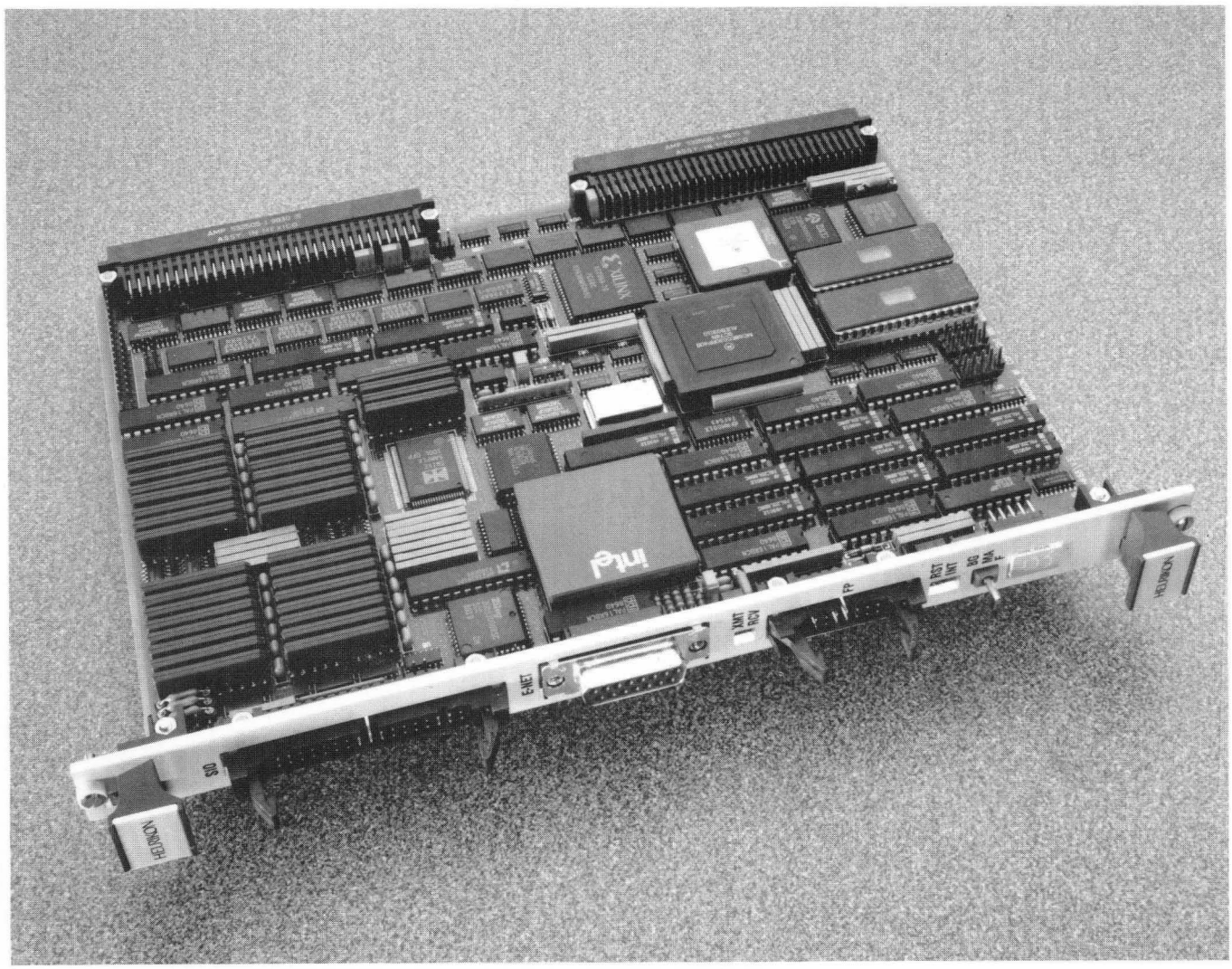
USER'S MANUAL
Revision A (Preliminary)
June 1991

HK68/V3D

68030-based VMEbus Single-board Computer



HEURIKON CORP.®
OPEN SYSTEMS : : OPEN TOOLS

HK68/V3D

VMEbus 68030-based Single Board Computer

USER'S MANUAL
Revision A (Preliminary)
June 1991

HEURIKON CORP.®
OPEN SYSTEMS : : OPEN TOOLS

REVISION HISTORY

Revision Level	Principal Changes	Date of Publication	Board Revision Level
A (Preliminary)	First publication	June 1991	EP0

Contents

Figures

Tables

1

Overview

1.1 INTRODUCTION

The HK68/V3D is a VMEbus single-board computer based on the Motorola 68030. The 68EC030, which has all the features of the 68030 except a memory management unit, can be ordered as an option. The HK68/V3D is fully VMEbus compatible; it also has two ports for serial I/O and a control panel interface. SCSI and Ethernet ports, and the 68882 floating point coprocessor, are optional.

1.2 COMPONENTS AND FEATURES

MPU

Motorola 68030 or 68EC030 microprocessor chip, running at 32 MHz
32-bit internal architecture
32-bit address and data paths
32 address lines
4-gigabyte addressing range
256-byte data cache
256-byte instruction cache
MMU standard (option for 68EC030 MPU without MMU)

FPU option

Optional 68882 floating point coprocessor
Uses the IEEE-P754 Binary Floating Point Standard

RAM

2-, 4-, or 16-megabyte capacity
One parity bit per byte
Uses 256K × 4 or 1024K × 4 DRAMs.
Hardware refresh

EPROM

Two ROM sockets
2-megabyte total capacity
Page-addressable ROM and EEPROM capability

NV-RAM	Nonvolatile static RAM for programmable functions 256 × 4 configuration Internal EEPROM 100-year retention 10,000 store cycle lifetime
VMEbus	32-bit addressing (4 gigabyte range) 32-bit data bus, compatible with 8-bit boards Seven bus interrupts.
Mailbox	Allows remote control of the HK68/V3D via specified VMEbus addresses MPU halt, reset, interrupt, and on-card bus lock functions
LEDs	One 7-segment LED under software control Three MPU/BUS status LEDs for master, bus (slave), and fail Two LEDs for Ethernet transmit and receive
Serial I/O	Two serial I/O ports (Zilog Z8530 Serial Communication Controller) Separate baud rate generators for each port Asynchronous and synchronous modes RS-232C interface, RS-422 option.
CIO	Zilog Z8536 counter/timer and parallel I/O unit; three 16-bit counter/timers Three parallel ports for on-card control functions
SCSI option	ANSI X3T9.2-compatible Small Computer System Interface (SCSI) controller Supports up to eight disk drive controllers or other devices. Synchronous protocol support
Ethernet option	Intel 82596CA Ethernet controller On-chip DMA and memory management to handle Ethernet transfers without host CPU intervention Ethernet transfers conform to the IEEE-802.3 or Ethernet 1.0 standard.
RTC option	Optional real-time clock module for time-of-day maintenance

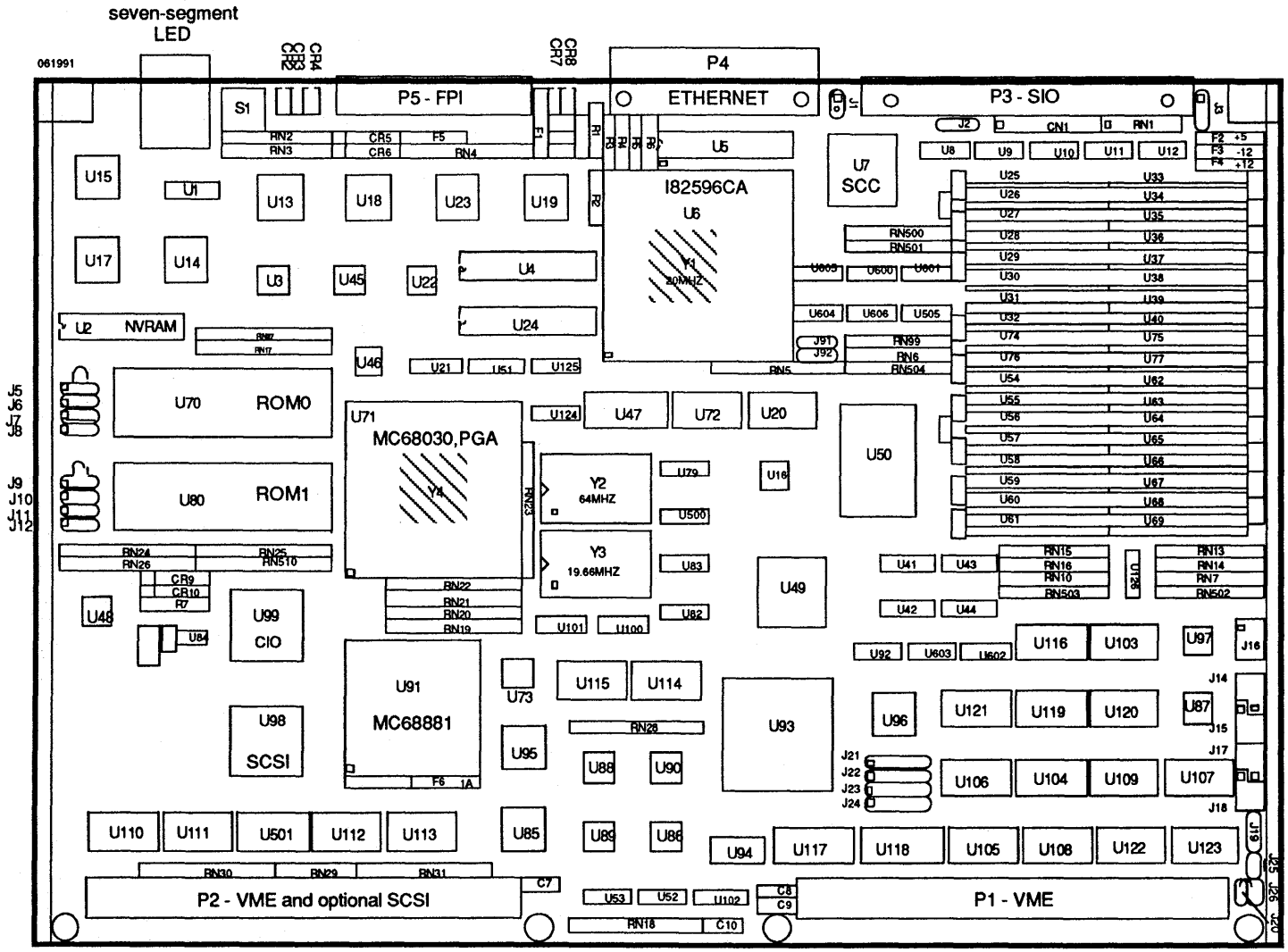


FIGURE 1-1. Component map

1.3 FUNCTIONAL DESCRIPTION

Principal functional blocks are shown in Figure 1-2.

The VMEbus provides high throughput for data transfers between boards or subsystems on the VMEbus, and is the main conduit for transferring system-level information between processor subsystems.

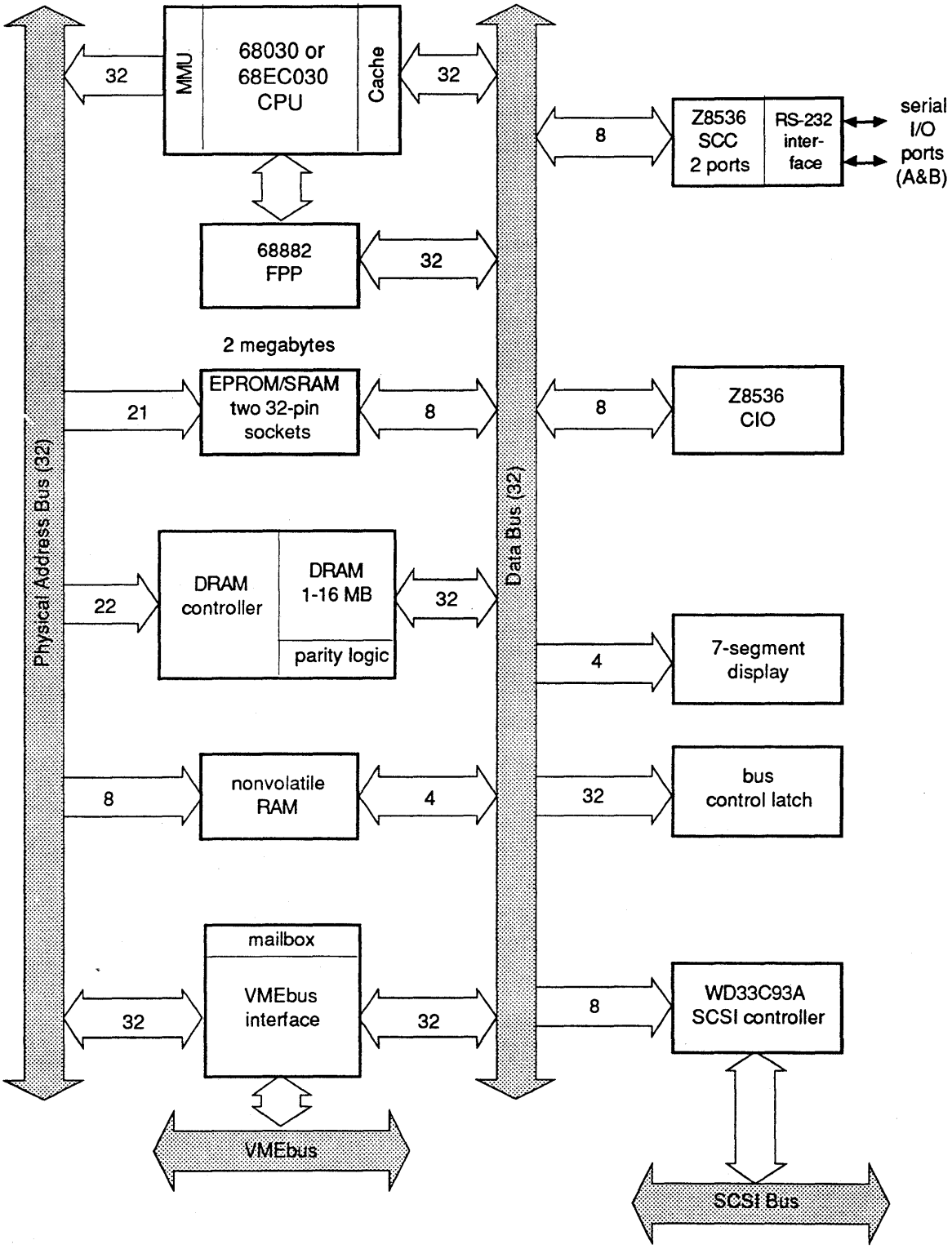


FIGURE 1-2. HK68/V3D block diagram

1.4 JUMPERS, CONNECTORS, AND SWITCHES

1.4.1 Jumpers

The HK68/V3D has 24 configurable jumpers. Jumpers J91 and J92 are factory-set and should not be altered.

ROM size J5–8 configuration selects ROM size for ROM 0, and J9–12 configuration selects ROM size for ROM1. These jumpers must be set to match each ROM type.

VMEbus arbitration J14 enables or disables VMEbus bus grant level 3 (BG3).
J15 enables or disables VMEbus bus grant level 2 (BG2).
J17 enables or disables VMEbus bus grant level 1 (BG1).
J18 enables or disables VMEbus bus grant level 0 (BG0).

VMEbus request level J16 selects bus request level. This jumper must be configured to match the bus arbitration jumpers, as described in section 7.

VMEbus system reset J19 selects SYSRESET* input to bus or output to bus. Installing J19:1-2 selects SYSRESET* input from the VMEbus.

VMEbus ACFAIL* control Install J20 to monitor ACFAIL* from the VMEbus.

VMEbus slave window size J21–J24 configure address lines A23–A20, as described in section 7.

VMEbus SYSCLK control Install J25 to drive the VMEbus SYSCLK signal.

VMEbus BCLR control Install J26 to drive BCLR from the VMEbus.

Serial I/O J2 selects +12V or -12V for RS-232. Installing J2:2–3 selects +12V (true).

J3 is used to configure port A for Ring Indicator or Data Carrier Detect.

Ethernet J1 selects Ethernet transceiver type. Installing J1 selects full-step mode (Ethernet 1.0, positive differential voltage). Removing J1 selects half-step mode (for IEEE-802.3-type transceivers, for example).

Detailed descriptions of jumpers and standard configurations are shown in Tables 2-2 and 14-2.

1.4.2 Connectors

P1 and P2 P1 and P2 are standard 96-pin VMEbus connectors. P2 is also used for the optional SCSI interface.

P3 P3 is a 34-pin male serial port connector that provides two RS-232 ports.

P4 Standard 15-pin male connector for the Ethernet option

P5 P5 is the front panel interface. P5 is a 14-pin header with a reset input, an interrupt input, and four output signals that can be connected to LED cathodes.

1.4.3 Interrupt or Reset Switch

This switch has two settings. Pressing the switch toward the "INT" side generates an interrupt. Pressing the switch toward the "RST" side resets the HK68/V3D and also resets the VMEbus if the HK68/V3D is jumpered as the VMEbus system controller.

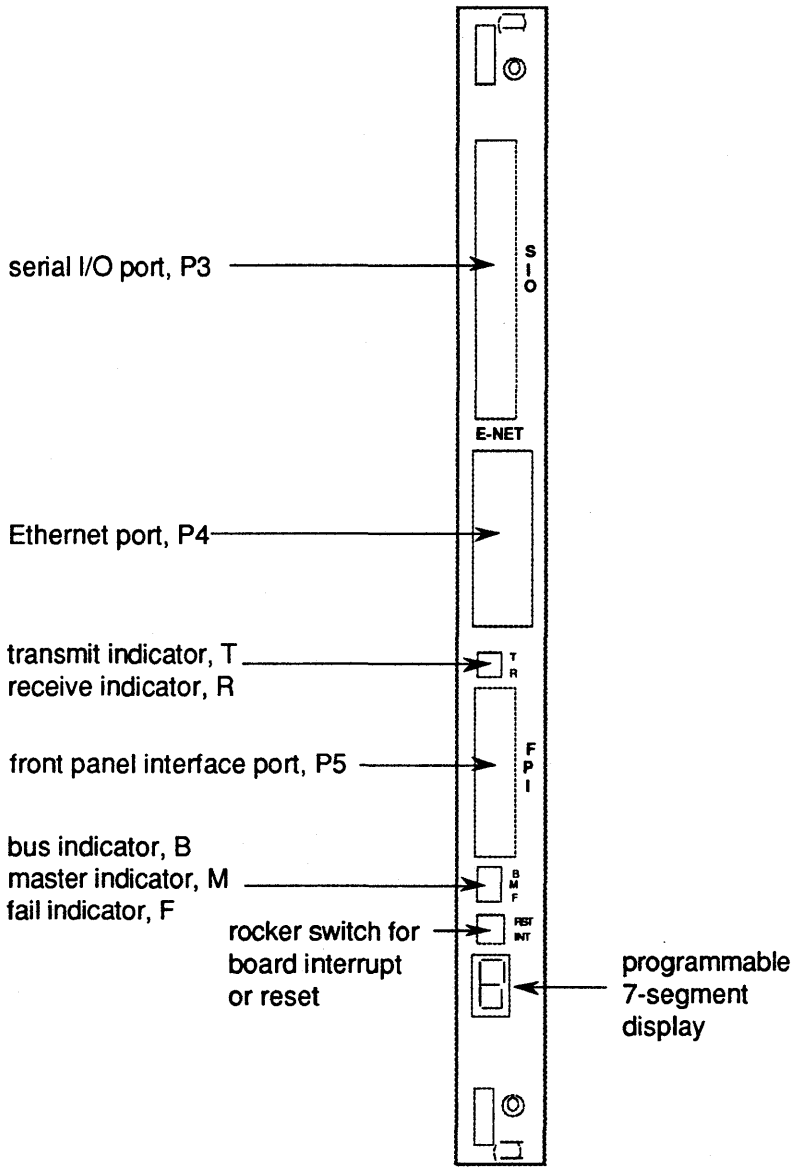


FIGURE 1-3. Front panel

1.5 OVERVIEW OF THE MANUAL

1.5.1 Terminology and Notation

Throughout this manual *byte* refers to 8 bits; *short* refers to 16 bits; *word* and *long word* refer to 32 bits; and *quad word* refers to 4 long words (that is, 128 bits).

Hexadecimal numbers are shown with a subscript *16* and binary numbers with a subscript *2*.

1.5.2 Additional Technical Information

Additional information is available on the HK68/V3D peripheral chips, either from the Heurikon sales department or directly from the chip manufacturers.

This manual describes Heurikon's implementation of the intelligent components of this board. Further information on basic operation and programming can be found in the following documents:

TABLE 1-1
Technical references

Device	Number	Document	HK68/V3D User's Manual Section
MPU	68030	*MC68030 User's Manual,* 2nd ed. (Englewood Cliffs, NJ: Prentice-Hall, 1989). This manual is also available from Heurikon (part number 001M216).	3
FPU	68882	*MC68881/MC68882 Floating Point Coprocessor User's Manual,* 1st ed. (Englewood Cliffs, NJ: Prentice-Hall, 1985). This manual is also available from Heurikon (part number 001M207).	5
CIO	Z8536	*Z8036 Z-CIO/Z8536 CIO Counter/Timer and Parallel I/O Unit Technical Manual* (Campbell, CA: Zilog, Inc., 1987). The user's manual is also available from Heurikon (part number 001M206).	9
Serial Interface	Z8530	*Z8030 Z-bus SCC/Z8530 SCC Serial Communications Controller Technical Manual* (Campbell, CA: Zilog, Inc., 1989). This manual is also available from Heurikon (part number 001M205).	10
SCSI	WD33C93	WD33C93 Technical Specification. This document is also available from Heurikon (part number 001M209).	11
Ethernet Interface	82596CA	*Intel 82596CA User's Manual* (Intel publication number 296443-001) and *Intel 82C501AD Data Sheet.* (Not currently available from Heurikon.)	12
Real-Time Clock	DS1216F	*Dallas Semiconductor 1990-91 Product Data Book* (Dallas, TX: Dallas Semiconductor). (Not currently available from Heurikon.)	13

Please contact our Customer Support Department at 1-800-327-1251 if you have questions. We are prepared to answer general questions and provide help with documentation and specific applications.

2

Setup and Installation

2.1 INTRODUCTION

The HK68/V3D is a general-purpose board that can be used with a power supply, card cage, and terminal as a single-board computer or in a multiprocessor system as a VMEbus slave or master. This section describes steps that should be taken when the board is installed.

CAUTION: **The HK68/V3D uses the P2 connector for VMEbus power and extended addressing, and for the optional SCSI interface. Do not connect P2 to a VSB backplane, or the HK68/V3D could be damaged.**

2.2 UNPACKING

Inspect the board for components that could have loosened during shipment. Save the antistatic bag and box for future shipping or storage.

2.3 RECORDING SERIAL NUMBERS

Before you install the HK68/V3D in a card cage or rack, record the board serial number, the serial number of the operating system, and the version number of the monitor, in case you need them for reference by our service department. The board serial number is inscribed on the edge of the board (Fig. 2-1) . The version number of the monitor is labelled on the monitor ROM, and the serial number of the operating system is labelled on the ROM or tape case.

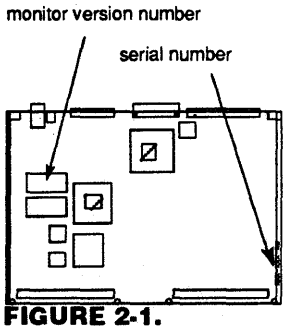


FIGURE 2-1.
Location of serial numbers

HK68/V3D serial number: _____

Operating system serial number: _____

Monitor version: _____

2.4 PROVIDING POWER

Be sure the power supply is sufficient for the board. The HK68/V3D requires about 35 watts maximum. Power requirements for the HK68/V3D are shown in Table 2-1.

TABLE 2-1
Power requirements

Voltage	Current	Usage
+5	7.0 A, max	All logic
+12	20 mA, max	Reset timing, RS-232 interface
-12	20 mA, max	RS-232 interface

Note: All of the "+5" and "Gnd" pins on P1 *and* P2 *must* be connected to ensure proper operation. P2 contains power pins for the VMEbus.

2.5 RESERVING SPACE

The board is a 6U board, 6.299" H × 9.187" W × 0.6" D (233.35 mm W × 160 mm L × 15.25 mm D), that occupies a single slot in a VMEbus card cage. If the board is the VMEbus system controller, it should be installed in the first slot.

2.6 PROVIDING AIR FLOW

CAUTION: **High operating temperatures will cause unpredictable operation. Because of the high chip density, fan cooling is required for all configurations, even when boards are placed on extenders.**

As with any printed circuit board, be sure that air flow to the board is adequate. Recommended air flow rate is about 2 to 3 cubic feet per minute, depending on card cage constraints and other factors. Operating temperature is specified at 0° to 55° C ambient, as measured at the board.

2.7 CHECKING OPERATION

You need the following items to set up and use the Heurikon HK68/V3D.

- ❑ Heurikon HK68/V3D microcomputer board
- ❑ Card cage and power supply
- ❑ Serial interface cable (RS-232)
- ❑ CRT terminal
- ❑ Heurikon EPROMs, which include both monitor and bootstrap

CAUTION: **Do not handle the board unless absolutely necessary.**

Ground your body before touching the HK68/V3D board.

All semiconductors should be handled with care. Static discharges can easily damage the components on the HK68/V3D. Keep the board in an antistatic bag whenever it is out of the system chassis.

CAUTION: **Do not install the board in a rack or remove the board from a rack while power is applied, at risk of damage to the board.**

All products are fully tested before they are shipped from the factory (please contact us if you would like to have current information on mean time between failures). When you receive your HK68/V3D, follow these steps to assure yourself that the system is operational:

1

Read the monitor manual in Appendix A and the operating system literature to become familiar with their features and available tools.

2

Visually inspect the board(s) for components that could have loosened during shipment. Visually inspect the chassis and all cables.

3

Check the jumpers; standard configurations are shown in Figure 2-2 and Table 2-2. The ROM size jumpers are configured to match your ROM (Table 2-3).

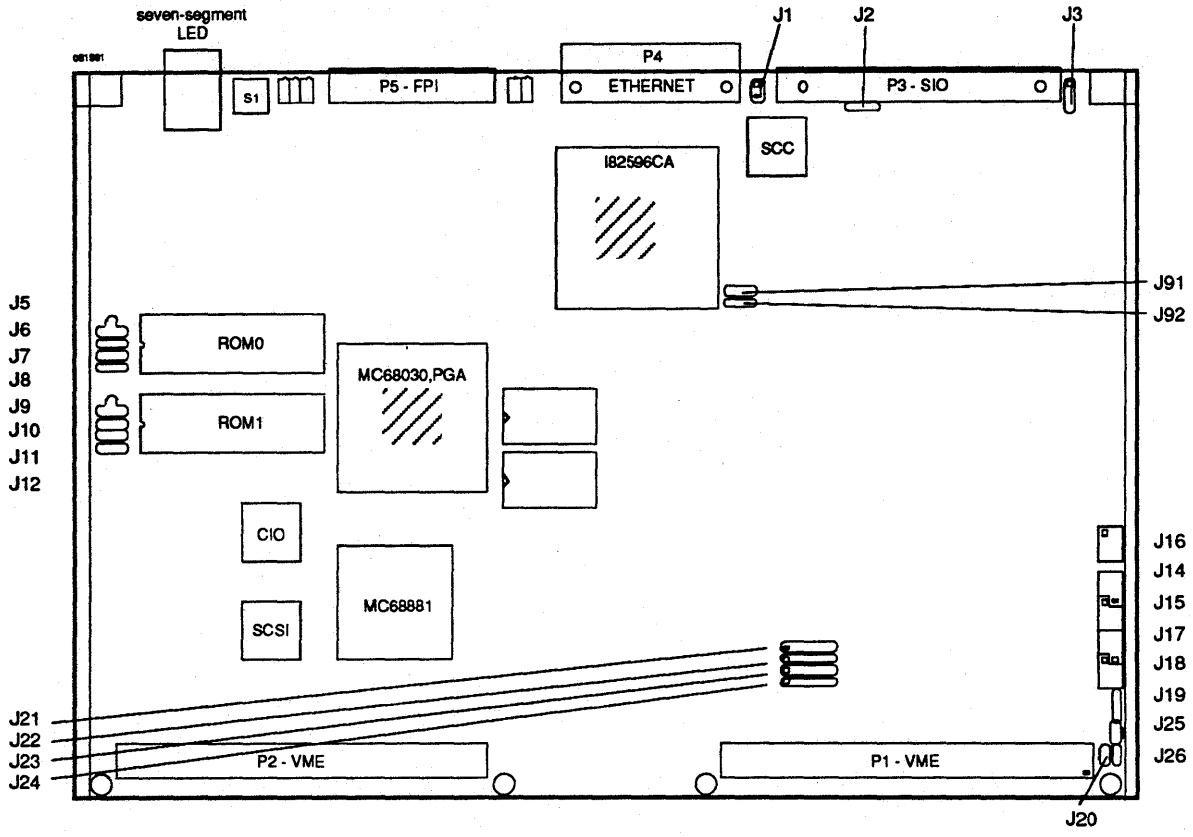


FIGURE 2-2. Guide to jumper locations

TABLE 2-2
Standard jumper settings

Jumper	Standard Configuration	Options	Function	HK68/V3D Manual Section
J1	Installed	J1 installed: + (positive) idle differential voltage on TX lines, full-step mode (for example, for Ethernet 1.0-type transceivers) J1 removed: 0 idle differential voltage on TX lines, half-step mode (for example, for IEEE-802.3-type transceivers)	Selects Ethernet differential voltage	12
J2	J2:1-2 False 	J2:1-2 False (+12V) J2:2-3 True (-12V)	RS-232 handshaking defaults	10
J3	J3:1-2 Ring Indicator 	J3:1-2 Ring Indicator J3:2-3 Data Carrier Detect	Selects Ring Indicator or Data Carrier Detect for SCC Port A.	10

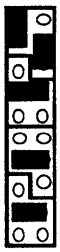
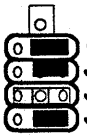
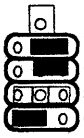
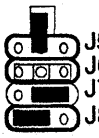
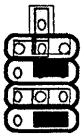
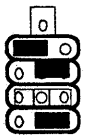
J5–J8	Matches ROM0 size. See Table 2-3.	2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27513 paged, 2864 R/W EEPROM, 2817 R/W EEPROM	Selects ROM 0 size (default is 2764)	5
J9-J12	Matches ROM1 size. See Table 2-3.	2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27513 paged, 2864 R/W EEPROM, 2817 R/W EEPROM	Selects ROM 1 size (default is 2764)	5
J14, J15, J17, J18	Bus Grant Level 3 	Bus Grant Level 3 Bus Grant Level 2 Bus Grant Level 1 Bus Grant Level 0	Selects VMEbus Bus Grant level	7
J16	Bus Request Level 3) 	Bus Request Level 3 Bus Request Level 2 Bus Request Level 1 Bus Request Level 0	VMEbus arbitration (bus request level 3, not system controller)	7
J19	J19:1-2 input from VMEbus 	J19:1-2 input from VMEbus J19:2-3 output to VMEbus	Enables VMEbus SYSRESET*	7
J20	Removed	J20 installed: Allows HK68/V3D to respond to ACFAIL* interrupt. J21 removed: HK68/V3D does not respond to ACFAIL* interrupt.	ACFAIL* connects to VMEbus	7
J21-J24	Matches memory size.	1, 2, 4, 8, or 16 megabytes	VMEbus slave window size	7
J25	Removed	J25 installed: drives SYSCLK J25 removed: does not drive SYSCLK	Disables SYSCLK	7
J26	Removed	J26 installed: HK68/V3D can drive BCLR*. J26 removed: HK68/V3D cannot drive BCLR*.	Disables BCLR*.	7
J91		Factory set for memory configuration. Do not alter.		
J92		Factory set for memory configuration. Do not alter.		

TABLE 2-3
ROM size options

ROM Type	ROM Capacity	Jumper Configuration
2764 27128 27513 paged	64 Kbits (8K × 8) 128 Kbits (16K × 8)	 J5 or J9 J6 or J10 J7 or J11 (either A or B) J8 or J12
27256	256 Kbits (32K × 8)	 J5 or J9 J6 or J10 J7 or J11 (either A or B) J8 or J12
27512	512 Kbits (64K × 8)	 J5 or J9 J6 or J10 J7 or J11 (either A or B) J8 or J12
27010	1 Mbits (128K × 8)	 J5 or J9 J6 or J10 (either A or B) J7 or J11 J8 or J12
27020 27040	2 Mbits (256K × 8) 4 Mbits (512K × 8)	 J5 or J9 J6 or J10 J7 or J11 J8 or J12
27080	8 Mbits (1M × 8)	 J5 or J9 J6 or J10 J7 or J11 J8 or J12
2864 R/W EEPROM	8K × 8	 J5 or J9 (any setting) J6 or J10 J7 or J11 (either A or B) J8 or J12
2817 R/W EEPROM	2K × 8	 J5 or J9 J6 or J10 J7 or J11 (either A or B) J8 or J12

4

Install the HK68/V3D in the VMEbus card cage. Be sure it is seated firmly.

CAUTION: **The HK68/V3D uses the P2 connector for VMEbus power and extended addressing, and for the optional SCSI interface. Do not connect P2 to a VSB backplane, or the HK68/V3D could be damaged.**

5

Connect a CRT terminal to serial port B (port A for the VxWorks operating system), via connector P3. If you are making your own cable, refer to the cable drawing in section 10. Be sure all cables are securely connected.

Set the terminal as follows:

- 9600 baud, full duplex
- Eight data bits (no parity)
- Two stop bits for transmit data
- One stop bit for receive data. If your terminal does not have separate controls for transmit and receive stop bits, select one stop bit for both transmit and receive.

6

Turn the system on.

7

Push the system RESET switch.

If you are using the HK68/V3D monitor or VxWorks, a sign-on message and prompt should appear on the screen. If the prompt does not appear, check your power supply voltages, EPROM jumpering, and CRT cabling.

8

Turn the power off before you remove boards from the card cage. Reconfigure the jumpers as necessary for your application. See section 12 for a summary of I/O device addresses.

- *When the HK68/V3D communicates with other boards over the bus:*

In a VMEbus system that uses multiple boards, one board must be system controller. For example, you might want to configure the HK68/V3D as the system controller in a multiple-board VxWorks system. If the HK68/V3D is the system controller in your system, install it in the first slot.

An example configuration of the VMEbus jumpers is shown in Figure 2-3. Under normal circumstances, the VMEbus system controller card provides the system bus clock and access timer, and participates in the arbitration logic. The HK68/V3D includes a bus timer and single-level VMEbus arbiter logic that is enabled via jumpers J14, J15, J17, and J18. The example shows jumpers J14, J15, J17, and J18 configured for VMEbus single-level arbitration with the HK68/V3D as system controller.

In the example, SYSRESET is configured as an input via jumper J19, even though the HK68/V3D is the system controller, when it is preferable to use an enclosure reset switch for reset. J21–J24 are configured for a 16-megabyte slave window size. J20 is rarely installed; if J20 is installed, the HK68/V3D drives ACFAIL.

The system controller board drives the bus clear (BCLR*) signal, which is used to tell the current bus owner to release control of the bus for a higher priority requester. This option is controller by jumper J26 on the HK68/V3D.

- *When the HK68/V3D is used as a stand-alone board or is not the system controller in a multiple-board system:*

If the HK68/V3D is not the system controller in your system, configure the VMEbus jumpers as shown in Figure 2-4. For example, if the HK68/V3D is the only board in a system, and you are using the HK68/V3D monitor to configure the board it does not need to be configured as a system controller. An example configuration of the VMEbus jumpers is shown in Figure 2-4.

Section 7 contains additional instructions for configuring the HK68/V3D in a VMEbus system.

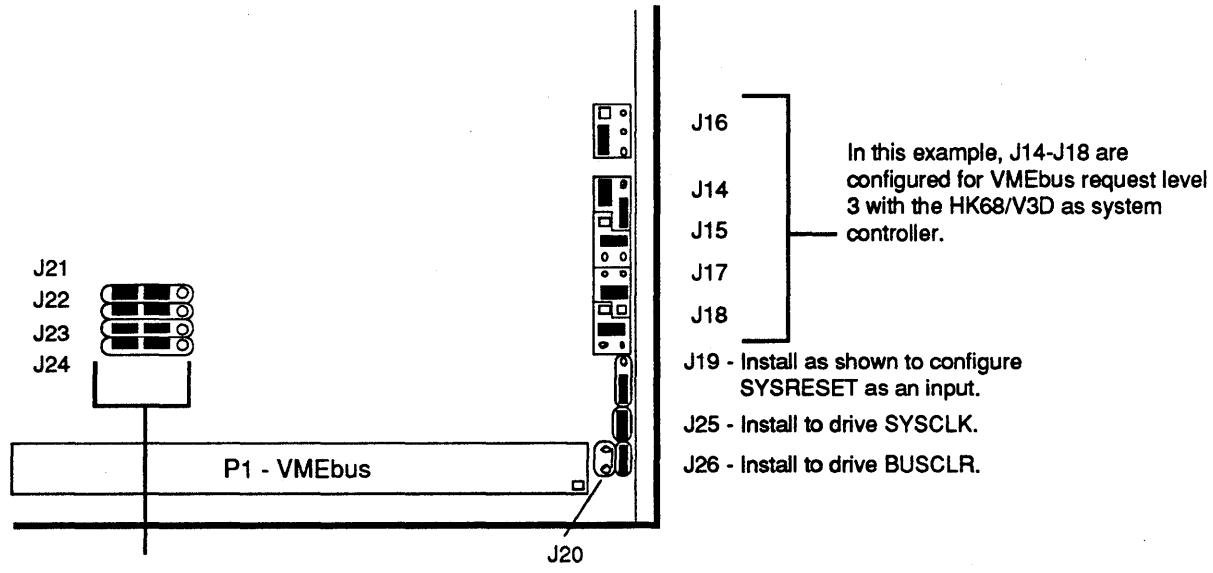


FIGURE 2-3. The HK68/V3D configured as VMEbus system controller via jumpers J14-J18

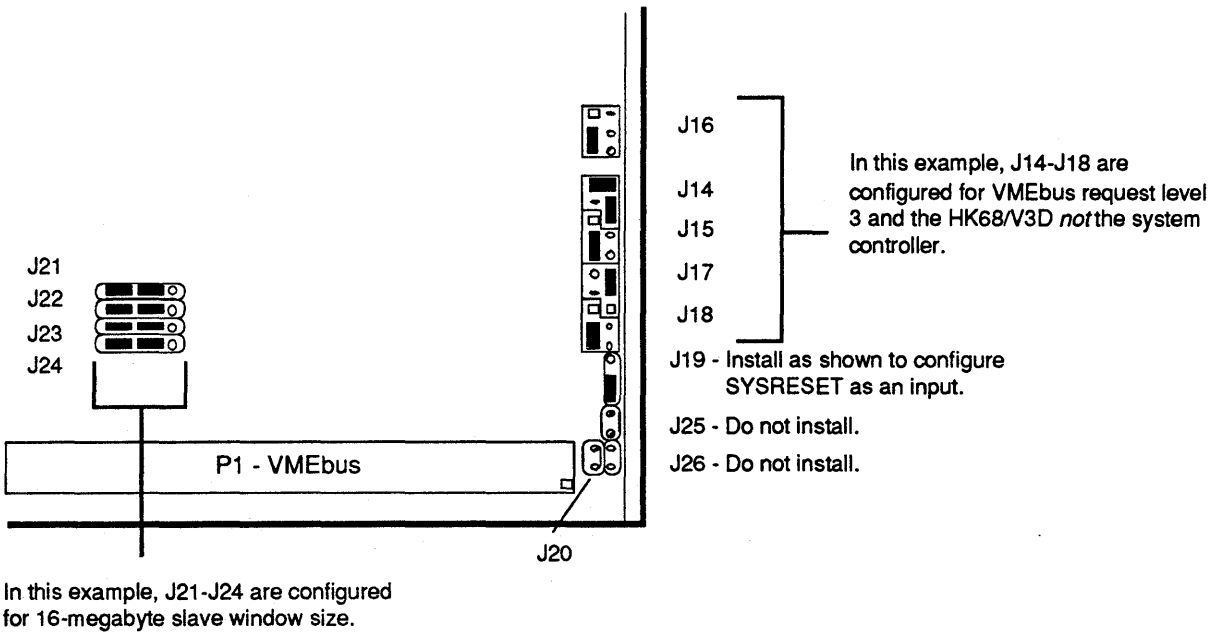


FIGURE 2-4. The HK68/V3D *not* configured as VMEbus system controller via jumpers J14-J18

9

If you change either ROM, be sure the ROM size jumpers **J5–J8** are set to match the size of ROM0 and. jumpers **J9–J12** are set to match the size of ROM1. The possible configurations are shown in Table 2-2.

