digital

KA650 CPU Module Technical Manual

Order Number EK-KA650-UG.003

KA650 CPU Module Technical Manual

Order Number EK-KA650-UG.003

digital equipment corporation
maynard, massachusetts

First Edition, December 1987
Second Edition, August 1988
Third Edition, March 1989

Contents

3 Architecture

Examples

Figures

Tables

About This Manual

The *KA650 CPU Module Technical Manual* documents the functional, physical, and environmental characteristics of the KA650-AA CPU module, and includes information on the MS650 memory expansion modules. The manual also covers the KA650-BA CPU module, designed for workstation usage. The KA650-BA is functionally equivalent to the KA650-AA, except that it does not support multiuser VMS and ULTRIX operating system licenses.

Intended Audience

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

Organization

The manual is divided into four chapters and four appendixes.

Chapter 1, **Overview**, introduces the KA650 MicroVAX CPU module and MS650 memory modules, including module features and specifications.

Chapter 2, **Installation and Configuration**, describes the installation and configuration of the KA650 and MS650 modules in Q22-bus backplanes and system enclosures.

Chapter 3, **Architecture**, describes the KA650 registers, instruction set, and memory.

Chapter 4, **KA650 Firmware**, describes the entry/dispatch code, boot diagnostics, device booting sequence, console program, and console commands.

Appendix A, **KA650 Specifications**, describes the physical, electrical, and environmental specifications for the KA650 CPU module.

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

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

Appendix D, **Acronyms**, lists the acronyms used in this manual.

Conventions

This manual uses the following 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 general purpose register R0.
[x:y]	Represents a range of bytes, from y through x.
$\boxed{\text{Return}}$	Text within a box identifies a key such as the $\boxed{\text{Return}}$ key.
Note	Provides general information you should be aware of.
Caution	Provides information to prevent damage to equipment.
n	Boldface small **n** indicates variables.

Related Documents

You can order the following documents from DIGITAL:

Document	Order Number
Microcomputer Interfaces Handbook	EB-20175-20
Microcomputers and Memories Handbook	EB-18451-20
VAX Architecture Handbook	EB-19580-20
VAX Architecture Reference Manual	EY-3459E-DP

You can order these documents from:

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

Attention: Documentation Products

1 Overview

This chapter provides a brief overview of the KA650 CPU module and MS650 memory modules.

1.1 KA650 Central Processor Module

The KA650 is a quad-height VAX processor module for the Q22-bus, also known as the extended LSI-11 bus. The KA650-AA is designed for use in high speed, real-time applications and for multiuser, multitasking environments. The KA650 incorporates a two-level cache to maximize performance.

The KA650 CPU module and MS650 memory modules combine to form a VAX CPU/memory subsystem that uses the Q22-bus to communicate with mass storage and I/O devices, as shown in Figure 1–3. The KA650 and MS650 modules are mounted in standard Q22-bus backplane slots that implement the Q22-bus in the AB rows and the CD interconnect in the CD rows. A single KA650 can support up to four MS650 modules, if enough Q22-bus/CD backplane slots are available.

The KA650 communicates with the console device through the H3600-SA CPU cover panel, which also contains configuration switches and an LED display.

Figure 1–1 shows the KA650 CPU module. Figure 1–2 shows the major functional blocks of the KA650 CPU module.

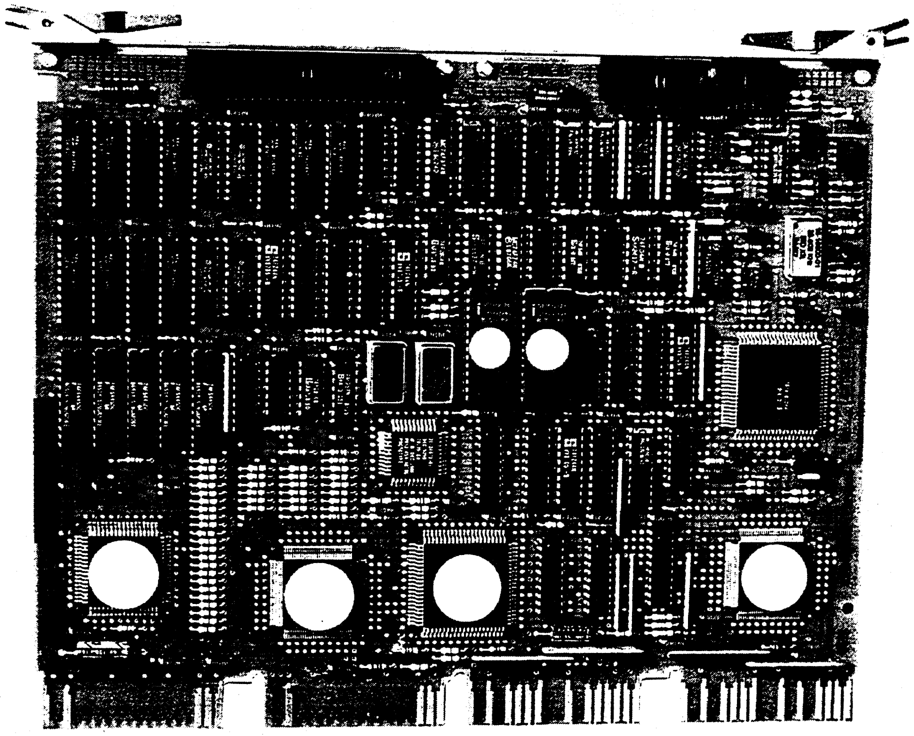


Figure 1–1 KA650 CPU Module

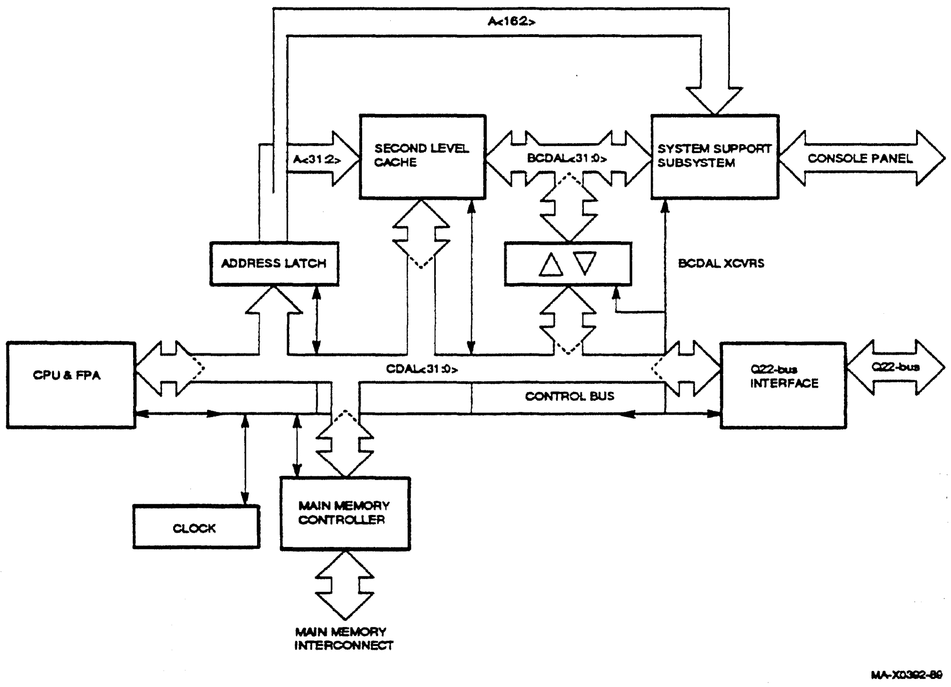


Figure 1–2 KA650 Block Diagram

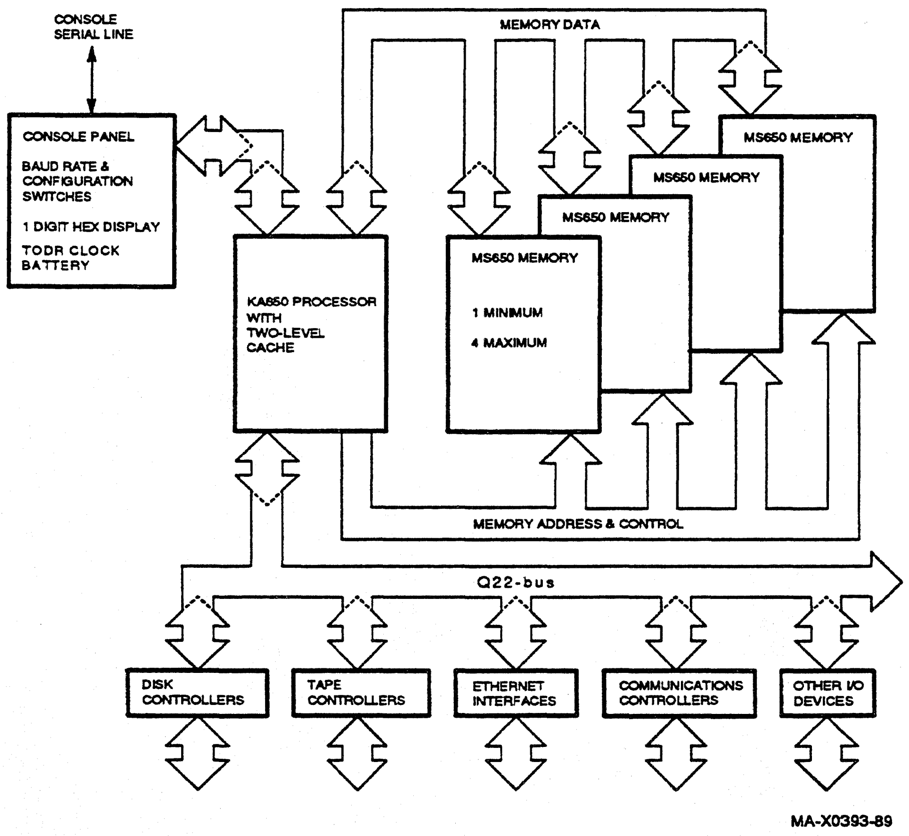


Figure 1–3 System Level Block Diagram

1.2 Clock Functions

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

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

1.3 Central Processing Unit

The central processing unit (CPU) is implemented by the CVAX chip. The CVAX chip contains approximately 180,000 transistors in an 84-pin CERQUAD surface mount package. The CPU achieves a 90 ns microcycle and a 180 ns bus cycle at an operating frequency of 22 MHz. The CVAX chip supports full VAX memory management and a 4 gigabyte virtual address space.

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

The CVAX chip provides the following functions:

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

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

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

- Byte
- Word
- Longword

- Quadword
- Character string
- Variable length bit field

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

1.4 Floating-Point Accelerator

The floating-point accelerator is implemented by the CFPA chip. The CFPA chip contains approximately 60,000 transistors in a 68-pin CERQUAD surface mount package. It executes 70 floating-point instructions. The CFPA chip receives opcode information from the CVAX chip, and receives operands directly from memory or from the CVAX chip. The floating-point result is always returned to the CVAX chip.

1.5 Cache Memory

The KA650 module incorporates a two-level cache to maximize CPU performance.

The first-level cache is implemented within the CVAX chip. The first-level cache is a 1 Kbyte, two-way associative, write through cache memory, with a 90 ns cycle time.

The second-level cache is implemented using 16K by 4-bit static RAMs. The second-level cache is a 64 Kbyte, direct mapped, write through cache memory, with a 180 ns cycle time for longword transfers, and 270 ns cycle time for quadword transfers.

1.6 Memory Controller

The main memory controller is implemented by a VLSI chip called the CMCTL. The CMCTL contains approximately 25,000 transistors in a 132-pin CERQUAD surface mount package. It supports up to 64 Mbytes of ECC memory, with a 450 ns cycle time for longword transfers and a 720 ns cycle time for quadword transfers. This memory resides on one to four MS650 memory modules, depending on the system configuration. The MS650 modules communicate with the KA650 through the MS650 memory interconnect, which utilizes the CD interconnect and a 50-pin ribbon cable.

1.7 MicroVAX System Support Functions

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

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

1.8 Resident Firmware

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

- Board initialization
- Power-up self-testing of the KA650 and MS650 modules
- Emulation of a subset of the VAX standard console (automatic/manual bootstrap, automatic/manual restart, and a simple command language for examining/altering the state of the processor)
- Booting from supported Q22-bus devices
- Multilingual capability

1.9 Q22-bus Interface

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

- A 16-entry map cache for the 8192-entry, main memory-resident scatter-gather map, used for translating 22-bit Q22-bus addresses into 26-bit main memory addresses
- Interrupt arbitration logic that recognizes Q22-bus interrupt requests BR7-BR4
- Q22-bus termination (240 Ω)

1.10 MS650-AA Memory Modules

The MS650-AA memory modules are 8 Mbyte, 450 ns, 39-bit wide arrays (32-bit data and 7-bit ECC) implemented with 256 Kbytes of dynamic RAMs in zig-zag in-line packages (ZIPs). MS650-AA memory modules are single, quad-height, Q22-bus modules, as shown in Figure 1–4.

1.11 MS650-BA Memory Modules

The MS650-BA memory modules are 16 Mbyte, 450 ns, 39-bit wide arrays (32-bit data and 7-bit ECC) implemented with 1 Mbyte dynamic RAMs in surface-mount packages. MS650-BA memory modules are single, quad-height, Q22-bus modules, as shown in Figure 1–4.

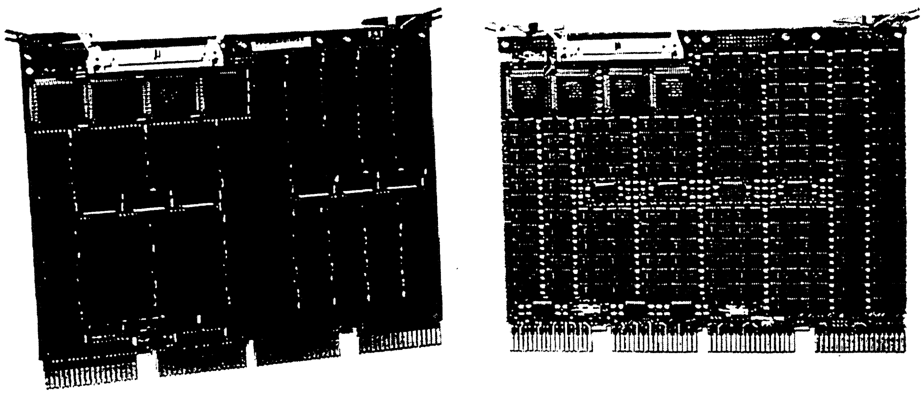


Figure 1–4 MS650-AA and MS650-BA Memory Modules

2
Installation and Configuration

This chapter describes how to install the KA650 in a system. The chapter discusses the following topics:

- Installing the KA650
- Configuring the KA650
- KA650 connectors
- CPU cover panel
- Configuration board
- Compatible system enclosures

2.1 Installing the KA650

The KA650 and MS650 modules must be installed in system enclosures having Q22-bus/CD backplane slots. These modules are not compatible with Q/Q backplane slots, and therefore should only be installed in Q22-bus/CD backplane slots.

The KA650 CPU module must be installed in slot 1 of the Q22-bus/CD backplane (Figure 2–1). MS650 memory modules must be installed in slots immediately adjacent to the CPU module. Up to four MS650 modules can be installed, occupying slots 2,3,4 and 5 respectively. A 50-pin ribbon cable is used to connect the KA650 CPU module and the MS650 memory module(s), as shown in Figure 2–2.

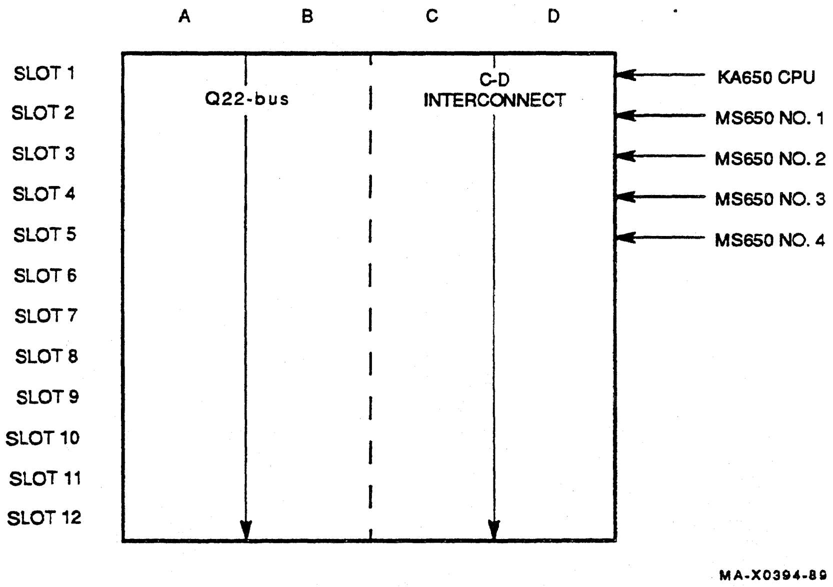


Figure 2–1 CPU and Memory Module Placement

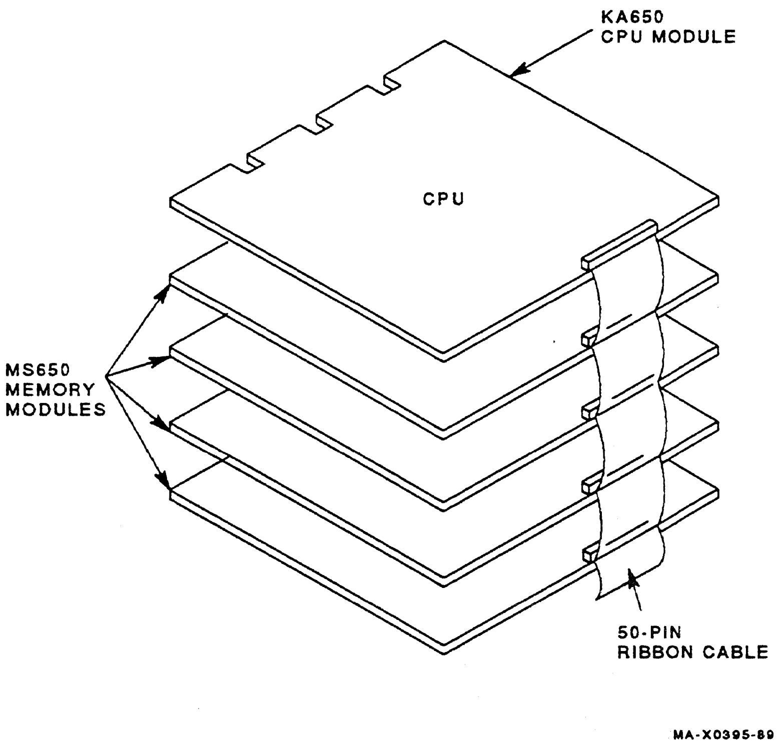


Figure 2–2 Cable Connections

2.2 Configuring the KA650

The following parameters must be configured on the KA650:

- Power-up mode
- Break enable switch
- Console serial line baud rate

These parameters are configured using either the H3600-SA CPU cover panel, or the KA630CNF configuration board (for servicing/troubleshooting and custom enclosures).

2.3 KA650 Connectors

The KA650 uses three connectors (J1, J2, and J3) and four rows of module fingers (A,B,C, and D) to communicate with the console device, main memory, and the Q22-bus. Users can configure the KA650 through the H3600-SA CPU cover panel or the KA630CNF configuration board. The slot pinouts on the fingers of the KA650 are listed in Appendix C.

The orientation of connectors J1, J2, and J3, and the LED indicators is shown in Figure 2–3.

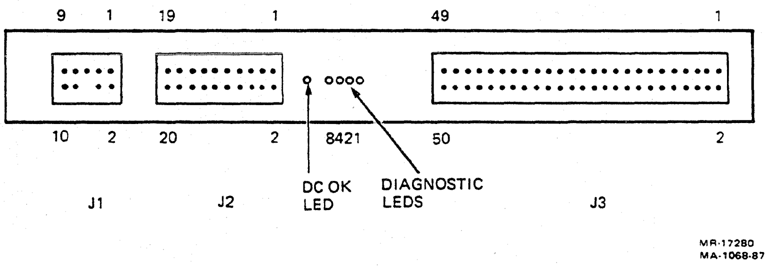


Figure 2–3 KA650 Pin and LED Orientation

2.3.1 Console SLU Connector (J1)

The 10-pin console SLU connector provides the connection between the KA650 and the console terminal. It is connected to the inside of the H3600-SA CPU cover panel by a 10-conductor cable, or directly to connector J3 of the KA630CNF configuration board. A cable from the outside of the H3600-SA CPU cover panel or J1 of the KA630CNF provides the external connection to the console terminal. Table 2–1 lists J1 pinouts.

Table 2–1 Console SLU Connector (J1) Pinouts

Pin	Signal	Meaning
01		Data terminal ready
02	GND	Ground
03	SLU OUT L	Console SLU output from the KA650
04	GND	Ground
05	GND	Ground
06		Key (no pin)
07	SLU IN +	Console SLU differential inputs to the
08	SLU IN -	KA650
09	GND	Ground
10	+12 V	Fused +12 volts

2.3.2 Configuration and Display Connector (J2)

The KA650 has no jumper or switch settings to change or set. The module is configured through switches on an H3600-SA CPU cover panel, or a KA630CNF configuration board. The 20-pin configuration and display connector is connected to the inside of the H3600-SA CPU cover panel by a 20-conductor cable, or directly to connector J2 of the KA630CNF configuration board. Table 2–2 lists J2 pinouts.

Table 2–2 Configuration and Display Connector (J2) Pinouts

Pin[1]	Signal	Meaning
01	GND	Ground
02	GND	Ground
03	GND	Ground
04	CPU CD0 L	CPU code <01:00>. This 2-bit code can be configured
05	CPU CD1 L	only by using switches 7 and 8 on the KA630CNF configuration board (Figure 2–7).
		CPU code <01:00> configuration
		00 Normal operation
		01 Reserved
		10 Reserved
		11 Reserved
		CPU code <01:00> is read by software from the BDR.

[1]The KA650 module has 4.7K ohm pull-up resistors for the 8 input signals (pins 4 and 5, 13 through 15, and 17 through 19).

Table 2–2 (Cont.) Configuration and Display Connector (J2) Pinouts

Pin[1]	Signal	Meaning
		If the CPU distribution panel insert is used, no connections are made to pins 4 and 5. In that case, signal levels are negated by pull-up resistors on the KA650.
06	GND	Ground
07	DSPL 00 L	Display register bits <03:00>. When asserted each of these four output signals lights a corresponding LED on the module.
08	DSPL 01 L	
09	DSPL 02 L	
11	DSPL 03 L	DSPL <03:00> are asserted (low) by power-up and by the negation of DCOK when the processor is halted. They are updated by boot and diagnostic programs from the BDR.
10	BTRY VCC	Battery backup voltage for TODR clock
12	GND	Ground
13	BDG CD0 L	Boot and diagnostic code <01:00>. This 2-bit code indicates power-up mode, and is read by software from the BDR.
14	BDG CD1 L	
15	BRK ENB L	Break enable. This input signal controls the response to an external halt condition. If BRK ENB is asserted (low), then the KA650 halts and enters the console program if any of the following occur: • The program executes a halt instruction in kernel mode • The console detects a break character • The Q22-bus halt line is asserted If BRK ENB is negated (high), then the halt line and break character are ignored and the ROM program responds to a halt instruction by restarting or rebooting the system. BRK ENB is read by software from the BDR.
16	GND	Ground
17	CSBR 02 L	Console baud rate <02:00>. These three bits are configured by using either the baud rate
18	CSBR 01 L	

[1]The KA650 module has 4.7K ohm pull-up resistors for the 8 input signals (pins 4 and 5, 13 through 15, and 17 through 19).

Table 2–2 (Cont.) Configuration and Display Connector (J2) Pinouts

Pin[1]	Signal	Meaning
19	CSBR 00 L	select switch on the H3600-SA CPU cover panel, or switches 2, 3, and 4 of the KA630CNF configuration board.
20	+5 V	Fused +5 volts

[1]The KA650 module has 4.7K ohm pull-up resistors for the 8 input signals (pins 4 and 5, 13 through 15, and 17 through 19).

2.3.3 Memory Expansion Connector (J3)

The 50-pin memory expansion connector provides the interface between the KA650 and MS650 memory modules installed in slots 2, 3, 4 and 5 of a Q22-bus backplane containing the CD interconnect. Table 2–3 lists J3 pinouts.

Table 2–3 Memory Expansion Connector (J3) Pinouts

Pin	Signal	Pin	Signal
01	GND	26	D MD10 H
02	D MD9 H	27	GND
03	D MD8 H	28	D MD29 H
04	D MD7 H	29	D MD28 H
05	GND	30	D MD27 H
06	D MD6 H	31	GND
07	D MD5 H	32	D MD26 H
08	D MD4 H	33	D MD25 H
09	D MD3 H	34	D MD24 H
10	GND	35	D MD23 H
11	D MD2 H	36	GND
12	D MD1 H	37	D MD22 H
13	D MD0 H	38	D MD21 H
14	D MD19 H	39	D MD20 H
15	GND	40	D MD38 H
16	D MD18 H	41	GND
17	D MD17 H	42	D MD37 H
18	D MD16 H	43	D MD36 H
19	D MD15 H	44	D MD35 H

Table 2–3 (Cont.) Memory Expansion Connector (J3) Pinouts

Pin	Signal	Pin	Signal
20	GND	45	D MD34 H
21	D MD14 H	46	GND
22	D MD13 H	47	D MD33 H
23	D MD12 H	48	D MD32 H
24	GND	49	D MD31 H
25	D MD11 H	50	D MD30 H

2.4 H3600-SA CPU Cover Panel

The H3600-SA CPU cover panel is a special I/O panel that is used in BA213 enclosures. A one-piece ribbon cable on the H3600-SA CPU cover panel plugs into the console SLU and baud rate connectors on the KA650. The H3600-SA CPU cover panel fits over backplane slots 1 and 2, covering both the KA650 CPU module and the first of four possible MS650 memory modules.

The H3600-SA CPU cover panel (Figure 2–4) includes the features and controls specified in Table 2–4.

Table 2–4 H3600-SA CPU Cover Panel Features and Controls

Outside	Inside
Modified modular jack (MMJ) SLU connector	Baud rate rotary switch
Power-up mode switch	Battery back-up unit (BBU) for TODR clock
Hex LED display	List of baud rate switch settings
Break enable switch	30-pin cable connector

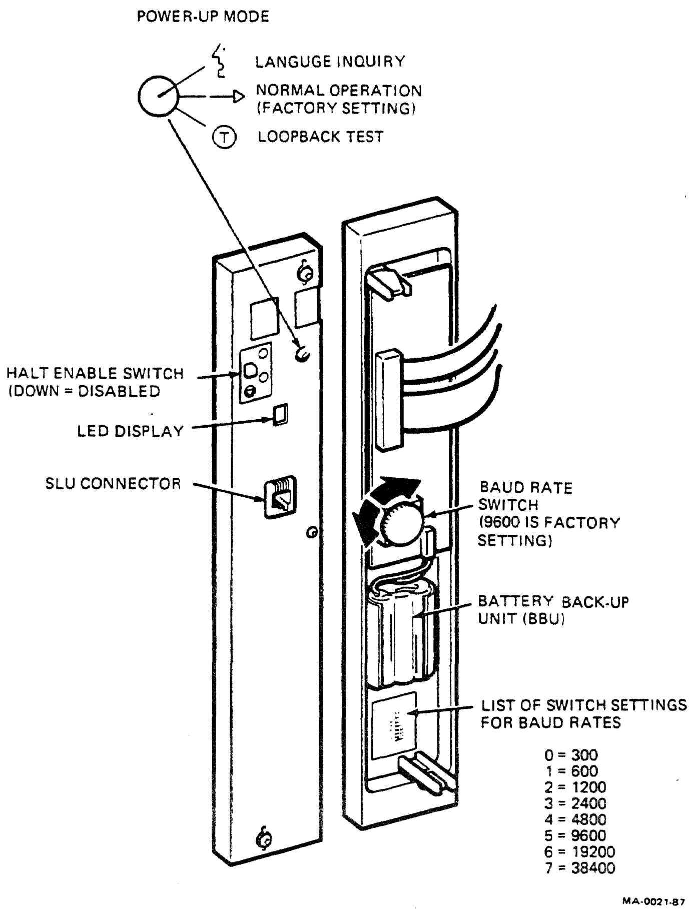

POWER-UP MODE
LANGUGE INQUIRY
NORMAL OPERATION (FACTORY SETTING)
LOOPBACK TEST
HALT ENABLE SWITCH (DOWN = DISABLED
LED DISPLAY
SLU CONNECTOR
BAUD RATE SWITCH (9600 IS FACTORY SETTING)
BATTERY BACK-UP UNIT (BBU)
LIST OF SWITCH SETTINGS FOR BAUD RATES
0 = 300
1 = 600
2 = 1200
3 = 2400
4 = 4800
5 = 9600
6 = 19200
7 = 38400
MA-0021-87

Figure 2–4 H3600-SA CPU Cover Panel

2.5 KA630CNF Configuration Board

A KA630CNF configuration board (H3263-00) (Figures 2–5, through 2–7) is provided with each KA650. The KA630CNF plugs directly into connectors J1 and J2 on the KA650. It allows the user to configure the KA650 by setting the 10 switches on SW1 as listed in Table 2–5.

Connector J1 is used to connect a cable to the console SLU. Connector J4 is for a BBU. The J4 pin closest to connector J1 is the positive pin.

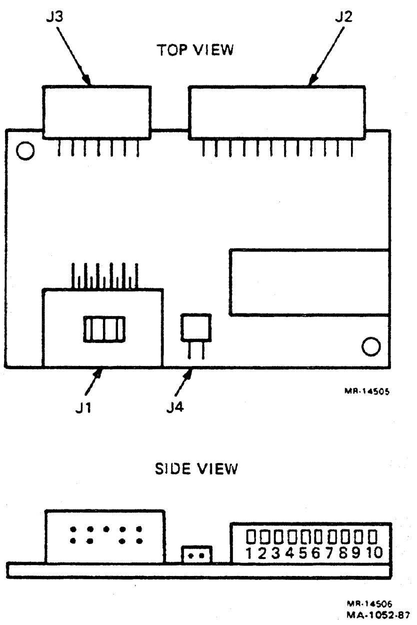


Figure 2–5 KA630CNF Configuration Board

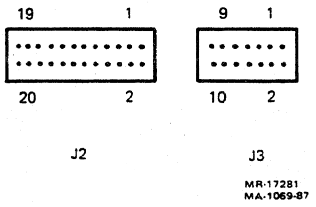


Figure 2–6 KA630CNF J2 and J3 Pin Orientation

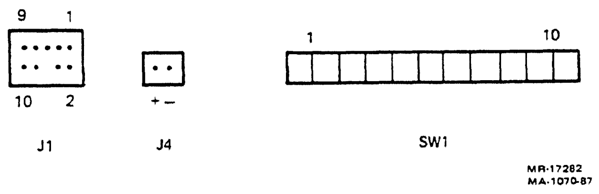


Figure 2–7 KA630CNF J1 and J4 Pin Orientation

Table 2–5 KA630CNF Switch Selections

Switch/Setting				Mode/Function
1				**Halt Mode**
Off				Disabled
On				Enabled
2	**3**	**4**		**Console Baud Rate**
Off	Off	Off		300
On	Off	Off		600
Off	On	Off		1,200
On	On	Off		2,400
Off	Off	On		4,800
On	Off	On		9,600
Off	On	On		19,200
On	On	On		38,400
5	**6**	**9**	**10**	**Power-Up Mode**[1]
Off	Off	On	Off	Normal operation. Transmit line connected. Receive line connected.
On	Off	On	Off	Language inquiry mode. Transmit line connected. Receive line connected.
Off	On	Off	On	Loopback test mode (maintenance). Transmit line connected to receive line and console.
On	On	On	Off	Manufacturing use only. Bypasses memory test.

[1]Do not use any other settings for switches 5, 6, 9, and 10.

Table 2–5 (Cont.) KA630CNF Switch Selections

Switch/Setting		Mode/Function
7	**8**	**CPU Operation Mode**
Off	Off	Normal operation
On	Off	Reserved
Off	On	Reserved
On	On	Reserved

Table 2–6 lists the pins on the KA650 J2 and J1, and the corresponding KA630CNF connectors and switches on SW1. Note that connectors J2 and J3 both have more connectors than there are pins on the corresponding KA650 connector. The two left and two right side connectors on J2 and J3 of the KA630CNF are unused. Switches 1 through 8 on SW1 set values that enable or disable halts; and determine CPU operation mode, power-up mode, and console baud rate. SW1 switches 9 and 10 connect transmit and receive lines as required for normal operation or loopback testing.

Table 2–6 KA630CNF Connector and Switches

CPU J2 Pin	Signal	CNF J2 Pin	CNF SW1 Switch	CNF J4 Pin
		1		
		2		
1	GND	3		
2	GND	4		
3	GND	5		
4	CPU CD0 L	6	7	
5	CPU CD1 L	7	8	
6	GND	8		
7	DSPL 00 L	9		
8	DSPL 01 L	10		
9	DSPL 02 L	11		
10	BTRY VCC	12		1[1]
11	DSPL 03 L	13		
12	GND	14		
13	BDG CD0 L	15	5	

[1]+5 V from BBU to TODR clock chip on CPU

Table 2–6 (Cont.) KA630CNF Connector and Switches

CPU J2 Pin	Signal	CNF J2 Pin	CNF SW1 Switch	CNF J4 Pin
14	BDG CD1 L	16	6	
15	BRK ENB L	17	1	
16	GND	18		
17	CSBR 02 L	19	2	
18	CSBR 01 L	20	3	
19	CSBR 00 L	21	4	
20	+5 V	22		
		23		
		24		

CPU J1 Pin	Signal	CNF J3 Pin	CNF SW1 Switch	CNF J1 Pin
		1		
		2		
1	DTR	3		
2	GND	4		2, 4, 5, 9
3	SLU OUT L	5	10	3
4	GND	6		2, 4, 5, 9
5	GND	7		2, 4, 5, 9
6	Key (no pin)	8		
7	SLU IN +	9		7
8	SLU IN -	10	9	
9	GND	11		2, 4, 5, 9
10	+12 V	12		10
		13		
		14		

2.6 Compatible System Enclosures

The KA650 is compatible with the following DIGITAL enclosures:

BA213

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

BA123-A

The BA123-A contains a 4 row by 12 slot backplane, with the Q22-bus implemented in the A/B rows of slots 1 through 12 (and the C/D rows of slots 5 through 12). The CD interconnect is implemented in the C/D rows of slots 1 through 4, allowing up to three memory modules to be used. The BA123-A has mounting space for up to five 13.2 cm (5.25 inches) mass storage devices. The BA123-A is equipped with a power supply that includes a master console and two regulators that each provide 36 A at +5 V and 7 A at +12 V. Total power from each regulator must not exceed 230 W.

BA23-A

The BA23-A contains a 4 row by 8 slot backplane, with the Q22-bus implemented in the A/B rows of slots 1 through 8 (and the C/D rows of slots 4 through 8). The CD interconnect is implemented in the C/D rows of slots 1 through 3, allowing up to two memory modules to be used. The BA23-A has mounting space for up to two 13.2 cm (5.25 inch) mass storage devices. The BA23-A is equipped with a power supply that includes a master console and provides 36 A at +5 V and 7 A at +12 V. Total power must not exceed 230 W.

The BA23-A is also available in an H9642 cabinet, which provides eight additional backplane slots and space for two 26.5 cm (10.5 inches) mass storage devices.

BA11-S

The BA11-S contains a 4 row by 9 slot backplane, with the Q22-bus implemented in the A/B rows of slots 1 through 9. The CD interconnect is implemented in the C/D rows of slots 1 through 9, allowing up to four memory modules to be used. The BA11-S is equipped with a power supply that includes a master console and provides 36 A at +5 V and 5 A at +12 V. Total power must not exceed 230 W.

3 Architecture

This chapter describes the KA650 registers, instruction set, and memory. The chapter covers the following KA650 topics:

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

3.1 Central Processor

The central processor of the KA650 supports the MicroVAX Chip subset (plus six additional string instructions) of the VAX instruction set and data types, and full VAX memory management. It is implemented by a single VLSI chip called the CVAX.

3.1.1 Processor State

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

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

3.1.1.1 General Purpose Registers

The KA650 implements 16 general purpose registers as specified in the *VAX Architecture Reference Manual*. These registers are used for temporary storage, as accumulators, and as base and index registers for addressing. These registers are denoted R0 through R15. The bits of a register are numbered from the right <0> through <31> (Figure 3–1).

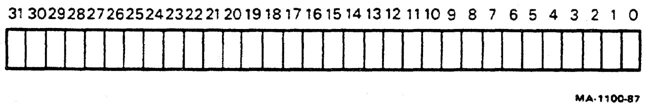


Figure 3–1 General Purpose Register Bit Map

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

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

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

3.1.1.2 Processor Status Longword

The KA650 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 bit map.

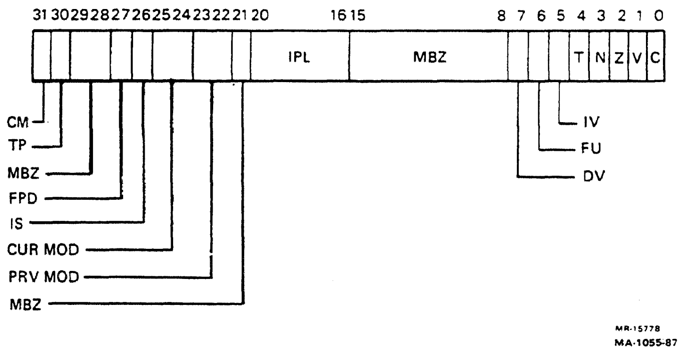


Figure 3–2 PSL Bit Map

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

Data Bit	Definition
<15:8>	Unused. Must be written as zero.
<7>	Decimal overflow trap enable (DV). Has no effect on KA650 hardware. Can be used by macrocode which emulates VAX decimal instructions.
<6>	Floating underflow fault enable (FU)
<5>	Integer overflow trap enable (IV)
<4>	Trace trap enable (T)
<3>	Negative condition code (N)
<2>	Zero condition code
<1>	Overflow condition code (V)
<0>	Carry condition code (C)

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

3.1.1.3 Internal Processor Registers

The KA650 internal processor registers (IPRs) can be accessed by using the MFPR and MTPR privileged instructions. Each IPR falls into one of the following seven categories:

1. Implemented by KA650 (in the CVAX chip) as specified in the *VAX Architecture Reference Manual*.
2. Implemented by KA650 (in the SSC) as specified in the *VAX Architecture Reference Manual*.
3. Implemented by KA650 (and all designs that use the CVAX chip) uniquely.

4. Implemented by KA650 (and all designs that use the SSC) uniquely.
5. Not implemented, timed out by the CDAL bus timer (in the SSC) after 4 µs. Read as 0. NOP on write.
6. Access not allowed; accesses result in a reserved operand fault.
7. Accessible, but not fully implemented. Accesses yield unpredictable results.

Refer to Table 3–1 for a listing of each of the KA650 IPRs, along with its mnemonic, its access type (read or write) and its category number.

Table 3–1 KA650 Internal Processor Registers

Decimal	Hex	Register	Mnemonic	Type	Category[1]
0	0	Kernel stack pointer	KSP	r/w	1
1	1	Executive stack pointer	ESP	r/w	1
2	2	Supervisor stack pointer	SSP	r/w	1
3	3	User stack pointer	USP	r/w	1
4	4	Interrupt stack pointer	ISP	r/w	1
7:5	7:5	Reserved			5
8	8	P0 base register	P0BR	r/w	1
9	9	P0 length register	P0LR	r/w	1
10	A	P1 base register	P1BR	r/w	1
11	B	P1 length register	P1LR	r/w	1
12	C	System base register	SBR	r/w	1
13	D	System length register	SLR	r/w	1
15:14	F:E	Reserved			5
16	10	Process control block base	PCBB	r/w	1
17	11	System control block base	SCBB	r/w	1
18	12	Interrupt priority level	IPL	r/w	1 I
19	13	AST level	ASTLVL	r/w	1 I
20	14	Software interrupt request	SIRR	w	1
21	15	Software interrupt summary	SISR	r/w	1 I
23:22	17:16	Reserved			5
24	18	Interval clock control/status	ICCS	r/w	3 I
25	19	Next interval count	NICR	w	5
26	1A	Interval count	ICR	r	5

[1]The I indicates that the register is initialized on power-up and by the negation of DCOK when the processor is halted.

Table 3–1 (Cont.) KA650 Internal Processor Registers

Decimal	Hex	Register	Mnemonic	Type	Category[1]
27	1B	Time-of-year clock register	TODR	r/w	2
28	1C	Console storage receiver status	CSRS	r/w	7 I
29	1D	Console storage receiver data	CSRD	r	7 I
30	1E	Console storage transmit status	CSTS	r/w	7 I
31	1F	Console storage transmit data	CSTD	w	7 I
32	20	Console receiver control/status	RXCS	r/w	4 I
33	21	Console receiver data buffer	RXDB	r	4 I
34	22	Console transmit control/status	TXCS	r/w	4 I
35	23	Console transmit data buffer	TXDB	w	4 I
36	24	Translation buffer disable	TBDR	r/w	5
37	25	Cache disable	CADR	r/w	3 I
38	26	Machine check error summary	MCESR	r/w	5
39	27	Memory system error	MSER	r/w	3 I
41:40	29:28	Reserved			5
42	2A	Console saved PC	SAVPC	r	3
43	2B	Console saved PSL	SAVPSL	r	3
47:44	2F:2C	Reserved			5
48	30	SBI Fault/status	SBIFS	r/w	5
49	31	SBI silo	SBIS	r	5
50	32	SBI silo comparator	SBISC	r/w	5
51	33	SBI maintenance	SBIMT	r/w	5
52	34	SBI error register	SBIER	r/w	5
53	35	SBI timeout address register	SBITA	r	5
54	36	SBI quadword clear	SBIQC	w	5
55	37	I/O bus reset	IORESET	w	4

[1]The I indicates that the register is initialized on power-up and by the negation of DCOK when the processor is halted.

Table 3–1 (Cont.) KA650 Internal Processor Registers

Decimal	Hex	Register	Mnemonic	Type	Category[1]
56	38	Memory management enable	MAPEN	r/w	1
57	39	TB invalidate all	TBIA	w	1
58	3A	TB invalidate single	TBIS	w	1
59	3B	TB data	TBDATA	r/w	5
60	3C	Microprogram break	MBRK	r/w	5
61	3D	Performance monitor enable	PMR	r/w	5
62	3E	System identification	SID	r	1
63	3F	Translation buffer check	TBCHK	w	1
64:127	40:7F	Reserved			6

[1]The I indicates that the register is initialized on power-up and by the negation of DCOK when the processor is halted.

VAX Standard Internal Processor Registers

Internal processor registers (IPRs) that are implemented as specified in the *VAX Architecture Reference Manual* are classified as category 1 IPRs. The *VAX Architecture Reference Manual* should be consulted for details on the operation and use of these registers. The category 1 registers listed in Table 3–2 are also referenced in other sections of this manual.

