory.
?6E	TIME OUT	During an X command, data failed to arrive in the time expected (60 seconds).
?6F	MEMORY ERROR	A machine check occurred with a code indicating a read or write memory error.
?70	UNIMPLEMENTED	Unimplemented function.
?71	NO VALUE QUALIFIER	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.
?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.

(continued on next page)

Table C–3 (Cont.) Console Error Messages

Code	Message	Description
?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.

D

Machine State on Powerup

This appendix describes the state of the KA680 after a power-up halt.

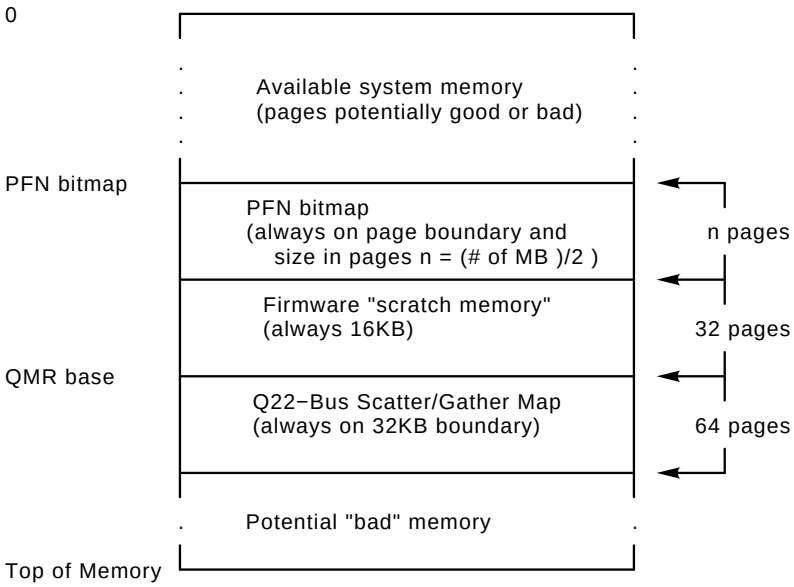
The descriptions in this section assume a machine with no errors, the machine has just been turned on, and only the power-up diagnostics have been run. The state of the machine is not defined if individual diagnostics are run or during any other halts other than a power-up halt (SAVPSL<13:8>(RESTART_CODE) = 3).

The following sections describe data structures that are guaranteed to be constant over future versions of the KA680 firmware. Placement and/or existence of any other structure(s) is not implied.

D.1 Main Memory Layout and State

Main memory is tested and initialized by the firmware on powerup. Figure D–1 is a diagram of how main memory is partitioned after diagnostics.

Figure D–1 Memory Layout after Power-up Diagnostics



D.1.1 Reserved Main Memory

In order to build the scatter/gather map and the bitmap, the firmware attempts to find a physically contiguous page-aligned 176 KB 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 KB, the upper 32 KB are dedicated to the Q22–bus scatter/gather map, as shown in Figure F-1. Of the lower portion, up to 128 KB at the bottom of the block is allocated to the PFN (page frame number) bitmap. The size of the PFN bitmap is dependent on the extent of physical memory. Each bit in the bitmap maps one page (512 bytes) of memory. The remainder of the block between the bitmap and scatter/gather map (a minimum of 16 KB) is allocated for the firmware.

D.1.1.1 PFN Bitmap

The PFN 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 an outcome of the memory tests on powerup. The bitmap always starts on a page boundary. The bitmap requires 1 KB for every 4 MB of main memory; hence, an 8 MB system requires 2 KB, 16 MB requires 4 KB, 32 MB requires 8 KB, and a 64 MB requires 16 KB. 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 correspondence between a page frame number (origin 0) and a bit index in the bitmap. A one in the bitmap indicates that the page is "good" and can be used. A zero indicates that the page is "bad" and should not be used. By default, a page is flagged "bad" if a multiple bit error occurs when referencing the page. Single bit errors, regardless of frequency, will not cause a page to be flagged "bad."

The PFN bitmap is protected by a checksum stored in the NVRAM. 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 facilitate diagnosis by service personnel.

D.1.1.2 Scatter/Gather Map

On powerup, the scatter/gather map is initialized by the firmware to map to the first 4 MB of of main memory. Main memory pages will not be mapped if there is a corresponding page in Q22–bus memory, or if the page is marked bad by the PFN bitmap.

On a processor halt other than powerup, the contents of the scatter/gather map is undefined, and is dependent on operating system usage.

Operating systems should not move the location of the scatter/gather map, and should access the map only on aligned longwords through the local I/O space of 20088000 to 2008FFFC, inclusive. The Q22–bus map base register, (QMBR) is set up by the firmware to point to this area, and should not be changed by software.

D.1.1.3 Firmware "Scratch Memory"

This section of memory is reserved for the firmware. However, it is used only after successful execution of the memory diagnostics and initialization of the PFN bitmap and scatter/gather map. This memory is primarily used for diagnostic purposes.

D.1.2 Contents of Main Memory

The contents of main memory are undefined after the diagnostics have run. Typically, nonzero test patterns will be left in memory.

The diagnostics will "scrub" all main memory so that no power-up induced errors remain in the memory system. On the KA680 memory subsystem, the state of the ECC bits and the data bits are undefined on initial powerup. This can result in single and multiple bit errors if the locations are read before written, because the ECC bits are not in agreement with their corresponding data bits. An aligned longword write to every location (done by diagnostics) eliminates all power-up induced errors.

D.2 Memory Controller Registers

The KA680 firmware assigns bank numbers to the MEMCONn registers in ascending order, without attempting to disable physical banks that contain errors. High order unused banks are set to zero. Error loggers should capture the following bits from each MEMCONn register:

MEMCONn <31> (bank enable bit). Since the firmware always assigns banks in ascending order, knowing which banks are enabled is sufficient information to derive the bank numbers. MEMCONn <1:0> (bank usage). This field determines the size of the banks on the particular memory board.

Additional information should be captured from the NMCDSR, MOAMR, MSER, and MEAR as needed.

D.2.1 On-chip Cache

The CPU on-chip cache is tested during the power-up diagnostics, flushed, and then turned on. The cache is also turned on by the BOOT and the INIT command.

D.2.2 Translation Buffer

The CPU translation buffer is tested by diagnostics on powerup, but is not used by the firmware because it runs in physical mode. The translation buffer can be invalidated by using PR$_TBIA, IPR 57.

D.2.3 Halt-Protected Space

On the KA680, halt-protected space spans the 512 KB FEPROM from E0040000 to E007FFFF. The firmware always runs in halt-protected space. When passing control to the bootstrap, the firmware exits the halt-protected space. Therefore, if halts are enabled, and the halt line is asserted, the processor will halt before booting.

E

MOP Support

This appendix describes the maintenance operation protocol (MOP) support features in the KA680 firmware.

E.1 Network "Listening"

While the KA680 is waiting for a load volunteer during bootstrap, it "listens" on the network for other maintenance messages directed to the node and periodically identifies itself at the end of each 8- to 12-minute interval prior to a bootstrap retry. In particular, this "listener" supplements the MOP functions of the VMB load requester typically found in bootstrap firmware and supports:

- A remote console server that generates COUNTERS messages in response to REQ_COUNTERS messages, unsolicited SYSTEM_ID messages every 8 to 12 minutes, and solicited SYSTEM_ID messages in response to REQUEST_ID messages as well as recognition of BOOT messages.
- A loopback server that responds to Ethernet LOOPBACK messages by echoing the message to the requester.
- An IEEE 802.2 responder that replies to both XID and TEST messages.

During network bootstrap operation, the KA680 complies with the requirements defined in the "NI Node Architecture Specification" for a primitive node. The firmware listens only to MOP "Load/Dump," MOP "Remote Console," Ethernet "Loopback Assistance," and IEEE 802.3 XID/TEST messages (listed in Table E–4) directed to the Ethernet physical address of the node. All other Ethernet protocols are filtered by the network device driver.

The MOP functions and message types that are supported by the KA680 are summarized in the following tables.

Table E–1 KA680 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 & ACK	MEM_LOAD
			or	MEM_ LOAD_ w_XFER
			or	PARAM_ LOAD_w_ XFER
	Server	——		——
Console	Requester	——		——
	Server	COUNTERS	in response to	REQ_ COUNTERS
		SYSTEM_ID[3]	in response to	REQUEST_ ID
				BOOT
Loopback	Requester	——		——
	Server	LOOPED_DATA[4]	in response to	LOOP_DATA
		IEEE 802.2 Messages[5]		
Exchange ID	Requester	——		——
	Server	XID_RSP	in response to	XID_CMD
Test	Requester	——		——
	Server	TEST_RSP	in response to	TEST_CMD

[1] All unsolicited messages are sent in Ethernet (MOP V3) and IEEE 802.2 (MOP V4) until the MOP version of the server is known. All solicited messages are sent in the format used for the request.

[2]The initial REQ_PROGRAM message is sent to the dumpload multicast address. If an assistance VOLUNTEER message is received, then the responder's address is used as the destination to repeat the REQ_PROGRAM message and for all subsequent REQ_MEM_LOAD messages.

[3]SYSTEM_ID messages are sent every 8 to 12 minutes to the remote console multicast address. On receipt of a REQUEST_ID message, they are sent to the initiator.

[4]LOOPED_DATA messages are sent in response to LOOP_DATA messages. These messages are actually in Ethernet LOOP TEST format, not in MOP format. 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 E–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 25 LQA 49 SGEC	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						

[1]MOP V3.0 only.

[2]MOP x4.0 only.

[3]Software ID field is loaded from the string stored in the 40-byte field, RPB$T_FILE, of the RPB on a solicited boot.

(continued on next page)

Table E–2 (Cont.) Supported MOP Messages

REMOTE CONSOLE							
REQUEST_ID	Code 05	Rsrvd xx	Recpt # nn-nn				
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 25 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)
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 field, RPB$T_FILE, of the RPB on a solicited boot.

[4]A BOOT message is not verified, because in this context, a boot is already in progress. However, a received BOOT message will cause the boot backoff timer to be reset to it's minimum value.

Table E–3 Ethernet and IEEE 802.3 Packet Headers

Ethernet MOP Message Format (MOP V3)

Dest_address	Src_address	Prot	Len	MOP msg	Pad	CRC
dd-dd-dd-dd-dd-dd	ss-ss-ss-ss-ss-ss	60-01	nn-nn	dd-...	xx-...	cc-cc
		60-02	nn-nn	dd-...		
		90-00	dd-...			

IEEE 802.3 SNAP SAP MOP Message Format (MOP V4)

Dest_address	Src_address	Len	DSAP	SSAP	Ctl	P_ID	MOP_msg	CRC
dd-dd-dd-dd-dd-dd	ss-ss-ss-ss-ss-ss	nn-nn	AA	AA	03	08-00-2B-60-01 08-00-2B-60-02 08-00-2B-90-00	dd-...	cc-cc

IEEE 802.3 XID/TEST Message Format (MOP V4)

Dest_address	Src_address	Len	DSAP	SSAP	Ctl [1]	Data	CRC
dd-dd-dd-dd-dd-dd	ss-ss-ss-ss-ss-ss	nn-nn	aa	bb	cc	ff-tt-ss (XID) Optional data (TEST)	cc-cc

[1]XID and TEST messages are identified in the IEEE 802.2 control field with binary 101x1111 and 111x0011, respectively. "x" denotes the Poll/Final bit that gets echoed in the response.

Table E–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.

E.2 MOP Counters

The following counters are kept for the Ethernet boot channel. All counters are unsigned integers. V4 counters roll over on overflow. All V3 counters "latch" at their maximum value to indicate overflow. Unless otherwise stated, all counters include both normal and multicast traffic. Furthermore, they include information for all protocol types. Frames received and bytes received counters do not include frames received with errors. Table E–5 displays the byte lengths and ordering of all the counters in both MOP Versions 3.0 and 4.0.

Table E–5 MOP Counter Block

	V3		V4		
Name	**Off**	**Len**	**Off**	**Len**	**Description**
TIME_SINCE_ CREATION	00	2	00	16	**Time since last zeroed.** The time that 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 V3, this time is the number of seconds. MOP V4 uses the UTC binary relative time format.
Rx_BYTES	02	4	10	8	**Bytes received.** The total number of user data bytes successfully received. This does not include Ethernet data link headers. This number is the number of bytes in the Ethernet data field, which includes any padding or length fields when they are enabled. These are bytes from frames that passed hardware filtering. When the number of frames received is used to calculate protocol overhead, the overhead plus bytes received provides a measurement of the amount of Ethernet bandwidth (over time) consumed by frames addressed to the local system.
Tx_BYTES	06	4	18	8	**Bytes sent.** 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.

(continued on next page)

Table E–5 (Cont.) MOP Counter Block

Name	V3 Off	V3 Len	V4 Off	V4 Len	Description
Rx_FRAMES	0A	4	20	8	**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.
Tx_FRAMES	0E	4	28	8	**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.
Rx_MCAST_BYTES	12	4	30	8	**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.
Rx_MCAST_FRAMES	16	4	38	8	**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.
Tx_INIT_DEFFERED	1A	4	40	8	**Frames sent, [1] 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.
Tx_ONE_COLLISION	1E	4	48	8	**Frames sent [1], 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.

[1]Only one of these three counters will be incremented for a given frame.

(continued on next page)

Table E–5 (Cont.) MOP Counter Block

Name	V3 Off	V3 Len	V4 Off	V4 Len	Description
Tx_MULTI_COLLISION	22	4	50	8	**Frames sent[1], 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. NO SINGLE FRAME IS COUNTED IN MORE THAN ONE OF THE ABOVE THREE COUNTERS.
TxFAIL_COUNT	26	2	-	-	**Send failure count.** [2] The total number of times a transmit attempt failed. Each time the counter is incremented, a type of failure is recorded. When read-counter function reads the counter, the list of failures is also read. When the counter is set to zero, the list of failures is cleared. In conjunction with total frames sent, provides a measure of significant transmit problems. TxFAIL_BITMAP contains the possible reasons.
TxFAIL_BITMAP	2C	2	-	-	**Send failure reason bitmap.** [2] This bitmap lists the types of transmit failures that occurred as summarized below. 0 - Excessive collisions 1 - Carrier detect failed 2 - Short circuit 3 - Open circuit 4 - Frame too long 5 - Remote failure to defer
TxFAIL_EXCESS_COLLS	-	-	58	8	**Send failure - Excessive collisions.** Exceeded the maximum number of retransmissions due to collisions. Indicates an overload condition on the Ethernet.
TxFAIL_CARIER_CHECK	-	-	60	8	**Send failure - Carrier check failed.** The data link did not sense the receive signal that is required to accompany the transmission of a frame. 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.

[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.

(continued on next page)

Table E–5 (Cont.) MOP Counter Block

Name	V3 Off	V3 Len	V4 Off	V4 Len	Description
TxFAIL_SHRT_CIRCUIT	-	-	68	8	**Send failure - Short circuit.** [3] There is a short somewhere in the local area network coaxial cable, or the transceiver or controller/transceiver cable has failed. This indicates a problem either in local hardware or global network. The two can be distinguished by checking to see if other systems are reporting the same problem.
TxFAIL_OPEN_CIRCUIT	-	-	70	8	**Send failure - Open circuit.** [3] There is a break somewhere in the local area network coaxial cable. This indicates a problem either in local hardware or global network. The two can be distinguished by checking to see if other systems are reporting the same problem.
TxFAIL_LONG_FRAME	-	-	78	8	**Send failure - Frame too long.** [3] The controller or transceiver cut off transmission at the maximum size. This indicates a problem with the local system. Either it tried to send a frame that was too long or the hardware cutoff transmission too soon.
TxFAIL_REMOTE_ DEFER	-	-	80	8	**Send failure - Remote failure to defer.** [3] A remote system began transmitting after the allowed window for collisions. This indicates either a problem with some other system's carrier sense or a weak transmitter.
RxFAIL_COUNT	2A	2	-	-	**Receive failure count.** [2] The total number of frames received with some data error. Includes only data frames that passed either physical or multicast address comparison. This counter includes failure reasons in the same way as the send failure counter. In conjunction with total frames received, provides a measure of data-related receive problems. RxFAIL_BITMAP contains the possible reasons.
RxFAIL_BITMAP	2C	2	-	-	**Receive failure reason bitmap.** [2] This bitmap lists the types of receive failures that occurred as summarized below. 0 - Block check failure 1 - Framing error 2 - Frame too long

[2]V3 send/receive failures are collapsed into one counter with bitmap indicating which failures occurred.

[3]Always zero.

(continued on next page)

Table E–5 (Cont.) MOP Counter Block

Name	V3 Off	V3 Len	V4 Off	V4 Len	Description
RxFAIL_BLOCK_CHECK	-	-	88	8	**Receive failure - Block check error.** A frame failed the CRC check. This indicates several possible failures, such as EMI, late collisions, or improperly set hardware parameters.
RxFAIL_FRAMING_ERR	-	-	90	8	**Receive failure - Framing error.** The frame did not contain an integral number of 8-bit bytes. This indicates several possible failures, such as EMI, late collisions, or improperly set hardware parameters.
RxFAIL_LONG_FRAME	-	-	98	8	**Receive failure - Frame too long.**[3] The frame was discarded because it was outside the Ethernet maximum length and could not be received. This indicates that a remote system is sending invalid length frames.
UNKNOWN_ DESTINATION	2E	2	A0	8	**Unrecognized frame destination.** 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	30	2	A8	8	**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.
NO_SYSTEM_BUFFER	32	2	B0	8	**System buffer unavailable**[3] 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.

[3]Always zero.

(continued on next page)

Table E–5 (Cont.) MOP Counter Block

Name	V3 Off	V3 Len	V4 Off	V4 Len	Description
NO_USER_BUFFER	34	2	B8	8	**User buffer unavailable.** [3] 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.
FAIL_COLLIS_DETECT	-	-	C0	8	**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.

[3]Always zero.

F

Q22–bus Specification

This appendix describes the specifications for the Q22–bus.

F.1 Introduction

The Q22–bus, also known as the extended LSI–11 bus, is the low-end member of Digital's bus family.

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 F–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 KA680 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.

F.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.

F.2 Q22–bus Signal Assignments

Table F–1 lists the data and address signal assignments. Table F–2 lists the control signal assignments. Table F–3 lists the power and ground signal assignments. Table F–4 lists the spare signal assignments.