10

If your HK68/V3D has the optional SCSI interface and the HK68/V3D is at the end of a SCSI cable, install resistor networks RN29. RN30, and RN31, which are socketed SCSI terminators located next to connector P2 (Fig. 1-1). The SCSI specification requires that the bus be terminated at both ends of the cable, so RN29, RN30, and RN31 should be installed *only* if the HK68/V3D is at an end of the SCSI interface cable. See section 11 for details.

2.8 TROUBLESHOOTING AND SERVICE INFORMATION

In case of difficulty, use this checklist:

- ❒ Be sure the system is not overheating.
- ❒ Inspect the power cables and connectors.
- ❒ If you are using the monitor program, run the diagnostics by executing the monitor command **testmem**.
- ❒ Check your power supply for proper DC voltages. If possible, use an oscilloscope to look for excessive power supply ripple or noise. Note that P2 contains power and ground pins for VMEbus. P2 must be used to meet the power specifications.
- ❒ Check the chips to be sure they are firmly in place. Look for chips with bent or broken pins. In particular, check the EPROM.
- ❒ Check your terminal switches and cables. Be sure the P3 connector is secure. If you have made your own cables, pay particular attention to the cable drawing in section 10.
- ❒ Check the jumpers to be sure your board is configured properly. Check the ROM jumpers, especially.
- ❒ The HK68/V3D monitor uses an on-card EEPROM to configure and set the baud rates for its console port. The lack of a prompt might be caused by incorrect terminal settings, an incorrect configuration of the EEPROM, or a malfunctioning EEPROM. Try holding down the **H** character during reset to abort autoboot from the EEPROM. If the prompt

comes up, the EEPROM was most likely configured incorrectly. Type **nvdisplay** to check the monitor configuration. For more information about the way the EEPROM configures the console port baud rates, refer to Appendix A.

❒ After you have checked all of the above items, call our Factory Service Department at 1-800-327-1251 for help. Please have the following information handy:

- The monitor program revision level (labelled on the monitor EPROM)
- The serial number of the operating system.
- The HK68/V3D p.c.b. serial number (inscribed along the card edge).
- Whether your board has been customized for options such as processor speed or configuration for networking and peripherals.

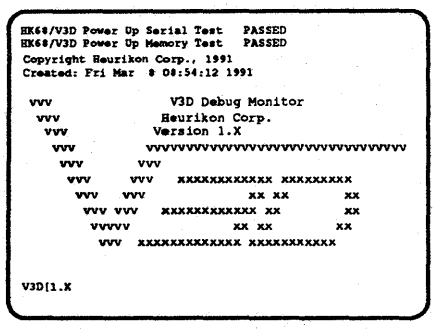
If you plan to return the board to Heurikon for service, contact our Customer Service Department to obtain a **Return Merchandise Authorization** (RMA) number. We will ask you to list which items you are returning and the board serial number, plus your purchase order number and billing information if your HK68/V3D is out of warranty. If you return the board, be sure to enclose it in an antistatic bag, such as the one in which it was originally shipped. Send it prepaid to:

Heurikon Corporation
Factory Service Department
8310 Excelsior Drive
Madison, WI 53717

RMA# _____

Please put the RMA number on the outside of the package so we can handle your problem efficiently. Our service department cannot accept material received without an RMA number.

2.9 MONITOR SUMMARY



An EPROM-based debug-monitor/bootstrap for the HK68/V3D is available as an option. The monitor allows you to access almost all of the hardware registers on the HK68/V3D. You can configure the master and slave bus interface registers, console port, and download port with the monitor.

General functions include the capability to:

- Manually download data or 68030 program code.
- Check the processor, memory, VMEbus and I/O devices.
- Execute a bootstrap (for example, boot an operating system).

The monitor uses the area between $0000,0000_{16}$ and $0001,0000_{16}$ for stack and uninitialized-data space. Any *writes* to that area can cause unpredictable operation of the monitor. The monitor initializes this area (that is, writes to it) to prevent parity errors, but it is left up to the programmer to initialize any other memory areas that are accessed.

Help

Type **help** to view a summary of the monitor. There is an on-line reference for using monitor commands, the command line editor, and the board memory map. Additional help is available for specific commands; type **help** and the command name for details.

Help memmap

Type **help memmap** to view the memory map for the HK68/V3D.

Help functions

Type **help functions** to view a list of functions that the monitor commands call. The functions can be used directly from the command line. It's better to use the monitor commands in most cases, because calling the functions directly from the command line bypasses argument checking.

Command editor

The monitor provides a command line editor that uses typical UNIX® *vi* editing commands. You can edit any command line you type. Press the ESC key from the command line to start the editor. Type **help editor** to access an on-line description of the editor.

Command history

The monitor maintains a command history. Press the ESC key from the command line to access the history. Then press **K** or - to find previous commands. Press **J** or + to find subsequent commands. Up to 50 command lines can be accessed for reuse.

Table 2-3 is a summary of monitor commands. A full description of the monitor and a command reference are in Appendix A.

TABLE 2-4
Monitor command summary

Help commands				
help commands	help editor	help functions	help memmap	
Booting up				
bootbus	bootrom	bootserial		
Manipulating memory				
checksummem	displaymem	findstr	writemem	
clearmem	fillmem	readmem	writestr	
cmpmem	findmem	setmem		
copymem	findnotmem	swapmem		
Manipulating nonvolatile memory (NVRAM)				
nvdisplay	nvinit	nvopen	nvset	nvupdate
Downloading and executing host applications				
call	download	transmode		
Debugging applications				
disassemble	dumpregs	exectrace	settrace	step
Checking arbiter status and displaying Ethernet ID				
prstatus				
Controlling VMEbus slave access				
slavedis				
slaveenable				
Controlling the timer				
starttimer	stoptimer			
Testing local and external RAM				
testmem				
Viewing and setting the date				
date	setdate			
Calculating with hex, decimal, octal, or binary integers				
add	sub	mul	div	rand

3

MPU

3.1 INTRODUCTION

This section details some of the important features of the 68030 MPU chip and, in particular, features that are specific to its implementation on the Heurikon HK68/V3D.

3.2 MPU INTERRUPTS

The MPU can internally set an interrupt priority level in such a way that interrupts of a lower priority will not be honored. Interrupt level seven, however, cannot be masked off.

TABLE 3-1
MPU interrupt levels

Level	Interrupt (bus)	Interrupt (on-card)
7	IRQ7	Parity error, highest priority, non-maskable, autovectored
6	IRQ6	CIO (vectored) (sub-priority: timer 3, port A, timer 2, port B, timer 1)
5	IRQ5	unused
4	IRQ4	SCSI (autovectored)
3	IRQ3	unused
2	IRQ2	SCC (vectored) (sub-priority: ports A and B) (sub-sub-priority: rcv ready, tx ready, status change)
1	IRQ1	Ethernet (autovectored)
0		Idle, no interrupt

When an interrupt is recognized by the MPU, the current instruction is completed and an interrupt acknowledge sequence is initiated, whose purpose is to acquire an interrupt vector from the interrupting device. The vector number is used to select one of 256

exception vectors located in reserved memory locations (see section 3.3 for a listing.) The exception vector specifies the address of the interrupt service routine.

If there are two interrupts pending at the same level, the on-card device is serviced before the bus interrupt. The VMEbus interrupts are masked on and off via the CIO. Refer to sections 10 and 9.4.

The SCC and CIO devices on the HK68/V3D are capable of generating more than one vector, depending on the particular condition which caused the interrupt. This significantly reduces the time required to service the interrupt because the program does not have to rigorously test for the interrupt cause. Section 7.5 has more information on the HK68/V3D interrupt logic. The VMEbus interrupts are vectored; the vector is automatically read from the interrupting device.

3.3 MPU EXCEPTION VECTORS

Exception vectors are memory locations from which the MPU fetches the address of a routine to handle an exception (interrupt). All exception vectors are two words long (four bytes), except for the reset vector which is four words. The listing below shows the vector space as it appears to the Heurikon HK68/V3D MPU. It varies slightly from the 68030 MPU manual listing due to particular implementations on the HK68/V3D board. Refer to the MPU documentation for more details. The vector table normally occupies the first 1024 bytes of RAM, but may be moved to other locations under software control. Unused vector positions may be used for other purposes (e.g., code or data) or point to an error routine.

TABLE 3-2
MPU exception vectors

Vector	Address Offset	Assignment
0	000	Reset: Initial SSP (Supervisor Stack Pointer)
1	004	Reset: Initial PC (Supervisor Program Counter)
2	008	Bus Error (Watchdog Timer, MMU Fault)
3	00C	Address Error
4	010	Illegal Instruction
5	014	Divide by Zero
6	018	CHK Instruction (register bounds)
7	01C	TRAPV Instruction (overflow)
8	020	Privilege Violation (STOP, RESET, RTE, etc)
9	024	Trace (Program development tool)
10	028	Instruction Group 1010 Emulator
11	02C	FPP Coprocessor not present
12	030	(reserved)
13	034	FPP Coprocessor Protocol Violation
14	038	Format Error
15	03C	Uninitialized Interrupt
16-23	040-05F	(reserved-8)
24	060	Spurious Interrupt, not used
25	064	Level 1 autovector, VSB
26	068	Level 2 autovector, not used
27	06C	Level 3 autovector, not used
28	070	Level 4 autovector, SCSI Interrupt
29	074	Level 5 autovector, not used
30	078	Level 6 autovector, not used
31	07C	Level 7 autovector, parity error, ACFAIL
32-47	080-0BF	TRAP Instruction Vectors (16)
48-54	0C0-0DB	FPP Exceptions (8)
55-63	0DC-0FF	(reserved-8)
64-255	100-3FF	User Interrupt Vectors (192)

Autovectoring is used for the parity error, SCSI and VSB interrupts. Interrupts from all other devices can be programmed to provide a vector number (which would likely point into the "User Interrupt Vector" area, above). VMEbus interrupts (IRQ1 - IRQ7) are vectored; the vector is supplied by the interrupting device over the VMEbus.

The following table gives suggested interrupt vectors for each of the possible (on-card) device interrupts which could occur. Note that the listing is in order of interrupt priority, highest priority first.

Note: The ACFAIL line is connected to the VMEbus and can cause a false level of interrupts if there is no power module monitor. For this reason jumper J20 has been provided to allow ACFAIL to be monitored only if the shunt is installed (see the jumper diagram in section 12).

TABLE 3-3
Suggested interrupt vectors

Level	Vector	Device	Condition
7	31		Parity err./ACFAIL autovectored interrupt
6	96	CIO	Timer 3
	79	CIO	External Interrupt (P6-11)
	77		EEPROM 1 Ready
	75		EEPROM 0 Ready
	73		Mailbox Interrupt
	69		VME Interrupt in Progress
	67, 65		
	98	CIO	Timer 2
	76, 74		
	72, 70		
	68, 66		
	64		
	100	CIO	Timer 1
	102	CIO	Timer, error
4	28	SCSI	SCSI Interface (autovectored)
2	92	SCC	Port A, Receive character available
	94		Port A, Special receive condition
	88		Port A, Transmit buffer empty
	90		Port A, External/Status change

	84		Port B, Receive character available
	86		Port B, Special receive condition
	80		Port B, Transmit buffer empty
	82		Port B, External/Status change
1	25	Ethernet	Ethernet Interface (autovectored)

The suggested interrupt vectors for the CIO and SCC devices take into account that the lower bit and upper four bits of the vectors are shared, e.g., all CIO Port A vectors have five bits which are the same for all interrupt causes.

Each vectored on-card device has interrupt enable and control bits which allow the actual interrupt priority levels to be modified under program control by temporarily disabling certain devices.

Of course, fewer vectors may be used if the devices are programmed not to use modified vectors or if interrupts from some devices are not enabled.

If you want to use the suggested vector numbers in the above table, the proper values to load into the device vector registers are:

TABLE 3-4
Device interrupt vector values (suggested)

Device	Hexadecimal Value	Decimal Value
SCC 1 (Ports A & B):	50_{16}	80
CIO, Port A:	41_{16}	65
CIO, Port B:	40_{16}	64
CIO, C/T vector:	60_{16}	96

Making your way through the Zilog CIO and SCC manuals in search of details on the interrupt logic is quite an experience. We suggest you start with these recommended readings from the CIO and SCC technical manuals:

Device	Item
CIO Z8536	Technical Manual
	Vector register: section 2.10.1
	Bit priorities: section 3.3.2
SCC Z8530	Technical Manual
	Port priorities: section 3.2.2, table 3-5
	Vector register: section 5.1.3
	Vectors: section 5.1.10, table 4-3

3.4 STATUS LEDS

There are three status LEDs which continuously show the state of the board as follows:

TABLE 3-5
Status LEDs

LED	Name	Meaning
F	Fail	The SYSFAIL line is being driven active by this board.
M	Master	The HK68/V3D is the master on the VMEbus. It owns the VMEbus.
B	Slave (bus grant acknowledge, BGACK*)	The HK68/V3D is not the VMEbus master. It has given up the local bus, which might be either VMEbus or Ethernet.

3.5 MONITORING MPU STATUS FROM THE FRONT PANEL INTERFACE

Four status outputs allow remote monitoring of the HK68/V3D processor. Connections are made through a 14-pin connector, P5.

TABLE 3-6a
Front panel interface, P5, output signals

P5 pin	Name	Meaning
2	Supr	The MPU is in the supervisor state.
4	User	The MPU is in the user state.
6		not connected
8	Halt	The MPU has halted. (Double bus fault, odd stack address or the system reset line is active.)
10	Bus grant acknowledge (BGACK*)	The HK68/V3D is being accessed as a slave on the VMEbus.
1,3,5,7,9	Vcc	Vcc (+5) volts

The output signals are low when true. Each is suitable for connection to a LED cathode. An external resistor must be provided for each output to limit current to 15 milliamps.

Two input signals are also provided on P5 for interrupt and reset.

TABLE 3-6b
Front panel interface, P5, input signals

P5 pin	Name	Function
11	INTR*	Connected to CIO bit A7, and pull-up Refer to section 9.2)
12	Gnd	
13	RESET*	When low, causes a local reset
14	Gnd	

A recommended mating connector for P5 is Molex P/N 15-29-8148.

3.6 MPU CACHE CONTROL

The 68030 caches may be controlled as follows:

TABLE 3-7
MPU cache control

Address	Function (write-only)
$02B0,0002_{16}$	MPU Cache Control
	D0 = 0, cache disabled (default)
	D0 = 1, caches enabled

The cache control register in the MPU itself must also be set properly to enable the MPU caches.

3.7 COPROCESSORS

The HK68/V3D supports a floating point coprocessor, which is described in section 4.

4

Optional Floating Point Coprocessor

4.1 FEATURE SUMMARY

The HK68/V3D allows the use of an optional MC68882 floating point processor that runs as a coprocessor with the MPU.

The MC68881 frequency may either run at a clock speed of 20 MHz (via the use of a jumper), or it may run at the same speed as the MPU clock

The MC68882 has the following features:

- Allows fully concurrent instruction execution with the main processor.
- Eight general-purpose floating-point data registers, each supporting a full 80-bit extended-precision real data format (a 64-bit mantissa plus a sign bit, and a 15-bit biased exponent).
- A 67-bit ALU to allow very fast calculations, with intermediate precision greater than the extended-precision format.
- A 67-bit barrel shifter for high-speed shifting operations (for normalizing, etc.)
- 46 instruction types, including 35 arithmetic operations.
- Fully conforms to the IEEE P754 standard, including all requirements and suggestions. Also supports functions not defined by the IEEE standard, including a full set of trigonometric and logarithmic functions.
- Supports seven data types: byte, word, and long integers; single, double, and extended-precision real numbers; and packed binary coded decimal string real numbers.
- Efficient mechanisms for procedure calls, context switches, and interrupt handling.

FPP programming details are available in the 68882 technical manual.

4.2 BYPASSING THE FLOATING POINT COPROCESSOR

The HK68/V3D will operate without the floating point chip. Simply unplug the MC68882 if it is not required. No wires or jumpers are needed.

If the Watchdog Timer is enabled, the software can determine if the floating point coprocessor is installed. An attempt to access a nonexistent floating point coprocessor results in a watchdog timeout and a bus error, forcing a line 1111 MPU exception, vector number 11.

5

System Error Handling

Many events could cause an error. The responses to these events are carefully controlled. The following error conditions might arise during MPU cycles:

Condition	Meaning
RAM Parity	Incorrect parity was detected during a read cycle from on-card RAM memory. This may be due to a true parity error (RAM data changed,) or because the memory location was not initialized prior to the read and it contained garbage. Parity errors generate a level 7 autovector interrupt. A pointer to the parity error handling routine should be loaded at Vector Base Register offset $00007C_{16}$. Parity checking cannot be disabled.
Watchdog Timeout	During an on-card access or VMEbus slave access, no acknowledge was received within a fixed time interval defined by a hardware timer (about 100 microseconds). This is usually the result of no bus device being assigned to the specified address. A timeout could also occur if an access from the bus is not terminated by the bus master. Accesses to the bus (VMEbus only) use the system watchdog timer and can hang indefinitely if the system watchdog is not enabled (see section 7.10). For an on-card bus cycle, the memory cycle is terminated, the BERR *(Bus Error)* exception is taken by the MPU and execution resumes at the location specified by the exception vector. If an access *from* the bus was in progress, no BERR exception occurs.

Double Bus Fault Another bus error occurred during the processing of a previous bus error, address error or reset exception. This error is the result of a major software bug or a hardware malfunction. A typical software bug which could cause this error would be an improperly initialized stack pointer, which points to an invalid address.

A double bus fault forces the MPU to enter the *HALT* state. Processing stops. The HALT status LED lights. The only way out of this condition is to issue a hardware reset.

Divide by Zero The value of the divisor for a divide instruction is zero. The instruction is aborted and *vector 5* is used to transfer to an error routine.

Privileged Violation A program executing in the user state attempted to execute a privileged instruction. The instruction is not executed. Exception *vector 8* is used to transfer control.

Address Error An odd address has been specified for an instruction. The bus cycle is aborted and *vector 3* is used to transfer control.

Illegal Instruction The bit pattern for the fetched instruction is not legal or is unimplemented. The instruction is not executed. Exception *vector 4, 10 or 11* is used to transfer control.

Format Error The format of the stack frame is not correct for an RTE instruction. The instruction is aborted and exception *vector 14* is used to transfer control.

Line 1111 Emulator The FPP or PMMU coprocessor is not present and a coprocessor instruction was fetched. The instruction is not executed. Exception *vector 11* will be taken.

FPP Exceptions The FPP coprocessor has detected a data processing error, such as an overflow or a divide by zero. The FPP causes the MPU to take one of seven exceptions in the range from *48* to *54*.

6

On-card Memory Configuration

6.1 INTRODUCTION

The Heurikon HK68/V3D microcomputer can accommodate a variety of RAM and ROM configurations. There are two ROM sockets for PROM, page-addressable ROM or EEPROM, 36 ZIP RAM positions, and a nonvolatile RAM. Off-card memory may be accessed via the VMEbus.

6.2 ROM

Each ROM occupies a fixed 4-megabyte physical address space. At power-on, the MPU fetches the reset vector from the first eight locations of ROM0. The reset vector specifies the initial program counter and status register values. ROM access time must be 250 nanoseconds or less.

TABLE 6-1
ROM address summary

Base Address	ROM	Component Number
$0000,0000_{16}$	0	U70
$0040,0000_{16}$	1	U80

Four jumpers for each ROM must be set according to the ROM type being used (Fig. 6-1). Jumpers J5, J6, J7, and J8 control ROM0 (U70); J9, J10, J11, and J12 control ROM1 (U80). It is possible to use two ROMs of different types.

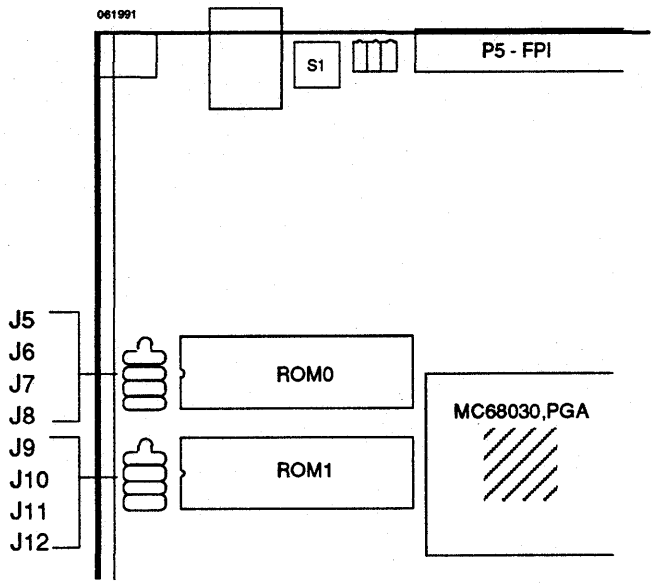


FIGURE 6-1. ROM jumpers

TABLE 6-2
ROM capacity and jumper positions

PROM Type	ROM Capacity	Total Board Capacity	Jumper Positions (for U70 and U80)			
			J5 or J9 B 0 0 0 0 A C		J6, J7, J8, J10, J11, J12 0 0 0 A B	
			J5/J9	**J6/J10**	**J7/J11**	**J8/J12**
2764	8 kilobytes	16 kilobytes	C	B	x	B
27128	16 kilobytes	32 kilobytes	C	B	x	B
27256	32 kilobytes	64 kilobytes	C	B	x	A
27512	64 kilobytes	128 kilobytes	B	B	x	A
27010	128 kilobytes	256 kilobytes	B	x	B	A
27020	256 kilobytes	512 kilobytes	B	A	B	A
27040	512 kilobytes	1 megabyte	B	A	B	A
27080	1 megabyte	2 megabytes	B	A	A	A
27513 paged	64 kilobytes	128 kilobytes	C	B	x	B
2864 R/W EEPROM	8 kilobytes	16 kilobytes	x	B	x	B
2817 R/W EEPROM	2 kilobytes	4 kilobytes	A	B	x	B

Each ROM contains consecutive (both even and odd) addresses. When programming PROMs, do not split even and odd bytes between the two chips.

Both ROM sockets are 32 pins. If you use a 28-pin device, justify it so socket pins 1, 2, 31 and 32 are empty. Twenty-four-pin devices are not supported. The ROM access time must be at most 250 nanoseconds.

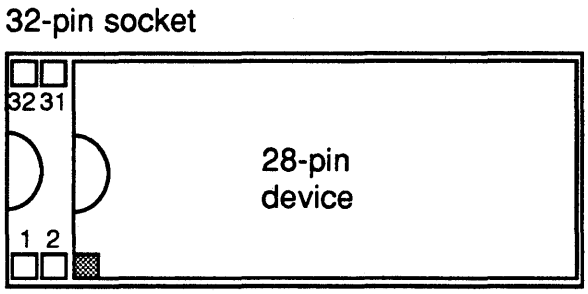


FIGURE 6-2. ROM position for 28-pin ROMs

The two ROM positions are not contiguous (although a mirror of the lower ROM will be contiguous with the upper ROM). The best way to create a contiguous image is to copy the contents of both ROMs to contiguous RAM areas.

Electrically erasable or paged PROMs may be used. An EEPROM allows specific addresses to be changed by writing to the ROM. For writes to the EEPROM, a delay must be provided *by the software* between write operations. For the 2864, this delay is 10 milliseconds. The EEPROM Busy/Ready signals are available at the CIO to facilitate this timing; see section 9.1.

Paged ROMs allow future growth of ROM capacity without adding address pins. A single device can contain multiple 16-kilobyte pages. A specific page is selected by *writing* the page value to the ROM. For example, to select page three of a 27513, write 03_{16} to address $0000{,}0000_{16}$.

6.3 ON-CARD RAM

The HK68/V3D uses 36 ZIP RAM packages, each four bits wide. There is one parity bit per byte. Standard memory configurations are 1, 2, 3 and 4 megabytes (4, 8, 12, and 16 megabytes when 4-megabit DRAMs are available). On-card RAM occupies physical addresses starting at $0300,0000_{16}$.

6.4 ON-CARD MEMORY SIZING

The V3D supports memory sizes ranging from 1 to 16 megabytes. Accessing nonexistent RAM can cause parity errors, so it is necessary to initialize and trap on parity interrupts. Use the following procedure and refer to the example code below:

1. Clear the minimum memory size (1 megabyte) starting at $0300,0000_{16}$.

2. Initialize the parity exception vector to point to a function that will set a flag indicating a failure.

3. For each 1-megabyte boundary starting a $0310,0000_{16}$:

   a. Write the long word pattern 1234,5678 at the current boundary as 4-byte accesses (12 at $0310,0000_{16}$; 23 at $0310,0001_{16}$, etc.).

   b. Read the long word pattern from the current boundary as a single long word.

   c. If the pattern read equals 1234,5678 and the parity error exception flag is not set, then continue. If not, then return the current boundary as the top of RAM.

Repeat these steps for $0320,0000_{16}$; $0330,0000_{16}$ . . . $0340,0000_{16}$ to determine memory size.

6.5 BUS MEMORY

See section 7 for details concerning the bus interface.

6.6 PHYSICAL MEMORY MAP

See section 14.2 for an I/O device address summary.

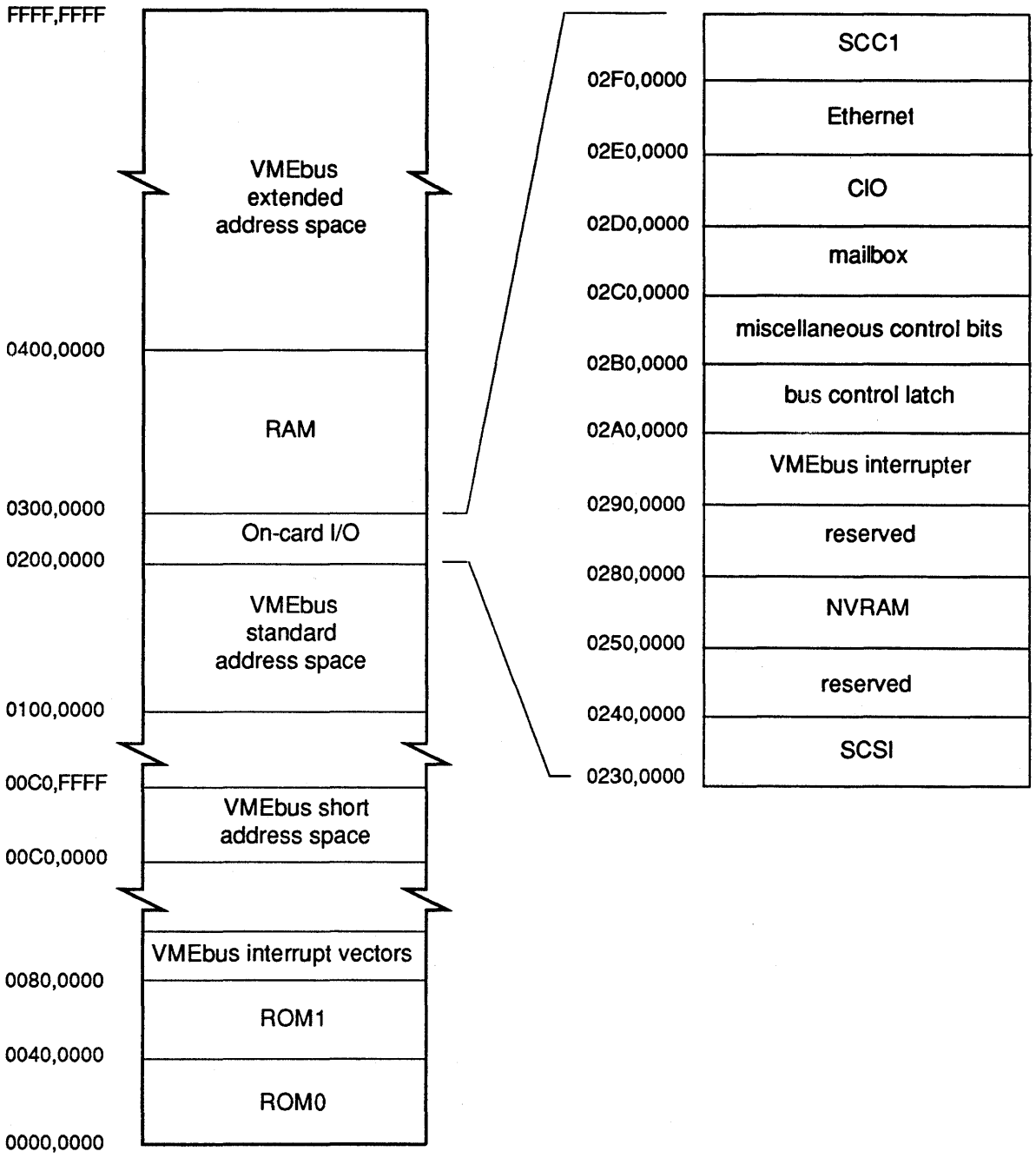


FIGURE 6-3. Physical memory map

6.7 MEMORY TIMING

The HK68/V3D memory logic has been carefully tuned to give optimum memory cycle times under a variety of conditions.

The base cycle time for a MC68030 is two clock cycles for a RAM read or write and one clock cycle for subsequent burst cycles. Although the MC68030 cannot perform memory accesses any faster than this, it can be made to perform accesses slower than this. The following chart shows total access times required to attain these base cycle times from a RAM interface. It should be noted that this is the time from address valid to data input setup of the MC68030, including clock skew and various other factors.

TABLE 6-3
Access time required for no wait states

CPU Speed (MHz)	Read Cycle (ns)	Write Cycle (ns)	Burst Cycle (ns)
32	15	32	32

As Table 6-3 shows, current DRAM technology with access times in the 60-nanosecond to 150-nanosecond range cannot support the base transfer rates of the MC68030, and additional cycles must be inserted in each cycle to meet DRAM access time requirements. The number of additional clock cycles inserted in each access depends on both the processor speed and on the RAM speed. Table 6-4 shows the number of extra cycles or "wait states" inserted in RAM read or write cycles and burst cycles.

TABLE 6-4
Inserting wait states into RAM cycles

100-nanosecond DRAMS			
CPU Speed	**Read Cycle**	**Write Cycle**	**Burst Cycle**
32 MHz	3	3	3-1-1-1
80-nanosecond DRAMS			
CPU Speed	**Read Cycle**	**Write Cycle**	**Burst Cycle**
32 MHz	3	2	3-1-1-1
60-nanosecond DRAMS			
CPU Speed	**Read Cycle**	**Write Cycle**	**Burst Cycle**
32 MHz	2	2	2-1-1-1

60-nanosecond, 80-nanosecond and 100-nanosecond times are estimates based on existing 256K × 1 DRAMs.

While the above information is important in comparing the relative performance of DRAM designs, the performance of

individual DRAM designs has much less impact on overall system performance than one might expect. The reason for this is that the internal cache(s) built into the MC68030 chip is provided to help decouple the processor from slower speed memories such as DRAMs. Therefore, the better the job the MC68030 cache is doing, the less difference in system performance DRAM speed will make.

6.8 NONVOLATILE RAM

A unique feature of the HK68/V3D is its non-volatile RAM (NVRAM), which allows precious data or system configuration information to be stored and recovered across power cycles. The RAM is configured as 256, four-bit words (low half of a byte). When the MPU reads a byte of data from the NV-AM, the upper four bits of the value it receives are indeterminate. The NVRAM is accessible as shown below.

TABLE 6-5
Nonvolatile RAM addresses

Address	Mode	Function
$0250,00XX_{16}$	R/W	Read/write RAM contents (4 bits).
$0270,0000_{16}$	Read	Recall RAM contents from nonvolatile memory.
$0260,0000_{16}$	Write	Store RAM contents in nonvolatile memory. The 68030 **tas** (test and set) instruction must be used for this operation.

Physically, the NVRAM (an Xicor X2212 or equivalent) consists of a static RAM overlaid bit-for-bit with a nonvolatile EEPROM. The store operation takes 10 milliseconds to complete. Recall time is approximately one microsecond. Allowances for those delays should be made in *software,* since the memory hardware does not stop the MPU during the store or recall cycles. The chip is rated for 10,000 store cycles, minimum. During a store operation, only those bits which have been changed are "cycled." The use of a **tas** instruction helps prevent an unintentional store operation by an errant program or a power failure glitch.

At power-up, the shadow RAM contents are indeterminate. Do a recall operation before accessing the NVRAM for the first time. Recall cycles do not affect the device lifetime.

The HK68/V3D monitor and certain system programs use the NVRAM. The exact amount reserved for Heurikon usage depends on the system. A major portion of the RAM, however, is available for customer use. Heurikon usage is summarized below (details are available separately):

TABLE 6-6
NV-RAM contents (partial)

Function
Magic number
Checksum
Accumulated number of writes
Board type, serial number and revision level
Hardware configuration information
Software configuration information
System configuration information

7

VMEbus Control

7.1 INTRODUCTION

The control logic for the VMEbus allows numerous bus masters to share the resources on the bus.

The VMEbus interface uses 32 address lines for a total of 4 gigabytes of VMEbus address space, and 32 data lines to support 8-, 16-, 24- or 32-bit data transfers. The "short address" mode, which uses only 16 address lines, is also supported.

There is an interrupter module as well as an interrupt handler. Both are capable of utilizing any or all of the seven VMEbus interrupt lines.

7.2 BUS CONTROL SIGNALS

The following signals on connector P1 and P2 are used for the VMEbus interface. Pin assignments are in section 7.12.

A01-A15 ADDRESS bus (bits 1-15). Three-state address lines that are used to broadcast a short address.

A16-A23 ADDRESS bus (bits 16-23). Three-state address lines that are used in conjunction with A01-A15 to broadcast a standard address.

A24-A31 ADDRESS bus (bits 24-31). Three-state address lines that are used in conjunction with A01-A23 to broadcast an extended address.

ACFAIL* AC FAILURE. An open-collector signal that indicates that the AC input to the power supply is no longer being provided or that the required AC input voltage levels are not being met. This signal is connected to MPU interrupt level 7.

AM0-AM5 ADDRESS MODIFIER (bits 0-5). Three-state lines that are used to broadcast information such as address size and cycle type. These lines are very similar in usage to the function lines on the MPU.

AS* ADDRESS STROBE. A three-state signal that indicates when a valid address has been placed on the address bus.

BBSY* BUS BUSY. An open-collector signal driven low by the current master to indicate that it is using the bus. When the master releases this line, the resultant rising edge causes the arbiter to sample the bus request lines and grant the bus to the highest priority requester. Early release mode is supported.

BCLR* BUS CLEAR. A totem-pole signal generated by an arbiter to indicate when there is a higher priority request for the bus. This signal requests the current master to release the bus. This signal is an input and an output of the HK68/V3D, associated with J26.

BERR* BUS ERROR. An open-collector signal generated by a slave or bus timer. This signal indicates to the master that the data transfer was not completed.

BG0IN*-BG3IN* BUS GRANT (0-3) IN. Totem-pole signals generated by the arbiter and requesters. "Bus grant in" and "bus grant out" signals form bus grant daisy chains. The "bus grant in" signal indicates, to the board receiving it, that it may use the bus if it wants.

BG0OUT*-BG3OUT* BUS GRANT (0-3) OUT. Totem-pole signals generated by requesters. The bus grant out signal indicates to the next board in the daisy-chain that it may use the bus.

BR0*-BR3* BUS REQUEST (0-3). Open-collector signals generated by requesters. A low level on one of these lines indicates that a master needs to use the bus.

D00-D31 DATA BUS. Three-state bidirectional data lines used to transfer data between masters and slaves.

DS0*, DS1* DATA STROBE ZERO, ONE. A three-state signal used in conjunction with LWORD* and A01 to indicate how many data bytes are being transferred (1, 2, 3, or 4). During a write cycle, the falling edge of the first data strobe indicates that valid data are available on the data bus.

DTACK* DATA TRANSFER ACKNOWLEDGE. An open-collector signal generated by a slave. The falling edge of this signal indicates that valid data are available on the data bus during a read cycle, or that data have been accepted from the data bus

during a write cycle. The rising edge indicates when the slave has released the data bus at the end of a read cycle.

IACK* INTERRUPT ACKNOWLEDGE. An open-collector or three-state signal used by an interrupt handler when it acknowledges an interrupt request. It is routed, via a backplane signal trace, to the IACKIN* pin of slot one, where it forms the beginning of the IACKIN*, IACKOUT* daisy-chain.

IACKIN* INTERRUPT ACKNOWLEDGE IN. A totem-pole signal. The IACKIN* signal indicates to the VMEbus board receiving it that it is allowed to respond to the interrupt acknowledge cycle that is in progress if it wants.

IACKOUT* INTERRUPT ACKNOWLEDGE OUT. A totem-pole signal. The IACKIN* and IACKOUT* signals form a daisy-chain. The IACKOUT* signal is sent by a board to indicate to the next board in the daisy-chain that it is allowed to respond to the interrupt acknowledge cycle that is in progress.

IRQ1*-IRQ7* INTERRUPT REQUEST (1-7). Open-collector signals, generated by an interrupter, that carry interrupt requests. When several lines are monitored by a single interrupt handler, the line with the highest number is given the highest priority.

LWORD* LONGWORD. A three-state signal used in conjunction with DS0*, DS1*, and A01 to select which byte location(s) within the 4-byte group are accessed during the data transfer.

RESERVED RESERVED. A signal line reserved for future VMEbus enhancements. This line must not be used.

SERCLK SERIAL CLOCK. A totem-pole signal that is used to synchronize the data transmission on the VMEbus. This signal is not implemented on the HK68/V3D.

SERDAT* SERIAL DATA. An open-collector signal that is used for VMEbus data transmission. Not implemented on the HK68/V3D.

SYSCLK SYSTEM CLOCK. A totem-pole driven signal that provides a constant 16-MHz clock signal that is independent of any other bus timing. This signal is controlled with J25.

SYSFAIL* SYSTEM FAIL. An open-collector signal that indicates a failure has occurred in the system. Also used at power-on to indicate that at least one VMEbus board is still in its power-on initialization phase. This signal may be generated by any board on the VMEbus. The HK68/V3D drives this line low at power-on. It is released by writing a one to address $02B0,000E_{16}$.

SYSRESET* SYSTEM RESET. An open-collector signal that, when low, causes the system to be reset. This signal is controlled by jumper J19.

WRITE* WRITE. A three-state signal generated by the MASTER to indicate whether the data transfer cycle is a read or a write. A high level indicates a read operation; a low level indicates a write operation.

+5V STDBY +5 Vdc STANDBY. This line supplies +5 Vdc to devices requiring battery backup. This signal is not used on the HK68/V3D.

7.3 BUS ARBITRATION AND RELEASE

When the MPU makes a request for VMEbus facilities, the arbitration logic takes over. If necessary, the requesting board enters a wait state until the bus is available (but only for the maximum time allowed by the watchdog timer).

Under normal circumstances, the VMEbus system controller card provides the system bus clock and access timer, and participates in the arbitration logic. A separate system controller card is *not* needed; however. The HK68/V3D includes a bus timer and four-level (prioritized) VMEbus arbiter logic, enabled via jumpers. The following table details the system controller functions provided by the HK68/V3D.

TABLE 7-1
System controller functions

Function	Setting
System Clock (SYSCLK*)	J25 (install)
System Reset (SYSRESET*)	J19:2–3 (output)
Bus Clear (BCLR*)	J26 (install)

When the HK68/V3D is acting as a system controller, it should be in the first slot (VMEbus slot 1).

There are four separate bus request lines on the VMEbus. Each bus request line has an associated bus grant daisy chain.

The following steps *must* be used to configure the HK68/V3D, whether or not the HK68/V3D is the system controller. Failure to follow these instructions could result in incorrect board operation.

1. Decide which level the board will use to request the VMEbus.

2. Set the Bus Request jumper, J16, to the chosen level according to Figure 7-1.

3. Decide if the HK68/V3D will be the system controller on the VMEbus.

4. Install J14, J15, J17, and J18 corresponding to the configuration chosen above. Select the appropriate setting from the eight legal settings shown for those jumpers in Figure 7-2.