Table 3–2 Category One IPRs

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

Table 3–2 (Cont.) Category One IPRs

Number	Register	Mnemonic	Section
58	Translation buffer invalidate single	TBIS	3.1.4.2
62	System identification	SID	3.1.6
63	Translation buffer check	TBCHK	3.1.4.2

Unique Internal Processor Registers

Internal processor registers (IPRs) that are implemented uniquely on the KA650 (for example, those that are not contained in, or do not fully conform to the standards in the *VAX Architecture Reference Manual*) are classified as category 2 IPRs and are described in detail in this manual. Refer to the sections listed in Table 3–3 for a description of these registers.

Table 3–3 Category Two IPRs

Number	Register	Mnemonic	Section
24	Interval clock control/status	ICCS	3.6.2
32	Console receiver control/status	RXCS	3.5.1.1
33	Console receiver data buffer	RXDB	3.5.1.2
34	Console transmit control/status	TXCS	3.5.1.3
35	Console transmit data buffer	TXDB	3.5.1.4
37	Cache disable	CADR	3.3.2.5
39	Memory system error	MSER	3.3.2.6
42	Console saved PC	SAVPC	3.1.5
43	Console saved PSL	SAVPSL	3.1.5
55	I/O bus reset	IORESET	3.7.5.1

3.1.2 Data Types

The KA650 CPU supports the following subset of the VAX data types:

- Byte
- Word
- Longword

- Quadword
- Character string
- Variable length bit field

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

3.1.3 Instruction Set

The KA650 CPU implements the following subset of the VAX instruction set types in microcode:

- Integer arithmetic and logical
- Address
- Variable length bit field
- Control
- Procedure call
- Miscellaneous
- Queue
- Character string moves (MOVC3, MOVC5, CMPC3[1] , CMPC5[1], LOCC[1], SCANC[1], SKPC[1], and SPANC[1])
- Operating system support
- F_floating
- G_floating
- D_floating

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

The KA650 CVAX chip provides special microcode assistance to aid the macrocode emulation of the following instruction groups:

- Character string (except MOVC3, MOVC5, CMPC3[1], CMPC5[1], LOCC[1], SCANC[1], SKPC[1], and SPANC[1])
- Decimal string
- CRC
- EDITPC

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

- Octaword
- Compatibility mode instructions

3.1.4 Memory Management

The KA650 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).

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

3.1.4.1 Translation Buffer

To reduce the overhead associated with translating virtual addresses to physical addresses, the KA650 employs a 28-entry, fully associative, translation buffer for caching VAX PTEs in modified form. Each entry can store a modified PTE for translating virtual addresses in either the VAX process space, or VAX system space. The translation buffer is flushed whenever memory management is enabled or disabled (for example, by writes to IPR 56), any page table base or length registers are modified (for example, by writes to IPRs 8 to 13) and by writing to IPR 57 (TBIA) or IPR 58 (TBIS).

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

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

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

3.1.4.2 Memory Management Control Registers

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

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

NOTE
Whenever software changes a valid 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 a MFPR instruction from any of these registers is undefined.

3.1.5 Exceptions and Interrupts

Both exceptions and interrupts divert execution from the normal flow of control. An exception is caused by the execution of the current instruction and is typically handled by the current process (for example, an arithmetic overflow), while 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).

3.1.5.1 Interrupts

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

Nonmaskable interrupts cause a halt through the hardware halt procedure which saves the PC, PSL, MAPEN<0> and a halt code in IPRs, raises the processor IPL to 1F, and then passes control to the resident firmware. The firmware dispatches the interrupt to the appropriate service routine based on the halt code and hardware event indicators. 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 since this halt code is generated after a hardware reset.

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

The various interrupt conditions for the KA650 are listed in Table 3–4 along with their associated priority levels and SCB offsets.

Table 3–4 Interrupts

Priority Level	Interrupt Condition	SCB Offset
Nonmaskable	BDCOK and BPOK negated then asserted on Q22-bus (power-up)	[1]
	BDCOK negated then asserted while BPOK asserted on Q22-bus (SCR<7> clear)	
	BDCOK negated then asserted while BPOK asserted on Q22-bus (SCR<7> set)	[2]
	BHALT asserted on Q22-bus	[2]
	BREAK generated by the console device	[2]
1F	Unused	
1E	BPOK negated on Q22-bus	0C
1D	CDAL bus parity error	60
	Q22-bus NXM on a write	60
	CDAL bus timeout during DMA	60
	Main memory NXM errors	60
	Uncorrectable main memory errors	60
1C - 1B	Unused	
1A	Second-level cache tag parity errors	54
	Correctable main memory errors	54
19 - 18	Unused	
17	BR7 L asserted	Q22-bus vector plus 200_{16}
16	Interval timer interrupt	C0
	BR6 L asserted	Q22-bus vector plus 200_{16}
15	BR5 L asserted	Q22-bus vector plus 200_{16}

[1]These conditions generate a hardware halt procedure with a halt code of 3 (hardware reset).

[2]These conditions generate a hardware halt procedure with a halt code of 2 (external halt).

Table 3–4 (Cont.) Interrupts

Priority Level	Interrupt Condition	SCB Offset
14	Console terminal	F8,F6
	Programmable timers	78,7C
	BR4 L asserted	Q22-bus vector plus 200 $_{16}$
13 through 10	Unused	
0F through 01	Software interrupt requests	84-BC

NOTE
Because the Q22-bus does not allow differentiation between the four bus grant levels (for example, a level 7 device could respond to a level 4 bus grant), the KA650 CPU raises the IPL to 17 after responding to interrupts generated by the assertion of either BR7 L, BR6 L, or BR4 L. The KA650 maintains the IPL at the priority of the interrupt for all other interrupts.

The interrupt system is controlled by three IPRs: IPR 18, the interrupt priority level register (IPL), IPR 20, the software interrupt request register (SIRR), and IPR 21, the software interrupt summary register (SISR). The IPL is used for loading the processor priority field in the PSL (bits <20:16>). The SIRR is used for creating software interrupt requests. The SISR records pending software interrupt requests at levels 1 through 15. The format of these registers is shown in Figure 3–3. Refer to the *VAX Architecture Reference Manual* for more information on these registers.

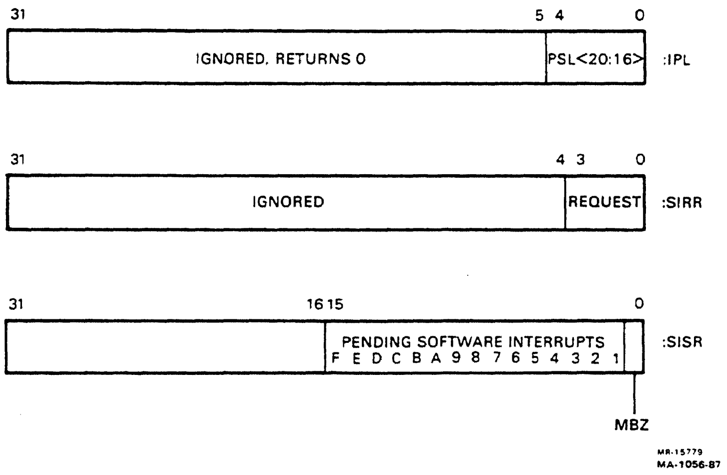


Figure 3–3 Interrupt Registers

3.1.5.2 Exceptions

Exceptions can be divided into three types:

- Trap
- Fault
- Abort

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

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

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

Exceptions are grouped into six classes:

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

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

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

Table 3–5 Exceptions

	Type	SCB Offset
Integer overflow	Trap	34
Integer divide-by-zero	Trap	34
Subscript range	Trap	34
Floating overflow	Fault	34
Floating divide-by-zero	Fault	34
Floating underflow	Fault	34
Memory Management Exceptions		
Access control violation	Fault	20
Translation not valid	Fault	24
Operand Reference Exceptions		
Reserved addressing mode	Fault	1C
Reserved operand fault	Abort	18
Instruction Execution Exceptions		
Reserved/privileged instruction	Fault	10
Emulated instruction	Fault	C8, CC
Change mode	Trap	40-4C
Breakpoint	Fault	2C
Tracing Exception		
Trace	Fault	28
System Failure Exceptions		
Interrupt stack not valid	Abort	[1]
Kernel stack not valid	Abort	08
Machine check	Abort	04
CDAL bus parity errors		
First-level cache parity errors		

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

Table 3–5 (Cont.) Exceptions

	Type	SCB Offset
Second-level cache data parity errors		
Q22-bus NXM errors		
Q22-bus device parity errors		
Q22-bus no grant errors		
CDAL bus timeout errors		
Main memory NXM errors		
Main memory uncorrectable errors		

3.1.5.3 Information Saved on a Machine Check Exception

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

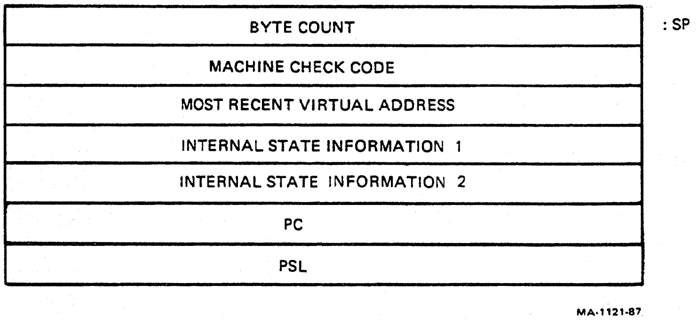

BYTE COUNT
: SP
MACHINE CHECK CODE
MOST RECENT VIRTUAL ADDRESS
INTERNAL STATE INFORMATION 1
INTERNAL STATE INFORMATION 2
PC
PSL
MA-1121-87

Figure 3–4 Information Saved on a Machine Check Exception

Figure 3–4 is explained in the following paragraphs.

Byte Count

Byte count <31:0> indicates the number of bytes of information that follow on the stack (not including the PC and PSL).

Machine Check Code Parameter

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

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

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

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

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

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

Hex Code	Error Description
9	A hardware interrupt was requested at an unused interrupt priority level (IPL).

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

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

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

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

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

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

Most Recent Virtual Address Parameter

Most recent virtual address <31:0> captures the contents of the virtual address pointer register at the time of the machine check. If a machine check other than a machine check 81 occurs on a read operation, this field represents the virtual address of the location that is being read when the error occurs, plus four. If machine check 81 occurs, this field represents the physical address of the location that is being read when the error occurs, plus four.

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

Internal State Information 1 Parameter

Internal state information 1 is divided into four fields. The contents of these fields is described as follows:

- <31:24> captures the opcode of the instruction that was being read or executed at the time of the machine check.
- <23:16> captures the internal state of the CVAX CPU chip at the time of the machine check. The four most significant bits are equal to <1111> and the four least significant bits contain highest priority software interrupt <3:0>.

- <15:8> captures the state of CADR<7:0> at the time of the machine check. See Section 3.3.2.5 for an interpretation of the contents of this register.
- <7:0> captures the state of the MSER<7:0> at the time of the machine check. See Section 3.3.2.6 for an interpretation of the contents of this register.

Internal State Information 2

Internal state information 2 is divided into five fields. The contents of these fields is described as follows:

- <31:24> captures the internal state of the CVAX CPU chip at the time of the machine check. This field contains SC register <7:0>.
- <23:16> captures the internal state of the CVAX CPU chip at the time of the machine check. The two most significant bits are equal to 11 (binary) and the six least significant bits contain state flags <5:0>.
- <15> captures the state of the VAX can't restart flag at the time of the machine check.
- <14:8> captures the internal state of the CVAX CPU chip at the time of the machine check. The three most significant bits are equal to 111 (binary) and the four least significant bits contain ALU condition codes.
- <7:0> captures the offset between the virtual address of the start of the instruction being executed at the time of the machine check (saved PC) and the virtual address of the location being accessed (PC) at the time of the machine check.

PC

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

PSL

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

3.1.5.4 System Control Block

The system control block (SCB) consists of two pages in main memory that contain the vectors by which interrupts and exceptions are dispatched to the appropriate service routines. The SCB is pointed to by IPR 17, the system control block base register (SCBB), represented in Figure 1–5. The system control block format is presented in Table 1–6.

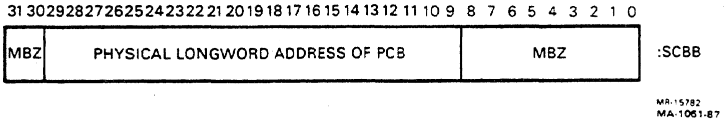


Figure 3–5 System Control Block Base Register

Table 3–6 System Control Block Format

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

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

SCB Offset	Interrupt/Exception Name	Type	Parameter	Notes
2C	Breakpoint instruction	Fault	0	
30	Unused			Compatibility mode in other VAXes
34	Arithmetic	Trap/fault	1	Parameter is type code
38:3C	Unused			
40	CHMK	Trap	1	Parameter is sign-extended operand word
44	CHME	Trap	1	Parameter is sign-extended operand word
48	CHMS	Trap	1	Parameter is sign-extended operand word
4C	CHMU	Trap	1	Parameter is sign-extended operand word
50	Unused			
54	Corrected read data	Interrupt	0	IPL is 1A (CRD L)
58:5C	Unused			
60	Memory error	Interrupt	0	IPL is 1D (MEMERR L)
64:6C	Unused			
78	Programmable timer 0	Interrupt	0	IPL is 14
7C	Programmable timer 1	Interrupt	0	IPL is 14
80	Unused			
84	Software level 1	Interrupt	0	
88	Software level 2	Interrupt	0	Ordinarily used for AST delivery
8C	Software level 3	Interrupt	0	Ordinarily used for process scheduling
90:BC	Software levels 4-15	Interrupt	0	
C0	Interval timer	Interrupt	0	IPL is 16 (INTTIM L)

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

SCB Offset	Interrupt/Exception Name	Type	Parameter	Notes
C4	Unused			
C8	Emulation start	Fault	10	Same mode exception, FPD=0; parameters are opcode, PC, specifiers
CC	Emulation continue	Fault	0	Same mode exception, FPD=1: no parameters
D0:DC	Unused			
E0:EC	Reserved for customer or CSS use			
F0:F4	Unused			Console storage registers on 11/750 and 11/730
F8	Console receiver	Interrupt	0	IPL is 14
FC	Console transmitter	Interrupt	0	IPL is 14
100:1FC	Adapter vectors	Interrupt	0	Not implemented by the KA650
200:3FC	Device vectors	Interrupt	0	Correspond to Q22-bus vectors 000:1FC; KA650 appends the assertion of bit <9,0>
400:FFC	Unused	Interrupt	0	

3.1.5.5 Hardware Detected Errors

The KA650 is capable of detecting thirteen types of error conditions during program execution.

- CDAL bus parity errors indicated by MSER<6> (on a read) or MEMCSR16<7> (on a write) being set. (This error cannot be distinguished if detected during a read reference.)
- First-level cache tag parity errors indicated by MSER<0> being set.
- First-level cache data parity errors indicated by MSER<1> being set.
- Second-level cache tag parity errors indicated by CACR<5> being set.

- Second-level cache data parity errors indicated by MSER<6> being set. (This error cannot be distinguished if detected during a read reference.)
- Q22-bus NXM errors indicated by DSER<7> being set.
- Q22-bus no sack errors (no indicator).
- Q22-bus no grant errors indicated by DSER<2> being set.
- Q22-bus device parity errors indicated by DSER<5> being set.
- CDAL bus timeout errors indicated by DSER<4> (only on DMA) being set.
- Main memory NXM errors indicated by DSER<0> (only on DMA) being set.
- Main memory correctable errors indicated by MEMCSR16<29> being set.
- Main memory uncorrectable errors indicated by MEMCSR16<31> and DSER<4> (only on DMA) being set.

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

3.1.5.6 Hardware Halt Procedure

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

Register	New Contents
SAVPC	Saved PC
SAVPSL<31:16, 7:0>	Saved PSL<31:16,7:0>
SAVPSL<15>	Saved MAPEN<0>
SAVPSL<14>	Valid PSL flag (unknown for halt code of 3)
SAVPSL<13:8>	Saved restart code

Register	New Contents
SP	Current interrupt stack
PSL	$041F\ 0000_{16}$
PC	$2004\ 0000_{16}$
MAPEN	0
ICCS	0 (for a halt code of 3)
MSER	0 (for a halt code of 3)
CADR	0 (for a halt code of 3, first-level cache is also flushed)
SISR	0 (for a halt code of 3)
ASTLVL	0 (for a halt code of 3)
All else	Undefined

The firmware uses the halt code in combination with any hardware event indicators to dispatch the execution or interrupt that caused the halt to the appropriate firmware routine (either console emulation, power-up, reboot, or restart). Table 3–7 and Table 3–8 list the interrupts and exceptions that can cause halts along with their corresponding halt codes and event indicators.

Table 3–7 Unmaskable Interrupts that can Cause a Halt

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

Table 3–8 Exceptions that can Cause a Halt

Halt Code	Exception Condition
6	Halt instruction executed in kernel mode.
Exceptions While Servicing an Interrupt or Exception	
4	Interrupt stack not valid during exception.
5	Machine check during normal exception.
7	SCB vector bits<1:0> = 11.
8	SCB vector bits<1:0> = 10.
A	CHMx executed while on interrupt stack.
B	CHMx executed to the interrupt stack.
10	ACV or TNV during machine check exception.
11	ACV or TNV during kernel stack not valid exception.
12	Machine check during machine check exception.
13	Machine check during kernel stack not valid exception.
19	PSL<26:24> = 101 during interrupt or exception.
1A	PSL<26:24> = 110 during interrupt or exception.
1B	PSL<26:24> = 111 during interrupt or exception.
1D	PSL<26:24> = 101 during REI.
1E	PSL<26:24> = 110 during REI.
1F	PSL<26:24> = 111 during REI.

3.1.6 System Identification

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

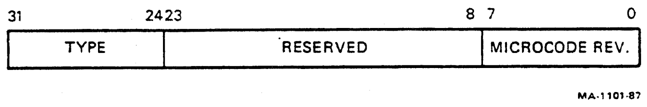


Figure 3–6 System Identification Register

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

In order to distinguish between different CPU implementations that use the same CPU chip, the KA650, as must all VAX processors that use the CVAX chip, implements a MicroVAX system type register (SYS_TYPE) at physical address 2004 0004 $_{16}$. This 32-bit read-only register is implemented in the KA650 ROM. The format of this register is shown in Figure 3–7.

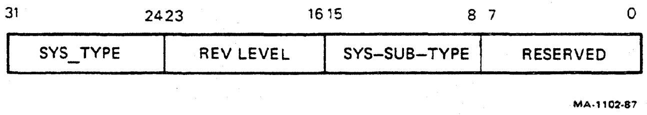


Figure 3–7 System Type Register

Data Bit	Definition
<31:24>	System type code (SYS_TYPE). This field reads as 01 $_{16}$ for all single-processor Q22-bus based systems.
<23:16>	Revision level (REV LEVEL). This field reflects the revision level of the KA650 firmware.
<15:8>	System subtype code (SYS_SUB_TYPE). This field reads as 01 $_{16}$ for the KA650.
<7:0>	Reserved for future use.

3.1.7 CPU References

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

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

3.1.7.1 Instruction-Stream Read References

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

3.1.7.2 Data-Stream Read References

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

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

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

3.1.7.3 Write References

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

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

3.2 Floating-Point Accelerator

The floating-point accelerator is implemented through a single VLSI chip called the CFPA.

3.2.1 Floating-Point Accelerator Instructions

The floating-point accelerator processes F_floating, D_floating, and G_ floating format instructions and accelerates the execution of MULL, DIVL, and EMUL integer instructions.

3.2.2 Floating-Point Accelerator Data Types

The KA650 floating-point accelerator supports byte, word, longword, quadword, F_floating, D_floating, and G_floating data types. The H_ floating data type is not supported, but may be implemented by macrocode emulation.

3.3 Cache Memory

To maximize CPU performance, the KA650 incorporates a two-level cache design. The first-level cache is implemented within the CVAX chip. The second-level cache is implemented using 16K by 4-bit static RAMs.

3.3.1 Cacheable References

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

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

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

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

The second-level cache only stores references on the CDAL bus that are part of a quadword transfer. Since quadword transfers on the CDAL bus can only be generated on cacheable references, the second-level cache is automatically configured to store the same type(s) of references as the first-level cache.

3.3.2 First-Level Cache

The KA650 includes a 1 Kbyte, two-way associative, write through first-level cache with a 90 ns cycle time. CPU read references access one longword at a time, while CPU writes can access one byte at a time. A single parity bit is generated, stored, and checked for each byte of data and each tag. The first-level cache can be enabled/disabled by setting/clearing the appropriate bits in the CADR. The first-level cache is flushed by any write to the CADR, as long as it is not in diagnostic mode.

3.3.2.1 First-Level Cache Organization

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

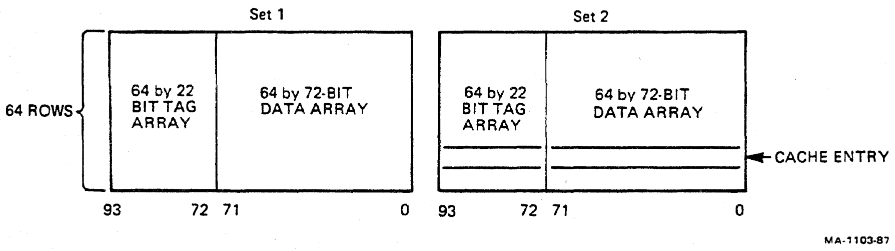


Figure 3–8 First-Level Cache Organization

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

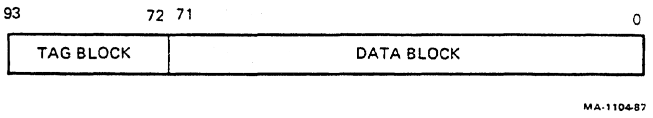


Figure 3–9 First-Level Cache Entry

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

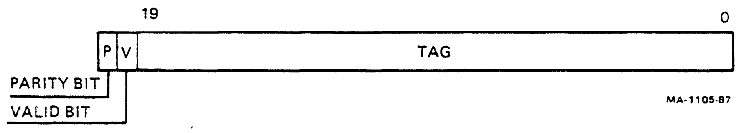


Figure 3–10 First-Level Cache Tag Block

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

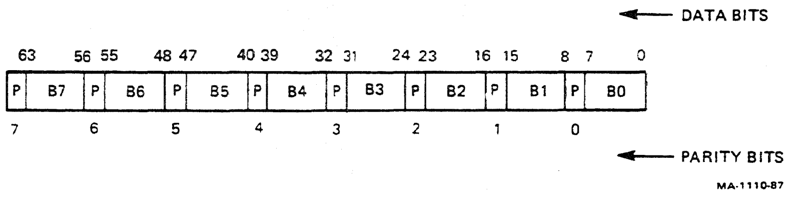


Figure 3–11 First-Level Cache Data Block

3.3.2.2 First-Level Cache Address Translation

Whenever the CPU requires an instruction or data, the contents of the first-level cache is checked to determine if the referenced location is stored there. The cache contents is checked by translating the physical address as follows:

- On noncacheable references, the reference is never stored in the cache, so a first-level cache miss occurs and a single longword reference is generated on the CDAL bus.
- On cacheable references, the physical address must be translated to determine if the contents of the referenced location is resident in the cache. The cache index field, bits <8:3> of the physical address, is used to select one of the 64 rows of the cache, with each row containing a single entry from each set. The cache tag field, bits <28:9> of the physical address, is then compared to the tag block of the entry from both sets in the selected row.

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

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

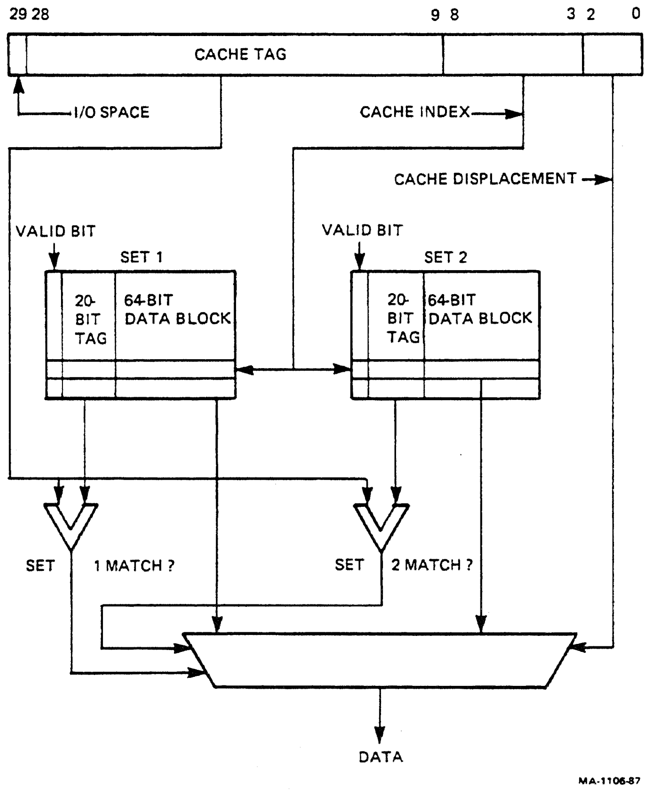


Figure 3–12 First-Level Cache Address Translation

3.3.2.3 First-Level Cache Data Block Allocation

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

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

Since the KA650 supports 64 Mbytes (8 M quadwords) of physical memory, up to 128K quadwords share each row (two data blocks) of the cache. Contiguous programs larger than 512 bytes or any noncontiguous programs separated by 512 bytes have a 50 percent chance of overwriting themselves when cache data blocks are allocated for the first time for data separated by 512 bytes (one page). After six allocations, there is a 97 percent probability both sets in a row will be filled.

3.3.2.4 First-Level Cache Behavior on Writes

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

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

3.3.2.5 Cache Disable Register

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

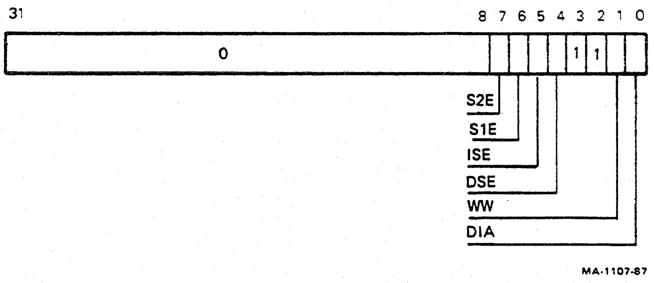


Figure 3–13 Cache Disable Register

Data Bit	Definition
<31:8>	Unused. Always read as zeros. Writes have no effect.
<7:6>	These bits are used to selectively enable or disable each set within the cache.
<7>	S2E. Read/write. When set, set 2 of the cache is enabled. When cleared, set 2 of the cache is disabled. Cleared on power-up and by the negation of DCOK when the processor is halted.
<6>	S1E. Read/write. When set, set 1 of the cache is enabled. When cleared, set 1 of the cache is disabled. Cleared on power-up and by the negation of DCOK when the processor is halted.
<5:4>	These bits are used to selectively enable or disable storing I-stream and D-stream references in the cache.
<5>	ISE. Read/write. When set, I-stream, memory space references are stored in both the first and second-level caches, if they are enabled. When cleared, I-stream memory references are not stored in either cache. Cleared on power-up and by the negation of DCOK when the processor is halted.
<4>	DSE. Read/write. When set, D-stream, memory space references are stored in both the first and second-level caches, if they are enabled. When cleared, D-stream memory references are not stored in either cache. Cleared on power-up and by the negation of DCOK when the processor is halted.

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

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

In I-stream only mode, the entire first-level cache is flushed whenever an REI instruction is executed. *The VAX Architecture Reference Manual* states that an REI instruction must be executed before executing instructions out of a page of memory that has been updated, whereas in the other two modes of operation, cache entries are invalidated on an individual basis, only if a DMA write operation results in a cache *hit*.

Data Bit	Definition
<3:2>	Unused. Always read as 1s.
<1>	Write wrong parity (WWP). Read/write. When set, incorrect parity is stored in the first-level cache whenever it is written. When cleared, correct parity is stored in the cache whenever the cache is written. Cleared on power-up and by the negation of DCOK when the processor is halted.
<0>	Diagnostic mode (DIA). Read/write. When cleared, the cache is in normal operating mode and writes to the CADR cause the first-level cache to be flushed, (all valid bits set to the invalid state) and the first-level cache is configured for write-through operation. When set, the first-level cache is in diagnostic mode and writes to the CADR will not cause the first-level cache to be flushed.

CPU write references with a longword destination (for example, MOVL) write the data into main memory (if it exists) as well as invalidate the corresponding cache entry irrespective of whether or not a cache hit occurred. CPU write references with a quadword destination (for example, MOVQ) write the data into main memory (if it exists) as well as cause the second longword of the quadword to be written into the longword of the cache data array that corresponds to the address of the first longword of the destination, irrespective of whether or not a cache hit occurred.

The data in the longword of the cache data array that corresponds to the address of the second longword of the destination remains unaltered. In addition, errors generated during write references, that would normally cause a machine check, are ignored (they do not cause a machine check trap to be generated, or prevent data from being stored in the cache).

Diagnostic mode is intended to allow the first-level cache tag store to be fully tested without requiring 512 Mbytes of main memory. This mode makes it possible for the tag block in a particular cache entry to be written with any pattern by executing a MOVQ instruction with bits <28:9> of the destination address equal to the desired pattern.

Two MOVQ instructions, one with a quadword aligned destination address and one with the next longword aligned destination address, are required to write to both longwords in the data block of a cache entry. Diagnostic mode does not affect read references. Cleared on power-up and by the negation of DCOK when the processor is halted.

NOTE
At least one read reference must occur between all write references made in diagnostic mode. Diagnostic mode should only be selected when one and only one of the two sets are enabled. Operation of this mode with both sets enabled or both sets disabled yields unpredictable results.

3.3.2.6 Memory System Error Register

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

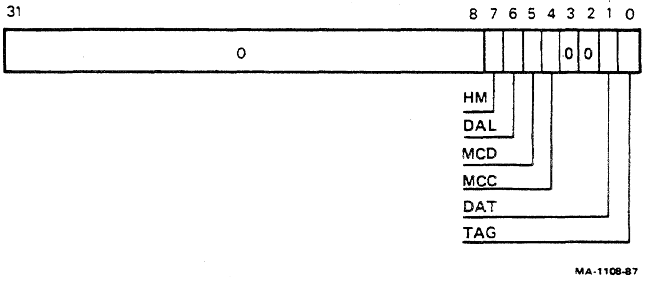


Figure 3–14 Memory System Error Register

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

3.3.2.7 First-Level Cache Error Detection

Both the tag and data arrays in the first-level cache are protected by parity. Each 8-bit byte of data and the 20-bit tag is stored with an associated parity bit. The valid bit in the tag is not covered by parity. Odd data bytes are stored with odd parity and even data bytes are stored with even parity. The tag is stored with odd parity.

The stored parity is valid only when the valid bit associated with the first-level cache entry is set. Tag and data parity (on the entire longword) are checked on read references that hit the first-level cache, while only tag parity is checked on CPU and DMA write references that hit the first-level cache.

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

- During a demand D-stream read reference, the entire first-level cache is flushed, the CADR is cleared (which disables the first level cache and causes the second-level cache to ignore all read operations). The cause of the error is logged in MSER<4,3:0> and a machine check abort is initiated.
- During a request I-stream read reference, the entire first-level cache is flushed (unless CADR<0> is set), the cause of the error is logged in MSER<1:0>, the prefetch is halted, but no machine check abort occurs, and both caches remain enabled.
- During a masked or unmasked write reference, the entire first-level cache is flushed (unless CADR<0> is set), the cause of the error is logged in MSER<0> (only tag parity is checked on CPU writes that hit the first-level cache), there is no effect on CPU execution, and both caches remain enabled.
- During a DMA write reference the cause of the error is logged in MSER<0> (only tag parity is checked on DMA writes that hit the first-level cache), there is no effect on CPU execution, both caches remain enabled, and no invalidate operation occurs.

3.3.3 Second-Level Cache

The KA650 also includes a 64 Kbyte, direct mapped, write through, second-level cache with a 180 ns cycle time for longword transfers and 270 ns cycle time for quadword transfers. CPU read references access one longword at a time. Cacheable references that miss the first-level cache can access up to one quadword at a time, while CPU writes can access a single byte at a time.

A single parity bit is generated, stored and checked for each tag. The cache does not generate or check parity on each data byte. The parity bits stored with each data byte are taken from the CDAL parity lines when a data block is written or allocated.

On second-level cache hits, these parity bits are placed back on the CDAL parity lines, so that parity checking on the data bytes is performed by the CVAX chip. This makes second-level cache data parity errors appear as CDAL bus parity errors.

The second-level cache can be enabled/disabled by setting/clearing the appropriate bit in the CACR. The second-level cache can be flushed by writing any value to each entry in the cache diagnostic space, as long as it is not in diagnostic mode.

3.3.3.1 Second-Level Cache Organization

The second-level cache, being direct mapped, consists of a single storage array called set 1. This array contains an 8K row by 12-bit tag array and an 8K row by 72-bit data array (Figure 3–15).

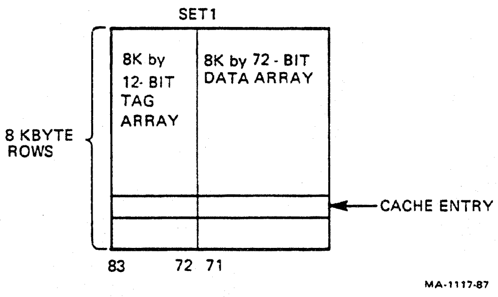


Figure 3–15 Second-Level Cache Organization

A row within the set corresponds to a single cache entry, so there are 8K entries in the entire cache. Each entry contains a 12-bit tag block and a 72-bit (eight-byte) data block. A cache entry is organized as shown in Figure 3–16.

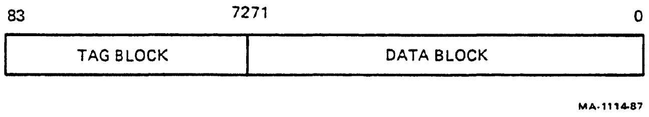


Figure 3–16 Second-Level Cache Entry

A tag block consists of a parity bit, a valid bit, and a 10-bit tag. A tag block is organized as shown in Figure 3–17.

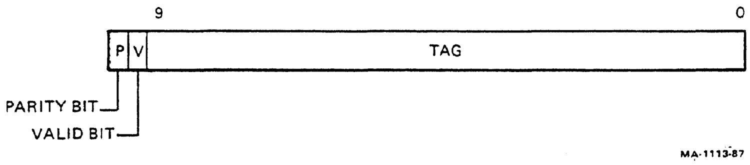


Figure 3–17 Second-Level Cache Tag Block

A data block consists of eight bytes of data, each with an associated parity bit. The total data capacity of the cache is 8K eight-byte blocks, or 64 Kbytes. A data block is organized as shown in Figure 3–18.

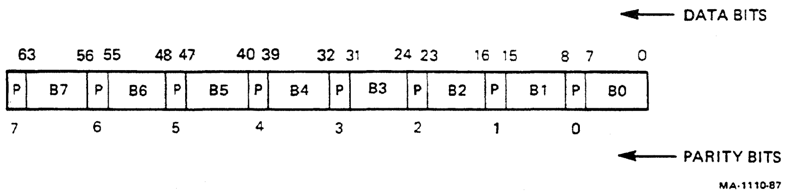


Figure 3–18 Second-Level Cache Data Block

3.3.3.2 Second-Level Cache Address Translation

Whenever a CPU reference that can be stored in the first-level cache causes a miss of the first-level cache, a quadword transfer is initiated on the CDAL bus and the second-level cache is checked to determine if the contents of the location(s) being addressed is stored there. The cache is checked by translating the address as follows:

- On noncacheable references, the reference is never stored in the cache, so a second-level cache miss occurs, the main memory cycle is allowed to complete and the data is provided by the main memory controller.
- On cacheable references, the physical address must be translated to determine if the contents of the referenced location(s) is resident in the cache. In this case, the cache index field, bits <15:3> of the physical address, is used to select one of the 8K entries (rows) in the set. The cache tag field, bits <28:16> of the physical address, is then compared to the tag block of the selected entry. Bits <28:26> of this field must be zero since the second-level cache is designed to support a maximum of 64 Mbytes of main memory.

If a match occurs with the tag block of the entry, and the valid bit within the entry is set, then the contents of the location is contained in the cache and a second-level cache hit occurs. The cache displacement field, bits <2:0> of the physical address, is used to select the longword within the block. Bits <1:0> of this field are ignored since the byte mask signals are used to select the desired byte(s) within a longword. Main memory cycles are initiated on all CDAL bus cycles, but they are aborted before completion on second-level cache hits.

If there is no match, then the contents of the location is not contained in the second-level cache, a cache miss occurs, the main memory cycle is allowed to complete, and the data is provided by the main memory controller (Figure 3–19).

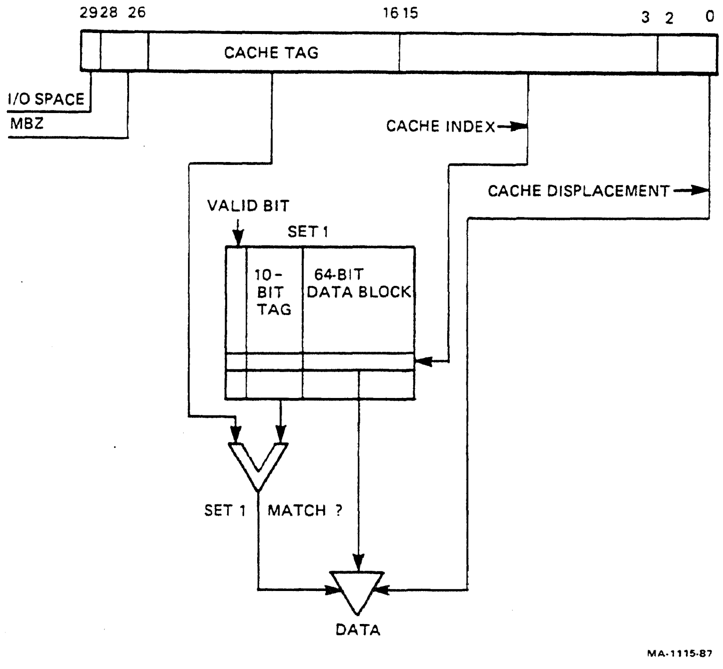


Figure 3–19 Second-Level Cache Address Translation

3.3.3.3 Second-Level Cache Data Block Allocation

On cacheable references that miss the first-level cache, a quadword read is initiated on the CDAL bus. If the requested quadword cannot be found in the second-level cache, it is provided by the main memory controller. Both caches allocate a data block for storing the entire quadword, and the requested longword is passed on to the CPU.

Because the second-level cache is direct mapped, there is one and only one data block in the cache that can be allocated to a given quadword. This data block is determined by the cache index field of the physical address of the quadword, which selects a unique row (data block) within the cache.

Since the KA650 supports 64 Mbytes (8 M quadwords) of physical memory, up to 1K quadwords share each data block (row) of the cache. Contiguous programs larger than 64 Kbytes, or noncontiguous programs separated by 64 Kbytes overwrite themselves in the cache when cache data blocks are allocated for memory references separated by 64 Kbytes.

3.3.3.4 Second-Level Cache Behavior on Writes

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

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

3.3.3.5 Cache Control Register

The cache control register (CACR), address 2008 4000 $_{16}$, controls the second-level cache and is unique to the KA650. Only the low byte of this register should be written (Figure 3–20).

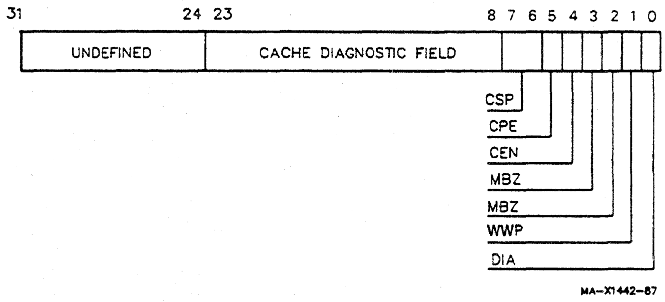


Figure 3–20 Cache Control Register

Data Bit	Definition
<31:24>	Undefined. Undefined on reads. Writes have no effect.
<23:8>	Cache diagnostic field. Read only.
<7:6>	CVAX cycle speed (CSP). Read only. Writes have no effect. These bits are used to indicate the speed of the CVAX chip being used. They are encoded as follows:

<7:6>	**Speed**
00	Reserved for future use
01	Reserved for future use
10	90 ns
11	100 ns

Data Bit	Definition
<5>	Cache parity error (CPE). Read/write to clear. This bit is set whenever a cache tag parity error is detected. Cleared by writing a 1, on power-up and the negation of DCOK when the processor is halted.
<4>	Cache enable (CEN). Read/write. When cleared, all references miss the cache except those to the cache diagnostic space and the allocation of cache blocks is prevented. When set, the configuration of the first-level cache determines which types of references are stored. CACR<0> and CACR<5> must be cleared before this bit can be set. Cleared on power-up and the negation of DCOK when the processor is halted. NOTE **Whenever the second-level cache is disabled, it should be flushed before re-enabling to ensure that data that may have become stale, while the cache was disabled, is not utilized.**
<3:2>	Unused. Always read as zero. Must be written as zero.

Data Bit	Definition
<1>	Write wrong parity (WWP). Read/write. When set, the tag parity bits stored in the tag block are incorrect (inverted), and the data parity bits stored in the data block are forced to all 1s whenever the cache data block is written. When cleared, correct parity is stored in both the tag block and the data block whenever the cache is written. Tag parity errors force a second-level cache miss, so the cache has to be read through the tag diagnostic space to check that parity was incorrectly written. Cleared on power-up and the negation of DCOK when the processor is halted.
<0>	Diagnostic mode DIA. Read/write. When set, the second-level cache is disabled, and writes to the cache diagnostic space set the valid bit for the entry that is written as well as load the tag field of the physical address into the tag block of the corresponding second-level cache entry. This mode allows the second-level cache tag block to be fully tested. When cleared, CACR<4> determines if the cache is enabled, and writes to the cache diagnostic space clear the valid bit for the entry that is written. This mode allows the second-level cache to be flushed by writing any value to each entry through the cache diagnostic space. Cleared on power-up and the negation of DCOK when the processor is halted.

3.3.3.6 Second-Level Cache Error Detection

Both the tag and data arrays in the second-level cache are protected by parity. Each 8-bit byte of cache data and the 10-bit tag is stored with an associated parity bit. Odd data bytes are stored with odd parity and even data bytes are stored with even parity. The tag is stored with odd parity. The stored parity is always valid regardless of the state of the valid bit.

Tag parity is checked by the second-level cache logic on CPU read, CPU write and DMA write references. Tag parity is generated by the second-level cache logic during the allocation of a cache block.

Data parity is checked on a byte basis by the CVAX chip for CPU read references that hit the cache. Data parity is taken directly from the CDAL bus parity lines on CPU write operations that hit the cache and during the allocation of a cache block.

Upon detecting second-level cache tag parity errors the entire second-level cache is disabled (CACR<4> is cleared), CACR<5> (second-level cache parity error) is set, and an interrupt at IPL 1A through vector 54 $_{16}$ is generated.