Table F–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

(continued on next page)

Table F–1 (Cont.) Data and Address Signal Assignments

Data and Address Signal	Pin Assignment
BDAL12	BS2
BDAL13	BT2
BDAL14	BU2
BDAL15	BV2
BDAL16	AC1
BDAL17	AD1
BDAL18	BC1
BDAL19	BD1
BDAL20	BE1
BDAL21	BF1

Table F–2 Control Signal Assignments

Control Signal	Pin Assignment
Data Control	
BDOUT	AE2
BRPLY	AF2
BDIN	AH2
BSYNC	AJ2
BWTBT	AK2
BBS7	AP2
Interrupt Control	
BIRQ7	BP1
BIRQ6	AB1
BIRQ5	AA1
BIRQ4	AL2
BIAKO	AN2
BIAKI	AM2
DMA Control	
BDMR	AN1
BSACK	BN1
BDMGO	AS2
BDMGI	AR2

(continued on next page)

Table F–2 (Cont.) Control Signal Assignments

Control Signal	Pin Assignment
System Control	
BHALT	AP1
BREF	AR1
BEVNT	BR1
BINIT	AT2
BDCOK	BA1
BPOK	BB1

Table F–3 Power and Ground Signal Assignments

Power and Ground	Pin Assignment
+5 B (battery) or +12 B (battery)	AS1
+12 B	BS1
+5 B	AV1
+5	AA2
+5	BA2
+5	BV1
+12	AD2
+12	BD2
+12	AB2
–12	AB2
–12	BB2
GND	AC2
GND	AJ1
GND	AM1
GND	AT1
GND	BC2
GND	BJ1
GND	BM1
GND	BT1

Table F–4 Spare Signal Assignments

Spare	Pin Assignment
SSpare1	AE1
SSpare3	AH1

(continued on next page)

Table F–4 (Cont.) Spare Signal Assignments

Spare	Pin Assignment
SSpare8	BH1
SSpare2	AF1
MSpareA	AK1
MSpareB	AL1
MSpareB	BK1
MSpareB	BL1
PSpare1	AU1
ASpare2	BU1

F.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 F–5 lists the data transfer bus cycles.

Table F–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 F–6 are used in the data transfer operations described in Table F–5.

Table F–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.

F.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.

F.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 set-up 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.

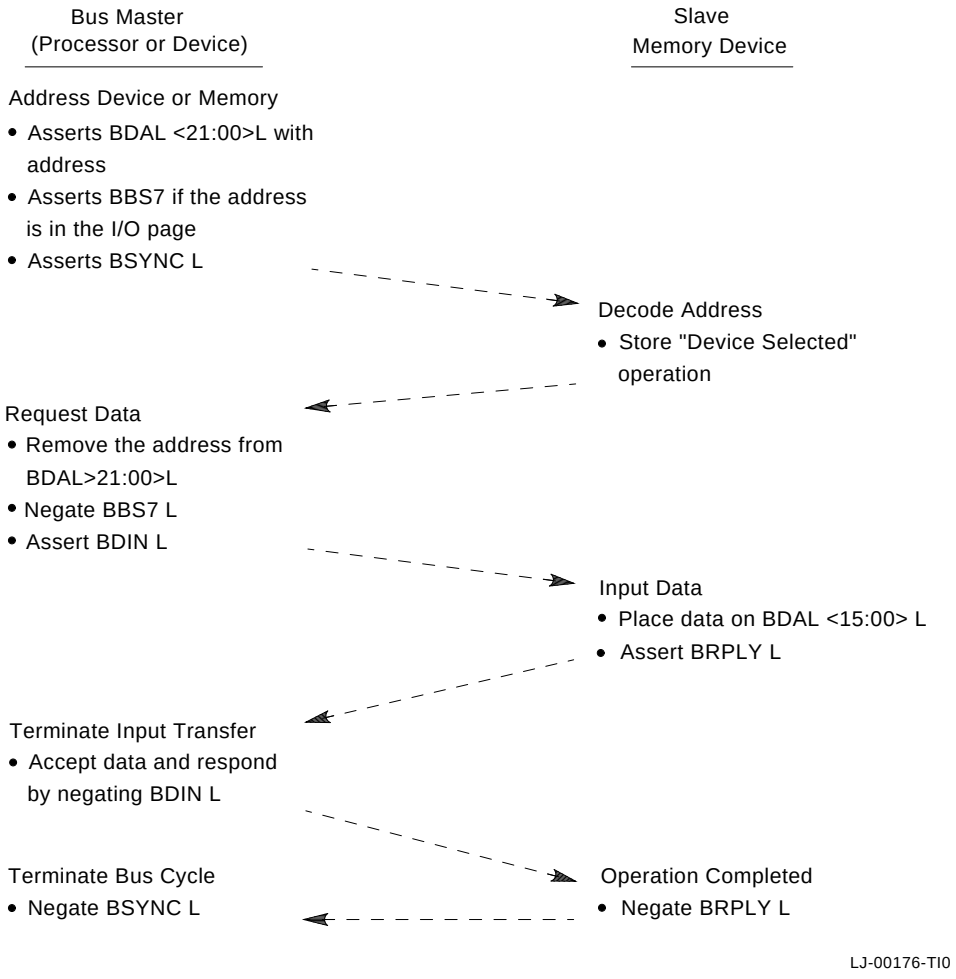
When asserted, BBS7 L indicates that the device address resides in the I/O page (the upper 4K address space). Memory devices generally do not respond to addresses in the I/O page; however, some system applications may permit memory to reside in the I/O page for use as DMA buffers, read-only memory bootstraps, and diagnostics.

DATI

The DATI bus cycle (Figure F–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

Figure F–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:

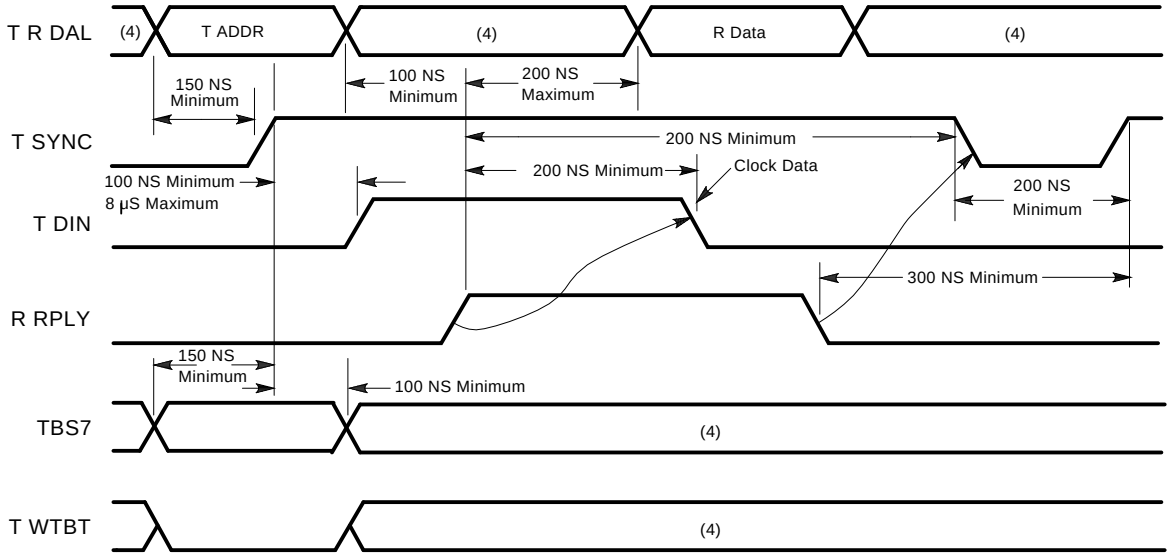
- BSYNC L must remain negated for 200 ns (minimum).
- BSYNC L must not become asserted within 300 ns of previous BRPLY L negation.

Figure F–2 shows DATI bus cycle timing.

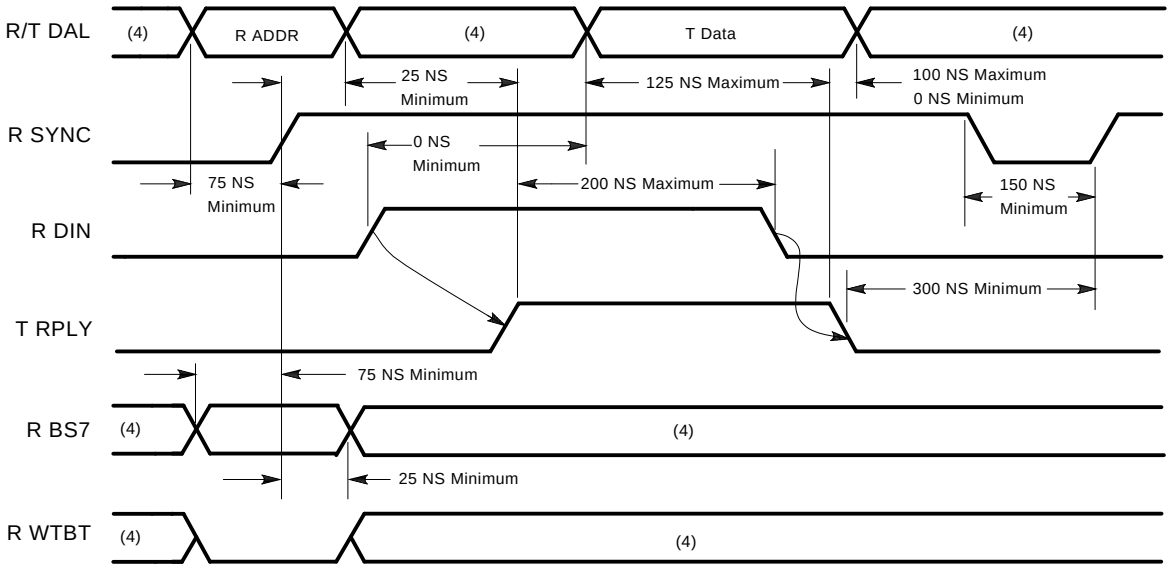
Note

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.

Figure F–2 DATI Bus Cycle Timing



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

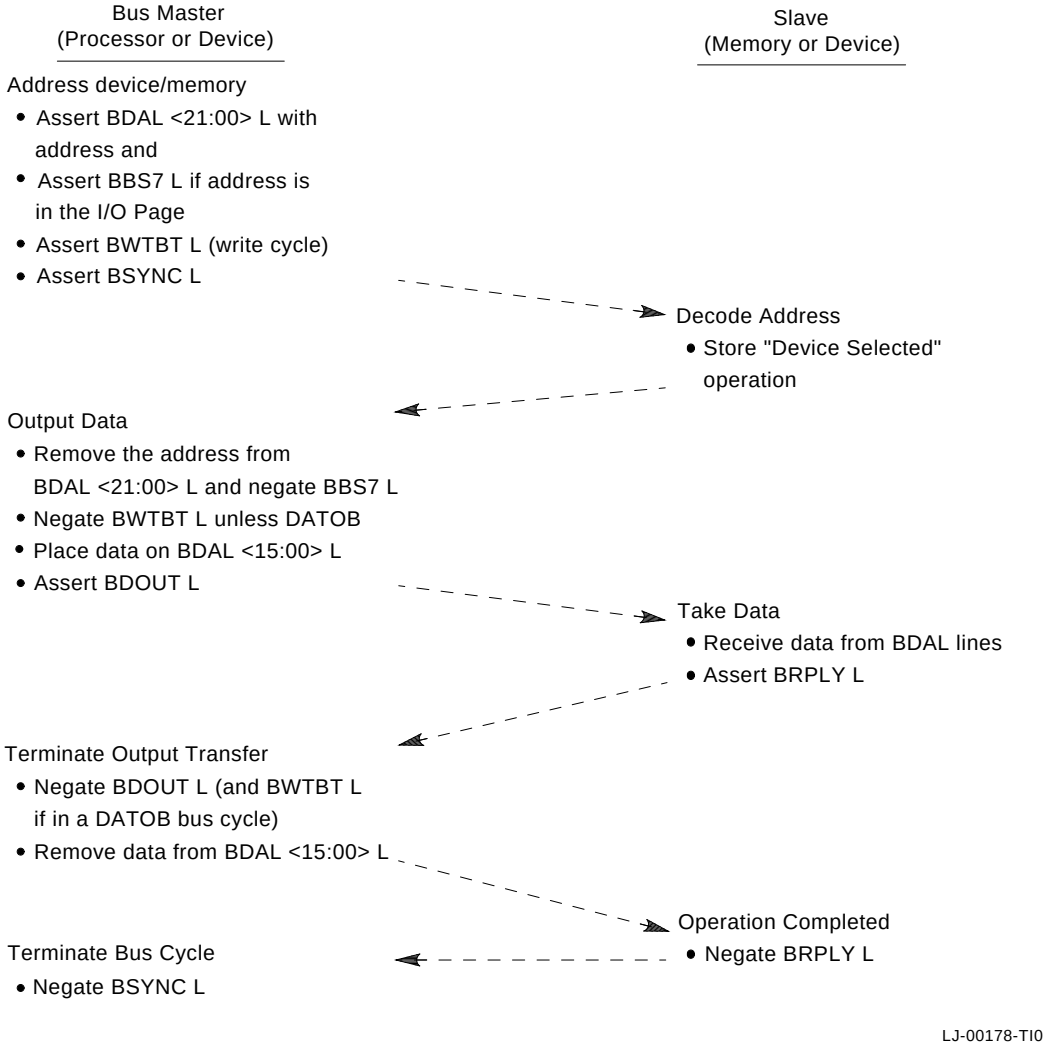
1. Timing shown at master and slave device bus driver inputs and bus receiver outputs.
2. Signal name prefixes are defined below:
   T=Bus Driver Input
   R=Bus Receiver Output
3. Bus driver output and bus receiver input signal names include a "B" prefix.
4. Don’t care condition.

LJ-00177-TI0

DATOB

DATOB (Figure F–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 F–3 DATO or DATOB Bus Cycle



The data transfer part of a DATOB bus cycle comprises a data set-up and deskew time, and a data hold and deskew time.

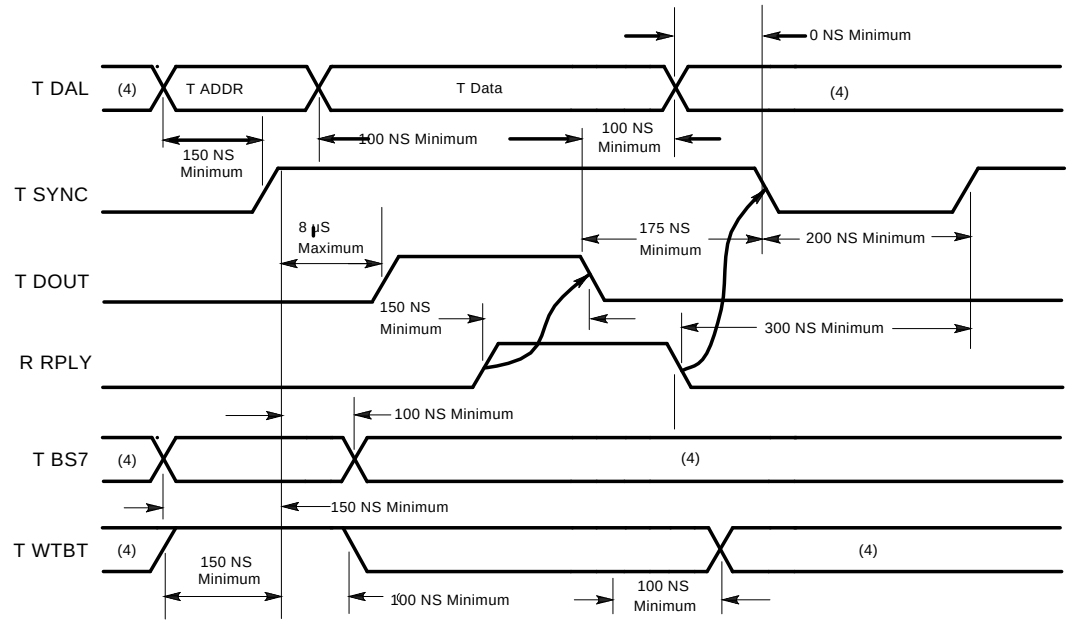
During the data set-up and deskew time, the bus master outputs the data on BDAL<15:00> L at least 100 ns after BSYNC L assertion. BWTBT L remains negated for the length of the bus cycle. If the transfer is a byte transfer, BWTBT L remains asserted. If it is the output of a DATIOB, BWTBT L becomes asserted 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 set-up and deskew time.

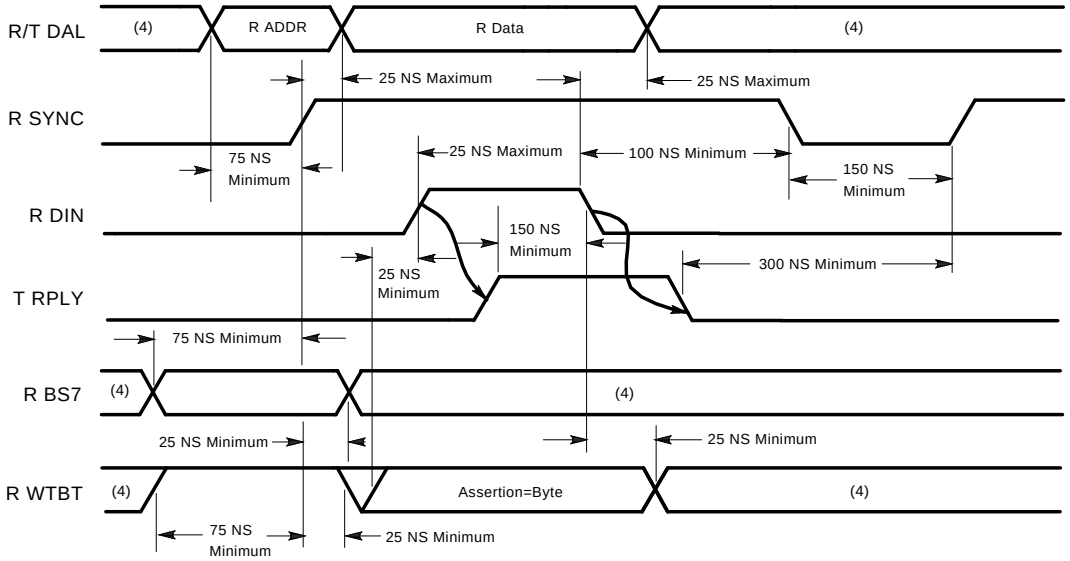
During the data hold and deskew time, the bus master receives BRPLY L and negates BDOUT L, which must remain asserted for at least 150 ns from the receipt of BRPLY L before being negated by the bus master. BDAL<17:00> L bus drivers remain asserted for at least 100 ns after BDOUT L negation. The bus master 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 F–4 shows DATOB bus cycle timing.