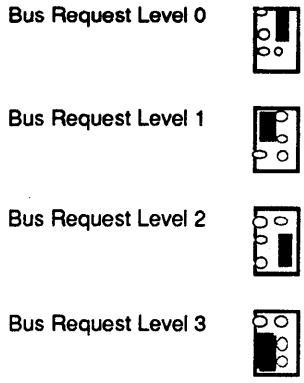


FIGURE 7-1. Bus request jumper settings, J16

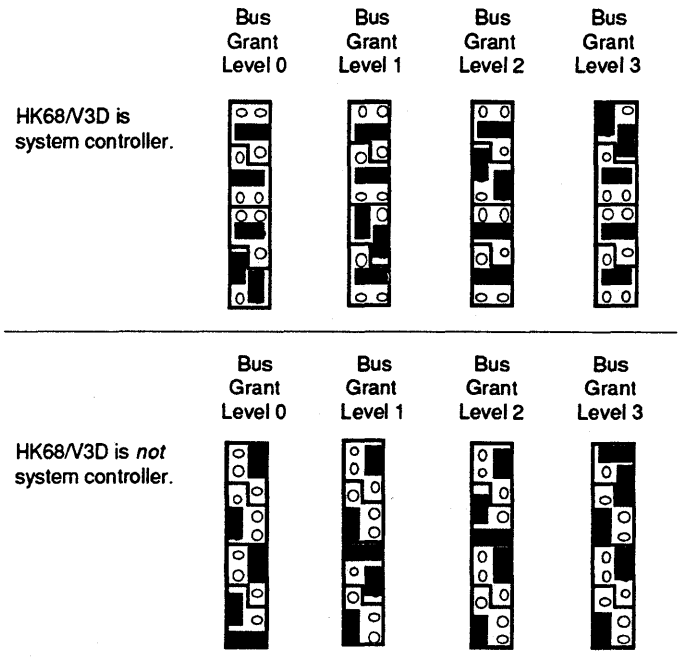


FIGURE 7-2. Bus grant level jumper settings J14, J15, J17, and J18

If the HK68/V3D is the bus master, when the requested bus operation is completed, the bus will be released according to the state of two bus control signals, BC1 and BC0. These signals are under software control.

TABLE 7-2
Bus control bits

BC1	BC0	Bus Release Status
0	0	**Release when done.** Release the bus after every operation.
0	1	**Release on request.** Release the bus if any other board has a request for the bus (or if BCLR is true).
1	1	**Release on priority.** Release the bus only if BCLR is true. This means release only if a higher priority request is pending.
1	0	**No release.** Never release the bus, once acquired. This state can be used to capture the bus.

The bus control bits are set (or reset) by writing to the appropriate bits of the bus control latch, described below.

7.4 ACCESSES *FROM* THE VMEbus (SLAVE MODE)

The slave address logic is enabled or disabled by writing the appropriate value to the slave mode control bit, as follows:

TABLE 7-3
Slave mode control

Address	Function (write-only)
$02B0,000C_{16}$	Slave mode enable
	D0 = 0, slave disable
	D0 = 1, slave enable

When the most significant VMEbus address lines match the slave compare address and the address modifier matches the slave address modifier" code, as set in the bus control Latch, a slave access is recognized. The most significant address lines (A24-A31) are tested only if the selected address modifier is "extended." The base address of the window into on-card RAM is also set by bits in the bus control latch. The size of the window is specified by J21 through J24 as shown in Figure 7-3 and Table 7-4.

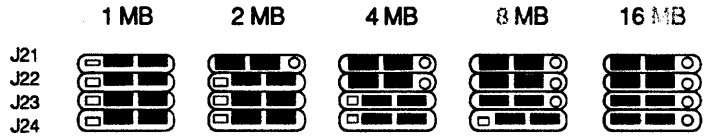


FIGURE 7-3. Slave window size jumper settings J21–J24

TABLE 7-4
Slave window size jumpers

Slave Window Size	J21	J22	J23	J24	Address Compare	Replacement Address
		A	B			
1 megabyte	B	B	B	B	A20-A31	A20-A23
2 megabytes	A	B	B	B	A21-A31	A21-A23
4 megabytes	A	A	B	B	A22-A31	A22-A23
8 megabytes	A	A	A	B	A23-A31	A23 only
16 megabytes	A	A	A	A	A24-A31	none

A 24-bit latch is used to specify various parameters concerning the operation of the VMEbus. This is a write-only register. The default state at power-up is all zeros.

The latch (Fig. 7-4) is composed of three 8-bit shift registers, which are set as follows:

1. Disable the VMEbus slave logic by writing a zero to address $02B0,000C_{16}$.

2. Write a 32-bit long word to the bus control latch at address $02A0,0000_{16}$. This is done by performing eight consecutive writes to the bus control latch. The data are automatically shifted into the shift registers. (See the code fragment in Example 7-1.)

3. Enable the VMEbus slave logic by writing a one to address $02B0,000C_{16}$.

EXAMPLE 7-1. Bus control latch loading routine

```
#define BUS_LATCH  (unsigned long *)0x02A00000
#define SLAVE_ENABLE (unsigned char *)0x02B0000C

WrBusLatch(value)
unsigned long value;
{
 int i;
 *SLAVE_ENABLE = 0; /* disable slave interface */
 for (i=0; i<8; i++ ) {
   *BUS_LATCH = (value >> i); /* shift in D16, D8 and D0 */
 }
 *SLAVE_ENABLE = 1; /* enable slave interface */
}
```

EXAMPLE 7-2. Setting the bus control latch with the HK68/V3D monitor

If you are using the HK68/V3D monitor, use the command **writemem** to set the bus control latch. In this example, a series of **writemem** commands write the value 00380040_{16} to the bus control latch. The effect of the write is to set the latch as follows:

Set the slave address modifier bits to extended space (32-bit)

Set the bus release mode to release-when-done via bus control bits BC0 and BC1

Set the replacement address to 0 (base of RAM)

Set the slave address to 40000000_{16}.

```
writemem -b 02B0000C 0            Slave disable
writemem -l 02A00000 0            Bits 0, 8, 16 are 0.
writemem -l 02A00000 0            Bits 1, 9, 17 are 0.
writemem -l 02A00000 0            Bits 2, 10, 18 are 0.
writemem -l 02A00000 00010000     1 on DB16 setting bit 19.
writemem -l 02A00000 00010000     1 on DB16 setting bit 20.
writemem -l 02A00000 00010000     1 on DB16 setting bit 21.
writemem -l 02A00000 00000001     1 on DB0 setting bit 6.
writemem -l 02A00000 0            Bits 7, 16, 23 are 0.
writemem -b 02B0000C 1            Slave enable
```

TABLE 7-5
Bus control latch (VMEbus slave logic)

Bit	Function
23	(reserved)
22	Indivisible Read Modify Writes
21	Slave Address Modifier 2
20	Slave Address Modifier 1
19	Slave Address Modifier 0
18	VMEbus Slave Release Without Hold
17	Bus Control BC 1
16	Bus Control BC 0
15	Replacement Address 23
14	Replacement Address 22
13	Replacement Address 21
12	Replacement Address 20
11	Slave Compare Address 23
10	Slave Compare Address 22
9	Slave Compare Address 21
8	Slave Compare Address 20
7	Slave Compare Address 31
6	Slave Compare Address 30
5	Slave Compare Address 29
4	Slave Compare Address 28
3	Slave Compare Address 27
2	Slave Compare Address 26
1	Slave Compare Address 25
0	Slave Compare Address 24

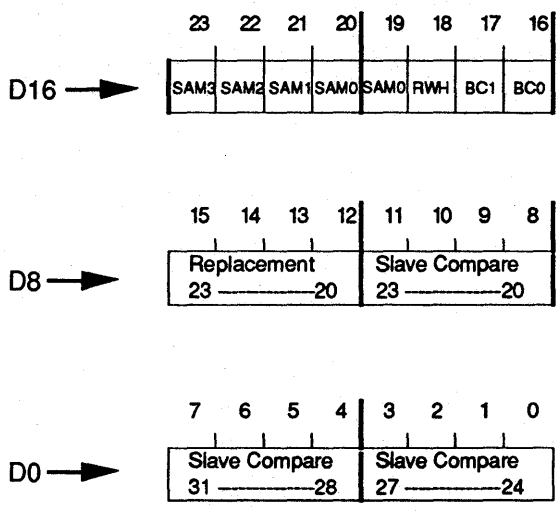


FIGURE 7-4. Bus control latch

The slave address modifier (SAM) is selected by three SAM bits in the bus control latch according to the following chart:

TABLE 7-6
Slave address modifiers

SAM2	SAM1	SAM0	Slave Address Space
0	0	0	No slave access allowed (disable)
0	0	1	Standard data
0	1	0	No slave access allowed
0	1	1	Standard (all)
1	0	0	Extended supervisor data
1	0	1	Extended data
1	1	0	No slave access allowed
1	1	1	Extended (all)

Once a valid bus request has been detected, an on-card bus request is generated to the MPU. When the current MPU cycle is completed, the MPU will release the on-card bus. The VMEbus address and data are then gated on.

Bit 22 of the bus control latch, when set, allows indivisible read-modify-writes to the VMEbus. Because the MC68030 asserts RMC during MMU translation table walks, it is necessary to break up the cycle to allow VMEbus memory cards to see the cycle as two separate addresses.

The bus address lines are utilized as shown in Figure 7-4:

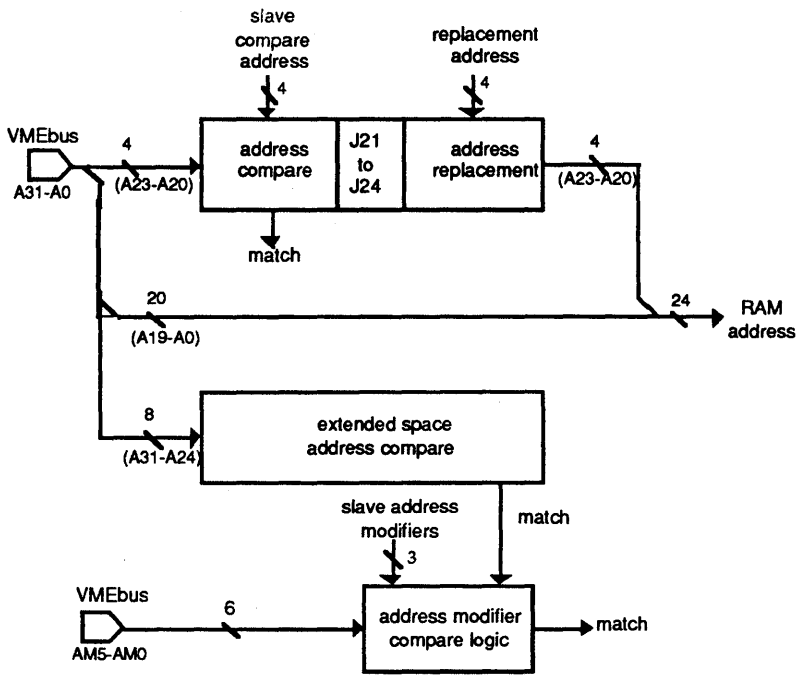


FIGURE 7-5. Memory accesses from the VMEbus

For example, if the bus control latch is set to 383050_{16} and J21–J24 are set to select a one megabyte window, then all extended space accesses from $5000,0000_{16}$ through $500F,FFFF_{16}$ are mapped to the fourth megabyte of on-card RAM at location $0330,0000_{16}$.

After a slave access, control of the on-card bus will not be returned to the MPU for approximately 500 nanoseconds. However, if the release-without-hold bit in the bus control latch (see above) is set, the bus will be returned immediately following the slave access. This mode can be used to maximize bus response time to the MPU and DMAC at the expense of having more overhead on slave accesses. If you expect rapid requests from the VMEbus, you may not want to use this mode.

The bus timer will automatically terminate any slave access which lasts longer than 100 microseconds.

7.5 VMEbus INTERRUPTS

The seven VMEbus interrupts are monitored and controlled by the MPU and CIO. A vectored interrupt to the MPU can be generated when a desired bus interrupt signal is on.

There are two functions described below. The *interrupter* generates bus interrupts; the *interrupt handler* receives interrupts from the bus.

7.5.1 Interrupter Module Operation

To *generate* a VMEbus interrupt, follow these steps:

1. Decide which of the seven VMEbus interrupt lines you wish to activate. IRQ7* has the highest priority.

2. Disable that level via the CIO so that the INTERRUPT HANDLER does not respond to the interrupt line you are about to use. If you fail to do this, you could interrupt yourself.

3. Write an eight bit value to the appropriate VMEbus Status/ID latch, as described below. This value is usually treated as a simple interrupt vector, but it could represent other information as well. This value is provided to the board that acknowledges the interrupt, which is done by executing an INTERRUPT ACKNOWLEDGE cycle on the VMEbus with *your* priority level encoded on address lines 1 to 3 (see the Interrupt Handler description, below.)

   The very act of writing to the Status/ID latch activates the INTERRUPTER circuitry, and the interrupt is generated.

TABLE 7-7
VMEbus interrupter addresses

Address	Vector Size	Function (write-only)
$0290,0004_{16}$	8	Interrupt level 1
$0290,0008_{16}$	8	Interrupt level 2
$0290,000C_{16}$	8	Interrupt level 3
$0290,0010_{16}$	8	Interrupt level 4
$0290,0014_{16}$	8	Interrupt level 5
$0290,0018_{16}$	8	Interrupt level 6
$0290,001C_{16}$	8	Interrupt level 7

Only one (outgoing) interrupt may be pending at a time.

The state of the on-card interrupt logic can be tested by the CIO. The Interrupt Active bit is true whenever an interrupt in still pending from this board.

7.5.2 Interrupt Handler Operation

Each bus interrupt generates an interrupt to the MPU at a specific MPU interrupt priority level, as detailed in section 3.2. When an interrupt is recognized, the MPU will execute an interrupt acknowledge cycle on the VMEbus to read the vector from the interrupting board. This vector is used as an index into the MPU vector table.

When an interrupt is generated on the VMEbus, the interrupt vector of the interrupting board may be (manually) determined by reading from the appropriate address, as shown below. The value returned is that value written by the interrupting board to its VMEbus Status/ID latch. Since the MPU automatically does interrupt acknowledge cycles on the bus, the main use for these ports is to clear a pending interrupt on the HK68/V3D (or another VMEbus interrupt source).

The HK68/V3D can generate and read only 8-bit interrupt vectors.

TABLE 7-8
Interrupt acknowledge port summary

8-bit Vector Priority Level	Address (read-only)
IRQ1	$0080,0003_{16}$
IRQ2	$0080,0005_{16}$
IRQ3	$0080,0007_{16}$
IRQ4	$0080,0009_{16}$
IRQ5	$0080,000B_{16}$
IRQ6	$0080,000D_{16}$
IRQ7	$0080,000F_{16}$

Accessing one of the above addresses also sends an interrupt acknowledge signal to the interrupting board. Acknowledging a nonexistent interrupt will result in a bus error.

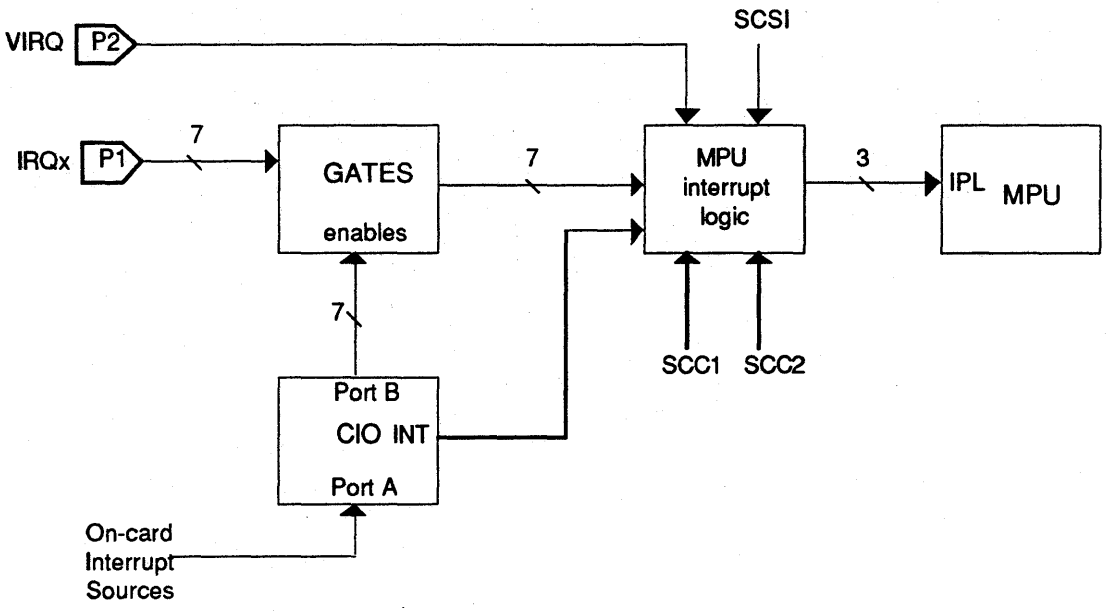


FIGURE 7-6. Interrupt signal routing

7.6 SYSFAIL CONTROL

The SYSFAIL line is driven low by the HK68/V3D after power-on. The SYSFAIL line will remain low on the VMEbus until all boards release this line after completing their initialization and self test sequences. The SYSFAIL line also signifies a system failure. The current state of this signal may be read via the CIO (see section 9.4).

On the HK68/V3D, SYSFAIL must be released under software control. SYSFAIL must be released by writing a one to CIO port C, bit D1 (see section 9.2).

7.7 BUS ADDRESSING (MASTER MODE)

The HK68/V3D supports three address modes, "short", "standard," and "extended." Short addresses use the lower 16 logical address lines to specify the target address. Standard addresses use 24 address lines, and extended addresses use all 32 address lines. The following table details the relationship between the on-card physical address and the corresponding VMEbus region.

TABLE 7-9
VMEbus regions

On-card addresses	VMEbus Region
$00C0,0000_{16}$ through $00C0,FFFF_{16}$	VMEbus Short Address (0000_{16} through $FFFF_{16}$)
$0100,0000_{16}$ through $01FF,FFFF_{16}$	VMEbus Standard ($00,0000_{16}$ through $FF,FFFF_{16}$)
$0400,0000_{16}$ through $FFFF,FFFF_{16}$	VMEbus Extended ($0400,0000_{16}$ and up)

7.8 MAILBOX INTERFACE

Certain on-card functions can be controlled via special addresses in the VMEbus *Supervisor Short Address Space,* that is, when the address modifier lines (AM5* to AM0*) are $2D_{16}$. The HK68/V3D will respond (as a slave) to a short address which matches the Mailbox select lines, as described below. The mailbox logic must be enabled by setting the control bit at address $02B0,0004_{16}$.

TABLE 7-10
Mailbox control

Address	Function (write-only)
$02B0,0004_{16}$	Mailbox control
	D0 = 0, disable (default)
	D0 = 1, enable

TABLE 7-11
Mailbox functions

Address	Function (Slave Mode)
Mbase + 0	CIO input D4 (see section 9.2) (mailbox interrupt)
Mbase + 2	HK68/V3D reset
Mbase + 4	On-card bus lock on
Mbase + 5	On-card bus lock off
Mbase + 6	MPU halt on
Mbase + 7	MPU halt off

The Mbase value is specified by 13 mailbox base bits in the mailbox address latch at address $02C0,0000_{16}$ (16-bits, write-only). Address lines A15 through A3 must match the corresponding data bits in the mailbox address latch. The lower three bits of the latch are not used.

The lock function, when on, prevents the use of the on-card bus by the MPU after the *next* access from the bus. The lock function

must be cleared before the MPU is allowed to resume operation. This feature can be used to reduce arbitration time during a block data transfer from the VMEbus. With the on-card bus locked, slave accesses will be acknowledged in 330 to 500 nanoseconds.

The SYSFAIL signal must be off for the mailbox halt function to operate. (See section 7.6.)

7.9 WATCHDOG AND BUS TIMER

The HK68/V3D has two timers which monitor board activity. One is used to monitor on-card activity; the other is for the VMEbus.

7.9.1 On-card Watchdog Timer

If the on-card watchdog timer is enabled and if the on-card physical address strobe stays on longer than 1.67 milliseconds, the timer will expire. This will cause the current memory cycle to be terminated. The watchdog timer is *disabled* by writing a one to address $02B0,0030_{16}$. The timer is *enabled* by writing a zero to address $02B0,0030_{16}$; this is the power-on default state.

See section 5.1 for more details on the watchdog timer.

7.9.2 VMEbus Timer

The second timer is associated only with activity on the VMEbus. The timer will expire during a long bus access (greater than 100 microseconds) by *any* bus master and generate a VMEbus error (BERR). This is normally a VMEbus system controller function.

The VMEbus timer is enabled by writing a 1 to address $02B0,0010_{16}$. The default state is disabled.

7.10 BUS CONTROL JUMPERS

TABLE 7-12
Bus control jumpers

Jumper	Function	Position
J14	Bus Arbitration Level	See section 7.3
J15	Bus Arbitration Level	See section 7.3
J16	Bus Request Level	See section 7.3
J17	Bus Arbitration Level	See section 7.3
J18	Bus Arbitration Level	See section 7.3
J19	SYSRESET*	See section 7.3
J21	VMEbus Slave Window Size	See section 7.4
J22	VMEbus Slave Window Size	See section 7.4
J23	VMEbus Slave Window Size	See section 7.4
J24	VMEbus Slave Window Size	See section 7.4
J25	SYSCLK*	See section 7.3
J26	BCLR*	See section 7.3

7.11 VMEBUS INTERFACE

The VMEbus consists of P1 address, data, and control signals. P2 is used for the extended VMEbus address and data lines as well as the optional SCSI interface, which is described in section 11 (Fig. 7-7).

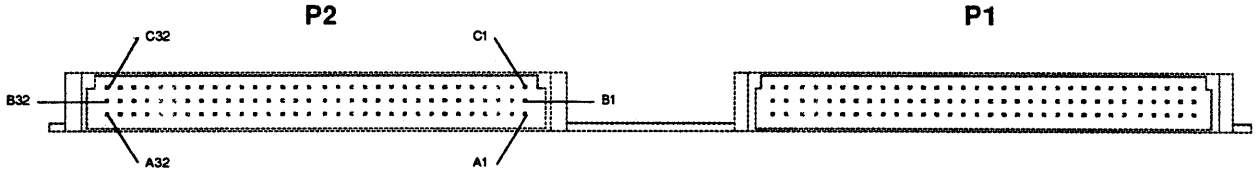


FIGURE 7-7. VMEbus connectors, P1 and P2

7.12 VMEBUS PIN ASSIGNMENTS, P1

TABLE 7-13
VMEbus pin assignments, P1

P1 Pin Number	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	Gnd	BG2OUT*	Gnd
10	SYSCLK	BG3IN*	SYSFAIL*
11	Gnd	BG3OUT	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	Gnd	BR3*	A23
16	DTACK*	AM0	A22
17	Gnd	AM1	A21
18	AS*	AM2	A20
19	Gnd	AM3	A19
20	IACK*	Gnd	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT*	A16
23	AM4	Gnd	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

7.13 P2 VMEbus PIN ASSIGNMENTS

P2 is used for both the VMEbus and the optional SCSI interface. The center row (B) of pins are the upper address and data lines of the VMEbus. The outer two rows (A and C) make up the SCSI interface. The use of P2 is *required* in order to meet VMEbus power specifications.

TABLE 7-14
VMEbus pin assignments, P2

P2 Pin Number	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	AD00	+5	AD01
2	AD02	Gnd	AD03
3	AD04	(reserved)	AD05
4	AD06	A24	AD07
5	AD08	A25	AD09
6	AD10	A26	AD11
7	AD12	A27	AD13
8	AD14	A28	AD15
9	AD16	A29	AD17
10	AD18	A30	AD19
11	AD20	A31	AD21
12	AD22	Gnd	AD23
13	AD24	+5	AD25
14	AD26	D16	AD27
15	AD28	D17	AD29
16	AD30	D18	AD31
17	Gnd	D19	Gnd
18	IRQ*	D20	Gnd
19	DS*	D21	Gnd
20	WR*	D22	Gnd
21	SPACE0	D23	SIZE0
22	SPACE1	Gnd	PAS*
23	LOCK*	D24	SIZE1
24	ERR*	D25	Gnd
25	Gnd	D26	ACK*
26	Gnd	D27	AC
27	Gnd	D28	ASACK1*
28	Gnd	D29	ASACK0*
29	Gnd	D30	CACHE*
30	Gnd	D31	WAIT*
31	BGIN*	Gnd	BUSY*
32	BREQ*	+5	BGOUT*

8

The 7-segment Display

There is one 7-segment display on the front panel (Fig. 8-1) that can be programmed (Table 8-2). Writing a zero turns the chosen segment on; writing a one turns it off. At power-on or after a system reset, the default character is an H (segments b, c, e, f, and g are on).

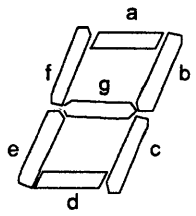


FIGURE 8-1. 7-segment display

TABLE 8-1
Addresses for the 7-segment display

Segment	Address (write-only)
a	$02B0,0010_{16}$
b	$02B0,0020_{16}$
c	$02B0,0030_{16}$
d	$02B0,0040_{16}$
e	$02B0,0050_{16}$
f	$02B0,0060_{16}$
g	$02B0,0070_{16}$

9

CIO Implementation

9.1 INTRODUCTION

The on-card CIO device performs a variety of functions. In addition to the three 16-bit timers, which may be used to generate interrupts or count events, the CIO has numerous parallel I/O bits.

The CIO has two independent 8-bit, bidirectional I/O ports (ports A and B) and a 4-bit special-purpose I/O port (port C). Data path polarity (whether bits are inverting or noninverting), data direction (whether bits are input or output), port configuration (bit port or handshake port), ones catchers, and open-drain outputs are programmable for all ports. The configuration and functions of the ports are programmed by means of the port specification registers for each port, which are described fully in the CIO technical manual.

9.2 PORT A BIT DEFINITION

Port A handles various control signals. All bits should be programmed as inputs.

TABLE 9-1
CIO port A bit definitions

Bit	Function	Data Path Polarity	Interface	HK68/V3D User's Manual Section
D7	External Interrupt	Negative True	P5-11	3.3
D6	EEPROM 1 Ready (U80)	Positive True	U80-1	6.2
D5	EEPROM 0 Ready (U70)	Positive True	U70-16	6.2
D4	Mailbox Interrupt	Negative True	—	7.8
D3	unused		—	
D2	VME Interrupt in Progress	Negative True	—	7.5
D1	SCSI Reset	Positive True	P2-A20	11
D0	Mailbox Halt	Positive True		

Bit D2 may be used to test if there is a pending interrupt still active from *this* board. The mailbox interrupt is a pulse, so the ones catcher should be used for that input bit.

9.3 PORT B BIT DEFINITION

The B port of the CIO is used to handle the Centronics interface interrupt (input) and generate the VMEbus interrupt mask bits (outputs).

Internal priorities of the CIO place D7 as highest (D0 as lowest) for simultaneous interrupts from either port.

TABLE 9-2
CIO port B bit definitions

Bit	Function	Data Path Polarity	Interface	HK68/V3D User's Manual Section
D7	Software Interrupt	Negative True	Interrupt Switch on front panel	12
D6	IRQ7 enable	Negative True	P1	3.2
D5	IRQ6 enable	Negative True	P1	3.2
D4	IRQ5 enable	Negative True	P1	3.2
D3	IRQ4 enable	Negative True	P1	3.2
D2	IRQ3 enable	Negative True	P1	3.2
D1	IRQ2 enable	Negative True	P1	3.2
D0	IRQ1 enable	Negative True	P1	3.2

9.4 PORT C BIT DEFINITION

Port C on the CIO chip is used to read four on-card status signals. SYSOK* turns off the SYSFAIL LED, and it must be true (1) before you can halt the CPU with the mailbox halt.

TABLE 9-3
CIO Port C bit definitions

Bit	Function	Data Path Polarity
D3	VMEbus ACFAIL	Positive True
D2	VMEbus SYSFAIL*	Negative True
D1	VMEbus SYSOK* utility bit	Positive True
D0	Port A Ring Indicator	Negative True

9.5 COUNTER/TIMERS

There are three independent, 16-bit counter/timers in the CIO. For long delays, timers 1 and 2 may be internally linked together to form a 32-bit counter chain. When programmed as timers, the following equation may be used to determine the time constant value for a particular interrupt rate.

TC = 2,457,600 / interrupt rate (in Hz)

When the timer is clocked internally, the count rate is 2.4576 MHz. The HK68/V3D board uses a 19.6608 MHz clock oscillator as the system time base. The frequency tolerance specification is ±0.01%. If you are using the 19.6608 MHz clock as the CIO time base, the maximum accumulative timing error will be about 9 seconds per day, although the typical error is less than one second per day. Better long-term accuracy may be achieved via a power line (60 Hz) interrupt, using a bus interrupt or the Real-Time Clock (RTC) option (see section 13).

9.6 REGISTER ADDRESS SUMMARY (CIO)

TABLE 9-4
CIO register addresses

Register	Address	Function
Port C, Data	$02D0,0001_{16}$	Miscellaneous Control Bits
Port B, Data	$02D0,0003_{16}$	Miscellaneous Control Bits
Port A, Data	$02D0,0005_{16}$	Miscellaneous Control Bits
Control Registers	$02D0,0007_{16}$	CIO Configuration and Control

All registers are eight bits wide.

9.7 CIO INITIALIZATION

The following figure shows a typical initialization sequence for the CIO. The first byte of each data pair in "ciotable" specifies an internal CIO register; the second byte is the control data. The specific directions of some of the PIO lines and interrupts need to be changed in the table, based on your application. An active low signal can be inverted (so that a "1" is read from the data port when the signal is true) by initializing the port to invert that particular bit. Refer to section 3 for information concerning CIO interrupt vectors.

EXAMPLE 9-1. CIO program (C portion)

```
char ciotable[] = {
0x00, 0x01, 0x00,/* reset, set chip ptr to reg zero */

/* port A initialization */
0x20, 0x06,  /* bit port, priority encoded vector */
0x22, 0x9c,  /* invert negative true bits */
0x23, 0xff,  /* all bits are inputs */
0x24, 0x10,  /* one's catcher */
0x25, 0x00,  /* pattern polarity register */
0x26, 0x00,  /* all levels (can't use transitions */
/* in "or priority mode") */
0x27, 0x10,  /* pattern mask, enable mailbox interrupt */
0x02, 0x41,  /* set interrupt vector */
0x08, 0xc0,  /* set int enable, no int on err */

/* port B initialization */
0x28, 0x06,  /* bit port, priority encoded vector */
0x2a, 0x80,  /* invert negative true bit */
0x2b, 0x80,  /* one bit is an input */
0x2c, 0x00,  /* normal input (no ones catchers) */
0x2d, 0xff,  /* bit interrupt on a one */
0x2e, 0x00,  /* no transition, levels only */
0x2f, 0x00,  /* no interrupts enabled */
0x03, 0x40,  /* set interrupt vector */
0x09, 0xc0,  /* set int enable, no int on err */

       /* port c initialization */
0x05, 0x0f,  /* invert negative true bits */
0x06, 0x0f,  /* all bits are inputs */
0x07, 0x00,  /* normal inputs */

/* timer 3 and other CIO initialization */
0x1e, 0x80,  /* set mode to auto reload */
0x1a, 0xa0,  /* high byte delay constant */
0x1b, 0x00,  /* low byte delay constant */
0x04, 0x60,  /* interrupt vector */
0x08, 0x20,  /* clear any port A ints */
0x08, 0x20,  /* clear any port A ints */
0x01, 0x94,  /* enable timer 3, port a and port b */
0x0c, 0xc6,  /* set interrupt enable and */
/* gate command bit and trigger cmd bit */
0x00, 0x9e   /* master int enable and vector includes */
/* status for timer 3, port A and port B */
};

struct cdevice {  /* CIO register structure */
  char dummy0; char cdata;    /* port C */
  char dummy1; char bdata;    /* port B */
  char dummy2; char adata;    /* port A */
  char dummy3; char ctrl; /* control port */
};
#define CIO ((struct cdevice *)0x02d00000)

cioinit()
{
  int i, t3intr();
 /*Don't forget to set CIO interrupt vectors. Example: */
  *(int(*))(0x60*4) = (int)t3intr;/* Timer 3 interrupt */
  i = CIO->ctrl;       /* assure register sync */
  CIO->ctrl = ciotable[0]; /* avoid clr instruction*/
```

```
  i = CIO->ctrl;      /* assure register sync */
  for (i = 0; i < sizeof(ciotable); i++)
       CIO->ctrl = ciotable[i];/* send ciotable to CIO chip
*/
}

Aintr()  /* clear Port A interrupt */
       /* one of 8 routines */
{  /* process port A interrupts here */
  CIO->ctrl = 0x08; CIO->ctrl = 0x20;
}

Bintr()/* clear Port B interrupt */ /* one of 8 routines */
{  /* process port B interrupts here */
  CIO->ctrl = 0x09; CIO->ctrl = 0x20;
}

timer3() /* clear Timer 3 interrupt, get here via t3intr */
{  /* process timer interrupt here */
  CIO->ctrl = 0x0c; CIO->ctrl = 0x24;
}
```

EXAMPLE 9-2. CIO Program example (assembly code portion)

```
     .globl  t3intr%, timer3

# the vector at 0x60*4 points to this routine

t3intr%: movm.l &0xFFFF,-(%sp) # save registers
     jsr   timer3     # to C portion
     movm.l (%sp)+,&0xFFFF # restore registers
     rte
```

9.8 CIO PROGRAMMING HINTS

1. To maintain compatibility with 68010 programs, do not use the 68030 **clr.b** instruction to set a CIO register to zero. On the 68000 and 68010, that instruction does a "phantom" read of the port before it does the zero write. The read operation will upset the CIO internal register selection sequencer. Similarly, when using a high level language, do not set a CIO register value to the constant "0" because the compiler may use a **clr.b**. Use a variable which is set to zero, or output the values from a lookup table. For example:

   ```
   zero = 0;
   *CIOcntrl = 0x20;
   *CIOcntrl = zero;
   ```

2. The ones catchers in a CIO port will be cleared whenever any bits are changed in the pattern mask register. Avoid changing the mask register if you are using a ones catcher. If this is not possible, a program that writes to the pattern

mask register should first OR the CIO data register into a memory variable. Later, that memory value can be ORed with the CIO data register to find out what the data register would have been if the CIO had not cleared it. Routines which respond to a ones catcher interrupt must clear the corresponding bits in the memory value and the CIO data register. There will still be a critical period where a fast input pulse could be missed, even when using this scheme.

3. If you get an unexpected interrupt from bit D0 of a CIO port, it may be because another enabled CIO input signal went false before the MPU initiated the interrupt acknowledge cycle. The use of a ones catcher may be appropriate to latch the input line.

4. If you turn on a bit in the pattern mask register, that bit will generate an interrupt (if the port is enabled) even if the input signal is false. To prevent this, disable the port while adjusting the pattern mask register.

5. The CIO may glitch the parallel port lines when a hardware reset is done, even if all lines are programmed as inputs. This may cause a problem in multi-processor systems because the glitches may produce spurious ACFAIL and SYSFAIL signals on other (operating) boards. To prevent this effect, disable the port (via software) prior to doing a board reset.

Refer to the Z8536 technical manual for more details on programming the CIO. Some people find the CIO technical manual difficult to understand. We encourage you to read all of it twice, before you pass judgment. Especially study sections.2.10.1 and 3.3.2

10

Serial I/O

10.1 INTRODUCTION

There are two RS-232C serial I/O ports on the HK68/V3D board (Fig. 10-1). Each port may optionally be configured for RS-422 operation with a special interface cable, as described in section 10.8. Each port has a separate baud rate generator and can operate in asynchronous or synchronous modes.

10.2 RS-232 PIN ASSIGNMENTS

Data transmission conventions are with respect to the external serial device. The HK68/V3D board is wired as data communications equipment (DCE). The connector pin assignments are shown in Table 10-1:

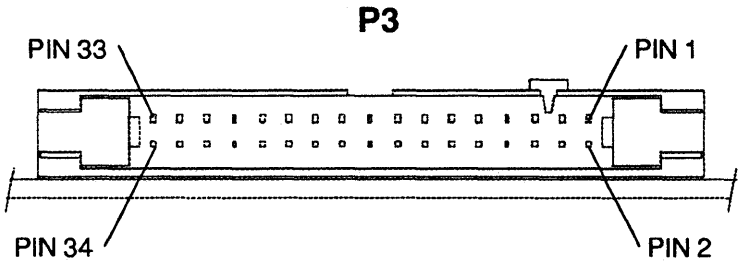


FIGURE 10-1. Serial connector, P3

TABLE 10-1A
Port A serial port pin assignments, P3

P3 Pin Number	"D" Pin	RS-232 Function	Direction	SCC Pin Function
1	2	Port A Tx Data	In	Rcv Data
2	15	Tx Clock	In	
3	3	Rcv Data	Out	Tx Data
4	16	(not used)		
5	4	Request to Send *	In	DCD
6	17	Rcv Clock	In	
7	5	Clear to Send	Out	DTR
8	18	Ring Detect	In	Ring Ind
9	6	Data Set Ready	Out	RTS
10	19	(not used)		
11	7	Gnd		Sig Gnd
12	20	Data Terminal Ready*	In	CTS

TABLE 10-1B
Port B serial port pin assignments, P3

P3 Pin Number	"D" Pin	RS-232 Function	Direction	SCC Pin Function
13	2	Port B Tx Data	In	Rcv Data
14	15	Tx Clock	In	
15	3	Rcv Data	Out	Tx Data
16	16	+12v (via J2)		
17	4	Request to Send *	In	DCD
18	17	Rcv Clock	Out	
19	5	Clear to Send	Out	DTR
20	18	+5v (via JX)		
21	6	Data Set Ready	Out	RTS
22	19	-12v (via J2)		
23	7	Gnd		Sig Gnd
24	20	Data Terminal Ready*	In	CTS

Note that the interconnect cable from P3 is arranged in such a manner that the "D" connector pin assignments are correct for RS-232C conventions. Not all pins on the "D" connectors are used. Recommended mating connectors are Ansley P/N 609-5001CE and Molex P/N 15-29-8508.

Signals indicated with "*" have default pull-up resistors, controlled by J2. NOTE: The serial ports may *appear* to be inoperative if J2 is set to default "FALSE" and if the device connected to the port does not drive the DTR and RTS pins TRUE. The HK68/V3D monitor software, for example, initializes the SCC channels to respect the state of DTR and RTS. The RI signals for port A is routed to the CIO. See section 10.9.

10.3 SIGNAL NAMING CONVENTIONS (RS-232)

Since the RS-232 ports are configured as DCE, the naming convention for the interface signals may be confusing. The interface signal names are with respect to the terminal device attached to the port while the SCC pins are with respect to the SCC as if it, too, is a terminal device. Thus all signal pairs, e.g., "RTS" and "CTS," are switched between the interface connector and the SCC. For example, "Transmit Data," P3-1, is the data transmitted from the device to the HK68/V3D board; the data appear at the SCC receiver as "Received Data." For the same reason, the "DTR" and "RTS" interface signals appear as the "CTS" and "DSR" bits in the SCC, respectively. If you weren't confused before, you might be by now. Study the chart below and see if that helps.

TABLE 10-2
Signal naming conventions

SCC Signal	Interface Signal	Direction
Tx Data	Rcv Data	to device
Rcv Data	Tx Data	from device
Tx Clock	Rcv Clock	from device (port A)
Tx Clock	Rcv Clock	to device (port B)
Rcv Clock	Tx Clock	from device
RTS	DSR	to device
CTS	DTR	from device
DTR	CTS	to device
DCD	RTS	from device
—	Ring Indicator	from device

The SCC was designed to look like a DTE. Using it as a DCE creates this nomenclature problem. Of course, if you connect the HK68/V3D board to a modem (DCE), then the SCC signal names are correct, however, a cable adapter is needed to properly connect to the modem. (Three pairs of signals must be reversed.)

TABLE 10-3
RS-232 cable reversal

SCC Signal	P3 Pin #s	"D" Pin # at HK68/V3D	"D" Pin # at modem	RS-232 Signal
x	x	1	1	Protective Ground
Rcv Data	1	2	3	Rcv Data
Tx Data	3	3	2	Tx Data
DCD	5	4	6	DSR
RTS	9	6	4	RTS
DTR	7	5	20	DTR
CTS	12	20	5	CTS
Ring Indicator	8	18	22	Ring Indicator
Signal Ground	11	7	7	Signal Ground

Summary: The HK68/V3D may be directly connected to a data "terminal" device (DTE). A cable reversal is required for a connection to a DCE device, such as a modem.