The action taken following the detection of a second-level cache data parity error is identical to that for CDAL bus parity errors and depends on the reference type:

- During a demand D-stream reference, the first-level cache entry is invalidated, the cause of the error is logged in MSER<6:5> and a machine check abort is initiated and the bad data in the second-level cache remains unaltered.
- During a request I-stream reference (prefetch), the row containing the first-level cache entry is invalidated, the prefetch operation is aborted, the cause of the error is logged in MSER<6>, but no machine check is generated and the bad data in the second-level cache remains unaltered.
- During a request D-stream or I-stream reference (fill), the first-level cache entry is invalidated, the first-level cache fill operation is aborted, the cause of the error is logged in MSER<6>, but no machine check is generated and the bad data in the second-level cache remains unaltered.

3.3.3.7 Second-Level Cache as Fast Memory

The second-level cache can be accessed as part of main memory for diagnostic purposes as well as for fast execution of bootstrap or self-test code. One thousand and twenty four copies of the second-level cache data array appear starting at the first address in the upper half of VAX memory space (physical addresses 1000 0000 $_{16}$ to 13FF FFFF $_{16}$). This area is called the *cache diagnostic space*. Read or write references to this address range access the second-level cache as high speed (180 ns) RAM. Read references will not affect the existing tag block for the accessed cache entry.

When the diagnostic mode bit CACR<0>, is cleared, write references invalidate any cache entry that is accessed through the cache diagnostic space. This prevents stale data from accumulating when the cache is used as high speed RAM.

When the diagnostic mode bit is set, write references set the valid bit in the tag block and write the tag field of the physical address into the tag of any entry that is accessed through the cache diagnostic space. This allows any of the 1024 possible cache tag bit-patterns to be written into the tag block of any cache entry by writing to one of the 1024 copies of the cache entry.

Data parity errors that occur while using the second-level cache as high speed RAM have the same effect as parity errors encountered during the normal operation of the cache. Tag parity is not checked on access to cache diagnostic space.

NOTE
To flush the second-level cache, each cache entry must be written through the cache diagnostic space with the diagnostic mode bit cleared.

3.4 Main Memory System

The KA650 includes a main memory controller implemented through a single VLSI chip called the CMCTL. The KA650 main memory controller communicates with the MS650 memory boards over the MS650 memory interconnect, which utilizes the CD interconnect for the address and control lines and a 50-pin, ribbon cable for the data lines. It supports up to four MS650 memory boards, for a maximum of 64 Mbytes of ECC memory.

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

Read references are aborted by the second-level cache on second-level cache hits, and by the Q22-bus interface if they are locked and the KA650 is not the Q22-bus master.

Table 3–9 CPU Read Reference Timing

Data Type	Timing (ns)
Longword	450
Quadword	720
First longword	450
Second longword	270
Aborted reference	450
Longword (locked)	990 minimum
Aborted reference	450
Retry (locked)	540

Table 3–10 CPU Write Reference Timing

Data Type	Timing (ns)
Longword	180
Longword (masked)	540

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

Table 3–11 Q22-bus Interface Read Reference Timing

Data Type	Timing (ns)
Longword	540
Quadword	900
First longword	540
Second longword	360
Longword (locked)	630

Table 3–12 Q22-bus Interface Write Reference Timing

Data Type	Timing (ns)
Longword	360
Longword (masked)	630
Quadword	630
First longword	360
Second longword	270
Quadword (masked)	1080
First longword	360
Second longword	720
Hexword	900
First longword	360
Second longword	270
Third longword	270
Hexword (masked)	1350
First longword	360
Second longword	270
Third longword	720
Octaword	1170
First longword	360
Second longword	270
Third longword	270
Fourth longword	270
Octaword (masked)	1620
First longword	360
Second longword	270
Third longword	270
Fourth longword	720

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

Data Type	Timing (ns)
Correctable error	0
Uncorrectable error	180 read
Uncorrectable error	90 write
CDAL parity error	90 write
Refresh collision	450

The main memory controller contains 18 registers. Sixteen registers are used to configure each of the 16 possible banks in main memory. One register is used to control the operating mode of all memory banks and one register captures state on main memory errors.

3.4.1 Main Memory Organization

Main memory is logically and physically divided into four boards which correspond to the four possible MS650 memory expansion modules that can be attached to a KA650. Each board can contain zero (no memory module present), or two (MS650-AA present) memory banks. Each bank contains 1,048,576 (1 M) aligned longwords. Each aligned longword is divided into four data bytes and is stored with seven ECC check bits, resulting in a memory array width of 39 bits.

3.4.2 Main Memory Addressing

The KA650 main memory controller is capable of controlling up to 16 banks of RAM, each bank containing 4 Mbytes of storage. Each bank of main memory has a programmable base address, determined by the state of bits <25:22> of the main memory configuration register associated with each bank.

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

The main memory controller accesses main memory on read/write references in parallel with the address translation process in the second-level cache. On CPU read references that hit the second-level cache, the memory controller reads the longword from main memory, but the operation is aborted before the data gets placed on the CDAL bus.

3.4.3 Main Memory Behavior on Writes

On unmasked CPU write references, the main memory controller operates in dump and run mode, terminating the CDAL bus transaction after latching the data, but before checking CDAL bus parity, calculating the ECC check bits, and transferring the data to main memory. This allows main memory to keep up with the second-level cache without impacting the CPU write performance.

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

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

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

3.4.4 Main Memory Error Status Register

The main memory status register (MEMCSR16), address 2008 0140 $_{16}$, is used to capture main memory error data. This register is unique to CPU designs that use the CMCTL memory controller chip (Figure 3–21).

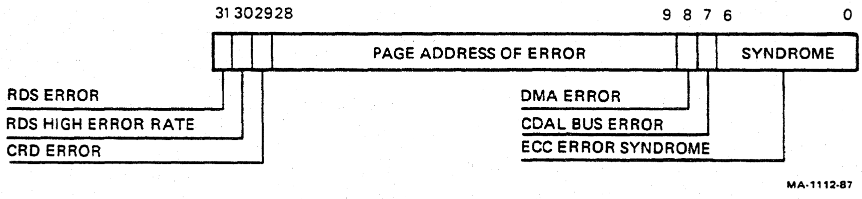


Figure 3–21 Format for MEMCSR16

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

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

Table 3–13 Error Syndromes

Syndrome<6:0>	Bit Position in Error
0000000	No error detected
Data bits (0 to 32 decimal)	
1011000	0
0011100	1
0011010	2
1011110	3
0011111	4
1011011	5
1011101	6
0011001	7
1101000	8
0101100	9
0101010	10

Table 3–13 (Cont.) Error Syndromes

Syndrome<6:0>	Bit Position in Error
1101110	11
0101111	12
1101011	13
1101101	14
0101001	15
1110000	16
0110100	17
0110010	18
1110110	19
0110111	20
1110011	21
1110101	22
0110001	23
0111000	24
1111100	25
1111010	26
0111110	27
1111111	28
0111011	29
0111101	30
1111001	31
Check bits (32 to 38 decimal)	
0000001	32
0000010	33
0000100	34
0001000	35
0010000	36
0100000	37
1000000	38
0000111	Result of incorrect check bits written on detection of a CDAL parity error.
All others	Multibit errors

3.4.5 Main Memory Control and Diagnostic Status Register

The main memory control and diagnostic status register (MEMCSR17), address 2008 0144 , is used to control the operating mode of the main memory controller as well as to store diagnostic status information. This register is unique to CPU designs that use the CMCTL memory controller chip (Figure 3–22).

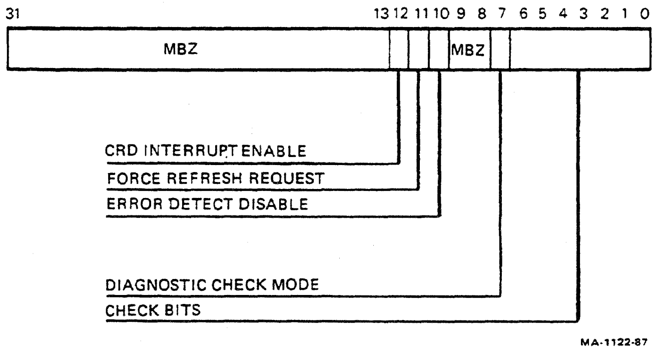


Figure 3–22 Format for MEMCSR17

Data Bit	Definition
<31:13>	Unused. Reads as zero must be written as zero.
<12>	CRD interrupt enable. Read/write. When cleared, single-bit errors are corrected by the ECC logic, but no interrupt is generated. When set, single-bit errors are corrected by the ECC logic and they cause an interrupt to be generated at IPL 1A with a vector of 54_{16} . This bit has no effect on the capturing of error information in MEMCSR16, or on the reporting of uncorrectable errors. Cleared on power-up and the negation of DCOK when the processor is halted.

Data Bit	Definition
<11>	Force refresh request. Read/write. When cleared, the refresh control logic operates in normal mode (refresh every 10.26 µs). When set, one memory refresh operation occurs immediately after the MEMCSR write reference that set this bit. Setting this bit provides a mechanism for speeding up the testing of the refresh logic during manufacturing test of the controller chip. This bit is cleared by the memory controller upon completion of the refresh operation. Cleared on power-up and the negation of DCOK when the processor is halted.
<10>	Memory error detect disable. Read/write. When set, error detection and correction (ECC) is disabled, so all memory errors go undetected. When cleared, error detection, correction, state capture and reporting (through MEMCSR16) is enabled. Cleared on power-up and the negation of DCOK when the processor is halted.
<9:8>	Unused. This field reads as zero and must be written as zero.
<7>	Diagnostic check mode. Read/write. When set, the contents of MEMCSR17<6:0> are written into the seven ECC check bits of the location (even if a CDAL parity error is detected) during a memory write reference. When cleared, the seven check bits calculated by the ECC generation logic are loaded into the seven ECC check bits of the location during a write reference and a memory read reference loads the state of the seven ECC check bits of the location that was read into MEMCSR17<6:0>. Cleared on power-up and the negation of DCOK when the processor is halted. **NOTE** **Diagnostic check mode is restricted to unmasked memory write references. No masked write references are allowed when diagnostic check mode is enabled.**

Data Bit	Definition
<6:0>	Check bits. Read/write. When the diagnostic check mode bit is set, these bits are substituted for the check bits that are generated by the ECC generation logic during a write reference. When the diagnostic check mode bit is cleared, memory read references load the state of the seven ECC check bits of the location that was read into MEMCSR16<6:0>. Cleared on power-up and the negation of DCOK when the processor is halted.

3.4.6 Main Memory Error Detection and Correction

The KA650 main memory controller generates CDAL bus parity on CPU read references, and checks CDAL bus parity on CPU write references.

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

- For unmasked CPU write references: incorrect check bits are written to main memory (potentially masking an as yet undetected memory error) along with the data; an interrupt is generated at IPL 1D through vector 60 $_{16}$ on the next cycle; and MCSR16<7> is set. The incorrect check bits are determined by calculating the seven correct check bits, and complementing the three least significant bits.
- For masked CPU write references: incorrect check bits are written to main memory (potentially masking an as yet undetected memory error) along with the data, unless an uncorrectable error is detected during the read part; MEMCSR16<7> is set; and a machine check abort is initiated. If an uncorrectable error is detected on the read part, no write operation takes place. The incorrect check bits are determined by calculating the seven correct check bits, and complementing the three least significant bits.

The memory controller protects main memory by using a 32-bit modified hamming code to encode the 32-bit data longword with seven check bits. This allows the controller to detect and correct single-bit errors in the data field and detect single-bit errors in the check bit field and double-bit errors in the data field. The most likely causes of these errors are failures in either the memory array or the 50-pin cable.

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

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

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

- On a demand read reference: the affected row of the first-level cache is invalidated; bit <31> of MEMCSR16 is set; bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error; and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable and a machine check abort is initiated. If the read is a local-miss, global- hit read, or a read of the Q22-bus map, MEMCSR16<8> and DSER<4> are also set, and DEAR<12:0> are loaded with the address of the page containing the location that caused the error.
- On a request read reference: the prefetch or fill cycle is aborted; but no machine check occurs; bit <31> of MEMCSR16 is set; bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error; and bits <6:0> are loaded with the error syndrome which indicates that the error was uncorrectable.
- On the read part of masked write reference: bit <31> of MEMCSR16 is set; bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error; and bits <6:0> are loaded with the error syndrome which indicates that the error is uncorrectable and a machine check abort is initiated.

- On a DMA read reference: bit <31> and bit <8> of MEMCSR16 are set; bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error; and bits <6:0> are loaded with the error syndrome which indicates that the error is uncorrectable. DSER<4> is set; DEAR<12:0> are loaded with the address of the page containing the location that caused the error; BDAL<17:16> are asserted on the Q22-bus along with the data to notify the receiving device (unless it was a map read by the Q22-bus interface during translation); and an interrupt is generated at IPL 1D through vector 60_{16}.
- On a DMA masked write reference: bit <31> and bit <8> of MEMCSR16 are set; bits <28:9> of MEMCSR16 are loaded with the address of the page containing the location that caused the error; and bits <6:0> are loaded with the error syndrome which indicates that the error is uncorrectable. DSER<4> is set; DEAR<12:0> are loaded with the address of the page containing the location that caused the error; IPCR<15> is set to notify the initiating device; and an interrupt is generated at IPL 1D through vector 60_{16}.

3.5 Console Serial Line

The console serial line provides the KA650 processor with a full duplex, RS-423 EIA, serial line interface, which is also RS-232C compatible. The only data format supported is 8-bit data with no parity and one stop bit. The four internal processor registers (IPRs) that control the operation of the console serial line are a superset of the VAX console serial line registers described in the *VAX Architecture Reference Manual.*

3.5.1 Console Registers

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

Table 3–14 Console Registers

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

3.5.1.1 Console Receiver Control/Status Register

The console receiver control/status register (RXCS), IPR 32, is used to control and report the status of incoming data on the console serial line (Figure 3–23).

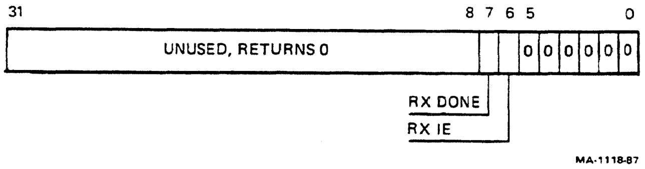


Figure 3–23 Console Receiver Control/Status Register

Data Bit	Definition
<31:8>	Unused. Read as zeros. Writes have no effect.
<7>	Receiver done (RX DONE). Read only. Writes have no effect. This bit is set when an entire character has been received and is ready to be read from the RXDB register. This bit is automatically cleared when RXDB is read. It is also cleared on power-up and the negation of DCOK when the processor is halted.

Data Bit	Definition
<6>	Receiver interrupt enable (RX IE). Read/write. When set, this bit causes an interrupt to be requested at IPL 14 with an SCB offset of F8 if RX done is set. When cleared, interrupts from the console receiver are disabled. This bit is cleared on power-up and the negation of DCOK when the processor is halted.
<5:0>	Unused. Read as zeros. Writes have no effect.

3.5.1.2 Console Receiver Data Buffer

The console receiver data buffer (RXDB), IPR 33, is used to buffer incoming data on the serial line and capture error information (Figure 3–24).

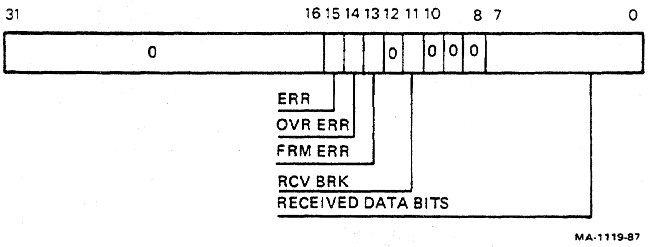


Figure 3–24 Console Receiver Data Buffer

Data Bit	Definition
<31:16>	Unused. Always read as zero. Writes have no effect.
<15>	Error (ERR). Read only. Writes have no effect. This bit is set if RBUF <14> or <13> is set. It is clear if these two bits are clear. This bit cannot generate a program interrupt. Cleared on power-up and the negation of DCOK when the processor is halted.

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

3.5.1.3 Console Transmitter Control/Status Register

The console transmitter control/status register (TXCS), internal processor register 34, is used to control and report the status of outgoing data on the console serial line (Figure 3–25).

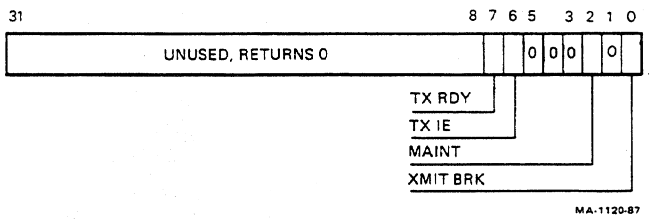


Figure 3–25 Console Transmitter Control/Status Register

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

3.5.1.4 Console Transmitter Data Buffer

The console transmitter data buffer (TXDB), internal processor register 35, is used to buffer outgoing data on the serial line (Figure 3–26).

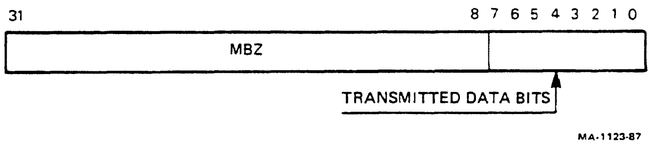


Figure 3–26 Console Transmitter Data Buffer

Data Bit	Definition
<31:8>	Unused. 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.

3.5.2 Break Response

The console serial line unit recognizes a break condition which consists of 20 consecutively received space bits. If the console detects a valid break condition, the RCV BRK bit is set in the RXDB register. If the break was the result of 20 consecutively received space bits, the FRM ERR bit is also set. If halts are enabled (HLT ENB asserted on the 20-pin connector), the KA650 halts and transfers program control to ROM location 2004 0000 when the RCV BRK bit is set. RCV BRK is cleared by reading RXDB. Another mark followed by 20 consecutive space bits must be received to set RCV BRK again.

3.5.3 Baud Rate

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

The user selects the desired baud rate through the baud rate select signals (BRS <2:0> L) which are received from an external 8-position switch through the 20-pin connector mounted at the top of the module. The KA650 firmware reads this code from boot and diagnostic register bits <6:4> and loads it into SSC configuration register bits <14:12>. Operating systems will not cause the baud rate to be transferred. The baud rate is only set at power-up.

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

Table 3–15 Baud Rate Select

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

3.5.4 Console Interrupt Specifications

The console serial line receiver and transmitter both generate interrupts at IPL 14. The receiver interrupts with a vector of F8 $_{16}$, while the transmitter interrupts with a vector of FC $_{16}$.

3.6 Time-of-Year Clock and Timers

The KA650 clocks include time-of-year clock (TODR) as defined in the *VAX Architecture Reference Manual*, a subset interval clock (subset ICCS), as defined in the *VAX Architecture Reference Manual*, and two additional programmable timers modeled after the VAX standard interval clock.

3.6.1 Time-of-Year Clock

The KA650 time-of-year clock (TODR), internal processor register 27, forms an unsigned 32-bit binary counter that is driven from a 100 Hz oscillator. Therefore, the least significant bit of the clock represents a resolution of 10 ms, with less than 0.0025 percent error. The register counts only when it contains a nonzero value. This register is implemented in the SSC (Figure 3–27).

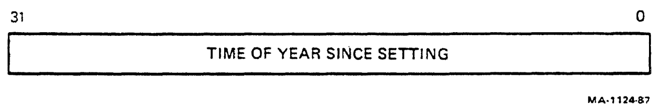


Figure 3–27 Time-of-Year Clock

The time-of-year clock is maintained during power failure by battery backup circuitry which interfaces, through the external connector, to a set of batteries which are mounted on the H3600-SA cover (or the CPU distribution insert). The TODR remains valid for greater than 162 hours when using the NiCad battery pack (three batteries in series).

The SSC configuration register contains a battery low (BLO) bit which, if set after initialization, the TODR is cleared, and remains 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 1 to it.

3.6.2 Interval Timer

The KA650 interval timer (ICCS), internal processor register 24, is implemented according to the *VAX Architecture Reference Manual* for subset processors. The interval clock control/status register (ICCS) is implemented as the standard subset of the standard VAX ICCS in the CVAX CPU chip, while NICR and ICR are not implemented (Figure 3–28).

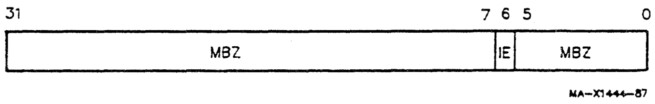


Figure 3–28 Interval Timer

Data Bit	Definition
<31:7>	Unused. Read as zeros, must be written as zeros.
<6>	Interrupt enable (IE). Read/write. This bit enables and disables the interval timer interrupts. When the bit is set, an interval timer interrupt is requested every 10 ms with an error of less than 0.01 percent. When the bit is clear, interval timer interrupts are disabled. This bit is cleared on power-up and the negation of DCOK when the processor is halted.
<5:0>	Unused. Read as zeros, must be written as zeros.

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

3.6.3 Programmable Timers

The KA650 features two programmable timers. Although they are modeled after the VAX standard interval clock, they are accessed as I/O space registers (rather than as internal processor registers) and a control bit has been added which stops the timer upon overflow. If so enabled, the timers interrupt at IPL 14 upon overflow. The interrupt vectors are programmable and are set to 78 and 7C by the firmware.

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

3.6.3.1 Timer Control Registers

The KA650 has two timer control registers, one for controlling timer 0 (TCR0), and one for controlling timer 1 (TCR1). TCR0 is accessible at address 2014 0100 $_{16}$, and TCR1 is accessible at 2014 0110 $_{16}$. These registers are implemented in the SSC (Figure 3–29).

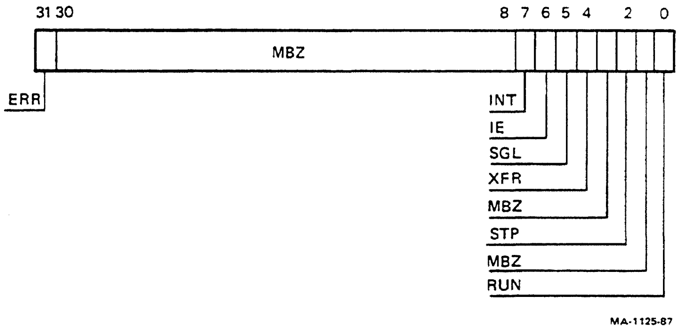


Figure 3–29 Timer Control Registers

Data Bit	Definition
<31>	Error (ERR). Read/write to clear. This bit is set whenever the timer interval register overflows and INT is already set. Thus, the ERR indicates a missed overflow. Writing a 1 to this bit clears it. Cleared on power-up and the negation of DCOK when the processor is halted.
<30:8>	Unused. 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 1 to this bit clears it. Cleared on power-up and the negation of DCOK when the processor is halted.
<6>	IE. Read/write. When this bit is set, the timer interrupts at IPL 14 when the INT bit is set. Cleared on power-up and the negation of DCOK when the processor is halted.

Data Bit	Definition
<5>	SGL. Read/write. Setting this bit causes the timer interval register to be incremented by 1 if the run bit is cleared. If the run bit is set, then writes to the SGL bit are ignored. This bit always reads as 0. Cleared on power-up and the negation of DCOK when the processor is halted.
<4>	XFR. Read/write. Setting this bit causes the timer next interval register to be copied into the timer interval register. This bit is always read as 0. Cleared on power-up and the negation of DCOK when the processor is halted.
<3>	Unused. Read as zeros. Must be written as zeros.
<2>	STP. Read/write. This bit determines whether the timer stops after an overflow when the run bit is set. If the STP bit is set at overflow, the run bit is cleared by the hardware at overflow and counting stops. Cleared on power-up and the negation of DCOK when the processor is halted.
<1>	Unused. 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 power-up and the negation of DCOK when the processor is halted.

3.6.3.2 Timer Interval Registers

The KA650 has two timer interval registers, one for timer 0 (TIR0), and one for timer 1 (TIR1). TIR0 is accessible at address 2014 0104 $_{16}$, and TIR1 is accessible at 2014 0114 $_{16}$.

The timer interval register is a read only register containing the interval count. When the run bit is 0, writing a 1 increments the register. When the run bit is 1, the register is incremented once every microsecond. When the counter overflows, the INT bit is set, and an interrupt is posted at IPL 14 if the IE bit is set. Then, if the run and STP bits are both set, the run bit is cleared and counting stops. Otherwise, the counter is reloaded. The maximum delay that can be specified is approximately 1.2 hours. This register is cleared on power-up and the negation of DCOK when the processor is halted (Figure 3–30).

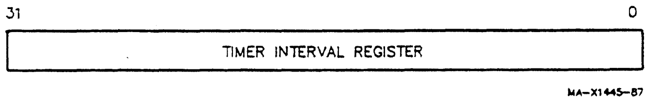


Figure 3–30 Timer Interval Register

3.6.3.3 Timer Next Interval Registers

The KA650 has two timer next interval registers, one for timer 0 (TNIR0), and one for timer 1 (TNIR1). TNIR0 is accessible at address 2014 0108 $_{16}$, and TNIR1 is accessible at 2014 0118 $_{16}$. These registers are implemented in the SSC.

This read/write register contains the value which is written into the timer interval register after overflow, or in response to a 1 written to the XFR bit. This register is cleared on power-up and the negation of DCOK when the processor is halted (Figure 3–31).

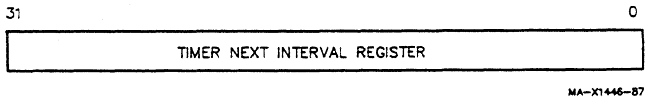


Figure 3–31 Timer Next Interval Register

3.6.3.4 Timer Interrupt Vector Registers

The KA650 has two timer interrupt vector registers, one for timer 0 (TIVR0), and one for timer 1 (TIVR1). TIVR0 is accessible at address 2014 010C $_{16}$, and TIVR1 is accessible at 2014 011C $_{16}$. These registers are implemented in the SSC and are set to 78 and 7C respectively by the resident firmware.

This read/write register contains the timer's interrupt vector. Bits <31:10> and <1:0> are read as 0 and must be written as 0. When TCRn<6> (IE) and TCRn<7> (INT) transition to 1, an interrupt is posted at IPL 14. When a timer's interrupt is acknowledged, the content of the interrupt vector register is passed to the CPU, and the INT bit is cleared. Interrupt requests can also be cleared by clearing either the IE or the INT bit. This register is cleared on power-up and the negation of DCOK when the processor is halted (Figure 3–32).

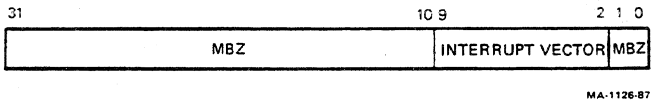


Figure 3–32 Timer Interrupt Vector Register

NOTE

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

3.7 Boot and Diagnostic Facility

The KA650 boot and diagnostic facility features two registers, two 28-pin ROM sockets containing 128 Kbytes of EPROM, and 1 Kbyte of battery backed-up RAM. The ROM and battery backed-up RAM may be accessed through longword, word or byte references.

The KA650 CPU module populates the ROM sockets with 64 Kbytes of 16-bit ROM (or EPROM). This ROM contains the KA650 resident firmware. If this ROM is replaced for special applications, the new ROM must initialize and configure the board, provide halt and console emulation, as well as provide boot diagnostic functionality.

3.7.1 Boot and Diagnostic Register

The boot and diagnostic register (BDR) is a byte-wide register located in the VAX I/O page at physical address 2008 4004 $_{16}$. It is implemented uniquely on the KA650. It can be accessed by KA650 software, but not by external Q22-bus devices. The BDR allows the boot and diagnostic ROM programs to read various KA650 configuration bits. Only the low byte of the BDR should be accessed. Bits <31:8> are undefined (Figure 3–33).

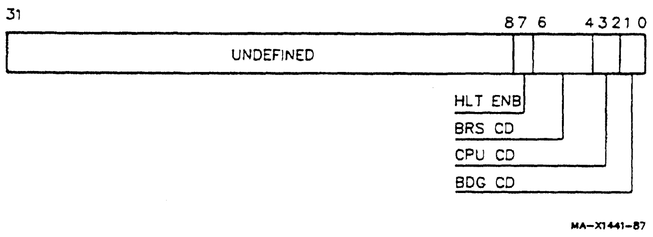


Figure 3–33 Boot and Diagnostic Register

Data Bit	Definition
<31:8>	Undefined. Should not be read or written.
<7>	Halt Enable (HLT ENB). Read only. Writes have no effect. This bit reflects the state of pin 15 (HLT ENB L) of the 20-pin connector. The assertion of this signal enables the halting of the CPU upon detection of a console break condition. On a power-up, the KA650 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 a halt instruction while in kernel mode, the KA650 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).

Data Bit	Definition
<6:4>	Baud rate select (BRS CD) <2:0>. Read only. Writes have no effect. These three bits originate from pins <19:17> (BRS<2:0>) of the 20-pin connector. They reflect the setting of the baud rate select switch on the H3600-SA cover (or on the CPU distribution insert). These bits are read only on power-up.

BDR<6:47>	Baud Rate
000	300
001	600
010	1200
011	2400
100	4800
101	9600
110	19200
111	38400

Data Bit	Definition
<3:2>	CPU code (CPU CD) <1:0>. Read only. Writes have no effect. These two bits originate from connector pins <5:4> (CPU CD<1:0>).

CPU CD <1:0>	Configuration
00	Normal operation
01	Reserved
10	Reserved
11	Reserved

Data Bit	Definition
<1:0>	Boot and diagnostic code (BDG CD) <1:0>. Read only. Writes have no effect. This 2-bit code reflects the status of configuration and display connector pins <14:13> (BDG CD<1:0>). The KA650 ROM programs use BDG CD <1:0> to determine the power-up mode as follows:

BDG CD <1:0>	Power-Up Mode
00	Run
01	Language inquiry
10	Test
11	Manufacturing

3.7.2 Diagnostic LED Register

The diagnostic LED register (DLEDR), address 2014 0030 $_{16}$, is implemented in the SSC and contains four read/write bits that control the external LED display. A 0 in a bit turns on the corresponding LED. All four bits are cleared on power-up and the negation of DCOK when the processor is halted to provide a power-up lamp test (Figure 3–34).

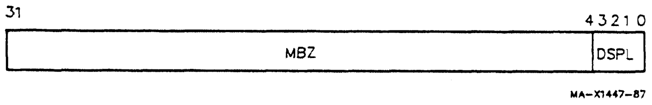


Figure 3–34 Diagnostic LED Register

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

3.7.3 ROM Memory

The KA650 supports up to 128 Kbytes of ROM memory for storing code for board initialization, VAX standard console emulation, board self-tests, and boot code. ROM memory may be accessed through byte, word and longword references. ROM accesses take 1300 ns. ROM is organized as a 64K by 8-bit array for one 64 Kbyte ROM, as a 32K by 16-bit array for two 32 Kbyte ROMs, and as a 64K by 16-bit array for two 64 Kbyte ROMs (ship configuration). CDAL bus parity is neither checked nor generated on ROM references.

3.7.3.1 ROM Socket

The KA650 provides two ROM sockets which contain two 64K by 8 EPROMs.

3.7.3.2 ROM Address Space

The entire 128 Kbyte boot and diagnostic ROM may be read from either the 128 Kbyte halt mode ROM space (hex addresses: 2004 0000 $_{16}$ through 2005 FFFF $_{16}$), or the 128 Kbyte run mode ROM space (hex addresses: 2006 0000 $_{16}$ through 2007 FFFF $_{16}$. Note that the run mode ROM space reads exactly the same ROM code as the halt mode ROM space.

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

Any I-stream read from the halt mode ROM space places the KA650 in halt mode. The Q22-bus SRUN signal is deasserted causing the front panel run light to go out and the CPU is protected from further halts.

Any I-stream read that does not access the halt mode ROM space, including reads from the run mode ROM space, places the KA650 in run mode. The Q22-bus SRUN signal is toggled turning on the front panel run. The CPU can be halted by asserting the Q22-bus BHALT line or by generating a break condition on the console serial line if BDR<7> (halt enable) is set.

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

3.7.3.3 Resident Firmware Operation

The KA650 CPU module populates the ROM socket with 128 Kbyte of 16-bit ROM (or EPROM). This ROM contains the KA650 resident firmware which can be entered by transferring program control to location 2004 0000 $_{16}$.

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

When running, the KA650 resident firmware provides the services expected of a VAX-11 console system. In particular, the following services are available:

- Automatic restart or bootstrap following processor halts or initial power-up.
- An interactive command language allowing the user to examine and alter the state of the processor.
- Diagnostic tests executed on power-up that check out the CPU, the memory system and the Q22-bus map.
- Support of video or hardcopy terminals as the console terminal as well as support of VCB01-based bit-mapped terminals.

Power-Up Modes

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

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

3.7.4 Battery Backed-Up RAM

The KA650 contains 1 Kbyte of battery backed-up static RAM for use as a console scratchpad. The power for the RAM is provided through pins 10 (BTRY VCC) and 12 (GND) of the 20-pin connector.

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 by 32-bit (one longword) array. The array appears in a 1 Kbyte block of the VAX I/O page at addresses 2014 0400 through 2014 07FF.

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

3.7.5 KA650 Initialization

The VAX architecture defines three kinds of hardware initialization:

- Power-up
- Processor
- I/O bus

3.7.5.1 Power-Up Initialization

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

3.7.5.2 Hardware Reset

A KA650 hardware reset occurs on power-up and the negation of DCOK when the processor is halted. A hardware reset initiates the hardware halt procedure (Section 3.1.5.6) with a halt code of 03. The reset also initializes some IPRs and most I/O page registers to a known state. Those IPRs that are affected by a module reset are noted in Section 3.1.1.3. The effect a hardware reset has on I/O space registers is documented in the description of the register.

3.7.5.3 I/O Bus Initialization

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

3.7.5.4 I/O Bus Reset Register

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

3.7.5.5 Processor Initialization

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

In addition to initializing those registers defined in the *VAX Architecture Reference Manual*, the KA650 firmware also configures main memory, the local I/O page, and the Q22-bus map during a processor initialization.

3.8 Q22-bus Interface

The KA650 includes a Q22-bus interface implemented through a single VLSI chip called the CQBIC. It contains a CDAL bus to Q22-bus interface that supports the following functions:

- A programmable mapping function (scatter-gather map) for translating 22-bit, Q22-bus addresses into 29-bit CDAL bus addresses, which allows any page in the Q22-bus memory space to be mapped to any page in main memory.
- A direct mapping function for translating 29-bit CDAL addresses in the local Q22-bus address space and local Q22-bus I/O page into 22-bit, Q22-bus addresses.
- Masked and unmasked longword reads and writes from the CPU to the Q22-bus memory and I/O space and the Q22-bus interface registers. Longword reads and writes of the local Q22-bus memory space are buffered and translated into 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 less than 16 words, the words are buffered and transferred to main memory using the most efficient combination of octaword, quadword, and longword asynchronous DMA transfers.

  The maximum write bandwidth for block mode references is 3.3 Mbytes per second. Block mode reads of main memory from the Q22-bus cause the Q22-bus interface to perform an asynchronous DMA quadword read of main memory and buffer all four words, so that on block mode reads, the next three words of the block mode read can be delivered without any additional CDAL bus cycles. The maximum read bandwidth for Q22-bus block mode references is 2.4 Mbytes per second. Q22-bus burst mode DMA transfers result in single-word reads and writes of main memory.
- Transfers from the CPU to the local Q22-bus memory space, that result in the Q22-bus map translating the address back into main memory (local-miss, global-hit transactions).

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

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

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

3.8.1 Q22-bus to Main Memory Address Translation

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

Q22-bus addresses are translated to main memory addresses as shown in Figure 3–35.

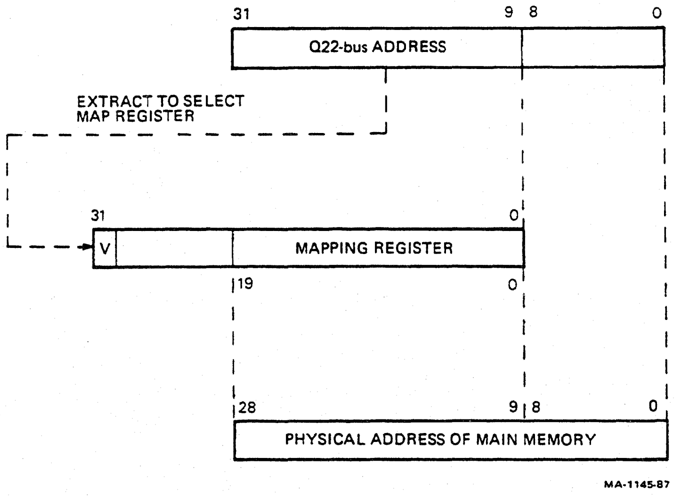

31
9 8
0
Q22-bus ADDRESS
EXTRACT TO SELECT MAP REGISTER
31
0
V
MAPPING REGISTER
19
0
28
9 8
0
PHYSICAL ADDRESS OF MAIN MEMORY
MA-1145-87

Figure 3–35 Q22-bus to Main Memory Address Translation

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

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

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

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

3.8.1.1 Q22-bus Map Registers

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

The local I/O space address of each register was chosen so that register address bits <14:2> are identical to Q22-bus address bits <21:9> of the Q22-bus page which the register maps. Table 3–16 lists the Q22-bus map registers.

Table 3–16 Q22-bus Map Registers

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

The Q22-bus map registers (QMRs) have the format shown in Figure 3–36.

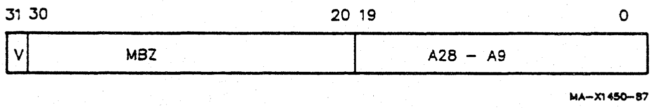


Figure 3–36 Q22-bus Map Registers

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

3.8.1.2 Accessing the Q22-bus Map Registers

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

NOTE

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

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

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

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

3.8.1.3 Q22-bus Map Cache

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

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

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

The format of a Q22-bus map cache entry is as shown in Figure 3–37.

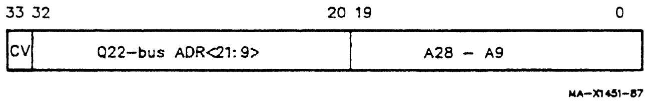


Figure 3–37 Q22-bus Map Cache Entry

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

Data Bit	Definition
<19:0>	Address bits (A28-A9). When a mapping register is selected by a Q22-bus address, and if that register's Cam Valid bit is set, then these 20 bits are used as main memory address bits 28 through 9. Q22-bus address bits 8 through 0 are used as local memory address bits 8 through 0. These bits are undefined on power-up.

3.8.2 CDAL Bus to Q22-bus Address Translation

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

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

3.8.3 Interprocessor Communication Register

The interprocessor communication register (IPCR), address 2000 1F40 $_{16}$, is a 16-bit register which resides in the Q22-bus I/O page address space and can be accessed by any device which can become Q22-bus master (including the KA650 itself). The IPCR, implemented in the CQBIC chip, is byte accessible, meaning that a write byte instruction can write to either the low or high byte without affecting the other byte.

The IPCR also appears at Q22-bus address 17 777 500 (Figure 3–38).

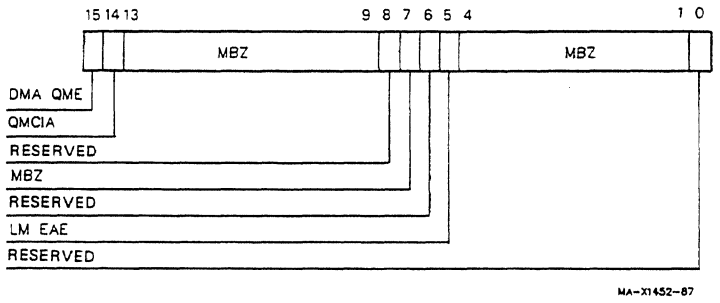


Figure 3–38 Interprocessor Communication Register

Data Bit	Definition
<15>	DMA QME. DMA Q22-bus address space memory error. Read/write to clear. Indicates that an error occurred when a Q22-bus device was attempting to read main memory. It sets if DMA system error register bit DSER<4> (main memory error) sets, or the CDAL bus timer expires. The main memory error bit indicates that an uncorrectable error occurred when an external device (or CPU) was accessing the KA650 local memory. The CDAL bus timer expiring indicates that the memory controller did not respond when the Q22-bus interface initiated a DMA transfer. Cleared by writing a 1 to it, on power-up by the negation of DCOK when the processor halts, by writes to IPR 55 (IORESET), and whenever DSER<4> clears.
<14>	Q22-bus invalidate all (QMCIA). Write only. Writing a 1 to this bit clears the Cam Valid bits in the cached copy of the map. Always reads as zero. Writing a 0 has no effect.
<13:9>	Unused. Read as zeros. Must be written as zeros.
<8>	Reserved for DIGITAL use.
<7>	Unused. Read as zero. Must be written as zero.
<6>	Reserved for DIGITAL use.

Data Bit	Definition
<5>	Local memory external access enable (LM EAE). Read/write when the KA650 is Q22-bus master. Read only when another device is Q22-bus master. Enables external access to local memory when set (through the Q22-bus map). Cleared on power-up and by the negation of DCOK when the processor halts.
<4:1>	Unused. Read as zeros. Must be written as zeros.
<0>	Reserved for DIGITAL use.

3.8.4 Q22-bus Interrupt Handling

The KA650 responds to interrupt requests BR7-4 with the standard Q22-bus interrupt acknowledge protocol (DIN followed by IAK). The console serial line unit, the programmable timers, and the interprocessor doorbell request interrupts at IPL 14 and have priority over all Q22-bus BR4 interrupt requests. After responding to any interrupt request BR7-4, the CPU sets the processor priority to IPL 17. All BR7-4 interrupt requests are disabled unless software lowers the interrupt priority level.

Interrupt requests from the KA650 interval timer are handled directly by the CPU. Interval timer interrupt requests have a higher priority than BR6 interrupt requests. After responding to an interval timer interrupt request, the CPU sets the processor priority to IPL 16. Thus, BR7 interrupt requests remain enabled.

3.8.5 Configuring the Q22-bus Map

The KA650 implements the Q22-bus map in an 8K longword (32 Kbytes) block of main memory. This map must be configured by the KA650 firmware during a processor initialization by writing the base address of the uppermost 32 Kbytes block of good main memory into the Q22-bus map base register. The base of this map must be located on a 32 Kbyte boundary.

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

3.8.5.1 Q22-bus Map Base Address Register

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

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

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

The contents of this register are undefined on power-up and the negation of DCOK when the processor halts. It is not affected by BINIT being asserted on the Q22-bus (Figure 3–39).

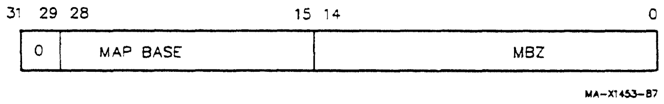


Figure 3–39 Q22-bus Map Base Address Register

3.8.6 System Configuration Register

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

The system configuration register (SCR) is longword, word, and byte accessible. Programmable option fields clear on power-up and by the negation of DCOK when the processor halts. The format of the SCR register is shown in Figure 3–40.