Figure F–4 DATO or DATOB Bus Cycle Timing



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

1. Timing shown at requesting device bus driver inputs and bus receiver outputs.
2. Signal name prefixes are defined below
   T=Bus Driver Input
   R=Bus Receiver Output
3. Bus driver output and bus receiver input signal names include a "B" prefix.
4. Don't care condition.

LJ-00179-TI0

DATIOB

The protocol for a DATIOB bus cycle (Figure F–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 F–6 shows the DATIOB bus cycle timing.

Figure F–5 DATIO or DATIOB Bus Cycle

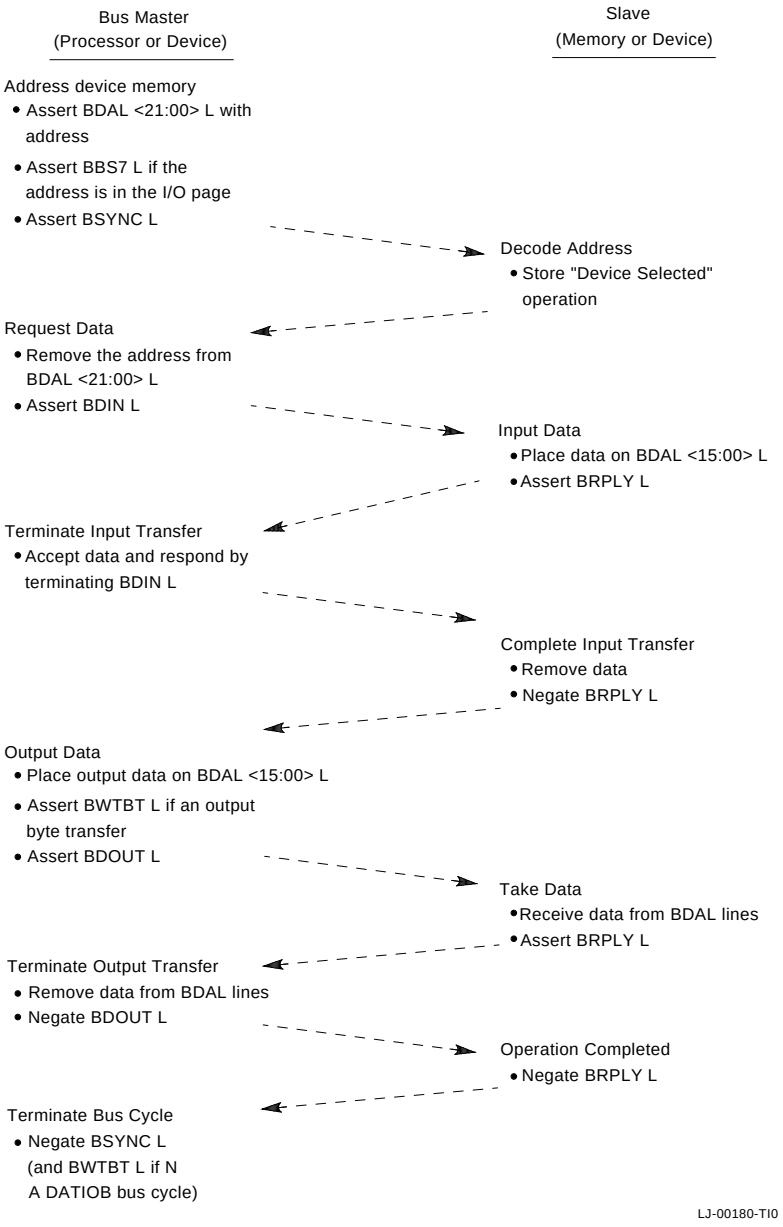
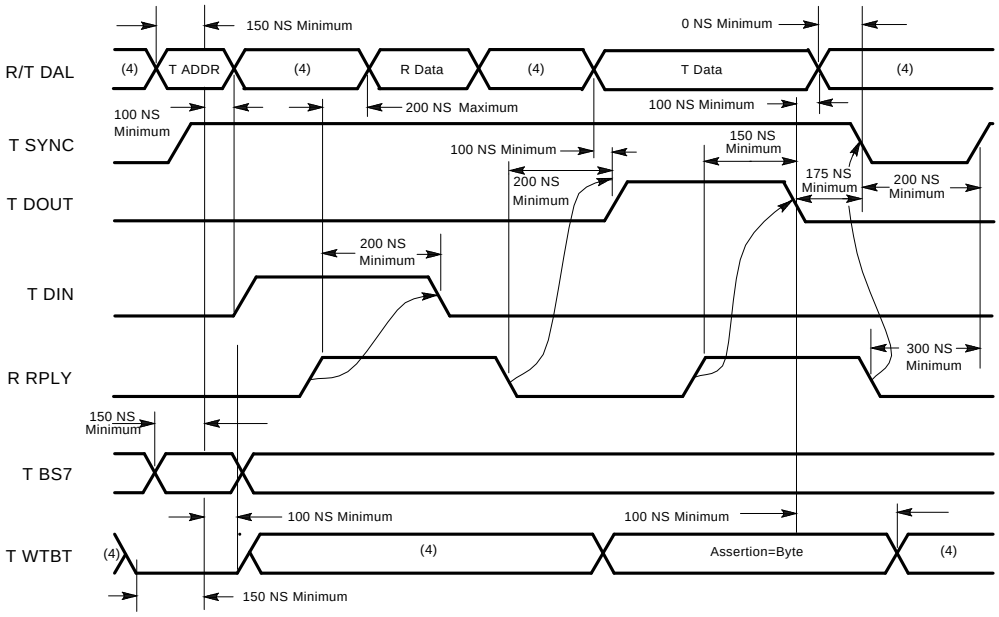
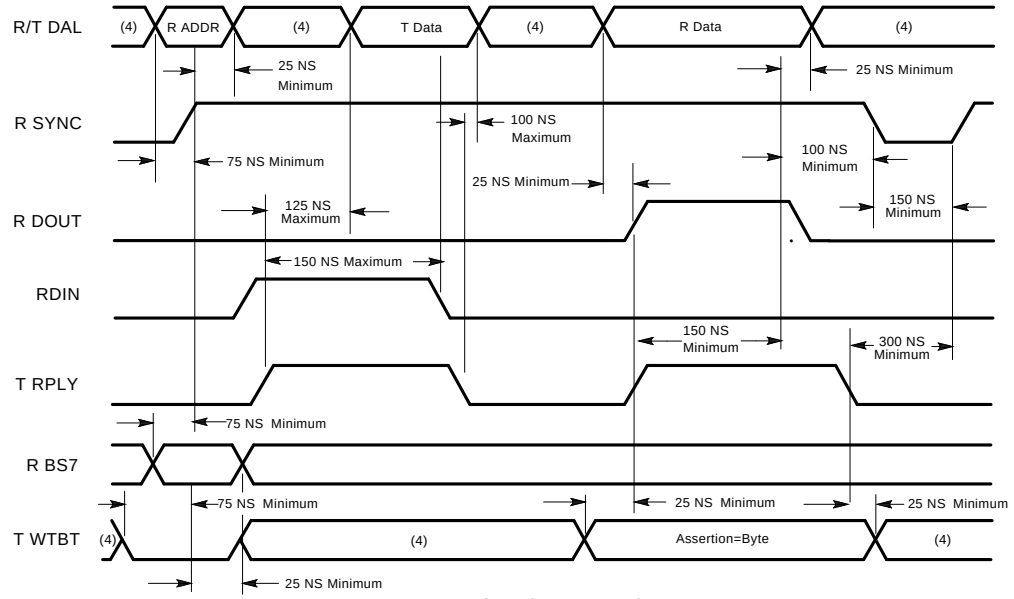


Figure F–6 DATIO or DATIOB Bus Cycle Timing



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

1. Timing shown at requesting device bus driver inputs and bus receiver outputs.
2. Signal name prefixes are defined below:
   T=Bus Driver Input
   R=Bus Receiver Output
3. Bus driver output and bus receiver input signal names include a "B" prefix.
4. Don't care condition.

LJ-00309-TI0

F.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

F.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 F–7 shows the DMA protocol, and Figure F–8 shows DMA request/grant timing.

Figure F–7 DMA Protocol

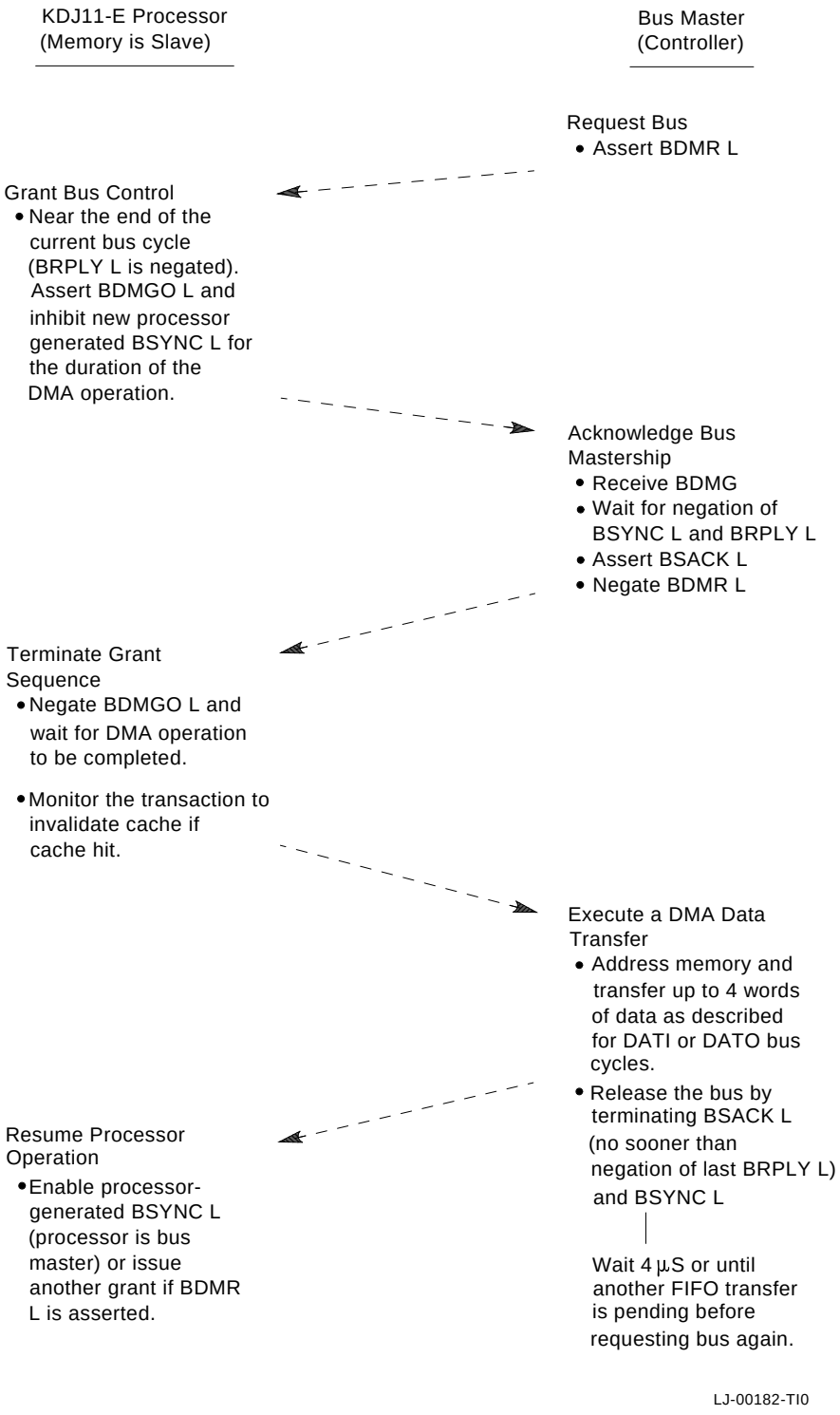
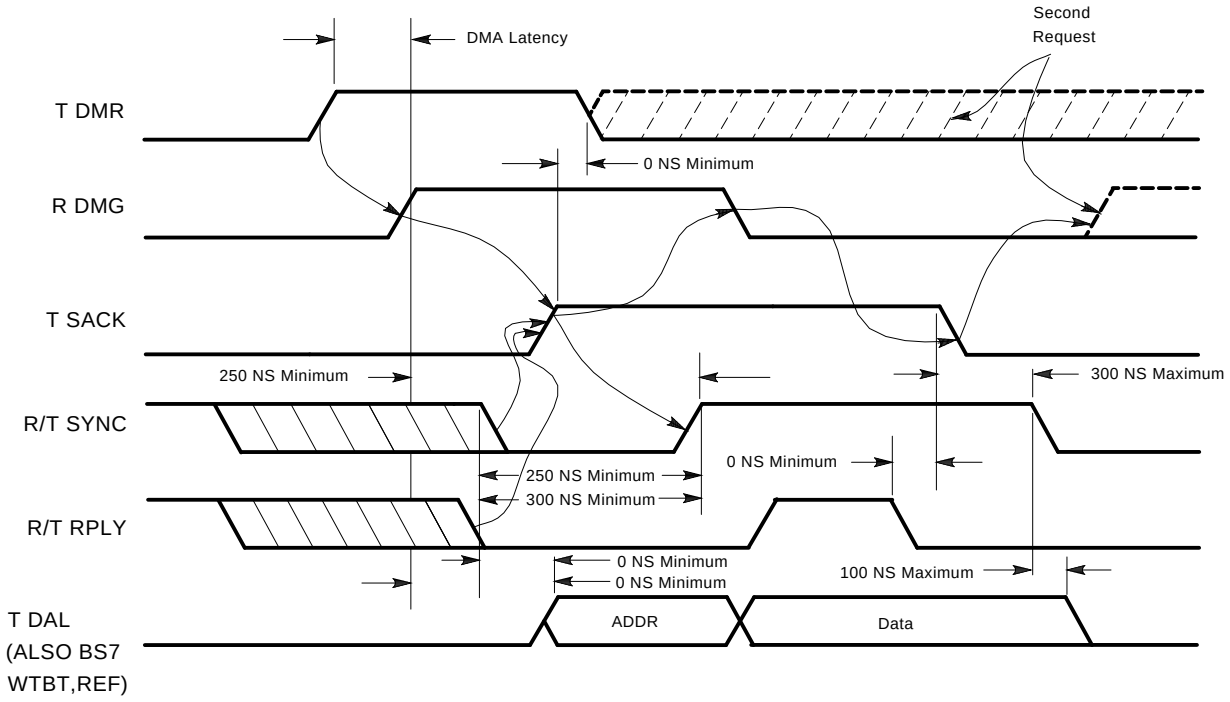


Figure F–8 DMA Request/Grant Timing



NOTES:

1. Timing shown at requesting device bus driver inputs and bus receiver outputs.
2. Signal name prefixes are defined below
   T=Bus Driver Input
   R=Bus Receiver Output
3. Bus driver output and bus receiver input signal names include a "B" prefix.

LJ-00183-TI0

F.4.2 Block Mode DMA

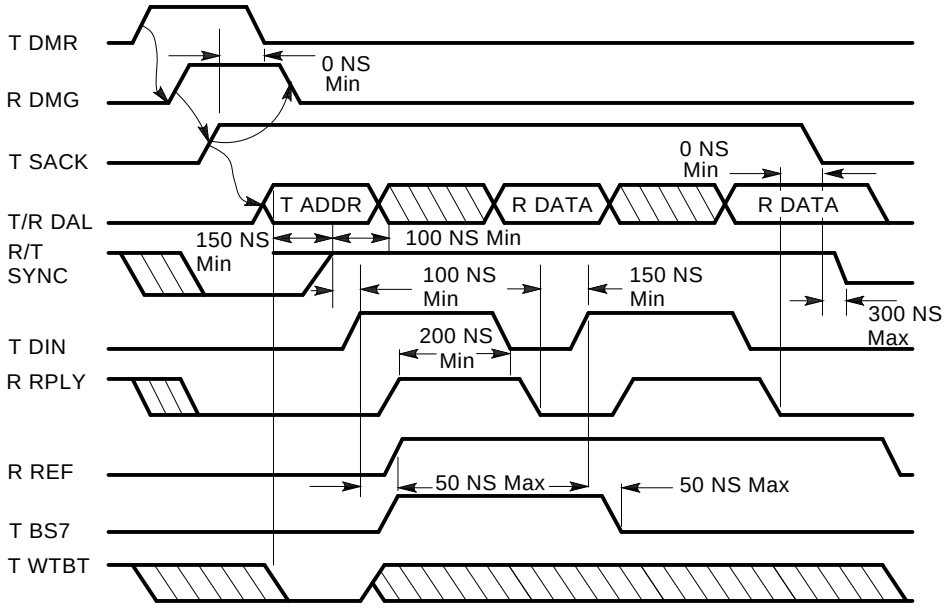
For increased throughput, block mode DMA can be implemented on a device for use with memories that support this type of transfer. In a block mode transaction, the starting memory address is asserted, followed by data for that address, and data for consecutive addresses.

By eliminating the assertion of the address for each data word, the transfer rate is almost doubled.

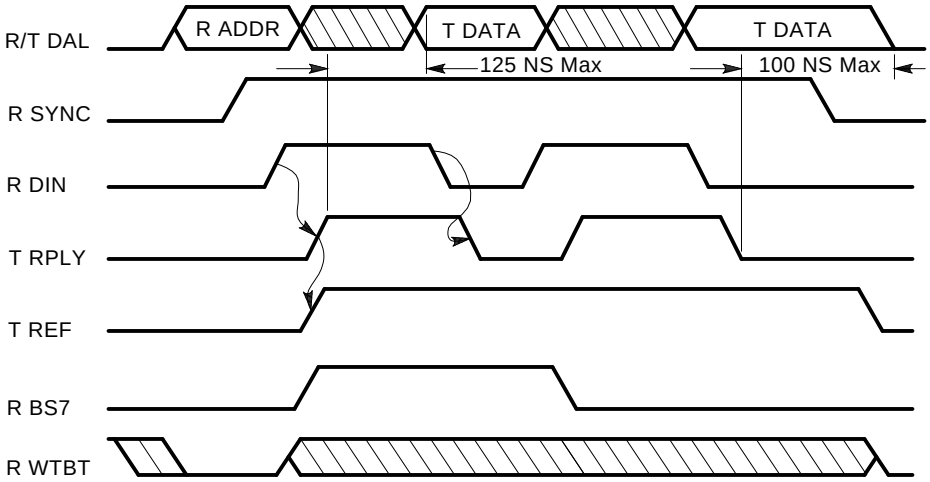
There are two types of block mode transfers, DATBI (input) and DATBO (output).

- Section F.4.2.1 describes the DATBI bus cycle (Figure F–9).
- Section F.4.2.2 describes the DATBO bus cycle (Figure F–10).