10.4 CONNECTOR CONVENTIONS

Paragraph 3.1 of the EIA RS-232-C standard says the following concerning the mechanical interface between data communications equipment:

> *"The female connector shall be associated with...the data communications equipment... An extension cable with a male connector shall be provided with the data terminal equipment... When additional functions are provided in a separate unit inserted between the data terminal equipment and the data communications equipment, the female connector...shall be associated with the side of this unit which interfaces with the data terminal equipment while the extension cable with the male connector shall be provided on the side which interfaces with the data communications equipment."*

Substituting "modem" for "data communications equipment" and "terminal" for "data terminal equipment" leaves us with the impression that the modem should have a *female* connector and the terminal should have a *male.*

The Heurikon HK68/V3D microcomputer interface cables are designed with female "D" connectors, because the serial I/O ports are configured as DCE (modems). Terminal manufacturers

typically have a female connector also, despite the fact that they are terminals, not modems. Thus, the extension cable used to run between a terminal and the HK68/V3D (or a modem) has male connectors at both ends.

When you work with RS-232 communications, you might end up with many types of cable adapters — double males, double females, double males and females with reversal, or cables with males and females at both ends. We will be happy to help make special cables to fit your needs.

10.5 SCC INITIALIZATION SEQUENCE

Table 10-4 shows a typical initialization sequence for the SCC. This example is for port A. Port B is programmed in the same manner, substituting the correct control port address.

TABLE 10-4
SCC initialization sequence

Data	Register Adress	Function
00	$02F0,0003_{16}$ (write)	Reset SCC register counter
09,C0	$02F0,0003_{16}$ (write)	Force reset (for port A only)
04,4C	$02F0,0003_{16}$ (write)	Async mode, x16 clock, 2 stop bits tx
05,EA	$02F0,0003_{16}$ (write)	Tx: RTS, Enable, 8 data bits
03,E1	$02F0,0003_{16}$ (write)	Rcv: Enable, 8 data bits
01,00	$02F0,0003_{16}$ (write)	No Interrupt, Update status
0B,56	$02F0,0003_{16}$ (write)	No Xtal, Tx & Rcv clk internal,BR out
0C,baudL	$02F0,0003_{16}$ (write)	Set Low half of baud rate constant
0D,baudH	$02F0,0003_{16}$ (write)	Set high half of baud rate constant
0E,03	$02F0,0003_{16}$ (write)	Null, BR enable

The notation "09,C0" (etc.) means the values 09 (hexadecimal) and C0 should be sent to the specified SCC port. The first byte selects the internal SCC register; the second byte is the control data. The above sequence only initializes the ports for standard asynchronous I/O without interrupts. The 'baudL' and 'baudH' values refer to the low and high halves of the baud rate constant, which may be determined from the Baud Rate Constants section below.

For information concerning SCC interrupt vectors, refer to section 3. Consult the Z8530 technical manual for more details on SCC programming.

To maintain compatibility with 68010 programs, do not use the 68030 **clr.b** instruction to set a SCC register to zero. On the 68000 and 68010, that instruction does a "phantom" read of the port before it does the zero write. The read operation will upset the SCC internal register selection sequencer. Similarly, when using a high level language, do not set a SCC register value to the constant "0" because the compiler may use a **clr.b**. Use a variable that is set to zero, or output the values from a lookup table. For example, this is correct:

```
zero = 0;
*SCCcntrl = 0x20;
*SCCcntrl = zero;
```

10.6 PORT ADDRESS SUMMARY

TABLE 10-5
SCC register addresses

Register	Port A	Port B	Port C	Port D
Control	$02F0,0003_{16}$	$02F0,0001_{16}$	$02E0,0003_{16}$	$02E0,0001_{16}$
Data	$02F0,0007_{16}$	$02F0,0005_{16}$	$02E0,0007_{16}$	$02E0,0005_{16}$

All ports are eight bits.

10.7 BAUD RATE CONSTANTS

If the internal SCC baud rate generator logic has been selected, the actual baud rate must be specified during the SCC initialization sequence by loading a 16-bit time constant value into each generator. Table 10-6 lists the values to use for some common baud rates. Other rates may be generated by applying the formula given below.

TABLE 10-6
Baud rate constants

Baud Rate	x1 clock rate	x16 clock rate
110	22,340	1,394
300	8,190	510
1200	2,046	126
2400	1,022	62
4800	510	30
9600	254	14
19,200	126	6
38,400	62	2

The time constant values listed above are computed as follows:

$$TC = 4{,}915{,}200/(2 * baud * factor) - 2$$

The ×16 mode will obtain better results with asynchronous protocols because the receiver can search for the middle of the start bit. (In fact, the ×1 mode will probably produce frequent receiver errors.)

The maximum SCC data speed is one megabit per second, using the ×1 clock and synchronous mode. For asynchronous transmission, the maximum practical rate using the ×16 clock is 51,200 baud.

10.8 RS-422 OPERATION

As an option, one or more of the serial ports on the HK68/V3D may be configured for RS-422 operation. The RS-422 option may either be installed when the board is ordered, or an existing HK68/V3D board may be factory-upgraded to add the option. Please contact Heurikon for more information.

10.9 RELEVANT JUMPERS (SERIAL I/O)

TABLE 10-7
Serial I/O jumpers

Jumper	Function	Options	Standard Configuration
J2	RS-232 ports A and B status default	J2-A (True) J2-B (False)	J2-A (True)
J3	Selects Ring Indicator of Data Carrier Detect for port A.	J3:1–2 (RI) J3:2–3 (DCD)	J3:2–3 (DCD)

10.10 SERIAL I/O CABLE

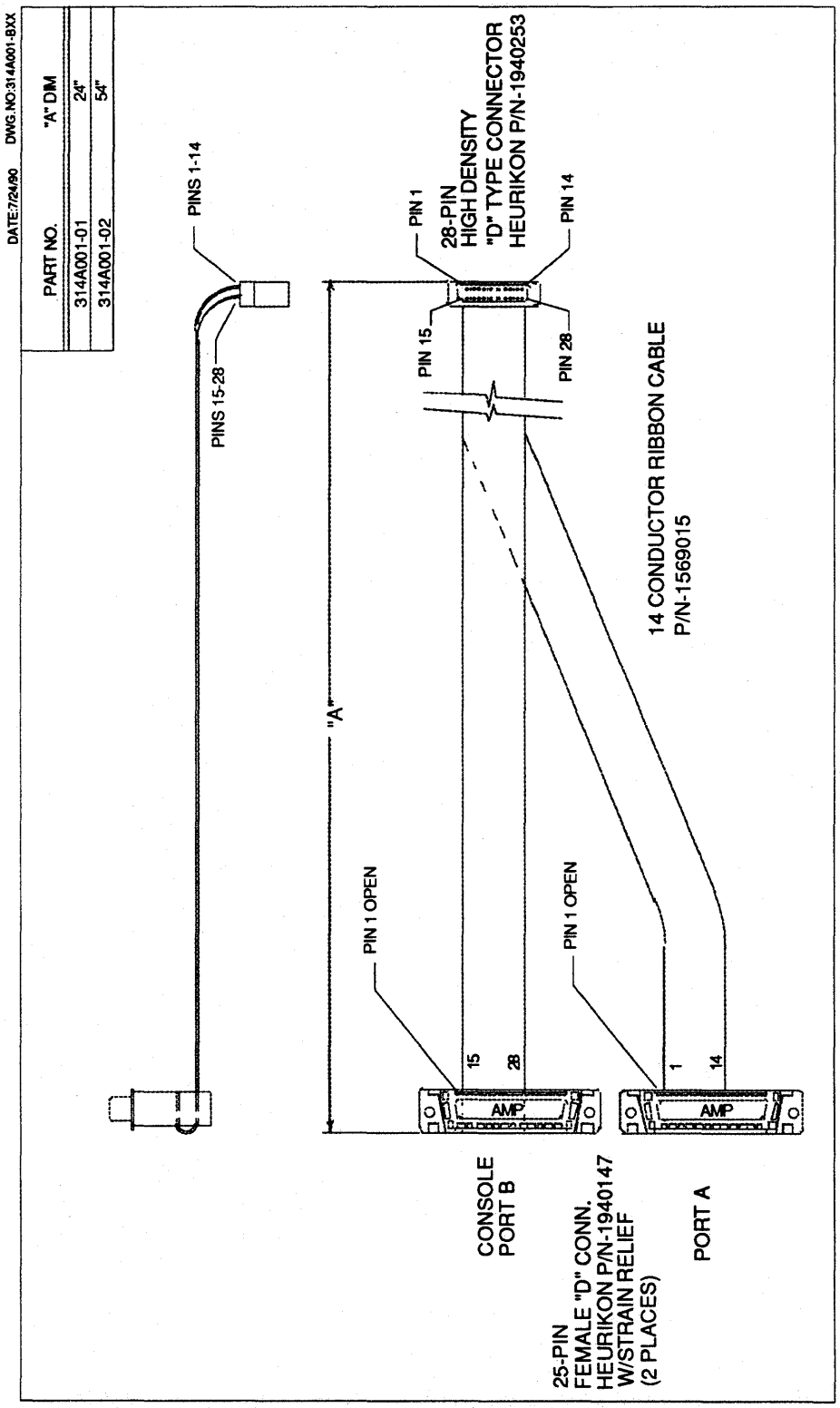


FIGURE 10-2. Serial I/O cable

11

Optional SCSI Port

11.1 INTRODUCTION

The HK68/V3D uses the Western Digital WD33C93 chip to implement a Small Computer System Interface (SCSI) port.

The SCSI port may be used to connect the HK68/V3D with a variety of peripheral devices, such as memory storage devices and streamer tape drives.

Supported features and modes include:

- Initiator role
- Target role
- Arbitration
- Disconnect
- Reconnect

11.2 SCSI IMPLEMENTATION NOTES

The SCSI Data Ready signal is routed to the CIO, which can cause an MPU interrupt. The interrupt from the SCSI chip generates a level 4 autovector. See MPU exception vectors, section 3.3 for details. Data transfer functions can be handled in a polled I/O mode.

11.3 REGISTER ADDRESS SUMMARY (SCSI)

TABLE 11-1
SCSI register address summary

Address	R/W	Bits	Function
$0230,0001_{16}$	W	8	Set Controller Address Register
$0230,0001_{16}$	R	8	Read Auxiliary Register
$0230,0003_{16}$	R/W	8	SCSI Controller Registers
$0240,0000_{16}$	R/W	8	SCSI Data Register (pseudo-DMA)
$02B0,0006_{16}$	W	1	SCSI Bus Reset (1=reset, 0=release)
$02B0,0020_{16}$	W	1	SCSI Interrupt Enable (1=enable)

11.4 SCSI PORT PINOUTS

The SCSI option uses rows A and C of connector P2 (Fig. 11-1 and Table 11-2).

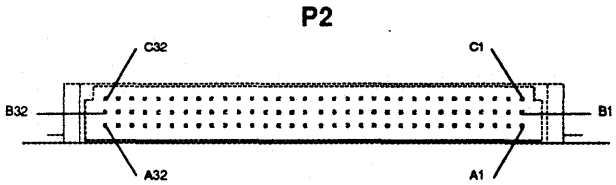


FIGURE 11-1. SCSI connector, P2

TABLE 11-2
SCSI pin assignments, P2

P2 Pin Number	Row A SCSI Signal Mnemonic	Row B VMEbus Signal Mnemonic	Row C
1	DB(0)	+5	Gnd
2	DB(1)	Gnd	Gnd
3	DB(2)	(reserved)	Gnd
4	DB(3)	A24	Gnd
5	DB(4)	A25	Gnd
6	DB(5)	A26	Gnd
7	DB(6)	A27	Gnd
8	DB(7)	A28	Gnd
9	DB(P)	A29	Gnd
10	Gnd	A30	Gnd
11	Gnd	A31	Gnd
12	Gnd	Gnd	Gnd
13	SCSI_VCC TERMPWR	+5	not used
14	Gnd	D16	Gnd
15	Gnd	D17	Gnd
16	ATN	D18	Gnd
17	Gnd	D19	Gnd
18	BSY	D20	Gnd
19	ACK	D21	Gnd
20	RST	D22	Gnd
21	MSG	D23	Gnd
22	SEL	Gnd	Gnd
23	C/D	D24	Gnd
24	REQ	D25	Gnd
25	I/O	D26	Gnd
26	Gnd	D27	not used
27	Gnd	D28	not used
28	Gnd	D29	not used
29	Gnd	D30	not used
30	Gnd	D31	not used
31	not used	Gnd	not used
32	not used	+5	not used

Recommended mating connectors are [Ansley P/N 609-5001CE and Molex P/N 15-29-8508].

11.5 SCSI BUS TERMINATION

The HK68/V3D provides the recommended [SCSI-2] termination of 110 ohms to 2.85 volts.

Resistor networks RN29. RN30, and RN31 are socketed SCSI terminators located next to connector P2 (Fig. 11-3). The SCSI specification requires that the bus be terminated at both ends of the cable, so RN29, RN30, and RN31 should be installed only if the module is at an end of the SCSI interface cable. Power for the SCSI termination on the HK68/V3D is taken from the SCSI bus TERMPWR signal (P2-A13).

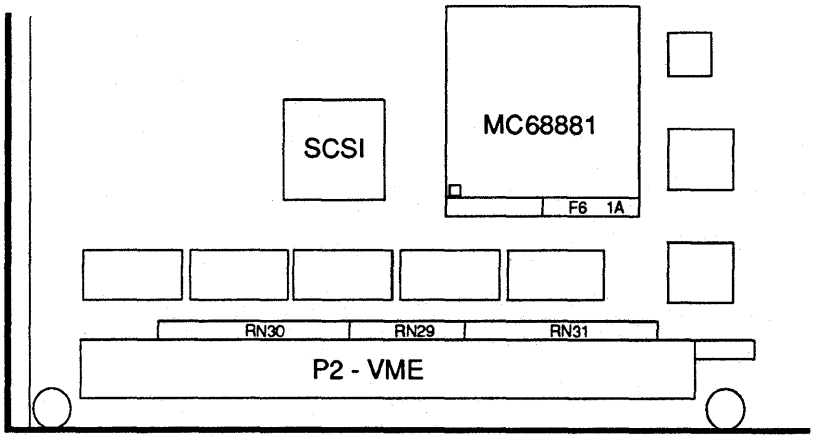


FIGURE 11-2. Location of SCSI terminating resistor networks and fuse F6

The SCSI specification requires that initiators supply power to the TERMPWR signal. The HK68/V3D drives TERMPWR through fuse F6 (Fig. 11-3). The HK68/V3D will not drive TERMPWR if the fuse is removed.

12

Optional Ethernet Interface

12.1 INTRODUCTION

The HK68/V3D can be order with an Ethernet interface option, which consists of a network interface controller and a serial network interface. The network interface controller is an Intel 82596CA 32-bit local area network coprocessor. The serial network interface is an 82C501AD encoder/decoder. Together, these components implement a standard IEEE-802.3 CSMA/CD 10BASE5 (10-megabit-per-second) Ethernet interface.

12.1.1 Network Interface Controller (82596CA)

The 82596CA performs complete CSMA/CD Medium Access Control (MAC) functions according to the IEEE 802.3 independently of the CPU. Significant features of the 82596CA include:

- On-chip memory management
- On-chip DMA with a 32-bit RAM interface
- Network statistics collection
- Transmit FIFOs and receive FIFOs
- Network monitor mode
- Self-test diagnostics and loopback mode

12.1.2 Serial Network Interface (82C501AD)

The 82C501AD interfaces the 82596CA to the Ethernet network and performs the required Manchester encoding and decoding of the Ethernet signals. Significant features of the 82C501AD include:

- Loopback capability for diagnostics
- Adaptability to either Ethernet 1.0 or IEEE-802.3 transceivers via jumper selection. On the HK68/V3D, jumper J1 is used for this configuration. See section 12.10.

12.2 ETHERNET ADDRESS

The importance of maintaining a correct Ethernet address for the HK68/V3D is best expressed by this excerpt from the IEEE document entitled *Discussion of the Use of 48-bit LAN Globally Assigned Address Block* (12-29-88):

> ***The concept of Global/Universal Addressing is based upon the idea that all potential members of a network need to have a unique identifier if they are to exist in a network. The advantage of a Global LAN Address is that a node with such an address can be attached to any LAN network in the world with a high degree of assurance that no other node on that network will share its address. The concept of the 48-bit address scheme originated with Xerox's ETHERNET, but it is applicable to all equipment meeting IEEE 802 committee address assignment protocol methods, and equivalent standards.***

The Ethernet address for your board is an identifier that gives your board a unique address on a network and must not be altered. The address consists of 48 bits divided into two equal parts. The upper 24 bits define a unique identifier that has been assigned to Heurikon Corporation by IEEE. The lower 24 bits are defined by Heurikon Corporation for unique identification of each of its products.

12.2.1 Verifying the Ethernet Address

For convenience, the binary address is referenced as 12 hexadecimal digits, separated into pairs. Each pair represents eight bits. Heurikon's identifier is **00 80 F9**. Heurikon uses the fourth group of eight bits as a product code, and the fifth and sixth groups to identify each board within the product group (Fig. 12-1).

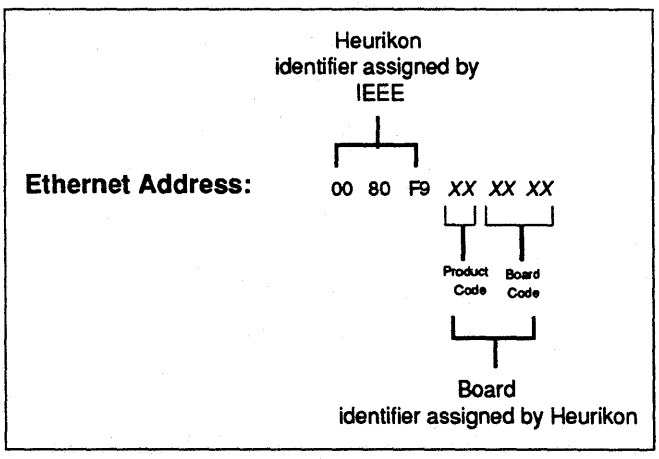


FIGURE 12-1. Ethernet address format

12.2.2 Ethernet Address on the HK68/V3D

Each HK68/V3D's address depends on information stored in nonvolatile memory. The address assigned to an HK68/V3D has the following form:

00 80 F9 *XX XX XX*

where the first three pairs (00 80 F9) are the Heurikon identifier, the fourth pair (*XX*) is the identifier for the HK68/V3D product group, and the fifth and sixth pairs (*XX XX*) constitute a unique value assigned to each HK68/V3D. The Ethernet address for your board is labelled on the 82596CA.

See Appendix A for information on how to read the board's address from its nonvolatile memory.

12.3 82596CA IMPLEMENTATION ON THE HK68/V3D

This section summarizes the configuration and limitations of the 82596CA as it is used on the HK68/V3D. Many of the items noted here are described in greater detail in subsequent sections.

12.3.1 82596CA Configuration on the HK68/V3D

Big-endian Byte Ordering

The 82596CA can be configured for use in either big-endian or little-endian mode.

On the HK68/V3D, the 82596CA is hard wired for big-endian mode.

32-bit Bus Width

The 82596CA can be configured for 32-bit and 16-bit bus widths.

On the HK68/V3D, the 82596CA is hard wired for 32-bit data bus operation.

Interrupt Enable

The 82596CA interrupt generates a level 1 interrupt vector (vector 25). The 82596CA itself provides no means to enable or disable the interrupt, but logic on the board provides that function.

12.3.2 82596CA Parameter Selections

The shared memory structure between the 82596CA and the HK68/V3D has four parts: Initialization Root, System Control Block, Command List, and Receive Frame Area. The Initialization Root contains the System Configuration Pointer and Intermediate System Configuration Pointer.

The System Configuration Pointer points to the Intermediate System Configuration Pointer, which, in turn, points to the System Control Block, where the CPU and the 82596CA exchange control and status information.

The System Configuration Pointer also contains the SYSBUS byte, which is used to determine addressing mode, bus throttle triggering method, and interrupt polarity, and to enable locked bus cycles.

The CPU can access the 82596CA directly via the $\overline{\text{PORT}}$ pin and CA (Channel Attention) pins. The first CA signal after a valid RESET causes the 82596CA to read the initialization sequence beginning either at a default address or at an alternate System Configuration Pointer (SCP) address, which can be changed directly through the $\overline{\text{PORT}}$ access. All subsequent CA signals cause the 82596CA to execute new command sequences from the System Control Block.

System Configuration Pointer Address

The 82596CA uses a default System Configuration Pointer address of $00FF,FFF4_{16}$.

For all applications, this address for the System Configuration Pointer must be changed via a Port command before issuing the first Channel Attention command.

Addressing Mode

The 82596CA supports three operational modes: 82586, 32-bit segmented, or linear.

On the HK68/V3D, the 82596CA supports linear addressing mode. Thirty-two-bit segmented mode should also work, but is not supported by Heurikon on the HK68/V3D. The 82596CA *cannot* be used in 82586-compatibility mode. Addressing mode is set by bits 1 and 2 of the SYSBUS byte of the System Configuration Pointer.

Bus Throttle Timer

The 82596CA is designed to accommodate internal or external triggering of the bus throttle timers.

On the HK68/V3D, the BREQ pin of the 82596CA is hard wired to ground. Therefore, bit 3 of the SYSBUS byte of the System Configuration Pointer must be 0_2 to use internal triggering of the bus throttle timers.

Locked Bus Cycles

Locked bus cycles by the 82596CA are supported as an option for semaphore operations with the HK68/V3D.

Interrupt Polarity

Bit 4 of the SYSBUS byte is used to set interrupt polarity active high or active low.

Logic on the HK68/V3D expects the 82596CA interrupt to be active high, so bit 4 of the SYSBUS byte of the System Configuration Pointer must be 0_2.

12.4 BYTE ORDERING

The 82596CA supports both big-endian and little-endian byte ordering. A review of the 82596CA user's manual shows, however, that the 82596CA is fundamentally a little-endian part with enhancements to support big-endian byte ordering. (Refer to section 1.6.2 for an explanation of big-endian and little-endian byte ordering.)

On the HK68/V3D, the 82596CA is hard wired to big-endian mode. As a programming reference, it is helpful to use the big-endian chapter of the 82596CA user's manual. The 82596 data sheet is written from a little-endian point of view and can be confusing when the chip is used in big-endian mode. The big-endian chapter of the user's manual can be helpful, but it must be used carefully because it contains many small errors and inconsistencies.

Programming Note

If all elements that constitute the 82596CA control structures are defined as 16-bit words, then the same structures definitions may be used for both big-endian and little-endian modes, and 82596CA driver software should be largely independent of the mode.

12.5 ETHERNET ACCESS

The HK68/V3D can communicate with the Ethernet by means of either Port or CA access, which are summarized in Table 12-1.

TABLE 12-1
Ethernet accesses

Access	R/W	Address	D19-D16	Function
$\overline{\text{PORT}}$	W	$02E0,0000_{16}$	0	Reset the 82596CA.
			1	Perform a self test on the 82596CA.
			2	Write a new SCP address.
			3	Dump the 82596CA registers.
CA	W	$02E0,0004_{16}$	X	Channel Attention

12.5.1 Port Access

The 82596CA has a CPU port access interface that allows the CPU to cause the 82596CA to execute any of the Port functions shown in Table 12-1.

$\overline{\text{PORT}}$ accesses require four writes on the HK68/V3D. Section 5.3.6.3 of the *82596 User's Manual* says that all $\overline{\text{PORT}}$ accesses must be 16-bit accesses. Thus, a 32-bit Port command requires two writes to the 82596CA's $\overline{\text{PORT}}$. Table 12-2 shows the order for writing the upper and lower words and the data lines on which the command value is transferred. Furthermore, we at Heurikon have found that it is necessary to repeat the port access, although this procedure is not documented in the 82596CA user's manual, for a total of four writes.

In practice, then, we write the Port command value *four times* to the 82596CA's $\overline{\text{PORT}}$ for a Port command to be executed.

Programming Note

Watch compiler optimization. A succession of four writes to the same address may be optimized by a compiler to a single write.

TABLE 12-2
Port access definition

	First Access	Second Access
Big endian	D15-D0 > Lower Command Word	D31-D16 > Upper Command Word

The format to the port commands as given in Table 12-5 of the Big-endian chapter of the 82596CA User's Manual is *incorrect*. The correct format, shown in Table 12-3 below, swaps the two halves of the port command long word.

TABLE 12-3
Port accesses

Address is $02E0,0000_{16}$									
Function	**D31 . . . . . D20**			**D19**	**D18**	**D17**	**D16**	**D15 . . . . . D0**		
Reset	A15...	Don't care	...A4	0	0	0	0	A31...	Don't care	...A16
Self-test	A15...	Self-test results address	...A4	0	0	0	1	A31...	Self-test results address	...A16
New SCP	A15...	Alternate SCP address	...A4	0	0	1	0	A31...	Alternate SCP address	...A16
Dump	A15...	Dump area pointer	...A4	0	0	1	1	A31...	Dump area pointer	...A16

12.5.2 Channel Attention (CA)

Accessing address $02E0,0004_{16}$ issues Channel Attention (CA) to the 82596CA and causes it to begin executing memory-resident command blocks. The first CA after a reset forces the 82596CA into the initialization sequence beginning at location $00FF,FFF4_{16}$ or an alternate SCP address written to the 82596CA using the $\overline{\text{PORT}}$ access mechanism. All subsequent CAs cause the 82596CA to begin executing new command sequences (memory-resident command blocks) from the System Control Block.

Since the default SCP address ($00FF,FFF4_{16}$) is not accessible memory on the HK68/V3D, the Alternate SCP PORT Access command must be issued prior to the first CA after a reset.

12.6 SYSBUS BYTE OF THE SYSTEM CONFIGURATION POINTER

The SYSBUS byte (Fig. 12-2 and Table 12-4) is composed of bits 7–0 of the first long word of the System Configuration Pointer.

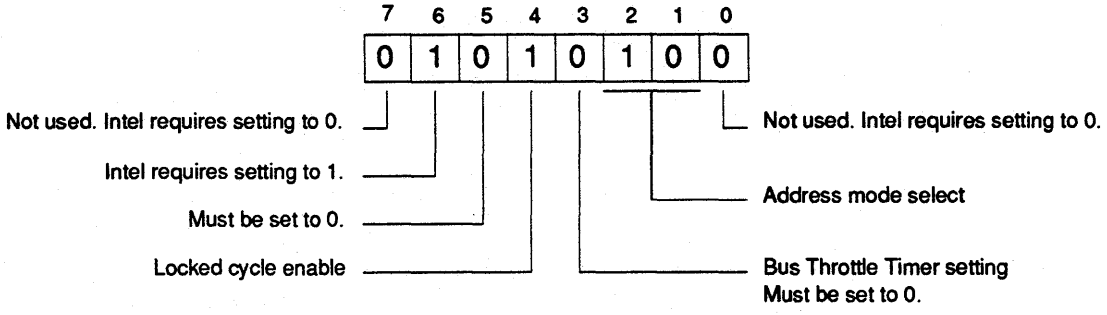


FIGURE 12-2. Required settings of the System Configuration Pointer SYSBUS byte

TABLE 12-4
SYSBUS byte selections

Bit	Function	Selections	Description
0	Not used	—	This bit must be set to 0_2, according to Intel documentation.
2 and 1	Address Mode select	00_2 = 82586 mode 01_2 = 32-bit segmented mode 10_2 = linear mode 11_2 = reserved	The HK68/V3D supports linear mode (bit 2:1 = 10_2). 32-bit segmented mode should also work but is not supported by Heurikon. 82586 mode *cannot* be used.
3	Bus Throttle Timer triggering	0_2 = internal 1_2 = external	The 82596CA's BREQ pin is tied to ground on the board, so external Bus Throttle timer triggering is not possible. Bit 3 must be 0_2.
4	Locked cycles enable	0_2 = enable 1_2 = disable	Locked cycles are an option that can be used for updating the Ethernet statistics counter. Both selections are supported on the HK68/V3D.
5	82596CA interrupt	0_2 = active high 1_2 = active low	Logic on the board expects the 82596CA's INT signal to be active high. Bit 5 must be 0_2.
6		1_2	The default must be used, according to an erratum from Intel.
7	Not used.	—	This bit must be set to 0_2, according to Intel documentation.

12.7 RECOMMENDED INITIALIZATION

1. Reset the 82596CA with a Port Reset command.

2. Construct the System Control Pointer (SCP), Intermediate System Control Pointer, and System Control Block structure. Initialize the SYSBUS byte of the SCP to 44_{16} or 54_{16}

Note: The Alternate SCP Port command (step 4) requires the SCP address to be 16-byte aligned, that is, at an address such as $XXXX,XXX0_{16}$.

3. Initialize interrupts.

4. Issue an Alternate SCP Port command to the 82596CA, followed by a Channel Attention.

Note: In big-endian mode, the SYSBUS byte is bits 7–0 of the first long word of the System Configuration Pointer.

12.8 ADDRESSES OF ETHERNET FUNCTIONS

All Ethernet functions may be accessed as long words at the addresses given in Table 12-5. Except for the Port command, each function may also be accessed as a byte at the byte address that corresponds to the least significant byte of the long word, that is, long word address plus 3.

TABLE 12-5
Ethernet peripheral addresses

Address	Function	Notes
$02E0,0000_{16}$	Write: 82596CA Port command	This address MUST be accessed as a long word. Writing to this address generates the $\overline{\text{PORT}}$ signal to the 82596CA. The data for the write is the 32-bit value to be latched by the 82596CA. See Table 12-3.
	Read: Not used.	
$02E0,0004_{16}$	Write: 82596CA Channel Attention command	Writing to this address generates the CA signal to the 82596CA. The data for the write is of no consequence.
	Read: Not used.	
$02E0,0008_{16}$	Write: Not used.	
	Read: Not used.	
$02E0,000C_{16}$	Write: Ethernet section interrupt clear.	Writing to this address clears an active Ethernet section interrupt. Data for the write is of no consequence. Clearing the interrupt turns off the interrupt signal but does not clear either of the causes of the interrupt.
	Read: Not used.	
$02E0,0010_{16}$	Write: 82596CA interrupt enable/disable	Writing a 1 to the least significant bit of this address enables the interrupt signal from the 82596CA. Writing a 0 to the least significant bit disables the interrupt. Reading from this address returns the state of the enable bit as the least significant bit. The other bits are undefined.
	Read: 82596CA interrupt enable/disable	
$02E0,0014_{16}$	Write: Abort interrupt enable/disable	Writing a 1 to the least significant bit of this address enables the 82596CA abort interrupt. Writing a 0 to the least significant bit disables the interrupt. Reading from this address returns the state of the enable bit as the least significant bit. The other bits are undefined.
	Read: Abort interrupt enable/disable	
$02E0,0018_{16}$	Write: Not used.	
	Read: 82596CA interrupt status	Reading from this address returns the state of the 82596CA interrupt signal as the least significant bit. A 1 bit indicates the interrupt is asserted; 0 indicates not asserted. The other bits are undefined.
$02E0,001C_{16}$	Write: Clear abort interrupt.	Writing to this address clears the 82596CA abort condition. The data for the write is of no consequence.
	Read: Abort interrupt status	Reading from this address returns the state of the 82596CA abort interrupt signal as the least significant bit. A 1 bit indicates the interrupt is asserted; 0 indicates not asserted. The other bits are undefined.
$02E0,0020_{16}$ - $02E0,0037_{16}$	Not used.	

Continues

TABLE 12-5 — *Continued*

Ethernet addresses

Address	Function	Notes
$02E0,0038_{16}$	Write: Hardware trigger point #1	Writing to these addresses produces a low-going pulse at one of two test points. The data are of no consequence. These addresses and test points are intended to aid debugging.
	Read: Not used.	
$02E0,003C_{16}$	Write: Hardware trigger point #2	
	Read: Not used	

12.8.1 Interrupts

The Ethernet interrupt causes a level 1 interrupt autovector to the CPU (vector 25).

The Ethernet interrupt combines interrupt conditions from two sources:

1. The interrupt signal from 82596CA controller itself.
2. The ABORT condition. The ABORT condition is generated by logic external to the 82596CA (see section 12.9.1). It is set when the 82596CA receives an exception acknowledge as the response to a bus cycle.

To cause an interrupt, an interrupt condition must be enabled. Each of the two interrupt conditions has its own enable.

Once the Ethernet interrupt is asserted, it stays asserted until the processor writes to the interrupt clear address. Likewise, each interrupt condition stays asserted until explicitly cleared or disabled by the processor.

The ABORT condition is cleared by writing to the ABORT clear address. The 82596CA interrupt is cleared by setting the appropriate acknowledge bits in the command word of the 82596CA's system control block (SCB), setting the next control commands in the command word of the SCB, and issuing a Channel Attention to the 82596CA.

The sequence of events for dealing with an Ethernet interrupt due to an ABORT condition are:

1. Enable the interrupt.

2. Assume that some time later the ABORT condition becomes asserted. Once asserted, it will stay asserted until explicitly cleared.

3. The enabled ABORT condition causes the Ethernet interrupt to be asserted.

   It will stay asserted until explicitly cleared.

4. The interrupt causes the HK68/V3D to execute the Ethernet interrupt service routine. The interrupt service routine of the processor clears the Ethernet interrupt.

5. The interrupt service routine reads the interrupt status bits to determine whether the interrupt is an ABORT condition or 82596CA interrupt signal. (This and the previous step may be interchanged.)

6. The interrupt service routine clears the ABORT condition. At this point, both the interrupt signal and the ABORT condition have been cleared.

If the second interrupt condition is enabled and occurs before the first is cleared, it will cause the interrupt signal to be asserted *only after* the first condition is cleared; that is, not after the Ethernet interrupt is cleared, but after the interrupting condition (ABORT or 82596CA interrupt) is cleared. Thus, the interrupt signal may be cleared early in an interrupt service routine knowing that it cannot be reasserted until later in the routine when the interrupting condition is cleared.

If the interrupt service routine checks the interrupt status bits and both are set, it is not possible to determine which of the two occurred first and thus which one to clear. In this case, the interrupt service routine should handle both cases and clear both conditions.

If an interrupt condition is true when it is enabled, an interrupt will occur immediately.

Disabling an interrupt source is equivalent to clearing it to the extent that it allows the other interrupt condition to generate an interrupt.

12.9 EXCEPTION CONDITIONS

HK68/V3D bus cycles may terminate abnormally in two ways: relinquish and retry, and exception. The 82596CA directly supports relinquish and retry via the $\overline{\text{BOFF}}$ (backoff) pin.

Logic on the HK68/V3D external to the 82596CA responds to an exception acknowledge by setting an ABORT condition. The ABORT condition asserts the 82596CA $\overline{\text{BOFF}}$ signal and keeps it asserted until the ABORT condition is cleared. Asserting the $\overline{\text{BOFF}}$ signal causes the 82596CA to relinquish its control of the HK68/V3D's local bus so that other bus masters may use it and keeps the 82596CA from generating any additional bus cycles until the processor intervenes.

The ABORT condition may also cause an interrupt to notify the HK68/V3D that 82596CA operation has been suspended.

When an ABORT condition occurs, there are three possible responses:

1. Simply clear the ABORT condition and let the 82596CA resume where it left off. If the exception acknowledge resulted from accessing an undefined address, the exception acknowledge will occur again.

2. Reset the 82596CA with a Port Reset command and then clear the ABORT condition. This allows the 82596CA to be reinitialized, but destroys any information about the cause of the exception acknowledge.

3. Issue a Port Dump command to the 82596CA and then clear the ABORT condition. According to Intel, a Port Dump command may be issued while $\overline{\text{BOFF}}$ is asserted and will take precedence over any transfers that were in progress when $\overline{\text{BOFF}}$ was asserted. The Port Dump command dumps the internal status of the 82596CA to memory, where it may provide some clues about what the 82596CA was doing when the exception acknowledge occurred. Following the Port Dump command, the 82596CA should probably be reset using a Port Reset command.

   Note that the Dump command will occur only when the $\overline{\text{BOFF}}$ signal is released, which is the same time that the ABORT condition is cleared.

12.10 ETHERNET JUMPER

The transmit differential signal pair for the Ethernet interface may be configured for either half- or full-step modes to facilitate its use with different types of transceivers, via configuration jumper J1.

The configuration of the jumper is briefly summarized in Table 12-6.

TABLE 12-6
Transmit differential line configuration (J1)

Position	Configuration
J1 installed	+ (positive) idle differential voltage on TX lines full-step mode (for example, for Ethernet 1.0-type transceivers)
J1 not installed	0 idle differential voltage on TX lines half-step mode (for example, for IEEE-802.3-type transceivers)

12.11 ETHERNET PORT PIN ASSIGNMENTS

Connector P4 is an Ethernet 15-pin D connector (Fig. 12-3).

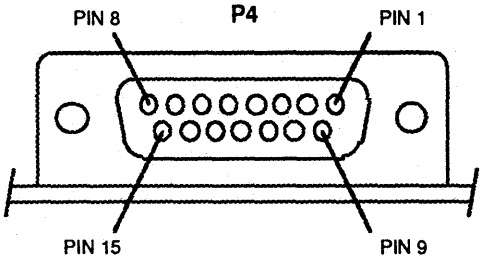


FIGURE 12-3. Ethernet connector, P4

TABLE 12-7
Ethernet connector pin assignments, P4

Pin Number	Name	Description	Direction	Transceiver Cable D Connector Pin Number
1	CLSNShld	Control In circuit Shield	In	1
2	CLSN-	Control In circuit -	In	9
3	CLSN+	Control In circuit +	In	2
4	TX-	Data Out circuit -	Out	10
5	TX+	Data Out circuit +	Out	3
6	TXShld	Transmit Shield	In	11
7	RxShld	Data In circuit Shield	In	4
8	RX-	Data In circuit -	In	12
9	RX+	Data In circuit +	In	5
10	VPLUS	Voltage Plus	Out	13
11	VCMN	Voltage Common	In	6
12	VShld	Voltage Shield	In	14
13	CTLO+	Control Out circuit +	Not connected	7
14	CTLO-	Control Out circuit -	Not connected	15
15	CTLOShld	Control Out circuit Shield	In	8

13

Optional Real-Time Clock (RTC)

13.1 INTRODUCTION

As an option, one PROM can be fitted with a special socket which has a built-in CMOS watch circuit and a lithium battery (Dallas Semiconductor, part number DS1216F). The DS1216F is a 32-pin, 600 mil-wide DIP socket that accepts any 32-pin bytewide ROM or nonvolatile RAM. The module socket is factory-installed in the first HK68/V3D PROM position (U70). The timekeeping function remains transparent to the memory device place above. The RTC monitors V_{CC} for an out-of-tolerance condition. When such a condition occurs, the battery automatically switches on to prevent loss of time and calendar data.

The timekeeping information provided by the RTC includes hundredths of seconds, seconds, minutes, hours, days, date, month, and year. The data at the end of the month is automatically adjusted for months with fewer than 31 days, including correction for leap years. The RTC operates in either 24-hour or 12-hour format with an AM/PM indicator.

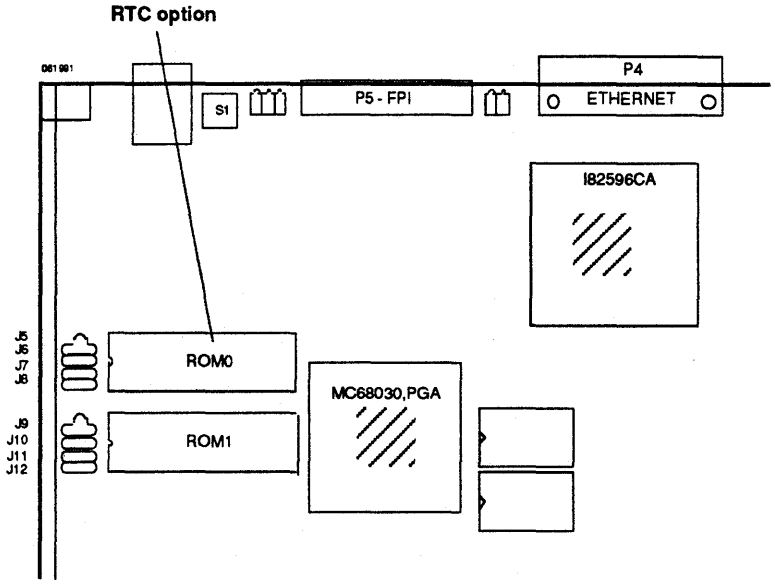


FIGURE 13-1. Real-time clock socket

The module socket can plug into the existing socket or replace it entirely. When the module socket is plugged into the existing socket the board profile is wider. The following table lists resulting board thickness values, depending on the installation method. The values include a standard PROM thickness.

TABLE 13-1
Effect of RTC intallation on board height

Configuration	Component Height Above Board	Minimum Board Spacing
RTC module plugged into existing ROM socket:	.75 in.	.85 in. (2 slots)

Only one card slot is required if the board is in the end slot. The RTC logic does not generate interrupts; a CIO timer channel is still used for that purpose. The RTC contents, however, may be used to check for long-term drift of the HK68/VE system clock, and as an absolute time and date reference after a power failure. Leap year accounting is included. Heurikon can provide complete operating system software support for the RTC module.