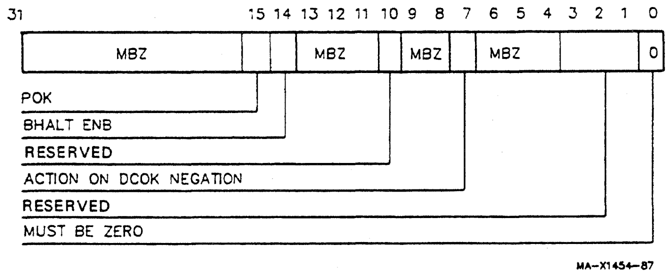


Figure 3–40 System Configuration Register

Data Bit	Definition
<31:16>	Unused. Read as zero. Must be written as zero.
<15>	Power ok (POK). Read only. Writes have no effect. Set if the Q22-bus BPOK signal asserts and clears if it negates. Cleared on power-up and by the negation of DCOK when the processor halts.
<14>	BHALT enable (BHALT EN). Read/write. Controls the effect the Q22-bus BHALT signal has on the CPU. When set, asserting the Q22-bus BHALT signal halts the CPU and asserts DSER<15>. When cleared, the Q22-bus BHALT signal has no effect. Cleared on power-up and by the negation of DCOK when the processor halts.
<13:11>	Unused. Read as zero. Must be written as zero.
<10>	Reserved for DIGITAL use.
<9:8>	Unused. Read as zero. Must be written as zero.

Data Bit	Definition
<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 through the hardware halt procedure with a halt code of 3 when DCOK is negated on the Q22-bus. When set, the Q22-bus interface asserts HALTIN (causing control to be passed to the resident firmware through the hardware halt procedure with a halt code of 2) when DCOK is negated on the Q22-bus. Cleared on power-up and the negation of DCOK when the processor halts.
<6:4>	Unused. Read as zero. Must be written as zero.
<3:1>	Reserved for DIGITAL use.
<0>	Unused. Read as 0. Must be written as zero.

3.8.7 DMA System Error Register

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

The DMA system error register is implemented in the CQBIC chip, and, logs main memory errors on DMA transfers, Q22-bus parity errors, Q22-bus nonexistent memory errors, and Q22-bus no grant errors.

The Q22-bus error address register contains the address of the page in Q22-bus space which caused a parity error during an access by the local processor. The DMA error address register contains the address of the page in local memory which caused a memory error during an access by an external device or the processor during a local-miss global-hit transaction. An access by the local processor which the Q22-bus interface maps into main memory provides error status to the processor when the processor does a retry for a read local-miss global-hit, or by an interrupt in the case of a local-miss global-hit write.

The DSER is a longword, word, or byte accessible read/write register available to the local processor. The bits in this register are cleared to 0 on power-up by the negation of DCOK when the processor halts, and by writes to IPR 55 (IORESET). All bits are set to 1 to record the occurrence of an event. They are cleared by writing a 1. Writing zeros has no effect (Figure 3–41).

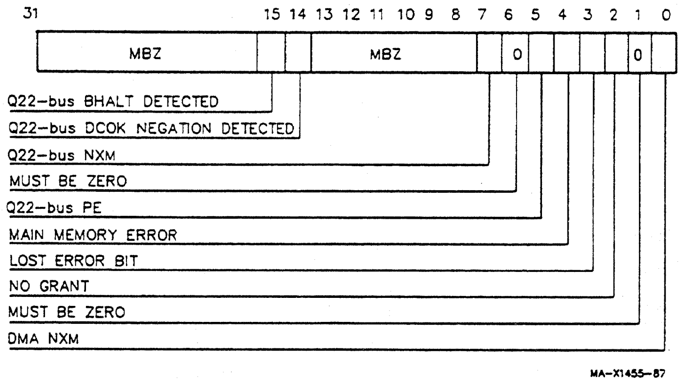


Figure 3–41 DMA System Error Register

Data Bit	Definition
<31:16>	Unused. Read as zero. Must be written as zero.
<15>	Q22-bus BHALT detected. Read/write to clear. Sets when the Q22-bus interface detects that the Q22-bus BHALT line was asserted and SCR<14> BHALT enable is set. Cleared by writing a 1, on power-up by the negation of DCOK when the processor is halted and by writes to IPR 55 (IORESET).
<14>	Q22-bus DCOK negation detected. Read/write to clear. Set when the Q22-bus interface detects the negation of DCOK on the Q22-bus and SCR<7> (action on DCOK negation) is set. Cleared by writing a 1, on power-up by the negation of DCOK when the processor halts and by writes to IPR 55 (IORESET).
<13:8>	Unused. Read as zero. Must be written as zero.

Data Bit	Definition
<7>	Master DMA NXM. Read/write to clear. Sets when the CPU performs a demand Q22-bus read cycle or write cycle that does not reply after 10 µs. Not set during interrupt acknowledge cycles or request read cycles. Cleared by writing a 1, on power-up, by the negation of DCOK when the processor halts 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. Sets when the CPU performs a Q22-bus demand read cycle which returns a parity error. Not set during interrupt acknowledge cycles or request read cycles. Cleared by writing a 1, on power-up, by the negation of DCOK when the processor halts and by writes to IPR 55 (IORESET).
<4>	Main memory error. Read/write to clear. Sets 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 1, on power-up, by the negation of DCOK when the processor halts and by writes to IPR 55 (IORESET).
<3>	Lost error. Read/write to clear. 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 that would have normally captured an address and set either DSER<7,5,4,0> flag. Cleared by writing a 1, on power-up, by the negation of DCOK when the processor halts and by writes to IPR 55 (IORESET).
<2>	No grant timeout. Read/write to clear. Sets 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. Not set during interrupt acknowledge or request read cycles. Cleared by writing a 1, on power-up, by the negation of DCOK when the processor halts and by writes to IPR 55 (IORESET).
<1>	Unused. Read as zero. Must be written as zero.

Data Bit	Definition
<0>	DMA NXM. Read/write to clear. Sets on a DMA transfer to a nonexistent main memory location. Includes local-miss global-hit cycles and map accesses to nonexistent memory. Cleared by writing a 1, on power-up, by the negation of DCOK when the processor halts and by writes to IPR 55 (IORESET).

3.8.8 Q22-bus Error Address Register

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

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

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

Q22-bus address bits <21:9> are loaded into QBEAR bits <12:0>. QBEAR bits <31:13> always read as zeros (Figure 3–42).

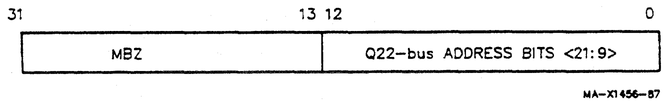


Figure 3–42 Q22-bus Error Address Register

NOTE
This is a read only register. If a write is attempted, a machine check generates.

3.8.9 DMA Error Address Register

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

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

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

Mapped Q22-bus address bits <28:9> are loaded into DEAR bits <19:0>. DEAR bits <31:20> always read as zeros (Figure 3–43).

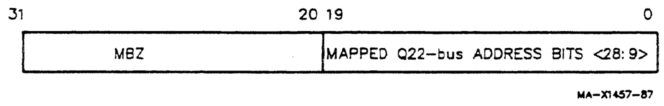


Figure 3–43 DMA Error Address Register

NOTE
This is a read only register. If a write is attempted, a machine check generates.

3.8.10 Error Handling

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

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

The Q22-bus interface maintains several timers to prevent incomplete accesses from hanging the system indefinitely. These 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 acknowledgment 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: the associated row in the first-level cache is invalidated; DSER<7> is set; the address of the Q22-bus page being accessed is captured in QBEAR<12:0>; and a machine check abort is initiated.

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

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

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

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

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

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

If a parity error is detected by another Q22-bus device on a prefetch request read by the CPU: the prefetch aborts; the associated row in the first-level cache is invalidated; 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 negates 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 reads from SCR<15>. The Q22-bus interface continues to operate after generating the power-fail trap, until DCOK negates.

4 KA650 Firmware

This chapter describes the functional operation of the KA650 firmware. The firmware is VAX-11 code that resides in ROM on the KA650 module, and gains control whenever the onboard CPU *halts*, or more precisely, performs a *processor restart* operation. A halt means only that control is transferred to the firmware. It does not mean that the processor actually stops executing instructions.

4.1 KA650 Firmware Features

The firmware is located in two 64 Kbyte EPROMs on the KA650. The KA650 firmware provides the following services:

- Diagnostic tests executed both at power-up and by request, which test all components on the board, and verify the correct operation of the CPU and memory modules.
- An interactive command language that allows the user to examine and alter the state of the processor.
- Automatic/manual bootstrap or restart of an operating system following processor halts.
- Support of various terminals and devices as the system console.
- Multilanguage support for displaying critical system messages and handling LK201 country specific keyboards.

4.1.1 Halt Entry, Exit, and Dispatch

The halt entry code is entered following system halts, resets, or severe errors. The main purpose of this code is to save the state of the machine on halt entry, transfer control to the firmware dispatcher, and restore the state of the machine on exit to program I/O mode.

Naturally, the halt exit code is entered whenever a transition is desired from halted state to the running state and it performs a restoration of the saved context prior to the transition. The halt dispatcher determines the nature of the halt, then transfers control to the appropriate code.

4.1.1.1 Halt Entry - Saving Processor State

The entry code, residing at physical address 2004 0000, is executed whenever a halt occurs. The processor will halt for a variety of reasons. The reason for the halt is stored in PR$_SAVPSL<13:8>(RESTART_CODE), IPR 43. A complete list of the halt reasons and the associated messages can be found in Table 4–10 in Section 4.8. PR$_SAVPC, IPR 42, contains the value of the PC when the processor is halted. On a power-up, PR$_SAVPC is undefined.

One of the first actions the firmware does after a halt is save the current LED code, then it writes an "E" to the diagnostic LEDs. This action occurs within several instructions upon entry into the firmware. The intent of this action is to let the user know that at least some instructions have been successfully executed.

The KA650 firmware unconditionally saves the following registers on any halt:

- R0 through R15, the general purpose registers
- PR$_SAVPSL, the saved PSL register
- PR$_SCBB, the system control block base register
- DLEDR, the diagnostic LED register
- SSCCR, the SSC configuration register
- ADxMCH & ADxMSK, the SSC address match and mask registers

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 initialization or bootstrap. This action ensures that the firmware itself can run and protects the board from physical damage.

Registers that fall into this category are:

- SSCCR, the SSC configuration register
- ADxMCH & ADxMSK, the SSC address match and mask registers
- CBTCR, the CDAL bus timeout control register
- TIVRx, the SSC timer interrupt vector registers

On every halt entry, the firmware sets the console serial line baud rate based on the value read from the BDR and extends the halt protection from 8 Kbyte to 128 Kbyte to include all of the EPROM.

4.1.1.2 Halt Exit - Restoring Processor State

When the firmware exits, it uses the currently defined saved context. This context is initially determined by what is saved on entry to the firmware, and may be modified by console commands, or automatic operations such as an automatic bootstrap on power-up.

When restoring the context, the firmware will flush both caches if enabled, and invalidate all translation buffer entries through the internal processor register PR$_TBIA, IPR 57.

In restoring the context, the console pushes the user's PSL and PC onto the user's interrupt stack, then executes an REI from that stack. This implies that the user's ISP is valid before the firmware can exit. This is done automatically on a bootstrap. However, it is suggested that the SP is set to a valid memory location before issuing the START or CONTINUE command. Furthermore, the user should validate PR$_SCBB prior to executing a NEXT command, since the firmware utilizes the trace trap vector for this function. At power-up, the user ISP is set to 200 (hex) and PR$_SCBB is undefined.

4.1.1.3 Halt Dispatch

The action taken by the firmware on a halt is dependent primarily on the following information:

- The halt enable switch, BDR<7>(HALT_ENABLE)
- The halt action field, CPMBX<1:0>(HALT_ACTION)
- The halt code, PR$_SAVPSL<13:8>(RESTART_CODE), in particular the power-up state

In general, the halt enable switch governs whether external halt conditions are recognized by the KA650. The halt action field in the console program mailbox, is a two bit field used by operating systems to force the firmware to enter the console, restart, or reboot following a halt, regardless of the setting of the halt enable switch. The halt (or restart) code is automatically deposited in PR$_SAVPSL on any processor restart operation. The action taken on a halt is summarized in Table 4–1.

Table 4–1 Halt Action Summary

Halt Enable	Power-Up	Halt Action	Action
T	T	x	diagnostics, halt
T	F	0	halt
F	T	x	diagnostics, bootstrap, halt
F	F	0	restart, bootstrap, halt
x	F	1	restart, halt
x	F	2	bootstrap, halt
x	F	3	halt

"*T*" indicates that the condition is true.
"*F*" indicates that the condition is false.
"*x*" indicates that the condition is "don't care".

Multiple actions mean that the first action is taken and only if it fails is the next action taken. Diagnostics are an exception, if diagnostics fail, the console is entered.

Because the KA650 does not support battery backed up main memory, an operating system restart operation is not attempted on a power-up.

4.1.1.4 External Halts

Several conditions can trigger an external halt (PR$_SAVPSL<13:8>(RESTART_CODE) = 2), and different actions are taken depending on the condition.

An external halt can be caused by:

1. Pressing BREAK on the system console terminal, if the break enable switch is set to enable.
2. Assertion of the BHALT line on the Q22-bus, if the SCR<14>(BHALT_ENABLE) bit in the CQBIC is set.
3. Negation of DCOK, if the SCR<7>(DCOK_ACT) bit is set.

NOTE
The switch labeled RESTART on some BA213 and BA215 system enclosures negates DCOK. The negation of DCOK may also be asserted by the DEQNA sanity timer, or any other Q22-bus module that chooses to implement the Q22-bus restart/reboot protocol.

4.1.2 Power-Up

On a power-up, the KA650 firmware performs actions that are unique to this condition. Among these actions are initial power-up tests, locating and identifying a console device, language query, and the remaining diagnostics. Certain actions are dependent on the state of the mode switch on the H3600-SA panel which has three settings: test, query, and normal. This section describes the sequence of events which occurs on power-up.

4.1.2.1 Initial Power-Up Test

The first action performed on power-up is the initial power-up test (IPT). The purpose of the IPT is to verify that the console private NVRAM is valid and if invalid to test and initialize the NVRAM. Prior to checking the NVRAM, the IPT waits for power to stabilize by monitoring SCR<5>(POK). Once power is stable, the IPT then tests to see if the backup batteries failed during the power failure by checking SSCCR<31>(BLO). If the batteries failed, then the IPT will initialize certain non-volatile data, such as the default boot device, to a known state. In any case, the IPT then initializes other data structures and performs a processor initialization. If the the mode switch is set to test, the IPT then tests the console serial line as described in Section 4.1.3.

NOTE
All IPT failures are considered fatal, and the KA650 will appear to *hang* with a value on the LEDs indicating the point of failure. Refer to Table 4–2 for the meaning of the LEDs.

4.1.2.2 Locating a Console Device

After the IPT has completed successfully, the firmware attempts to locate a console device and find out what type of device it is. Normally, this is the device attached to the console serial line. In this case, the firmware will send out a device attributes escape sequence to the console serial line to determine the type of terminal attached and the functions it supports. Terminals that do not respond to the device attributes request correctly are assumed to be hardcopy devices. If a QDSS device is present, it is used as the primary console device.

NOTE
If a QDSS device is present, it is assumed that the Q22-bus interface is working. At this point in the firmware the Q22-bus has not yet been tested. Any faults on Q22-bus devices may prevent correct operation of the console.

Once a console device has been found, the firmware displays the KA650 banner message, similar to that displayed below.

```
    KA650-A V5.3, VMB 2.7
```

The banner message contains the processor name, the version of the firmware, and the version of VMB. The letter code in the firmware version indicates whether the firmware is pre-field test ("X"), field test ("T") or an official release ("V"). The first digit indicates the major release number and the trailing digit indicates the minor release number.

Next, if the designated console device supports DEC mutlinational 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 Example 4–1 in Section 4.1.4.

After the language query, the firmware invokes the ROM-based diagnostics, and eventually displays the console prompt.

4.1.3 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. The purpose of this test is to verify that the console serial line connections from the KA650 through the H3600-SA panel are intact.

NOTE
An external loopback connector should be inserted in the serial line connector on the H3600-SA panel prior to cycling power to invoke this test.

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 power-up.

During the passive state (about 7 seconds long), the LEDs are set to 3.

If at any time the firmware detects an error (parity, framing, overflow, or no characters), the firmware hangs with a 6 on the LEDs.

4.1.4 Mode Switch Set to Query

If the mode switch is set to query (or the firmware detects that the battery failed during a power loss), the firmware queries the user for a language which is used for displaying critical system messages.

The language query menu is shown in Example 4–1. If no response is received within 30 seconds, the language defaults to English (United States/Canada).

NOTE
This action is only taken 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 this inquiry, the firmware proceeds as if the mode switch were set to normal, as described in Section 4.1.5.

```
 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):
```

Example 4–1 Language Selection Menu

4.1.5 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 message text, a *countdown* is displayed to indicate diagnostic test progress. A successful diagnostic countdown is shown in Example 4–2.

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

Example 4–2 Normal Diagnostic Countdown

In the case of diagnostic failures, a diagnostic register dump is performed similar to that shown in Example 4–3. Depending on the failure, the remaining diagnostics may execute and the countdown continue. For a detailed description of the register dump refer to Section 4.4.

```
Performing normal system tests.
40..39..38..37..36..35..34..

?34 2 08 FF 00 0000

P1=00000000  P2=00000003  P3=00000031  P4=00000011  P5=00002000
P6=FFFFFFFF  P7=00000000  P8=00000000  P9=00000000 P10=2005438F
r0=00114B98  r1=FFFFFFFF  r2=2005D2F0  r3=55555555  r4=AAAAAAAA
r5=00000000  r6=AAAAAAAA  r7=00000000  r8=00000000 ERF=80000180
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..
Normal operation not possible.
```

Example 4–3 Abnormal Diagnostic Countdown

If the diagnostics have successfully completed and halts are enabled, the firmware displays the console prompt, >>>, and enters console I/O mode. If the diagnostics have successfully completed and halts are disabled, the firmware attempts to boot an operating system (Example 4–4).

```
Loading system software.
No default boot device has been specified.
Devices:
-DUA0 (RD54)
-XQA0 (08-00-2B-05-85-02)
Device? [XQA0]:

(BOOT/R5:0 XQA0)


  2..
-XQA0
```

Example 4–4 Console Boot Display with no Default Boot Device

4.1.6 LED Codes

In addition to the console diagnostic countdown, a hexadecimal value is displayed by the LEDs on the H3600-SA panel. The same value is displayed by the four red LEDs on the KA650 module. 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 4–2 lists all LED codes and the associated actions which are performed at power-up. The LED code is changed before the corresponding test or action is performed.

Table 4–2 LED Codes

LED Value	Actions
F	Initial state on power-up, no code has executed
E	Entered ROM, some instructions have executed
D	Waiting for power to stabilize (POK)
C	SSC and ROM tests
B	CPU tests
A	FPA tests

Table 4–2 (Cont.) LED Codes

LED Value	Actions
9	CMCTL tests
8	Memory tests
7	CQBIC (Q22-bus) tests
6	Console loopback tests (optionally QDSS tests)
5	Board-level cache tests
4	Miscellaneous 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

4.2 Console Service

The KA650 is by definition halted, whenever the console program is running and the triple angle prompt, >>>, is displayed on the console terminal. When halted, the firmware provides most of the services of a standard VAX console.

4.2.1 Console Control Characters

In console I/O mode, several characters have special meanings.

- Return — ends a command line. No action is taken on a command until after it is terminated by a carriage return. A null line terminated by a carriage return is treated as a valid, null command. No action is taken, and the console re-prompts for input. Carriage return is echoed as carriage return, line feed.
- Rubout — when the operator presses 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 hardcopy terminal. For hard copy terminals, when Rubout is pressed, the console echoes with a backslash (\), followed by the character being deleted. If the operator presses additional Rubouts, the additional characters deleted are echoed. When the operator types a non-rubout character, the console echoes another backslash, followed by the character typed. The result is to echo the characters deleted, surrounding them with backslashes.

```
For example:

        The operator types: EXAMI;E<Rubout><Rubout>NE<Return>

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

        The console sees the command line: EXAMINE<Return>
```

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

The console does not delete characters past the beginning of a command line. If the operator presses more Rubouts than there are characters on the line, the extra rubouts are ignored. If a Rubout is pressed on a blank line, it is ignored.

- Ctrl C — causes the console to echo ^C and to abort processing of a command. Ctrl C has no effect as part of a binary load data stream. Ctrl C clears Ctrl S, and reenables output stopped by Ctrl O.
- Ctrl O — causes the console to throw away transmissions to the console terminal until the next Ctrl O is entered. Ctrl O is echoed as ^O<CR> when it disables output, but is not echoed when it reenables output. Output is reenabled if the console prints an error message, or if it prompts for a command from the terminal. Displaying a REPEAT command does not reenable output. When output is reenabled for reading a command, the console prompt is displayed. Output is also enabled Ctrl S.
- Ctrl Q — resumes output to the console terminal. Additional Ctrl Qs are ignored. Ctrl S and Ctrl Q are not echoed.
- Ctrl S — stops output to the console terminal until Ctrl Q is pressed. Ctrl S and Ctrl Q are not echoed.
- Ctrl U — the console echoes ^U<CR>, and deletes the entire line. If Ctrl U is pressed on an empty line, it is echoed, and the console prompts for another command.

- Ctrl R — causes the console to echo <CR><LF> followed by the current command line. This function can be used to improve the readability of a command line that has been heavily edited. When Ctrl C is pressed as part of a command line, the console deletes the line as it does with Ctrl U.
- BREAK - If the console is in console I/O mode, BREAK is equivalent to Ctrl C, but 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, since 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.

Note that control codes from 128 to 159, the C1 control codes, cannot be entered by any present DIGITAL terminal. The character with code 7 and the character with code 135 will both echo as ^G.

4.2.2 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 character.

Multiple adjacent spaces and tabs are treated as a single space by the console. Leading and trailing spaces and tabs are ignored. Tabs are echoed as spaces.

Command qualifiers can appear after the command keyword, or after any symbol or number in the command. A qualifier is any contiguous set of non whitespace characters that is started with a slash (ASCII code 47 decimal).

All numbers (addresses, data, counts) are in hexadecimal. Note, though, that symbolic register names number the registers in decimal. The console does not distinguish between upper and lower case either in numbers or in commands; both are accepted.

4.2.3 Console Command Keywords

The KA650 firmware implements a variant of the VAX SRM console command set. The only commands defined in the VAX SRM and not supported by the KA650 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 KA650 console is similar to other VAX consoles. Table 4–3 lists command and qualifier keywords.

Table 4–3 Command, Parameter, and Qualifier Keywords

Command Keywords		
Processor Control	**Data Transfer**	**Console Control**
B*OOT	D*EPOSIT	CONF*IGURE
C*ONTINUE	E*XAMINE	F*IND
H*ALT	M*OVE	R*EPEAT
I*NITIALIZE	SEA*RCH	SET
N*EXT	X	SH*OW
S*TART		T*EST
U*NJAM		!
SET & SHOW Parameter Keywords		
BO*OT	BF*L(A)G	DE*VICE
ET*HERNET	H*OST	L*ANGUAGE
M*EMORY	Q*BUS	RL*V12
U*QSSP	VERS*ION	

Table 4–3 (Cont.) Command, Parameter, and Qualifier Keywords

Qualifier Keywords		
Data Control	**Address Space Control**	**Command Specific**
/B	/G	/IN*STRUCTION
/W	/I	/NO*T
/L	/P	/R5: or /
/Q	/V	/RP*B or /ME*M
/N:	/M	/F*ULL
/S*TEP:	/U	/DU*P or /MA*INTENANCE
/WR*ONG		/U*QSSP
		/DI*SK or /T*APE
		/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 4–5 following the command descriptions in Section 4.2.7.

4.2.4 Console Command Qualifiers

All qualifiers in the console command syntax are global. That is, they may appear in any place on the command line after the command keyword.

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

A summary of the qualifiers recognized by the KA650 console is provided in Table 4–6 following the command descriptions in Section 4.2.7.

4.2.5 Command Address Specifiers

Several commands take an address or addresses as arguments. In the context of the console, an address has two components, the address space, and the offset into that space. The console supports 6 address spaces: physical memory (/P qualifier), virtual memory (/V qualifier), general purpose registers (/G qualifier), internal processor registers (/I qualifier), protected memory (/U qualifier), and the PSL (/M qualifier).

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

The KA650 console supports symbolic references to addresses. A symbolic reference simultaneously defines the address space for a given symbol. Table 4–4 lists the symbolic addresses supported by the console grouped according to address space.

Table 4–4 Console Symbolic Addresses

Symbol	Address	Symbol	Address
	/G—General Purpose Registers		
R0	00	R11	0B
R1	01	R12	0C
R2	02	R13	0D
R3	03	R14	0E
R4	04	R15	0F
R5	05	AP	0C
R6	06	FP	0D
R7	07	SP	0E
R8	08	PC	0F
R9	09	PSL	—
R10	0A		
	/I - Internal Processor Registers		
PR$_KSP	00	PR$_SISR	15
PR$_ESP	01	PR$_ICCR	18
PR$_SSP	02	PR$_RXCS	20
PR$_USP	03	PR$_RXDB	21
PR$_ISP	04	PR$_TXCS	22
PR$_P0BR	08	PR$_TXDB	23
PR$_P0LR	09	PR$_TBDR	24
PR$_P1BR	0A	PR$_CADR	25
PR$_P1LR	0B	PR$_MCESR	26
PR$_SBR	0C	PR$_MSER	27
PR$_SLR	0D	PR$_SAVPC	2A
PR$_PCBB	10	PR$_SAVPSL	2B

Note: All symbolic values in this table are in hexadecimal.

Table 4–4 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address
/I - Internal Processor Registers			
PR$_SCBB	11	PR$_ IORESET	37
PR$_IPL	12	PR$_MAPEN	38
PR$_ASTLV	13	PR$_TBIA	39
PR$_SIRR	14	PR$_TBIS	3A
PR$_NICR	19	PR$_SID	3E
PR$_ICR	1A	PR$_TBCHK	3F
PR$_TODR	1B		
/P - Physical (VAX I/O Space)			
QBIO	2000 0000	QBMEM	3000 0000
QBMBR	2008 0010		
ROM	2004 0000	CACR	2008 4000
BDR	2008 4004		
DSCR	2008 0000	DSER	2008 0004
DMEAR	2008 0008	DSEAR	2008 000C
IPCR0	2000 1F40	IPCR1	2000 1F42
IPCR2	2000 1F44	IPCR3	2000 1F46
SSC_RAM	2014 0400	SSC_CR	2014 0010
SSC_CDAL	2014 0020	SSC_DLEDR	2014 0030
SSC_ AD0MAT	2014 0130	SSC_ AD0MSK	2014 0134
SSC_ AD1MAT	2014 0140	SSC_ AD1MSK	2014 0144
SSC_TCR0	2014 0100	SSC_TIR0	2014 0104
SSC_TNIR0	2014 0108	SSC_TIVR0	2014 010C
SSC_TCR1	2014 0110	SSC_TIR1	2014 0114
SSC_TNIR1	2014 0118	SSC_TIVR1	2014 011C
MEMCSR0	2008 0100	MEMCSR1	2008 0104
MEMCSR2	2008 0108	MEMCSR3	2008 010C
MEMCSR4	2008 0110	MEMCSR5	2008 0114
MEMCSR6	2008 0118	MEMCSR7	2008 011C
MEMCSR8	2008 0120	MEMCSR9	2008 0124
MEMCSR10	2008 0128	MEMCSR11	2008 012C

Table 4–4 (Cont.) Console Symbolic Addresses

Symbol	Address	Symbol	Address
	/P - Physical (VAX I/O Space)		
MEMCSR12	2008 0130	MEMCSR13	2008 0134
MEMCSR14	2008 0138	MEMCSR15	2008 013C
MEMCSR16	2008 0140	MEMCSR17	2008 0144
	Any Address Space		
"*"	The last location successfully referenced in an EXAMINE or DEPOSIT command.		
"+"	The location immediately following the last location successfully referenced in an EXAMINE or DEPOSIT command. For references to physical or virtual memory spaces, the location referenced is the last address, plus the size of the last reference (1 for byte, 2 for word, 4 for longword, 8 for quadword). For other address spaces, the address is the last address referenced plus one.		
"-"	The location immediately preceding the last location successfully referenced in an EXAMINE or DEPOSIT command. For references to physical or virtual memory spaces, the location referenced is the last address minus the size of this reference (1 for byte, 2 for word, 4 for longword, 8 for quadword). For other address spaces, the address is the last addressed referenced minus one.		
"@"	The location addressed by the last location successfully referenced in an EXAMINE or DEPOSIT command.		

4.2.6 References to Processor Registers and Memory

The KA650 console is implemented by macrocode executing from EPROM. Actual processor registers cannot be modified by the console command interpreter. When the console is entered, the console saves the processor registers in console memory and all command references to them are directed to the corresponding saved values, not to the registers themselves.

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

The following registers are saved by the console, and any direct reference to these registers will be intercepted by the console and the access will be to the saved copies:

- R0...R15, the general purpose registers
- PR$_IPL, the interrupt priority level register
- PR$_SCBB, the system control block base register
- PR$_ISP, the interrupt stack pointer
- PR$MAPEN, the memory management enable register

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

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

4.2.7 Console Commands

The following sections define the commands accepted by the console, when it is in console I/O mode. The following conventions are used to describe command syntax:

- [ ] denotes command elements that are optional.
- { } denotes a command element.
- ... denotes a list of command elements.

4.2.7.1 BOOT

Format :

BOOT [qualifier] [{boot_device}[:]]

Description :

The console initializes the processor and transfers execution to VMB. VMB attempts to boot the operating system from the specified device or the default boot device if none is specified. The console qualifies the bootstrap operation by passing a boot flag to VMB in R5. A more detailed description of the bootstrap process and how the default bootstrap device is determined is described in Section 4.3.

In the case where either the qualifiers or the device name is absent, then the corresponding default value is used. Explicitly stating the boot flags or the boot device overrides the current default value for the current boot request, but does not change the corresponding default value in NVRAM.

The default boot device and boot flags may be set in the following three ways:

1. The operating system may write a default boot device and flags into the appropriate locations in NVRAM (Section 4.7.3).
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 prompts the user for the default boot device, if any of the following conditions are met:
   - The power-up mode switch is set to query mode.
   - The console detects that the battery failed, and therefore the contents of 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 ESA0 or neither (1) nor (2) has been performed. Simply stated, the console will prompt the user on each and every powerup for a default boot device, until such a request has been satisfied.

On power-up if no default boot device is specified in NVRAM, the console issues a list of potential bootable devices and then prompts the user for a device name. If no device name is entered within 30 seconds, ESA0 is used. However, ESA0 does not become the default 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 4–1 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 can be any string, up to 17 characters long. Longer strings cause a VAL TOO BIG error message to be issued from the console. Otherwise the console makes no attempt at interpreting or validating the device name. The console converts the string to all upper case, and passes VMB a string descriptor to this device name in R0. A default boot device may be specified using the SET BOOT command and displayed with the SHOW BOOT command. The factory default device is the Ethernet port, ESA0.

Examples :

```
>>>show boot
DUA0
>>>show bflag
0
>>>b                        ! Boot using default boot flags and device.
(BOOT/R5:0 DUA0)

  2..
-DUA0

>>>bo xqa0                  ! Boot using default boot flags and specified
(BOOT/R5:0 XQA0)              device.

  2..
-XQA0

>>>boot/10                  ! Boot using specified boot flags and default
(BOOT/R5:10 DUA0)             device.

  2..
-DUA0

>>>boot /r5:220 xqa0        ! Boot using specified boot flags and device.
(BOOT/R5:220 XQA0)

  2..
-XQA0
```

4.2.7.2 CONFIGURE

Format :

CONFIGURE

Description :

CONFIGURE is similar to the VMS SYSGEN CONFIG utility. This feature provides information, that is typically available only with a running operating system, to simplify system configuration.

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.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>config
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
Numbers:
 1 to 255, default is 1
Device,Number? rqdx3,2
Device,Number? dhv11
Device,Number? qdss
Device,Number? tqk50
Device,Number? tqk70
Device,Number? exit
```

```
Address/Vector Assignments
-772150/154 RQDX3
-760334/300 RQDX3
-774500/260 TQK50
-760444/304 TQK70
-760500/310 DHV11
-777400/320 QDSS
>>>
```

4.2.7.3 CONTINUE

Format :

CONTINUE

Description :

The processor begins instruction execution at the address currently contained in the program counter. Processor initialization is not performed. The console enters program I/O mode. 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.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>continue
>>>
```

4.2.7.4 DEPOSIT

Format :

DEPOSIT [qualifier_list] {address} {data} [{data}...]

Description :

Deposits data into the address specified. If no address space or data size qualifiers are specified, the defaults are the last address space and data size used in a DEPOSIT, EXAMINE, MOVE or SEARCH command. After processor initialization, the default address space is physical memory, the default data size is a longword and the default address is zero. If conflicting address space or data sizes are specified, the console ignores the command and issues an error message.

Qualifiers :

- **/B** — The data size is byte.
- **/W** — The data size is word.
- **/L** — The data size is longword.
- **/Q** — The data size is quadword.
- **/G** — The address space is the general purpose register set, R0 through R15. The data size is always long.
- **/I** — The address space is internal processor registers (IPRs). These are the registers only accessible by the MTPR and MFPR instructions. The data size is always long.
- **/M** — The address space is the 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 forced to a value of 3 (ECC bits of 3 will always generate a double bit error).

Arguments :

- **{address}** — A long word address that specifies the first location into which data is deposited. The address can be any legal address specifier as defined in Section 4.2.5 and Table 4–4.
- **{data}** — The data to be deposited. If the specified data is larger than the deposit data size, the console ignores the command and issues an error response. If the specified data is smaller than the deposit data size, it is extended on the left with zeros.
- **[{data}]** — Additional data to be deposited (up to a maximum of 6 values).

Examples :

```
>>>d/p/b/n:1FF 0 0          ! Clear first 512 bytes of physical memory.

>>>d/v/l/n:3 1234 5         ! Deposit 5 into four longwords starting at
                              virtual memory address 1234.
>>>d/n:8 R0 FFFFFFFF        ! Loads GPRs R0 through R8 with -1.

>>>d/n:200 - 0              ! Starting at previous address, clear 513 bytes.

>>>d/l/p/n:10/s:200 0 8     ! Deposit 8 in the first longword of
                              the first 17 pages in physical memory.
>>>
```

4.2.7.5 EXAMINE

Format :

EXAMINE [qualifier_list] [{address}]

Description :

Examines the contents of the memory location or register specified by the address. If no address is specified, + is assumed. The display line consists of a single character address specifier, the hexadecimal physical address to be examined, and the examined data also in hexadecimal.

EXAMINE uses the same qualifiers as DEPOSIT. However, the /WRONG qualifier 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.

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.
- **/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 4.2.5 and Table 4–4. If no address is specified, + is assumed.

Examples :

```
>>>ex pc                                       ! Examine the PC.
  G 0000000F FFFFFFFC
>>>ex sp                                       ! Examine the SP.
  G 0000000E 00000200
>>>ex psl                                      ! Examine the PSL.
  M 00000000 041F0000
>>>e/m                                         ! Examine PSL another way.
  M 00000000 041F0000
>>>e r4/n:5                                    ! Examine R4 through R9.
  G 00000004 00000000
  G 00000005 00000000
  G 00000006 00000000
  G 00000007 00000000
  G 00000008 00000000
  G 00000009 801D9000
>>>ex pr$_scbb                                 ! Examine the SCBB, IPR 17.
  I 00000011 2004A000
>>>e/p 0                                       ! Examine local memory 0.
  P 00000000 00000000
>>>ex /ins 20040000                            ! Examine 1st byte 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)
>>>
```

4.2.7.6 FIND

Format :

FIND [qualifier-list]

Description :

The console searches main memory starting at address zero for a page-aligned 128 Kbyte segment of good memory, or a restart parameter block (RPB). If the segment or block is found, its address plus 512 is left in SP (R14). If the segment or block is not found, an error message is issued, and the contents of SP are preserved. If no qualifier is specified, /RPB is assumed.

Qualifiers :

- **/MEMORY** — Search memory for a page aligned block of good memory, 128 Kbytes in length. The search looks only at memory that is deemed usable by the bitmap. This command leaves the contents of memory unchanged.
- **/RPB** — Search all of physical memory for a restart parameter block. The search does not use the bitmap to qualify which pages are looked at. The command leaves the contents of memory unchanged.

Arguments :

None

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.
>>>
```

4.2.7.7 HALT

Format :

HALT

Description :

This command has no effect and is included for compatibility with other consoles.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>halt                                                  ! Pretend to halt.
>>>
```

4.2.7.8 HELP

Format :

HELP

Description :

This command has been included to help the console operator answer simple questions about command syntax and usage.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>help
Following is a brief summary of all the commands supported by the console:
        UPPERCASE  denotes a keyword that you must type in
        |          denotes an OR condition
        []         denotes optional parameters
        <>         denotes a field that must be filled in
                   with a syntactically correct value
Valid qualifiers:
    /B /W /L /Q /INSTRUCTION
    /G /I /V /P /M
    /STEP: /N: /NOT
    /WRONG /U
```

```
Valid commands:
    DEPOSIT [<qualifiers>] <address> [<datum> [<datum>]]
    EXAMINE [<qualifiers>] [<address>]
    MOVE [<qualifiers>] <address> <address>
    SEARCH [<qualifiers>] <address> <pattern> [<mask>]
    SET BFL(A)G <boot_flags>
    SET BOOT <boot_device>[:]
    SET HOST/DUP/UQSSP </DISK | /TAPE> <controller_number> [<task>]
    SET HOST/DUP/UQSSP <physical_CSR_address> [<task>]
    SET HOST/MAINTENANCE/UQSSP/SERVICE <controller_number>
    SET HOST/MAINTENANCE/UQSSP <physical_CSR_address>
    SET LANGUAGE <language_number>
    SHOW BFL(A)G
    SHOW BOOT
    SHOW DEVICE
    SHOW ETHERNET
    SHOW LANGUAGE
    SHOW MEMORY [/FULL]
    SHOW RLV12
    SHOW QBUS
    SHOW UQSSP
    SHOW VERSION
    HALT
    INITIALIZE
    UNJAM
    CONTINUE
    START <address>
    REPEAT <command>
    X <address> <count>
    FIND [/MEMORY | /RPB]
    TEST [<test_code> [<parameters>]]
    BOOT [/R5:<boot_flags> | /<boot_flags>] [<boot_device>[:]]
    NEXT [count]
    CONFIGURE
    HELP
>>>
```

4.2.7.9 INITIALIZE

Format :

INITIALIZE

Description :

A processor initialization is performed. The following registers are initialized, as specified in the *VAX Architecture Reference Manual*.

Register	**Initialized Value**
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	Disabled, all entries invalid
Instruction buffer	Unaffected
Console previous reference	Longword, physical, address 0
TODR	Unaffected
Main memory	Unaffected
General registers	Unaffected
Halt code	Unaffected
Bootstrap in progress flag	Unaffected
Internal restart in progress flag	Unaffected

The KA650 firmware performs the following additional initialization:

- The CDAL bus timer is initialized.
- The address decode and match registers are initialized.
- The programmable timer interrupt vectors are initialized.
- The BDR registers are read to determine the baud rate, and then the SSCCR is configured accordingly.
- All error status bits are cleared.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>init
>>>
```

4.2.7.10 MOVE

Format :

MOVE [qualifier-list] {src_address} {dest_address}

Description :

The console copies the block of memory starting at the source address to a block beginning at the destination address. Typically, this command is used with the /N: qualifier to transfer large blocks of data. The destination will correctly reflect the contents of the source, regardless of the overlap between the source and the data.

The MOVE command actually performs byte, word, longword, and quadword reads and writes as needed in the process of moving the data. Moves are only supported for the PHYSICAL and VIRTUAL address spaces.

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 is not 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 overridden 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 a value of 3.

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 4.2.5 and Table 4–4. If no address is specified, + is assumed.

Examples :

```
>>>ex /n:4 0                                  ! Observe destination.
  P 00000000 00000000
  P 00000004 00000000
  P 00000008 00000000
  P 0000000C 00000000
  P 00000010 00000000
>>>ex /n:4 200                                ! Observe source data.
  P 00000200 58DD0520
  P 00000204 585E04C1
  P 00000208 00FF8FBB
  P 0000020C 5208A8D0
  P 00000210 540CA8DE
>>>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
>>>
```

4.2.7.11 NEXT

Format :

NEXT {count}

Description :

The NEXT command causes the processor to "step" the specified number of macro instructions. If no count is specified, "single-step" is assumed. The console does not however enter spacebar step mode as described in the *VAX Architecture Reference Manual*, but rather returns to the console prompt.

The console uses the trace and trace pending bits in the PSL, and the SCB trace pending vector to implement the NEXT function. This creates the following restrictions on the usage of the NEXT command:

- If memory management is enabled, the NEXT command works if and only if the first page in SSC RAM is mapped somewhere in S0 (system) space.
- The NEXT command, due to the instructions executed in implementation, does not work where time critical code is being executed.
- The NEXT command elevates the IPL to 31 for long periods of time (milliseconds) while single stepping over several commands.
- Unpredictable results occur if the macro instruction being stepped over modifies the SCBB, or the trace trap entry. This means that the NEXT command cannot be used in conjunction with other debuggers. This also implies that the user should validate PR$_SCCB before using the NEXT command.

Qualifiers :

None

Arguments :

- **{count}** — A value representing the number of macro instructions to execute.

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...
  P 00001002   D6 INCL    R0
>>>n
  P 00001004   D1 CMPL    S^#05,R0
>>>n
```

```
  P 00001007   12 BNEQ    00001002
>>>n
  P 00001002   D6 INCL    R0
>>>n 5                                          ! ...or multiple step the
  P 00001004   D1 CMPL    S^#05,R0                program.
  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
>>>
```

4.2.7.12 REPEAT

Format :

REPEAT {command}

Description :

The console repeatedly displays and executes the specified command. Press Ctrl C to stop the repeating. Any valid console command can be specified for the command with the exception of the REPEAT command.

Qualifiers :

None

Arguments :

- **{command}** — A valid console command other than REPEAT.

Examples :

```
>>>repeat ex pr$_todr                                ! Watch the clock.
  I 0000001B 5AFE78CE
  I 0000001B 5AFE78D1
  I 0000001B 5AFE78FD
  I 0000001B 5AFE7900
  I 0000001B 5AFE7903
  I 0000001B 5AFE7907
  I 0000001B 5AFE790A
  I 0000001B 5AFE790D
  I 0000001B 5AFE7910
  I 0000001B 5AFE793C
  I 0000001B 5AFE793F
  I 0000001B 5AFE7942
  I 0000001B 5AFE7946
  I 0000001B 5AFE7949
  I 0000001B 5AFE794C
  I 0000001B 5AFE794F
  I 0000001B 5^C
>>>
```

4.2.7.13 SEARCH

Format :

SEARCH [qualifier_list] {address} {pattern} [{mask}]

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.

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 4.2.5 and Table 4–4. If no address is specified, + is assumed.
- **{pattern}** — The target data.
- **[{mask}]** — A longword containing the bits in the target which are to be masked out.

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.
```

4.2.7.14 SET

Format :

SET {parameter} {value}

Description :

Sets the indicated console parameter to the indicated value. The following are console parameters and their acceptable values:

Parameters :

- **BFL(A)G** — Set the default R5 boot flags. The value must be a hexadecimal number of up to 8 hex digits.

- **BOOT** — Set the default boot device. The value must be a valid device name as specified in Section 4.2.7.1 on the BOOT command.

- **HOST** — Invoke 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 — Use the DUP protocol to examine/modify parameters of a device on the Q22-bus. The optional value for SET HOST /DUP is a task name for the selected DUP driver to execute.