Figure F–9 DATBI Bus Cycle Timing



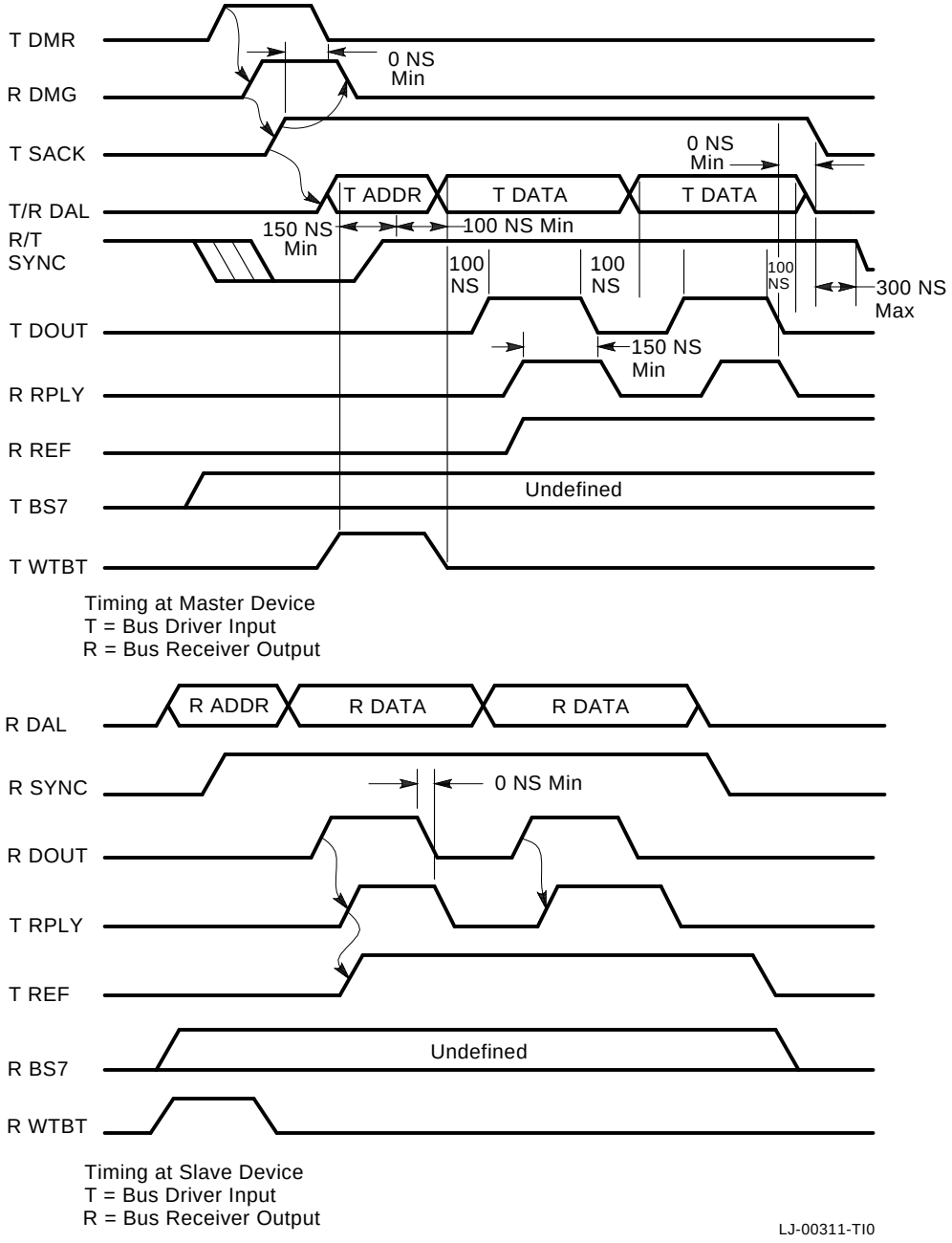
Timing at Master Device
T = Bus Driver Input
R = Bus Receiver Output



Timing at Slave Device
T = Bus Driver Input
R = Bus Receiver Output

LJ-00310-TI0

Figure F–10 DATBO Bus Cycle Timing



F.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 that 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 no 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 both RBS7 and TREF were not 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.

F.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 that 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 were 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 no 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 were 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 were 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.

F.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.

F.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 KA680 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

F.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.

F.5.2 Interrupt Protocol

Interrupt protocol on the Q22–bus has three phases:

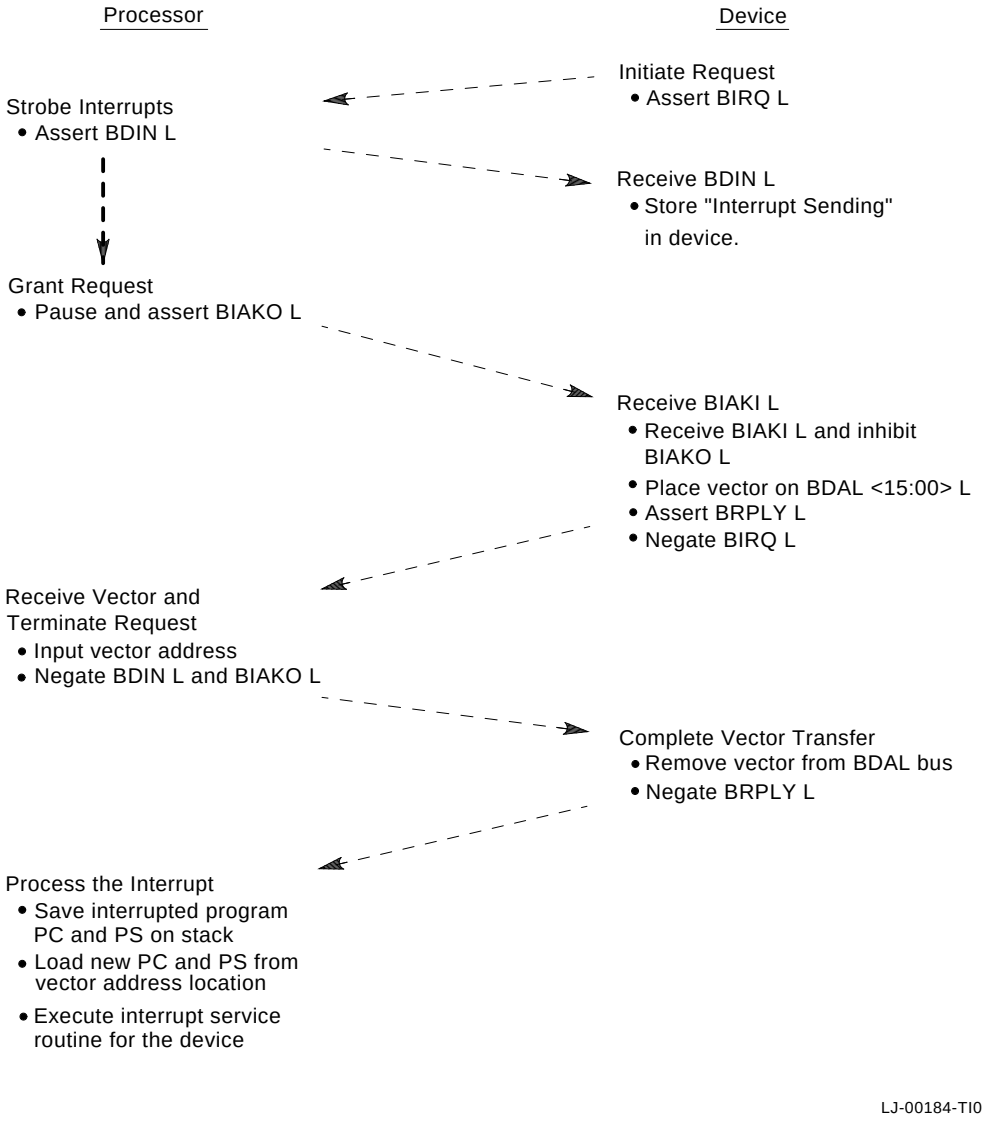
- Interrupt request
- Interrupt acknowledge and priority arbitration
- Interrupt vector transfer phase

The interrupt request phase begins when a device meets its specific conditions for interrupt requests. For example, the device is ready, done, or an error occurred. The interrupt enable bit in a device status register must be set. The device then initiates the interrupt by asserting the interrupt request line(s). BIRQ4 L is the lowest hardware priority level and is asserted for all interrupt requests for compatibility with previous Q22–bus processors. The level at which a device is configured must also be asserted. A special case exists for level 7 devices that must also assert level 6. The following list gives the interrupt levels and the corresponding Q22–bus interrupt request lines. For an explanation, refer to Section F.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 F–11 shows the interrupt request/acknowledge sequence.

Figure F–11 Interrupt Request/Acknowledge Sequence



The interrupt request line remains asserted until the request is acknowledged.

During the interrupt acknowledge and priority arbitration phase, the processor acknowledges interrupts under the following conditions:

- The device interrupt priority is higher than the current PS<7:5>.
- The processor has completed instruction execution and no additional bus cycles are pending.

The processor acknowledges the interrupt request by asserting BDIN L, and 150 ns (minimum) later asserting BIAKO L. The device electrically closest to the processor receives the acknowledge on its BIAKI L bus receiver.

At this point, the two types of arbitration must be discussed separately. If the device that receives the acknowledge uses the 4-level interrupt scheme, it reacts as follows:

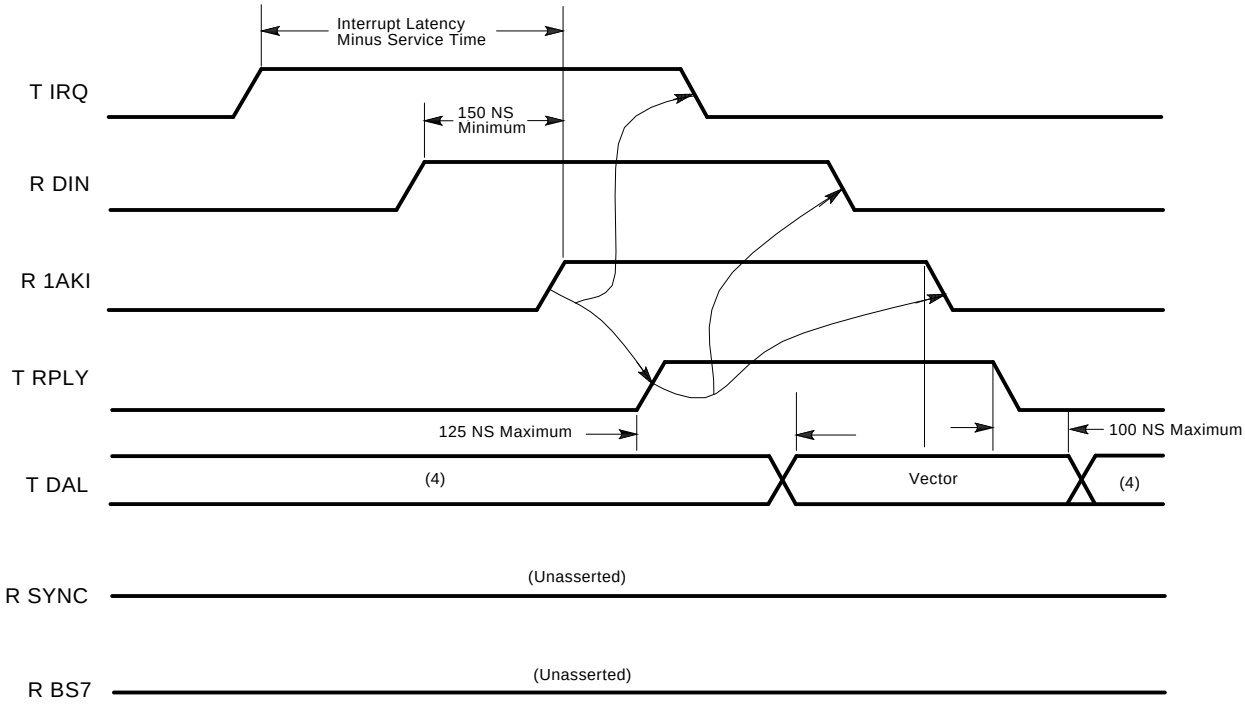
- If not requesting an interrupt, the device asserts BIAKO L and the acknowledge propagates to the next device on the bus.
- If the device is requesting an interrupt, it must check that no higher-level device is currently requesting an interrupt. This is done by monitoring higher-level request lines. The 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 F–12).

Figure F–12 Interrupt Protocol Timing



NOTES:

1. Timing shown at requesting device bus driver inputs and bus receiver outputs.
2. Signal name prefixes are defined below
   T=Bus Driver Input
   R=Bus Receiver Output
3. Bus driver output and bus receiver input signal names include a "B" prefix.
4. Don't care condition.

LJ-00185-TI0

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.

F.5.3 Q22–bus 4-level Interrupt Configurations

If you have high-speed peripherals and desire better software performance, you can use the 4-level interrupt scheme. Both position-independent and position-dependent configurations can be used with the 4-level interrupt scheme.

Figure F–13 shows the position-independent configuration. This allows peripheral devices that use the 4-level interrupt scheme to be placed in the backplane in any order. These devices must send out interrupt requests and monitor higher-level request lines as described. The level 4 request is always asserted from a requesting device regardless of priority. If two or more devices of equally high priority request an interrupt, the device physically closest to the processor wins arbitration. Devices that use the single-level interrupt scheme must be modified, or placed at the end of the bus, for arbitration to function properly.

Figure F–13 Position-Independent Configuration

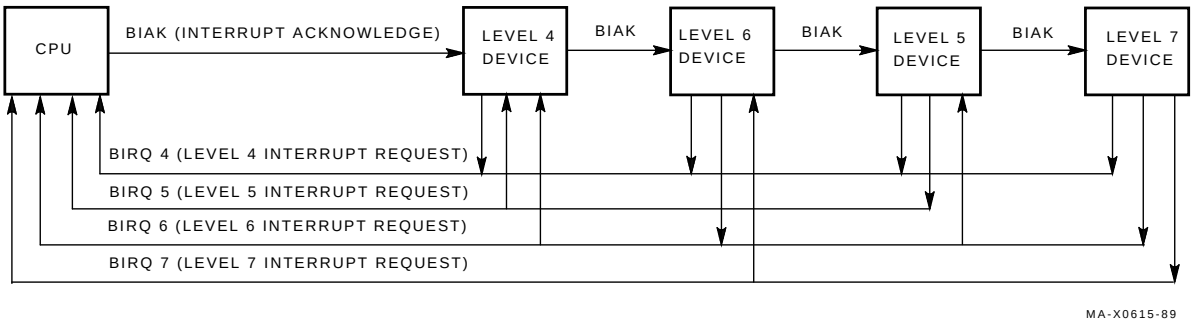
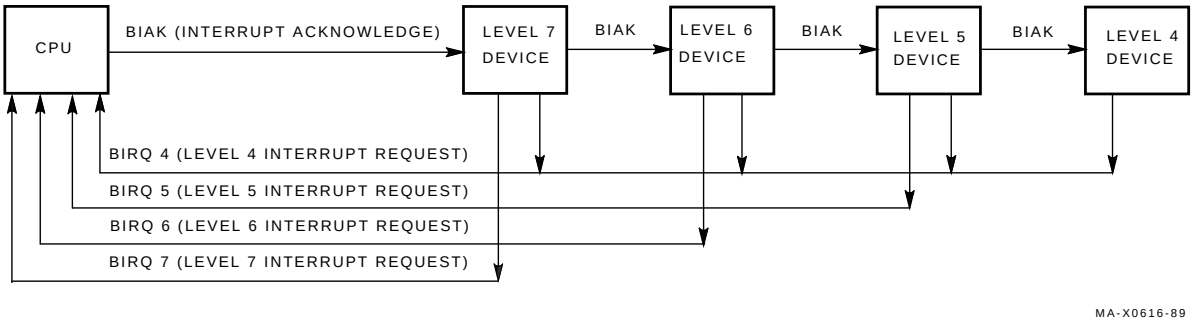


Figure F–14 shows the position-dependent configuration. This configuration is simpler to implement. A constraint is that peripheral devices must be inserted with the highest priority device located closest to the processor, and the remaining devices placed in the backplane in decreasing order of priority (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 F–14 Position-Dependent Configuration



F.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

F.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.

F.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.

F.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).

F.7 Q22–bus Electrical Characteristics

Section F.7.1 lists the input and output logic levels for Q22–bus signals.

F.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)

F.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.

F.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).

F.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.

F.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.

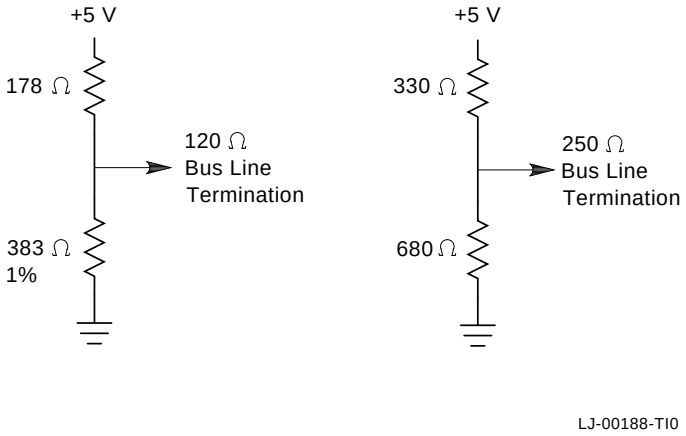
AC Specifications

- 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.

F.7.6 Bus Termination

The 120-ohm Q22–bus must be terminated at each end by an appropriate terminator, as shown in Figure F–15. This is to be done as a voltage divider with its Thevenin equivalent equal to 120 ohms and 3.4 V (nominal). This type of termination is provided by an REV11-A refresh/boot/terminator, BDV11-AA, KPV11-B, TEV11, or by certain backplanes and expansion cards.

Figure F–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.

F.7.7 Bus Interconnecting Wiring

The following sections give specific information about bus interconnecting wiring.

F.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).
- Cross talk between any two lines must be no greater than 5 percent. Note that worst-case cross talk 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.

F.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.

F.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.