The RTC module time resolution is 10 milliseconds. The RTC internal oscillator is accurate to one minute per month, at 25 degrees C.

13.2 READING AND SETTING THE RTC

The clock contents are set or read using a special sequence of ROM read commands, as detailed in the program example, below. The RTC module "monitors" ROM accesses and, if a certain sequence of 64 ROM addresses occur, takes temporary control of the ROM space, allowing data to be read from or written to the module. Writing is done by twiddling an address line, which the module uses as a data input bit. There are never any MPU write cycles directed to the PROM space.

Note: Do not execute the module access instructions out of ROM. The instruction fetch cycles will interfere with the module access sequence. Also, be certain the reset disable bit (rtc_data.day bit D4) is always written as a "1".

EXAMPLE 13-1. Real-Time Clock Software

```
#define WATCHBASE (unsigned char *)0x00000000 /* ROM socket */
#define WR0_WATCH (unsigned char *)(WATCHBASE+2) /* write 0 */
#define WR1_WATCH (unsigned char *)(WATCHBASE+3) /* write 1 */
#define RD_WATCH (unsigned char *)(WATCHBASE+4) /* read */
struct rtc_data {    /* D7 D6 D5 D4  D3 D2 D1 D0  range */
  unsigned char dotsec;/*-0.1 sec-:-0.01 sec-;00-99 */
  unsigned char sec;  /* --10 sec-:-seconds-; 00-59 */
  unsigned char min;  /* --10 min-:-minutes-; 00-59 */
  unsigned char hour; /*A 0 B Hr:-hours-; 00-23 */
  unsigned char day;  /*0 0 0 1:-day--; 01-07 */
  unsigned char date; /*-10 date-:-date-; 01-31 */
  unsigned char month; /*-10 month-:-month-; 01-12 */
  unsigned char year; /*-10 year--: --year----- ; 00-99 */
}; /* "A" = "0" for 00-23 hour mode, "1" for 01-12 hour mode */
 /* "B" = MSB of the 10 hours value (if 00-23 hour mode) else
     = "0" for PM or "1" for AM (if 01-12 hour mode) */

rtc_wr(data)      /* set the real-time clock */
register unsigned char *data; /* rtc_data pointer */
{
  register int i, bit;
  unsigned char temp;
  static unsigned char key[] = { /* the unlock pattern */
    0xC5, 0x3A, 0xA3, 0x5C, 0xC5, 0x3A, 0xA3, 0x5C };

  if ( data ) {
    rtc_wr(0);   /* send key pattern */
  } else { /* this is the unlock function */
    i = *RD_WATCH; /* reset */
    data = key;
  }
  for( i=0; i<8; data++, i++ )
    for( bit = 1; bit & 0xff; bit <<= 1 )
      temp = ( *data & bit ) ? *WR1_WATCH : *WR0_WATCH;
}

rtc_rd(data)      /* read the real-time clock */
register unsigned char *data; /* rtc_data pointer */
{
  register int i, bit;

  rtc_wr(0); /* send key pattern */
  for( i=0; i<8; data++, i++ ) {
    *data = 0;
    for( bit = 1; bit & 0xff; bit <<= 1 )
      *data |= (*RD_WATCH & 1) ? bit : 0 ;
  }
}
```

13.3 PIN ASSIGNMENTS

The DS1216F uses pins 1, 10, 12, 13, 22, and 24. All pins pass through to the socket receptacle except pin 22 (CE/), which is inhibited during the transfer of time information.

TABLE 13-2
Pin assignments, real-time clock

32-pin RTC Pin Number	Name	Function
1	RST\	RESET
10	A2	Address Bit 2 (READ/WRITE\)
12	A0	Address Bit 0 (Data Input)
13	DQ0	I/O_0 (Data Output)
16	GND	Ground
22	CE\	Conditioned Chip Enable
24	OE\	Output Enable
32	VCC	+5 VDC to the socket

13.4 RTC OPERATION

A highly structured sequence of 64 cycles is used to gain access to time information and temporarily disconnects the mated memory from the system bus. Information transfer into and out of the RTC is achieved by using address bits A0 and A2, control signals OE\ and CE\, and data I/O line DQ0. All RTC data transfers are accomplished by executing read cycles to the mated memory address space. Write and read functions are determined by the level of address bit A2. When address bit A2 is low, a write cycle is enabled, and data must be input on address bit A0. When address bit A2 is high, a read cycle is enabled, and data is output on data I/O line DQ0. Either control signal (OE\ or CE\) must transition low to begin and high to end memory cycles that are directed to the RTC; however, both control signals must be in an active state during a memory cycle.

Communication with the RTC is established by pattern recognition of a serial bit stream of 64 bits, which must be matched by executing 64 consecutive write cycles, placing address bit A2 low with the proper data on address bit A0.

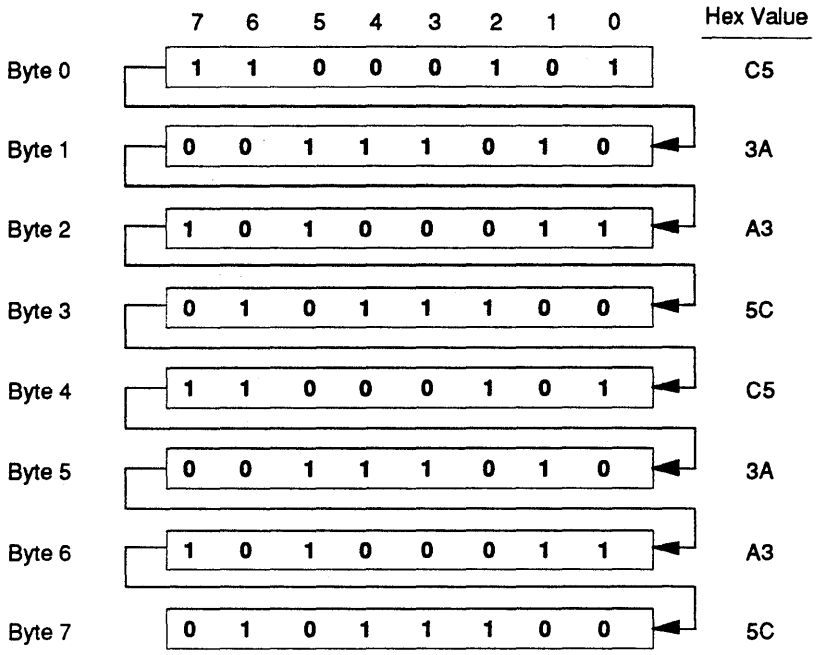


FIGURE 13-2. RTC comparison register definition

The 64 write cycles are used only to gain access to the RTC. Prior to executing the first of 64 write cycles, a read cycle should be executed by holding A2 high. The read cycle will reset the comparison register pointer within the RTC, ensuring that pattern recognition starts with the first bit of the sequence. When the first write cycle is executed, it is compared with bit 0 of the 64-bit comparison register. If a match if found, the pointer increments to the next location of the comparison register and awaits the next write cycle. If a match is not found, the pointer does not advance and all subsequent write cycles are ignored. If a read cycle occurs at any time during pattern recognition, the current sequence is aborted and the comparison register pointer is reset. Pattern recognition continues for a total of 64 write cycles, as described above, until all the bits in the comparison register have been matched (this bit pattern is shown in Figure 13-2).

With a correct match for 64 bits, the RTC is enabled and data transfer to or from the timekeeping registers may proceed. The next 64 cycles will cause the RTC to either receive data on Data In (A0) or transmit data on Data Out (DQ0), depending on the level of READ/WRITE\ (A2). Cycles to other locations outside the memory block can be interleaved with CE\ and OE\ cycles without interrupting the pattern recognition sequence or data transfer sequence to the RTC.

An unconditional reset to the RTC occurs by either bringing up A14 (RESET\) low if enabled, or on power-up. The RESET\ can

occur during pattern recognition or while accessing the the RTC registers. RESET\ causes access to abort and forces the comparison register pointer back to bit 0 without changing registers.

13.5 NONVOLATILE CONTROLLER OPERATION

The RTC performs circuit functions required to make the timekeeping function nonvolatile. First, a switch is provided to direct power from the battery or VCC supply, depending on which voltage is greater. The second function provides power-fail detection. Power-fail detection typically occurs at 4.25 volts. Finally, the nonvolatile controller protects the RTC register contents by ignoring any inputs after power-fail detection has occurred. Power-fail detection also has the same effect on data transfer as the RESET\ input.

13.6 RTC REGISTERS

The RTC information is contained in eight registers, each containing eight bits. The registers are accessed in sequence, one bit at a time, after the 64-bit pattern recognition sequence has been completed. When updating the RTC registers, each must be handled in groups of eight bits. Writing and reading individual bits within a register could produce erroneous results. These read/write registers are defined in Figure 13-3.

Data contained in the RTC registers is in BCD (binary coded decimal) format. Reading and writing the registers is always accomplished by stepping through all eight registers, starting with bit 0 of register 0 and ending with bit 7 of register 7.

13.7 AM-PM/12/24 MODE

Bit 7 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20-23 hours).

13.8 OSCILLATOR AND RESET BITS

Bits 4 and 5 of the day register are used to control the RESET\ and oscillator functions. Bit 4 controls the RESET\ (pin 1). When the RESET\ bit is set to logic 1, the RESET input pin is ignored. When the RESET\ bit is set to logic 0, a low input on the RESET\ pin will cause the RTC to abort data transfer without changing data in the watch registers. Bit 5 controls the oscillator. When set to logic 1, the oscillator is turned off. When set to logic 0, the

oscillator turns on and the watch becomes operational. Both bits are set to a logic 1 when shipped from the factory.

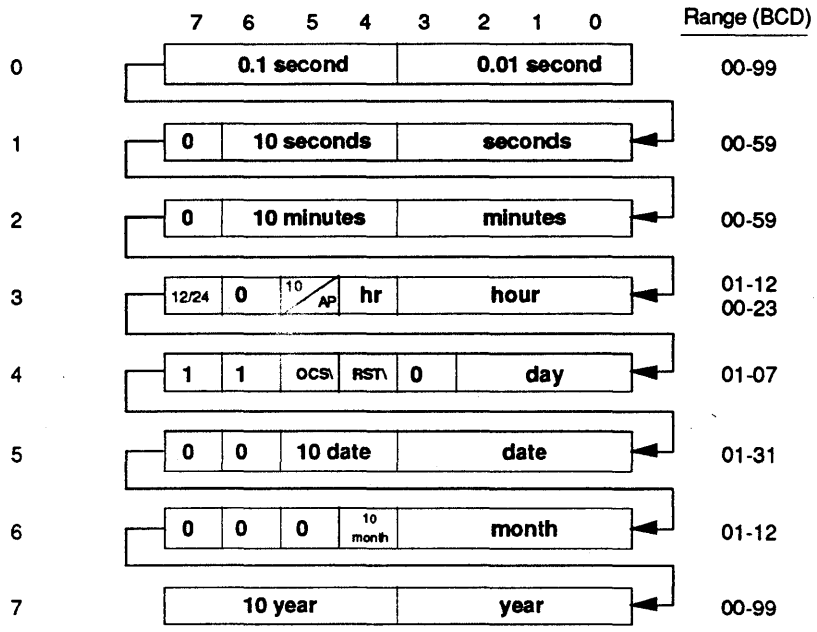


FIGURE 13-3. RTC register definition

13.9 ZERO BITS

Registers 1, 2, 3, 4, 5, and 6 contain one or more bits that will always read logic 0. When writing these locations, either a logic 1 or 0 is acceptable.

14

Hardware Summary

14.1 SOFTWARE INITIALIZATION SUMMARY

This section outlines the steps for initializing the facilities on the HK68/V3D board. Certain steps must be performed in sequence, while others may be rearranged or omitted entirely, depending on your application.

1. The MPU automatically fetches the reset vector following a system reset and loads the supervisor stack pointer and program counter. The reset vector is in the first 8 bytes of ROM.
2. Recall the NVRAM contents. (Reference: section 6.8)
3. Determine RAM configuration. (Reference: section 6.4)
4. Set the bus control latch. (Reference: section 7.8)
5. Clear on-card RAM to prevent parity errors due to uninitialized memory reads. (Reference: section 5.1)
6. Load the 68030 Vector Base Register with the location of your exception vector table (usually at the start of RAM).
7. Initialize the exception vector table in RAM (at the selected base address.) This step links the various exception and interrupt sources with the appropriate service routines. (Reference: section 3.3)
8. Initialize the CIO. (Reference: section 9.7)
9. Initialize the serial ports. (Reference: section 10.5)
10. Initialize the SCSI port. (Reference: section 11)
11. Initialize the Ethernet port. (Reference: section 12)
12. Initialize the 7-segment display (Reference: section 8.1)
13. Release the VMEbus SYSFAIL line. (Reference: section 7.6)
14. Initialize off-card memory and I/O devices, as necessary.
15. Enable system interrupts, as desired. (Reference: section 3.2)

14.2 ON-CARD I/O ADDRESSES

This section is a summary of the on-card port addresses. It is intended as a general reference for finding additional information about a particular device. Refer to section 6.6 for a pictorial description of the system memory map.

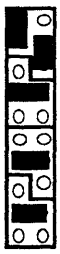
TABLE 14-1
Address summary

Address	Type	Device	HK68/V3D User's Manual Section
$4xxx,xxxx_{16}$	R/W	VMEbus (Extended Address Mode)	7.7
$04xx,xxxx_{16}$	R/W	VMEbus	7.7, 7.9
$03xx,xxxx_{16}$	R/W	HK68/V3D on-card RAM	6.3
$02F0,000x_{16}$	R/W	SCC1 (Ports A & B)	10
$02E0,000x_{16}$	R/W	Ethernet	12
$02D0,000x_{16}$	R/W	CIO	9
$02C0,0000_{16}$	W	Mailbox Base Address	7.8
$02B0,0070_{16}$	R/W	Display segment g	8
$02B0,0060_{16}$	R/W	Display segment f	8
$02B0,0050_{16}$	R/W	Display segment e	8
$02B0,0040_{16}$	R/W	Display segment d	8
$02B0,0030_{16}$	R/W	Display segment c	8
$02B0,0020_{16}$	R/W	Display segment b	8
$02B0,0010_{16}$	R/W	Display segment a	8
$02B0,000E_{16}$	W	VMEbus Bus Timer	7.X
$02B0,000C_{16}$	W	VMEbus Slave Enable	7.4
$02B0,000A_{16}$	W	On-card Watchdog Enable	
$02B0,0008_{16}$	W	SCSI Interrupt Mask	11
$02B0,0006_{16}$	W	SCSI Reset	11
$02B0,0004_{16}$	W	Mailbox Enable	7.8
$02B0,0002_{16}$	W	MPU Cache Disable	3.6
$02A0,0000_{16}$	W	Bus Control Latch	7.4
$0290,000x_{16}$	W	VMEbus Interrupt Request	7.5
$0270,0000_{16}$	R	NV-RAM Recall	6.8
$0260,0000_{16}$	W	NV-RAM Store (tas)	6.8
$0250,00xx_{16}$	R/W	NV-RAM Data	6.8
$0230,000x_{16}$	R/W	SCSI	11
$0240,0000_{16}$		unused	
$01xx,xxxx_{16}$	R/W	VMEbus (Standard Space)	7.7
$00C0,xxxx_{16}$	R/W	VMEbus (Short Space)	7.7
$0080,000x_{16}$	R	VMEbus Interrupt Vectors	
$0040,0000_{16}$	R	ROM1	6.2
$0000,0000_{16}$	R	ROM0	6.2

14.3 HARDWARE CONFIGURATION JUMPERS

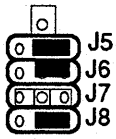
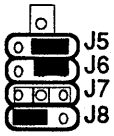
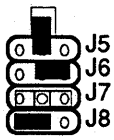
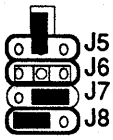
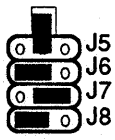
Jumper settings are detailed in the manual section pertaining to the associated device. This section can be used as a cross reference for finding additional information about the jumpers.

TABLE 14-2
Standard jumper settings

Jumper	Standard Configuration	Options	Function	HK68/V3D Manual Section
J1	Installed	J1 installed: + (positive) idle differential voltage on TX lines, full-step mode (for example, for Ethernet 1.0-type transceivers) J1 removed: 0 idle differential voltage on TX lines, half-step mode (for example, for IEEE-802.3-type transceivers)	Selects Ethernet differential voltage	12
J2	J2:1-2 False 	J2:1-2 False (+12V) J2:2-3 True (-12V)	RS-232 handshaking defaults	10
J3	J3:1-2 Ring Indicator 	J3:1-2 Ring Indicator J3:2-3 Data Carrier Detect	Selects Ring Indicator or Data Carrier Detect for SCC Port A.	10
J5–J8	Matches ROM0 size. See Table 14-3.	2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27513 paged, 2864 R/W EEPROM, 2817 R/W EEPROM	Selects ROM 0 size (default is 2764)	5
J9-J12	Matches ROM1 size. See Table 14-3.	2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27513 paged, 2864 R/W EEPROM, 2817 R/W EEPROM	Selects ROM 1 size (default is 2764)	5
J14, J15, J17, J18	Bus Grant Level 3 	Bus Grant Level 3 Bus Grant Level 2 Bus Grant Level 1 Bus Grant Level 0	Selects VMEbus Bus Grant level	7
J16	Bus Request Level 3) 	Bus Request Level 3 Bus Request Level 2 Bus Request Level 1 Bus Request Level 0	VMEbus arbitration (bus request level 3, not system controller)	7

J19	J19:1-2 input from VMEbus 	J19:1-2 input from VMEbus J19:2-3 output to VMEbus	Enables VMEbus SYSRESET*	7
J20	Removed	J20 installed: Allows HK68/V3D to respond to ACFAIL* interrupt. J21 removed: HK68/V3D does not respond to ACFAIL* interrupt.	ACFAIL* connects to VMEbus	7
J21-J24	Matches memory size.	1, 2, 4, 8, or 16 megabytes	VMEbus slave window size	7
J25	Removed	J25 installed: drives SYSCLK J25 removed: does not drive SYSCLK	Disables SYSCLK	7
J26	Removed	J26 installed: HK68/V3D can drive BCLR*. J26 removed: HK68/V3D cannot drive BCLR*.	Disables BCLR*.	7
J91		Factory set for memory configuration. Do not alter.		
J92		Factory set for memory configuration. Do not alter.		

TABLE 14-3
ROM size options

ROM Type	ROM Capacity	Jumper Configuration
2764 27128 27513 paged	64 Kbits (8K × 8) 128 Kbits (16K × 8)	
27256	256 Kbits (32K × 8)	
27512	512 Kbits (64K × 8)	
27010	1 Mbits (128K × 8)	
27020 27040	2 Mbits (256K × 8) 4 Mbits (512K × 8)	

27080	8 Mbits (1M × 8)	J5 or J9 J6 or J10 J7 or J11 J8 or J12
2864 R/W EEPROM	8K × 8	J5 or J9 (any setting) J6 or J10 J7 or J11 (either A or B) J8 or J12
2817 R/W EEPROM	2K × 8	J5 or J9 J6 or J10 J7 or J11 (either A or B) J8 or J12

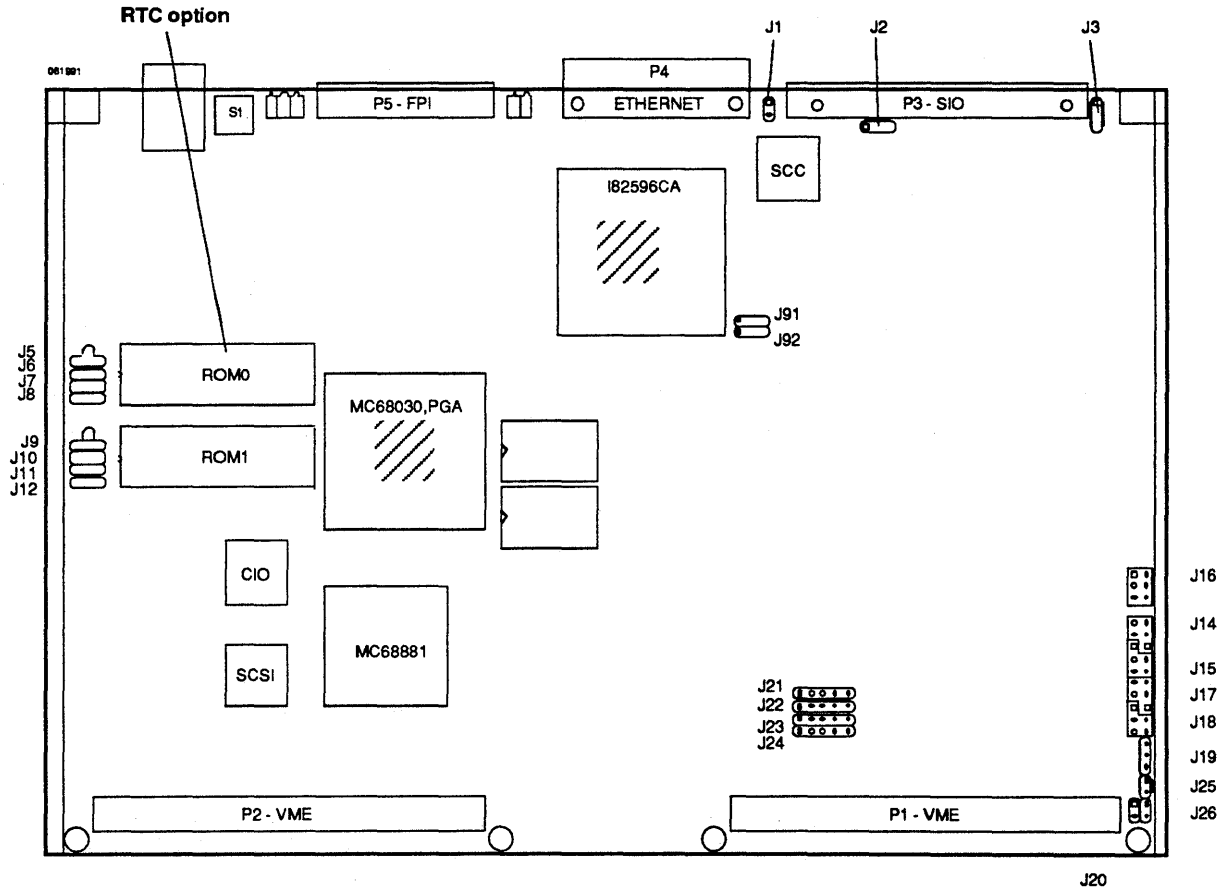


Figure 14-1. Jumper locations

14.4 POWER REQUIREMENTS

TABLE 14-4
Power requirements

Voltage	Current	Usage
+5	7.0A, max	All logic
+12	20ma, max	RS-232 interface
-12	20ma, max	RS-232 interface

The "+5" and "Gnd" pins on P2 *must* be connected for proper operation.

14.5 ENVIRONMENTAL

Operating temperature: 0 to +55 degrees Centigrade, ambient, at board.
Humidity: 0% to 85%.
Storage temperature: -40 to +70 degrees C.
Power dissipation is about 35 watts.

Fan cooling is required if the HK68/V3D board is placed in an enclosure or card rack.

Fan cooling is also recommended when using an extender board for more than a few minutes.

14.6 MECHANICAL SPECIFICATIONS

TABLE 14-5
Mechanical specifications

Width	Depth	Height (above board)
9.187 in.	6.299 in.	0.6 in. (0.8 in.)
233.35 mm	160 mm	15.25 mm (20.35 mm)

If the real-time clock (RTC) option is installed, see Table 13-1 for information on the effect of the RTC on board height.

Standard board spacing is 0.8 inches. The HK68/V3D is a 10-layer board.

Appendix A

The HK68/V3D Monitor

This appendix includes an introduction to monitor operation, instructions for command sequences that configure the HK68/V3D, a command reference, and a function reference.

INTRODUCTION

The monitor consists of a set of about 150 C functions. A subset of these functions constitute the monitor commands, which are parsed into function calls. The monitor commands have been designed to provide easy-to-use tools for (1) HK68/V3D configuration at power-up or reset, and (2) communications, downloads, program tracing, and other common uses. A command line editor and history have been included to reduce the need to retype commands. The monitor uses nonvolatile memory to store all values.

USING NONVOLATILE MEMORY TO CONFIGURE THE HK68/V3D

nvdisplay

The **nvdisplay** command allows you to access almost all of the hardware registers on the HK68/V3D by editing fields that contain configuration values. The fields have been collected into the main groups shown below. Each field can be edited from the display.

Group	Fields
Console	Port, Baud, Parity, Data, StopBits, XOnXOff, ChBaudOnBreak, RstOnBreak
Download	Port, Baud, Parity, Data, StopBits, XOnXOff, ChBaudOnBreak, RstOnBreak
VmeBus	ExtSlaveMap, StdSlaveMap, AddrModSel, Replace Addr, EnblSlave, MastRelModes, SlaveRelOnReq, LocalBusTimer, VmeBusTimer, Sysfail, IndivRMC
Mailbox	ShtSlaveMap, EnblSht
Cache	InstrCache, DataCache
Misc	PowerUpMemClr, ClrMemOnReset, PowerUpDiag, CountValue
BootParams	BootDev, LoadAddress, RomBase, RomSize, DevType, DevNumber, ClrMemOnBoot

Three other groups — HardwareConfig, Manufacturing, and Service — are reserved for use by Heurikon manufacturing and are read only.

Once fields have been edited, the new field values can be saved to nonvolatile memory with the **nvupdate** command.

The nonvolatile memory configuration information is used to completely configure the HK68/V3D at reset. The **configboard** command can also be used to reconfigure the board after modifications to the nonvolatile memory.

COMMAND SUMMARY

Additional commands for a wide range of uses are summarized below. If you need additional assistance with the monitor, please call a Heurikon customer support representative at 1-800-327-1251.

Access documentation for the HK68/V3D:

help
help editor
help functions
help memmap

For details on a specific command, type **help** and any command name listed in this summary.

Initialize, display, or change the contents of Heurikon-defined and user-defined memory fields in nonvolatile memory:

nvdisplay
nvinit
nvopen
nvset
nvupdate

Download and execute an application program from a host:

call
download
transmode

Test local and external memory boards:

testmem

Display, copy, or modify data:

checksummem
clearmem
cmpmem
copymem
displaymem
fillmem
findmem
findnotmem
findstr
readmem
setmem
swapmem
writemem
writestr

Load and execute a program or operating system from a boot device:

bootbus
bootrom
bootserial

Display, trace, or execute application programs:

disassemble
dumpregs
exectrace
settrace
step

Display Ethernet ID or check whether the HK68/V3D is VMEbus system controller:

prstatus

Control accessibility of the HK68/V3D in VMEbus short, standard or extended space:

slavedis
slaveenable

Enable, disable, or set up bus interfaces and devices:

configboard
date
setdate
starttimer
stoptimer

Add, subtract, multiply, or divide two numbers:

add
div
mul
rand
sub

FUNCTIONS

The functions described in the function reference can be called directly from the command line, but no argument checking will take place. It is advisable instead to use the monitor commands whenever possible.

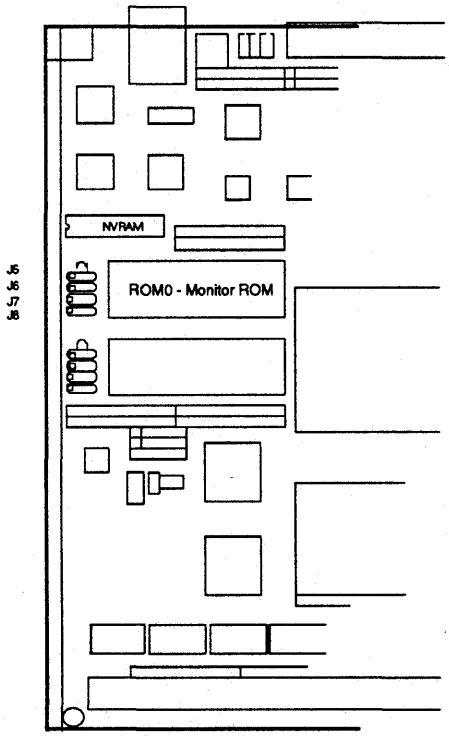
EEPROM CONFIGURATION MEMORY

The monitor uses an 128-byte EEPROM for nonvolatile memory. A general description of the organization of nonvolatile memory is given in the "On-card Memory Configuration" section (section 5) earlier in this manual. A portion of nonvolatile memory is reserved for the monitor and is read-only. All other memory areas of nonvolatile memory are both read-accessible and write-accessible for other uses.

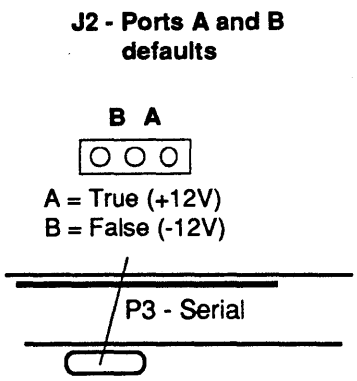
The start address, size, and description of the monitor EEPROM are shown below:

EEPROM addresses

Device Address	Byte Offsets	Data
$0270,0000_{16}$	$0 - 15FF_{16}$	User-defined data area
$0270,B000_{16}$	$1600_{16} - 17FF_{16}$	Monitor/board initialization
$0270,C000_{16}$	$1800_{16} - 1FFF_{16}$	Manufacturing/service hardware information (write protected)



MONITOR INSTALLATION AND SETUP



Be sure the ROM size jumpers J5–J8 are configured to match the size of ROM on the board (settings are described in the section "On-card Memory Configuration," section 5). The serial cable should connect the terminal with port B on the HK68/V3D. Terminal settings should be 9600 baud, 8 data bits, 1 stop bit, no parity.

If no console device is connected to serial port B, be sure the RS-232 default jumper J2 is set as J2:2–3, which sets the port to true (+12V). Otherwise, the monitor will hang while it waits for the serial chip to transmit the start-up message. The "Setup and Installation" section (section 2) describes full installation instructions for the HK68/V3D.

RESET SEQUENCE

At power-up or a board reset, ROM-based power-up diagnostics check the serial port and memory. A function called StartMonitor performs hardware initialization, autoboot procedures, free memory initialization, and, if necessary, initializes the monitor to bring up the command line editor.

The processor stacks and configuration are initialized before StartMonitor is called.

StartMonitor does the following:

1. Initializes the nonvolatile memory configuration structures to their default state.

2. The minimum set of hardware initialization is completed on the basis of the nonvolatile memory configuration structures. This usually includes a reset of devices to a known state.

3. After initialization, the monitor tries to read the current nonvolatile memory configuration from the nonvolatile device.

Invalid configuration information

If the configuration information is invalid, a warning message appears:

```
Warning protected region cannot be initialized.
```

The board is fully configured using the default nonvolatile configuration.

Valid configuration information

If the configuration information is valid, a countdown to the autoboot begins.

If you allow the countdown to finish, autoboot begins and the board is fully configured according to the current nonvolatile device configuration. If the auto-boot portion of the configuration requires auto-boot, the correct device is opened and booted. If no auto-boot is necessary, then the board logo is printed, memory is initialized, and the line editor is started.

If you cancel configuration before the autoboot begins, the board is configured with the default nonvolatile configuration, which is summarized below.

Console defaults

Port B, 9600 baud, no parity, 8 data bits, 2 stop bits, XOn/XOff protocol on, no reset or baud change on break.

Download defaults

Port A, 9600 baud, no parity, 8 data bits, 2 stop bits, XOn/XOff protocol on, no reset or baud change on break.

VMEbus defaults

Slave extended space mapped to $8000,0000_{16}$.
Slave standard space mapped to 000000_{16}.
Address modifier select is "ExAll".
Slave standard space replacement address is $0000,0000_{16}$.
Slave is enabled.
Master release mode is release-on-request.
Slave release-on-request is enabled.
The on-card bus timer is 32 microseconds.
The VMEbus bus timer is 64 microseconds.
SYSFAIL is off.
Indivisible read-modify-write cycles are disabled.

Mailbox defaults

Slave short space mapped to FFF8, slave short space disabled.

Cache defaults

Instruction cache is on. Data cache is off.

Miscellaneous defaults

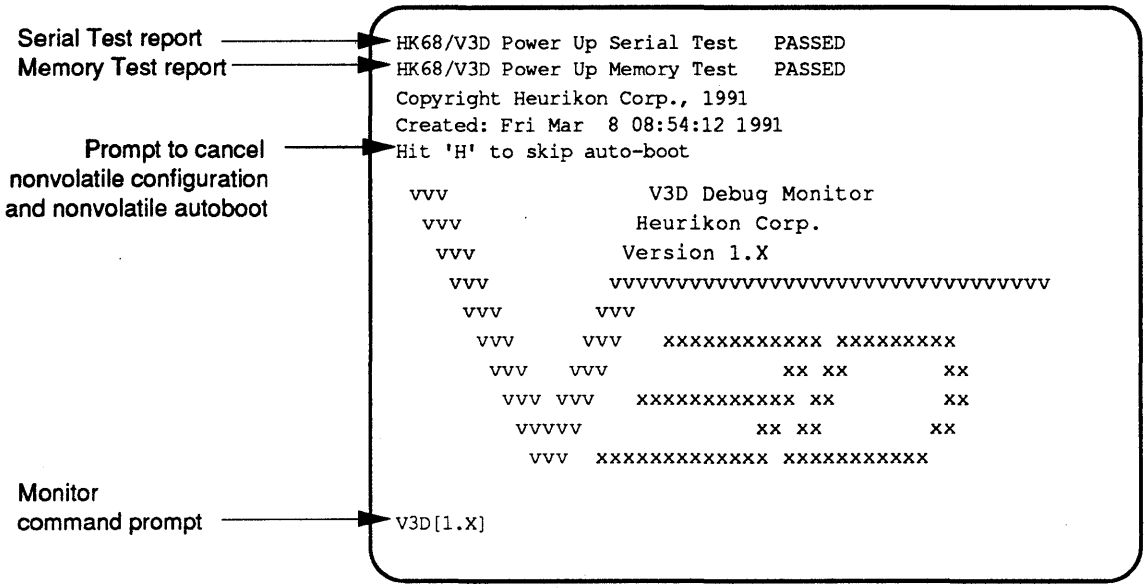
Clear memory on power-up, clear memory on reset, autoboot countdown set to longest value (7).

BootParams defaults

No boot device specified, load address is 03010000_{16}, ROM base is at 00400000_{16}, ROM size is 00020000_{16}, device type and number are 0, and memory is not cleared at boot-up.

START-UP DISPLAY

At power-up or after a reset, the monitor runs diagnostics, reports the results, displays the name of the board, and then displays a prompt for commands.



Serial Test and Memory Test reports

The results of the self-diagnostic tests are displayed at power-up or after a reset. If the Memory Test fails, the display will show a DBug prompt instead of the usual monitor command prompt. A failed Memory Test could indicate a hardware malfunction that should be reported to our factory service department, 1-800-327-1251.

Nonvolatile Configuration and Nonvolatile Autoboot

At power-up and reset, the monitor configures the board according to the contents of nonvolatile configuration memory. If the configuration indicates that an autoboot device has been selected, the monitor attempts to load an application program from the specified device.

You can cancel both the nonvolatile configuration sequence and the autoboot sequence by pressing the **H** key on the console keyboard before the boot ends. The monitor is then in a "manual" mode from which you can execute commands and call functions. The monitor also enters manual mode if the autoboot fails. Instructions for downloading and executing remote programs are given in the command reference and function reference.

Monitor Command Prompt

The monitor provides a command line interface that includes a command history and a *vi*-like line editor. The command line interface has two modes: Entry mode and Command mode. In Entry mode, you can type text on the command line. In Command mode, you can move the cursor along the command line and modify commands. Each new line is brought up in Entry mode.

COMMAND-LINE HISTORY

The monitor maintains a history of up to 50 command lines for reuse. Press the ESC key from the command line to access the history.

k or - Move backward in the command history to access a previous command.

j or + Move forward in the command history to access a subsequent command.

COMMAND-LINE EDITOR

The command line editor uses typical UNIX® *vi* editing commands.

help editor To access an on-line description of the editor, type **help editor** or **h editor**.

<ESC> To exit Entry mode and start the editor, press <ESC>. You can use most common *vi* commands, such as **x, i, a, A, $, 0, w, cw, dw, r,** and **e.**

<cr> To execute the current command and exit the editor, press Enter or Return.

<DEL> To discard an entire line and create a new command line, press <DEL> at any time.

Editing Commands

a or **A** Append text on the command line.
i or **I** Insert text on the command line.
x or **X** Delete a single character.
r Replace a single character.
c Change. Use additional commands with **c** to change words or groups of words, as shown below.
cw or **cW** Change a word after the cursor (capital W ignores punctuation).
ce or **cE** Change text to the end of a word. (capital E ignores punctuation).
cb or **cB** Change the word before the cursor (capital B ignores punctuation).
c$ Change text from the cursor to the end of the line.
d Delete. Use additional commands with **d** to delete words or groups of words, as shown below.
dw or **dW** Delete a word after the cursor (capital W ignores punctuation).
de or **dE** Delete to the end of a word (capital E ignores punctuation).
db or **dB** Delete the word before the cursor (capital B ignores punctuation).
d$ Delete text from the cursor to the end of the line.

MONITOR COMMANDS

Command Syntax

There is no distinction between upper case and lower case. Press Enter or Return to end each command with a carriage return <cr>.

Each command may be typed with the shortest number of characters that uniquely identify the command. For example, you can type **nvdisp** instead of **nvdisplay,** or **disa** instead of **disassemble**. Note, however, that abbreviated command names cannot be used with on-line help; you must type **help** and the full command name.

Arguments to commands must be separated by spaces.

Command Format

The command line accepts three input formats: string, numeric, and symbolic.

Monitor commands that expect numeric arguments assume a default numeric base for each argument. The expected arguments and the default numeric bases are described in the command reference.

Specifying the base

The numeric base can be specified by entering a colon (:) followed by the base. Several examples are provided below.

`1234ABCD:16`	hexadecimal
`123456789:10`	decimal
`1234567:8`	octal
`101010:2`	binary

The default numeric base for functions is hexadecimal. Some commands use a different default base.

Put string arguments in double quotes.

String arguments must start and end with double quotation marks ("). For example, typing the argument "Foo" would result in a string argument with the value Foo, which is passed to the command.

Put character arguments in single quotes.

A character argument is a single character that begins and ends with a single quotation mark ('). The argument 'A' would result in the character *A* being passed to the command.

Start flags with a hyphen.

A flag argument is a single character that begins with a hyphen (-). For example, the flag arguments **-b, -w** or **-l** could be used for a byte, word or long flag.

There is a symbol entry for every function and command defined in the monitor. Each command must begin with a symbol. While all functions of the monitor can be executed, only those supported by the monitor as commands type-check and validate the arguments.

Commands that are not symbolic are assumed to be numeric, and the hexadecimal, decimal, and character value of the number is printed.

MONITOR FUNCTIONS

No argument checking will take place for functions that are called directly from the command line. It is advisable instead to use the monitor commands whenever possible.

The functions require spaces between the function name and its arguments. No parentheses or other punctuation is necessary.

EXAMPLES

```
UnMaskInts 1

ConnectHandler 0xf8 0x1000
```

Using the commands

This section includes instructions for common uses of the monitor. Full descriptions of the commands and functions are in the reference section.

INITIALIZING MEMORY

The monitor uses the area between $0000,0000_{16}$ and $0001,0000_{16}$ for stack and uninitialized-data space. Any *writes* to that area can cause unpredictable operation of the monitor. The monitor initializes all local memory on power-up and on reset, depending on the configuration of nonvolatile memory. The monitor initializes this area (that is, writes to it) to prevent parity errors, but it is left up to the programmer to initialize any other memory areas that are accessed, such as off-card or module memory.

CHANGING BOARD CONFIGURATION

The **nvdisplay** command shows the groups and fields in nonvolatile memory configuration that are used to configure the board. You can modify the groups and fields that are shown when you use **nvdisplay**. Then use **nvupdate** to save the new values. If you decide *not* to save your changes, type **nvopen** to re-read the previous values.

EXAMPLE

1. At the monitor prompt, type:

```
nvdisplay
```

2. Press <cr> until the group you want to modify is displayed. An example for the group "Console" is shown below.

```
Group 'Console'
    Port          A          (A, B, C, D)
    Baud          9600
    Parity        None       (Even, Odd, None, Force)
    Data          8-bits     (5-Bits, 6-Bits, 7-Bits, 8-Bits)
    StopBits      2-bits     (1-Bit, 2-Bits)
    XOnXOff       On         (Off, On)
    ChBaudOnBreak  False     (False, True)
    RstOnBreak     False     (False, True)

[SP, CR to continue] or [E, e to Edit]
```

3. Press E to edit the group.
4. Press <cr> until the field you want to change is displayed.

5. Type a new value. For most fields, legal options are displayed in parentheses.
6. Press ESC or Q to quit the display.
7. Type **nvupdate** to save the new value or **nvopen** to cancel the change by reading the old value.