  NOTE
  The KA650 DUP driver only supports SEND DATA IMMEDIATE messages, and hence those devices which also support them.

  /UQSSP — Select the Q22-bus device using one of the following three methods.

  /DISK n — Specify the disk controller number, where **n** is from 0 to 255. (The resulting fixed address for **n** =0 is 20001468 and the floating rank for **n** >0 is 26.)
  /TAPE n — Specify the tape controller number, where **n** is from 0 to 255. (The resulting fixed address for **n** =0 is 20001940 and the floating rank for **n** >0 is 30.)
  csr_address — Specify the Q22-bus I/O page CSR address for the device.

  /MAINTENANCE — Use the MAINTENANCE protocol to examine/modify KFQSA EEPROM configuration parameters. Note that SET HOST /MAINTENANCE does not accept a task value.

  /UQSSP —

  /SERVICE n — Specify the KFQSA controller number **n** of a KFQSA in service mode, where n is from 0 to 3. (The resulting fixed address of a KFQSA in service mode is 20001910+4***n** .)
  csr_address — Specify the Q22-bus I/O page CSR address for the KFQSA.

- **LANGUAGE** — Set console language and keyboard type. 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

Qualifiers :

On a per parameter basis.

Arguments :

None

Examples :

```
>>>
>>>set bflag 220
>>>
>>>set boot dua0
>>>
>>>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
>>>
```

4.2.7.15 SHOW

Format :

SHOW {parameter}

Description :

Displays the console parameter indicated.

Parameters :

- **BFL(A)G** — Show the default R5 boot flags.

- **BOOT** — Show the default boot device.

- **DEVICE** — Show a list of all devices in the system.

- **ETHERNET** — Show the hardware Ethernet address for all Ethernet adapters that can be found. Displays as blank, if no Ethernet adapter is present.

- **LANGUAGE** — Show the console language and keyboard type. Refer to the corresponding SET LANGUAGE command for the meaning.

- **MEMORY** — Show main memory configuration on a board by board basis. Also report the addresses of bad pages, as defined by the bitmap.

  /FULL Additionally show the normally inaccessible areas of memory, such as, the PFN bitmap pages, the console scratch memory pages, and the Q22-bus scatter/gather map pages.

- **QBUS** — Show all Q22-bus I/O addresses that respond to an aligned word read. For each address, the console displays the address in the VAX I/O space in hex, the address as it would appear in the Q22-bus I/O space in octal, and the word data that was read in hex.

  This command may take several minutes to complete, so the user may want to issue a Ctrl C to terminate the command. The command disables the scatter/gather map for the duration of the command.

- **RLV12** — Show all RL01 and RL02 disks which appear on the Q22-bus.

- **UQSSP** — Show the status of all disks and tapes that can be found on the Q22-bus which support the UQSSP protocol. For each such disk or tape on the Q22-bus, the console displays the controller number, the controller CSR address, and the boot name and type of each device connected to the controller. The command does not indicate the bootability of the device.

  The device information is obtained from the media type field of the MSCP command GET UNIT STATUS. In the case where the node is not running, or is not capable of running, an MSCP server, then no device information is displayed.

- **VERSION** — Show the current version of the firmware.

Qualifiers :

On a per parameter basis.

Arguments :

None

Examples :

```
>>>
>>>show bflag
00000220
>>>
>>>show boot
DUA0
>>>
>>>show device
UQSSP Disk Controller 0 (772150)
-DUA0 (RF30)

UQSSP Disk Controller 1 (760334)
-DUB1 (RF30)

UQSSP Disk Controller 2 (760340)
-DUC4 (RF30)
```

```
UQSSP Disk Controller 3 (760344)
-DUD5 (RF30)

Ethernet Adapter
-XQA0 (08-00-2B-0B-82-29)

>>>
>>>show Ethernet
Ethernet Adapter

-XQA0 (08-00-2B-0B-82-29)

>>>
>>>show language
English (United States/Canada)
>>>
>>>show memory
Memory 0: 00000000 to 003FFFFF, 4 MBytes, 0 bad pages

Total of 4 MBytes, 0 bad pages, 98 reserved pages
>>>
>>>show memory /full
Memory 0: 00000000 to 003FFFFF, 4 MBytes, 0 bad pages

Total of 4 MBytes, 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 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
KA650-A V5.3, VMB 2.6
>>>
```

4.2.7.16 START

Format :

START [{address}]

Description :

The console starts instruction execution at the specified address. If no address is given, the current PC is used. If memory mapping is enabled, macro instructions are executed from virtual memory, and the address is treated as a virtual address. The START command is equivalent to a DEPOSIT to PC, followed by a CONTINUE. No INITIALIZE is performed.

Qualifiers :

None

Arguments :

- **[{address}]** — The address at which to begin execution. This is loaded in the user's PC.

Examples :

```
>>>start  1000
```

4.2.7.17 TEST

Format :

TEST [{test_number} [{test_arguments}]]

Description :

The console invokes a diagnostic test program specified by the test number. If a test number of 0 is specified, the power-up script is executed. The console accepts an optional list of up to five additional hexadecimal arguments.

A more detailed explanation of the diagnostics may be found in Section 4.4.

Qualifiers :

None

Arguments :

- **{test_number}** — A two digit hexadecimal number specifying the test to be executed.
- **{test_arguments}** — Up to five additional test arguments. These arguments are accepted but no meaning is attached to them by the console. For the interpretation of these arguments, consult the test specification for each individual test.

Examples :

```
>>>                 ! Execute the power-up diagnostic script
>>>                 ! Warning...this has the same affect as a power-up!
>>>t 0

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

>>>
>>>                 ! List all of the diagnostic tests.
>>>
>>>t 9e
```

Test #	Address	Name	Parameters
	2004 BE00	De_SCB	
C6	2004 D4D8	SSC_powerup	*********
C7	2004 D59A	CBTCR timeout	***
34	2004 D654	ROM logic test	*
33	2004 D71C	CMCTL_powerup	*
32	2004 D764	CMCTL regs	MEMCSR0_addr *********
91	2004 D888	CQBIC_powerup	**
90	2004 D918	CQBIC regs	*
80	2004 D971	CQBIC-memory	**********
60	2004 DE5F	Console serial	start_baud end_baud ******
61	2004 E1AC	Console QVSS	mark_not_present ***
62	2004 E254	console QDSS	mark_not_present selftest_r0 selftest_r1 ****
63	2004 E4DC	QDSS self-test	input_csr selftest_r0 selftest_r1 ******
51	2004 E642	CFPA	*****
52	2004 E82E	Prog timer	which_timer wait_time_us ***
53	2004 EAF4	TOY clock	repeat_test_250ms_ea Tolerance ********
55	2004 ED55	Interval timer	*
5A	2004 EDCC	VAX CMCTL CDAL	dont_report_memory_bad repeat_count *
45	2004 EEDC	cache_mem_cqbic	start_addr end_addr addr_incr ****
46	2004 F1C4	Cache1_diag_md	addr_incr *********
9E	2004 F7EE	List diags	*
81	2004 F814	MSCP-QBUS test	IP_csr ******
82	2004 F9D6	DELQA	device_num_addr ****
C1	2004 FBB1	SSC RAM	*
C2	2004 FD78	SSC RAM ALL	*
C5	2004 FEE8	SSC regs	*
54	2004 FFD9	Virtual mode	******
36	2005 0258	cache2_memory	start_addr end_addr addr_incr *******
35	2005 07D8	Cach2_integrty	start_addr end_addr addr_incr *******
44	2005 0F3C	Cache_memory	addr_incr *********
43	2005 0F95	Cache1_Cache2	addr_incr *********
41	2005 12C4	Board Reset	***
42	2005 14B1	Check_for_intrs	***
31	2005 14F4	MEM_Setup_CSRs	**********
30	2005 1AFC	MEM_Bitmap	*** mark_Hard_SBEs ******
4F	2005 1E71	MEM_Data	start_add end_add add_incr cont_on_err ******
4E	2005 2032	MEM_Byte	start_add end_add add_incr cont_on_err ******
4D	2005 214A	MEM_Address	start_add end_add add_incr cont_on_err ******
4C	2005 22E2	MEM_ECC_Error	start_add end_add add_incr cont_on_err ******
4B	2005 2671	MEM_Maskd_Errs	start_add end_add add_incr cont_on_err ******
4A	2005 284B	MEM_Correction	start_add end_add add_incr cont_on_err ******
49	2005 2A5E	MEM_FDM_Logic	*** cont_on_err ******
48	2005 308C	MEM_Addr_shrts	start_add end_add * cont_on_err pat1 pat2 ***
47	2005 36C6	MEM_Refresh	start end incr cont_on_err time_seconds *****
40	2005 3855	MEM_Count_Errs	First_board Last_board Soft_errs_allowed ****
9D	2005 3BB7	Utilities	Expnd_err_msg get_mode init_LEDs clr_ps_cnt
9C	2005 3CBB	List CPU regs	*

```
9F  2005 4271  Create script   ******

>>>
>>>              ! Show the diagnostic state.
>>>
>>>t fe

 bitmap=00BF3400, length=0C00, checksum=007E
 busmap=00BF8000
 return_stack=201406A4
 subtest_pc=2004EBB0
 timeout=00000001, error=00, de_error=00
 de_error_vector=00, severity_code=02, total_error_count=0000
 previous_error=00000000, 00000000, 00000000, 00000000, 00000000
 last_exception_pc=2004EBDA
 flags=21FFFD7F, test_flags=20
 highest_severity=00
 led_display=06
 console_display=00
 save_mchk_code=80, save_err_flags=000000
 parameter_1=00000000  2=00000000  3=00000000  4=00000000  5=00000000
 parameter_6=00000001  7=00000000  8=2004EBE0  9=00000000 10=20056056

>>>
>>>              ! Display the CPU registers.
>>>
>>>t 9c
TOY  =76BA1D75  ICCS =00000000
TCR0 =00000000  TIR0 =00000000  TNIR0=00000000  TIVR0=00000078
TCR1 =00000001  TIR1 =02BD7971  TNIR1=0000000F  TIVR1=0000007C
RXCS =00000000  RXDB =0000000D  TXCS =00000000  TXDB =00000030
MSER =00000000  CADR =0000000C
BDR  =FFFFFFD0  DLEDR=0000000C  SSCCR=00D45033  CBTCR=C0000004
SCR  =0000C000  DSER =00000080  QBEAR=0000000F  DEAR =00000000
QBMBR=00000000  IPCRn=0020

MEM_FRU 1  MEMCSR_0=80000015   1=00000015   2=00000015   3=00000015
MEM_FRU 2  MEMCSR_4=80400016   5=80800016   6=00000016   7=00000016
MEM_FRU 3  MEMCSR_8=00000000   9=00000000  10=00000000  11=00000000
MEM_FRU 4  MEMCSR12=00000000  13=00000000  14=00000000  15=00000000
           MEMCSR16=00000044  17=0000203C
>>>
```

4.2.7.18 UNJAM

Format :

UNJAM

Description :

An I/O bus reset is performed. This is implemented by writing 1 to IPR 55.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>unjam
>>>
```

4.2.7.19 X - Binary Load and Unload

Format :

X {address} {count} <CR> {line_checksum}
{data} {data_checksum}

Description :

The X command is for use by automatic systems communicating with the console. It is not intended for use by operators.

The console loads or unloads (that is, writes to memory, or reads from memory) the specified number of data bytes, starting at the specified address through the console serial line, regardless of which device is serving as the system console.

If bit 31 of the count is clear, data is to be received by the 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 then the console issues an error message. If an error occurs during loading, the contents of the memory being loaded are UNPREDICTABLE.

Echo is suppressed during the receiving of the data string and checksums.

To avoid treating flow control characters from the terminal as valid command line checksums, all flow control is terminated at the reception of the carriage return terminating the command line.

It is possible to control the console serial line through the use of the control characters (Ctrl C, Ctrl S, Ctrl O) during a binary unload. It is not possible during a binary load, as all received characters are valid binary data.

Data being loaded with a binary load command must be received by the console at a rate of at least one byte every 60 seconds. The command checksum that precedes the data must be received by the console within 60 seconds of the carriage return that terminates the command line. The data checksum must be received within 60 seconds of the last data byte. If any of these timing requirements are not met the console aborts the transmission by issuing an error message and prompting for input.

The entire command, including the checksum, can be sent to the console as a single burst of characters at the console serial line's specified character rate. The console is able to receive at least 4 Kbytes of data in a single X command.

Qualifiers :

None

Arguments :

None

Examples :

None

4.2.7.20 ! - Comment

Format :

!

Description :

The comment command is used to document command sequences. The comment character can appear anywhere on the command line. All characters following the comment character are ignored.

Qualifiers :

None

Arguments :

None

Examples :

```
>>>! The console ignores this line.
>>>
```

4.2.8 Conventions for Tables 4–5 and 4–6

Tables Table 4–5 lists a complete summary of the console commands. Table 4–6 is a summary of the qualifiers recognized by the KA650 console.

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

- UPPERCASE denotes the command or qualifier keyword.
- {} denotes a mandatory item which must be syntactically correct.
- [ ] denotes an optional item.
- ! denotes an "or" condition.
- *boot_flags, count, size, address, & 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.
- *language_type* denotes the language value, from 1 to 15.
- *command* denotes a console command other than REPEAT.
- *data, pattern, & mask* denote hex values of the current size.
- *test_number* denotes hex byte test number.

Table 4–5 Console Command Summary

Command	Qualifiers	Argument	Other(s)
BOOT	/R5:{boot_flags} /{boot_flags}	[{boot_device}]	—
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	—	—

Table 4–5 (Cont.) Console Command Summary

Command	Qualifiers	Argument	Other(s)
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 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}	—
SHOW BFL(A)G	—	—	—
SHOW BOOT	—	—	—
SHOW ETHERNET	—	—	—
SHOW LANGUAGE	—	—	—
SHOW MEMORY	/FULL	—	—
SHOW QBUS	—	—	—
SHOW RLV12	—	—	—

Table 4–5 (Cont.) Console Command Summary

Command	Qualifiers	Argument	Other(s)
SHOW UQSSP	—	—	—
SHOW VERSION	—	—	—
START	—	{address}	—
TEST	—	{test_number}	[{parameters}]
UNJAM	—	—	—
X	—	{address}	{count}

Table 4–6 Console Qualifier Summary

Data Control	
/B	Byte, legal for memory references only
/W	Word, legal for memory references only
/L	Longword, the default for GPR and IPR references
/Q	Quadword, legal for memory references only
/N:{count}	Specify number of additional operations
/STEP:{size}	Override the default step incrementing size with the value specified for the current reference
/WRONG	Ignore ECC bits on reads. Use an ECC value of 3 on writes

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

Table 4–6 (Cont.) Console Qualifier Summary

Command Specific	
/INSTRUCTION	EXAMINE command only. Disassemble the instruction at address specified.
/NOT	SEARCH command only. Invert the sense of the match.
/R5:{boot_flags}, /{boot_flags}	BOOT command only. Specify a function bitmap to pass to VMB through R5. Refer to Figure 4–1 for a bit description of R5. Either form of the command is acceptable.
/RPB, /MEM	FIND command only. Search for valid RPB or good block of memory.
/DUP, /UQSSP, /DISK, /TAPE, /MAINTENANCE, /SERVICE	SET HOST command only. Refer to command description for usage.

4.3 Bootstrapping

Bootstrapping is the process of loading and transferring control to an operating system. The KA650 supports bootstrap of the following operating systems: VAX/VMS, Ultrix-32, and VAXELN. Additionally, the KA650 will boot MDM diagnostics and any user application image which conforms to the boot formats described herein.

On the KA650 a bootstrap occurs whenever a BOOT command is issued at the console or whenever the processor halts and the conditions specified in the Table 4–1 for automatic bootstrap are satisfied.

4.3.1 Boot Devices

The KA650 firmware passes the address of a descriptor of the boot device name to VMB through R0. The device name used for the bootstrap operation is one of the following:

- XQA0, if no default boot device has been specified
- The default boot device specified at initial power-up or through 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 all upper case, and passes VMB the address of a string descriptor for the device name in R0.

Table 4–7 correlates the boot device names expected in a BOOT command with the corresponding supported devices.

Table 4–7 KA650 Supported Boot Devices

	Boot Name[1]	Controller Type	Device Type(s)
Disk:			
	DUcn	RQDX3 MSCP	RD52, RD53, RD54, RX33, RX50
		KDA50 MSCP	RA70, RA80, RA81, RA82, RA90
		KFQSA MSCP	RF30, RF71
		KLESI	RC25
	DLcn	RLV12	RL01, RL02
Tape:			
	MUcn	TQK50 MSCP	TK50
		TQK70 MSCP	TK70
		KFQSA MSCP	
		KLESI	TU81E
Network:			
	XQcn	DEQNA	—
		DELQA	—
		DESQA	—

[1] Boot device names consist of minimally a two letter device code, followed by a single character controller letter (A...Z), and terminating in a device unit number (0...65535). DSSI device names may optionally include a node prefix, consisting of either a node number (0...7) or a node name (a string of up to 8 characters), terminating in a "$".

Table 4–7 (Cont.) KA650 Supported Boot Devices

	Boot Name[1]	Controller Type	Device Type(s)
PROM:			
	PRA0	MRV11	—

[1] Boot device names consist of minimally a two letter device code, followed by a single character controller letter (A...Z), and terminating in a device unit number (0...65535). DSSI device names may optionally include a node prefix, consisting of either a node number (0...7) or a node name (a string of up to 8 characters), terminating in a "$".

NOTE
Table 4–7 presents a definitive list of boot devices which the KA650 supports. However, the KA650 will likely boot other devices which adhere to the MSCP standards.

4.3.2 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 4–1 shows the location of the boot flags used by VMB in the boot flag longword and Table 4–8 describes each flag's function.

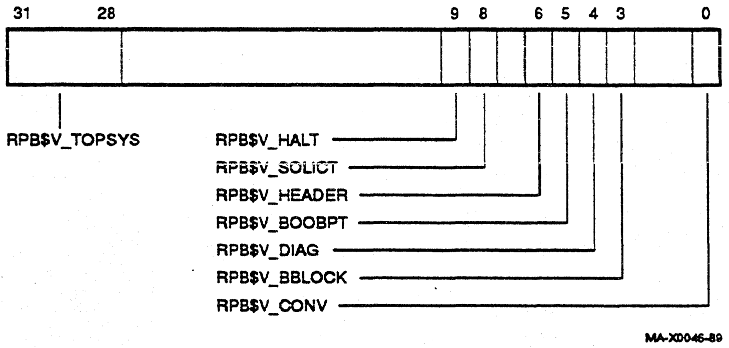


Figure 4–1 VMB Boot Flags

Table 4–8 VMB Boot Flags

Field	Name	Description
0	RPB$V_CONV	Conversational bootstrap
3	RPB$V_ BBLOCK	Secondary bootstrap from bootblock. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the bootblock format. If in conformance, the block is executed to continue the bootstrap. No attempt to perform a Files-11 bootstrap is made.
4	RPB$V_DIAG	Diagnostic bootstrap. When this bit is set, the load image requested over the network is [SYS0.SYSMAINT]DIAGBOOT.EXE.
5	RPB$V_ BOOBPT	Bootstrap breakpoint. If this flag is set, a breakpoint instruction is executed in VMB and control is transferred to XDELTA prior to boot.
6	RPB$V_ HEADER	Image header. If this bit is set, VMB transfers control to the address specified by the file's image header. If this bit is not set, VMB transfers control to the first location of the load image.
8	RPB$V_SOLICT	File name solicit. When this bit is set, VMB prompts the operator for the name of the application image file. A maximum of a 39 character file specification is permitted.
9	RPB$V_HALT	Halt before transfer. When this bit is set, VMB halts before transferring control to the application image.
31:28	RPB$V_ TOPSYS	This field can be any value from 0 through F. This flag changes the top level directory name for the system disks with multiple operating systems. For example, if TOPSYS is 1, the top level directory name is [SYS1...].

4.3.3 Preparing for the Bootstrap

Prior to dispatching to the primary bootstrap (VMB), the firmware initializes the system to a known state. The initialization sequence is the following:

1. Check CPMBX<2>(BIP). If it is set, bootstrap fails.
2. If this is an automatic bootstrap, print the message "Loading system software" on the console terminal.
3. Validate the boot device name. If none exists, supply a list of available devices and prompt user for a device. If no device is entered within 30 seconds, use XQA0.
4. Write a form of this BOOT request including the active boot flags and boot device on the console, for example "(BOOT/R5:0 DUA0)".
5. Set CPMBX<2>(BIP).
6. Initialize the Q22-bus scatter/gather map.
   a. Clear IPCR<5>(LMEAE).
   b. Perform an UNJAM.
   c. 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.
   d. Perform an INIT.
   e. Set IPCR<5>(LMEAE).
7. Validate the PFN bitmap. If invalid, rebuild it.
8. Search for a 128 Kbyte contiguous block of good memory as defined by the PFN bitmap. If 128 Kbyte cannot be found, the bootstrap fails.

9. Initialize the general purpose registers.

    R0 = address of descriptor of the boot device name or 0 if none specified
    R2 = length of PFN bitmap in bytes
    R3 = address of PFN bitmap
    R4 = time of day from PR$_TODR at power-up
    R5 = boot flags
    R10 = halt PC value
    R11 = halt PSL value (without halt code and map enable)
    AP = halt code
    SP = base of 128 Kbyte good memory block + 512
    PC = base of 128 Kbyte good memory block + 512
    R1, R6, R7, R8, R9, FP = 0

10. Copy the VMB image from EPROM to local memory beginning at the base of the 128 Kbyte good memory block + 512.

11. Exit from the firmware to memory resident VMB.

On entry to VMB the processor is running at IPL 31 on the interrupt stack with memory management disabled.

4.3.4 Primary Bootstrap, VMB

Virtual memory boot (VMB) is the primary bootstrap for booting VAX processors. On the KA650, VMB is resident in the firmware and is copied into main memory before control is transferred to it. VMB then loads the secondary bootstrap image and transfers control to it.

NOTE
In certain cases, such as VAXELN, VMB actually loads the operating system directly. However, for the purpose of this discussion *secondary bootstrap* refers to any VMB loadable image.

VMB inherits a well defined environment and is responsible for further initialization. The following summarizes the operation of VMB:

1. Initialize a two-page SCB on the first page boundary above VMB.
2. Allocate a three-page stack above the SCB.
3. Initialize the restart parameter block (RPB).
4. Initialize the secondary bootstrap argument list.
5. If not a PROM boot, locate a minimum of 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.

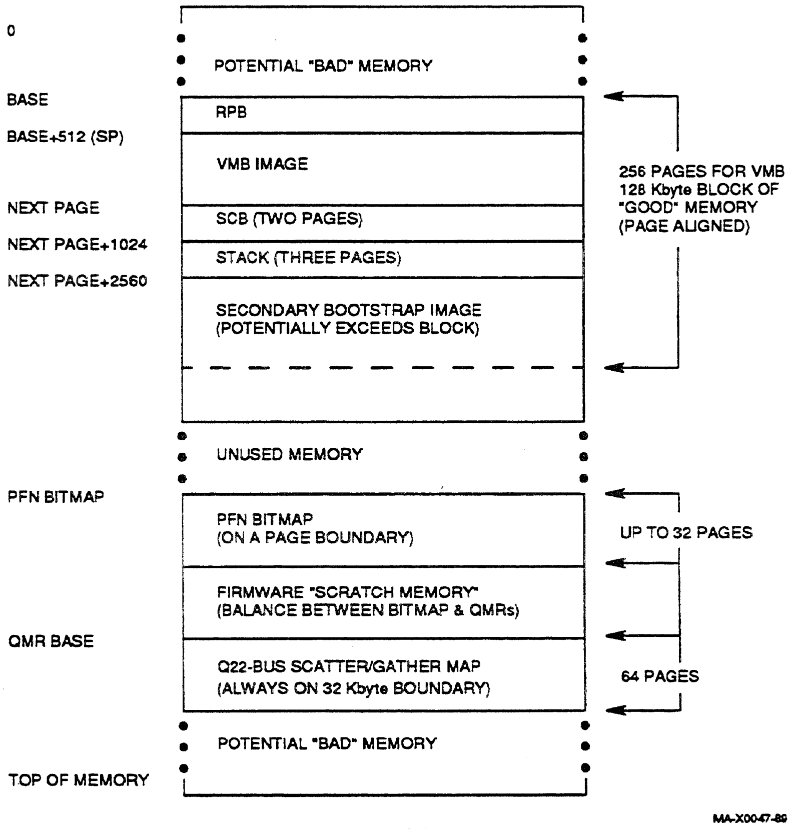
NOTE
VMB makes no assumptions about the location of Q22-bus memory. However, VMB searches through the Q22-bus map registers (QMRs) for the first QMR marked *valid*. VMB requires minimally 3 and maximally 129 contiguous *valid* maps to complete a bootstrap operation. If the search exhausts all map registers or there are fewer than the required number of *valid* maps, a bootstrap cannot be performed. It is recommended that a suitable block of Q22-bus memory address space be available (unmapped to other devices) for proper operation.

Below is a sample console display of a successful automatic bootstrap:

```
Loading system software.
(BOOT/R5:0 DUA0)

 2..
-DUA0
 1..0..
```

After a successful bootstrap operation, control is passed to the secondary bootstrap image with the memory layout as shown in Figure 4–2.



0
BASE
BASE+512 (SP)
NEXT PAGE
NEXT PAGE+1024
NEXT PAGE+2560
PFN BITMAP
QMR BASE
TOP OF MEMORY
POTENTIAL "BAD" MEMORY
RPB
VMB IMAGE
SCB (TWO PAGES)
STACK (THREE PAGES)
SECONDARY BOOTSTRAP IMAGE (POTENTIALLY EXCEEDS BLOCK)
UNUSED MEMORY
PFN BITMAP (ON A PAGE BOUNDARY)
FIRMWARE "SCRATCH MEMORY" (BALANCE BETWEEN BITMAP & QMRs)
Q22-BUS SCATTER/GATHER MAP (ALWAYS ON 32 Kbyte BOUNDARY)
POTENTIAL "BAD" MEMORY
256 PAGES FOR VMB 128 Kbyte BLOCK OF "GOOD" MEMORY (PAGE ALIGNED)
UP TO 32 PAGES
64 PAGES
MA-X0047-89

Figure 4–2 Memory Layout at VMB Exit

In the event that an operating system has an extraordinarily large secondary bootstrap which overflows the 128 Kbytes of good memory, VMB loads the remainder of the image in memory above the good block. However, if there are not enough contiguous good pages above the block to load the remainder of the image, the bootstrap fails.

4.3.5 Device-Dependent Bootstrap Procedures

As mentioned earlier, the KA650 supports bootstrapping from a variety of boot devices. The following sections describe the various device-dependent boot procedures.

4.3.5.1 Disk and Tape Bootstrap Procedure

The disk and tape bootstrap supports Files-11 lookup (supporting only the ODS level 2 file structure) or the boot block mechanism (used in PROM boot also). Of the standard DEC operating systems VMS and ELN use the Files-11 bootstrap procedure and Ultrix-32 uses the boot block mechanism.

VMB first attempts a Files-11 lookup, unless the RPB$V_BBLOCK boot flag is set. If VMB determines that the designated boot disk is a Files-11 volume, it searches the volume for the designated boot program, usually [SYS0.SYSEXE]SYSBOOT.EXE. However, VMB can request a diagnostic image or prompt the user for an alternate file specification (Section 4.3.2). 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 is set, the boot block mechanism proceeds as follows:

1. Read logical block 0 of the selected boot device (this is the boot block).
2. Validate that the contents of the boot block conform to the boot block format (Figure 4–3).
3. Use the boot block to find and read in the secondary bootstrap.
4. Transfer control to the secondary bootstrap image, just as for a Files-11 boot.

The format of the boot block must conform to that shown in Figure 4–3.

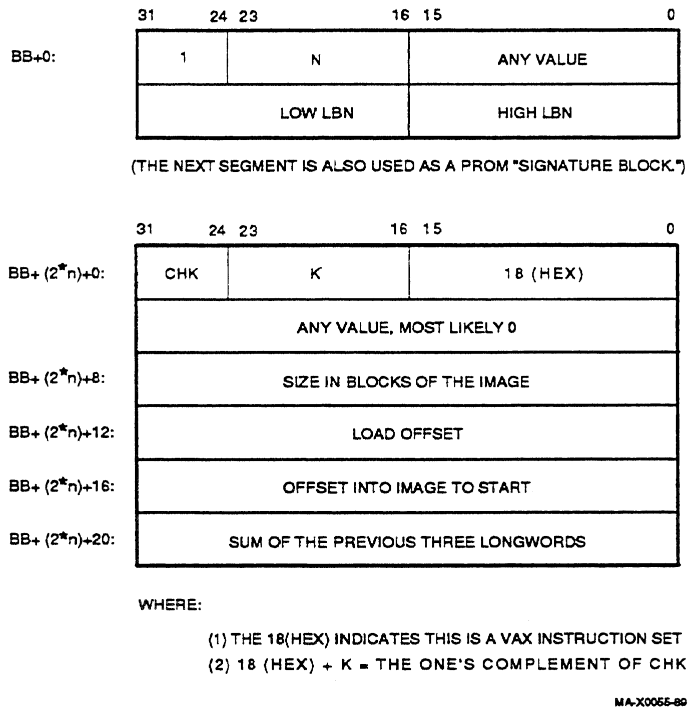

31 24 23 16 15 0
BB+0:
1
N
ANY VALUE
LOW LBN
HIGH LBN
(THE NEXT SEGMENT IS ALSO USED AS A PROM "SIGNATURE BLOCK.")
31 24 23 16 15 0
BB+ (2*n)+0:
CHK
K
18 (HEX)
ANY VALUE, MOST LIKELY 0
BB+ (2*n)+8:
SIZE IN BLOCKS OF THE IMAGE
BB+ (2*n)+12:
LOAD OFFSET
BB+ (2*n)+16:
OFFSET INTO IMAGE TO START
BB+ (2*n)+20:
SUM OF THE PREVIOUS THREE LONGWORDS
WHERE:
(1) THE 18(HEX) INDICATES THIS IS A VAX INSTRUCTION SET
(2) 18 (HEX) + K = THE ONE'S COMPLEMENT OF CHK
MA-X0055-89

Figure 4–3 Boot Block Format

4.3.5.2 PROM Bootstrap Procedure

The PROM bootstrap uses a variant of the boot block mechanism. VMB searches through Q22-bus memory on 16 Kbyte boundaries for a valid PROM *signature block*, the second segment of the boot block defined in Figure 4–3.

At each boundary, VMB:

1. Validates the readability of that Q22-bus memory page.
2. If readable, check to see if it contains a valid PROM signature block.

If verification passes, the PROM image will be copied into main memory and VMB will transfer control to that image at the offset specified in the PROM bootblock. If not, the next page will be tested.

NOTE
It is not necessary that the boot image actually reside in PROM. Any boot image in Q22-bus memory space with a valid signature block on a 16 Kbyte boundary is a candidate.

The PROM image is copied into main memory in 127 page "chunks" until the entire PROM is moved. All destination pages beyond the primary 128 Kbyte block are verified to make sure they are marked good in the PFN bitmap. The PROM must be copied contiguously and if all required pages cannot fit into the memory immediately following the VMB image, the boot fails.

4.3.5.3 Network Bootstrap Procedure

Whenever a network bootstrap is selected on a KA650, VMB makes continuous attempts to boot from the network. VMB uses the DNA maintenance operations protocol (MOP) as the transport protocol for network bootstraps and other network operations. Once a network boot has been invoked, VMB turns on the designated network link and repeats load attempts, until either a successful boot occurs or it is halted from the operator console.

The KA650 supports the load of a standard operating system, a diagnostic image, or a user-designated program through 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. Hence, if both bits are set, then the solicited file is requested. (Refer to Figure 4–1 for the usage of these bits.)

NOTE
VMB accepts a maximum of a 39 character file specification for solicited boots. If the network server is running VMS the following defaults apply to the file specification: the directory MOM$LOAD:, and an extension .SYS. Therefore, the 39 character file specification need only consist of the filename if the default directory and extension attributes are used.

The KA650 VMB uses the MOP program load sequence for bootstrapping the module and the MOP "dump/load" protocol type for load related message exchanges. The MOP message types used in the exchange are listed in Table 4–9.

VMB, the requester, starts by sending a REQ_PROGRAM message in the appropriate envelope to the MOP dump/load multicast address. 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 either:

- MEM_LOAD message, while there is still more to load
- MEM_LOAD_w_XFER, if it is the end of the image
- PARAM_LOAD_w_XFER, if it is the end of the image and operating system parameters are required

The load number field in the load messages is used to synchronize the load sequence. At the beginning of the exchange, both the requester and server initialize the load number. The requester only increments the load number if a load packet has been successfully received and loaded. This forms the acknowledge to each exchange. The server will resend a packet with a specific load number, until it sees 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.

During the boot sequence if no response is made to the REQ_PROGRAM message in the current time-out limit, the time-out limit is increased linearly up to a maximum of about 4 minutes. The initial time-out limit is 4 seconds.

4.3.5.4 Network Listening

While the KA650 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 8 to 12 minute intervals. In particular, this listener supplements the MOP functions of the VMB load requester typically found in bootstrap firmware and supports the following:

- A remote console server that generates unsolicited SYSTEM_ID messages every 8 to 12 minutes and solicited SYSTEM_ID messages in response to REQUEST_ID messages, as well as, COUNTERS messages in response to REQ_COUNTERS messages.
- A loopback server that responds to Ethernet LOOPBACK messages by echoing the message to the requester.
- An IEEE 802.2 responder which replies to both XID and TEST messages.

During network operation the firmware listens only to MOP "Load/Dump," MOP "Remote Console," and Ethernet "Loopback Assistance" messages protocols directed to the Ethernet physical address of the node. All other Ethernet protocols are filtered by the network device driver. Additionally, IEEE 802.3 messages are also processed by the network listener.

The MOP functions and message types which are supported by the KA650 are summarized in the following tables.

Table 4–9 KA650 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	—		—

[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.

Table 4–9 (Cont.) KA650 Network Maintenance Operations Summary

Function	Role	Transmit		Receive
		MOP Ethernet and IEEE 802.3 Messages [1]		
	Server	SYSTEM_ID[3]	in response to	REQUEST_ID
		COUNTERS	in response to	REQ_ COUNTERS 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.

[3]SYSTEM_ID messages are sent out every 8 to 12 minutes to the remote console multicast address and on receipt of a REQUEST_ID message they are sent to the initiator.

[4]LOOPED_DATA messages are sent out in response to LOOP_DATA messages. These messages are actually in Ethernet LOOP TEST format, not in MOP format, and when sent in Ethernet frames omit the additional length field (padding is disabled).

[5]IEEE 802.2 support of XID and TEST is limited to Class 1 operations.

4.4 Diagnostics

The ROM based diagnostics constitute a major portion of the firmware on the KA650 and consist of an initial power-up test and a series of functional component diagnostics. These diagnostics run automatically on power-up and can be executed interactively as a whole, or as individual tests using the TEST command (Section 4.2.7.17). This section summarizes their operation.

The purpose of the ROM based diagnostics is multifaceted:

1. During power-up, the diagnostics determine if enough of the KA650 is working to allow the console to run.
2. During the manufacturing process, the diagnostics verify that the board is correctly built.
3. In the field, the diagnostics verify that the board is operational, and to report all detected errors.
4. The diagnostics allow sophisticated users and Field Service technicians to run individual diagnostics interactively, with the intent of isolating errors to the FRU.

To accommodate these requirements, the diagnostics have been designed as a collection of individual parameterized tests. A data structure, called a *script*, and a program, called the *diagnostic executive*, orchestrate the running of these tests in the right order with the right parameters.

A script is a data structure that points to various tests. There are several scripts, one for the field, and several for manufacturing, depending on where on the manufacturing line the board is. Sophisticated users may also create their own scripts interactively. Additionally, the script contains other information as follows:

- What parameters need to be passed to the test.
- What is to be displayed, if anything, on the console.
- What is to be displayed, if anything, on the LED.
- What to do on errors (halt, loop, or continue).
- Where the tests may be run from. For example, there are certain tests that can only be run from the EPROM. Other tests are position independent code (PIC), and may be run from EPROM, or main memory in the interests of execution speed.

The diagnostic executive interprets scripts to determine what tests are to be run. There are several built in scripts on the KA650 that are used for manufacturing, power-up, and Field Service personnel. The diagnostic executive automatically invokes the correct script based on the current environment of the KA650. 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 errors 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.

4.4.1 Error Reporting

Before a console is established, the only error reporting is through the KA650's 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. Example 4–5 shows a typical error display.

```
?34 2 08 FF 00 0000                                                   (1)