F.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

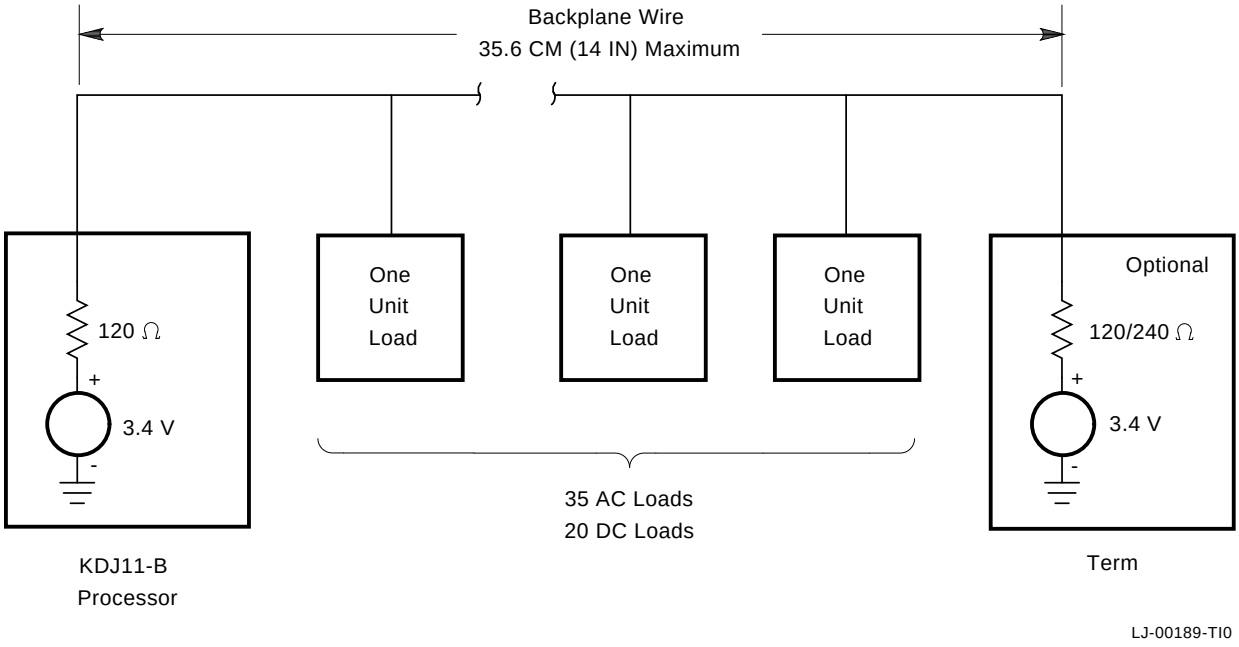
The ac and dc loads and the power consumption of the processor module, terminator module, and backplane must be included in determining the total loading of a backplane.

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 F–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.

Figure F–16 Single Backplane Configuration

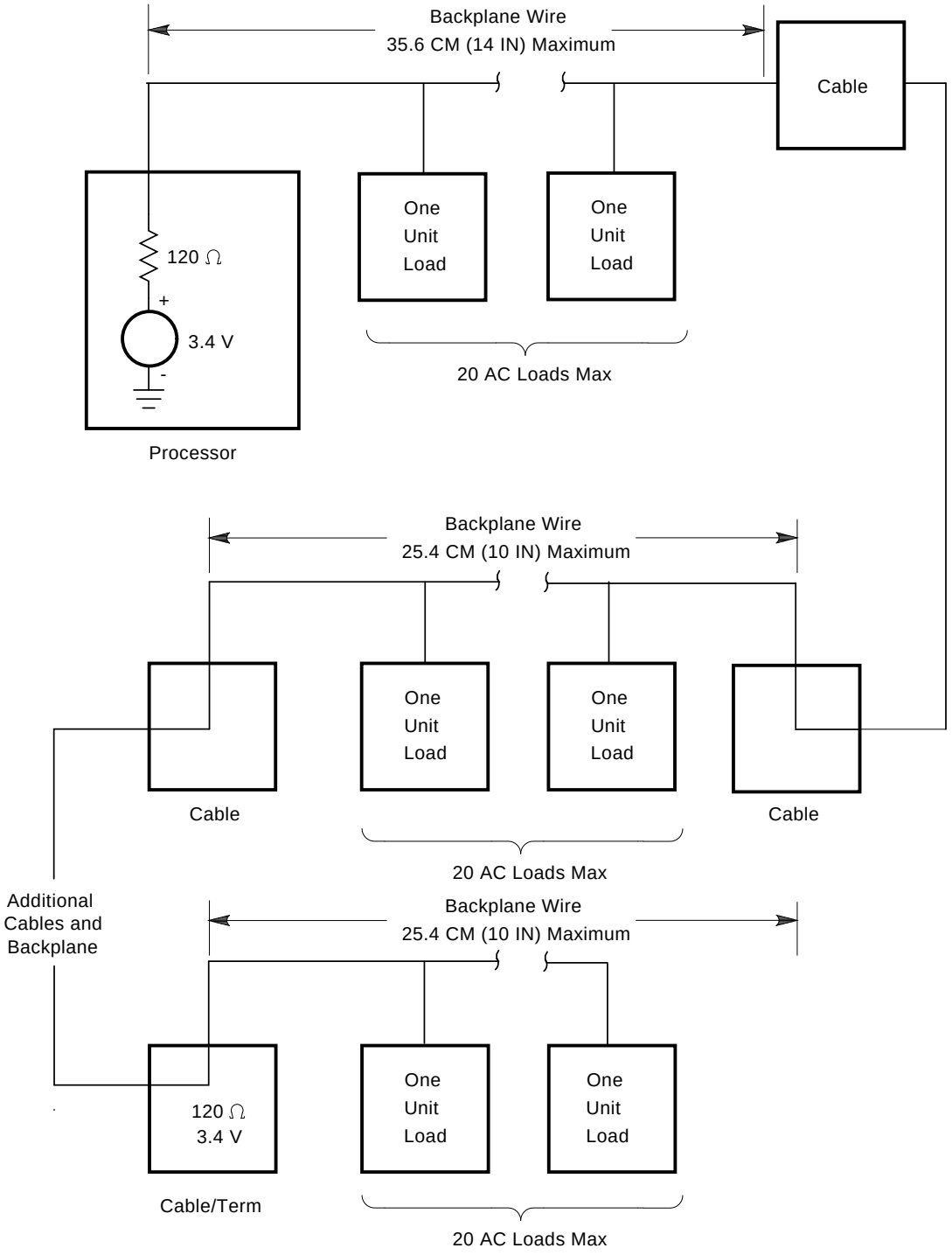


Rules for configuring multiple backplane systems are as follows:

- Figure F–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 on either 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.

Figure F–17 Multiple Backplane Configuration



Notes:
1. Two cables (max) 4.88 M (16 FT) (Max) total length.
2. 20 DC loads total (max).

LJ-00312-TI0

- 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.

F.8.1 Power Supply Loading

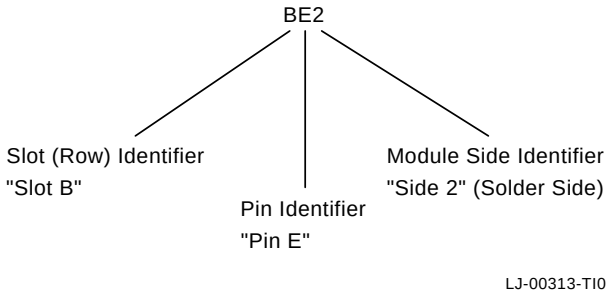
Total power requirements for each backplane can be determined by obtaining the total power requirements for each module in the backplane. Obtain separate totals for +5 V and +12 V power. Power requirements for each module are specified in the *Microcomputer Interfaces Handbook*.

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.

F.9 Module Contact Finger Identification

All of Digital's plug-in modules use the same contact finger (pin) identification system. A typical pin is shown in Figure F–18.

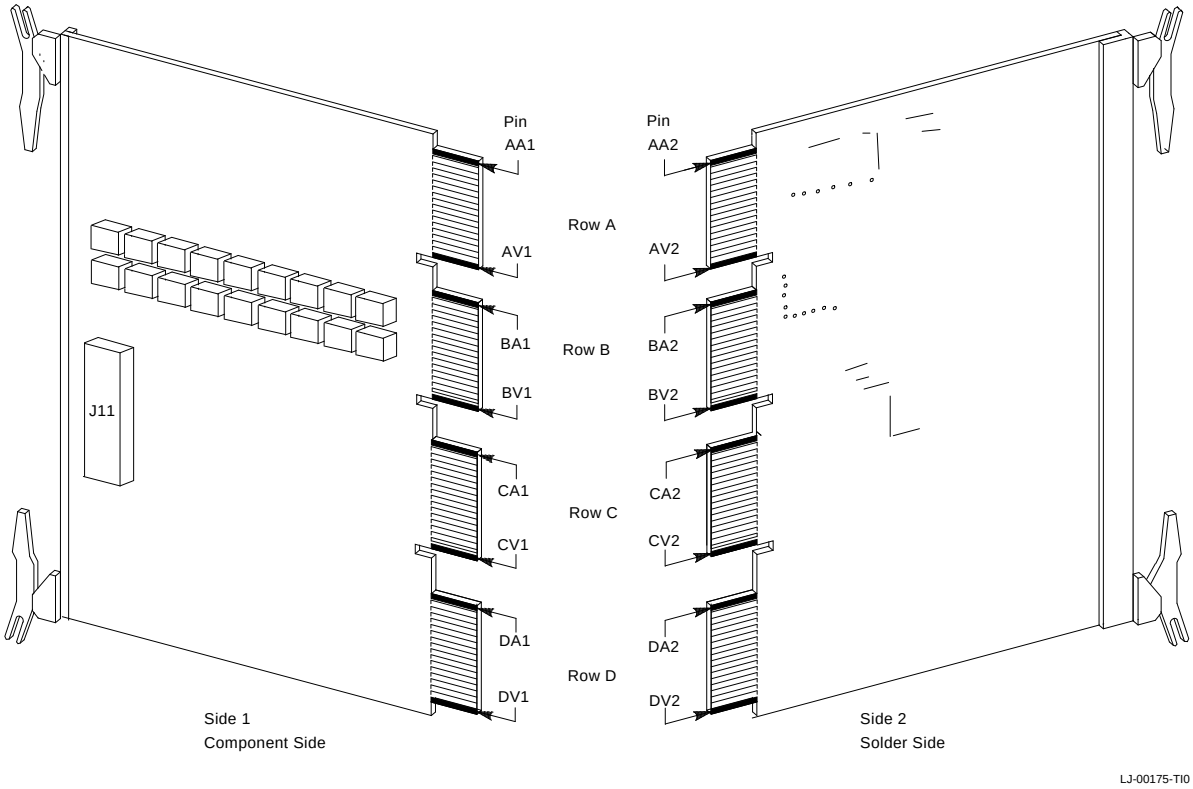
Figure F–18 Typical Pin Identification System



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 and the solder side is designated side 2, as shown in Figure F–19.

Figure F–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 F–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 F–20 represents the dimensions for a typical Q22–bus module.

Figure F–20 Typical Q22–bus Module Dimensions

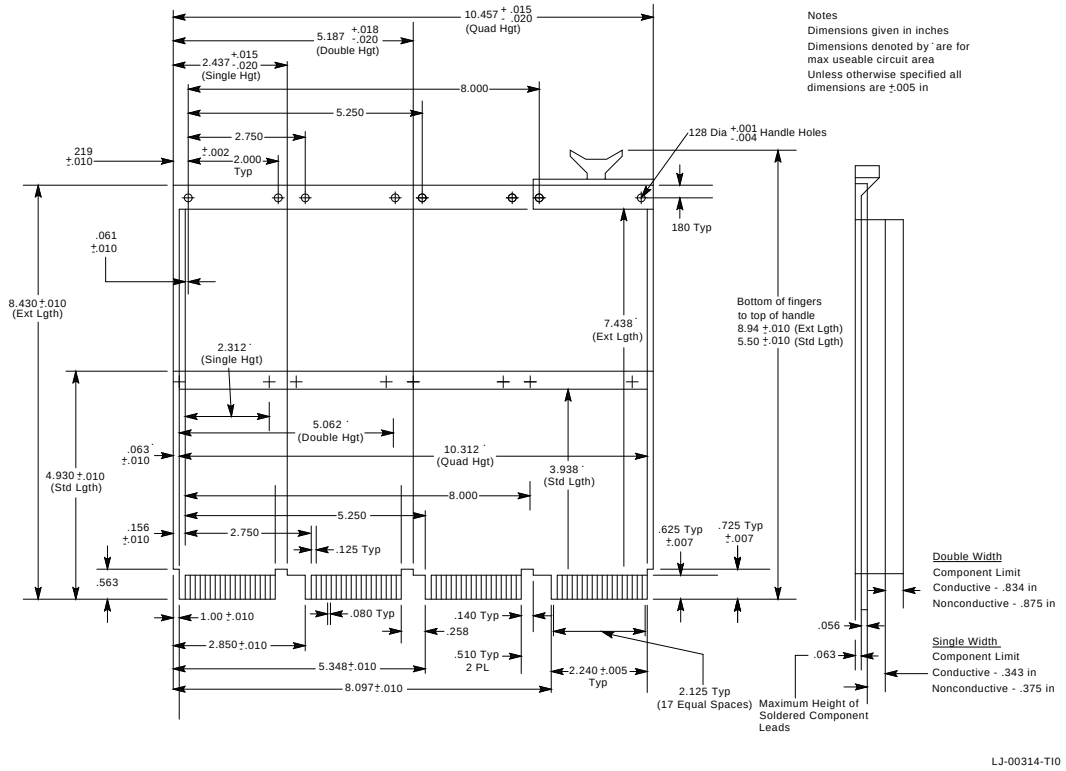


Table F–7 Bus Pin Identifiers

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.

(continued on next page)

Table F–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AF1	SSPARE2	Special spare — Not assigned or bused in Digital's cable or backplane assemblies. Available for user interconnection. In the highest priority device slot, the processor can use this pin for a signal to indicate its run state.
AH1	SSPARE3 SRUN	Special spare — Not assigned or bused simultaneously in Digital's cable or backplane assemblies; available for user interconnection. An alternate SRUN signal can be connected in the highest priority set.
AJ1	GND	Ground — System signal ground and dc return.
AK1	MSPAREA	Maintenance spare — Normally connected together on the backplane at each option location (not a bused connection).
AL1	MSPAREB	Maintenance spare — Normally connected together on the backplane at each option location (not a bused connection).
AM1	GND	Ground — System signal ground and dc return.
AN1	BDMR L	DMA request — A device asserts this signal to request bus mastership. The processor arbitrates bus mastership between itself and all DMA devices on the bus. If the processor is not bus master (it has completed a bus cycle and BSYNC L is not being asserted by the processor), it grants bus mastership to the requesting device by asserting BDMGO L. The device responds by negating BDMR L and asserting BSACK L.
AP1	BHALT L	Processor halt — When BHALT L is asserted for at least 25 µs, the processor services the halt interrupt and responds by halting normal program execution. External interrupts are ignored but memory refresh interrupts in Q22–bus operations are enabled if W4 on the M7264 and M7264-YA processor modules is removed and DMA request /grant sequences are enabled. The processor executes the ODT microcode, and the console device operation is invoked.

(continued on next page)

Table F–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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 Digital backplanes. A jumper is required on all Q22–bus options to open (disconnect) the back-up 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.
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.

(continued on next page)

Table F–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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 Digital 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.

(continued on next page)

Table F–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.

(continued on next page)

Table F–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AL2	BIRQ4 L	Interrupt request priority level 4 — A level 4 device asserts this signal when its interrupt enable and interrupt request flip-flops are set. If the PS word bit 7 is 0, the processor responds by acknowledging the request by asserting BDIN L and BIAKO L.
AM2 AN2	BIAKI L BIAKO L	Interrupt acknowledge — In accordance with interrupt protocol, the processor asserts BIAKO L to acknowledge receipt of an interrupt. The bus transmits this to BIAKI L of the device electrically closest to the processor. This device accepts the interrupt acknowledge under two conditions. • The device requested the bus by asserting 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.

(continued on next page)

Table F–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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.
BE2 BF2 BH2 BJ2 BK2 BL2 BM2 BN2 BP2 BR2 BS2 BT2 BU2 BV2	BDAL2 L BDAL3 L BDAL4 L BDAL5 L BDAL6 L BDAL7 L BDAL8 L BDAL9 L BDAL10 L BDAL11 L BDAL12 L BDAL13 L BDAL14 L BDAL15 L	Data/address lines — These 14 lines are part of the 16-line data/address bus.

G

Specifications

This appendix describes the physical, electrical, and environmental characteristics of the KA680 CPU module.

G.1 Dimensions

The KA680 and MS690 are quad-height modules with the following dimensions:

```
Height -   10.457 +.015/ -.020 inches
Length -    8.430 +.010/ -.010 inches
Width   -     .375 inches maximum (nonconductive)
       .343 inches maximum (conductive)
```

Note

Width, as defined for Digital modules, is the height of components above the surface of the module.

G.2 KA680 Connectors

The KA680 has two connector interfaces: the 270-pin backplane connector and the 100-pin console module connector.