EXAMPLE

The default configuration for the VMEbus SYSFAIL* signal is to turn on at boot-up. In this example, **nvdisplay** and **nvupdate** are used to turn off the SYSFAIL* signal when the system boots and the HK68/V3D is not system controller.

1. At the monitor prompt, type:

```
nvdisplay
```

2. Press <cr> until the "VmeBus" group is displayed.

```
Group 'VmeBus'
  ExtSlaveMap   0x80000000
  StdSlaveMap   0x200000
  AddrModSel    ExAll       (None, StAll, ExDat, StDat, ExSuDat, ExAll)
  ReplaceAddr   0x0
  EnblSlave     True        (False, True)
  MastRelModes  OnRequest   (WhenDone, OnRequest, OnClear, Never)
  SlaveRelOnReq On          (Off, On)
  LocalBusTimer 32u         (4us, 16u, 32u, 64u, 128u, 256u, 512u, Off)
  VmeBusTimer   64u         (4us, 16u, 32u, 64u, 128u, 256u, 512u, Off)
  Sysfail       Off         (Off, On)
  IndivRMC      Off         (Off, On)

[SP, CR to continue] or [E, e to Edit]
```

3. Press E to edit the group.
4. Press <cr> until the "Sysfail" field is displayed.
5. Type the new value "Off".
6. Press ESC or Q to quit the monitor.
7. Type the monitor command **nvupdate** to save the new value to nonvolatile memory.

ATTEMPTING TO CHANGE PROTECTED FIELDS

Some of the Heurikon-defined groups shown with **nvdisplay,** namely, Hardware, Manufacturing, and Service, are write-protected. Attempts to modify these fields result in the display of an error message:

```
Warning, protected region was not modified.
```

If you see this message, either re-read the nonvolatile memory defaults for these protected regions by typing the **nvopen** command, or return any fields you tried to edit to their original values.

READING AND WRITING MEMORY

Use **readmem** or **displaymem** to read memory, and **writemem** or **setmem** to write memory.

Required flags

readmem, writemem and **setmem** require one of the following flags, which determine the data size:

- **-b** indicates the data is in bytes.
- **-w** indicates the data is in words.
- **-l** indicates the data is in long words.

Number bases

All arguments default to hexadecimal. Specify other bases by typing a colon (:) and the base after the value.

For example, type 52:10 for decimal 52.

`displaymem` *startaddr lines*

displays *lines* of memory starting at *startaddr*. If the *lines* argument is not specified, 16 lines are displayed. After you type this command, pressing <cr> displays the next block of memory. Access size is bytes.

`readmem -[b,w,l]` *address*

reads a memory location specified by *address*. This command displays the data in hexadecimal, decimal, octal, binary, or string format.

`setmem -[b,w,l]` *address*

allows memory locations to be modified starting at *address*. **setmem** first displays the value that was read. Then you can type new data for the value. If you press <cr> after the data, the address counts up. If you press <ESC> after the data, the address counts down.

`writemem -[b,w,l]` *address value*

writes *value* to a memory location specified by *address*.

CONFIGURING THE DEFAULT BOOT DEVICE

The default boot device is defined in the nonvolatile memory group "BootParams", in the field "BootDev". When the HK68/V3D is reset or powered up, the monitor checks this field and attempts to boot from the specified device.

Currently, the monitor supports Serial, ROM, and Bus as standard for all boards. If you edit the "BootDev" field and define a device that is unsupported on your board, the monitor will display the message:

```
Unknown boot device
```

Defining "BootDev" as "Serial" calls the function *BootSerial*, defining "BootDev" as "ROM" calls the function *BootROM*, and defining "BootDev" as "Bus" calls the function *BootBus*. See the function reference for details on these functions.

EXAMPLE

In this example, **nvdisplay** and **nvupdate** are used to change the default boot device from the bus to the ROM. The changes are made to the "BootParams" group.

Note: The fields in the "BootParams" group have different meanings for each device. For example, "DevType" values are not used for Bus devices, but are used by Serial devices to select the format for downloading. Consult the command reference for **bootbus**, **bootROM**, and **bootserial** for details.

1. At the monitor prompt, type:

```
nvdisplay
```

2. Press <cr> until the "BootParams" group is displayed.

```
Group 'BootParams'
 BootDev Bus (None,Disk,Floppy,Tape,Serial, Ethernet, ROM, Bus)
 LoadAddress          0x03010000
 ROMBase 0x00400000
 ROMSize 0x00020000
 DevType 1
 DevNumber            0
 ClrMemOnBoot         False    (False, True)

 [SP, CR to continue] or [E, e to Edit]
```

3. Press E to edit the group.
4. Press <cr> until the "BootDev" field is displayed.
5. Type the new value "ROM".

6. Press <cr> to display the "LoadAddress" field.
7. Type the address where execution begins.
8. Press <cr> to display the "ROMBase" field.
9. Type the ROM base address.
10. Press <cr> to display the "ROMSize" field.
11. Type the ROM size.
12. Press ESC or Q to quit the display.
13. Type **nvupdate** to save the new values.

EXAMPLE

In this example, **nvdisplay** and **nvupdate** are used to change the default boot device from the bus to the serial port. The changes are made to the "BootParams" group.

1. At the monitor prompt, type:

```
nvdisplay
```

2. Press <cr> until the "BootParams" group is displayed.
3. Press E to edit the group.
4. Press <cr> until the "BootDev" field is displayed.
5. Type the new value "Serial".
6. Press <cr> until the "DevType" field is displayed.
7. Type the new value for "DevType"; for example, 2 selects downloads in Heurikon binary format.
8. Edit any other fields you want to modify. Whether you use the "DevType" and "DevNumber" fields depends on the application.
9. Press ESC or Q to quit the display.
10. Type **nvupdate** to save the new values.

SETTING THE BUS CONTROL LATCH

If you are using the HK68/V3D monitor, use the command **writemem** to set the bus control latch (also see section 7.4). In this example, a series of **writemem** commands write the value 00380040_{16} to the bus control latch. The effect of the write is to set the latch as follows:

Set the slave address modifier bits to extended space (32-bit)

Set the bus release mode to release-when-done via bus control bits BC0 and BC1

Set the replacement address to 0 (base of RAM)

Set the slave address to 40000000_{16}.

EXAMPLE: Writing the value 00380040 to the bus control latch at address 02A00000.

```
writemem -b 02B0000C 0            Slave disable
writemem -l 02A00000 0            Bits 0, 8, 16 are 0.
writemem -l 02A00000 0            Bits 1, 9, 17 are 0.
writemem -l 02A00000 0            Bits 2, 10, 18 are 0.
writemem -l 02A00000 00010000     1 on DB16 setting bit 19.
writemem -l 02A00000 00010000     1 on DB16 setting bit 20.
writemem -l 02A00000 00010000     1 on DB16 setting bit 21.
writemem -l 02A00000 00000001     1 on DB0 setting bit 6.
writemem -l 02A00000 0            Bits 7, 16, 23 are 0.
writemem -b 02B0000C 1            Slave enable
```

DOWNLOADING APPLICATIONS AND DATA

The monitor commands **transmode, download,** and **call** are used for downloading applications and data in hex-Intel format, S-record format, or binary format.

transmode stands for "transparent mode," which means that the console port is connected to the download port via software. In this mode, a terminal connected to the console port can communicate with a host connected to the download port through the HK68/V3D as though the HK68/V3D were transparent. This allows you to edit your source code, recompile, initiate and complete the download, and return to the monitor, all from one terminal. This is convenient for downloading, because a single control sequence issues a carriage return to the host and issues a download command to the HK68/V3D.

Configuring the Download Port

EXAMPLE

In this example, the **nvdisplay** command changes fields in the "Download" group, which contains fields for port selection, baud rate, parity, number of data bits, and number of stop bits:

1. At the monitor prompt, type:

```
nvdisplay
```

2. Press <cr> until the "Download" group is displayed.
3. Press E to edit the group.
4. Press <cr> until the "Baud" field is displayed.
5. Type a new value.
6. Change other fields in the same way.
7. <cr> over all fields whether you edit them or not, until the monitor prompt reappears.
8. Type **nvupdate** to save the new value.

Notes: A cable reverser might be necessary for the connection.

Download Formats

Hex-Intel and S-record are common formats for representing binary object code as ASCII for reliable and manageable file downloads.

Both formats send data in blocks called records, which are ASCII strings. Records may be separated by any ASCII characters except for the start-of-record characters — "S" for S-records and ":" for

hex-Intel records. In practice, records are usually separated by a convenient number of carriage returns, linefeeds, or nulls to separate the records in a file and make them easily distinguishable by humans.

All records contain fields for the length of the record, the data in the record, and some kind of checksum. Some records also contain an address field. Most software requires that the hexadecimal characters that make up a record be in upper case only.

Hex-Intel Format

Hex-Intel format supports addresses up to 20 bits (1 megabyte). This format sends a 20-bit absolute address as two (possibly overlapping) 16-bit values. The least significant 16 bits of the address constitute the *offset,* and the most significant 16 bits constitute the *segment.* Segments can only indicate a *paragraph,* which is a 16-byte boundary. Stated in C, for example:

```
address = (segment << 4) + offset;
```

or	segment	ssss
		+
	offset	oooo
	address	aaaaa

For addresses with fewer than 16 bits, the segment portion of the address is unnecessary. The hex-Intel checksum is a two's complement checksum of all data in the record except for the initial colon (:). In other words, if you add all the data bytes in the record, including the checksum itself, the lower 8 bits of the result will be 0 if the record was received correctly.

Four types of records are used for hex-Intel format — extended address record, data record, optional start address record, and end-of file record. A file composed of hex-Intel records must end with a single end-of-file record.

Extended Address Record

```
:02000002sssscs
```

:	is the record start character.
02	is the record length.
0000	is the load address field, always 0000.
02	is the record type.
ssss	is the segment address field.
cs	is the checksum.

The extended address record is the upper 16 bits of the 20-bit address. The segment value is assumed to be zero unless one of these records sets it to something else. When such a record is encountered, the value it holds is added to the subsequent offsets until the next extended address record.

Here, the first 02 is the byte count (only the data in the ssss field are counted). 0000 is the address field; in this record the address field is meaningless so it is always 0000. The second 02 is the record type; in this case, an extended address record. *cs* is the checksum, which is a checksum of all the fields except the initial colon.

EXAMPLE

```
:020000020020DC
```

In this example, the segment address is 0020_{16}. This means that all subsequent data record addresses should have 200_{16} added to their addresses to determine the absolute load address.

Data Record

```
:11aaaa00d1d2d3...dncs
```

:	is the record start character.
11	is the record length.
aaaa	is the load address. This is the load address of the first data byte in the record (*d1*) relative to the current segment, if any.
00	is the record type.
d1...dn	are data bytes.
cs	is the checksum.

EXAMPLE

```
:0400100050D55ADF8E
```

In this example, there are four data bytes in the record. They will be loaded to address 10_{16}; if any segment value was previously specified, it is added to the address. 50_{16} is loaded to address 10_{16}, $D5_{16}$ to address 11_{16}, $5A_{16}$ to address 12_{16}, and DF_{16} to address 13_{16}. The checksum is $8E_{16}$.

Start Address Record

```
:04000003ssssooooCS
```

:	is the record start character.
04	is the record length.
0000	is the load address field, always 0000.
03	is the record type.
ssss	is the start address segment.
oooo	is the start address offset.
cs	is the checksum.

EXAMPLE

```
:040000035162000541
```

In this example, the start address segment is 5162_{16}, and the start address offset is 0005_{16}, so the absolute start address is 51625_{16}.

End-of-file Record

```
:00000001FF
```

:	is the record start character.
00	is the record length.
0000	is the load address field, always 0000.
01	is the record type.
FF	is the checksum.

This is the end-of-file record, which must be the last record in the file. It is the same for all output files.

EXAMPLE: COMPLETE HEX-INTEL FILE

```
:080000002082E446A80A6CCE3F
:020000020001FA
:08000000D0ED0A2744617EFFE7
:0400000300010002F5
:04003000902BB4FD5F
:00000001FF
```

Here is a line-by-line explanation of the example file:

`:080000002082E446A80A6CCE3F` loads:

byte 20_{16} to address 00_{16}
byte 82_{16} to address 01_{16}
byte $E4_{16}$ to address 02_{16}
byte 46_{16} to address 03_{16}
byte $A8_{16}$ to address 04_{16}
byte $0A_{16}$ to address 05_{16}
byte $6C_{16}$ to address 06_{16}
byte CE_{16} to address 07_{16}

`:020000020001FA` sets the segment value to 1, so 10_{16} must be added to all subsequent load addresses.

`:08000000D0ED0A2744617EFFE7` loads:

byte $D0_{16}$ to address 10_{16}
byte ED_{16} to address 11_{16}
byte $0A_{16}$ to address 12_{16}
byte 27_{16} to address 13_{16}
byte 44_{16} to address 14_{16}
byte 61_{16} to address 15_{16}
byte $7E_{16}$ to address 16_{16}
byte FF_{16} to address 17_{16}

`:0400000300010002F5` indicates that the start address segment value is 1_{16}, and the start address offset value is 2_{16}, so the absolute start address is 12_{16}.

`:04003000902BB4FD5F` loads:

byte 90_{16} to address 40_{16}
byte $2B_{16}$ to address 41_{16}
byte $B4_{16}$ to address 42_{16}
byte FD_{16} to address 43_{16}

`:00000001FF` terminates the file.

S-record Format

S-records are named for the ASCII character "S," which is used for the first character in each record. After the S character is another character that indicates the record type. Valid types are 0, 1, 2, 3, 5, 7, 8, and 9. After the type character is a sequence of characters that represent the length of the record, and possibly the address. The rest of the record is filled out with data and a checksum.

The checksum is the one's complement of the 8-bit sum of the binary representation of all elements of the record except the "S" and the record type character. In other words, if you sum all the bytes of a record except for the "S" and the character immediately following it with the checksum itself, you should get FF for a proper record.

S0-records (user defined)

```
S0nnd1d2d3...dncs
```

S0	indicates the record type.
nn	is the count of data and checksum bytes.
d1...dn	*d1* through *dn* are the data bytes.
cs	is the checksum.

S0 records are optional, and can contain any user-defined data.

EXAMPLE

```
S008763330627567736D
```

In this example, the length of the field is 8, and the data characters are the ACSII representation of "v30bugs." The checksum is $6D_{16}$.

S1-, S2-, and S3-records (Data Records)

```
S1nnaaaad1d2d3...dncs
S2nnaaaaaad1d2d3...dncs
S3nnaaaaaaaad1d2d3...dncs
```

S1	indicates the record type.
nn	is the count of data and checksum bytes.
a...a	is a 4-, 6-, or 8-digit address field.
d1...dn	*d1* through *dn* are the data bytes.
cs	is the checksum.

These are data records. They differ only in that S1-records have 16-bit addresses, S2-records have 24-bit addresses, and S3-records have 32-bit addresses.

EXAMPLES

```
S10801A00030FFDC95B6
```

In this example, the bytes 00_{16}, 30_{16}, FF_{16}, DC_{16}, and 95_{16} are loaded into memory starting at address $01A0_{16}$.

```
S30B30000000FFFF5555AAAAD3
```

In this example, the bytes FF_{16}, FF_{16}, 55_{16}, 55_{16}, AA_{16}, and AA_{16} are loaded into memory starting at address $3000,0000_{16}$. Note that this address requires an S3-record because the address is too big to fit into the address range of an S1-record or S2-record.

S5-records (Data Count Records)

S5*nnd1d2d3...dncs*

S5	indicates the record type.
nn	is the count of data and checksum bytes.
d1...dn	*d1* through *dn* are the data bytes.
cs	is the checksum.

S5-records are optional. When they are used, there can be only one per file. If an S5-record is included, it is a count of the S1-, S2-, and S3-records in the file. Other types of records are not counted in the S5-record.

EXAMPLE

```
S5030343B6
```

In this example, the number of bytes is 3, the checksum is $B6_{16}$, and the count of the S1-records, S2-records, and S3-records in the file is 343_{16}.

S7-, S8-, and S9-records (Termination and Start Address Records)

S705*nnaaaacs*

S804*nnaaaaaacs*

S903*nnaaaaaaaacs*

S7, S8, or S9	indicates the record type.
05, 04, 03	Count of address digits and the *cs* field.
a...a	is a 4-, 6-, or 8-digit address field.
cs	is the checksum.

These are trailing records. There can be only one trailing record per file, and it must be the last record in the output file. Included in the data for this record is the initial start address for the downloaded code.

EXAMPLES

```
S903003CC0
```

In this example, the start address is $3C_{16}$.

```
S8048000007B
```

In this example, the start address is 800000_{16}.

EXAMPLE: COMPLETE S-RECORD FILE

```
S0097A65726F6A756D707A
S10F000000001000000000084EFAFFFE93
S5030001FB
S9030008F4
```

Here is a line-by-line explanation of the example file:

`S0097A65726F6A756D707A` contains the ASCII representation of the string "zerojump".

`S10F000000001000000000084EFAFFFE93` loads the following data to the following addresses:

byte 00_{16} to address 00_{16}
byte 00_{16} to address 01_{16}
byte 10_{16} to address 02_{16}
byte 00_{16} to address 03_{16}
byte 00_{16} to address 04_{16}
byte 00_{16} to address 05_{16}
byte 00_{16} to address 06_{16}
byte 08_{16} to address 07_{16}
byte $4E_{16}$ to address 08_{16}
byte FA_{16} to address 09_{16}
byte FF_{16} to address $0A_{16}$
byte FE_{16} to address $0B_{16}$

`S5030001FB` indicates that only one S1-record, S2-record, or S3-record was sent.

`S9030008F4` indicates that the start address is 00000008_{16}.

Binary Format

The binary download format consists of a two parts:

Part 1. *Magic number* (which is 0x12345670) + *number of sections*

Part 2. For each section,

1. The load address (unsigned long)
2. The section size (unsigned long)
3. A checksum (unsigned long), which is the long word sum of the memory bytes from load address to load address, plus section size.
4. Data

Note: If you download from a UNIX host in binary format, be sure to disable the host from mapping carriage return <cr> to carriage return line feed <cr-lf>. The download port is specified in the nonvolatile memory configuration.

Transparent Mode — transmode

The **transmode** command is a "transparent mode" for communications between a host system and the HK68/V3D.

Note: For transparent mode, the "Baud" fields in the "Console" and "Download" groups must be the same.

1. At the monitor prompt, start transparent mode by typing:

```
transmode
```

2. Use one of these key sequences to start the download:

   For hex-Intel format: CTRL-@-Return

   or CTRL-@-h

   For Motorola Exormax format (S0, S1, S2, S3,S7, S8, and S9 records): CTRL-@-m

   For binary format: CTRL-@-b

3. To return to the monitor, type

   CTRL-@-ESC

EXAMPLE

If the host is a UNIX system and you have a hex-Intel file called *foo.hex* in a directory *foodir* to download, you can use the following sequence:

```
V3D[1.X] transmode
    UNIXprompt>cd foodir
    UNIXprompt>cat foo.hex
```

Press **CTRL-@-Return**.

```
.....................[dots continue during download]

V3D[1.X]
```

Serial Downloads — download

The **download** command lets you do serial downloads from a UNIX system to the HK68/V3D. Add a **-b** flag to the command for binary format, **-h** for hex-Intel format, or **-m** for Motorola S-record format. If no flag is added, the default is hex-Intel format.

For example

```
download -b
```

downloads a binary file.

Executing a Downloaded Program — call

The **call** command lets you execute a downloaded program. Use the syntax:

```
call function arg0 arg1 . . . arg7
```

You can specify up to eight arguments. The arguments can be in numeric, character, flag, string, or symbolic format.

DEBUGGING APPLICATIONS

The following commands are available for program debugging:

disassemble
dumpregs
settrace
step
exectrace

The **settrace** command allows you to set up control configuration for tracing applications. A trace is started by calling **exectrace.** The **step** command allows you to single-step through a program after **exectrace** has been called. The **disassemble** command can be called at any time to disassemble a block of memory, and **dumpregs** can be called at any time to display register contents.

The **exectrace**, **step**, and **settrace** commands call the functions *ExecTrace, Step,* and *SetTrace,* which are described together in the "Trace" page in the function reference. Details for the **disassemble** command are given on the "DisAssemble" page of the function reference, and details for the **dumpregs** command are on the "DumpRegs" page of the function reference.

Command Reference

TYPOGRAPHIC CONVENTIONS

In the following descriptions, *italic* type indicates that you must substitute your own selection for the italicized text. Square brackets [ ] enclose selections from which you *must* select *one* item.

FORMAT FOR MEMORY COMMANDS

Memory commands take the following arguments:

Arguments

value is the data operand.

startaddr is the starting address of the operation.

endaddr is the ending address of the operation.

source is the source address of the action to be performed

destination is the destination address of the action to be performed

bytecount is the number of sequential bytes to be operated on.

Required flags

For some memory commands, the data size is determined by the following flags:

-b for data in bytes (8 bits)

-w for data in 16-bit words

-l for data in 32-bit long words.

Number bases

All arguments default to hexadecimal. Specify other bases by typing a colon (:) and the base after the value.

For example, type 52:10 for decimal 52.

NONVOLATILE MEMORY

The nonvolatile memory support functions provide the interface to the nonvolatile memory. The nonvolatile commands deal only with the monitor- and Heurikon-defined sections of the nonvolatile memory. The monitor-defined sections of nonvolatile memory are read/write and can be modified by the monitor. The Heurikon-defined sections of nonvolatile memory are read only and cannot be modified. Attempts to modify these sections will result in an error message.

add

`add` *number number*

adds two integers in hexadecimal (the default), binary, octal, or decimal.

The default numeric base is decimal. Specify hexadecimal by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2". The result of the operation is displayed in hex, decimal, octal, and binary.

bootbus

`bootbus`

is an autoboot device that allows you to boot an application program over a bus interface. This command is used for fast downloads to reduce development time.

bootbus uses the *LoadAddress* field from the nonvolatile memory (group "Boot") definitions as the base address of a shared memory communications structure, described below:

```
struct BusComStruct {
    unsigned long MagicLoc;
    unsigned long CallAddress;
};
```

The structure consists of two unsigned long locations. The first is used for synchronization, and the second is the entry address of the application. The sequence of events used for loading an application is described below:

1. The host board waits for the target to write the value 0x496d4f6b to "MagicLoc" to show that the target is initialized and waiting.

2. The host board downloads the application program over the bus, then writes the entry point to "CallAddress", and then writes 0x596f4f6b to "MagicLoc" to show that the application is ready for the target.

3. Target writes value 0x42796521 to "MagicLoc" to show that the application was found and then calls the application at "CallAddress".

When the application is called, four parameters are passed to the application from the nonvolatile memory boot configuration section. The parameters are seen by the application as shown below:

```
Application(Device, Number, RomSize, RomBase)
unsigned char Device, Number;
unsigned long RomSize, RomBase;
```

bootrom

`bootrom`

is an autoboot device that allows you to boot an application program from ROM.

When the application is called, two parameters are passed to the application from the nonvolatile memory boot configuration section. The parameters are seen by the application as shown below:

```
Application(Device, Number)
unsigned char Device, Number;
```

There are no arguments for this command. The nonvolatile configuration is modified with the commands **nvdisplay** and **nvupdate**.

bootserial

`bootserial`

is an autoboot device that allows you to boot an application program from a serial port.

It determines the format of the download and the entry execution address of the downloaded application from the nonvolatile memory configuration. The nonvolatile configuration is modified with the commands **nvdisplay** and **nvupdate**.

When the application is called, three parameters are passed to the application from the nonvolatile memory boot configuration section. The parameters are seen by the application as shown below:

```
Application(Number, RomSize, RomBase)
unsigned char Number;
unsigned long RomSize, RomBase;
```

call

`call` *address arg arg arg arg arg arg arg arg*

allows execution of a program after a download from one of the board's interfaces. This function allows up to eight arguments to be passed to the called address from the command line. Arguments can be symbolic, numeric, character, flag, or string. The default numeric base is hexadecimal.

Also see transmode, download

checksummem

`checksummem` *source bytecount*

reads *bytecount* bytes starting at address *source* and computes the checksum for that region of memory. The checksum is the 16-bit sum of the bytes in the memory block.

clearmem

`clearmem` *source bytecount*

clears *bytecount* bytes starting at address *source*.

cmpmem

`cmpmem` *source destination bytecount*

compares *bytecount* bytes at the *source* address with those at the *destination* address. Any differences are displayed.

configboard

`configboard`

configures the board to the state specified by the nonvolatile memory configuration.

configboard can be used to reconfigure the board's various interfaces after modification of the nonvolatile memory configuration. This function accepts no parameters.

copymem

`copymem -[b,w,l]` *source destination bytecount*

copies *bytecount* bytes from the *source* address to the *destination* address.

date

`date`

reads the real time clock.

The **date** command displays the date in the format:

```
Friday June 22, 1990   12:25:31.10
```

If the real-time clock is not set up an error message is displayed:

```
Warning: Real Time clock is invalid.
```

disassemble

`disassemble` *startaddr lines*

disassembles memory into MPU assembly language. This command accepts a variable number of arguments. The start address must be given.

startaddr the address to start the display. The address is assumed to be hexadecimal.

lines the number of lines to display. If the number of lines is not specified, the default is 20 lines.

The disassembler recognizes all of the MC68030 instructions except for floating point. Floating point instructions are displayed as unrecognized instructions that are represented with the .word directive. The format of the disassembler should correspond to the format used in the MC68030 instruction set manual.

Unrecognized instructions can cause the disassembler to lose synchronization with an assembly program, which can result in an error in the display. This usually corrects itself within several instructions.

displaymem

`displaymem` *startaddr lines*

displays *lines* of memory starting at *startaddr*. Press any key to interrupt the display. If the *lines* argument is not specified, 16 lines are displayed. If the previous command was **displaymem**, pressing <cr> displays the next block of memory.

div

`div` *number number*

divides two integers in hexadecimal (the default), binary, octal, or decimal.

The default numeric base is decimal. Specify hex by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2". The result of the operation is displayed in hex, decimal, octal, and binary.

download

`download -[b,h,m]` *address*

provides a serial download from a host computer to the board. **download** uses binary, hex-Intel, or Motorola S-record format, as specified by flags -b, -h or -m:

Flags If no flag is specified, the default format is hex-Intel.

-b binary

-h hex-Intel

-m Motorola S-record

The binary download format is described briefly below:

1. Magic number (0x12345670) + number of sections
2. Each section:
   - Load address (unsigned long)
   - Section size (unsigned long)
   - Checksum (unsigned long)
   - Data

The checksum is the long word sum of the memory bytes from load address to load address, plus section size.

Note: If you download from a UNIX host in binary format, be sure to disable the host from mapping <cr> to <cr-lf>. The download port is specified by in the nonvolatile memory configuration.

dumpregs

`dumpregs`

dumps the contents of the MPU registers from the last processor exception that occurred. This command accepts no arguments.

exectrace

`exectrace` *address arg arg arg arg arg arg*

is used to execute the application program with the trace modes enabled. This command accepts up to 7 arguments from the command line. Arguments can be in symbolic, numeric, character, flag or string format. The default numeric base is hexadecimal.

fillmem

`fillmem -[b,w,l]` *value startaddr endaddr*

fills memory with *value* starting at address *startaddr* to address *endaddr.*

For example, to fill the second megabyte of memory with the data 0x12345678 type:

```
fill -l 12345678 100000 200000
```

findmem

`findmem -[b,w,l]` *searchval startaddr endaddr*

searches memory for a value from address *startaddr* to address *endaddr* for memory locations specified by the data *searchval.*

findnotmem

`findnotmem -[b,w,l]` *searchval startaddr endaddr*

searches from address *startaddr* to address *endaddr* for memory locations that are different from the data specified by *searchval.*

findstr

`findstr` *searchstr startaddr endaddr*

searches from address *startaddr* to address *endaddr* for a match to the same string specified by the data string *searchstr.*

help

Use the help command to view the definitions and descriptions of monitor commands.

For instructions on editing command lines, type `help editor`.

For a list of command-line functions, type `help functions`.

For a detailed memory map, type `help memmap`.

For instructions on using the monitor entry points, type `help entrypoint`.

For details on a specific command, type `help` and a command name.

mul

`mul` *number number*

multiplies two integers in hexadecimal (the default), binary, octal, or decimal from the monitor.

The default numeric base is decimal. Specify hex by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2". The result of the operation is displayed in hex, decimal, octal, and binary.

nvdisplay

`nvdisplay`

used to display the Heurikon-defined and monitor-defined nonvolatile sections. The values are displayed in groups. Each group has a number of fields. Fields are displayed as hexadecimal or as a list of legal values.

To display the next group, press <space> or <cr>.

To edit fields within the displayed group, press **E**.

To quit the display, press ESC or Q.

To save the changes, type the command **nvupdate**.

To quit without saving the changes, type the command **nvopen**.

The following error message indicates an attempt to change a write-protected field:

```
Warning, protected region was not modified.
```

The table on the following pages shows all the groups and fields you can edit when you use the **nvdisplay** command:

Group	Fields	Purpose	Heurikon Default	Optional Values
Console and Download				
	Port	Selects communications port.	A (Download) B (Console)	(A, B, C, D)
	Baud	Selects baud rate.	9600	
	Parity	Selects parity type.	None	(Even, Odd, None, Force)
	Data	Selects the number of data bits for transfer.	8-Bits	(5-Bits, 6-Bits, 7-Bits, 8-Bits)
	StopBits	Selects the number of stop bits for transfer.	2-Bits	(1-Bit, 2-Bits)
	XOnXOff	Selects XOnXOff protocol.	On	(Off, On)
	ChBaudOnBreak	Break character causes baud rate change.	False	(False, True)
	RstOnBreak	Break character causes reset.	False	(False, True)
VmeBus				
	ExtSlaveMap	Address to map slave extended space.	0x80000000	
	StdSlaveMap	Address to map slave standard space.	0x000000	

	AddrModSel		ExAll	(None, StDat, StAll, ExSuDat, ExDat, ExAll) The abbreviations stand for: no slave access allowed (disable), standard data, all standard, extended supervisor data, extended data, all extended (see section 7.4)
	ReplaceAddr	Standard space replacement address	0x00000000	
	EnbleSlave	Enable/disable slave standard space.	True	(False, True)
	MastRelModes	Select master release modes.	OnRequest	(WhenDone, OnRequest, OnClear, Never)
	SlaveRelOnReq	Enable/disable slave release-on-request	On	(Off, On)
	LocalBusTimer	Select duration of on-card bus timer.	32µ	(4µs, 16µ, 32µ, 64µ, 128µ, 256µ, 512µ, Off)
	VmeBusTimer	Select duration of VMEbus timer.	64µ	(4µs, 16µ, 32µ, 64µ, 128µ, 256µ, 512µ, Off)
	Sysfail	Turn SYSFAIL* on or off.	Off	(Off, On)
	IndivRMC	Turn indivisible read-modify-write on or off.	Off	(Off, On)
Mailbox				
	ShtSlaveMap	Address to map slave short space	0xfff8	
	EnblSht	Enable/disable short space.	False	(False, True)
Cache				
	InstrCache	Turn instruction cache on or off.	On	(Off, On)
	DataCache	Turn data cache on or off.	Off	(Off, On)
Misc				
	PowerUpMemClr	Clear memory on power-up.	True	(False, True)
	ClrMemOnReset	Clear memory on reset.	True	(False, True)
	PowerUpDiag	Use power-up diagnostics.	On	(Off, On)
	CountValue	Choose shortest (0) to longest (7) duration for autoboot countdown.	7	(0, 1, 2, 3, 4, 5, 6, 7)
BootParams				
	BootDev	Select boot device.	None	(None, Disk, Floppy, Tape, Serial, Ethernet, ROM, Bus)
	LoadAddress	Define load address.	0x03010000	
	RomBase	Define ROM base.	0x00400000	This field is used only when BootDev is defined as ROM.
	RomSize	Define ROM size.	0x00020000	This field is used only when BootDev is defined as ROM.

DevType	Define device type.	0	Whether you use this field depends on the application. When BootDev is defined as Bus or ROM, DevType refers to a device type. When BootDev is defined as Serial, DevType selects a download format (0 for hex-Intel, 1 for S-records, 2 for Heurikon binary).
DevNumber	Define device number.	0	Whether you use this field depends on the application.
ClrMemOnBoot	Clear memory on boot.	False	(False, True)

nvinit

`nvinit` *sernum revlev ecolev writes*

used to initialize the nonvolatile memory to the default state defined by the monitor. First **nvinit** clears the memory and then writes the Heurikon and monitor data back to EEPROM.

CAUTION: **nvinit** clears any values you have changed from the default. Use nvinit only if the nonvolatile configuration data structures might be in an unknown state and you must return them to a known state.

Arguments

sernum	serial number
revlev	revision level
ecolev	standard ECO level
writes	the number of writes to nonvolatile memory

Potential error

```
Warning, protected region cannot be initialized.
```

This message appears if you try to use **nvinit** to clear write-protected memory.

nvopen

`nvopen`

reads and checks the monitor and Heurikon-defined sections. If the nonvolatile sections do not validate then error messages are displayed.

nvset

`nvset` *group field value*

used to modify the Heurikon-defined and monitor-defined nonvolatile sections. To modify the list with the **nvset** command, you must specify the group and field to be modified and the new value. The group, field, and value can be abbreviated, as in the examples below:

```
nvset console port B
nvset con dat 6
```

CAUTION: Use **nvdisplay** instead of **nvset** to reduce the risk of invalidating nonvolatile memory.

Note: The nonvolatile memory support functions provide the interface to the nonvolatile memory. The nonvolatile commands deal only with the monitor- and Heurikon-defined sections of the nonvolatile memory. The monitor-defined sections of nonvolatile memory are read/write and can be modified by the monitor. The Heurikon-defined section of nonvolatile memory is read only and cannot be modified. Attempts to modify these sections will result in an error message when the store is done.

nvupdate

`nvupdate`

attempts to write the Heurikon- and monitor-defined nonvolatile sections back to the EEPROM. First the data is verified, and then it is written to the device. The write is verified and all errors are reported.

prstatus

`prstatus`

This command prints the physical Ethernet ID for the board based on the model and serial number and then indicates if the board is set up as the system controller.

rand

`rand`

is a linear congruent random number generator that uses a function "Seed" and a variable "Value." "Value" is generated by the real time clock. The random number returned is an unsigned long.

readmem

`readmem -[b,w,l]` *address*

reads a memory location specified by *address*. This command displays the data in hexadecimal, decimal, octal, binary, or string format.

setdate

`setdate` ***dayofwk mon dayofmon year hour min AM/PM***

sets the clock. The month, day of week, and AM/PM values are assumed to be character strings; other parameters may be numeric.

dayofwk may be abbreviated (Su, M, Tu, W, Th, F, Sa).

month may also be abbreviated (Ja, F, Mar, Ap, May, Jun, Jul, Au, S, O, N, D).

dayofmon is restricted to the range 0-31.

year ranges from 1990 to 2089.

hour is restricted to the range 0-23.

min is restricted to the range 0-59.

AM/PM is the string AM or PM.

Also see date.

setmem

`setmem -[b,w,l]` ***address***

allows memory locations to be modified starting at ***address***. **setmem** first displays the value that was read. Then you can type new data for the value. If you press <cr> after the data, the address counts up. If you press <ESC> after the data, the address counts down.

settrace

`settrace`

displays and modifies the current trace configuration. The trace configuration display is shown below:

```
MPU Trace Configuration Display:
  SingleStep        On              (Off, On)
  Branch            Off             (Off, On)
  Call              Off             (Off, On)
  Return            Off             (Off, On)
  Prereturn         On              (Off, On)
  Breakpoints       Off             (Off, On)
  BreakPoint1       0x0
  BreakPoint2       0x0
  BreakPoint3       0x0
  BreakPoint4       0x0
  BreakPoint5       0x0
  BreakPoint6       0x0
  BreakPoint7       0x0
  BreakPoint8       0x0
```

The trace configuration indicates the state of the various trace modes and break points. Trace modes can be turned on and off to allow tracing on every instruction, branches, calls or returns. There is a switch to stop tracing when a key is pressed and a switch to display instructions as they are executed.

slaveenable

`slaveenable -[e,s,c]` *address*

enables the specified VMEbus address space.

- **-e** VMEbus extended space
- **-s** VMEbus standard space
- **-c** communications. Signifies VMEbus short space.

address should contain the base address that will be be mapped, where the base address is a hex value. The useful portion of the address field is defined as:

- `FFxxxxxx` extended space
- `xxFxxxxx` standard space
- `xxxxFFxx` short space

slavedis

`slavedis -[e,s,c]`

disables the specified VMEbus address space.

- **-e** VMEbus extended space
- **-s** VMEbus standard space
- **-c** communications. Signifies VMEbus short space

starttimer

This command only serves as a working example for initializing the timer/clock to generate interrupts and for handling the interrupts. **starttimer** initializes the CIO, attaches the interrupt handler, and then starts the counter timer. In this example the variable "NumTicks" is incremented for every interrupt received and the LED display is incremented for every interrupt. The interrupts are turned off with the **stoptimer** command, which disconnects the interrupt handler. This command currently initializes the CIO to generate an interrupt every 10 milliseconds using vector number 82_{16}.

step

`step`

is used to continue execution of an application program after a trace exception has occurred. This command can only be run after the **exectrace** command has been executed.

stoptimer

`stoptimer`

turns off the **starttimer** command.

sub

`sub` *number number*

subtracts two integers in hexadecimal (the default), binary, octal, or decimal.

The default numeric base is decimal. Specify hexadecimal by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2". The result of the operation is displayed in hex, decimal, octal, and binary.

swapmem

`swapmem` *source destination bytecount*

swaps *bytecount* bytes at the *source* address with those at the *destination* address.

testmem

`testmem` *startaddr endaddr*

performs a nondestructive memory test.

This command can be used to verify memory (DRAM, SRAM, VMEbus,). If no arguments are specified, the command reads the nonvolatile configuration and tests the on-card dynamic memory. If *startaddr* and *endaddr* are specified, then an alternate memory area can be tested. The default numeric base is hexadecimal.

transmode

`transmode`

provides an interface to UNIX® through the board by connecting the console to a download port. Several key sequences are used to leave transparent mode and to initiate a download:

CTRL-@-RETURN	Download hex-Intel.
CTRL-@-h	Download hex-Intel.
CTRL-@-m	Download Motorola S-records.
CTRL-@-b	Download binary.
CTRL-@-ESC	Return to monitor.

A cable reverser might be necessary for the connection.

writemem

`writemem -[b,w,l]` *address value*

writes *value* to a memory location specified by *address*.

writestr

`writestr` "*string*" *address*

writes the ASCII string specified by *string* to a memory location specified by *address*. The string must be enclosed in double quotes (" ").

Crease and tear along perforation.

◆ REMOTE HOST COMMANDS

transmode

provides an interface to UNIX® through the board via the console to a download port. Several key characters are used to leave transparent mode and to initiate a download:

Download hex-Intel:
CTRL-@-h or CTRL-@-RETURN
Download Motorola S-records:
CTRL-@-m
Download binary:
CTRL-@-b
Return to Monitor:
CTRL-@-ESC

download -[b,h,m] *address*

provides a serial download using binary (-b), hex-Intel (-h), or Motorola S-record (-s) format.

call *address arg arg arg arg arg arg arg arg*

allows execution of a program after a download from one of the board's interfaces. This command allows up to 8 arguments to be passed to the called address from the command line. Arguments can be symbolic, numeric, character, flag, or strings. The default numeric base is hexadecimal.

◆ TRACE COMMANDS

disassemble *startaddr lines*

disassembles memory into MPU assembly language. The display of *lines* starts at *startaddr*.

dumpregs

dumps the contents of the registers from the last fault that occurred.

exectrace *address arg arg arg arg arg arg arg*

executes an application program with the trace modes enabled. This command allows up to 7 arguments to be passed to the called address from the command line. Arguments can be in symbolic, numeric, character, flag or string format.

step

continues execution of an application program after a trace exception has occurred. This command can only be run after the **exectrace** command has been executed.

settrace

displays and modifies the current trace configuration.