P1=00000000  P2=00000003  P3=00000031  P4=00000011  P5=00002000       (2)
P6=FFFFFFFF  P7=00000000  P8=00000000  P9=00000000 P10=2005438F       (3)
r0=00114B98  r1=FFFFFFFF  r2=2005D2F0  r3=55555555  r4=AAAAAAAA       (4)
r5=00000000  r6=AAAAAAAA  r7=00000000  r8=00000000 ERF=80000180       (5)
```

Example 4–5 Diagnostic Register Dump

The numbers in parenthesis on the right side of Example 4–5 refer to lines of the display and are not a part of the diagnostic dump. The information on these lines is summarized as follows:

1. Test summary containing six hexadecimal fields.
    a. ?34, **test** identifies the diagnostic test.
    b. 2, **severity** is the level of a test failure, as dictated by the script. Failure of a severity level 2 test causes the display of this five-line error printout, and halts an autoboot to console I/O mode. An error of severity level 1 displays the first line of the error printout, but does not interrupt an autoboot. Most tests have a severity level of 2.

c. 08, **error** is a number, that in conjunction with listing files, isolates to within a few instructions where the diagnostic detected the error. This field is also called the subtestlog.

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. 00, **vector** is the SCB vector (if non-zero) through which an unexpected exception or interrupt trapped, when the **de_error** field indicates an unexpected exception or interrupt (FE or EF).

f. 0000, **count** is the number of previous errors that have occurred.

2. P1...P5 are the first five longwords of the diagnostic state. This is internal information that is used by repair personnel.
3. P6...P10 are the last five longwords of the diagnostic state.
4. R0...R4 are the first five GPRs at the moment the error was detected.
5. R5...R8 are additional GPRs and ERF is a diagnostic summary longword.

4.4.2 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.

4.4.3 Areas not Covered

The goal is to achieve the highest possible coverage on the KA650 and the memory boards. However, the testing of the KA650 while running with memory management turned on is minimal. Also, due to the way the firmware is implemented (a polled environment running at IPL 31), the testing of interrupts is not extensive.

These diagnostics are not intended to be used as system level tests. There are no tests that completely verify that access to the Q22-bus will work. Thus, a disk, a controller, the backplane, or portions of the CQBIC may be faulty, and the diagnostics may not detect the fault. Such a fault may result later as an inability to boot.

4.5 Operating System Restart

An operating system restart is the process of bringing up the operating system from a known initialization state following a processor halt. This procedure is often called *restart* or *warmstart*, and should not be confused with a processor restart which results in firmware entry.

On the KA650 a restart occurs if the conditions specified in Table 4–1 are satisfied.

To restart a halted operating system, the firmware searches system memory for the restart parameter block (RPB), a data structure constructed for this purpose by VMB. If a valid RPB is found, the firmware passes control to the operating system at an address specified in the RPB.

The firmware keeps a *restart in progress* (RIP) flag in CPMBX which it uses to avoid repeated attempts to restart a failing operating system. An additional RIP flag is maintained by the operating system in the RPB.

The firmware uses the following algorithm to restart the operating system:

1. Check CPMBX<3>(RIP). If it is set, restart fails.
2. Print the message "Restarting system software" on the console terminal.
3. Set CPMBX<3>(RIP).
4. Search for a valid RPB. If none is found, restart fails.
5. Check the operating system RPB$L_RSTRTFLG<0>(RIP) flag. If it is set, restart fails.

6. Write "0" on the diagnostic LEDs.
7. Dispatch to the restart address, RPB$L_RESTART, with :

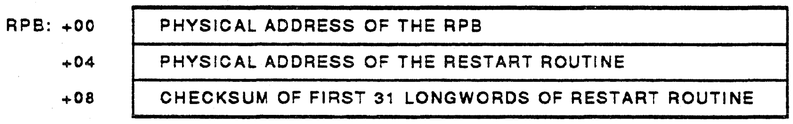
   SP = the physical address of the RPB plus 512
   AP = the halt code
   PSL = 041F 0000
   PR$_MAPEN = 0.

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.

4.5.1 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 4–4.



RPB: +00	PHYSICAL ADDRESS OF THE RPB
+04	PHYSICAL ADDRESS OF THE RESTART ROUTINE
+08	CHECKSUM OF FIRST 31 LONGWORDS OF RESTART ROUTINE

MA-X0056-89

Figure 4–4 RPB Signature Format

The firmware uses the following algorithm to find a valid RPB:

1. Search for a page of memory that contains its address in the first longword. If none is found, the search for a valid RPB has failed.
2. Read the second longword in the page (the physical address of the restart routine). If it is not a valid physical address, or if it is zero, return to step 1. The check for zero is necessary to ensure that a page of zeros does not pass the test for a valid RPB.
3. Calculate the 32 bit twos-complement sum (ignoring overflows) of the first 31 longwords of the restart routine. If the sum does not match the third longword of the RPB, return to step 1.
4. A valid RPB has been found.

4.6 Machine State on Power-Up

This section describes the state of the KA650 after a power-up halt.

The descriptions in this section assume a machine with no errors, that the machine has just been turned on and that only the power-up diagnostics have been run. The state of the machine is not defined if individual diagnostics are run or during any other halts other than a power-up halt (SAVPSL<13:8>(RESTART_CODE) = 3).

The following sections describe data structures that are guaranteed to be constant over future versions of the KA650 firmware. Placement and/or existence of any other structure(s) is not implied.

4.6.1 Main Memory Layout and State

The firmware tests and initializes the main memory on power-up. Figure 4–5 is a diagram of how main memory is partitioned after diagnostics.

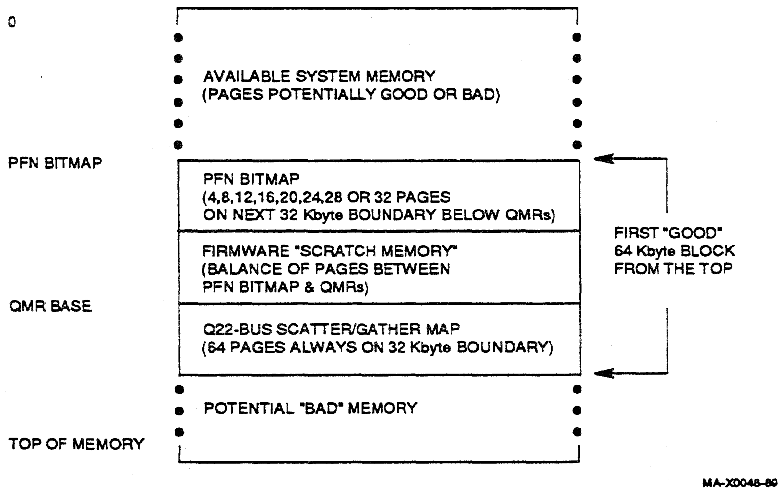

0
AVAILABLE SYSTEM MEMORY
(PAGES POTENTIALLY GOOD OR BAD)
PFN BITMAP
PFN BITMAP
(4,8,12,16,20,24,28 OR 32 PAGES
ON NEXT 32 Kbyte BOUNDARY BELOW QMRs)
FIRMWARE "SCRATCH MEMORY"
(BALANCE OF PAGES BETWEEN
PFN BITMAP & QMRs)
QMR BASE
Q22-BUS SCATTER/GATHER MAP
(64 PAGES ALWAYS ON 32 Kbyte BOUNDARY)
FIRST "GOOD"
64 Kbyte BLOCK
FROM THE TOP
POTENTIAL "BAD" MEMORY
TOP OF MEMORY
MA-X0048-89

Figure 4–5 Memory Layout after Power-Up Diagnostics

4.6.1.1 Reserved Main Memory

In order to build the scatter/gather map and the bitmap, the firmware attempts to find a physically contiguous page aligned 64 Kbyte block of memory at the highest possible address that has no multiple bit errors. Single bit errors are tolerated in this section.

Of the 64 Kbytes, the upper 32 Kbytes is dedicated to the Q22-bus scatter/gather map, as shown in Figure 4–5. Of the lower 32 Kbytes, up to 16 Kbytes at the bottom of the block is allocated to the page frame number (PFN) 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 (minimally 16 Kbytes) is allocated for the firmware.

4.6.1.2 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 a side effect of the memory tests on power-up. The bitmap always starts on a page boundary. The bitmap requires 1 Kbyte for every 4 Mbytes of main memory, hence, an 8 Mbyte system requires 2 Kbytes, 16 Mbyte requires 4 Kbytes, 32 Mbyte requires 8 Kbytes, and a 64 Mbyte requires 16 Kbytes. The bitmap does not map itself or anything above it. There may be memory above the bitmap which has both good and bad pages.

Each bit in the PFN bitmap corresponds to a page in main memory. There is a one to one 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.

4.6.1.3 Scatter/Gather Map

On power-up, the scatter/gather map is initialized by the firmware to map to the first 4 Mbytes of main memory. Main memory pages will not be mapped if there is a corresponding page in Q22-bus memory, or if the page is marked bad by the PFN bitmap.

On a processor halt other than power-up, the contents of the scatter/gather map is undefined, and is dependent on operating system usage.

Operating systems should not move the location of the scatter/gather map, and should access the map only on aligned longwords through the local I/O space of 2008 8000 to 2008 FFFC, inclusive. The Q22-bus map base register, (QMBR) is set up by the firmware to point to this area, and should not be changed by software.

4.6.1.4 Contents of Main Memory

The contents of main memory are undefined after the diagnostics have run. Typically, non-zero test patterns will be left in memory.

The diagnostics will scrub all of main memory, so that no power-up induced errors remain in the memory system. On the KA650 memory subsystem, the state of the ECC bits and the data bits are undefined on initial power-up. This can result in single and multiple bit errors if the locations are read before written because the ECC bits are not in agreement with their corresponding data bits. An aligned longword write to every location (done by diagnostics) eliminates all power-up induced errors.

4.6.2 CMCTL Registers

The KA650 firmware assigns bank numbers to CMCTL 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 MEMCSR register:

- MEMCSR<31> (bank enable bit). As the firmware always assigns banks in ascending order, knowing which banks are enabled is sufficient information to derive the bank numbers.
- MEMCSR<1:0> (bank usage). This field determines the size of the banks on the particular memory board.

Additional information should be captured from the MCSR16 and MCSR17 as needed.

4.6.3 First Level Cache

The first level cache is tested during the power-up diagnostics, flushed and then turned off. The cache is again turned on by the BOOT and the INIT command. Otherwise, the state of the first level cache is disabled.

4.6.4 Translation Buffer

The CPU translation buffer is tested by diagnostics on power-up, but not used by the firmware since it runs in physical mode. The translation buffer can be invalidated by using PR$_TBIA, IPR 57.

4.6.5 Second Level Cache

The second level cache is tested during the power-up diagnostics, flushed and then turned off. During a bootstrap, the second level cache is turned off before invoking VMB but not flushed. The second level cache is turned off, but not flushed, on an INIT command.

The second level cache should always be flushed before turning it on.

4.6.6 Halt Protected Space

Halt protected space is from 2004 0000 to 2005 FFFF, inclusive, for the 128 Kbytes of firmware on the KA650. The halt unprotected space is from 2006 0000 to 2008 FFFF.

The firmware always runs in halt protected space. When passing control to the bootstrap, the firmware exits the halt protected space, so if halts are enabled, and the halt line is asserted, the processor will then halt before booting.

The SSC decodes both spaces (256 Kbytes). That is, the ROMs appear twice in the address space. However, the halt protected space is set to 128 Kbytes, the size of the EPROMs.

4.7 Public Data Structures and Entry Points

This section describes public data structures and subroutine entry points that are public and are guaranteed to be compatible over future versions of the KA650 firmware.

4.7.1 Firmware EPROM Layout

The KA650 uses two 64 Kbyte EPROMs (128 Kbytes of total EPROM). Approximately 120 Kbytes of this is used for code, with the remaining reserved for future expansion and customer usage. There are two copies of the firmware, one in halt protected space, and one in halt unprotected space. Both copies are identical.

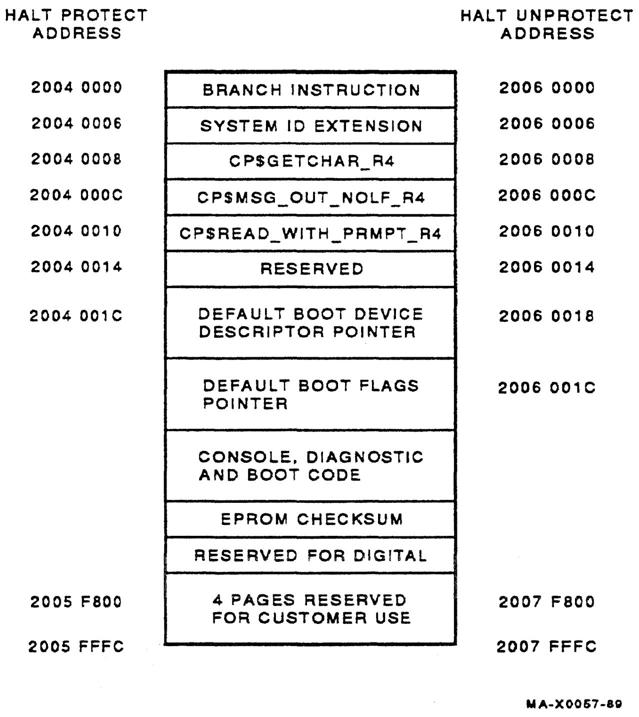


Figure 4–6 KA650 EPROM Layout

The first instruction executed on halts is a branch around the system ID extension (SIE) and the callback entry points. This allows the public data structures to reside in fixed locations in the EPROM.

The callback area entry points provide a simple interface to the currently defined console for VMB and secondary bootstraps. This is discussed further in the next section.

The EPROM checksum is a longword checksum from 2004 0000 to the checksum inclusive. The diagnostics use this to determine that the EPROMs can correctly be read.

The memory between the checksum and the 4-page user area at the end of the EPROMs is reserved for DIGITAL for future expansion of the KA650 firmware. The contents of this area is set to FF.

The 4 pages reserved for customer use are at the top of the EPROMs, and start at address 2005 F800 (halt protected space) or 2007 F800 (halt unprotected space). These areas are not burned and may be reburned by OEMs or end users. The area is not tested by the KA650 firmware and is not included in the checksum.

4.7.2 Call-Back Entry Points

The KA650 firmware provides several entry points that facilitate I/O to the designated console device. Users of these entry points do not need to be aware of the console device type, be it a video terminal or workstation.

The primary intent of these routines is to provide a simple console device to VMB and secondary bootstraps, before operating systems load their own terminal drivers.

These are JSB (subroutine as opposed to procedure) entry points located in fixed locations in the firmware. These locations branch to code that in turn calls the appropriate routines.

All of the entry points are designed to run at IPL 31 on the interrupt stack in physical mode. Virtual mode is not supported. Due to internal firmware architectural restrictions, users are encouraged to only call into the halt protected entry points. These entry points are listed below.

CP$GET_CHAR_R4	2004 0008
CP$MSG_OUT_NOLF_R4	2004 000C
CP$READ_WTH_PRMPT_R4	2004 0010

4.7.2.1 CP$GETCHAR_R4

This routine returns the next character entered by the operator in R0. A timeout interval can be specified. If the timeout interval is zero, no timeout is generated. If a timeout is specified and if timeout occurs, a value of 18 (CAN) is returned instead of normal input.

Registers R0, R1, R2, R3, and R4 are modified by this routine, all others are preserved.

```
;-----------------------------------------------------------------------
; Usage with timeout:

movl    #timeout_in_tenths_of_second,r0 ; Specify timeout.
jsb     @#CP$GET_CHAR_R4                ; Call routine.
cmpb    r0,#^x18                        ; Check for timeout.
beql    timeout_handler                 ; Branch if timeout.
; Input is in R0.
;-----------------------------------------------------------------------
; Usage without timeout:

clrl    r0                              ; Specify no timeout.
jsb     @#CP$GET_CHAR_R4                ; Call routine.
; Input is in R0.
;-----------------------------------------------------------------------
```

4.7.2.2 CP$MSG_OUT_NOLF_R4

This routine outputs a message to the console. The message is specified either by a message code or a string descriptor. The routine distinguishes between message codes and descriptors by requiring that any descriptor be located outside of the first page of memory. Hence, message codes are restricted to values between 0 and 511.

Registers R0, R1, R2, R3, and R4 are modified by this routine, all others are preserved.

```
;-----------------------------------------------------------------------
; Usage with message code:

movzbl  #console_message_code,r0        ; Specify message code.
jsb     @#CP$MSG_OUT_NOLF_R4            ; Call routine.
;-----------------------------------------------------------------------
; Usage with a message descriptor (position dependent).

movaq   5$,r0                           ; Specify address of desc.
jsb     @#CP$MSG_OUT_NOLF_R4            ; Call routine.
.
.
5$:     .ascid  /This is a message/     ; Message with descriptor.
```

```
;-------------------------------------------------------------------------
; Usage with a message descriptor (position independent).

pushab  5$                              ; Generate message desc.
pushl   #10$-5$                         ; on stack.
movl    sp,r0                           ; Pass desc. addr. in R0.
jsb     @#CP$MSG_OUT_NOLF_R4            ; Call routine.
clrq    (sp)+                           ; Purge desc. from stack.
.
.
5$:     .ascii  /This is a message/     ; Message.
10$:     ;
;-------------------------------------------------------------------------
```

4.7.2.3 CP$READ_WTH_PRMPT_R4

This routine outputs a prompt message and then inputs a character string from the console. When the input is accepted, Delete, Ctrl U and Ctrl R functions are supported.

As with CP$MSG_OUT_NOLF_R4, either a message code or the address of a string descriptor is passed in R0 to specify the prompt string. A value of zero results in no prompt.

A descriptor of the input string is returned in R0 and R1. R0 contains the length of the string and R1 contains the address. This routine inputs the string into the console program string buffer and therefore the caller need not provide an input buffer. Successive calls however destroy the previous contents of the input buffer.

Registers R0, R1, R2, R3, and R4 are modified by this routine, all others are preserved.

```
;-------------------------------------------------------------------------
; Usage with a message descriptor (position independent).

pushab  10$                             ; Generate prompt desc.
pushl   #10$-5$                         ; on stack.
movl    sp,r0                           ; Pass desc. addr. in R0.
jsb     @#CP$READ_WTH_PRMPT_R4          ; Call routine.
clrq    (sp)+                           ; Purge prompt desc.
.                                       ; Input desc in R0 and R1.
.
5$:     .ascii  /Prompt> /  ; Prompt string.
10$:
;-------------------------------------------------------------------------
```

4.7.3 SSC RAM Layout

The KA650 firmware uses the 1 Kbyte of battery backed up (BBU) RAM in the SSC for storage of firmware specific data structures and other information that must be preserved across power cycles. This nonvolatile RAM (NVRAM) resides in the SSC chip starting at address 2014 0400. The NVRAM should not be used by the operating systems except as shown in Figure 4–7. This NVRAM is not reflected in the bitmap built by the firmware.

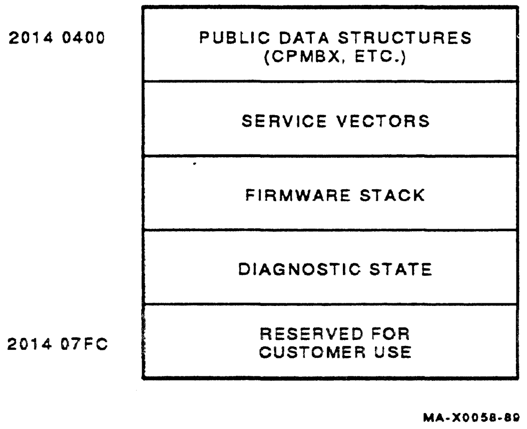


Figure 4–7 KA650 SSC NVRAM Layout

4.7.3.1 Public Data Structures

The following is a list of the public data structures in NVRAM used by the console.

Fields that are designated as reserved and/or internal use should not be written, since there is no protection against such corruption.

Console Program Mailbox

The console program mailbox (CPMBX) is a software data structure located at the beginning of NVRAM (2014 0400). The CPMBX is used to pass information between the KA650 firmware and diagnostics, VMB, or an operating system. It consists of three bytes referred to here as NVR0, NVR1, and NVR2, shown in Figures 4–8 through 4–10.

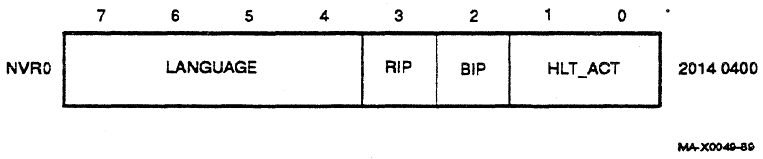


Figure 4–8 NVR0

Field	Name	Description
7:4	LANGUAGE	This field specifies the current selected language for displaying halt and error messages on terminals which support MCS.
3	RIP	If set, a restart attempt is in progress. This flag must be cleared by the operating system, if the restart succeeds.
2	BIP	If set, a bootstrap attempt is in progress. This flag must be cleared by the operating system if the bootstrap succeeds.
1:0	HLT_ ACT	Processor halt action - this field in conjunction with the conditions specified in Table 4–1 is used to control the automatic restart/bootstrap procedure. HLT_ACT is normally written by the operating system. 0 : Restart; if that fails, reboot; if that fails, halt. 1 : Restart; if that fails, halt. 2 : Reboot; if that fails, halt. 3 : Halt.

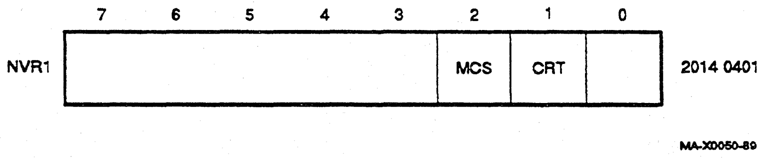


Figure 4–9 NVR1

Field	Name	Description
2	MCS	If set, indicates that the attached terminal supports multinational character set (MCS). If clear, MCS is not supported.
1	CRT	If set, indicates that the attached terminal is a CRT. If clear, indicates that the terminal is hardcopy.

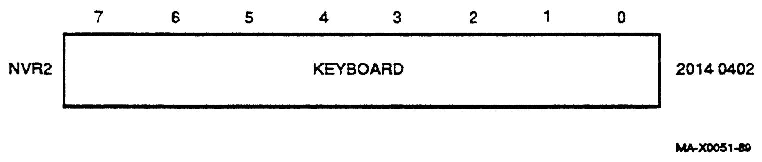


Figure 4–10 NVR2

Field	Name	Description
7:0	KEYBOARD	This field indicates the national keyboard variant in use.

4.7.3.2 Firmware Stack

This area contains the stack that is used by all of the firmware, with the exception of VMB, which has its own built in stack.

4.7.3.3 Diagnostic State

This area is used by the firmware resident diagnostics. This section is not documented here.

4.7.3.4 USER Area

The KA650 console reserves the last longword (address 2014 07FC) of the NVRAM for customer use. This location is not tested by the console firmware. Its value is undefined.

4.8 Error Messages

The error messages issued by the KA650 firmware fall into three categories: halt code messages, VMB error messages, and console messages.

4.8.1 Halt Code Messages

Except on power-up, which is not treated as an error condition, a message is issued by the firmware whenever the processor halts.

For example:

```
?06 HLT INST
PC = 800050D3
```

The number preceding the halt message is the *halt code*. This number is obtained from SAVPSL<13:8>(RESTART_CODE), IPR 43, which is written on any CVAX processor restart operation.

Table 4–10 lists messages that may appear on the console terminal when a system error occurs.

Table 4–10 HALT Messages

Code	Message	Description
?02	EXT HLT	External halt, caused by either console BREAK condition or Q22-bus BHALT_L.
_03	—	Power-up, no halt message is displayed; _03 is not displayed. However, the presence of the firmware banner and diagnostic countdown indicates this halt reason.
?04	ISP ERR	In attempting to push state onto the interrupt stack during an interrupt or exception, the processor discovered that the interrupt stack was mapped NO ACCESS or NOT VALID.
?05	DBL ERR	The processor attempted to report a machine check to the operating system, and a second machine check occurred.
?06	HLT INST	The processor executed a HALT instruction in kernel mode.
?07	SCB ERR3	The SCB vector had bits <1:0> equal to 3.
?08	SCB ERR2	The SCB vector had bits <1:0> equal to 2.
?0A	CHM FR ISTK	A change mode instruction was executed when PSL<IS> was set.
?0B	CHM TO ISTK	The SCB vector for a change mode had bit <0> set.
?0C	SCB RD ERR	A hard memory error occurred while the processor was trying to read an exception or interrupt vector.

Table 4–10 (Cont.) HALT Messages

Code	Message	Description
?10	MCHK AV	An access violation or an invalid translation occurred during machine check exception processing.
?11	KSP AV	An access violation or translation not valid occurred during processing of a kernel stack not valid exception.
?12	DBL ERR2	Double machine check error. A machine check 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 of exception.
?1B	PSL EXC7[1]	PSL<26:24> = 7 on interrupt or exception.
?1D	PSL REI5[1]	PSL<26:24> = 5 on an REI instruction
?1E	PSL REI6[1]	PSL<26:24> = 6 on an REI instruction.
?1F	PSL REI7[1]	PSL<26:24> = 7 on an REI instruction.

[1]For the last six cases, the VAX architecture does not allow execution on the interrupt stack while in a mode other than kernel. In the first three cases, an interrupt is attempting to run on the interrupt stack while not in kernel mode. In the last three cases, an REI instruction is attempting to return to a mode other than kernel and still run on the interrupt stack.

4.8.2 Console Error Messages

Table 4–11 lists error messages issued in response to a console command that has error(s).

Table 4–11 Console Error Messages

Code	Message	Description
?20	CORRPTN	The console program database has been corrupted.
?21	ILL REF	Illegal reference. The requested reference would violate virtual memory protection, the address is not mapped, the reference is invalid in the specified address space, or the value is invalid in the specified destination.
?22	ILL CMD	The command string cannot be parsed.
?23	INV DGT	A number has an invalid digit.

Table 4–11 (Cont.) Console Error Messages

Code	Message	Description
?24	LTL	The command was too large for the console to buffer. The message is issued only after receipt of the terminating carriage return.
?25	ILL ADR	The address specified falls outside the limits of the address space.
?26	VAL TOO LRG	The value specified does not fit in the destination.
?27	SW CONF	Switch conflict, for example, two different data sizes are specified for an EXAMINE command.
?28	UNK SW	The switch is unrecognized.
?29	UNK SYM	The symbolic address in an EXAMINE or DEPOSIT command is unrecognized.
?2A	CHKSM	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*.
?2B	HLTED	The operator entered a HALT command.
?2C	FND ERR	A FIND command failed either to find the RPB or 128 Kbytes of good memory.
?2D	TMOUT	During an X command, data failed to arrive in the time expected (60 seconds).
?2E	MEM ERR	A machine check occurred with a code of 80 (hex) or 81 (hex), indicating a read or write memory error.
?2F	UNXINT	Unexpected interrupt or exception
?30	UNIMPLEMENTED	Unimplemented function
?31	QUAL NOVAL	Qualifier does not take a value
?32	QUAL AMBG	Ambiguous qualifier
?33	QUAL REQ VAL	Qualifier requires a value
?34	QUAL OVERF	Too many qualifiers
?35	ARG OVERF	Too many arguments
?36	AMBG CMD	Ambiguous command
?37	INSUF ARG	Insufficient arguments

4.8.3 VMB Error Messages

Table 4–12 lists errors issued by VMB.

Table 4–12 VMB Error Messages

Code	Message	Description
?40	NOSUCHDEV	No bootable devices found.
?41	DEVASSIGN	Device is not present.
?42	NOSUCHFILE	Program image not found.
?43	FILESTRUCT	Invalid boot device file structure.
?44	BADCHKSUM	Bad checksum on header file.
?45	BADFILEHDR	Bad file header.
?46	BADIRECTORY	Bad directory file.
?47	FILNOTCNTG	Invalid program image format.
?48	ENDOFFILE	Premature end of file encountered.
?49	BADFILENAME	Bad file name given.
?4A	BUFFEROVF	Program image does not fit in available memory.
?4B	CTRLERR	Boot device I/O error.
?4C	DEVINACT	Failed to initialize boot device.
?4D	DEVOFFLINE	Device is offline.
?4E	MEMERR	Memory initialization error.
?4F	SCBINT	Unexpected SCB exception or machine check.
?50	SCB2NDINT	Unexpected exception after starting program image.
?51	NOROM	No valid ROM image found.
?52	NOSUCHNODE	No response from load server.
?53	INSFMAPREG	Invalid memory configuration.
?54	RETRY	No devices bootable, retrying.

A
KA650 Specifications

This appendix contains the physical, electrical and environmental specifications for the KA650 CPU module.

A.1 Physical Specifications

The physical specifications for the KA650 are as follows:

Dimension	**Measurement**
Height	10.457 (+0.015/-0.020) inches
Length	8.430 (+0.010/-0.010) inches
Width	0.375 inches maximum (nonconductive) 0.343 inches maximum (conductive)

NOTE
Width, as defined for Digital modules, is the height of components above the surface of the module.

A.2 Electrical Specifications

The power requirements for the KA650 CPU module are as follows:

+5 V ±5%	+12 V ±5%
6.0 A maximum	0.14 A maximum

Typical currents are 10% less than the specified maximum.

The bus loads for the KA650 CPU module are as follows:

- 3.5 ac loads
- 1.0 dc loads

A.3 Environmental Specifications

The environmental specifications for the KA650 CPU module are as follows:

Operating Conditions

Temperature	5°C (41°F) to 60°C (140°F) with a rate of change no greater than 20±2°C (36 ±4°F) per hour at sea level. For operation above sea level, decrease the operating temperature by 1.8°C for each 1000 meters (1°F for each 1000 feet).
Humidity	0% to 95% noncondensing with a maximum wet bulb temperature of 32°C (90°F) and a minimum dew point temperature of 2°C (36°F).
Altitude	Up to 2,400 meters (8,000 feet) with a rate of change no greater than 300 meters per minute (1000 feet per minute).

Nonoperating Conditions Less Than 60 Days

Temperature	-40°C to +66°C (-40°F to +151°F) with a rate of change no greater than 11 ±2 °C (20 ±4°F) per hour at sea level. For operation above sea level, decrease the nonoperating temperature by 1.8°C for each 1000 meters (1°F for each 1000 feet).
Humidity	Up to 95% noncondensing.
Altitude	Up to 4,900 meters (16,000 feet) with a rate of change no greater than 600 meters per minute (2000 feet per minute).

Nonoperating Conditions Greater Than 60 days

Temperature	+5°C to +60°C (+40°F to +140°F) with a rate of change no greater than 20 ±2°C (36 ±4°F) per hour at sea level. For operation above sea level, decrease the nonoperating temperature by 1.8°C for each 1000 meters (1°F for each 1000 feet).
Humidity	10% to 95% noncondensing with a maximum wet bulb temperature of 32°C (90°F) and a minimum dew point temperature of 2°C (36°F).
Altitude	Up to 2,400 meters (8,000 feet) with a rate of change no greater than 300 meters per minute (1000 feet per minute).

B Address Assignments

B.1 General Local Address Space Map

Table B–1 lists the VAX memory space.

Table B–1 VAX Memory Space

Address Range	Contents
0000 0000 through 03FF FFFF	Local memory space (64 Mbytes)
0400 0000 through 07FF FFFF	Reserved memory space (64 Mbytes)
0800 0000 through 0BFF FFFF	Reserved memory space (64 Mbytes)
0C00 0000 through 0FFF FFFF	Reserved memory space (64 Mbytes)
1000 0000 through 13FF FFFF	Cache diagnostic space (64 Mbytes)
1400 0000 through 17FF FFFF	Reserved cache diagnostic space (64 Mbytes)
1800 0000 through 1BFF FFFF	Reserved cache diagnostic space (64 Mbytes)
1C00 0000 through 1FFF FFFF	Reserved cache diagnostic space (64 Mbytes)

Table B–2 lists the VAX input/output memory space.

Table B–2 VAX Input/Output Space

Address Range	Contents
2000 0000 through 2000 1FFF	Local Q22-bus I/O space (8 Kbytes)
2000 2000 through 2003 FFFF	Reserved local I/O space (248 Kbytes)

Table B–2 (Cont.) VAX Input/Output Space

Address Range	Contents
2004 0000 through 2005 FFFF	Local ROM space, halt protected space (128 Kbytes)
2006 0000 through 2007 FFFF	Local ROM space, halt unprotected space (128 Kbytes)
2008 0000 through 201F FFFF	Local register I/O space (1.5 Mbytes)
2020 0000 through 23FF FFFF	Reserved local I/O space (62.5 Mbytes)
2400 0000 through 27FF FFFF	Reserved local I/O space (64 Mbytes)
2800 0000 through 2BFF FFFF	Reserved local I/O space (64 Mbytes)
2C08 0000 through 2FFF FFFF	Reserved local I/O space (64 Mbytes)
3000 0000 through 303F FFFF	Local Q22-bus memory space (4 Mbytes)
3040 0000 through 33FF FFFF	Reserved local I/O space (60 Mbytes)
3400 0000 through 37FF FFFF	Reserved local I/O space (64 Mbytes)
3800 0000 through 3BFF FFFF	Cache tag diagnostic space (64 Mbytes) [1]
3C00 0000 through 3FFF FFFF	Reserved cache tag diagnostic space (64 Mbytes)

[1]Not visible during normal operation.

B.2 Detailed Local Address Space Map

Table B–3 describes the contents of the VAX memory space.

Table B–3 VAX Memory Space

Address Range	Contents
0000 0000 through 03FF FFFF	Local memory space (up to 64 Mbytes) [1]
0400 0000 through 0FFF FFFF	Reserved memory space
1000 0000 through 13FF FFFF	Cache diagnostic space
1800 0000 through 1FFF FFFF	Reserved cache diagnostic space

[1]Q22-bus map top 32 Kbytes of main memory

Table B–4 describes the contents of the VAX input/output memory space.

Table B–4 VAX Input/Output Space

Address Range	Contents
2000 0000 through 2000 1FFF	**Local Q22-bus I/O space**
2000 0000 through 2000 0007	Reserved Q22-bus I/O space
2000 0008 through 2000 07FF	Q22-bus floating address space
2000 0800 through 2000 0FFF	User reserved Q22-bus I/O space
2000 1000 through 2000 1F3F	Reserved Q22-bus I/O space
2000 1F40	Interprocessor communication register (normal operation)
2000 1F42	Interprocessor communication register (reserved)
2000 1F44	Interprocessor communication register (reserved)
2000 1F46	Interprocessor communication register (reserved)
2000 1F48 through 2000 1FFF	Reserved Q22-bus I/O space
2000 2000 through 2003 FFFF	Reserved Local I/O space
2004 0000 through 2007 FFFF	**Local ROM space**
2004 0000 through 2005 FFFF	Local ROM protected space
2004 0004	MicroVAX system type register (in ROM)
2006 0000 through 2007 FFFF	Local ROM unprotected space
2008 0000 through 201F FFFF	**Local Register I/O space**
2008 0000	DMA system configuration register
2008 0004	DMA system error register
2008 0008	Q22-bus error address register
2008 000C	DMA error address register
2008 0010	Q22-bus map base register
2008 0014 through 2008 013C	Reserved local register I/O space
2008 0140	Main memory error status register
2008 0144	Main memory control/diagnostic status register
2008 0018 through 2008 3FFF	Reserved local register I/O space
2008 4000	Cache control register
2008 4004	Boot and diagnostic register
2008 4008 through 2008 7FFF	Reserved local register I/O space

Table B–4 (Cont.) VAX Input/Output Space

Address Range	Contents
2008 8000 through 2008 FFFF	Q22-bus map registers
2009 0000 through 2014 0020	Reserved local register I/O space
2014 0030	Diagnostic LED register
2014 0034 through 2014 0068	Reserved local register I/O space
2014 006C through 2001 40FF	Diagnostic registers
2014 0100	Timer 0 control register
2014 0104	Timer 0 interval register
2014 0108	Timer 0 next interval register
2014 010C	Timer 0 interrupt vector
2014 0110	Timer 1 control register
2014 0114	Timer 1 interval register
0118	Timer 1 next interval register
2014 011C	Timer 1 interrupt vector
2014 0120 through 2014 03FF	Reserved local register I/O space
2014 0400 through 2014 07FF	Battery backed-up RAM
2014 0800 through 201F FFFF	Reserved local register I/O space
2020 0000 through 2FFF FFFF	Reserved local I/O space
3000 0000 through 303F FFFF	Local Q22-bus memory space
3040 0000 through 37FF FFFF	Reserved local register I/O space
3800 0000 through 3BFF FFFF [1]	Cache tag diagnostic space
3C00 0000 through 3FFF FFFF	Reserved cache tag diagnostic space

[1]Not visible during normal operation

B.3 External IPRs

Several of the internal processor registers (IPRs) on the KA650 are implemented in the SSC rather than the CVAX chip. These registers are referred to as external IPRs, and are listed in Table B–5.

Table B–5 External IPRs

IPR Number	Register Name	Abbreviation
27	Time of year register	TOY
28	Console storage receiver status	CSRS[1]
29	Console storage receiver data	CSRD [1]
30	Console storage transmitter status	CSTS[1]
31	Console storage transmitter data	CSDB[1]
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

[1]These registers are not fully implemented. Accesses yield unpredictable results.

B.4 Global Q22-bus Address Space Map

The addresses and memory contents of the Q22-bus memory space is as listed in Table B–6.

Table B–6 Q22-bus Memory Space Map

Address Range	Contents
0000 0000 through 1777 7777	Q22-bus memory space (octal)

The contents and addresses of the Q22-bus I/O space with BBS7 asserted is listed in Table B–7.

Table B–7 Q22-bus Input/Output Space with BBS7 Asserted

Address Range	Contents
1776 0000 through 1777 7777	**Q22-bus I/O space** (octal)
1776 0000 through 1776 0007	Reserved Q22-bus I/O space
1776 0010 through 1776 3777	Q22-bus floating address space
1776 4000 through 1776 7777	User reserved Q22-bus I/O space
1777 0000 through 1777 7477	Reserved Q22-bus I/O space
1777 7500	Interprocessor communication register (normal operation)
1777 7502	Interprocessor communication register (reserved)
1777 7504	Interprocessor communication register (reserved)
1777 7506	Interprocessor communication register (reserved)
1777 7510 through 1777 7777	Reserved Q22-bus I/O space

C Q22-bus Specification

C.1 Introduction

The Q22-bus, also known as the extended LSI-11 bus, is the low-end member of DIGITAL's bus family. All of DIGITAL's microcomputers, such as the MicroVAX I, MicroVAX II, MicroVAX 3500, MicroVAX 3600, and MicroPDP-11 use the Q22-bus.

The Q22-bus consists of 42 bidirectional and 2 unidirectional signal lines. These form the lines along which the processor, memory, and I/O devices communicate with each other.

Addresses, data, and control information are sent along these signal lines, some of which contain time-multiplexed information. The lines are divided as follows:

- Sixteen multiplexed data/address lines — BDAL<15:00>
- Two multiplexed address/parity lines — BDAL<17:16>
- Four extended address lines — BDAL<21:18>
- Six data transfer control lines — BBS7, BDIN, BDOUT, BRPLY, BSYNC, BWTBT
- Six system control lines — BHALT, BREF, BEVNT, BINIT, BDCOK, BPOK
- Ten interrupt control and direct memory access control lines — BIAKO, BIAKI, BIRQ4, BIRQ5, BIRQ6, BIRQ7, BDMGO, BDMR, BSACK, BDMGI

In addition, a number of power, ground, and space lines are defined for the bus. Refer to Table C–1 for a detailed description of these lines.

The discussion in this appendix applies to the general 22-bit physical address capability. All modules used with the KA655 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.

C.1.1 Master/Slave Relationship

Communication between devices on the bus is asynchronous. A master/slave relationship exists throughout each bus transaction. Only one device has control of the bus at any one time. This controlling device is termed the bus master, or arbiter. The master device controls the bus when communicating with another device on the bus, termed the slave.

The bus master (typically the processor or a DMA device) initiates a bus transaction. The slave device responds by acknowledging the transaction in progress and by receiving data from, or transmitting data to, the bus master. Q22-bus control signals transmitted or received by the bus master or bus slave device must complete the sequence according to bus protocol.

The processor controls bus arbitration, that is, which device becomes bus master at any given time. A typical example of this relationship is a disk drive, as master, transferring data to memory as slave. Communication on the Q22-bus is interlocked so that, for certain control signals issued by the master device, there must be a response from the slave in order to complete the transfer. It is the master/slave signal protocol that makes the Q22-bus asynchronous. The asynchronous operation precludes the need for synchronizing with, and waiting for, clock pulses.

Since bus cycle completion by the bus master requires response from the slave device, each bus master must include a timeout error circuit that aborts the bus cycle if the slave does not respond to the bus transaction within 10 μs. The actual time before a timeout error occurs must be longer than the reply time of the slowest peripheral or memory device on the bus.

C.2 Q22-bus Signal Assignments

Table C–1 lists the data and address signal assignments. Table C–2 lists the control signal assignments. Table C–3 lists the power and ground signal assignments. Table C–4 lists the spare signal assignments.

Table C–1 Data and Address Signal Assignments

Data and Address Signal	Pin Assignment
BDAL0	AU2
BDAL1	AV2
BDAL2	BE2
BDAL3	BF2
BDAL4	BH2
BDAL5	BJ2
BDAL6	BK2
BDAL7	BL2
BDAL8	BM2
BDAL9	BN2
BDAL10	BP2
BDAL11	BR2
BDAL12	BS2
BDAL13	BT2
BDAL14	BU2
BDAL15	BV2
BDAL16	AC1
BDAL17	AD1
BDAL18	BC1
BDAL19	BD1
BDAL20	BE1
BDAL21	BF1

Table C–2 Control Signal Assignments

Control Signal	Pin Assignment
Data Control	
BDOUT	AE2
BRPLY	AF2
BDIN	AH2
BSYNC	AJ2
BWTBT	AK2
BBS7	AP2
Interrupt Control	
BIRQ7	BP1
BIRQ6	AB1
BIRQ5	AA1
BIRQ4	AL2
BIAKO	AN2
BIAKI	AM2
DMA Control	
BDMR	AN1
BSACK	BN1
BDMGO	AS2
BDMGI	AR2
System Control	
BHALT	AP1
BREF	AR1
BEVNT	BR1
BINIT	AT2
BDCOK	BA1
BPOK	BB1

Table C–3 Power and Ground Signal Assignments

Power and Ground	Pin Assignment
+5 B (battery) or +12 B (battery)	AS1
+12 B	BS1
+5 B	AV1
+5	AA2
+5	BA2
+5	BV1
+12	AD2
+12	BD2
+12	AB2
-12	AB2
-12	BB2
GND	AC2
GND	AJ1
GND	AM1
GND	AT1
GND	BC2
GND	BJ1
GND	BM1
GND	BT1

Table C–4 Spare Signal Assignments

Spare	Pin Assignment
SSpare1	AE1
SSpare3	AH1
SSpare8	BH1
SSpare2	AF1
MSpareA	AK1
MSpareB	AL1
MSpareB	BK1
MSpareB	BL1
PSpare1	AU1
ASpare2	BU1

C.3 Data Transfer Bus Cycles

Data transfer bus cycles, executed by bus master devices, transfer 32-bit words or 8-bit bytes to or from slave devices. In block mode, multiple words can be transferred to sequential word addresses, starting from a single bus address. Data transfer bus cycles are listed and defined in Table C–5.

Table C–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 C–6 are used in the data transfer operations described in Table C–5.

Table C–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

Table C–6 (Cont.) Bus Signals for Data Transfers

Signal	Definition	Function
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.

C.3.1 Bus Cycle Protocol

Before initiating a bus cycle, the previous bus transaction must have been completed (BSYNC L negated) and the device must become bus master. The bus cycle can be divided into two parts: addressing and data transfer. During addressing, the bus master outputs the address for the desired slave device, memory location, or device register. The selected slave device responds by latching the address bits and holding this condition for the duration of the bus cycle until BSYNC L becomes negated. During data transfer the actual data transfer occurs.

C.3.2 Device Addressing

Device addressing of a data transfer bus cycle comprises an address setup and deskew time, and an address hold and deskew time. During address setup and deskew time, the bus master does the following operations:

- Asserts BDAL<21:00> L with the desired slave device address bits.
- Asserts BBS7 L if a device in the I/O page is being addressed.
- Asserts BWTBT L if the cycle is a DATO(B) or DATBO bus cycle.

During this time, the address, BBS7 L, and BWTBT L signals are asserted at the slave bus receiver for at least 75 ns before BSYNC goes active. Devices in the I/O page ignore the nine high-order address bits BDAL<21:13>, and instead, decode BBS7 L along with the 13 low-order address bits. An active BWTBT L signal during address setup time indicates that a DATO(B) or DATBO operation follows, while an inactive BWTBT L indicates a DATI, DATBI, or DATIO(B) operation.

The address hold and deskew time begins after BSYNC L is asserted.

The slave device uses the active BSYNC L bus received output to clock BDAL address bits, BBS7 L, and BWTBT L into its internal logic. BDAL<21:00> L, BBS7 L, and BWTBT L remain active for 25 ns minimum after the BSYNC L bus receiver goes active. BSYNC L remains active for the duration of the bus cycle.

Memory and peripheral devices are addressed similarly, except for the way the slave device responds to BBS7 L. Addressed peripheral devices must not decode address bits on BDAL<21:13> L. Addressed peripheral device can respond to a bus cycle when BBS7 L is asserted (low) during the addressing of the cycle. When asserted, BBS7 L indicates that the device address resides in the I/O page (the upper 4K address space). Memory devices generally do not respond to addresses in the I/O page; however, some system applications may permit memory to reside in the I/O page for use as DMA buffers, read-only memory bootstraps, and diagnostics.

DATI

The DATI bus cycle, shown in Figure C–1, is a read operation. During DATI, data is input to the bus master. Data consists of 16-bit word transfers over the bus. During data transfer of the DATI bus cycle, the bus master asserts BDIN L 100 ns minimum after BSYNC L is asserted. The slave device responds to BDIN L active as follows:

- Asserts BRPLY L 0 ns minimum (8 ns maximum to avoid bus timeout) after receiving BDIN L, and 125 ns maximum before BDAL bus driver data bits are valid.
- Asserts BDAL<21:00> L with the addressed data and error information 0 ns (minimum) after receiving BDIN, and 125 ns (maximum) after assertion of BRPLY.

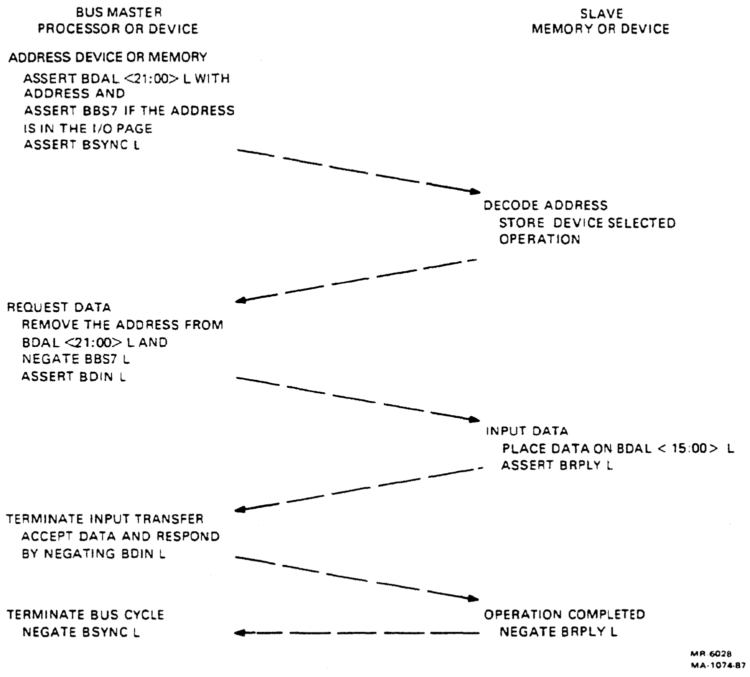


Figure C–1 DATI Bus Cycle

When the bus master receives BRPLY L, it does the following:

- Waits at least 200 ns deskew time and then accepts input data at BDAL<17:00> L bus receivers. BDAL <17:16> L are used for transmitting parity errors to the master.
- Negates BDIN L 200 ns minimum to 2 µs maximum after BRPLY L goes active.

The slave device responds to BDIN L negation by negating BRPLY L and removing read data from BDAL bus drivers. BRPLY L must be negated 100 ns maximum prior to removal of read data. The bus master responds to the negated BRPLY L by negating BSYNC L.

Conditions for the next BSYNC L assertion are as follows:

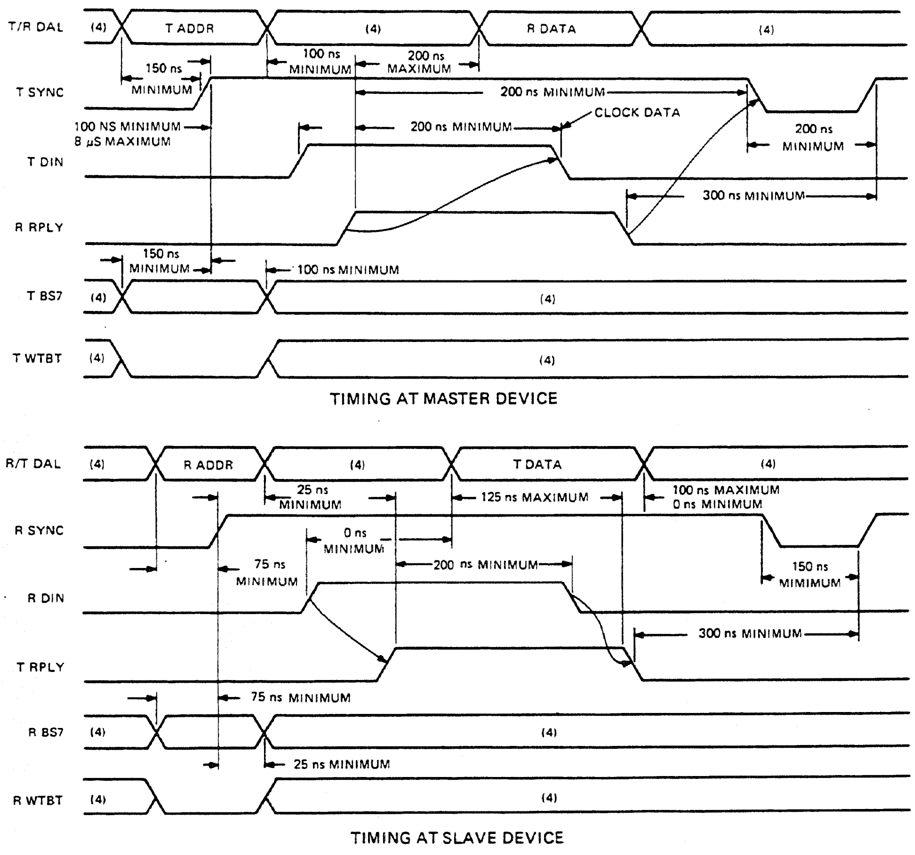
- BSYNC L must remain negated for 200 ns minimum.
- BSYNC L must not become asserted within 300 ns of previous BRPLY L negation.

Figure C–2 shows DATI bus cycle timing.

NOTE
Continuous assertion of BSYNC L retains control of the bus by the bus master, and the previously addressed slave device remains selected. This is done for DATIO(B) bus cycles where DATO or DATOB follows a DATI without BSYNC L negation and a second device addressing operation. Also, a slow slave device can hold off data transfers to itself by keeping BRPLY L asserted, which causes the master to keep BSYNC L asserted.

DATOB

DATOB, shown in Figure C–3, is a write operation. Data is transferred in 32-bit words (DATO) or 8-bit bytes (DATOB) from the bus master to the slave device. The data transfer output can occur after the addressing part of a bus cycle when BWTBT L has been asserted by the bus master, or immediately following an input transfer part of a DATIOB bus cycle.



NOTES:
1. TIMING SHOWN AT MASTER AND SLAVE DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR-6037
MA-1084-87

Figure C–2 DATI Bus Cycle Timing

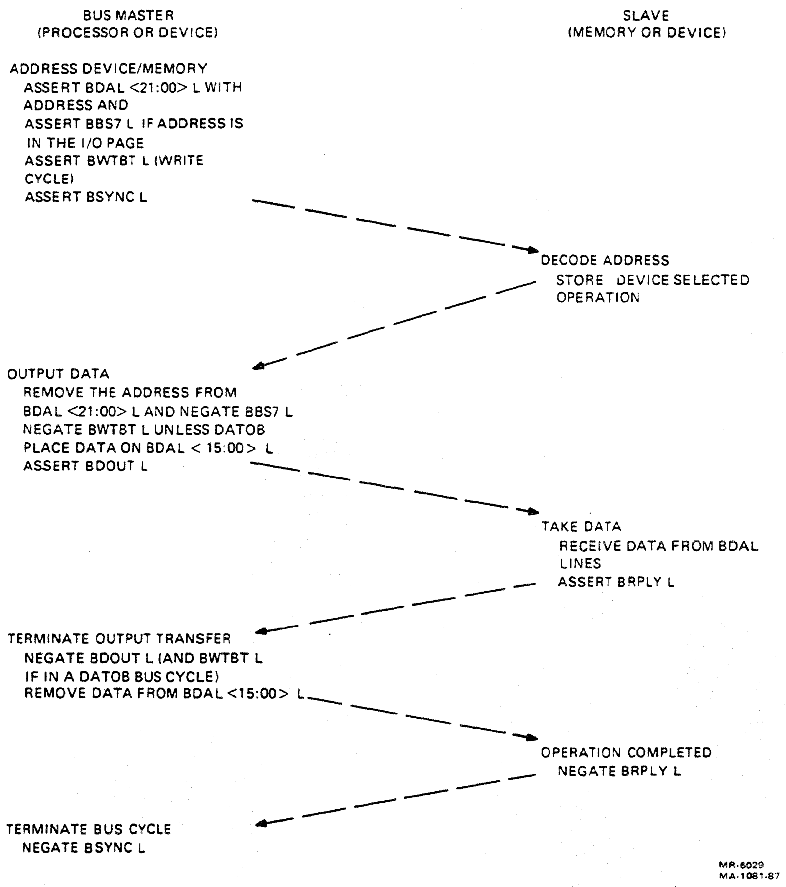

BUS MASTER
(PROCESSOR OR DEVICE)
SLAVE
(MEMORY OR DEVICE)
ADDRESS DEVICE/MEMORY
ASSERT BDAL <21:00> L WITH ADDRESS AND
ASSERT BBS7 L IF ADDRESS IS IN THE I/O PAGE
ASSERT BWTBT L (WRITE CYCLE)
ASSERT BSYNC L
DECODE ADDRESS
STORE DEVICE SELECTED OPERATION
OUTPUT DATA
REMOVE THE ADDRESS FROM BDAL <21:00> L AND NEGATE BBS7 L
NEGATE BWTBT L UNLESS DATOB
PLACE DATA ON BDAL < 15:00> L
ASSERT BDOUT L
TAKE DATA
RECEIVE DATA FROM BDAL LINES
ASSERT BRPLY L
TERMINATE OUTPUT TRANSFER
NEGATE BDOUT L (AND BWTBT L IF IN A DATOB BUS CYCLE)
REMOVE DATA FROM BDAL <15:00> L
OPERATION COMPLETED
NEGATE BRPLY L
TERMINATE BUS CYCLE
NEGATE BSYNC L
MR-6029
MA-1081-87

Figure C–3 DATO or DATOB Bus Cycle

The data transfer part of a DATOB bus cycle comprises a data setup and deskew time and a data hold and deskew time.

During the data setup and deskew time, the bus master outputs the data on BDAL<15:00> L at least 100 ns after BSYNC L assertion. BWTBT L remains negated for the length of the bus cycle. If the transfer is a byte transfer, BWTBT L remains asserted. If it is the output of a DATIOB, BWTBT L becomes asserted and lasts the duration of the bus cycle.

During a byte transfer, BDAL<00> L selects the high or low byte. This occurs in the addressing part of the cycle. If asserted, the high byte (BDAL<15:08> L) is selected; otherwise, the low byte (BDAL<07:00> L) is selected. An asserted BDAL 16 L at this time forces a parity error to be written into memory if the memory is a parity-type memory. BDAL 17 L is not used for write operations. The bus master asserts BDOUT L at least 100 ns after BDAL and BDWTBT L bus drivers are stable. The slave device responds by asserting BRPLY L within 10 µs to avoid bus timeout. This completes the data setup and deskew time.

During the data hold and deskew time, the bus master receives BRPLY L and negates BDOUT L, which must remain asserted for at least 150 ns from the receipt of BRPLY L before being negated by the bus master. BDAL<17:00> L bus drivers remain asserted for at least 100 ns after BDOUT L negation. The bus master then negates BDAL inputs.

During this time, the slave device senses BDOUT L negation. The data is accepted and the slave device negates BRPLY L. The bus master responds by negating BSYNC L. However, the processor does not negate BSYNC L for at least 175 ns after negating BDOUT L. This completes the DATOB bus cycle. Before the next cycle, BSYNC L must remain unasserted for at least 200 ns. Figure C–4 shows DATOB bus cycle timing.

DATIOB

The protocol for a DATIOB bus cycle is identical to the addressing and data transfer part of the DATI and DATOB bus cycles, and is shown in Figure C–5. After addressing the device, a DATI cycle is performed as explained earlier; however, BSYNC L is not negated. BSYNC L remains active for an output word or byte transfer (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 C–6 illustrates DATIOB bus cycle timing.

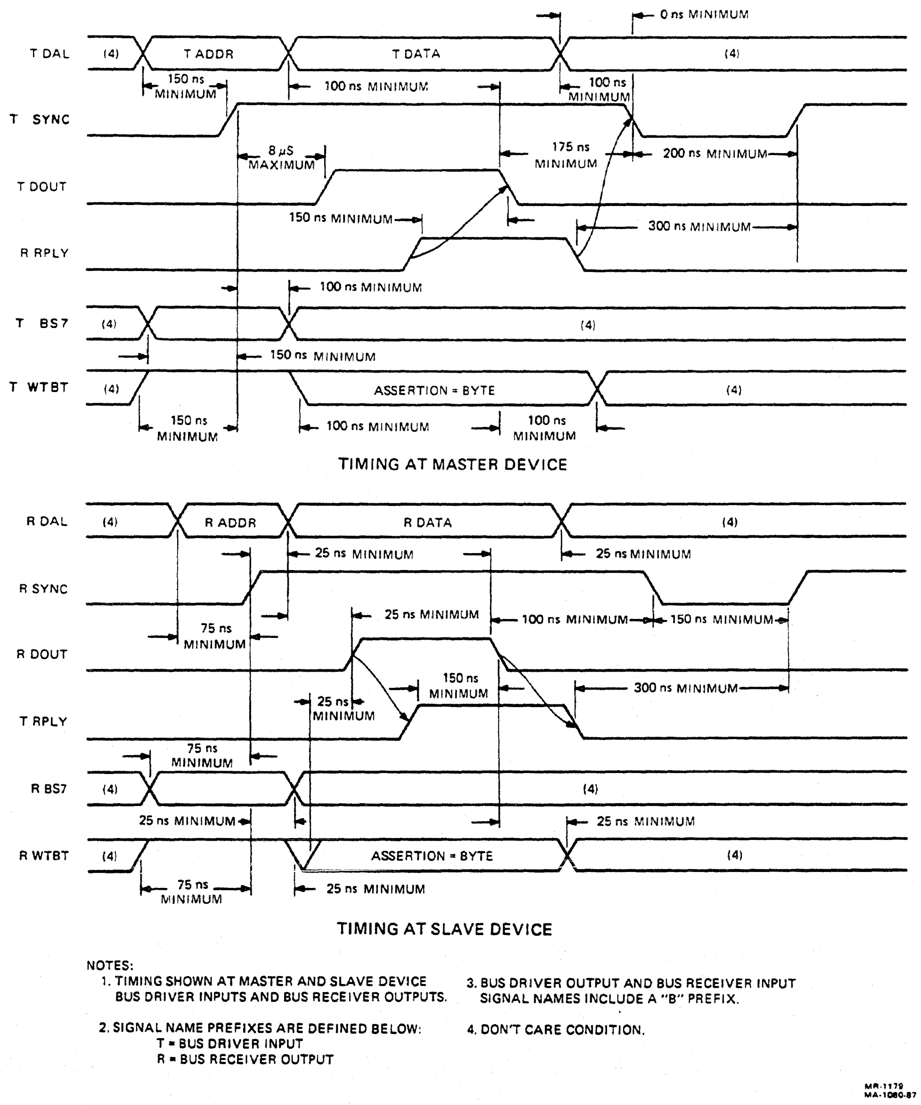


Figure C–4 DATO or DATOB Bus Cycle Timing

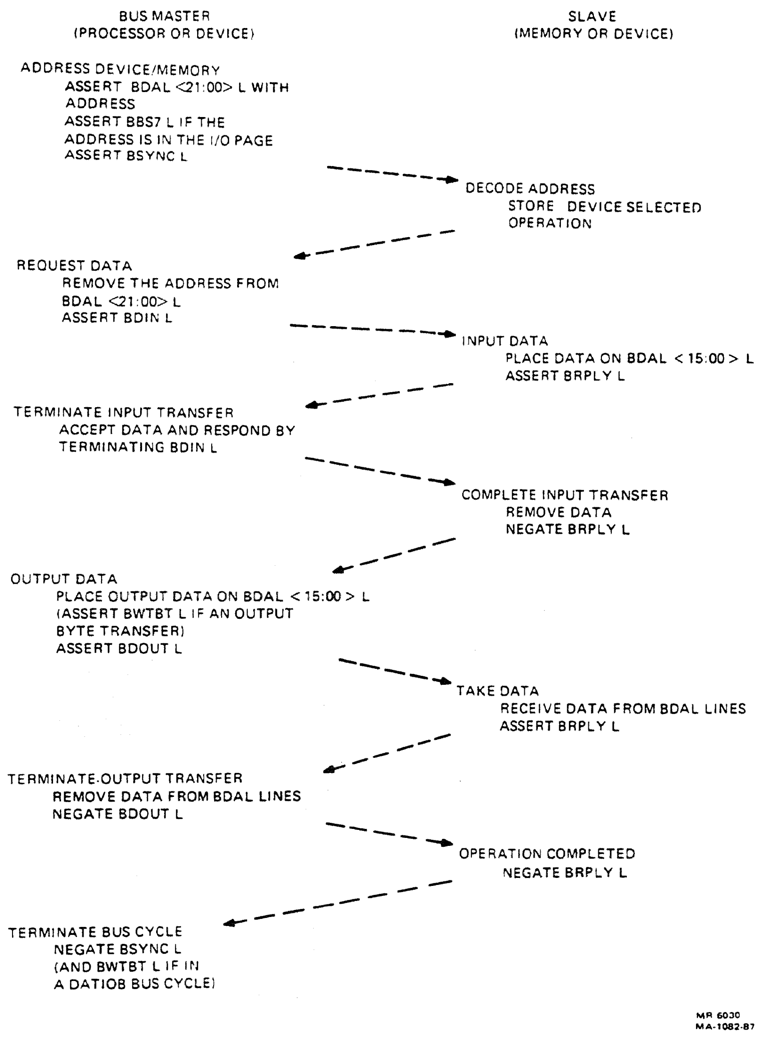
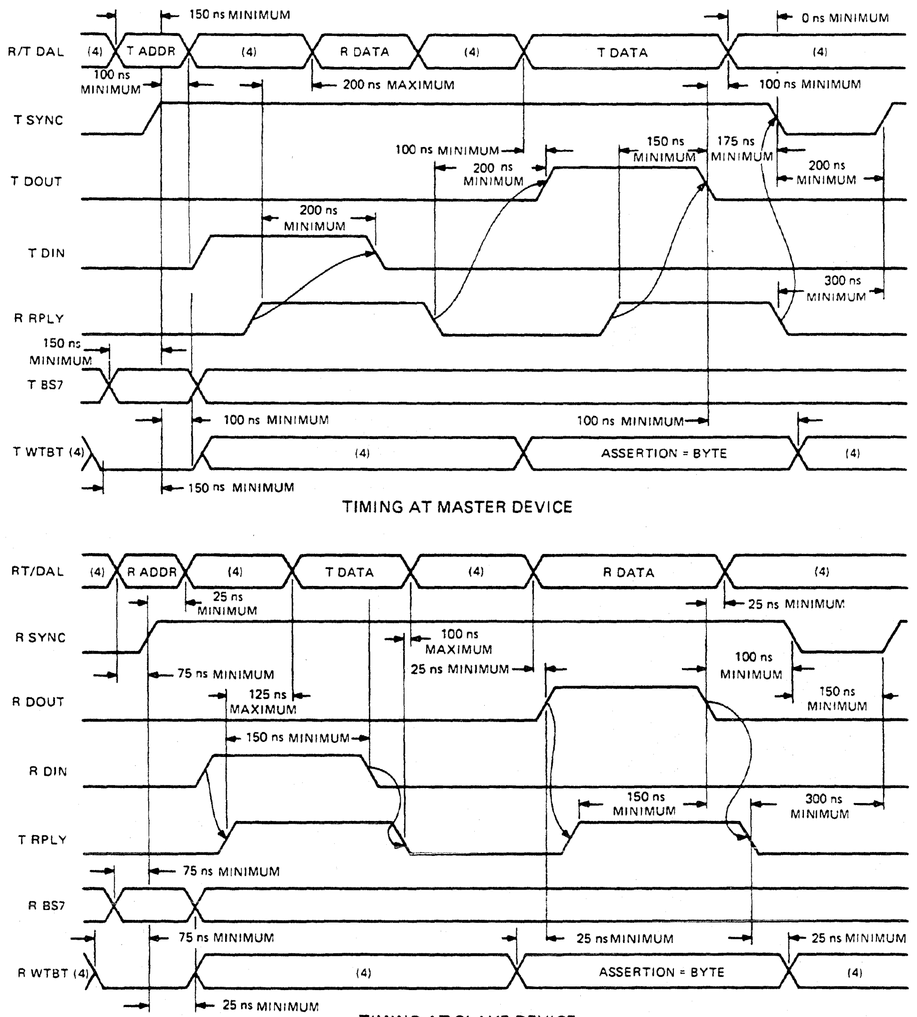

BUS MASTER
(PROCESSOR OR DEVICE)
SLAVE
(MEMORY OR DEVICE)
ADDRESS DEVICE/MEMORY
ASSERT BDAL <21:00> L WITH ADDRESS
ASSERT BBS7 L IF THE ADDRESS IS IN THE I/O PAGE
ASSERT BSYNC L
DECODE ADDRESS
STORE DEVICE SELECTED OPERATION
REQUEST DATA
REMOVE THE ADDRESS FROM BDAL <21:00> L
ASSERT BDIN L
INPUT DATA
PLACE DATA ON BDAL < 15:00 > L
ASSERT BRPLY L
TERMINATE INPUT TRANSFER
ACCEPT DATA AND RESPOND BY TERMINATING BDIN L
COMPLETE INPUT TRANSFER
REMOVE DATA
NEGATE BRPLY L
OUTPUT DATA
PLACE OUTPUT DATA ON BDAL < 15:00 > L
(ASSERT BWTBT L IF AN OUTPUT BYTE TRANSFER)
ASSERT BDOUT L
TAKE DATA
RECEIVE DATA FROM BDAL LINES
ASSERT BRPLY L
TERMINATE OUTPUT TRANSFER
REMOVE DATA FROM BDAL LINES
NEGATE BDOUT L
OPERATION COMPLETED
NEGATE BRPLY L
TERMINATE BUS CYCLE
NEGATE BSYNC L
(AND BWTBT L IF IN A DATIOB BUS CYCLE)
MR 6030
MA-1082-87

Figure C–5 DATIO or DATIOB Bus Cycle



NOTES:
1. TIMING SHOWN AT REQUESTING DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR 6036
MA-1060-87

Figure C–6 DATIO or DATIOB Bus Cycle Timing

C.4 Direct Memory Access

The direct memory access (DMA) capability allows direct data transfer between I/O devices and memory. This is useful when using mass storage devices (for example, disks) that move large blocks of data to and from memory. A DMA device needs to 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

C.4.1 DMA Protocol

A DMA transaction can be divided into the following three phases:

- Bus mastership acquisition phase
- Data transfer phase
- Bus mastership relinquishment phase

During the bus mastership acquisition phase, a DMA device requests the bus by asserting BDMR L. The processor arbitrates the request and initiates the transfer of bus mastership by asserting BDMGO L.

The maximum time between BDMR L assertion and BDMGO L assertion is DMA latency. This time is processor-dependent. BDMGO L/BDMGI L is one signal that is daisy-chained through each module in the backplane.

It is driven out of the processor on the BDMGO L pin, enters each module on the BDMGI L pin, and exits on the BDMGO L pin. This signal passes through the modules in descending order of priority until it is stopped by the requesting device. The requesting device blocks the output of BMDGO L and asserts BSACK L. If BDMR L is continuously asserted, the bus hangs.

During the data transfer phase, the DMA device continues asserting BSACK L. The actual data transfer is performed as described earlier.

The DMA device can assert BSYNC L for a data transfer 250 ns minimum after it received BDMGI L and its BSYNC L bus receiver is negated.

During the bus mastership relinquishment phase, the DMA device gives up the bus by negating BSACK L. This occurs after completing (or aborting) the last data transfer cycle (BRPLY L negated). BSACK L can be negated up to a maximum of 300 ns before negating BSYNC L.

NOTE
If multiple data transfers are performed during this phase, consideration must be given to the use of the bus for other system functions, such as memory refresh (if required).

Figure C–7 shows the DMA protocol, and Figure C–8 shows DMA request/grant timing.

C.4.2 Block Mode DMA

For increased throughput, block mode DMA can be implemented on a device for use with memories that support this type of transfer. In a block mode transaction, the starting memory address is asserted, followed by data for that address, and data for consecutive addresses.

By eliminating the assertion of the address for each data word, the transfer rate is almost doubled.

There are two types of block mode transfers, DATBI (input) and DATBO (output). The DATBI bus cycle is described in Section C.4.2.1 and illustrated in Figure C–9.

The DATBO bus cycle is described in Section C.4.2.2 and illustrated in Figure C–10.

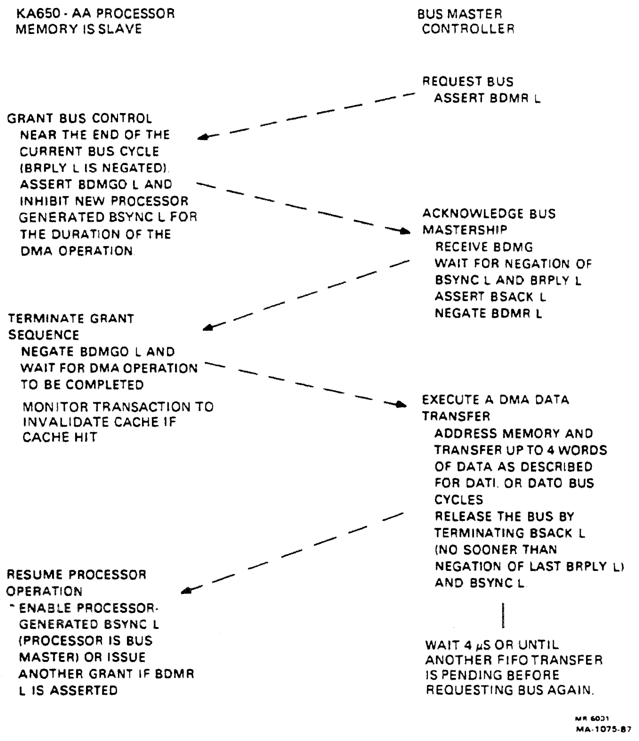
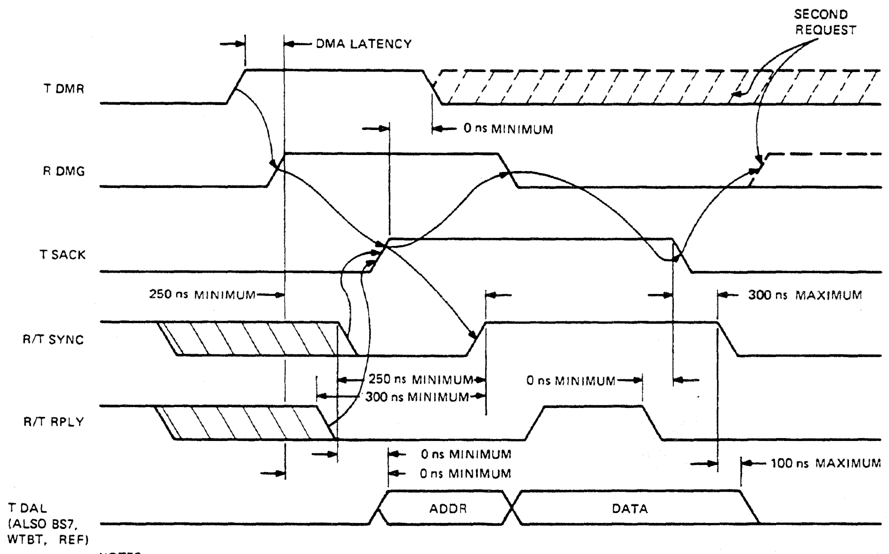

KA650 - AA PROCESSOR MEMORY IS SLAVE
BUS MASTER CONTROLLER
REQUEST BUS
ASSERT BDMR L
GRANT BUS CONTROL
NEAR THE END OF THE CURRENT BUS CYCLE (BRPLY L IS NEGATED).
ASSERT BDMGO L AND INHIBIT NEW PROCESSOR GENERATED BSYNC L FOR THE DURATION OF THE DMA OPERATION.
ACKNOWLEDGE BUS MASTERSHIP
RECEIVE BDMG
WAIT FOR NEGATION OF BSYNC L AND BRPLY L
ASSERT BSACK L
NEGATE BDMR L
TERMINATE GRANT SEQUENCE
NEGATE BDMGO L AND WAIT FOR DMA OPERATION TO BE COMPLETED
MONITOR TRANSACTION TO INVALIDATE CACHE IF CACHE HIT
EXECUTE A DMA DATA TRANSFER
ADDRESS MEMORY AND TRANSFER UP TO 4 WORDS OF DATA AS DESCRIBED FOR DATI, OR DATO BUS CYCLES
RELEASE THE BUS BY TERMINATING BSACK L (NO SOONER THAN NEGATION OF LAST BRPLY L) AND BSYNC L
RESUME PROCESSOR OPERATION
ENABLE PROCESSOR-GENERATED BSYNC L (PROCESSOR IS BUS MASTER) OR ISSUE ANOTHER GRANT IF BDMR L IS ASSERTED
WAIT 4 µS OR UNTIL ANOTHER FIFO TRANSFER IS PENDING BEFORE REQUESTING BUS AGAIN.
MR 6031
MA-1075-87

Figure C–7 DMA Protocol



NOTES:
1. TIMING SHOWN AT REQUESTING DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
T = BUS DRIVER INPUT
R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.

MR-3690
MA-1078-87

Figure C–8 DMA Request/Grant Timing

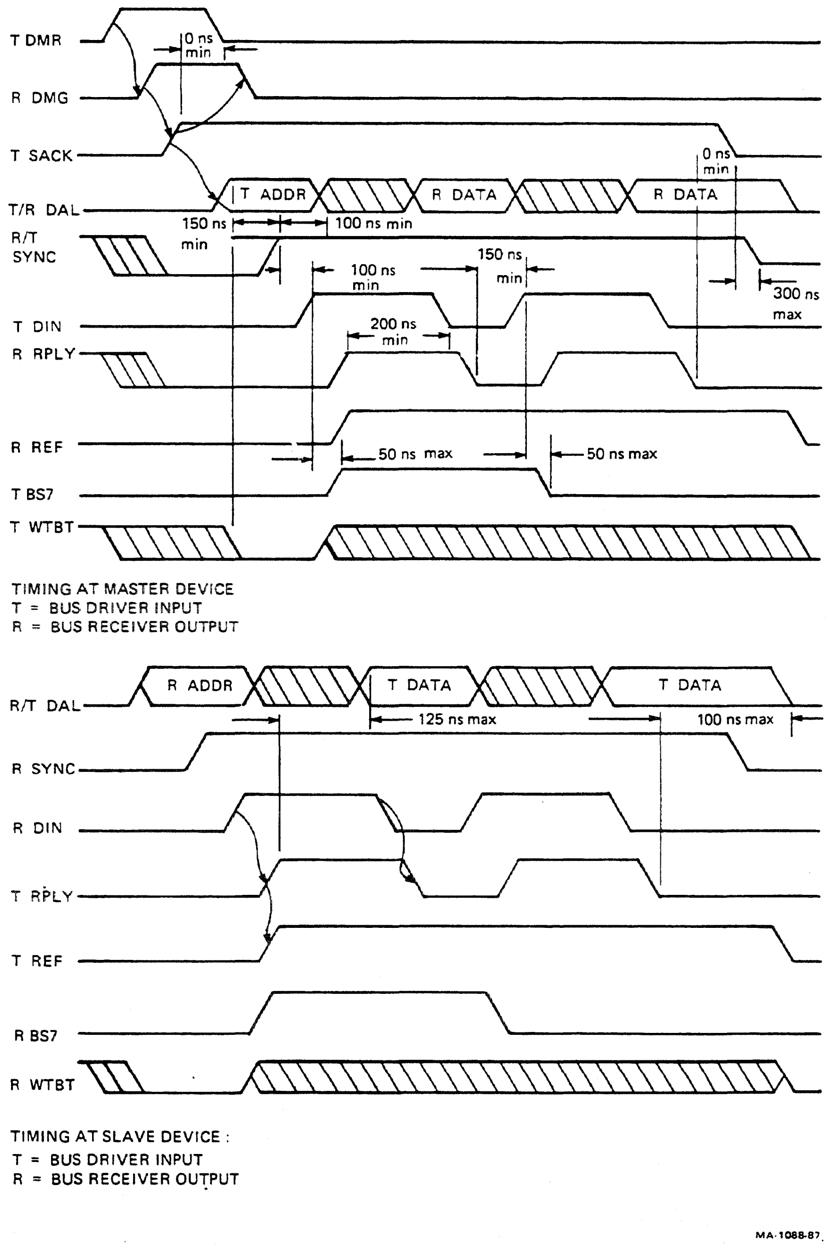


Figure C–9 DATBI Bus Cycle Timing

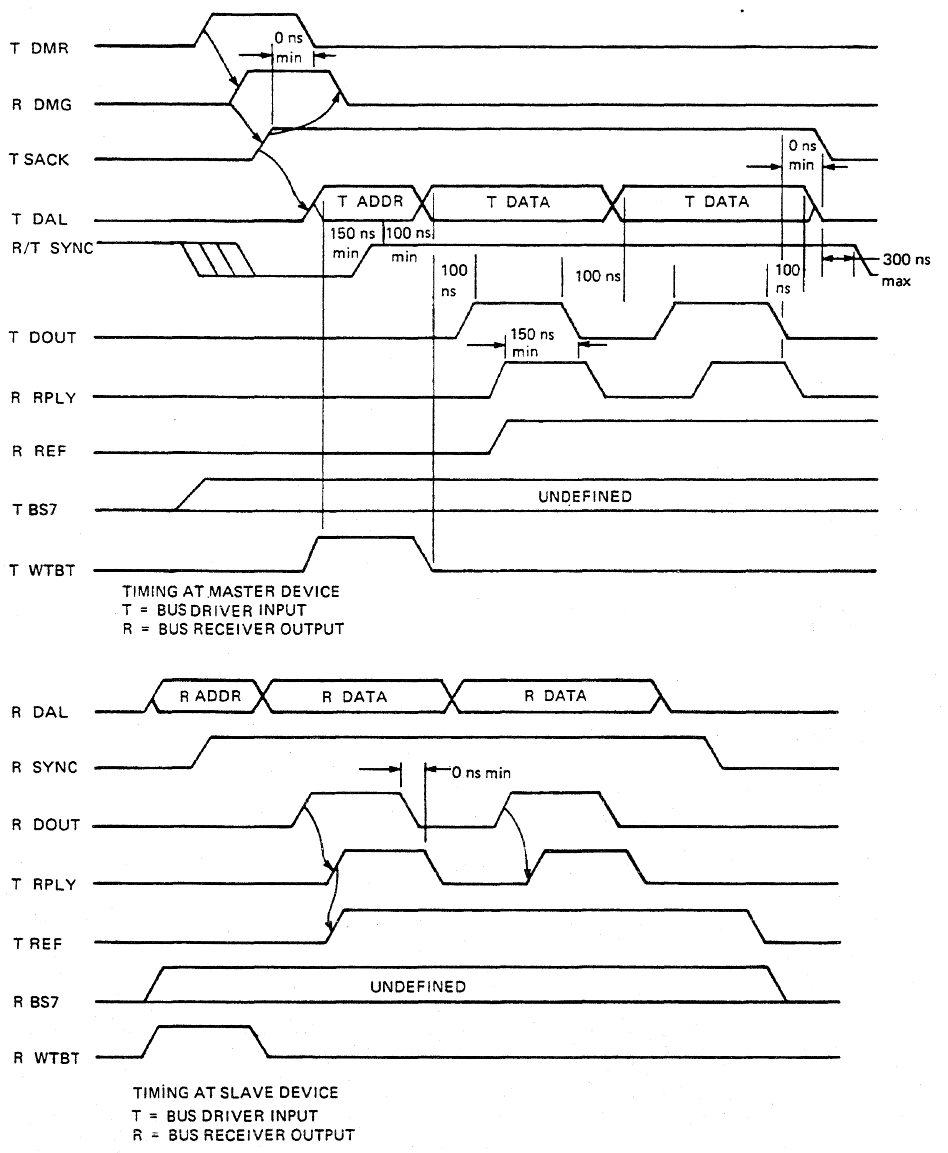


Figure C–10 DATBO Bus Cycle Timing

C.4.2.1 DATBI Bus Cycle

Before a DATBI block mode transfer can occur, the DMA bus master device must request control of the bus. This occurs under conventional Q22-bus protocol.

A block mode DATBI transfer is executed as follows:

- **Address device memory**–the address is asserted by the bus master on TADDR<21:00> along with the negation of TWTBT. The bus master asserts TSYNC 150 ns minimum after gating the address onto the bus.
- **Decode address**–the appropriate memory device recognizes that it must respond to the address on the bus.
- **Request data**–the address is removed by the bus master from TADDR<21:00> 100 ns minimum after the assertion of TSYNC. The bus master asserts the first TDIN 100 ns minimum after asserting TSYNC. The bus master asserts TBS7 50 ns maximum after asserting TDIN for the first time. TBS7 remains asserted until 50 ns maximum after the assertion of TDIN for the last time. In each case, TBS7 can be asserted or negated as soon as the conditions for asserting TDIN are met. The assertion of TBS7 indicates the bus master is requesting another read cycle after the current read cycle.
- **Send data**–the bus slave asserts TRPLY 0 ns minimum (8000 ns maximum to avoid a bus timeout) after receiving RDIN. The bus slave asserts TREF concurrent with TRPLY if, and only if, it is a block mode device which can support another RDIN after the current RDIN. The bus slave gates TDATA<15:00> onto the bus 0 ns minimum after receiving RDIN and 125 ns maximum after the assertion of TRPLY.

  NOTE
  Block mode transfers must not cross 16-word boundaries.

- **Terminate input transfer**–the bus master receives stable RDATA<15:00> from 200 ns maximum after receiving RRPLY until 20 ns minimum after the negation of RDIN. (The 20 ns minimum represents total minimum receiver delays for RDIN at the slave and RDATA<15:00> at the master.) The bus master negates TDIN 200 ns minimum after receiving RRPLY.

- **Operation completed**–the bus slave negates TRPLY 0 ns minimum after receiving the negation of RDIN. If RBS7 and TREF are both asserted when TRPLY negates, the bus slave prepares for another DIN cycle. RBS7 is stable from 125 ns after RDIN is received until 150 ns after TRPLY negates. If TBS7 and RREF were both asserted when TDIN negated, the bus master asserts TDIN 150 ns minimum after receiving the negation of RRPLY and continues with the timing relationship in send data above. RREF is stable from 75 ns after RRPLY asserts until 20 ns minimum after TDIN negates. (The 0 ns minimum represents total minimum receiver delays for RDIN at the slave and RREF at the master.)

  NOTE
  The bus master must limit itself to not more than eight transfers unless it monitors RDMR. If it monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- **Terminate bus cycle**–if RBS7 and TREF were not both asserted when TRPLY negated, the bus slave removes TDATA<15:00> from the bus 0 ns minimum and 100 ns maximum after negating TRPLY. If TBS7 and RREF were not both asserted when TDIN negated, the bus master negates TSYNC 250 ns minimum after receiving the last assertion of RRPLY and 0 ns minimum after the negation of that RRPLY.

- **Release the bus**–the DMA bus master negates TSACK 0 ns after negation of the last RRPLY. The DMA bus master negates TSYNC 300 ns maximum after it negates TSACK. The DMA bus master must remove RDATA<15:00>, TBS7, and TWTBT from the bus 100 ns maximum after clearing TSYNC.

At this point the block mode transfer is complete, and the bus arbitration logic in the CPU enables processor-generated TSYNC or issues another bus grant (TDMGO) if RDMR is asserted.

C.4.2.2 DATBO Bus Cycle

Before a block mode transfer can occur, the DMA bus master device must request control of the bus. This occurs under conventional Q22-bus protocol.

A Block mode DATBO transfer is executed as follows:

- **Address device memory**–the address is asserted by the bus master on TADDR<21:00> along with the aasertion of TWTBT. The bus master asserts TSYNC 150 ns minimum after gating the address onto the bus.

- **Decode address**–the appropriate memory device recognizes that it must respond to the address on the bus.
- **Send data**–the bus master gates TDATA<15:00> along with TWTBT 100 ns minimum after the assertion of TSYNC. TWTBT is negated. The bus master asserts the first TDOUT 100 ns minimum after gating TDATA<15:00>.

  NOTE
  During DATBO cycles, TBS7 is undefined.

- **Receive data**–the bus slave receives stable data on RDATA<15:00> from 25 ns minimum before receiving RDOUT until 25 ns minimum after receiving the negation of RDOUT. The bus slave asserts TRPLY 0 ns minimum after receiving RDOUT. The bus slave asserts TREF concurrent with TRPLY if, and only if, it is a block mode device which can support another RDOUT after the current RDOUT.

  NOTE
  Block mode transfers must not cross 16-word boundaries.

- **Terminate output transfer**–the bus master negates TDOUT 150 ns minimum after receiving RRPLY.
- **Operation completed**–the bus slave negates TRPLY 0 ns minimum after receiving the negation of RDOUT. If RREF was asserted when TDOUT negated and if the bus master wants to transfer another word, the bus master gates the new data on TDATA<15:00> 100 ns minimum after negating TDOUT. RREF is stable from 75 ns maximum after RRPLY asserts until 20 ns minimum after RDOUT negates. (The 20 ns minimum represents minimum receiver delays for RDOUT at the slave and RREF at the master). The bus master asserts TDOUT 100 ns minimum after gating new data on TDATA<15:00> and 150 ns minimum after receiving the negation of RRPLY. The cycle continues with the timing relationship in receive data above.

  NOTE
  The bus master must limit itself to not more than 8 transfers unless it monitors RDMR. If it monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- **Terminate bus cycle**–if RREF was not asserted when RRPLY negated or if the bus master has no additional data to transfer, the bus master removes data on TDATA<15:00> from the bus 100 ns minimum after negating TDOUT. If RREF was not asserted when TDOUT negated, the bus master negates TSYNC 275 ns minimum after receiving the last RRPLY and 0 ns minimum after the negation of the last RRPLY.