G.2.1 KA680 Backplane Connector

The pinout of the KA680's 270-pin backplane connector is as follows:

Pin	Signal Name
001	SH2_DATA <7> L
002	GROUND
003	SH2_DATA <6> L
004	SH2_DATA <5> L
005	+ 5V
006	SH2_DATA <4> L
007	SH2_DATA <3> L
008	VM12VOLTSL1

Pin	Signal Name
009	SH2_DATA <2> L
010	SH2_DATA <1> L
011	GROUND
012	SH2_DATA <0>
013	SH2_DP L
014	V12VOLTS1
015	SH2_BSY L
016	SH2_ACK L
018	SH2_RST L
019	SH2_SEL L
021	SH2_CD L
022	SH2_REQ L
023	GROUND
024	SH2_IO L
026	+ 5V
027	BIRQ L<5>
028	BIRQ L<6>
029	VD3A
030	BDAL L<16>
031	BDAL L<17>
033	BDOUT L
034	SRUN L
035	GROUND
036	BRPLY L
037	BDIN L
039	BSYNC L
040	BWTBT L
042	BIRQ L<4>
045	BDMR L
046	BIAKO L
047	GROUND
048	BHALT L
049	BBS7 L
050	+ 5V
051	BREF L
054	BDMGO L
055	BINIT L
056	VM12VOLTSL2
057	BDAL L<0>
058	BDAL L<1>

Pin	Signal Name
059	GROUND
060	BDCOK
061	BPOK
062	V12VOLTS2
063	BDAL L<18>
064	BDAL L<19>
066	BDAL L<20>
067	BDAL L<2>
069	BDAL L<21>
070	BDAL L<3>
071	GROUND
072	BDAL L<4>
073	BDAL L<5>
074	+ 5V
075	BDAL L<6>
076	BDAL L<7>
077	VD3B
078	BDAL L<8>
079	BSACK L
081	BDAL L<9>
082	BIRQ L<7>
083	GROUND
084	BDAL L<10>
085	BEVENT L
087	BDAL L<11>
088	BDAL L<12>
090	BDAL L<13>
091	BDAL L<14>
093	BDAL L<15>
094	TRST
095	GROUND
096	MD<0>
097	MD<1>
098	+ 5V
099	MD<2>
100	MD<3>
101	V12VOLTS3
102	MD<4>
103	MD<5>
105	MD<6>

Pin	Signal Name
106	MD<7>
107	GROUND
108	MD<8>
109	MD<9>
111	MD<10>
112	MD<11>
114	MD<12>
115	MD<13>
117	MD<14>
118	MD<15>
119	GROUND
120	MD<16>
121	MD<17>
122	+ 5V
123	MD<18>
124	MD<19>
125	VD3C
126	MD<20>
127	MD<21>
129	MD<22>
130	MD<23>
131	GROUND
132	MD<24>
133	MD<25>
135	MD<26>
136	MD<27>
138	MD<28>
139	MD<29>
141	MD<30>
142	MD<31>
143	GROUND
144	MD<64>
145	MD<65>
146	+ 5V
147	MD<66>
148	MD<67>
149	V12VOLTS4
150	MD<68>
151	MD<69>
152	GROUND

Pin	Signal Name
153	MD<70>
154	NO CONNECTION
155	GROUND
156	CASA_H
157	CASB_H
160	SE_H
161	GROUND
162	WE_H
163	RAS_TIME_H
165	BANK_SEL<0>
167	GROUND
168	BANK_SEL<1>
169	BANK_SEL<2>
170	+ 5V
171	BANK_SEL<3>
172	MODE_SEL<0>
173	VD3D
176	GROUND
177	MODE_SEL<1>
178	MA<0>
179	GROUND
180	MA<1>
181	MA<2>
182	+ 5V
183	NMCJTDI
184	NMCJTDO
186	MA<3>
187	MA<4>
188	GROUND
189	MA<5>
190	MA<6>
191	GROUND
192	NMCJTMS
193	NMCJTCK
195	MA<7>
197	GROUND
198	MA<8>
199	NCAJTDI
201	NCAJTDO
202	NCAJTMS

Pin	Signal Name
203	GROUND
204	MA<9>
205	NCAJTCK
206	+ 5V
207	MA<10>
209	VD3E
211	MD<32>
213	MD<33>
214	MD<34>
215	GROUND
216	MD<35>
217	MD<36>
219	MD<37>
220	MD<38>
221	GROUND
222	MD<39>
223	MD<40>
225	MD<41>
226	MD<42>
227	GROUND
228	MD<43>
229	MD<44>
230	+ 5V
231	MD<45>
232	MD<46>
233	GROUND
234	MD<47>
235	MD<48>
237	MD<49>
238	MD<50>
239	GROUND
240	MD<51>
243	MD<52>
244	MD<53>
245	GROUND
246	MD<54>
247	MD<55>
249	MD<56>
250	MD<57>
251	GROUND

Pin	Signal Name
252	MD<58>
253	MD<59>
254	+ 5V
255	MD<60>
256	MD<61>
257	VD3F
258	MD<62>
259	MD<63>
260	NO CONNECTION
261	MD<71>
263	GROUND
264	NVAX_JTDI
265	NVAX_JTDO
267	NVAX_JTMS
268	NVAX_JTCK
270	TSTPHIIN

G.2.2 KA680 Console Connector (J2)

The 100-pin console connector provides the connection between the KA680 and the H3604 console module. Table G–1 lists the J2 pinouts.

Table G–1 KA680 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
1	GND	Ground	Signal Ground.
2 - 3	SH1_DATA<0> L	DSSI	DSSI #1 Data Bus Bit 0.
4	GND	Ground	Signal Ground.
5 - 6	SH1_DATA<1> L	DSSI	DSSI #1 Data Bus Bit 1.
7	GND	Ground	Signal Ground.
8 - 9	SH1_DATA<2> L	DSSI	DSSI #1 Data Bus Bit 2.
10	GND	Ground	Signal Ground.
11 - 12	SH1_DATA<3> L	DSSI	DSSI #1 Data Bus Bit 3.
13	GND	Ground	Signal Ground.
14 - 15	SH1_DATA<4> L	DSSI	DSSI #1 Data Bus Bit 4.
16	GND	Ground	Signal Ground.
17 - 18	SH1_DATA<5> L	DSSI	DSSI #1 Data Bus Bit 5.
19	GND	Ground	Signal Ground.
20 - 21	SH1_DATA<6> L	DSSI	DSSI #1 Data Bus Bit 6.
22	GND	Ground	Signal Ground.
23 - 24	SH1_DATA<7> L	DSSI	DSSI #1 Data Bus Bit 7.

(continued on next page)

Table G–1 (Cont.) KA680 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
25	GND	Ground	Signal Ground.
26 - 27	SH1_DP L	DSSI	DSSI #1 Data Bus Parity Line.
28	GND	Ground	Signal Ground.
29 - 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 - 33	SH1_RST L	DSSI	DSSI Pin RESET.
34	GND	Ground	Signal Ground.
35 - 36	SH1_SEL L	DSSI	DSSI Pin SELECT A signal. Used by the initiator to select a target.
37	GND	Ground	Signal Ground.
38 - 39	SH1_C/D L	DSSI	Pin Command/Data. A signal driven by a target that indicates whether control or data information is on the data bus. Asserted (low) indicates control.
40	GND	Ground	Signal Ground.
41 - 42	SH1_REQ L	DSSI	REQUEST. A signal driven by a target to indicate a request for a REQ/ACK data transfer handshake.
43	GND	Ground	Signal Ground.
44 - 45	SH1_I/O L	DSSI	Input/output. A signal driven by a target that controls the direction of data movement on the data bus with respect to the initiator. Asserted (low) indicates input.
46	GND	Ground	Signal Ground.
47 - 48	SH1_BSY L	DSSI	BUSY. 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.

(continued on next page)

Table G–1 (Cont.) KA680 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
53	TXD H	Ethernet	Console Terminal Data Out. This signal outputs serial character data from the console terminal transmitter.
54	GND	Ground	Signal Ground.
55	RXD H	Ethernet	Console Terminal Data In. 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	This is the transmit data signal.
60	GND	Ground	Signal Ground.
61	XMTEN H	Ethernet	This is the transmit enable signal.
62	GND	Ground	Signal Ground.
63	RCAR H	Ethernet	This is the receive enable signal.
64	GND	Ground	Signal Ground.
65	COL H	Ethernet	Collision Detect.
66	GND	Ground	Signal Ground.
67	RDATA H	Ethernet	RECEIVE DATA.
68	GND	Ground	Signal Ground.
69	TCLK H	Ethernet	Transmit Clock.
70	GND	Ground	Signal Ground.
71	RCLK H	Ethernet	Receive Clock.
72	GND	Ground	Signal Ground.
73	BITRATE <2> L	Console	Bit rate field bit 2.
74	BITRATE <1> L	Console	Bit rate field bit 1.
75	BITRATE <0> L	Console	Bit rate field bit 0.
76	LEDCODE <3> L	Console	This is bit 3 (MSB) of the LED code going to the hexadecimal display on the console module.

(continued on next page)

Table G–1 (Cont.) KA680 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
77	LEDCODE <2> L	Console	This is bit 2 of the LED code going to the hexadecimal display on the console module.
78	LEDCODE <1> L	Console	This is bit 1 of the LED code going to the hexadecimal display on the console module.
79	LEDCODE <0> L	Console	This is bit 0 (LSB) of the LED code going to the hexadecimal display on the console module.
80	VDDI H	Console	This pin is in the battery back-up supply for the SSC TOY clock.
81	DSSI1_UID <2> L	DSSI	DSSI #1 Node Identification (ID) number bit 2 (MSB).
82	DSSI1_UID <1> L	DSSI	DSSI #1 Node Identification (ID) number bit 1.
83	DSSI1_UID <0> L	DSSI	DSSI #1 Node Identification (ID) number bit 0 (LSB).
84	DSSI2_UID <2> L	DSSI	DSSI #2 Node Identification (ID) number bit 2 (MSB).
85	DSSI2_UID <1> L	DSSI	DSSI #2 Node Identification (ID) number bit 1.
86	DSSI2_UID <0> L	DSSI	DSSI #2 Node Identification (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	This is 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	–	–

(continued on next page)

Table G–1 (Cont.) KA680 Console Connector (J2) Pinout

Pin	Signal Name	Usage	Meaning
96	NC	–	–
97	NC	–	–
98	NC	–	–
99	NC	–	–
100	CABLE_OK_IN L	Console	This pin is used to indicate if the cable is properly installed.

G.3 DC Power Consumption

The KA680 CPU and MS690 memory module power requirements are as follows:

Module	Current (Amps)				Power (Watts)	Q22-Bus Loads	
	+5 Vdc	+3.3 Vdc	+12 Vdc	-12 Vdc	(total)	ac	dc
MS690-BA	5.3 A	0.0 A	0.0 A	0.0 A	26.5 W	0	0
MS690-CA	4.2 A	0.0 A	0.0 A	0.0 A	21.0 W	0	0
MS690-DA	6.4 A	0.0 A	0.0 A	0.0 A	32.0 W	0	0
KA680-AA(3)	2.8 A	3.2 A	0.0 A	0.0 A	24.6 W	4	1
KA680-AA(4)	4.8 A	3.2 A	1.6 A	0.0 A	53.8 W	4	1

NOTE: 1) MS690 Current and Wattage values are unique depending on option.
2) Memory modules are in dedicated slots 4 through 1.
3) Power data includes CPU module only.
4) Power power data includes CPU module, H3604 & Backplane power.

G.4 Battery Back-up Specifications

When dc power is supplied to the KA680 module, it charges the external batteries from +5 volts through a 240-ohm resistor.

When dc power is removed from the KA680 module, it drains the external batteries at a rate of 1.0 milliampere.

Note

These batteries supply power to the KA680 time-of-year clock and SSC RAM only. There is no battery backup for the memory system.

G.5 Operating Conditions

The KA680 module will meet or exceed the requirements for operation in a DEC Standard 102 Class B system environment. This includes an allowed 5°C rise for box preheating of the air.

TEMPERATURE

+5°C to +45°C (+40°F to +113°F) with a rate of change no greater than 20°C ±2°C (36°F ±4°F) per hour at sea level. The maximum temperature must be derated by 1.8°C per 1000 meters (1°F per 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 per minute (1000 feet per minute).

AIRFLOW

The airflow required to meet these specifications is 200 lfm.

G.6 Nonoperating Conditions (Fewer Than 60 Days)

TEMPERATURE

-40°C to +66°C (-40°F to +151°F) with a rate of change no greater than 11°C ±2°C (20°F ±4°F) per hour at sea level. The maximum temperature must be derated by 1.8°C per 1000 meters (1°F per 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 per minute (2000 feet per minute).

G.7 Nonoperating Conditions (More Than 60 Days)

TEMPERATURE

+5°C to +60°C (-40°F to +140°F) with a rate of change no greater than 20°C ±2°C (36°F ±4°F) per hour at sea level. The maximum temperature must be derated by 1.8°C per 1000 meters (1°F per 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 per minute (1000 feet per minute).

G.8 Mean Time Between Failures (MTBF) Estimate

The estimated module failure rate for the KA680 is one error per 322,000 hours at 32°C.

The estimated failure rate at 32°C for the MS690 memory modules are as follows:

Module	MTBF (Hours)	
MS690-BA (32 MB)	210,000	(With ECC off)
MS690-BA	417,000	(With ECC on)
MS690-CA (64 MB)	118,000	(With ECC off)
MS690-CA	426,000	(With ECC on)

H

VAX Instruction Set

The information in this appendix is for reference only.

The standard notation for operand specifiers is:

```
<name>.<access type><data type>
```

where:

1. Name is a suggested name for the operand in the context of the instruction. It is the capitalized name of a register or block for implied operands.
2. 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
   ```

3. 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)
   ```

4. Implied operands, that is, locations accessed by the instruction, but not specified in an operand, are denoted by curly braces {}.

The abbreviations for condition codes are:

```
*  =   conditionally set/cleared
-  =   not affected
0  =   cleared
1  =   set
```

The abbreviations for exceptions are:

```
rsv  =  reserved operand fault
iov  =  integer overflow trap
idvz =  integer divide by zero trap
fov  =  floating overflow fault
fuv  =  floating underflow fault
fdvz =  floating divide by zero fault
dov  =  decimal overflow trap
ddvz =  decimal divide by zero trap
sub  =  subscript range trap
prv  =  privileged instruction fault
```

Integer Arithmetic and Logical Instructions

Opcode	Instruction	N	Z	V	C	Exceptions
58	ADAWI add.rw, sum.mw	*	*	*	*	iov
80	ADDB2 add.rb, sum.mb	*	*	*	*	iov
C0	ADDL2 add.rl, sum.ml	*	*	*	*	iov
A0	ADDW2 add.rw, sum.mw	*	*	*	*	iov
81	ADDB3 add1.rb, add2.rb, sum.wb	*	*	*	*	iov
C1	ADDL3 add1.rl, add2.rl, sum.wl	*	*	*	*	iov
A1	ADDW3 add1.rw, add2.rw, sum.ww	*	*	*	*	iov
D8	ADWC add.rl, sum.ml	*	*	*	*	iov
78	ASHL cnt.rb, src.rl, dst.wl	*	*	*	0	iov
79	ASHQ cnt.rb, src.rq, dst.wq	*	*	*	0	iov
8A	BICB2 mask.rb, dst.mb	*	*	0	-	
CA	BICL2 mask.rl, dst.ml	*	*	0	-	
AA	BICW2 mask.rw, dst.mw	*	*	0	-	
8B	BICB3 mask.rb, src.rb, dst.wb	*	*	0	-	
CB	BICL3 mask.rl, src.rl, dst.wl	*	*	0	-	
AB	BICW3 mask.rw, src.rw, dst.ww	*	*	0	-	
88	BISB2 mask.rb, dst.mb	*	*	0	-	
C8	BISL2 mask.rl, dst.ml	*	*	0	-	
A8	BISW2 mask.rw, dst.mw	*	*	0	-	
89	BISB3 mask.rb, src.rb, dst.wb	*	*	0	-	
C9	BISL3 mask.rl, src.rl, dst.wl	*	*	0	-	
A9	BISW3 mask.rw, src.rw, dst.ww	*	*	0	-	
93	BITB mask.rb, src.rb	*	*	0	-	
D3	BITL mask.rl, src.rl	*	*	0	-	
B3	BITW mask.rw, src.rw	*	*	0	-	
94	CLRB dst.wb	0	1	0	-	
D4	CLRL{=F} dst.wl	0	1	0	-	
7C	CLRQ{=D=G} dst.wq	0	1	0	-	
B4	CLRW dst.ww	0	1	0	-	
91	CMPB src1.rb, src2.rb	*	*	0	*	
D1	CMPL src1.rl, src2.rl	*	*	0	*	
B1	CMPW src1.rw, src2.rw	*	*	0	*	
98	CVTBL src.rb, dst.wl	*	*	0	0	
99	CVTBW src.rb, dst.wl	*	*	0	0	
F6	CVTLB src.rl, dst.wb	*	*	*	0	iov
F7	CVTLW src.rl, dst.ww	*	*	*	0	iov
33	CVTWB src.rw, dst.wb	*	*	*	0	iov
32	CVTWL src.rw, dst.wl	*	*	0	0	

97	DECB dif.mb	* * * *	iov
D7	DECL dif.ml	* * * *	iov
B7	DECW dif.mw	* * * *	iov
86	DIVB2 divr.rb, quo.mb	* * * 0	iov,idvz
C6	DIVL2 divr.rl, quo.ml	* * * 0	iov,idvz
A6	DIVW2 divr.rw, quo.mw	* * * 0	iov,idvz
87	DIVB3 divr.rb, divd.rb, quo.wb	* * * 0	iov,idvz
C7	DIVL3 divr.rl, divd.rl, quo.wl	* * * 0	iov,idvz
A7	DIVW3 divr.rw, divd.rw, quo.ww	* * * 0	iov,idvz
7B	EDIV divr.rl, divd.rq, quo.wl, rem.wl	* * * 0	iov,idvz
7A	EMUL mulr.rl, muld.rl, add.rl, prod.wq	* * 0 0	
96	INCB sum.mb	* * * *	iov
D6	INCL sum.ml	* * * *	iov
B6	INCW sum.mw	* * * *	iov
92	MCOMB src.rb, dst.wb	* * 0 -	
D2	MCOML src.rl, dst.wl	* * 0 -	
B2	MCOMW src.rw, dst.ww	* * 0 -	
8E	MNEGB src.rb, dst.wb	* * * *	iov
CE	MNEGL src.rl, dst.wl	* * * *	iov
AE	MNEGW src.rw, dst.ww	* * * *	iov
90	MOVB src.rb, dst.wb	* * 0 -	
D0	MOVL src.rl, dst.wl	* * 0 -	
7D	MOVQ src.rq, dst.wq	* * 0 -	
B0	MOVW src.rw, dst.ww	* * 0 -	
9A	MOVZBW src.rb, dst.wb	0 * 0 -	
9B	MOVZBL src.rb, dst.wl	0 * 0 -	
3C	MOVZWL src.rw, dst.ww	0 * 0 -	
84	MULB2 mulr.rb, prod.mb	* * * 0	iov
C4	MULL2 mulr.rl, prod.ml	* * * 0	iov
A4	MULW2 mulr.rw, prod.mw	* * * 0	iov
85	MULB3 mulr.rb, muld.rb, prod.wb	* * * 0	iov
C5	MULL3 mulr.rl, muld.rl, prod.wl	* * * 0	iov
A5	MULW3 mulr.rw, muld.rw, prod.ww	* * * 0	iov
DD	PUSHL src.rl, {-(SP).wl}	* * 0 -	
9C	ROTL cnt.rb, src.rl, dst.wl	* * 0 -	
D9	SBWC sub.rl, dif.ml	* * * *	iov
82	SUBB2 sub.rb, dif.mb	* * * *	iov
C2	SUBL2 sub.rl, dif.ml	* * * *	iov
A2	SUBW2 sub.rw, dif.mw	* * * *	iov
83	SUBB3 sub.rb, min.rb, dif.wb	* * * *	iov
C3	SUBL3 sub.rl, min.rl, dif.wl	* * * *	iov
A3	SUBW3 sub.rw, min.rw, dif.ww	* * * *	iov
95	TSTB src.rb	* * 0 0	
D5	TSTL src.rl	* * 0 0	
B5	TSTW src.rw	* * 0 0	
8C	XORB2 mask.rb, dst.mb	* * 0 -	
CC	XORL2 mask.rl, dst.ml	* * 0 -	
AC	XORW2 mask.rw, dst.mw	* * 0 -	
8D	XORB3 mask.rb, src.rb, dst.wb	* * 0 -	
CD	XORL3 mask.rl, src.rl, dst.wl	* * 0 -	
AD	XORW3 mask.rw, src.rw, dst.ww	* * 0 -	