◆ UTILITIES

configboard

configures the board to the state specified by the nonvolatile memory configuration. **configboard** can be used to reconfigure the board's various interfaces after modification of the nonvolatile memory configuration.

date

displays the date in the format:

```
Friday April 19, 1991   12:25:31.10
```

setdate *dayofwk mon dayofmon year hour min AM/PM*

sets the clock.

starttimer

This command only serves as a working example. **starttimer** initializes the CIO, attaches the interrupt handler, and then starts the counter timer. In this example the variable "NumTicks" is incremented for every interrupt received and the LED display is incremented for every interrupt. The interrupts are turned off with the **stoptimer** command, which disconnects the interrupt handler.

◆ ARITHMETIC FUNCTIONS

add *number number*
sub *number number*
mul *number number*
div *number number*
rand

The default base is hexadecimal. To use another base, add a colon (:) and the base after the number.

HK68/V3D Monitor

◆

Quick-Reference to Commands

HEURIKON®

◆ HELP COMMANDS

help

displays a summary of the monitor.

help functions

displays a list of monitor functions.

help memmap

displays the memory map for the HK68/V3D.

◆ NV-RAM COMMANDS

nvdisplay

Displays the nonvolatile memory contents by group and field. Press E to edit a field.

nvopen

Reads and checks Heurikon nonvolatile memory.

nvupdate

Saves changes to nonvolatile memory.

◆ MEMORY COMMANDS

All numeric arguments default to hexadecimal. Specify other bases by typing a colon (:) and the base after the value. For example, type 52:10 for decimal 52.

checksummem *source bytecount*

reads *bytecount* bytes starting at address *source*; indicates the checksum for that region of memory.

clearmem *source bytecount*

clears *bytecount* bytes starting at address *source*.

cmpmem *source destination bytecount*

compares *bytecount* bytes at *source* with those at *destination*. Any differences are displayed.

copymem -[b,w,l] *source destination bytecount*

copies *bytecount* bytes from *source* to *destination*.

displaymem *startaddr lines*

displays *lines* of memory starting at *startaddr*. *Lines* defaults to 16. <cr> displays the next block.

fillmem -[b,w,l] *value startaddr endaddr*

fills memory with *value* between *startaddr* and *endaddr* in bytes, words, or longs.

findmem -[b,w,l] *searchval startaddr endaddr*

searches from *startaddr* to *endaddr* for memory patterns specified by *searchval*.

findnotmem -[b,w,l] *searchval startaddr endaddr*

searches from *startaddr* to *endaddr* for memory patterns that are different from *searchval*.

findstr *searchstr startaddr endaddr*

searches from *startaddr* to *endaddr* for a match to the same string specified by *searchstr*.

readmem -[b,w,l] *address*

reads a memory location specified by *address* and displays the data in hexadecimal, decimal, octal, binary, and string format.

setmem -[b,w,l] *address*

allows memory locations to be modified starting at *address*. **setmem** first displays the value that was read. Then you can type new data for the value. If you press <cr> after the data, the address counts up. If you press <ESC> after the data, the address counts down.

swapmem *source destination bytecount*

swaps *bytecount* bytes at the *source* address with those at the *destination* address.

testmem *startaddr endaddr*

performs a nondestructive memory test.

writemem -[b,w,l] *address value*

writes *value* to a memory location specified by *address*.

writestr "*string*" *address*

writes the ASCII string specified by *string* to a memory location specified by *address*. The string must be enclosed in double quotes (").

◆ BUS COMMANDS

slaveenable -[e,s,c] *address*

enables the VMEbus extended (-e), standard (-s) or short (-c) space. *address* should contain the base address that will be mapped. The base address is a hex value. The useful portion of the address field is defined as:
`FFxxxxxx` (extended space)
`xxFxxxxx` (standard space)
`xxxxFFxx` (short space)

slavedis -[e,s,c]

disables the VMEbus extended (-e), standard (-s) or short (-c) space.

prstatus

displays the Ethernet ID and whether the board is VMEbus system controller.

◆ BOOT COMMANDS

bootbus

receives applications over the backplane. Addresses and sizes are specified in nonvolatile memory.

bootrom

loads applications from ROM and executes. Addresses and sizes are specified in nonvolatile memory. Useful for booting user code or an operating system.

bootserial

loads applications from a serial port and executes. Addresses and sizes are specified in nonvolatile memory.

Crease and tear along perforation.

◆ REMOTE HOST COMMANDS

transmode

provides an interface to UNIX® through the board via the console to a download port. Several key characters are used to leave transparent mode and to initiate a download:

Download hex-Intel:
CTRL-@-h or CTRL-@-RETURN
Download Motorola S-records:
CTRL-@-m
Download binary:
CTRL-@-b
Return to Monitor:
CTRL-@-ESC

download -[b,h,m] *address*

provides a serial download using binary (-b), hex-Intel (-h), or Motorola S-record (-s) format.

call *address arg arg arg arg arg arg arg arg*

allows execution of a program after a download from one of the board's interfaces. This command allows up to 8 arguments to be passed to the called address from the command line. Arguments can be symbolic, numeric, character, flag, or strings. The default numeric base is hexadecimal.

◆ TRACE COMMANDS

disassemble *startaddr lines*

disassembles memory into MPU assembly language. The display of *lines* starts at *startaddr*.

dumpregs

dumps the contents of the registers from the last fault that occurred.

exectrace *address arg arg arg arg arg arg arg*

executes an application program with the trace modes enabled. This command allows up to 7 arguments to be passed to the called address from the command line. Arguments can be in symbolic, numeric, character, flag or string format.

step

continues execution of an application program after a trace exception has occurred. This command can only be run after the **exectrace** command has been executed.

settrace

displays and modifies the current trace configuration.

◆ UTILITIES

configboard

configures the board to the state specified by the nonvolatile memory configuration. **configboard** can be used to reconfigure the board's various interfaces after modification of the nonvolatile memory configuration.

date

displays the date in the format:

```
Friday April 19, 1991   12:25:31.10
```

setdate *dayofwk mon dayofmon year hour min AM/PM*

sets the clock.

starttimer

This command only serves as a working example. **starttimer** initializes the CIO, attaches the interrupt handler, and then starts the counter timer. In this example the variable "NumTicks" is incremented for every interrupt received and the LED display is incremented for every interrupt. The interrupts are turned off with the **stoptimer** command, which disconnects the interrupt handler.

◆ ARITHMETIC FUNCTIONS

add *number number*
sub *number number*
mul *number number*
div *number number*
rand

The default base is hexadecimal. To use another base, add a colon (:) and the base after the number.

HK68/V3D Monitor

◆

Quick-Reference to Commands

HEURIKON®

◆ HELP COMMANDS

help

displays a summary of the monitor.

help functions

displays a list of monitor functions.

help memmap

displays the memory map for the HK68/V3D.

◆ NV-RAM COMMANDS

nvdisplay

Displays the nonvolatile memory contents by group and field. Press E to edit a field.

nvopen

Reads and checks Heurikon nonvolatile memory.

nvupdate

Saves changes to nonvolatile memory.

◆ MEMORY COMMANDS

All numeric arguments default to hexadecimal. Specify other bases by typing a colon (:) and the base after the value. For example, type 52:10 for decimal 52.

checksummem *source bytecount*

reads *bytecount* bytes starting at address *source*; indicates the checksum for that region of memory.

clearmem *source bytecount*

clears *bytecount* bytes starting at address *source*.

cmpmem *source destination bytecount*

compares *bytecount* bytes at *source* with those at *destination*. Any differences are displayed.

copymem -[b,w,l] *source destination bytecount*

copies *bytecount* bytes from *source* to *destination*.

displaymem *startaddr lines*

displays *lines* of memory starting at *startaddr*. *Lines* defaults to 16. <cr> displays the next block.

fillmem -[b,w,l] *value startaddr endaddr*

fills memory with *value* between *startaddr* and *endaddr* in bytes, words, or longs.

findmem -[b,w,l] *searchval startaddr endaddr*

searches from *startaddr* to *endaddr* for memory patterns specified by *searchval*.

findnotmem -[b,w,l] *searchval startaddr endaddr*

searches from *startaddr* to *endaddr* for memory patterns that are different from *searchval*.

findstr *searchstr startaddr endaddr*

searches from *startaddr* to *endaddr* for a match to the same string specified by *searchstr*.

readmem -[b,w,l] *address*

reads a memory location specified by *address* and displays the data in hexadecimal, decimal, octal, binary, and string format.

setmem -[b,w,l] *address*

allows memory locations to be modified starting at *address*. **setmem** first displays the value that was read. Then you can type new data for the value. If you press <cr> after the data, the address counts up. If you press <ESC> after the data, the address counts down.

swapmem *source destination bytecount*

swaps *bytecount* bytes at the *source* address with those at the *destination* address.

testmem *startaddr endaddr*

performs a nondestructive memory test.

writemem -[b,w,l] *address value*

writes *value* to a memory location specified by *address*.

writestr "*string*" *address*

writes the ASCII string specified by *string* to a memory location specified by *address*. The string must be enclosed in double quotes (").

◆ BUS COMMANDS

slaveenable -[e,s,c] *address*

enables the VMEbus extended (-e), standard (-s) or short (-c) space. *address* should contain the base address that will be mapped. The base address is a hex value. The useful portion of the address field is defined as:

`FFxxxxxx` (extended space)
`xxFxxxxx` (standard space)
`xxxxFFxx` (short space)

slavedis -[e,s,c]

disables the VMEbus extended (-e), standard (-s) or short (-c) space.

prstatus

displays the Ethernet ID and whether the board is VMEbus system controller.

◆ BOOT COMMANDS

bootbus

receives applications over the backplane. Addresses and sizes are specified in nonvolatile memory.

bootrom

loads applications from ROM and executes. Addresses and sizes are specified in nonvolatile memory. Useful for booting user code or an operating system.

bootserial

loads applications from a serial port and executes. Addresses and sizes are specified in nonvolatile memory.

Errors and Screen Messages

Most commands return an explanatory message for misspelled or mistyped commands, missing arguments, or invalid values. This table lists errors that can be attributed to other causes, especially errors that indicate a problem in the nonvolatile memory configuration.

Some errors can be resolved only with a call to Heurikon Customer Support, **1-800-327-1251**.

Message	Source and suggested solution
Error while clearing NV memory. Error while reading NV memory. Error while storing NV memory.	NV memory has become corrupted. Type **nvinit** to restore defaults. If the problem persists, call a Heurikon customer representative.
Hit 'H' to skip bus auto-boot	Consult the introduction to this appendix for information about power-up conditions.
No help for ___.	The topic for help was misspelled or is not available. Check the spelling. If the topic was a command name, type **help** to check the spelling of the command. You must use the full command name, not an abbreviation.
Power-up Memory Test FAILED. Power-up Serial Test FAILED.	A failed Memory Test or Serial Test could mean a hardware malfunction. Report the error to Heurikon Customer Support.
Unable to change ID.	The Module ID can be changed only by Heurikon.
Unknown ___	The Module ID is incorrect. Report the error to Heurikon Customer Support.
Unknown boot device	The boot device is invalid. Use **nvdisplay** to check and edit the "BootParams" group, "BootDev" field. Save a new value with **nvupdate**.
Unexpected _____ Exception at _____ .	There are many possible sources for this error. If the error is displayed during boot, it could mean that autoboot is enabled and invalid parameters are being used. If the error is displayed at reset or power-up and autoboot is *not* enabled, report the error to Heurikon Customer Support. If the error is displayed after a command has been executed, probably an attempt has been made to perform an operation that causes an exception.
Warning NV memory board initialization skipped.	Only minimum configuration has been completed. The configuration data structures are invalid.
Warning NV memory is invalid - using defaults.	Consult the introduction to this appendix for information about reset conditions.

Warning protected region of NV memory cannot be initialized. Warning protected region of NV memory was not modified. Warning protected region of NV memory is corrupt.	An attempt was made to change a write-protected NV field. Either re-read the nonvolatile memory defaults for these protected regions by typing the **nvopen** command, or return any fields you tried to edit to their original values.
Warning: Real Time clock is invalid.	The real-time clock has not been set up. See the RTC section of this manual and code samples in Appendix B for setup information.

Function Reference

The reference pages have been alphabetically sorted, but some pages contain the descriptions for several related functions. Use the cross-reference to function names to locate each function.

No argument checking will take place for functions that are called directly from the command line. It is advisable instead to use the monitor commands whenever possible.

The functions require spaces between the function name and its arguments. No parentheses or other punctuation is necessary.

EXAMPLES

```
UnMaskInts 1

ConnectHandler 0xf8 0x1000
```

FUNCTION SUMMARY

Examples of common uses of monitor functions and the functions available for each use are listed below. This list is not exhaustive, and not all functions on the list might be supported on the HK68/V3D.

Callable functions that are key entry points into the monitor:

StartMonitor

Interrupt support to read registers and tables used for interrupts:

MaskIntsO
UnMaskIntsO
VecToVecAddr(Vector)
VectInitO
ConnectHandler(Vector)
DisConnectHandler(Vector)

Register and cache functions:

FlushCacheO

Configuring the board by using NV memory parameters:
ConfigVmeBusO
ConfigVsbBusO
ConfigCioO
ConfigScsiO
ConfigSerDevsO

Initializing the board to the default conditions:

InitBoard()
InitCio()
InitScsi()

Controlling the serial ports:

PutC(Char)
RPutC(Char)
GetC()
RGetC()
KBHit()
RKBHit()
TxMT()
RTxMT()
RChBaud(BaudRate)
ChBaud(BaudRate)

Writing to CIO data ports:

ReadCIOPortA()
ReadCIOPortB()
ReadCIOPortC()
WriteCIOPortB()

Unmasking VMEbus interrupts:

UnMaskVMEInt()

Writing to the LED display:

SetLedDisplay()

Writing or reading the memory image to or from the NV memory device at the specified offset:

NVOp(Operation, MemImagePtr, Size, Offset)
(Operations 0-4 are fix, clear, check, open and save.)

Executing the device I/O:

NVRamAcc(Flag, ByteNumber)

Setting the memory images to the default monitor values:

SetNvDefaults()

Booting a program from the specified drive:

BootUp()

Setting the memory management functions and determining where free memory resides:

MemReset()
MemAdd(Address, Size)
MemStats()
MemTop()
MemBase()

Memory management — allocating and returning memory:

Malloc(Size)
Free(Address)
Calloc(Number, Size)
CFree(Address)
ReAlloc(Address,Size)

Function Name	Page Title	Section Title
Add()	Add	Monitor (Std)
atob()	atoh	Monitor (Std)
atod()	atoh	Monitor (Std)
atoh()	atoh	Monitor (Std)
atoo()	atoh	Monitor (Std)
atoX()	atoh	Monitor (Std)
BinToHex()	atoh	Monitor (Std)
BootBus()	BootBus	Monitor (Std)
BootRom()	BootRom	Monitor (Std)
BootSerial()	BootSerial	Monitor (Std)
BootUp()	BootUp	Monitor (Std)
Call()	Call	Monitor (Std)
Calloc()	MemMng	Monitor (Std)
CFree()	MemMng	Monitor (Std)
ChBaud()	Serial	Monitor (Std)
CheckSumMem()	BlockMem	Monitor (Std)
ClearMem()	BlockMem	Monitor (Std)
CmpMem()	BlockMem	Monitor (Std)
CmpStr()	Strings	Monitor (Std)
ConnectHandler()	Exceptions	Processor (MC68030)
CopyMem()	BlockMem	Monitor (Std)
Date()	Date	Monitor (Std)
DisAssemble()	DisAssemble	Processor (MC68030)
DisConnectHandler()	Exceptions	Processor (MC68030)
DisDataCache()	Cache	Processor (MC68030)
DisInstCache()	Cache	Processor (MC68030)
DispGroup()	NVSupport	Monitor (Std)
DisplayMem()	DisplayMem	Monitor (Std)
Div()	Add	Monitor (Std)
DownLoad()	DownLoad	Monitor (Std)
DumpRegs()	DumpRegs	Processor (MC68030)
EnbDataCache()	Cache	Processor (MC68030)
EnbInstCache()	Cache	Processor (MC68030)
ExecTrace()	Trace	Processor (MC68030)
FastFillMem()	FastFillMem	Processor (MC68030)
FillMem()	BlockMem	Monitor (Std)
FindBitSet()	atoh	Monitor (Std)
FindMem()	FindMem	Monitor (Std)
FindNotMem()	FindMem	Monitor (Std)
FindStr()	FindMem	Monitor (Std)
FlushCache()	Cache	Processor (MC68030)
Free()	MemMng	Monitor (Std)
FromFifo()	InitFifo	Monitor (Std)
GetC()	Serial	Monitor (Std)
Help()	Help	Monitor (Std)
HexToBin()	atoh	Monitor (Std)
InitFifo()	InitFifo	Monitor (Std)
InitTrace()	Trace	Processor (MC68030)
IsLegal()	IsLegal	Monitor (Std)
KBHit()	Serial	Monitor (Std)
Malloc()	MemMng	Monitor (Std)

```
MaskInts() ..... Interrupts ..... Processor (MC68030)
MemAdd() ..... MemMng ..... Monitor (Std)
MemReset() ..... MemMng ..... Monitor (Std)
MemStats() ..... MemMng ..... Monitor (Std)
Mul() ..... Add ..... Monitor (Std)
NVDisplay() ..... NVMemory ..... Monitor (Std)
NVInit() ..... NVMemory ..... Monitor (Std)
NVOp() ..... NVSupport ..... Monitor (Std)
NVOpen() ..... NVMemory ..... Monitor (Std)
NVSet() ..... NVMemory ..... Monitor (Std)
NVUpdate() ..... NVMemory ..... Monitor (Std)
Probe() ..... Exceptions ..... Processor (MC68030)
PutC() ..... Serial ..... Monitor (Std)
Rand() ..... Add ..... Monitor (Std)
RChBaud() ..... Serial ..... Monitor (Std)
ReAlloc() ..... MemMng ..... Monitor (Std)
RGetC() ..... Serial ..... Monitor (Std)
RKBHit() ..... Serial ..... Monitor (Std)
RPutC() ..... Serial ..... Monitor (Std)
RTxMT() ..... Serial ..... Monitor (Std)
Seed() ..... Add ..... Monitor (Std)
SetDate() ..... Date ..... Monitor (Std)
SetMem() ..... DisplayMem ..... Monitor (Std)
SetNvDefaults() ..... NVSupport ..... Monitor (Std)
SetTrace() ..... Trace ..... Processor (MC68030)
Step() ..... Trace ..... Processor (MC68030)
StrCat() ..... Strings ..... Monitor (Std)
StrCmp() ..... Strings ..... Monitor (Std)
StrCpy() ..... Strings ..... Monitor (Std)
StrLen() ..... Strings ..... Monitor (Std)
Sub() ..... Add ..... Monitor (Std)
SwapMem() ..... BlockMem ..... Monitor (Std)
TestMem() ..... TestMem ..... Monitor (Std)
ToFifo() ..... InitFifo ..... Monitor (Std)
TransMode() ..... TransMode ..... Monitor (Std)
TxMT() ..... Serial ..... Monitor (Std)
UnMaskInts() ..... Interrupts ..... Processor (MC68030)
VectInit() ..... Exceptions ..... Processor (MC68030)
VecToVecAddr() ..... Exceptions ..... Processor (MC68030)
xprintf() ..... xprintf ..... Monitor (Std)
xsprintf() ..... xprintf ..... Monitor (Std)
```

SYNOPSIS

```
Add(Arg1, Arg2)
unsigned long Arg1, Arg2;

Sub(Arg1, Arg2)
unsigned long Arg1, Arg2;

Mul(Arg1, Arg2)
unsigned long Arg1, Arg2;

Div(Arg1, Arg2)
unsigned long Arg1, Arg2;

unsigned long Rand()

Seed(Value)
unsigned long Value;
```

DESCRIPTION

These functions are provided to allow the monitor to do basic arithmetic operations on the command line using a variety of numeric bases. Each function accepts two arguments *Arg1* and *Arg2* to perform the arithmetic operation and returns the results. For the *Add* and *Mul* functions argument order is not important. The *Sub* function performs *Arg1 minus Arg2*. The *Div* function performs the *Arg1 divided by Arg2* operation checking to avoid division by zero.

The function *Rand* is a linear congruent random generator. The random number returned is an unsigned long. The function *Seed* is used to seed the random number generator. The variable *Value* should be generated from the real time clock.

SYNOPSIS

```
unsigned long atoh(p)
char *p;

unsigned long atod(p)
char *p;

unsigned long atoo(p)
char *p;

unsigned long atob(p)
char *p;

unsigned long atoX(p, Base)
char *p;
int Base;

BinToHex(Val)
unsigned long Val;

HexToBin(Val)
unsigned long Val;

FindBitSet(Number)
unsigned long Number;
```

DESCRIPTION

These functions are a collection of numeric conversion programs used to convert character strings to numeric values, convert Hex to BCD, BCD to Hex, and to search for bit values.

The *atoh* function provides conversion of an ascii string to a hex number. The *atoh* function provides conversion of an ascii string to a decimal number. The *atoo* function provides conversion of an ascii string to an octal number. The *atob* function provides conversion of an ascii string to a binary number.

The function *atoX* accepts both the character string *p* and the numeric base *Base* to be used in converting the string. This can be used for numeric bases other than the standard bases 16, 10, 8 and 2.

The *BinToHex* function provides conversion of a binary value to packed nibbles (BCD). The *HexToBin* function provides conversion of packed nibbles (BCD) to binary. This function accepts the parameter *Val*, which is assumed to contain a single hex number of value 0-99.

The *FindBitSet* function searches the *Number* for the first non-zero bit. The bit position of the least significant non-zero bit is returned. This function accepts the parameter *Val*, which is assumed to contain a single BCD number of value 0-99.

SYNOPSIS

```
ClearMem(Dest, ByteCount)
unsigned char *Dest;
unsigned long ByteCount;

FillMem(Flag, Value, StartAddr, EndAddr)
unsigned long Value, StartAddr, EndAddr;
char Flag;

CopyMem(Src, Dest, ByteCount)
unsigned char *Src, *Dest;
unsigned long ByteCount;

SwapMem(Src, Dest, ByteCount)
char *Src, *Dest;
int ByteCount;

CmpMem(Src, Dest, ByteCount)
char *Src, *Dest;
int ByteCount;

CheckSumMem(Addr, ByteCount)
unsigned char *Addr;
unsigned long ByteCount;
```

DESCRIPTION

These functions provide the ability to clear, fill, copy, swap, compare, and checksum blocks of memory. All of the functions treat memory as a block of bytes except for the *FillMem* function, which can treat memory blocks as bytes, words, or longs.

The function *ClearMem* clears the number of bytes specified by *ByteCount* starting at address *Dest*.

The function *FillMem* fills memory starting at address *StartAddr* to address *EndAddr* with the specified *Value*. Memory is treated as bytes, words, or longs as specified by the character *Flag* which must be b, w, or l for byte, word, and long.

The function *CopyMem* copies from source address *Src* to destination address *Dest* the number of bytes specified by *ByteCount*.

The function *SwapMem* swaps two memory blocks of size specified by *ByteCount*. The blocks are located at the addresses specified by *Src* and *Dest*.

The function *CmpMem* compares two memory blocks of size specified by *ByteCount*. The blocks are located at the addresses specified by *Src* and *Dest*. If the memory blocks are different, a message indicating where and how they differ is printed.

The function *CheckSumMem* computes the checksum for the memory block of size *ByteCount*. The memory block is specified by the *Address* parameter. The checksum is the 16-bit sum of the bytes in the memory block.

SYNOPSIS

BootBus(PowerUp)
int PowerUp;

DESCRIPTION

The *BootBus* function is one of the autoboot devices supported by the monitor. The purpose of this function is to provide a method of loading an application program over a bus interface. This is accomplished by communicating with another board on the bus through a shared memory location. This provides very fast downloads that reduce software development time. This function uses the *LoadAddress* field from the NV memory configuration as the base address of a shared memory communications structure described below:

```
struct BusComStruct {
unsigned long MagicLoc;
unsigned long CallAddress;
};
```

This structure consists of two unsigned long locations. The first is used for synchronization and the second is the entry address of the application. The sequence of events used for loading an application is described below:

First, the host board waits for the target to write the value 0x496d4f6b (character string "ImOk") to the magic location *MagicLoc*, indicating the target is initialized and waiting for a download.

Second, the host board downloads the application program over the bus, writes the entry point or execution address of the application to *CallAddress*, and then writes 0x596f4f6b (character string "YoOk") to *MagicLoc*, indicating the application is ready for the target.

Finally, the target detects the host has written to the magic location, copies the application program to local memory, and then sets the value to 0x42796521 (character string "Bye!") indicating the application was found. The target then calls the application at *CallAddress*. When the application is called, four parameters that are pulled from the NV memory boot configuration section are passed to the application. The parameters as seen by the application are shown below:

```
Application(Device, Number, RomSize, RomBase)
unsigned char Device, Number;
unsigned long RomSize, RomBase;
```

These parameters allow multiple boards using the same facility to receive different configuration information from the monitor.

SEE ALSO

BootUp()

SYNOPSIS

BootRom(PowerUp)
int PowerUp;

DESCRIPTION

The *BootRom* function is one of the autoboot devices supported by the monitor. The purpose of this function is to provide a method of loading an application program from ROM. If only one ROM socket is provided, the application must be loaded into the same ROM as the monitor. The monitor must be located in either the highest or lowest portion of the ROM, depending on where the processor expects the monitor at reset. The 80960CA and Gmicro processors require the monitor in the high portion, and the 68000 family requires the monitor in the lowest portion.

The location, size and load address of the application is specified in the NV memory boot configuration space. The NV memory configuration parameters used are *RomBase*, *RomSize* and *LoadAddress*.

This monitor function, when called, copies the number of bytes specified by the NV memory parameter *RomSize* from the ROM location specified by *RomBase* to the memory location specified by *LoadAddress*. After the memory is loaded, the application is called at *LoadAddress*. When the application is called, two parameters that are pulled from the NV memory boot configuration section are passed to the application. The parameters as seen by the application are shown below:

```
Application(Device, Number)
unsigned char Device, Number;
```

These parameters allows multiple boards using the same facility to receive configuration information from the monitor.

ARGUMENTS

The flag *PowerUp* indicates if this function is called for the first time. If so, memory must be cleared.

SEE ALSO

BootUp()

SYNOPSIS

BootSerial(PowerUp)
int PowerUp;

DESCRIPTION

The *BootSerial* function is one of the autoboot devices supported by the monitor. The purpose of this function is to provide a method of loading an application program from a serial port. This function uses the *LoadAddress* and *DevType* fields from the NV memory configuration to determine the format of the download and the entry execution address of the downloaded application. The *DevType* field selects one of the download formats specified below:

DEVICE NUMBER		DOWNLOAD FORMAT
INT_MCS86	0	Intel MCS-86 Hexadecimal Format
MOT_EXORMAT	1	Motorola Exormax Format (S0-S3,S7-S9 Records).
HK_BINARY	2	Heurikon Binary Format.

When the application is called, three parameters that are pulled from the NV memory boot configuration section are passed to the application. The parameters as seen by the application are shown below:

```
Application(Number, RomSize, RomBase)
unsigned char Number;
unsigned long RomSize, RomBase;
```

These parameters allow multiple boards using the same facility to receive different configuration information from the monitor.

SEE ALSO

BootUp()

SYNOPSIS

BootUp(PowerUp)
int PowerUp;

DESCRIPTION

The *BootUp* function is called immediately after the NV memory device has been opened and the board has been configured according to the NV configuration. First, this function determines if memory is to be cleared according to the NV configuration and the flag *PowerUp*.

The monitor provides an autoboot feature that allows an application to be loaded from a variety of devices and executed. This function uses the NV configuration to determine which device to boot from and calls the appropriate boot strap program. The monitor supports the ROM, BUS, and SERIAL autoboot devices, which are not hardware-specific. The remainder of the devices may or may not be supported by board-specific functions described elsewhere. Currently, the board specific devices are SCSI (floppy, disk, and tape) and ethernet.

ARGUMENTS

The flag *PowerUp* indicates if this function is being called for the first time. If so, memory must be cleared.

SEE ALSO

StartMon.c, NvMonDefs.h, NVTable.c BootRom(), BootBus() BootWinch(), BootFloppy(), BootTape()

SYNOPSIS

FlushCache()
EnbInstCache()
DisInstCache()
EnbDataCache()
DisDataCache()

DESCRIPTION

These functions are used to enable, disable and flush the instruction and data caches. The *FlushCache* function flushes both the instruction and data caches.

The functions *EnbInstCache* and *EnbDataCache* enable the instruction and data caches respective by turning on the enables is the CACR register.

The functions *DisInstCache* and *DisDataCache* disable the instruction and data caches respective by turning off the enables in the CACR register. Before a cache is disabled it is flushed.

SEE ALSO

SYNOPSIS

```
Call(Funct, Arg0, Arg1, Arg2, Arg3, Arg4, Arg5, Arg6, Arg7)
int (*Funct)();
unsigned long Arg0, Arg1, Arg2, Arg3, Arg4, Arg5, Arg6, Arg7;
```

DESCRIPTION

The *Call* command allows execution of programs that have been downloaded through one of the board's interfaces. This function allows up to eight arguments to be passed to the called function from the command line. If the application program wants to return to the monitor, it is important that the processor stack registers and special purpose registers remain unchanged.

ARGUMENTS

The first argument *Funct* is the address of the application program to be executed. The next arguments *Arg0* through *Arg7* are the arguments to be passed to the application program.

SEE ALSO

DownLoad(), TransMode().

SYNOPSIS

Date()

SetDate(DayOfWeek, Month, DayOfMon, Year, Hour ,Min, Period)
unsigned long Hour, Minutes, DayOfMonth, Year;
char *Month, *DayOfWeek, *Period;

DESCRIPTION

The *Date* and *SetDate* commands provide the real-time clock support for the monitor. The *Date* function initializes a monitor time structure defined below by reading from the real-time clock device. This is done by calling the *RtcAcc* function. The structure entries are then checked for illegal values and the date is printed.

```
struct tm {
unsigned long tm_fsec;        * fract of seconds (0 - 99) *
unsigned long tm_sec;         * seconds (0 - 59)          *
unsigned long tm_min;         * minutes (0 - 59)          *
unsigned long tm_hour;        * hours   (0 - 23)          *
unsigned long tm_mday;        * day of month (1 - 31)     *
unsigned long tm_mon;         * month of year (0 - 11)    *
unsigned long tm_year;        * Year - 1900               *
unsigned long tm_wday;        * day of week (sunday = 0)  *
};
```

The *SetDate* function accepts 7 parameters that describe the *DayOfWeek, Month, DayOfMonth, Year, Hour, Minute,* and *Period* (AM/PM). The month, day of week, and period are assumed to be character strings. All other parameters are numeric. This information is verified and used to initialize the time structure described above. After verifications, the structure is written to the real-time device, and the time is again printed.

ARGUMENTS

The variable *DayOfWeek* is a character string that contains enough characters to uniquely define one of the following character strings:

```
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
```

The variable *Month* is a character string that contains enough characters to uniquely define one of the following character strings:

```
January, February, March,     April,   May,      June,
July,    August,   September, October, November, December
```

The variable *DayOfMonth* is a numeric value from 0 to 31, the variable *Year* is a numeric value from 1990 to 2089, the variable *Hour* is a numeric value from 0 to 23, the variable *Minute* is a numeric value from 0 to 59, and the variable *Period* is a character string that contains either the character string *AM* or *PM*.

SEE ALSO

RctAcc().

SYNOPSIS

```
DisplayMem(Address, Lines)
unsigned long Address, Lines;

SetMem(Flag, Address)
int Flag;
unsigned long Address;
```

DESCRIPTION

The *DisplayMem* function and the *SetMem* functions are used to display and modify memory locations.

The *SetMem* function allows interactive modification of memory starting at the location specified by the argument *Address* using the format specified by the character *Flag*, which indicates byte (b), word (w), or long (l). After the value is read and displayed, new data may be entered. If a <cr> follows the data entered, the address counts up. If <ESC> follows the data entered, the address counts down. If an empty line is entered, the data for that location is left unchanged. To quit this function, type any illegal hex character.

The *DisplayMem* function displays memory in lines of 16 bytes each, starting at the location specified by the argument *Address*. The data is displayed first as hex character values on the right, and then as the ascii equivilent on the left, if printable. Non-printable ascii characters are printed as a dot. The number of lines displayed is specified by the parameter *Lines*. If *Lines* is not specified (equals NULL), the default number of lines (16) is displayed. The display can be interrupted by hitting any character. This function returns the next address to be displayed so the command can be reentered from the last displayed location.

SYNOPSIS

```
DisAssemble(Addr, Cnt)
unsigned short *Addr;
int Cnt;
```

DESCRIPTION

The *DisAssemble* function starts reading instructions at the memory address specified by *Addr* and displays the assembly langauge equivilent of memory. The argument *Cnt* indicates the number of instructions to be disassembled. If *Cnt* is not specified (0) then the default number of lines are printed.

The disassembler knows about all of the MC68030 instructions with the exception of floating point. Floating point instructions will be displayed as unrecognized instructions which are represented with the .WORD directive. The format of the disassembler should correspond to the format used in the motorola MC68030 instruction set manual.

Unrecognized instructions can cause the disassembler to loose synchronization with an assembly program which can result in an error in the display. This ussually corrects itself within several instructions.

SEE ALSO

SYNOPSIS

DownLoad(Flag, Address)
char Flag;
unsigned long Address;

DESCRIPTION

This monitor command provides a serial download using either Hex-Intel, Motorola S-Records, or binary format. The argument *Flag*, which is one of the following characters, indicates the download mode:

```
h       Intel MCS-86 Hexadecimal Format
m       Motorola Exormax Format (S0-S3,S7-S9 Records).
b       Heurikon Binary Format.
```

If *Flag* is NULL then this function defaults to using Hex-Intel. If the second parameter is specified (not NULL) the specified *Address* is added to those found in the download records. This allows a download to another board across a bus interface (which requires an offset).

When the binary download format is used, the data are moved in raw 8-bit format. This improves the download time by about 220%. This format requires a header be sent to describe the data location, size, and checksum. This format is described briefly below.

First received is the magic number and number of sections. The magic number is the unsigned long value 0x12345670 where the lowest nibble specifies the number of sections expected. Each section following the magic number requires a 12-byte header that specifies the load address, section size, and checksum of the data. After the header are the raw data. The section header is described below:

```
struct BinaryHeader {
unsigned long Address;
unsigned long Size;
unsigned long CheckSum;
} BinHdr;
```

For the magic number and section header, the bytes are sent most significant byte first. As an example, the magic number would be sent in the order 0x12, 0x34, 0x56, 0x73.

SEE ALSO

BootSerial().

SYNOPSIS

DumpRegs()

DESCRIPTION

The *DumpRegs* function dumps a display of the processor registers at the point of the last exception. This function does not display the current register contents which would be meaningless but instead displays the registers values stored at the last exception. The stored register values are kept in a structure *ProcRegs* which has the following format.

```
struct RegFile {
unsigned long DataRegs[8];      * data registers 0-7          *
unsigned long AddrRegs[8];      * address registers 0-7       *
unsigned long CtrlRegs[16];     * Control Regs PC SFC DFC     *
* VBR CACR CAAR SSP ISP MSP SR *
} RegFile;
```

Currently the floating point registers are not displayed because of the lack of floating point support in xprintf. The trace mechanism interacts with this function by copying its register display structure over *ProcRegs* and then calling this function. After a trace exception the *DumpRegs* command can be used to display the registers saved at the exception as long as another exception does not occur.

SEE ALSO

SYNOPSIS

```
VectInit()

unsigned long *VecToVecAddr(Vector)
unsigned long Vector;

ConnectHandler(Vector, Handler)
unsigned long Vector;
int (*Handler)();

DisConnectHandler(Vector)
unsigned long Vector;

Probe(DirFlag, SizeFlag, Address, Data)
char DirFlag, SizeFlag;
unsigned long Address;
unsigned long Data;
```

DESCRIPTION

These functions are the 68030 processor specific functions which provide interrupt and exception handling support.

The function *VectInit* initializes the entire interrupt table to reference the unexpected interrupt handler. This assures that the board will not hang when unexpected interrupts are received. The unexpected interrupt handler saves the state of the processor at the point the interrupt was detected and then calls the *IntrErr* function, which displays the error and restarts the monitor.

The function *VectToVectAddr* converts the argument *Vector* to the vector address contained in the interrupt table associated with the vector. This allows modification of vectors without knowing where the interrupt table is located in memory.

The function *ConnectHandler* allocates an interrupt wrapper, links the wrapper into the interrupt table and then initializes the wrapper to call the *Handler* address. The argument *Vector* indicates the vector number to be connected and the argument *Handler* should be the address of the function that will handle the interrupts. The Interrupt Wrapper is a relocatable assembly language module that can be placed in free memory and linked into the interrupt table. This allows the programmer to avoid using assembly language programming for interrupts.

The function *DisConnectHandler* modifies the interrupt table entry associated with *Vector* to use the unexpected interrupt handler and then de-allocates the memory used for the interrupt wrapper allocated by *ConnectHandler*. Because both *ConnectHandler* and *DisConnectHandler* use the *Malloc and Free* facilities it is necessary for memory management to be initialized.

The function *Probe* should be used to access memory locations that may or may not result in a watchdog timeout or bus error. This function returns TRUE if the location was accessed and FALSE if the access resulted in a bus error. The argument *DirFlag* indicates whether a read (0) or a write (1) should be attempted. The argument *SizeFlag* indicates whether a byte access (1), a word access (2) or a long access (4) should be attempted. The argument *Address* indicates the address to be accessed and the argument *Data* is a pointer to where the read or write data is.

SEE ALSO

SYNOPSIS

```
FastFillMem(Value, StartAddress, EndAddress)
unsigned long Value;
unsigned long *StartAddress, *EndAddress;
```

DESCRIPTION

The *FastFillMem* function provides a fast method for filling memory with the *Value* specified. The *FillMem* monitor command is too slow to clear large amounts of memory (megabytes). This function takes advantage of the burst ability of the processor, which can achieve much higher data rates than single reads and writes.

The parameters *StartAddress* and *EndAddress* indicate the start and end of the block of memory to be filled. The argument *Value* is the value used to fill memory. The value is always assumed to be an unsigned long value and the start and end pointers are assumed to be long word aligned addresses.

SEE ALSO

SYNOPSIS

```
FindNotMem(Flag, SearchVal, StartAddr, EndAddr)
unsigned long StartAddr, EndAddr;
unsigned long SearchVal;
char Flag;

FindStr(SearchStr, StartAddr, EndAddr)
unsigned long StartAddr, EndAddr;
char *SearchStr;

FindMem(Flag, SearchVal, StartAddr, EndAddr, InvFlag)
unsigned long StartAddr, EndAddr;
unsigned long SearchVal, InvFlag;
char Flag;
```

DESCRIPTION

These functions are used to search memory for a particular pattern or lack of a pattern. If the specified pattern is found, the location of the pattern is displayed. All of these functions can be interrupted by hitting any character on the console device.

The function *FindNotMem* searches memory from address *StartAddr* to address *EndAddr* for memory locations that are not the same as the data specified by *SearchVal*. The data size is determined by the character *Flag*, which indicates byte (b), word (w), or long (l).

The function *FindStr* searches memory from address *StartAddr* to address *EndAddr* for the occurrence of the string specified by *SearchStr*.

The function *FindMem* searches memory from address *StartAddr* to address *EndAddr* for memory locations that are the same as the data specified by *SearchVal*. The data size is determined by the character *Flag*, which indicates byte (b), word (w), or long (l). The last argument *InvFlag*, if TRUE, causes the search to act like the *FindNotMem* function.

SYNOPSIS

```
Help(Name)
char *Name;
```

DESCRIPTION

The help function provides the on-line help facilities for the monitor. The monitor provides an on-line manual page describing each monitor command. Also provided is a set of auxiliary manual pages, which are not tied to any particular command.

This function accepts the character string *Name*, which is used to search the symbol table and auxiliary manual table for a match. If a match is found, the manual page is printed. If no match is found, this function indicates there is no help for the specified string. If the argument *Name* is not specified (NULL), then the auxiliary manual page describing the help facility itself is displayed.

SEE ALSO

SYNOPSIS

```
InitFifo(FPtr, StartAddr, Length)
struct Fifo *FPtr;
unsigned char *StartAddr;
int Length;

ToFifo(FPtr, c)
struct Fifo *FPtr;
unsigned char c;

FromFifo(FPtr, Ptr)
struct Fifo *FPtr;
unsigned char *Ptr;
```

DESCRIPTION

These functions provide the necessary interface to initialize, read, and write a software fifo. The fifo is used for buffering serial I/O when using transparent mode, but could be used for a variety of applications. All three functions accept as the first argument a pointer *FPtr* to a fifo structure that is used to manage the fifo. This fifo structure is described briefly below:

```
struct Fifo {
unsigned char *Top;
unsigned char *Bottom;
int Length;
unsigned char *Front;
unsigned char *Rear;
int Count;
} Fifo;
```

The function *InitFifo* initializes the fifo control structure specified by *FPtr* to use the unsigned character buffer starting at *StartAddr* that is of size *Length*.