- **Release the bus**–the DMA bus master negates TSACK 0 ns after negation of the last RRPLY. The DMA bus master negates TSYNC 300 ns maximum after it negates TSACK. The DMA bus master must remove TDATA, TBS7, and TWTBT from the bus 100 ns maximum after clearing TSYNC.

At this point the block mode transfer is complete, and the bus arbitration logic in the CPU enables processor-generated TSYNC or issues another bus grant (TDMGO) if RDMR is asserted.

C.4.3 DMA Guidelines

- Systems with memory refresh over the bus must not include devices that perform more than one transfer per acquisition.
- Bus masters that do not use block mode are limited to four DATI, four DATO, or two DATIO transfers per acquisition.
- Block mode bus masters that do not monitor BDMR are limited to eight transfers per acquisition.
- If BDMR is not asserted after the seventh transfer, block mode bus masters that do monitor BDMR may continue making transfers until the bus slave fails to assert BREF, or until they reach the total maximum of 16 transfers. Otherwise, they stop after eight transfers.

C.5 Interrupts

The interrupt capability of the Q22-bus allows an I/O device to temporarily suspend (interrupt) current program execution and divert processor operation to service the requesting device. The processor inputs a vector from the device to start the service routine (handler). Like the device register address, hardware fixes the device vector at locations within a designated range below location 001000. The vector indicates the first of a pair of addresses. The processor reads the contents of the first address, the starting address of the interrupt handler. The contents of the second address is a new processor status word (PS).

The new PS can raise the interrupt priority level, thereby preventing lower-level interrupts from breaking into the current interrupt service routine. Control is returned to the interrupted program when the interrupt handler is ended. The original interrupted program's address (PC) and its associated PS are stored on a stack. The original PC and PS are restored by a return from interrupt (RTI or RTT) instruction at the end of the handler. The use of the stack and the Q22-bus interrupt scheme can allow interrupts to occur within interrupts (nested interrupts), depending on the PS.

Interrupts can be caused by Q22-bus options or the MicroVAX CPU. Those interrupts that originate from within the processor are called traps. Traps are caused by programming errors, hardware errors, special instructions, and maintenance features.

The following Q22-bus signals are used in interrupt transactions:

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

C.5.1 Device Priority

The Q22-bus supports the following two methods of device priority:

- Distributed arbitration — priority levels are implemented on the hardware. When devices of equal priority level request an interrupt, priority is given to the device electrically closest to the processor.
- Position-defined arbitration — priority is determined solely by electrical position on the bus. The closer a device is to the processor, the higher its priority is.

C.5.2 Interrupt Protocol

Interrupt protocol on the Q22-bus has three phases:

- Interrupt request
- Interrupt acknowledge and priority arbitration
- Interrupt vector transfer phase

The interrupt request phase begins when a device meets its specific conditions for interrupt requests. For example, the device is ready, done, or an error occurred. The interrupt enable bit in a device status register must be set. The device then initiates the interrupt by asserting the interrupt request line(s). BIRQ4 L is the lowest hardware priority level and is asserted for all interrupt requests for compatibility with previous Q22-bus processors. The level at which a device is configured must also be asserted. A special case exists for level 7 devices that must also assert level 6. The following list gives the interrupt levels and the corresponding Q22-bus interrrupt request lines. For an explanation, refer to Section C.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 C–11 shows the interrupt request/acknowledge sequence.

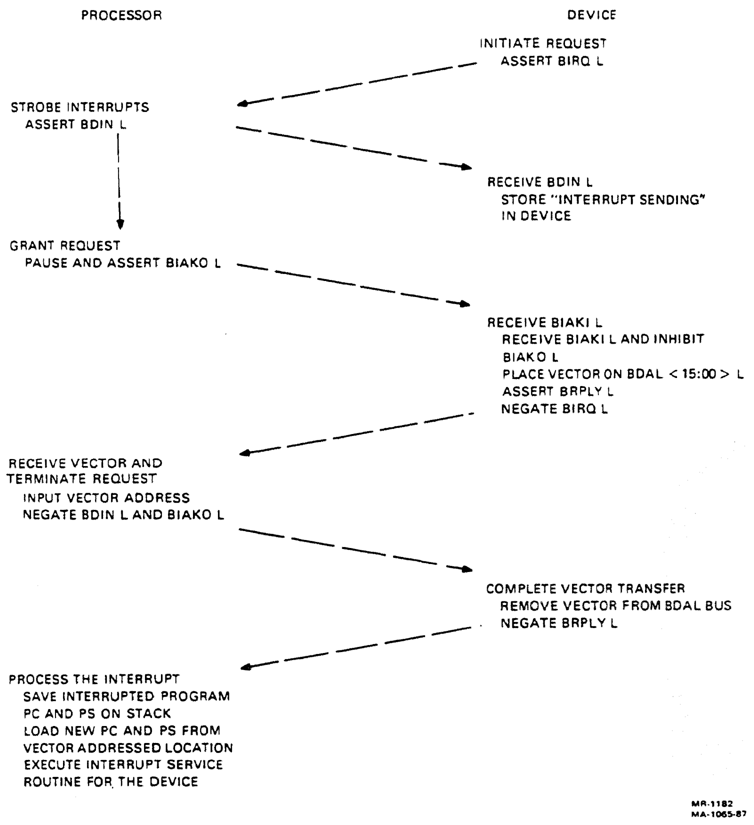

PROCESSOR
DEVICE
INITIATE REQUEST
ASSERT BIRQ L
STROBE INTERRUPTS
ASSERT BDIN L
RECEIVE BDIN L
STORE "INTERRUPT SENDING" IN DEVICE
GRANT REQUEST
PAUSE AND ASSERT BIAKO L
RECEIVE BIAKI L
RECEIVE BIAKI L AND INHIBIT BIAKO L
PLACE VECTOR ON BDAL <15:00> L
ASSERT BRPLY L
NEGATE BIRQ L
RECEIVE VECTOR AND TERMINATE REQUEST
INPUT VECTOR ADDRESS
NEGATE BDIN L AND BIAKO L
COMPLETE VECTOR TRANSFER
REMOVE VECTOR FROM BDAL BUS
NEGATE BRPLY L
PROCESS THE INTERRUPT
SAVE INTERRUPTED PROGRAM PC AND PS ON STACK
LOAD NEW PC AND PS FROM VECTOR ADDRESSED LOCATION
EXECUTE INTERRUPT SERVICE ROUTINE FOR THE DEVICE
MR-1182
MA-1065-87

Figure C–11 Interrupt Request/Acknowledge Sequence

The interrupt request line remains asserted until the request is acknowledged.

During the interrupt acknowledge and priority arbitration phase, the LSI-11/23 processor acknowledges interrupts under the following conditions:

- The device interrupt priority is higher than the current PS<7:5>.
- The processor has completed instruction execution and no additional bus cycles are pending.

The processor acknowledges the interrupt request by asserting BDIN L, and 150 ns minimum later asserting BIAKO L. The device electrically closest to the processor receives the acknowledge on its BIAKI L bus receiver.

At this point, the two types of arbitration must be discussed separately. If the device that receives the acknowledge uses the four-level interrupt scheme, it reacts as follows:

- If not requesting an interrupt, the device asserts BIAKO L and the acknowledge propagates to the next device on the bus.
- If the device is requesting an interrupt, it must check that no higher-level device is currently requesting an interrupt. This is done by monitoring higher-level request lines. The table below lists the lines that need to be monitored by devices at each priority level.

  In addition to asserting levels 7 and 4, level 7 devices must drive level 6. This is done to simplify the monitoring and arbitration by level 4 and 5 devices. In this protocol, level 4 and 5 devices need not monitor level 7 because level 7 devices assert level 6. Level 4 and 5 devices become aware of a level 7 request because they monitor the level 6 request. This protocol has been optimized for level 4, 5, and 6 devices, since level 7 devices are very seldom necessary.

Device Priority Level	**Line(s) Monitored**
4	BIRQ5, BIRQ6
5	BIRQ6
6	BIRQ7
7	–

- If no higher-level device is requesting an interrupt, the acknowledge is blocked by the device. (BIAKO L is not asserted.) Arbitration logic within the device uses the leading edge of BDIN L to clock a flip-flop that blocks BIAKO L. Arbitration is won and the interrupt vector transfer phase begins.
- If a higher-level request line is active, the device disqualifies itself and asserts BIAKO L to propagate the acknowledge to the next device along the bus.

Signal timing must be considered carefully when implementing four-level interrupts (Figure C–12).

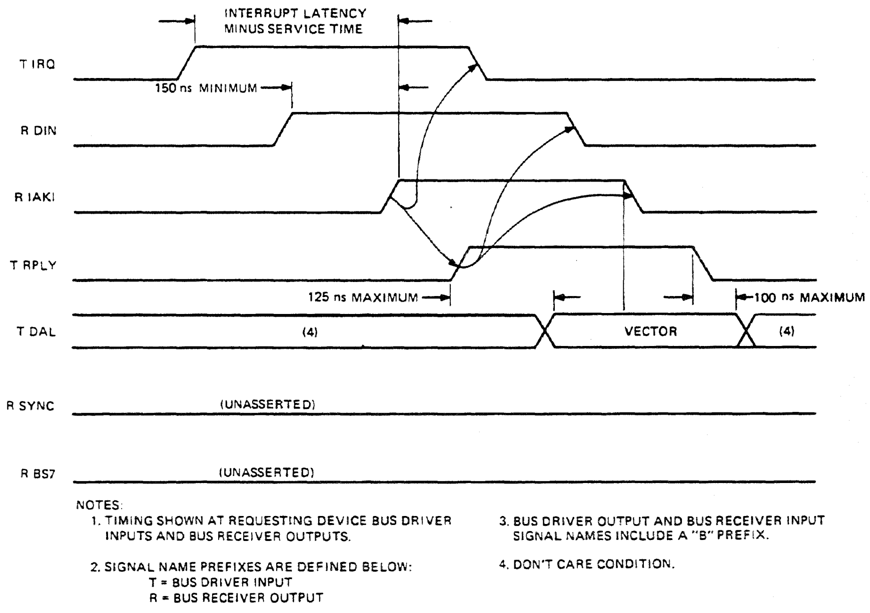

INTERRUPT LATENCY MINUS SERVICE TIME
T IRQ
150 ns MINIMUM
R DIN
R IAKI
T RPLY
125 ns MAXIMUM
100 ns MAXIMUM
T DAL
(4)
VECTOR
(4)
R SYNC
(UNASSERTED)
R BS7
(UNASSERTED)
NOTES:
1. TIMING SHOWN AT REQUESTING DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
T = BUS DRIVER INPUT
R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.
MR-1183
MA-1076-87

Figure C–12 Interrupt Protocol Timing

If a single-level interrupt device receives the acknowledge, it reacts as follows:

- If not requesting an interrupt, the device asserts BIAKO L and the acknowledge propagates to the next device on the bus.
- If the device was requesting an interrupt, the acknowledge is blocked using the leading edge of BDIN L, and arbitration is won. The interrupt vector transfer phase begins.

The interrupt vector transfer phase is enabled by BDIN L and BIAKI L. The device responds by asserting BRPLY L and its BDAL<15:00> L bus driver inputs with the vector address bits. The BDAL bus driver inputs must be stable within 125 ns maximum after BRPLY L is asserted. The processor then inputs the vector address and negates BDIN L and BIAKO L. The device then negates BRPLY L and 100 ns maximum later removes the vector address bits. The processor then enters the device's service routine.

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.

C.5.3 Q22-bus Four-Level Interrupt Configurations

If you have high-speed peripherals and desire better software performance, you can use the four-level interrupt scheme. Both position-independent and position-dependent configurations can be used with the four-level interrupt scheme.

Figure C–13 shows the position-independent configuration. This allows peripheral devices that use the four-level interrupt scheme to be placed in the backplane in any order. These devices must send out interrupt requests and monitor higher-level request lines as described. The level 4 request is always asserted from a requesting device regardless of priority. If two or more devices of equally high priority request an interrupt, the device physically closest to the processor wins arbitration. Devices that use the single-level interrupt scheme must be modified, or placed at the end of the bus, for arbitration to function properly.

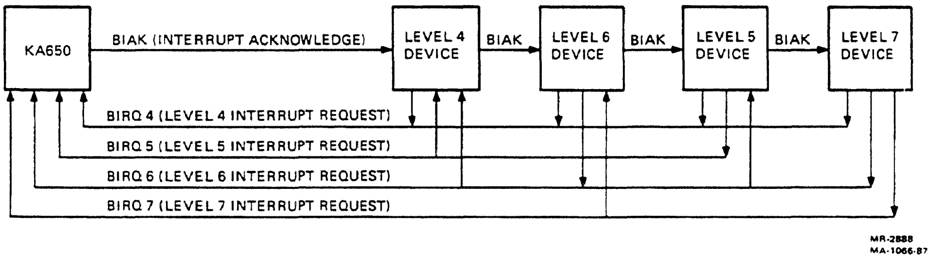

KA650
BIAK (INTERRUPT ACKNOWLEDGE)
LEVEL 4 DEVICE
BIAK
LEVEL 6 DEVICE
BIAK
LEVEL 5 DEVICE
BIAK
LEVEL 7 DEVICE
BIRQ 4 (LEVEL 4 INTERRUPT REQUEST)
BIRQ 5 (LEVEL 5 INTERRUPT REQUEST)
BIRQ 6 (LEVEL 6 INTERRUPT REQUEST)
BIRQ 7 (LEVEL 7 INTERRUPT REQUEST)
MR-2888
MA-1066-87

Figure C–13 Position-Independent Configuration

Figure C–14 shows the position-dependent configuration. This configuration is simpler to implement. A constraint is that peripheral devices must be inserted with the highest priority device located closest to the processor, and the remaining devices placed in the backplane in decreasing order of priority (with the lowest priority devices farthest from the processor). With this configuration, each device has to assert only its own level and level 4. Monitoring higher-level request lines is unnecessary. Arbitration is achieved through the physical positioning of each device on the bus. Single-level interrupt devices on level 4 should be positioned last on the bus.

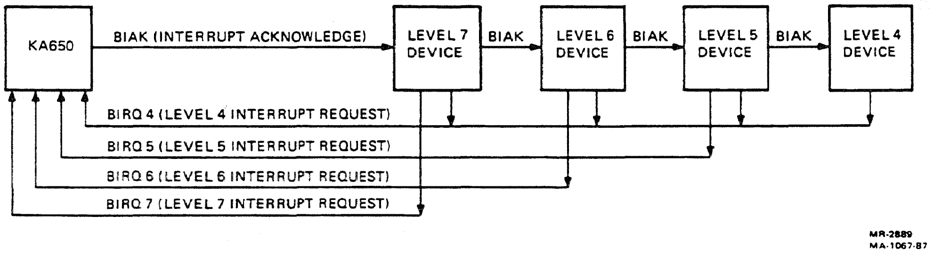

KA650
BIAK (INTERRUPT ACKNOWLEDGE)
LEVEL 7 DEVICE
BIAK
LEVEL 6 DEVICE
BIAK
LEVEL 5 DEVICE
BIAK
LEVEL 4 DEVICE
BIRQ 4 (LEVEL 4 INTERRUPT REQUEST)
BIRQ 5 (LEVEL 5 INTERRUPT REQUEST)
BIRQ 6 (LEVEL 6 INTERRUPT REQUEST)
BIRQ 7 (LEVEL 7 INTERRUPT REQUEST)
MR-2889
MA-1067-87

Figure C–14 Position-Dependent Configuration

C.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

C.6.1 Memory Refresh

If BREF is asserted during the address part of a bus data transfer cycle, it causes all dynamic MOS memories to be addressed simultaneously. The sequence of addresses required for refreshing the memories is determined by the specific requirements for each memory. The complete memory refresh cycle consists of a series of refresh bus transactions. A new address is used for each transaction. A complete memory refresh cycle must be completed within 1 or 2 ms. Multiple data transfers by DMA devices must be avoided since they could delay memory refresh cycles. This type of refresh is done only for memories that do not perform on-board refresh.

C.6.2 Halt

Assertion of BHALT L for at least 25 ns interrupts the processor, which stops program execution and forces the processor unconditionally into console I/O mode.

C.6.3 Initialization

Devices along the bus are initialized when BINIT L is asserted. The processor can assert BINIT L as a result of executing a reset instruction as part of a power-up or power-down sequence. BINIT L is asserted for approximately 10 µs when reset is executed.

C.6.4 Power Status

Power status protocol is controlled by two signals, BPOK H and BDCOK H. These signals are driven by an external device (usually the power supply).

C.6.5 BDCOK H

When asserted, this control indicates that dc power has been stable for at least 3 ms. Once asserted, this line remains asserted until the power fails. It indicates that only 5 µs of dc power reserve remains.

C.6.6 BPOK H

When asserted, this control indicates there is at least an 8 ms reserve of dc power, and that BDCOK H has been asserted for at least 70 ms. Once BPOK has been asserted, it must remain asserted for at least 3 ms. The negation of this line, the first event in the power-fail sequence, indicates that power is failing and that only 4 ms of dc power reserve remains.

C.6.7 Power-Up and Power-Down Protocol

Power-up protocol begins when the power supply applies power with BDCOK H negated. This forces the processor to assert BINIT L. When the dc voltages are stable, the power supply or other external device asserts BDCOK H. The processor responds by clearing the PS, floating-point status register (FPS), and floating-point exception register (FEC). BINIT L is asserted for 12.6 µs, and then negated for 110 µs. The processor continues to test for BPOK H until it is asserted. The power supply asserts BPIK H 70 ms minimum after BDCOK H is asserted. The processor then performs its power-up sequence. Normal power must be maintained at least 3 ms before a power-down sequence can begin.

A power-down sequence begins when the power supply negates BPOK H. When the current instruction is completed, the processor traps to a power-down routine at location 24. The end of the routine is terminated with a halt instruction to avoid any possible memory corruption as the dc voltages decay.

When the processor executes the halt instruction, it tests the BPOK H signal. If BPOK H is negated, the processor enters the power-up sequence. It clears internal registers, generates BINIT L, and continues to check for the assertion of BPOK H. If it is asserted and dc voltages are still stable, the processor performs the rest of the power-up sequence. Figure C–15 shows power-up and power-down timing.

NOTE
The KA655 does not follow this protocol. Refer to Section 3.7.5 for a description of KA655 initialization.

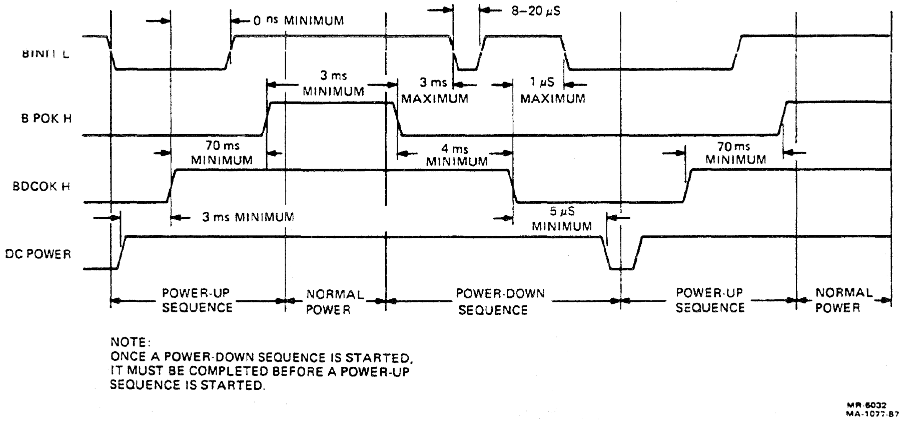


Figure C–15 Power-Up and Power-Down Timing

C.7 Q22-bus Electrical Characteristics

The input and output logic levels for Q22-bus signals are given in Section C.7.1.

C.7.1 Signal Level Specifications

The signal level specifications for the Q22-bus are as follows:

Input Logic Level

TTL logical low	0.8 Vdc maximum
TTL logical high	2.0 Vdc minimum

Output Logic Level

TTL logical low	0.4 Vdc maximum
TTL logical high	2.4 Vdc minimum

C.7.2 Load Definition

AC loads make up the maximum capacitance allowed per signal line to ground. A unit load is defined as 9.35 pF of capacitance. DC loads are defined as maximum current allowed with a signal line driver asserted or unasserted. A unit load is defined as 210 µA in the unasserted state.

C.7.3 120-Ohm Q22-bus

The electrical conductors interconnecting the bus device slots are treated as transmission lines. A uniform transmission line, terminated in its characteristic impedance, propagates an electrical signal without reflections. Since bus drivers, receivers, and wiring connected to the bus have finite resistance and nonzero reactance, the transmission line impedance is not uniform, and introduces distortions into pulses propagated along it. Passive components of the Q22-bus (such as wiring, cabling, and etched signal conductors) are designed to have a nominal characteristic impedance of 120 ohms.

The maximum length of interconnecting cable, excluding wiring within the backplane, is limited to 4.88 m (16 feet).

C.7.4 Bus Drivers

Devices driving the 120-ohm Q22-bus must have open collector outputs and meet the following specifications:

DC Specifications

- Output low voltage when sinking 70 mA of current is 0.7 V maximum.
- Output high leakage current when connected to 3.8 Vdc is 25 µA (even if no power is applied, except for BDCOK H and BPOK H).
- These conditions must be met at worst-case supply temperature, and input signal levels.

AC Specifications

- Bus driver output pin capacitance load should not exceed 10 pF.
- Propagation delay should not exceed 35 ns.
- Skew (difference in propagation time between slowest and fastest gate) should not exceed 25 ns.
- Transition time (from 10% to 90% for positive transition—rise time, from 90% to 10% for negative transition—fall time) must be no faster than 10 ns.

C.7.5 Bus Receivers

Devices that receive signals from the 120-ohm Q22-bus must meet the following requirements:

DC Specifications

- Input low voltage maximum is 1.3 V.
- Input high voltage minimum is 1.7 V.
- Maximum input current when connected to 3.8 Vdc is 80 µA (even if no power is applied).

These specifications must be met at worst-case supply voltage, temperature, and output signal conditions.

AC Specifications

- Bus receiver input pin capacitance load should not exceed 10 pF.
- Propagation delay should not exceed 35 ns.
- Skew (difference in propagation time between slowest and fastest gate) should not exceed 25 ns.

C.7.6 Bus Termination

The 120-ohm Q22-bus must be terminated at each end by an appropriate terminator, as shown in Figure C–16. This is to be done as a voltage divider with its Thevenin equivalent equal to 120 ohms and 3.4 V (nominal). This type of termination is provided by an REV11-A refresh/boot/terminator, BDV11-AA, KPV11-B, TEV11, or by certain backplanes and expansion cards.

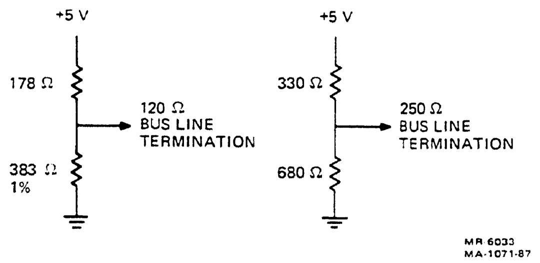


Figure C–16 Bus Line Terminations

Each of the several Q22-bus lines (all signals whose mnemonics start with the letter B) must see an equivalent network with the following characteristics at each end of the bus:

Bus Termination Characteristic	Value
Input impedance (with respect to ground)	120 ohm +5%, -15%
Open circuit voltage	3.4 Vdc +5%
Capacitance load	Not to exceed 30 pF

NOTE
The resistive termination can be provided by the combination of two modules. (The processor module supplies 220 ohms to ground. This, in parallel with another 220-ohm card, provides 120 ohms.) Both terminators must reside physically within the same backplane.

C.7.7 Bus Interconnecting Wiring

The following sections give specific information about bus interconnecting wiring.

C.7.7.1 Backplane Wiring

The wiring that connects all device interface slots on the Q22-bus must meet the following specifications:

- The conductors must be arranged so that each line exhibits a characteristic impedance of 120 ohms (measured with respect to the bus common return).
- Crosstalk between any two lines must be no greater than 5%. Note that worst-case crosstalk is manifested by simultaneously driving all but one signal line and measuring the effect on the undriven line.
- DC resistance of the signal path, as measured between the near-end terminator and the far-end terminator module (including all intervening connectors, cables, backplane wiring, and connector-module etch) must not exceed 20 ohms.
- DC resistance of the common return path, as measured between the near-end terminator and the far-end terminator module (including all intervening connectors, cables, backplane wiring 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.

C.7.7.2 Intrabackplane Bus Wiring

The wiring that connects the bus connector slots within one contiguous backplane is part of the overall bus transmission line. Owing to implementation constraints, the nominal characteristic impedance of 120 ohms may not be achievable. Distributed wiring capacitance in excess of the amount required to achieve the nominal 120-ohm impedance may not exceed 60 pF per signal line per backplane.

C.7.7.3 Power and Ground

Each bus interface slot has connector pins assigned for the following dc voltages. The maximum allowable current per pin is 1.5 A. +5 Vdc must be regulated to 5% with a maximum ripple of 100 mV pp. +12 Vdc must be regulated to 3% with a maximum ripple of 200 mV pp.

- +5 Vdc — three pins (4.5 A maximum per bus device slot)
- +12 Vdc — two pins (3.0 A maximum per bus device slot)
- Ground — eight pins (shared by power return and signal return)

NOTE
Power is not bused between backplanes on any interconnecting bus cables.

C.8 System Configurations

Q22-bus systems can be divided into two types:

- Systems containing one backplane
- Systems containing multiple backplanes

Before configuring any system, three characteristics for each module in the system must be identified.

- Power consumption — +5 Vdc and +12 Vdc are the current requirements.
- AC bus loading — the amount of capacitance a module presents to a bus signal line. AC loading is expressed in terms of ac loads, where one ac load equals 9.35 pF of capacitance.
- DC bus loading—the amount of dc leakage current a module presents to a bus signal when the line is high (undriven). DC loading is expressed in terms of dc loads, where one dc load equals 210 µA (nominal).

Power consumption, ac loading, and dc loading specifications for each module are included in the *Microcomputer Interfaces Handbook*.

NOTE
The ac and dc loads and the power consumption of the processor module, terminator module, and backplane must be included in determining the total loading of a backplane.

Rules for configuring single-backplane systems are as follows:

- When using a processor with 220-ohm termination, the bus can accommodate modules that have up to 20 ac loads before additional termination is required (Figure C–17). If more than 20 ac loads are included, the other end of the bus must be terminated with 120 ohms, and then up to 35 ac loads may be present.

- With 120-ohm processor termination, up to 35 ac loads can be used without additional termination. If 120-ohm bus termination is added, up to 45 ac loads can be configured in the backplane.
- The bus can accommodate modules up to 20 dc loads (total).
- The bus signal lines on the backplane can be up to 35.6 cm (14 inches) long.

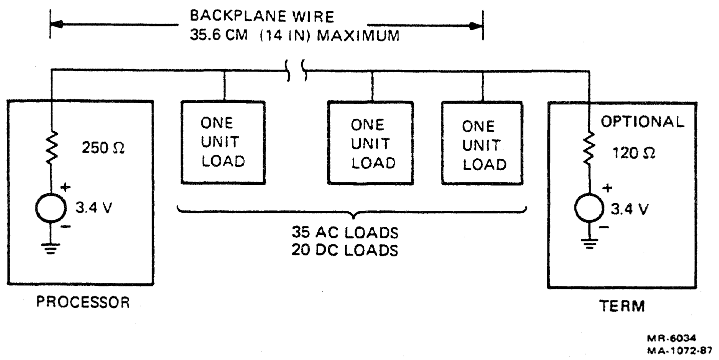
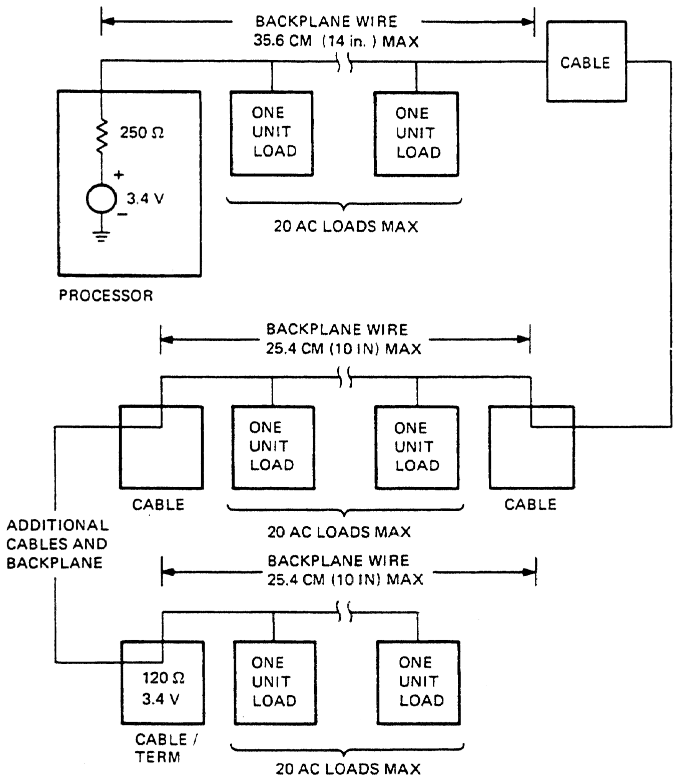


Figure C–17 Single-Backplane Configuration

Rules for configuring multiple backplane systems are as follows:

- Figure C–18 shows that up to three backplanes can make up the system.
- The signal lines on each backplane can be up to 25.4 cm (10 inches) long.
- Each backplane can accommodate modules that have up to 22 ac loads. Unused ac loads from one backplane may not be added to another backplane if the second backplane loading exceeds 22 ac loads. It is desirable to load backplanes equally, or with the highest ac loads in the first and second backplanes.



NOTES:
1. TWO CABLES (MAX) 4.88 M (16 FT) (MAX) TOTAL LENGTH.
2. 20 DC LOADS TOTAL (MAX).

MR-6035
MA-1073-87

Figure C–18 Multiple Backplane Configuration

- DC loading of all modules in all backplanes cannot exceed 20 loads.
- Both ends of the bus must be terminated with 120 ohms. This means the first and last backplanes must have an impedance of 120 ohms. To achieve this, each backplane can be lumped together as a single point. The resistive termination can be provided by a combination of two modules in the backplane – the processor providing 220 ohms to ground in parallel with an expansion paddle card providing 250 ohms to give the needed 120-ohm termination.

  Alternately, a processor with 120-ohm termination would need no additional termination on the paddle card to attain 120 ohms in the first box. The 120-ohm termination in the last box can be provided in two ways: the termination resistors may reside either on the expansion paddle card, or on a bus termination card (such as the BDV11).
- The cable(s) connecting the first two backplanes is 61 cm (2 feet) or more in length.
- The cable(s) connecting the second backplane to the third backplane is 122 cm (4 feet) longer or shorter than the cable(s) connecting the first and second backplanes.
- The combined length of both cables cannot exceed 4.88 m (16 feet).
- The cables used must have a characteristic impedance of 120 ohms.

C.8.1 Power Supply Loading

Total power requirements for each backplane can be determined by obtaining the total power requirements for each module in the backplane. Obtain separate totals for +5 V and +12 V power. Power requirements for each module are specified in the *Microcomputer Interfaces Handbook*.

When distributing power in multiple backplane systems, do not attempt to distribute power 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.

C.9 Module Contact Finger Identification

DIGITAL's plug-in modules all use the same contact finger (pin) identification system. A typical pin is shown in Figure C–19.

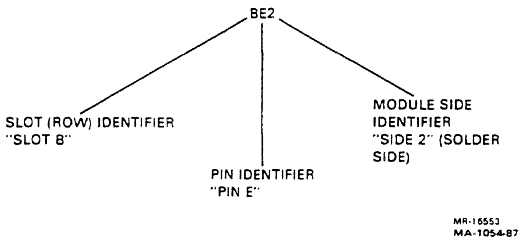


Figure C–19 Typical Pin Identification System

The Q22-bus is based on the use of quad-height modules that plug into a 2-slot bus connector. Each slot contains 36 lines (18 lines on both the component side and the solder side of the circuit board).

Slots, row A, and row B include a numeric identifier for the side of the module. The component side is designated side 1, the solder side is designated side 2, as shown in Figure C–20.

Letters ranging from A through V (excluding G, I, O, and Q) identify a particular pin on a side of a slot. Table C–7 lists and identifies the bus pins of the quad-height module. A bus pin identifier ending with a 1 is found on the component side of the board, while a bus pin identifier ending with a 2 is found on the solder side of the board.

The positioning notch between the two rows of pins mates with a protrusion on the connector block for correct module positioning.

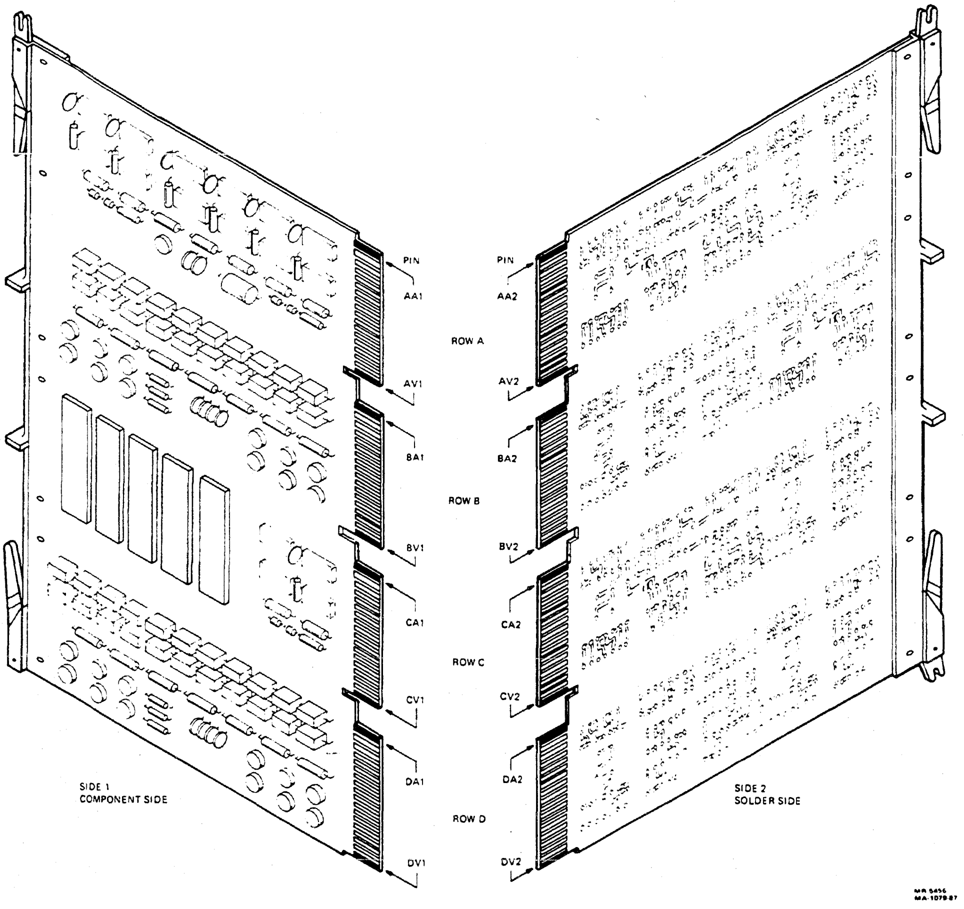


Figure C–20 Quad-Height Module Contact Finger Identification

The dimensions for a typical Q22-bus module are represented in Figure C–21.

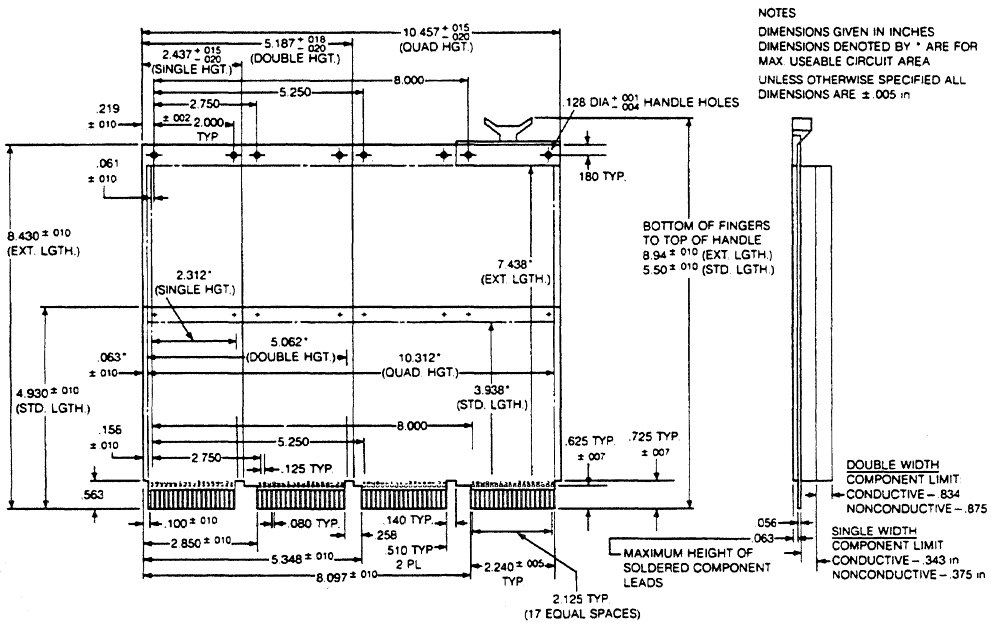


Figure C–21 Typical Q22-bus Module Dimensions

Table C–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.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AE1	SSPARE1 (alternate +5B)	Special spare — not assigned or bused in DIGITAL's cable or backplane assemblies. Available for user connection. Optionally, this pin can be used for +5 V battery (+5 B) back-up power to keep critical circuits alive during power failures. A jumper is required on Q22-bus options to open (disconnect) the +5 B circuit in systems that use this line as SSPARE1.
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.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AN1	BDMR L	DMA request — a device asserts this signal to request bus mastership. The processor arbitrates bus mastership between itself and all DMA devices on the bus. If the processor is not bus master (it has completed a bus cycle and BSYNC L is not being asserted by the processor), it grants bus mastership to the requesting device by asserting BDMGO L. The device responds by negating BDMR L and asserting BSACK L.
AP1	BHALT L	Processor halt — when BHALT L is asserted for at least 25 µs, the processor services the halt interrupt and responds by halting normal program execution. External interrupts are ignored but memory refresh interrupts in Q22-bus operations are enabled if W4 on the M7264 and M7264-YA processor modules is removed and DMA request/grant sequences are enabled. The processor executes the ODT microcode, and the console device operation is invoked.
AR1	BREF L	Memory refresh — asserted by a DMA device. This signal forces all dynamic MOS memory units requiring bus refresh signals to be activated for each BSYNC L/BDIN L bus transaction. It is also used as a control signal for block mode DMA. **CAUTION** **The user must avoid multiple DMA data transfers (burst or hot mode) that could delay refresh operation if using DMA refresh. Complete refresh cycles must occur once every 1.6 ms if required.**

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AS1	+12 B or +5 B	+12 Vdc or +5 V battery back-up power to keep critical circuits alive during power failures. This signal is not bused to BS1 in all of DIGITAL's backplanes. A jumper is required on all Q22-bus options to open (disconnect) the backup circuit from the bus in systems that use this line at the alternate voltage.
AT1	GND	Ground — system signal ground and dc return.
AU1	PSPARE 1	Spare — not assigned. Customer usage not recommended. Prevents damage when modules are inserted upside down.
AV1	+5 B	+5 V battery power — secondary +5 V power connection. Battery power can be used with certain devices.
BA1	BDCOK H	DC power OK — a power supply generated signal that is asserted when the available dc voltage is sufficient to sustain reliable system operation.
BB1	BPOK H	Power OK — asserted by the power supply 70 ms after BDCOK is negated when ac power drops below the value required to sustain power (approximately 75% of nominal). When negated during processor operation, a power-fail trap sequence is initiated.
BC1	SSPARE4 BDAL18 L (22-bit only)	Special spare in the Q22-bus — not assigned. Bused in 22-bit cable and backplane assemblies. Available for user interconnection.
BD1	SSPARE5 BDAL19 L (22-bit only)	**CAUTION** **These pins may be used by manufacturing as test points in some options.**
BE1	SSPARE6 BDAL20 L	In the Q22-bus, these bused address lines are address lines <21:18>. Currently not used during data time.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
BF1	SSPARE7 BDAL21 L	In the Q22-bus, these bused address lines are address lines <21:18>. Currently not used during data time.
BH1	SSPARE8	Special spare — not assigned or bused in DIGITAL's cable and backplane assemblies. Available for user interconnection.
BJ1	GND	Ground — system signal ground and dc return.
BK1 BL1	MSPAREB MSPAREB	Maintenance spare — normally connected together on the backplane at each option location (not a bused connection).
BM1	GND	Ground — system signal ground and dc return.
BN1	BSACK L	This signal is asserted by a DMA device in response to the processor's BDMGO L signal, indicating that the DMA device is bus master.
BP1	BIRQ7 L	Interrupt request priority level 7.
BR1	BEVNT L	External event interrupt request — when asserted, the processor responds by entering a service routine through vector address 1008. A typical use of this signal is as a line time clock (LTC) interrupt.
BS1	+12 B	+12 Vdc battery back-up power (not bused to AS1 in all of DIGITAL's backplanes).
BT1	GND	Ground — system signal ground and dc return.
BU1	PSPARE2	Power spare 2 — not assigned a function and not recommended for use. If a module is using -12 V (on pin AB2), and, if the module is accidentally inserted upside down in the backplane, -12 Vdc appears on pin BU1.
BV1	+5	+5 V power — normal +5 Vdc system power.
AA2	+5	+5 V power — normal +5 Vdc system power.

Table C–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.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
AJ2	BSYNC L	Synchronize — BSYNC L is asserted by the bus master device to indicate that it has placed an address on BDAL<0:17> L. The transfer is in process until BSYNC L is negated.
AK2	BWTBT L	Write/byte — BWTBT L is used in two ways to control a bus cycle. • It is asserted at the leading edge of BSYNC L to indicate that an output sequence (DATO or DATOB), rather than an input sequence, is to follow. • It is asserted during BDOUT L, in a DATOB bus cycle, for byte addressing.
AL2	BIRQ4 L	Interrupt request priority level 4 — a level 4 device asserts this signal when its interrupt enable and interrupt request flip-flops are set. If the PS word bit 7 is 0, the processor responds by acknowledging the request by asserting BDIN L and BIAKO L.
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.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
		If these conditions are not met, the device asserts BIAKO L to the next device on the bus. This process continues in a daisy chain fashion until the device with the highest interrupt priority receives the interrupt acknowledge signal.
AP2	BBS7 L	Bank 7 select — the bus master asserts this signal to reference the I/O page (including that part of the page reserved for nonexistent memory). The address in BDAL<0:12> L when BBS7 L is asserted is the address within the I/O page.
AR2 AS2	BDMGI L BDMGO L	Direct memory access grant — the bus arbitrator asserts this signal to grant bus mastership to a requesting device, according to bus mastership protocol. The signal is passed in a daisy-chain from the arbitrator (as BDMGO L) through the bus to BDMGI L of the next priority device (the device electrically closest on the bus). This device accepts the grant only if it requested to be the bus master (by a BDMR L). If not, the device passes the grant (asserts BDMGO L) to the next device on the bus. This process continues until the requesting device acknowledged the grant. **CAUTION** **DMA device transfers must not interfere with the memory refresh cycle.**
AT2	BINIT L	Initialize — this signal is used for system reset. All devices on the bus are to return to a known, initial state; that is, registers are reset to zero, and logic is reset to state 0. Exceptions should be completely documented in programming and engineering specifications for the device.

Table C–7 (Cont.) Bus Pin Identifiers

Bus Pin	Signal	Definition
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.

D Acronyms

This appendix lists and defines the acronyms that are most frequently used in this manual.

ACRONYM	DEFINITION
ACV	Access control violation
AIE	Alarm interrupt enable
ANSI	American National Standards Institute
AP	Argument pointer
ASTLVL	Asynchronous system trap level
BBU	Battery back-up unit
BCD	Binary coded decimal
BDR	Boot and diagnostic register
BM	Byte mask
BRS	Baud rate select signals
CADR	Cache disable register
CMCTL	CVAX memory controller chip
CPMBX	Console program mailbox
CQBIC	CVAX Q22-bus interface chip
CRC	Cyclic redundancy check
CSR	Control and status register
CSTD	Console storage transmit data
CSTS	Console storage transmit status
DEAR	DMA error address register
DIP	Dual in-line package
DM	Data mode
DMA	Direct memory access

ACRONYM	DEFINITION
DSE	Daylight saving enable
EDITPC	EDIT packed to character string
EIA	Electronic Industries Association
EPROM	Erasable programmable read-only memory
ERR	Error signal
ESP	Executive stack pointer
FP	Frame pointer
FPA	Floating-point accelerator
FPU	Floating-point unit
GPR	General purpose register
ICCS	Interval clock control and status register
ICR	Interval count register
IORESET	I/O bus reset register
IPCR	Interprocessor communication register
IPL	Interrupt priority level
IPR	Internal processor register
ISP	Interrupt stack pointer
KSP	Kernel stack pointer
LSI	Large scale integration
MAPEN	Memory management mapping enable register
MBRK	Microprogram break register
MBZ	Must be zero
MCESR	Machine check error summary register
MCS	Multinational character set
MFPR	Move from process register
MMU	Memory management unit
MOP	Maintenance operation protocol
MOS	Metal oxide semiconductor
MSER	Memory system error register
MTPR	Move to process register
NICR	Next interval count register
NXM	Nonexistent memory
P0BR	P0 base register
P1BR	P1 base register
PC	Program counter
PCB	Process control block
PCBB	Process control block base
PIE	Periodic interrupt enable

ACRONYM	DEFINITION
P0LR	P0 length register
P1LR	P1 length register
PMR	Performance monitor enable register
P0PT	P0 page table
P1PT	P1 page table
PROM	Programmable read only memory
PSL	Processor status longword
PSW	Processor status word
PTE	Page table entry
QBEAR	Q22-bus error address register
RAM	Random-access memory
RPB	Restart parameter block
RXCS	Console receiver control/status register
RXDB	Console receiver data buffer
SAVPC	Console saved PC register
SAVPSL	Console saved PSL register
SBR	System base register
SCA	System communications architecture
SCB	System control block
SCBB	System control block base
SID	System identification register
SIE	System identification extension
SIRR	Software interrupt request register
SISR	Software interrupt summary register
SLR	System length register
SLU	Serial line unit
SP	Stack pointer
SPT	System page table
SQWE	Square-wave enable
SSC	System support chip
SSP	Supervisor stack pointer
TBCHK	Translation buffer check register
TBDATA	Translation buffer data
TBDR	Translation buffer disable register
TBIA	Translation buffer invalidate all
TBIS	Translation buffer invalidate single
TNV	Translation not valid
TODR	Time-of-year register

ACRONYM	DEFINITION
TXCS	Console transmit control/status register
TXDB	Console transmit data buffer
UIE	Update interrupt enable
UIP	Update in progress
USP	User stack pointer
VLSI	Very large scale integration
VPN	Virtual page number
VRT	Valid RAM and time
VMB	Virtual memory bootstrap
XFC	Extended function call
ZIP	Zig-zag in-line package

Index

KA650 CPU Module
Technical Manual
EK-KA650-UG-003

READER'S COMMENTS

Your comments and suggestions will help us in our efforts to improve the quality of our publications.

1. **How did you use this manual? (Circle your response.)**

(a) Installation	(c) Maintenance	(e) Training
(b) Operation/use	(d) Programming	(f) Other (Please specify.) ______

2. **Did the manual meet your needs? Yes ☐ No ☐ Why?** ______

3. **Please rate the manual on the following categories. (Circle your response.)**

	Excellent	Good	Fair	Poor	Unacceptable
Accuracy	5	4	3	2	1
Clarity	5	4	3	2	1
Completeness	5	4	3	2	1
Table of Contents, Index	5	4	3	2	1
Illustrations, examples	5	4	3	2	1
Overall ease of use	5	4	3	2	1

4. **What things did you like *most* about this manual?** ______

5. **What things did you like *least* about this manual?** ______

6. **Please list and describe any errors you found in the manual.**

Page	**Description/Location of Error**

Name ______	Job Title ______
Street ______	Company ______
City ______	Department ______
State/Country ______	Telephone Number ______
Postal (ZIP) Code ______	Date ______

THANK YOU FOR YOUR COMMENTS AND SUGGESTIONS.

Please do not use this form to order manuals. Contact your representative at Digital Equipment Corporation or (in the USA) call our DECdirect™ department at this toll-free number: 800-258-1710.

MYO

FOLD HERE AND TAPE. DO NOT STAPLE.

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Educational Services/Quality Assurance
12 Crosby Drive BUO/E08
Bedford, MA 01730-1493
USA

FOLD HERE AND TAPE. DO NOT STAPLE.