Address Instructions

Opcode	Instruction	N	Z	V	C	Exceptions
9E	MOVAB src.ab, dst.wl	*	*	0	-	
DE	MOVAL{=F} src.al, dst.wl	*	*	0	-	
7E	MOVAQ{=D=G} src.aq, dst.wl	*	*	0	-	
3E	MOVAW src.aw, dst.wl	*	*	0	-	
9F	PUSHAB src.ab, {-(SP).wl}	*	*	0	-	
DF	PUSHAL{=F} src.al, {-(SP).wl}	*	*	0	-	
7F	PUSHAQ{=D=G} src.aq, {-(SP).wl}	*	*	0	-	
3F	PUSHAW src.aw, {-(SP).wl}	*	*	0	-	

Variable Length Bit Field Instructions

Opcode	Instruction	N	Z	V	C	Exceptions
EC	CMPV pos.rl, size.rb, base.vb, {field.rv}, src.rl	*	*	0	*	rsv
ED	CMPZV pos.rl, size.rb, base.vb, {field.rv}, src.rl	*	*	0	*	rsv
EE	EXTV pos.rl, size.rb, base.vb, {field.rv}, dst.wl	*	*	0	-	rsv
EF	EXTZV pos.rl, size.rb, base.vb, {field.rv}, dst.wl	*	*	0	-	rsv
F0	INSV src.rl, pos.rl, size.rb, base.vb, {field.wv}	-	-	-	-	rsv
EB	FFC startpos.rl, size.rb, base.vb, {field.rv}, findpos.wl	0	*	0	0	rsv
EA	FFS startpos.rl, size.rb, base.vb, {field.rv}, findpos.wl	0	*	0	0	rsv

Control Instructions

Opcode	Instruction	N	Z	V	C	Exceptions
9D	ACBB limit.rb, add.rb, index.mb, displ.bw	*	*	*	-	iov
F1	ACBL limit.rl, add.rl, index.ml, displ.bw	*	*	*	-	iov
3D	ACBW limit.rw, add.rw, index.mw, displ.bw	*	*	*	-	iov
F3	AOBLEQ limit.rl, index.ml, displ.bb	*	*	*	-	iov
F2	AOBLSS limit.rl, index.ml, displ.bb	*	*	*	-	iov
1E	BCC{=BGEQU} displ.bb	-	-	-	-	
1F	BCS{=BLSSU} displ.bb	-	-	-	-	
13	BEQL{=BEQLU} displ.bb	-	-	-	-	
18	BGEQ displ.bb	-	-	-	-	
14	BGTR displ.bb	-	-	-	-	
1A	BGTRU displ.bb	-	-	-	-	
15	BLEQ displ.bb	-	-	-	-	
1B	BLEQU displ.bb	-	-	-	-	
19	BLSS displ.bb	-	-	-	-	
12	BNEQ{=BNEQU} displ.bb	-	-	-	-	
1C	BVC displ.bb	-	-	-	-	
1D	BVS displ.bb	-	-	-	-	
E1	BBC pos.rl, base.vb, displ.bb, {field.rv}	-	-	-	-	rsv
E0	BBS pos.rl, base.vb, displ.bb, {field.rv}	-	-	-	-	rsv
E5	BBCC pos.rl, base.vb, displ.bb, {field.mv}	-	-	-	-	rsv
E3	BBCS pos.rl, base.vb, displ.bb, {field.mv}	-	-	-	-	rsv
E4	BBSC pos.rl, base.vb, displ.bb, {field.mv}	-	-	-	-	rsv
E2	BBSS pos.rl, base.vb, displ.bb, {field.mv}	-	-	-	-	rsv
E7	BBCCI pos.rl, base.vb, displ.bb, {field.mv}	-	-	-	-	rsv
E6	BBSSI pos.rl, base.vb, displ.bb, {field.mv}	-	-	-	-	rsv

E9	BLBC src.rl, displ.bb	- - - -	
E8	BLBS src.rl, displ.bb	- - - -	
11	BRB displ.bb	- - - -	
31	BRW displ.bw	- - - -	
10	BSBB displ.bb, {-(SP).wl}	- - - -	
30	BSBW displ.bw, {-(SP).wl}	- - - -	
8F	CASEB selector.rb, base.rb, limit.rb, displ.bw-list	* * 0 *	
CF	CASEL selector.rl, base.rl, limit.rl, displ.bw-list	* * 0 *	
AF	CASEW selector.rw, base.rw, limit.rw, displ.bw-list	* * 0 *	
17	JMP dst.ab	- - - -	
16	JSB dst.ab, {-(SP).wl}	- - - -	
05	RSB {(SP)+.rl}	- - - -	
F4	SOBGEQ index.ml, displ.bb	* * * -	iov
F5	SOBGTR index.ml, displ.bb	* * * -	iov

Procedure Call Instructions

Opcode	Instruction	N Z V C	Exceptions
FA	CALLG arglist.ab, dst.ab, {-(SP).w*}	0 0 0 0	rsv
FB	CALLS numarg.rl, dst.ab, {-(SP).w*}	0 0 0 0	rsv
04	RET {(SP)+.r*}	* * * *	rsv

Miscellaneous Instructions

Opcode	Instruction	N Z V C	Exceptions
B9	BICPSW mask.rw	* * * *	rsv
B8	BISPSW mask.rw	* * * *	rsv
03	BPT {-(KSP).w*}	0 0 0 0	
00	HALT {-(KSP).w*}	- - - -	prv
0A	INDEX subscript.rl, low.rl, high.rl, size.rl, indexin.rl, indexout.wl	* * 0 0	sub
DC	MOVPSL dst.wl	- - - -	
01	NOP	- - - -	
BA	POPR mask.rw, {(SP)+.r*}	- - - -	
BB	PUSHR mask.rw, {-(SP).w*}	- - - -	
FC	XFC {unspecified operands}	0 0 0 0	

Queue Instructions

Opcode	Instruction	N Z V C	Exceptions
5C	INSQHI entry.ab, header.aq	0 * 0 *	rsv
5D	INSQTI entry.ab, header.aq	0 * 0 *	rsv
0E	INSQUE entry.ab, pred.ab	* * 0 *	
5E	REMQHI header.aq, addr.wl	0 * * *	rsv
5F	REMQTI header.aq, addr.wl	0 * * *	rsv
0F	REMQUE entry.ab, addr.wl	* * * *	

Operating System Support Instructions

Opcode	Instruction	N Z V C	Exceptions
BD	CHME param.rw, {-(ySP).w*}	0 0 0 0	
BC	CHMK param.rw, {-(ySP).w*}	0 0 0 0	
BE	CHMS param.rw, {-(ySP).w*}	0 0 0 0	
BF	CHMU param.rw, {-(ySP).w*} Where y=MINU(x, PSL<CURRENT_MODE>)	0 0 0 0	
06	LDPCTX {PCB.r*, -(KSP).w*}	- - - -	rsv, prv
DB	MFPR procreg.rl, dst.wl	* * 0 -	rsv, prv
DA	MTPR src.rl, procreg.rl	* * 0 -	rsv, prv
0C	PROBER mode.rb, len.rw, base.ab	0 * 0 -	
0D	PROBEW mode.rb, len.rw, base.ab	0 * 0 -	
02	REI {(SP)+.r*}	* * * *	rsv
07	SVPCTX {(SP)+.r*, PCB.w*}	- - - -	prv

Floating-Point Instructions

Opcode	Instruction	N Z V C	Exceptions
60	ADDD2 add.rd, sum.md	* * 0 0	rsv,fov,fuv
40	ADDF2 add.rf, sum.mf	* * 0 0	rsv,fov,fuv
40FD	ADDG2 add.rg, sum.mg	* * 0 0	rsv,fov,fuv
61	ADDD3 add1.rd, add2.rd, sum.wd	* * 0 0	rsv,fov,fuv
41	ADDF3 add1.rf, add2.rf, sum.wf	* * 0 0	rsv,fov,fuv
41FD	ADDG3 add1.rg, add2.rg, sum.wg	* * 0 0	rsv,fov,fuv
71	CMPD src1.rd, src2.rd	* * 0 0	rsv
51	CMPF src1.rf, src2.rf	* * 0 0	rsv
51FD	CMPG src1.rg, src2.rg	* * 0 0	rsv

6C	CVTBD src.rb, dst.wd	* * 0 0	
4C	CVTBF src.rb, dst.wf	* * 0 0	
4CFD	CVTBG src.rb, dst.wg	* * 0 0	
68	CVTDB src.rd, dst.wb	* * * 0	rsv, iov
76	CVTDF src.rd, dst.wf	* * 0 0	rsv, fov
6A	CVTDL src.rd, dst.wl	* * * 0	rsv, iov
69	CVTDW src.rd, dst.ww	* * * 0	rsv, iov
48	CVTFB src.rf, dst.wb	* * * 0	rsv, iov
56	CVTFD src.rf, dst.wd	* * 0 0	rsv
99FD	CVTFG src.rf, dst.wg	* * 0 0	rsv
4A	CVTFL src.rf, dst.wl	* * * 0	rsv, iov
49	CVTFW src.rf, dst.ww	* * * 0	rsv, iov
48FD	CVTGB src.rg, dst.wb	* * * 0	rsv, iov
33FD	CVTGF src.rg, dst.wf	* * 0 0	rsv,fov,fuv
4AFD	CVTGL src.rg, dst.wl	* * * 0	rsv, iov
49FD	CVTGW src.rg, dst.ww	* * * 0	rsv, iov
6E	CVTLD src.rl, dst.wd	* * 0 0	
4E	CVTLF src.rl, dst.wf	* * 0 0	
4EFD	CVTLG src.rl, dst.wg	* * 0 0	
6D	CVTWD src.rw, dst.wd	* * 0 0	
4D	CVTWF src.rw, dst.wf	* * 0 0	
4DFD	CVTWG src.rw, dst.wg	* * 0 0	
6B	CVTRDL src.rd, dst.wl	* * * 0	rsv, iov
4B	CVTRFL src.rf, dst.wl	* * * 0	rsv, iov
4BFD	CVTRGL src.rg, dst.wl	* * * 0	rsv, iov
66	DIVD2 divr.rd, quo.md	* * 0 0	rsv,fov,fuv, fdvz
46	DIVF2 divr.rf, quo.mf	* * 0 0	rsv,fov,fuv, fdvz
46FD	DIVG2 divr.rg, quo.mg	* * 0 0	rsv,fov,fuv, fdvz
67	DIVD3 divr.rd, divd.rd, quo.wd	* * 0 0	rsv,fov,fuv, fdvz
47	DIVF3 divr.rf, divd.rf, quo.wf	* * 0 0	rsv,fov,fuv, fdvz
47FD	DIVG3 divr.rg, divd.rg, quo.wg	* * 0 0	rsv,fov,fuv, fdvz
72	MNEGD src.rd, dst.wd	* * 0 0	rsv
52	MNEGF src.rf, dst.wf	* * 0 0	rsv
52FD	MNEGG src.rg, dst.wg	* * 0 0	rsv
70	MOVD src.rd, dst.wd	* * 0 -	rsv
50	MOVF src.rf, dst.wf	* * 0 -	rsv
50FD	MOVG src.rg, dst.wg	* * 0 -	rsv
64	MULD2 mulr.rd, prod.md	* * 0 0	rsv,fov,fuv
44	MULF2 mulr.rf, prod.mf	* * 0 0	rsv,fov,fuv
44FD	MULG2 mulr.rg, prod.mg	* * 0 0	rsv,fov,fuv
65	MULD3 mulr.rd, muld.rd, prod.wd	* * 0 0	rsv,fov,fuv
45	MULF3 mulr.rf, muld.rf, prod.wf	* * 0 0	rsv,fov,fuv
45FD	MULG3 mulr.rg, muld.rg, prod.wg	* * 0 0	rsv,fov,fuv
62	SUBD2 sub.rd, dif.md	* * 0 0	rsv,fov,fuv
42	SUBF2 sub.rf, dif.mf	* * 0 0	rsv,fov,fuv
42FD	SUBG2 sub.rg, dif.mg	* * 0 0	rsv,fov,fuv
63	SUBD3 sub.rd, min.rd, dif.wd	* * 0 0	rsv,fov,fuv
43	SUBF3 sub.rf, min.rf, dif.wf	* * 0 0	rsv,fov,fuv
43FD	SUBG3 sub.rg, min.rg, dif.wg	* * 0 0	rsv,fov,fuv

73	TSTD src.rd	* * 0 0	rsv
53	TSTF src.rf	* * 0 0	rsv
53FD	TSTG src.rg	* * 0 0	rsv

Microcode-Assisted Emulated Instructions

The NVAX CPU provides microcode assistance for the macrocode emulation of these instructions. The CPU processes the operand specifiers, creates a standard argument list, and invokes an emulation routine to perform emulation.

Opcode	Instruction	N Z V C	Exceptions
20	ADDP4 addlen.rw, addaddr.ab, sumlen.rw, sumaddr.ab	* * * 0	rsv, dov
21	ADDP6 add1len.rw, add1addr.ab, add2len.rw, add2addr.ab, sumlen.rw, sumaddr.ab	* * * 0	rsv, dov
F8	ASHP cnt.rb, srclen.rw, srcaddr.ab, round.rb, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
35	CMPP3 len.rw, src1addr.ab, src2addr.ab	* * 0 0	
37	CMPP4 src1len.rw, src1addr.ab, src2len.rw, src2addr.ab	* * 0 0	
0B	CRC tbl.ab, inicrc.rl, strlen.rw, stream.ab	* * 0 0	
F9	CVTLP src.rl, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
36	CVTPL srclen.rw, srcaddr.ab, dst.wl	* * * 0	rsv, iov
08	CVTPS srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
09	CVTSP srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
24	CVTPT srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
26	CVTTP srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab	* * * 0	rsv, dov
27	DIVP divrlen.rw, divraddr.ab, divdlen.rw, divdaddr.ab, quolen.rw, quoaddr.ab	* * * 0	rsv,dov,ddvz
38	EDITPC srclen.rw, srcaddr.ab, pattern.ab, dstaddr.ab	* * * *	rsv, dov
39	MATCHC objlen.rw, objaddr.ab, srclen.rw, srcaddr.ab	0 * 0 0	
34	MOVP len.rw, srcaddr.ab, dstaddr.ab	* * 0 0	
2E	MOVTC srclen.rw, srcaddr.ab, fill.rb, tbladdr.ab, dstlen.rw, dstaddr.ab	* * 0 *	
2F	MOVTUC srclen.rw, srcaddr.ab, esc.rb, tbladdr.ab, dstlen.rw, dstaddr.ab	* * * *	
25	MULP mulrlen.rw, mulraddr.ab, muldlen.rw, muldaddr.ab, prodlen.rw, prodaddr.ab	* * * 0	rsv, dov
22	SUBP4 sublen.rw, subaddr.ab, diflen.rw, difaddr.ab	* * * 0	rsv, dov
23	SUBP6 sublen.rw, subaddr.ab, minlen.rw, minaddr.ab, diflen.rw, difaddr.ab	* * * 0	rsv, dov

I Address Assignments

This appendix provides a map of the VAX memory space.

I.1 KA680 General Local Address Space Map

VAX Memory Space

Address Range	Contents
0000 0000 - 1FFF FFFF	Local Memory Space (512 MB)

VAX I/O Space

Address Range	Contents
2000 0000 - 2000 1FFF	Local Q22-bus I/O Space (8 KB)
2000 2000 - 2003 FFFF	Reserved Local I/O Space (248 KB)
2008 0000 - 201F FFFF	Local Register I/O Space (1.5 MB)
2020 0000 - 23FF FFFF	Reserved Local I/O Space (62.5 MB)
2400 0000 - 27FF FFFF	Reserved Local I/O Space (64 MB)
2008 0000 - 2BFF FFFF	Reserved Local I/O Space (64 MB)
2C08 0000 - 2FFF FFFF	Reserved Local I/O Space (64 MB)
3000 0000 - 303F FFFF	Local Q22-bus Memory Space (4 MB)
3040 0000 - 33FF FFFF	Reserved Local I/O Space (60 MB)
3400 0000 - 37FF FFFF	Reserved Local I/O Space (64 MB)
3800 0000 - 3BFF FFFF	Reserved Local I/O Space (64 MB)
3C00 0000 - 3FFF FFFF	Reserved Local I/O Space (64 MB)
E004 0000 - E007 FFFF	Local ROM Space

I.2 KA680 Detailed Local Address Space Map

Local Memory Space (up to 512 MB)	0000 0000 - 1FFF FFFF
Q22-bus Map - Top 32 KB of Main Memory	
VAX I/O Space	

Local Q22-bus I/O Space	2000 0000 - 2000 1FFF
Reserved Q22-bus I/O Space	2000 0000 - 2000 0007
Q22-bus Floating Address Space	2000 0008 - 2000 07FF
User Reserved Q22-bus I/O Space	2000 0800 - 2000 0FFF
Reserved Q22-bus I/O Space	2000 1000 - 2000 1F3F
Interprocessor Comm Reg	2000 1F40
Reserved Q22-bus I/O Space	2000 1F44 - 2000 1FFF
Local Register I/O Space	2000 2000 - 2003 FFFF
Reserved Local Register I/O Space	2000 4000 - 2000 402F
SHAC1 SSWCR	2000 4030
Reserved Local Register I/O Space	2000 4034 - 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 - 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 - 2000 422F

KA680 DETAILED LOCAL ADDRESS SPACE MAP (Cont.)

SHAC2 SSWCR	2000 4230
Reserved Local Register I/O Space	2000 4234 - 2000 4243
SHAC2 SSHMA	2000 4244
SHAC2 PQBBR	2000 4248
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 - 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 - 2000 7FFF
NICSR0 - Vector Add, IPL, Sync/Async	2000 8000
NICSR1 - Polling Demand Register	2000 8004
NICSR2 - Reserved	2000 8008
NICSR3 - Receiver List Address	2000 800C
NICSR4 - Transmitter List Address	2000 8010
NICSR5 - Status Register	2000 8014
NICSR6 - Command and Mode Register	2000 8018
NICSR7 - System Base Address	2000 801C
NICSR8 - Reserved	2000 8020*
NICSR9 - Watchdog Timers	2000 8024*
NICSR10- Reserved	2000 8028*
NICSR11- Rev Num & Missed Frame Count	2000 802C*
NICSR12- Reserved	2000 8030*
NICSR13- Breakpoint Address	2000 8034*
NICSR14- Reserved	2000 8038*
NICSR15- Diagnostic Mode & Status	2000 803C
Reserved Local Register I/O Space	2000 8040 - 2003 FFFF

Q-22 Bus Local Register I/O Space		2008 0000 - 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 - 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 - 2008 3FFF
Boot and Diagnostic Reg (32 Copies)		2008 4000 - 2008 407C
Reserved Local Register I/O Space		2008 4080 - 2008 7FFF

I.2 KA680 Detailed Local Address Space Map

NCA CSRs	
Error Status Register	2102 0000
Mode Control and Diagnostic Reg	2102 0004
CP1 Slave Error Address Register	2102 0008
CP2 Slave Error Address Register	2102 000C
CP1 IO Error Address Register	2102 0010
CP2 IO Error Address Register	2102 0014
NDAL Error Address Register	2102 0018
Local UVROM Space	E004 0000 - E007 FFFF
VAX System Type Register (In ROM)	E004 0004
Local UVROM - (Halt-Protected)	E004 0000 - E007 FFFF

```
**********************************************************************
The following addresses allow those KA680 internal processor
registers that are implemented in the SSC chip (external, internal
processor registers) to be accessed via the local I/O page. These
addresses are documented for diagnostic purposes only and should
not be used by 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 - 2014 00DB

    I/O Bus Reset Register              2014 00DC
    Reserved Local Register I/O Space   2014 00E0

    Reserved Local Register I/O Space   2014 00FC - 2014 00FF

 * These registers are not fully implemented, accesses yield
   unpredictable results.


**********************************************************************


Local Register I/O Space (Cont.)
    Timer 0 Control Register            2014 0100
    Timer 0 Interval Register           2014 0104
    Timer 0 Next Interval Register      2014 0108
    Timer 0 Interrupt Vector            2014 010C
    Timer 1 Control Register            2014 0110
    Timer 1 Interval Register           2014 0114
    Timer 1 Next Interval Register      2014 0118
    Timer 1 Interrupt Vector            2014 011C
    Reserved Local Register I/O Space   2014 0120 - 2014 012F

    BDR Address Decode Match Register   2014 0140
    BDR Address Decode Mask Register    2014 0144
    Reserved Local Register I/O Space   2014 0138 - 2014 03FF

    Battery Backed-Up RAM               2014 0400 - 2014 07FF
    Reserved Local Register I/O Space   2014 0800 - 201F FFFF

Reserved Local I/O Space                2020 0000 - 2FFF FFFF

Local Q22-bus Memory Space              3000 0000 - 303F FFFF

Reserved Local Register I/O Space       3040 0000 - 3FFF FFFF
```

I.3 External, Internal Processor Registers

Several of the internal processor registers (IPRs) on the KA680 are implemented in the NCA or SSC chip rather than the CPU chip. These registers are referred to as external, internal

processor registers and are listed below.

IPR #	Register Name	Abbrev.
27	Time-of-Year Register	TOY
28	Console Storage Receiver Status	CSRS*
29	Console Storage Receiver Data	CSRD*
30	Console Storage Transmitter Status	CSTS*
31	Console Storage Transmitter Data	CSDB*
32	Console Receiver Control/Status	RXCS
33	Console Receiver Data Buffer	RXDB
34	Console Transmitter Control/Status	TXCS
35	Console Transmitter Data Buffer	TXDB
55	I/O System Reset Register	IORESET

* These registers are not fully implemented, accesses yield unpredictable results.

I.4 Global Q22–bus Address Space Map

Q22-bus Memory Space	
Q22-bus Memory Space (Octal)	0000 0000 - 1777 7777
Q22-bus I/O Space (BBS7 Asserted)	
Q22-bus I/O Space (Octal)	1776 0000 - 1777 7777
Reserved Q22-bus I/O Space	1776 0000 - 1776 0007
Q22-bus Floating Address Space	1776 0010 - 1776 3777
User Reserved Q22-bus I/O Space	1776 4000 - 1776 7777
Reserved Q22-bus I/O Space	1777 0000 - 1777 7477
Interprocessor Comm Reg	1777 7500
Reserved Q22-bus I/O Space	1777 7502 - 1777 7777

J

Configurable Machine State

The KA680 CPU module has many control registers that need to be configured for proper operation of the module. The following list shows the normal state of all configurable bits in the CPU module as they are left after the successful completion of power-up ROM diagnostics.

```
NCA:
-----
NCA_CSR1:       Mode Control and Diagnostic Status Register  (2102 0004)
                15:14:  CP2 MT Timer Prescaler
                        11 = 144000 cycles*  - needed for CQBIC 10 ms No Grant
                                timeout

                13:12:  CP1 MT Timer Prescaler
                        00 = 144 cycles - minimum for passive releases, no
                                cycle should take longer than this

                11:10:  NDAL Timeout Prescaler
                        00 = 3200 cycles* - this is longer than both NCA and
                                NMC transactions timeouts, preserves timeout
                                order

                9:      QBUS_TRANS enable (formerly CQBIC_PRESENT)
                        0 =  QBUS_TRANS signal disabled* -

                8:      IO2 ID enable
                        1 = enabled

                7:      Force wrong CP2 bus parity
                        0 = off* - diagnostic use only

                6:      Force wrong CP1 bus parity
                        0 = off* - diagnostic use only

                5:      Force wrong NDAL master parity
                        0 = off* - diagnostic use only

                4:      Force wrong NDAL slave parity
                        0 = off* - diagnostic use only

                3:      Enable prefetch
                        1 = enable CP bus prefetch on DMA reads

                2:      Force write buffer hit
                        0 = off* - diagnostic use only

                1:      Force CP2 bus owner
                        0 = disabled - diagnostic use only

                0:      Force CP1 bus owner
                        0 = disabled - diagnostic use only

ICCS:           Interval Clock Control and Status Register  (2100 0060)

                NOTE: VMS sets ICCS, NICR to proper values

                6:      Interrupt enable
                        0 = disabled*
```

```
                5:      Single step
                        0 = off*

                4:      Transfer
                        0 = disabled*

                0:      Run - increment every 1 µs
                        0 = do not increment*

NICR:           Next Interval Count Register  (2100 0064)
                31:0    Initial count value for ICR (FFFFD8F0* (10 ms))


NMC:
-----
MEMCON0-7:     Memory Configuration Registers  (2101 8000 through 2101 801C)

                NOTE: Diagnostics set these registers based on available memory

                31:     Base Address Valid
                        0 = not valid*
                        1 = valid

                28:24:  Base Address (0 on reset)
                        1 MB RAM - all address bits used
                        4 MB RAM - only <28:26> used

                2:1     RAM size
                        00 = 1 MB RAM*
                        01 = 1 MB RAM
                        10 = 4 MB RAM
                        11 = nonexistent bank

                0:      Mode
                        1 = 64-bit mode

MMCDSR   :      Mode Control and Diagnostic Status Register  (2101 8048)
                31:     Fast Diagnostic Mode (FDM)
                        0 = disabled* - diagnostic use only

                30:     FDM Second pass
                        0 = disabled* - diagnostic use only

                29:     Diagnostic Checkbit mode
                        0 = disabled* - diagnostic use only

                28:     QBus on I01
                        0 = QBus on IO2*

                27:     Enable soft error log (NDAL & memory related)
                        0 = disabled* - VMS enables this

                26:     Flush BCache
                        0 = don't flush*

                24:17:  Memory diagnostic check bits
                        0 - meaningful only in diagnostic check mode* (may or
                                may not be read as 0)

                8:7:    NDAL Timeout Scaler
                        00 = 2600 cycles* - maximum, to preserve timeout order

                6:      Disable memory error
                        0 = memory errors deteted and corrected*

                5:      Refresh interval timer select
                        0 = 328 cycles*

                4:2:    Force wrong parity on NDAL transactions
                        0 = off* - diagnostic use only

                1:      Disable memory refresh
                        0 = memory refreshed*
```

```
                0:      Force refresh
                        0 = normal refresh*

MOAMR    :      O-bit Address and Mode Register  (2101 804C)
                16:     Ignore O-bit mode
                        0 = O-bits checked*

                15:     Disable O-bit error
                        0 = O-bit errors detected*

                14:6:   O-bit segment address (0*) - meaningful only during
                                O-bit data register access

                5:3:    O-bit mask (0*) - meaningful only during O-bit data
                                register access

                2:0:    O-bit operation mode
                        000 = reconstruction mode* - meaningful only during
                                O-bit data register access

MODR     :      O-bit Data Registers  (2101 0000 through 2101 7FFF)
                23:12:  O-bit field 1 (0*) - used only during Fast Memory test

                11:0:   O-bit field 0 (0*) - used only during Fast O-bit test
                        mode

NVAX:
-----
CPUID:          CPU ID Register  (IPR E)
                7:0:    CPU identifcation = 0 (for single processor config.)

SID:            System Identification Register  (IPR 3E)
                NOTE: This register may only be written by microcode

                31:24:  CPU type - 13hex (NVAX code)

                13:8:   Patch revision

                7:0:    Microcode revision

ICSR:           IBox Control and Status Register (IPR D3)
                0:      VIC enable
                        1 = enabled

ECR:            EBox Control Register  (IPR 7D)
                13:     FBox test enable
                        0 = disabled* - diagnostic use only

                7:      Interval time mode
                        1 = full CPU implemented interval timer

                5:      S3 stall timeout
                        0 = counts cycles w/ timeout_enable asserted* (~3 sec)

                3:      FBox stage 4 bypass
                        1 = enabled - result from stage 3 passed directly to
                                FBox output interface (improves FBox latency)

                2:      S3 external time base timeout
                        0 = disabled* - use internal time base

                1:      FBox enable
                        1 = enabled

                0:      Vector present
                        0 = no* - no vector option available at this time

MMAPEN:         Memory Map Enable Register  (IPR E6)
                0:      Memory map enable
                        0 = disabled* - VMS enables this
```

```
PAMODE:         Physical Address Mode Register  (IPR E7)
                0:      Physical address mode
                        0 = 30-bit physical address space*

PCCTL:          PCache Control Register  (IPR F8)
                8:      PCache Electrical disable
                        0 = PCache enabled*

                7:5     MBox performance monitor mode
                        0 - diagnostic use only*

                4:      PCache error enable
                        1 = enables PCache error detection

                3:      Bank select during force hit mode
                        0 = left bank selected if force hit mode enabled*
                                - diagnostic use only

                2:      Force hit
                        0 = disabled* - diagnostic use only

                1:      I_enable
                        1 = enable PCache for IREAD, INVAL, I_CF commands

                0:      D_enable
                        1 = enable PCache for INVAL, D-stream read/write/fill
                            commands
CCTL:           CBox Control Register  (IPR A0)
                30:     Software ETM
                        0 = disabled* - diagnostic use only

                16:     Force NDAL parity error
                        0 = off* - diagnostic use only

                15:11:  Performance monitoring BCache access and hit type
                        0 - configures BCache for performance monitoring* -
                                meaningful only during performance monitoring

                10:     Disable CBox write packer
                        0 = write packer enabled* - improves write latency

                9:      Read timeout counter test
                        0 = test disabled* - use external time base for read
                            timeout counter

                8:      Software ECC
                        0 = use correct ECC*

                7:      Disable BCache errors
                        0 = BCache errors detected*

                6:      Force Hit
                        0 = disabled* - diagnostic use only

                5:4:    BCache size
                        00 = 128 KB* (KA680)

                3:2:    Data store speed
                        01 = 3 cycle read, 4 cycle write

                1:      Tag store speed
                        1 = 4 cycle read, 4 cycle write

                0:      Enable BCache
                        1 = enabled

CQBIC:
------
SCR:            System Configuration Register  (2008 0000)
                14:     Halt enable
                        1 = BHALT to CQBIC HALTIN pin to cause halts
```

```
                12:     Page prefetch disable
                        1 = map prefetch disabled - historical latency reasons

                7:      Restart enable
                        0 = QBus restart causes ARB power-up reset*

                3:1:    ICR offset address select bits
                        0 = no effect (AUX mode not supported)*

ICR:            Interprocessor Communication Register  (2000 1F40)
                8:      AUX Halt
                        0 = no halt (AUX mode not supported)

                6:      ICR interrupt enable
                        0 = interprocessor interrupts disabled - only
                                uniprocessor config. allowed

                5:      Local memory external access enable
                        0 = external access disabled* - VMS will configure map

QBMBR:          Q-Bus Map Base Address Register (2008 0010)
                28:15:  address where 8K QBus mapping register are located
                        (undefined at reset) - VMS will configure map

SHAC:
-----
  NOTE:  All SHAC registers are subsequently configured by VMS driver

PQBBR:          Port Queue Block Base Register  (2000 4048)
                20:0:   upper bits of physical address of base of Port Queue
                        block.  Contains HW version, FW version, shared host
                        memory version and CI port maintenance ID at power-up.

PPR:            Port Parameter Register  (2000 4058)
                31:29:  Cluster size.  For SHAC value = 0.

                28:16:  Internal buffer length = 0* (For SHAC value = 1010 hex)

                7:0:    Port number.  Same as SHAC's DSSI ID.

PMCSR:          Port Maintenance Control and Status Register  (2000 405C)
                2:      Interrupt enable
                        0 = disabled*

                1:      Maintenance timer disable
                        0 = enabled*


SGEC:
-----
  NOTE:  All SGEC registers are susequently configured by VMS driver

NICSR0:         Vector Address, IPL, Synch/Asynch Register  (2000 8000)
                31:30:  Interrupt priority
                        00 = 14*

                29:     Synch/Asynch bus master operating mode
                        0 = asynchronous*

                15:0:   Interrupt vector = 0003hex*

NICSR6:         Command and Mode Register  (2000 8018)
                30:     Interrupt enable
                        0 = disabled*

                28:25:  Burst limit mode
                        maximum number of longwords transferred in a single DMA
                        burst.  1*,2,4,8 when NICSR <19>is clear;
   1*,4 when set.

                20:     Boot message enable mode
                        0 = disabled*
```

```
                    19:     Single cycle enable mode
                            0 = disabled*

                    11:     Start/Stop transmission command
                            0 = SGEC transmission process in stopped state*

                    10:     Start/Stop reception command
                            0 = SGEC reception process in stopped state*

                    9:8:    Operating mode
                            00 = normal mode*

                    7:      Disable data chaining mode
                            0 = frames too long for current receive buffer will be
                                transferred to the next buffer(s) in receive list*

                    6:      Force collision mode (internal loopback mode only)
                            0 = no collision*

                    3:      Pass bad frames mode
                            0 = bad frames discarded*

                    2:1:    Address filtering mode
                            00 = normal mode*

NICSR7:             System Base Register  (2000 801C)
                    29:0:   System base address - physical starting address of the
                            VAX system page table (unpredictable after reset)

NICSR9:             Watchdog Timers Register  (2000 8024)
                    31:16:  Receive watchdog timeout
                            0 = never timeout*
                            default = 1250 = 2 ms
                            range = 72 µs (45) to 100 ms

                    15:0:   Transmit watchdog timeout
                            0 = never timeout*
                            default = 1250 = 2 ms
                            range = 72 µs (45) to 100 ms

SSC:
----
SSCBAR:             SSC Base Address Register  (2014 0000)
                    29:0    20140000 = Base address*

SSCCR:              SSC Configuration Register  (2014 0010)
                    27:     Interrupt vector disable
                            0 = interrupt vector enabled*

                    25:24:  IPL Level
                            00 = 14*

                    23:     ROM access time
                            0 = 350 ns*

                    22:20:  ROM size
                            101 = 256 KB

                    18:16:  Halt protected space
                            101 = 20040000 - 2007FFFF (historical)

                    15:     Control P enable
                            0 = only 20 spaces recognized as break* (historical)

                    14:12:  Terminal UART baud rate
                            101 = 9600 (historical)

                    6:      Programmable address strobe 1 ready enable (for BDR)
                            1 = ready asserted after address strobe

                    5:4:    Programmable address strobe 1 enable (for BDR)
                            11 = read enabled, write enabled
```

```
                2:      Programmable address strobe 0 ready enable
                        0 = no ready after address strobe* - not used by Omega

                1:0:    Programmable address strobe 0 enable
                        00 = read disabled, write disabled* - not used by Omega

RXCS:           Console Receiver Control and Status Register  (2014 0080)
                6:      Interrupt enable
                        0 = disabled*  - polled in console mode

TXCS:           Console Transmitter Control and Status Register (2014 0088)
                6:      Interrupt enable
                        0 = disabled*

                2:      Loopback enable
                        0 = disabled* - diagnostic use only

                0:      Break transmit
                        0 = terminate SPACE condition*

SSCBT:          SSC Bus Time Out Register  (2014 0020)
                23:0:   Bus timeout interval = 4000hex (16.384 ms)
                        range = 1 to FFFFFF (1 µs to 16.77 s)

ADS0MAT:        Programmable Address Strobe 0 Match Register  (2014 0130)
                29:2:   Match address
                        0 = disabled* - not used

ADS0MAS:        Programmable Address Strobe 0 Mask Register  (2014 0134)
                29:2:   Mask address bits - not used

ADS1MAT:        Programmable Address Strobe 1 Match Register  (2014 0140)
                29:2:   Match address = 20084000 (for BDR)

ADS1MAS:        Programmable Address Strobe 1 Mask Register  (2014 0144)
                29:2:   Mask address bits = 7C (for BDR)

T1CR:           Programmable Timer 0 Control Register  (2014 0100)
                6:      Interrupt enable
                        0 = disabled*

                2:      STP
                        0 = run after overflow*

                0:      RUN
                        0 = counter not running*  (historical)

T1CR:           Programmable Timer 1 Control Register  (2014 0110)
                6:      Interrupt enable
                        0 = disabled*

                2:      STP
                        0 = run after overflow*

                0:      RUN
                        1 = counter incrementing every microsecond (historical)

TNIR:           Programmable Timer Next Interval Registers  (2014 0108,
                                                             2014 0118)
                31:0:   Timer next interval count (use 2's complement)
                        range = 0* to 1.2 hours

T0IV:           Programmable Timer 0 Interrupt Vector Register  (2014 010C)
                9:2:    Timer interrupt vector = 78hex

T1IV:           Programmable Timer 1 Interrupt Vector Registers (2014 011C)
                9:2:    Timer interrupt vector = 7Chex

TOY:            Time of Year Register  (2014 006C)
                31:0:   Number of 10 ms intervals since written
```

```
DLEDR:          Diagnostic LED Register  (2014 0030)
                3:0:    Display bits
                        0 = LEDs on*  (historical)
```

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X