The function *ToFifo* writes the byte *c* to the specified fifo, This function returns TRUE if there is room in the fifo, FALSE if the fifo is full.

The function *FromFifo* reads a byte from the specified fifo. If a character is available, it is written to the address specified by the pointer *Ptr* and the function returns TRUE. If no character is available, the function returns FALSE.

SYNOPSIS

UnMaskInts()
MaskInts()

DESCRIPTION

The functions *UnMaskInts* and *MaskInts* are used to enable and disable interrupts at the processor. The function *UnMaskInts* sets the interrupt level bits in the processor status register to 0 allowing all levels to interrupt the processor. The function *MaskInts* sets the interrupt level bits in the processor status register to 7 disabling all interrupts except the non-maskable level 7 interrupt.

SEE ALSO

SYNOPSIS

IsLegal(Type,Str)
unsigned char Type;
char *Str;

DESCRIPTION

This function is used to determine if the specified character string *Str* contains legal values to allow the string to be parsed as decimal, hex, upper case, or lower case. The function *IsLegal* traverses the character string until a NULL is reached. Each character is verified according to the *Type* argument. The effects of specifying each type are described below:

Type / Value		Legal Characters
DECIMAL	0x8	0 - 9
HEX	0x4	0 - 9, A - F, a - f
UPPER	0x2	A - Z, 0 - 9
LOWER	0x1	a - z
ALPHA	0x3	A - Z, a - z, 0 - 9

If the character string contains legal characters, this function returns TRUE; otherwise, it returns FALSE. The string equivalent of the character functions isalpha(), isupper(), islower(), and isdigit() can be constructed from this function, which deals with the entire string instead of a single character.

SYNOPSIS

```
char *Malloc(NumBytes)
unsigned long NumBytes;

char *Calloc(NumElements, Size)
unsigned long NumElements, Size;

Free(MemLoc)
unsigned long *MemLoc;

CFree(Block)
unsigned long *Block;

char *ReAlloc(Block, NumBytes)
char *Block;
unsigned long  NumBytes;

MemReset()

MemAdd(MemAddr, MemBSize)
unsigned long MemAddr, MemBSize;

MemStats()
```

DESCRIPTION

The memory management functions provide basic functions necessary to allocate and free memory from a memory pool. The monitor initializes the memory pool to use all on-card memory after the monitor's *bss* section. If any of the autoboot features are used, the memory pool is not initialized and the application program is required to set up the memory pool if these functions are to be used.

The functions *Malloc, Calloc* and *ReAlloc* are used to allocate memory from the memory pool. Each of these functions returns a pointer to the memory requested if the request can be satisfied and NULL if there is not enough memory to satisfy the request. The function *Malloc* accepts one argument *NumBytes* indicating the number of bytes requested. The function *Calloc* accepts two arguments *NumElements* and *Size* indicating a request for a specified number of elements of the specified size. The function *ReAlloc* allows the reallocation of a memory block by either returning the block specified by *Block* to the free pool and allocating a new block of size *NumBytes* or by determining that the memory block specified by *Block* is big enough and returning the same block to be reused.

The functions *Free* and *CFree* are used to returns blocks of memory to the free memory pool which were requested by either *Malloc, Calloc, or ReAlloc.* The address of the block to be returned is specified by the argument *MemLoc,* which must be the same value returned by one of the allocation functions. An attempt to return memory to the free memory pool which was not acquired by the allocation functions is a fairly reliable way of blowing up a program and should be avoided.

The function *MemReset* sets the free memory pool to the empty state. This function must be called once for every reset operation before the memory management facilities can be used. It is also necessary to call this function before every call to *MemAdd.*

The function *MemAdd* is used to initialize the free memory pool to use the memory starting at the

address specified by *MemAddr* of size specified by *MemSize*. This function currently allows for only one contiguous memory pool and must be preceded by a function call to *MemReset* whenever called.

The function *MemStats* provides the ability to monitor the memory usage. This function outputs a table showing how much memory is available and how much is used and lost as a result of overhead.

SEE ALSO

MemTop(), MemBase().

SYNOPSIS

NVDisplay()

NVUpdate()

NVOpen()

NVSet(GroupName, FieldName, Value)
char *GroupName, *FieldName, *Value;

NVInit(SerNum, RevLev, ECOLev, Writes)
int SerNum, ECOLev, RevLev, Writes;

DESCRIPTION

The NV memory support functions provide the interface to the NV memory. All of these functions deal only with the monitor- and Heurikon-defined sections of the NV memory. The monitor-defined sections of NV memory are read/write and can be modified by the user. The Heurikon-defined section of NV memory is read only and cannot be modified. Attempts to modify the Heurikon defined sections will result in an error message when the store is done.

The *NVOpen* function reads and checks the monitor and Heurikon-defined sections. If the NV sections do not validate, then an error message is displayed.

The *NVUpdate* function attempts to write the Heurikon- and monitor-defined NV sections back to NV memory. The data are first verified, and then written to the device. The write is verified and all errors are reported.

The *NVInit* function is used to initialize the NV memory to the default state defined by the monitor. It first clears the memory and then writes the Heurikon and monitor data back to NV memory. This function accepts as arguments the serial number, revision level, ECO level and the number of writes to NV Memory. If the monitor-defined NV memory section somehow becomes corrupt, the command sequence *NVInit* followed by *NVUpdate* should result in the monitor-defined NV memory resetting to the default state. This sequence of commands will result in error messsages that indicate the Heurikon-defined section was not changed. These messages can be ignored.

The *NVDisplay* and *NVSet* commands are used to display and modify the Heurikon-defined and monitor-defined NV sections. The values are displayed in logical groups. Each group has a number of fields. Fields are displayed as hex, decimal, or a list of legal values. An example of the display is shown below:

```
Group 'Console'
Port            A           (A, B, C, D)
Baud            9600
Parity          None        (Even, Odd, None)
Data            8-Bits      (5-Bits, 6-Bits, 7-Bits, 8-Bits)
StopBits        2-Bits      (1-Bit, 2-Bits)
```

After each group is displayed, the user has the option of moving to the next group display, editing the current group display, or quitting the display completely. If an edit is requested, all fields of the group are prompted for modification one-by-one. An empty line indicates that no modification is necessary.

To modify a field using *NVSet*, the group and field to be modified are specified and the new value is provided. This command allows abbreviation of the field and group names. The *NVDisplay* function allows fields to be changed interactively during the display.

SYNOPSIS

```
SetNvDefaults(Groups, NumGroups)
NVGroupPtr Groups;
int NumGroups;

DispGroup(Group, EditFlag)
NVGroupPtr Group;
unsigned long EditFlag;

NVOp(NVOpCmd, Base, Size, Offset)
unsigned long NVOpCmd, Size, Offset;
unsigned char *Base;
```

DESCRIPTION

The support functions used for displaying, initializing, and modifying the NV memory data structures can also be used to manage other data structures which may or may not be stored in NV memory.

The method used to create a display of a data structure is to create a second structure that contains a description of every field of the first structure. This description is done using the *NVGroup* structure. Each entry in the *NVGroup* structure describes a field name, pointer to the field, size of the field, indication of how the field is to be displayed, and the initial value of the field.

An example data structure is shown below as well as the *NVGroup* data structure necessary to describe the data structure. This example might describe the coordinates and depth of a window structure.

```
struct NVExample {
NV_Internal Internal;
unsigned long  XPos, YPos;
unsigned short Mag;
} NVEx;

NVField ExFields[] = {
{ "XPos",    (char *) &NVEx.XPos, sizeof(NVEx.XPos),
NV_TYPE_DECIMAL, 0, 100, NULL},
{ "YPos",    (char *) &NVEx.YPos, sizeof(NVEx.YPos),
NV_TYPE_DECIMAL, 0, 200, NULL},
{ "Depth"    (char *) &NVEx.Mag, sizeof(NVEx.Mag),
NV_TYPE_DECIMAL, 0, 4, NULL}
}

NVGroup ExGroups[] = {
{ "Window",  sizeof(ExFields)/sizeof(NVField), ExFields }
};
```

If passed a pointer to the ExGroups structure, the function *DispGroup* generates the display shown below. The second parameter *EditFlag* indicates whether to allow changes to the data structure after it is displayed (Same as in the NVDisplay command).

```
Window Display Configuration
XPos            100
YPos            200
```

```
Magnitude       4
```

The *SetNvDefaults* function, when called with a pointer to the ExGroup structure, can be used to initialize the data structure to those values specified in the *NVGroup* structure. The second parameter *NumGroups* indicates the number of groups to be initialized.

The *NVOp* function can be used to store and recover data structures from NV memory. The only requirement of the data structure to be stored in NV memory is that the first field of the structure be *NVInternal*, which is where all the bookkeeping for the NV memory section is done. The first parameter *NvOpCmd* indicates the command to be performed. A summary of the commands is shown below:

Command	Value	Description
NV_OP_FIX	0	Fix NV section checksum
NV_OP_CLEAR	1	Clear NV section
NV_OP_CK	2	Check if NV section is valid
NV_OP_OPEN	3	Open NV Section
NV_OP_SAVE	4	Save NV Section
NV_OP_CMP	5	Compare NV Section data

The second parameter, *Base*, indicates the base address of the data structure to be operated on, and the *Size* parameter indicates the size of the data structure to be operated on. The *Offset* parameter indicates the byte offset in the NV memory device where the data structure is to be stored. An example of how to initialize, store, and recall the example data structure is shown below.

```
NVOp(NV_OP_CLEAR, &NVEx, sizeof(NVEx), 0);
NVOp(NV_OP_SAVE , &NVEx, sizeof(NVEx), 0);
NVOp(NV_OP_OPEN , &NvEx, sizeof(NVEx), 0);

NVOp(NV_OP_FIX,   &NVEx, sizeof(NVEx), 0);
NVOp(NV_OP_SAVE , &NVEx, sizeof(NVEx), 0);
```

The clear, save, and open operations cause the NV device to be cleared and filled with the NVEx data structure; then the data structure is filled from NV memory. The fix and save operation are used to modify the NV device, which updates the internal data structures and then writes them back to the NV memory device.

If errors are encountered during the check, save or compare operations, an error message is returned from the function *NvOp*. The error codes are listed below.

Error number		Description
NVE_NONE	0	No errors.
NVE_OVERFLOW	1	NV device write count exceeded.
NVE_MAGIC	2	Bad magic number read from NV device.
NVE_CKSUM	3	Bad checksum read from NV device.
NVE_STORE	4	Write to NV device failed.
NVE_CMD	5	Unknown operation requested.
NVE_CMP	6	Data does not compare to NV device.

SEE ALSO

NVFields.h

SYNOPSIS

```
char GetC()
char RGetC()

PutC(c)
char c;
RPutC(c)
char c;

KBHit()
RKBHit()

TxMT()
RTxMT()

ChBaud(Baud)
int Baud;
RChBaud(Baud)
int Baud;
```

DESCRIPTION

The serial support functions defined here provide the ability to read, write, and poll the monitor serial devices. The monitor initializes and controls two serial devices: one is the console, which provides the user interface, and the other is the modem (also known as "download" or "remote") device, which can be used to connect to a development system. Each console function has a complement function that performs the same operation on the modem device. The modem device functions are prefixed with the letter 'R' for remote. Each serial port is configured at reset according to the NV memory configuration.

The functions *GetC* and *RGetC* are used to read characters from the console and modem devices respectively. When called, these functions will not return until a character has been received from the serial port. The character read is returned to the calling function.

The functions *PutC* and *RPutC* are used to write characters from the console and modem devices respectively. When called, these functions will not return until a character has been accepted by the serial port. The character *c* is the only argument these functions accept.

The functions *KBHit* and *RKBHit* are used to poll the console and modem devices for available characters. If the receiver indicates a character is available, these functions return TRUE; otherwise, they return FALSE.

The functions *TxMT* and *RTxMT* are used to poll the console and modem devices if the transmitter can accept more characters. If the transmitter indicates a character can be sent, these functions return TRUE; otherwise, they return FALSE.

The functions *ChBaud* and *RChBaud* allow modification of the console and modem device baud rates. The argument *Baud* specifies the new baud rate to use for the port. Because these functions accept any baud rate, care must be taken to request only baud rates the terminal or host system can support.

SEE ALSO

GetChar(), PutChar(), KeyHit(), TxEmpty(), ChangeBaud().

SYNOPSIS

```
CmpStr(Str1, Str2)
char *Str1, *Str2;

StrCmp(Str1, Str2)
char *Str1, *Str2;

StrCpy(Dest, Source)
char *Dest, *Source;

StrLen(Str)
char *Str;

StrCat(DestStr, SrcStr)
char *DestStr, *SrcStr;
```

DESCRIPTION

These functions provide the basic string manipulation functions necessary to compare, copy, concatenate, and determine the length of strings.

The function *CmpStr* compares the two null terminated strings pointed to by *Str1* and *Str2*. If they are equal, it returns TRUE; otherwise, it returns FALSE. Note that this version does not act the same as the UNIX® *strcmp* function. *CmpStr* is non-case-sensitive and only matches characters up to the length of *Str1*. This is useful for pattern matching and other functions.

The function *StrCmp* compares the two null terminated strings pointed to by *Str1* and *Str2*. If they are equal, it returns TRUE; otherwise, it returns FALSE. Note that this version acts the same as the UNIX *strcmp* function.

The function *StrCpy* copies the null terminated string *Source* into the string specified by *Dest*. There are no checks to verify that the string is large enough or is null terminated. The only limit is the monitor-defined constant MAXLN (80), which is the largest allowed string length the monitor supports. The length of the string is returned to the calling function.

The function *StrLen* determines the length of the null terminated string *Str* and returns the length. If the length exceeds the monitor defined limit MAXLN, then the function returns MAXLN.

The function *StrCat* concatenates the string *SrcStr* onto the end of the string *DestStr*.

SEE ALSO

SYNOPSIS

TestMem(Base, Top)
unsigned long Base, Top;

DESCRIPTION

The *Testmem* function is a non-destructive memory test. The variables *Base* and *Top* indicate the range to be tested. If the variable *Top* is set to 0, then the base and top addresses are obtained from the monitor memory functions *MemBase* and *MemTop*. When called, this function prints the progress of the test and summarizes the number of passes and failures of the test. This function can be interrupted after each pass of the test by hitting any character during the test.

SEE ALSO

MemTop(), MemBase().

SYNOPSIS

```
ExecTrace(Funct, Arg0, Arg1, Arg2, Arg3, Arg4, Arg5, Arg6)
int (*Funct)();
unsigned long Arg0, Arg1, Arg2, Arg3, Arg4, Arg5, Arg6;

Step(Cnt)
unsigned long Cnt;

InitTrace()

SetTrace()
```

DESCRIPTION

The functions defined in this module are used to initiate maintain and manage the configuration and exception traces for the 68030 Processor. The trace facilities allow the programmer to step instruction by instruction through an application program. The tracing mechanism allows the programmer to select a variety of events to trace on. The trace events include every instruction, branches jumps, returns, or up to 8 instruction addresses. The trace can also be initialized to print every instruction or stop when a key is hit.

The function *ExecTrace* initiates the trace mechanism for the function specified by the argument *Funct* and begins tracing by passing the arguments *Arg0* through *Arg6* to the function to be traced.

The function *Step* re-enters the trace mechanism after an exception has occured. This function can only be used after a trace is initiated by the *ExecTrace* function. The argument *Cnt* indicates the number of events to be skipped before stopping the trace.

The function *InitTrace* initializes the structures used by the trace facilities to a default state. This function must be called at reset.

The function /fISetTrace/fR provides the ability to change the trace configuration. The trace configuration display allows the trace configuration to be modified using the same type of display as the NV memory display. The tracing configuration is maintained through the use of the *SetNVDefualts* and *DispGroup* functions.

When using trace facilities it is important to understand how the trace mechanism works. Because the stack and interrupt table are used by the trace functions the processor stack pointer and vector base register cannot be modified by the program which is being traced. The trace mechanism currently stops on every instruction and determines if an event has been reached. This results in the program running much slower than normal.

SEE ALSO

SYNOPSIS

TransMode()

DESCRIPTION

This function connects the console port to the modem port to provide a connection to a development system through the board. Several key characters are used to leave transparent mode (CTRL-@-ESC) and to initiate a download (CTRL-@-RETURN). To initiate a download using a specific download format, type the command that generates the download records without hitting return. Then use one of the following character sequences:

```
CTRL-@-RETURN    Download hex-intel
CTRL-@-h         Download hex-intel
CTRL-@-m         Download Motorola S-Records
CTRL-@-b         Download binary
```

This function uses software fifos to buffer characters between the two systems. This seems to work reasonably well for most processors but can lose characters if large numbers of characters are displayed. In general, the only complete solution is to use serial interrupts rather than polling. Since this is not likely to happen, beware that the transparent mode command will allow execution of commands without problems, but may have problems if text editing is attempted.

SEE ALSO

DownLoad().

SYNOPSIS

```
xprintf(CtrlStr, Arg0, Arg1 ... ArgN)
char *CtrlStr;
unsigned long Arg0, Arg1, ... ArgN;

xsprintf(Buffer, CtrlStr, Arg0, Arg1 ... ArgN)
char *Buffer, *CtrlStr;
unsigned long Arg0, Arg1, ... ArgN;
```

DESCRIPTION

This function serves as a System V UNIX®-compatible printf() without floating point. It implements all features of %d, %o, %u, %x, %X %c and %s. An additional control statement has been added to allow printing of binary values (%b).

The *xprintf* and *xsprintf* functions format an argument list according to a control string which indicates the format of the arguments. The function *xprintf* prints the parsed control string to the console while the function *xsprintf* writes the characters to the buffer pointed to be the argument *Buffer*. The control string format is a string that contains plain characters to be processed as is and special characters that are used to indicate the format of the next argument in the argument list. There must be at least as many arguments as special characters, or the function may act unreliably.

Special character sequences are started with the character %. The characters after the % can provide information about left or right adjustment, blank and zero padding, argument conversion type, precision and more things too numerous to list.

If detailed information on the argument formats and argument modifiers is required, seek your local C programmer's manual for details. Not all of the argument formats are supported. The supported formats are %d, %o, %u, %x, %X %c and %s.

SEE ALSO

Appendix B

Code Examples

This appendix contains the example code listed below:

Board.c This file is the catchall for the miscellaneous board-related functions.

Board.h This file describes the hardware addresses and data structures for the board.

Bug.h This file is intended to provide standard constants and data structures common to all files independent of processor, compiler, and board model.

Proc.c This file contains processor-specific functions for interrupt support and exception-handling support.

Proc.h The interrupt wrapper is a relocatable assembly language module that is allocated on the stack. The interrupt table vector location is initialized to point to the wrapper and the wrapper is initialized to point to the interrupt handler. This level of indirection will reduce the necessity for assembly code.

ProcAsm.s This file contains assembly language functions used by the board, monitor, and processor functions to perform processor-specific functions.

RTC.c The function in this file provides low-level real-time clock support for the monitor.

SCC.c The function in this file provides low-level I/O necessary to read, write, and configure the Z85C30 serial controller..

Timer.c This file contains example functions for initializing the CIO counter timers.

VME.c This file contains the functions necessary to initialize the VMEbus as well as examples for performing several basic VME functions.

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"
#include "NvMonDefs.h"

/*****************************************************************************
 * Board.c: This file is the catchall for the miscellaneous board-related
 *          functions. Defined in this module are:
 *
 ***/

extern NV_HkDefined  HKFields;
extern NV_MonDefs    NvMonDefs;
char BoardModel[] = "V3D";

/*****************************************************************************
 * DOCSEC:       ConfigBoard  1   V3D   Board
 *
 * SYNOPSIS:     InitBoard()
 *
 *               ConfigBoard()
 *
 *               ConfigCaches()
 *
 *
 * DESCRIPTION: These functions provide configuration of the boards
 *               interfaces at various points in the monitor. All
 *               of these functions use the NV memory configuration
 *               to determine how to configure an interface so it is
 *               necessary that the NV memory data structures contain
 *               valid data before any of these functions are called.
 *
 *               The InitBoard function initializes the minimum
 *               set of hardware to the default state defined by the
```



```
 *               NV device structures. The hardware initialized is
 *               the serial port, the CIO, and the tracing mechanism.
 *
 *               The function ConfigBoard does a complete
 *               initialization of all the hardware interfaces which
 *               are specified by the NV memory configuration. This
 *               includes the serial ports, VME interface, and
 *               processor caches.
 *               After changing the NV memory configuration using the
 *               NVDisplay or NVSet functions the board can
 *               be fully configured by calling this function.
 *
 *               The ConfigCaches function initializes the processor
 *               caches to either the on or off state as defined by the
 *               NV device configuration.
 *
 *
 ***/

InitBoard()
{
    Delay(10);
    ConfigSerDevs();      /* Initialize serial to default state.       */
    ConfigVmeBus();       /* Initialize VMEBus to default state.       */
    InitCio();
    InitScsi();
    InitTrace();
    return TRUE;
}


ConfigBoard()
{
    Delay(20);            /* Allow all characters to be printed        */
    ConfigSerDevs();      /* Initialize serial to NV specified state.  */
    ConfigVmeBus();       /* Initialize VMEbus to NV specified state.  */
    ConfigScsi();
    ConfigCaches();
    return TRUE;
}


ConfigCaches()
{
    register NV_MonDefPtr Conf = &NvMonDefs;

    if (DataCacheEnble(Conf)) {
        EnbDataCache();
    } else {
        DisDataCache();
    }
    if (InstCacheEnble(Conf)) {
        EnbInstCache();
    } else {
        DisInstCache();
    }
    return TRUE;
}


/*****************************************************************************
 * DOCSEC:       Misc  1   V3D   Board
 *
 * SYNOPSIS:     PrStatus()
```

```
 *
 *              SetLedDisplay(Value)
 *              unsigned long Value;
 *
 *              unsigned char *MemTop()
 *
 *              unsigned char *MemBase()
 *
 *              Delay(HundSec)
 *              int HundSec;
 *
 * DESCRIPTION: This is a collection of miscellaneous board support
 *              functions.
 *
 *              The PrStatus function should print useful
 *              information about the board configuration. Currently
 *              this function determines if the board is configured
 *              as a system controller and determines if a corebus
 *              module is present and what type of module is attached.
 *
 *              The SetLedDisplay function presents the lower
 *              four bits of the argument Value on the user LEDs.
 *
 *              The functions MemTop and MemBase are used to
 *              determine the address of the last and first long word
 *              in free memory. The size of DRAM is determined by the
 *              NV memory configuration. The base of free memory is
 *              determined by the compiler-created variable End
 *              which indicates the end of the monitors bss section.
 *
 *              The Delay function is intended to provide a fixed
 *              delay for timing. It isn't very accurate and depends
 *              widely on whether the caches are enabled or disabled.
 *              As a crude delay generator this function can be used to
 *              delay in increments of 1/100 of a second as specified
 *              by the HundSec argument.
 ***/


PrStatus()
{
    unsigned long Temp;

    xprintf("\nVME System controller -> ");
    if (IsSystemController()) {
        xprintf("On\n");
    } else {
        xprintf("Off\n");
    }
    if (IsModPresent()) {
        ModIDGet(FALSE);
    } else {
        xprintf("No module found\n");
    }
}


PrStatus()
{
    xprintf("PrStatus(): not implemented\n");
    return TRUE;
}


SetLedDisplay(Value)
register unsigned long Value;
```



```
{
    *LED1 = (~Value);
    *LED2 = (~Value >> 1);
    *LED3 = (~Value >> 2);
    *LED4 = (~Value >> 3);
    return TRUE;
}



unsigned char *MemTop()
{
    return((unsigned char *) (RAM_BASE + HKFields.Hardware.DRAMSize - 4));
}



extern unsigned long end[];
unsigned char *MemBase()
{
    return((unsigned char *) end);
}


#define HUND_SEC_DELAY 2000

Delay(HundSec)
int HundSec;
{
    volatile int i;

    for(i=HundSec * HUND_SEC_DELAY; i; i--);
    return TRUE;
}


/*****************************************************************************
 * DOCSEC:      IntrErr  1   V3F   Board
 *
 * SYNOPSIS:    IntrErr(AccAddr, Addr, Vector)
 *              unsigned long AccAddr;
 *              unsigned long Addr;
 *              char Vector;
 *
 *              SetUnExpIntFunct(Funct)
 *              unsigned long Funct;
 *
 * DESCRIPTION: When an unexpected interrupt is received it is necessary to
 *              remove the error condition before returning to the monitor.
 *              This function is called from the function UnExpIntr which
 *              parses the interrupt record for the address and the vector
 *              associated with the interrupt. The device is dealt with
 *              accordingly and the monitor is resumed.
 *
 *              Because the interrupt condition may be a program that
 *              may continually generate exceptions it is
 *              necessary to abort the program and return directly to
 *              the monitor level. This is done by calling the function
 *              RestartMon, which causes the processor to return
 *              into the line editor.
 *
 *              If desired a program can call the SetUnExpIntFunct
 *              function and then attach their own interrupt handler to
 *              all unexpected interrupts. This function attaches the
 *              handler specified by Funct to the unexpected interrupt
 *              handler. The new interrupt handler must determine the
```

```
 *              source of the unexpected interrupt and remove the interrupt.
 *
 ***/

/* Generic resonse messages */
static char ExcErrStr[] = "\n\n^GUnexpected %s Exception at 0x%.8X (Acc at %x)\n";
static char DevIntStr[] = "\n\n^GUnexpected %s Interrupt at 0x%.8X\n";
static char UnkIntStr[] = "\n\n^GUnexpected Interrupt at 0x%.8X (%x) Vector 0x%x\n";

IntrErr(AccAddr, Addr, Vector)
register long AccAddr, Vector;
register char *Addr;
{
    switch(Vector) {
        case BUS_ERROR: {
            xprintf(ExcErrStr,"Bus Error", Addr, AccAddr);
            break;
        }
        case ADDRESS_ERROR: {
            xprintf(ExcErrStr,"Address Error", Addr, AccAddr);
            break;
        }
        case ILLEGAL_INSTR: {
            xprintf(ExcErrStr,"Illegal Instruction", Addr, AccAddr);
            break;
        }
        case ZERO_DIVIDE: {
            xprintf(ExcErrStr,"Zero Divide", Addr, AccAddr);
            break;
        }
        case PRIV_VIOLATION: {
            xprintf(ExcErrStr,"Priv. Violation", Addr, AccAddr);
            break;
        }
        case TRACE_FAULT: {
            xprintf(ExcErrStr,"Trace fault", Addr, AccAddr);
            break;
        }
        case EMULATOR_1010: {
            xprintf(ExcErrStr,"Emul 1010", Addr, AccAddr);
            break;
        }
        case EMULATOR_1111: {
            xprintf(ExcErrStr,"Emul 1111", Addr, AccAddr);
            break;
        }
        case SPURIOUS_INTR: {
            xprintf(ExcErrStr,"Spurious Interrupt", Addr, AccAddr);
            break;
        }
        case PARITY_ERROR: {
            xprintf(ExcErrStr,"Parity Error", Addr, AccAddr);
            break;
        }
        case VSB_VECTOR: {
            xprintf(DevIntStr,"VSB",Addr);
            break;
        }
        case SCSI_VECTOR: {
            ConfigScsi();
            xprintf(DevIntStr,"SCSI",Addr);
            break;
        }
        case CIO_VECTOR: {
            ConfigCio();
            xprintf(DevIntStr,"CIO",Addr);
```



```
            break;
        }
        case SCC_AB_VECTOR:
        case SCC_CD_VECTOR: {
            ConfigSerDevs();
            xprintf(DevIntStr,"SCC",Addr);
            break;
        }
        default: {
            xprintf(UnkIntStr, Addr, AccAddr, Vector);
            break;
        }
    }
    DumpRegs();
    RestartMon();       /* Restart Monitor.*/
}

/*************************************************************************
 * NotSupported: For those commands not supported.
 ***/

NotSupported()
{
    xprintf ("\nThis function is unsupported\n");
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 * AUTHOR
 *
 * MODIFICATIONS:
 *
 *****/

/*****************************************************************************
 * Board.h: This file describes the v3d hardware addresses and data
 *          structures. included in this file are the definitions for:
 *
 *           Z85C36 CIO Counter Timer.
 *           Z85C30 SCC Serial Controller, Ports A-B
 *           WD33C93 SCSI Controller.
 *           DS1216F Realtime clock.
 *           NMI Status Latch.
 *           82596CA Ethernet Controller.
 *           28C64 EEPROM
 *
 ***/

#define MON_REV_LEVEL "1.1"         /* define monitor revision level     */

/*****************************************************************************
 * Interrupt Vector assignments for v3d (68030).
 ***/

#define BUS_ERROR       0x02
#define ADDRESS_ERROR   0x03
#define ILLEGAL_INSTR   0x04
#define ZERO_DIVIDE     0x05
#define PRIV_VIOLATION  0x08
#define TRACE_FAULT     0x09
#define EMULATOR_1010   0x0A
#define EMULATOR_1111   0x0B
#define SPURIOUS_INTR   0x18
#define VSB_VECTOR      0x19
#define SCSI_VECTOR     0x1C
#define PARITY_ERROR    0x1F
```



```
#define CIO_VECTOR      0x90
#define SCC_AB_VECTOR   0xA0
#define SCC_CD_VECTOR   0xB0


/*****************************************************************************
 * DRAM
 ***/

#define    RAM_BASE    0x03000000

/*****************************************************************************
 * CIO:  Definitions for the Z85C36 CIO Counter Timer and parrallel ports
 ***/

#define    CIOPORT     0x02D00001

#define CIO_AData    ((volatile unsigned char *) (CIOPORT + 0x04))
#define CIO_BData    ((volatile unsigned char *) (CIOPORT + 0x02))
#define CIO_CData    ((volatile unsigned char *) (CIOPORT + 0x00))
#define CIO_CTRL     ((volatile unsigned char *) (CIOPORT + 0x06))


/*****************************************************************************
 * SCC:  Definition for the Z85C30 Serial ports A-D.
 ***/

#define SCC_REG_SPREAD  0x03        /* Distance between registers       */
#define SCC_PORT_SPREAD 0x02        /* Distance between ports           */

#define BaudToTimeConst(baud) (((19660800 / (64 * baud)) - 3) / 2)

struct SCCPort {                    /* Serial device structure          */
    unsigned char Control;
    unsigned char Dummy[SCC_REG_SPREAD];
    unsigned char Data;
};                                  /* Define port addresses            */

#define SCC_PORTB   ((struct SCCPort *) 0x02F00001)
#define SCC_PORTA   ((struct SCCPort *) ((int) SCC_PORTB + SCC_PORT_SPREAD))
#define SCC_PORTD   ((struct SCCPort *) 0x02E00001)
#define SCC_PORTC   ((struct SCCPort *) ((int) SCC_PORTD + SCC_PORT_SPREAD))


/*****************************************************************************
 * SCSI:  Definition for the WD33C93 Scsi interface.
 ***/

#define SCSI_ADDR       0x02300001      /* Base Address of SCSI schip   */
#define SCSI_ENABLE     ((unsigned char *) 0x02B00020)
#define SCSI_RESET      ((unsigned char *) 0x02B00006)   /* Bus reset   */

struct SCSIChip {                       /* Define scsi structure        */
    unsigned char SC_AddrPtr;
    unsigned char SC_Dummy[1];
    unsigned char SC_Register;
};                                      /* Define macros to read and write */

#define SCSI ((struct SCSIChip *) SCSI_ADDR)

#define SCWriteReg(Reg, Val)    SCSI->SC_AddrPtr = Reg;\
                                SCSI->SC_Register = Val

#define SCReadReg(Reg, Val)     SCSI->SC_AddrPtr = Reg;\
                                Val = SCSI->SC_Register
```

```
/*************************************************************************
 * SCSI bus interface controller registers
 ***/

#define SREG_OWNID        0x00
#define SREG_CTRL         0x01
#define SREG_TIMEOUT      0x02
#define SREG_TSECT        0x03
#define SREG_THEAD        0x04
#define SREG_TCYLH        0x05
#define SREG_TCYLL        0x06
#define SREG_HH_LADR      0x07
#define SREG_HM_LADR      0x08
#define SREG_LM_LADR      0x09
#define SREG_LL_LADR      0x0A
#define SREG_SECT         0x0B
#define SREG_HEAD         0x0C
#define SREG_CYLH         0x0D
#define SREG_CYLL         0x0E
#define SREG_TLUN         0x0F
#define SREG_CPHASE       0x10
#define SREG_SYNT         0x11
#define SREG_HTCNT        0x12
#define SREG_MTCNT        0x13
#define SREG_LTCNT        0x14
#define SREG_DEST_ID      0x15
#define SREG_SRC_ID       0x16
#define SREG_SCSI_STAT    0x17
#define SREG_CMD          0x18
#define SREG_DATA         0x19

#define SREG_CDB1         0x03
#define SREG_CDB2         0x04
#define SREG_CDB3         0x05
#define SREG_CDB4         0x06
#define SREG_CDB5         0x07
#define SREG_CDB6         0x08
#define SREG_CDB7         0x09
#define SREG_CDB8         0x0A
#define SREG_CDB9         0x0B
#define SREG_CDB10        0x0C
#define SREG_CDB11        0x0D
#define SREG_CDB12        0x0E


#define SCDMA_ADDRESS     0x02400000  /* DMA Acknowledge address          */


/*************************************************************************
 * RTC: Data structures and addresses for the real time clock
 ***/

#define WATCHBASE         ((volatile unsigned char *)  0x00000000)
#define WR0_WATCH         ((volatile unsigned char *) (WATCHBASE + 2))
#define WR1_WATCH         ((volatile unsigned char *) (WATCHBASE + 3))
#define RD_WATCH          ((volatile unsigned char *) (WATCHBASE + 4))

struct rtc_data {         /*  D7  D6  D5  D4  :  D3  D2  D1  D0  */
  unsigned char dotsec;   /* -- 0.1 sec ----- : -- 0.01 sec ----- */
  unsigned char sec;      /* -- 10 sec ------ : -- seconds ------ */
  unsigned char min;      /* ---10 min ------ : -- minutes ------ */
  unsigned char hour;     /* -- A 0 B Hr ---- : -- hours -------- */
  unsigned char weekday;  /* -- 0 0 0 1 ----- : -- day ---------- */
  unsigned char date;     /* -- 10 date------ : -- date --------- */
  unsigned char month;    /* -- 10 Month ---- : -- month -------- */
```



```
  unsigned char year;     /* -- 10 year ----- : -- year --------- */
};



/*************************************************************************
 * VME: Must Write this
 ***/

#define MBOX_BASE           ((unsigned short *) 0x02C00000)
#define ENBL_DOG            ((unsigned char *)  0x02B00030)
#define VME_TIMER           ((unsigned char *)  0x02B00010)
#define SYSFAIL             ((unsigned char *)  0x02B0000E)
#define ENBL_MBOX           ((unsigned char *)  0x02B00004)
#define BUS_LATCH           ((unsigned long *)  0x02A00000)
#define SLAVE_ENABLE        ((unsigned char *)  0x02B0000C)

/*************************************************************************
 * X2212 NVRAM:  Definition for the NV Memory Interface
 ***/

#define NV_BASE         0x02500000      /* Base address of NV memory        */
#define NV_SIZE         0x00000080      /* Size in bytes of NV memory       */
#define NV_PROTECTED    0x00000060      /* Beginning of protected NV memory */
#define NV_MON_DEFS     0x00000028      /* Beginning of monitor NV defs.    */

#define NV_MAX_NBR_WRITES    10000      /* Limit on the number of writes    */
#define NV_PAGE_SIZE         1          /* Page size of 32 for fast program */
#define NV_SPACING           1          /* Number of bytes between bytes    */

#define NV_STORE  ((unsigned char *) 0x02600000)
#define NV_RECALL ((unsigned char *) 0x02700000)
```

```
 # ------------------------------------------------------------------------------
 # This file contains much of the 68030-specific data structures and functions
 # necessary to configure the v3d properly. Many of the processor-specific
 # functions must be configured as seen in this file for the v3d monitor to
 # function reliably.
 #
 # ------------------------------------------------------------------------------

                file  "BoardAsm.s"
                text
                even

                global start_ip
                global ColdStart
                global MonEntryPt
                global end

 # Pause 500 mSec for RAM and then do 8 RAS/CAS cycles to initialize
 # memory.

MonEntryPt:
ColdStart:      mov.l   &0x02B00040, %a0
                mov.l   &0xFFFFFFFF, %d0
                mov.l   %d0, (%a0)                      # Clear LED's
                mov.l   %d0, 0x10(%a0)
                mov.l   %d0, 0x20(%a0)
                mov.l   %d0, 0x30(%a0)

start_ip:       mov.l   &0x000, %d0                     # Counter
                mov.l   &int_table, %a0                 # Any RAM Address
                mov.l   &0x010, %d1                     # Loop Count
RamInit:        mov.l   %d0, (%a0)
                dbra    %d1, RamInit

                mov.l   &0xeeeeeeee, %d0
                mov.l   &int_table, %a0
                mov.l   &end, %a1

ClearSysMem:    mov.l   %d0, (%a0)+
                cmp.l   %a0, %a1
                ble     ClearSysMem

SetState:       mov.l   &sup_stack,%a7      # New supervisory stack

StartMon:       jsr     VectInit            # Initialize Vector Table.
                mov.l   &int_table, %a0     # Link in new table
                mov.l   %a0, %vbr
                jsr     StartMonitor        # Start program.
                rts

                global _warm

_warm:          jsr     VectInit            # Initialize Vector Table.
                bra     SetState


 # ------------------------------------------------------------------------------
 # RestartMon: Reeboots the line editor after reseting the stack pointer.
 # ------------------------------------------------------------------------------

                global  RestartMon

RestartMon:     mov.l   &sup_stack,%a7      # Reset stack pointer
                jsr     LineEdit            # Start program.

 # ------------------------------------------------------------------------------
```



```
 # AtomicAccess: Performs a RMW cycle on the address specified.
 # ------------------------------------------------------------------------------

                global  AtomicAccess

AtomicAccess:   movm.l  &0x0700,-12(%a7)        # save regs
                mov.l   4(%a7),%a0              # memory address
                tas      (%a0)
                movm.l  -12(%sp),&0x0700
                rts

 # ------------------------------------------------------------------------------
 # Powerup detection: The following routines determine powerup conditions and
 #                    allow the user to set the powerup magic number
 # ------------------------------------------------------------------------------

                set   POWER_UP_MAGIC_NUMBER, 0x52364767

                text
                even

                global  IsPowerUp

IsPowerUp:      mov.l   %a6,-(%sp)                      # Save %A6
                mov.l   %d1,-(%sp)
                mov.l   %a1,-(%sp)

                mov.l   %sp,%a6                         # Save sp
                mov.l   &int_table + 0x7C,%a1           # parity vector location
                mov.l   (%a1),%d1                       # Save old parity error
                mov.l   &pu_yes,(%a1)                   # Load new vector

                mov.l   PwrUpLoc,%d0            # Get power up magic
                cmp.l   %d0,&POWER_UP_MAGIC_NUMBER      # Is it right value
                bne     pu_yes                  # If so is power up

                mov.l   &0,%d0                  # return
                bra     pu_exit

pu_yes:         mov.l   &1,%d0                  # Yes is power up

pu_exit:        mov.l   %d1,(%a1)               # Restore old vector
                mov.l   %a6,%sp                 # Restore stack pointer
                mov.l   (%sp)+,%a1              # Restore registers
                mov.l   (%sp)+,%d1
                mov.l   (%sp)+,%a6
                rts

                global  SetNotPowerUp

SetNotPowerUp:  mov.l   &POWER_UP_MAGIC_NUMBER, PwrUpLoc
                rts

 # ------------------------------------------------------------------------------
 # STACK DEFINITIONS: The following data definitions define the stacks for the
 #                    68030. The interrupt, supervisory and user stacks are
 #                    defined. Depending on the application, the size of these
 #                    definitions may be increased or decreased.
 #
 # DATA STRUCTURES:   Space for the interrupt, fault and system procedure
 #                    tables are defined here. The size of these tables is a
 #                    fixed quantity. Details of how these structures are used
 #                    can be found in the 68030 manual. The initialization of
 #                    these structures is performed by other functions.
 #
 # ------------------------------------------------------------------------------
```

```
        even

        global int_table
        global sup_stack
        global PwrUpLoc

        lcomm  int_table, 0x0400
        lcomm  top_stack, 0x4000
        lcomm  sup_stack, 0x40
        lcomm  PwrUpLoc,  0x04
        lcomm  Reserved,  0x0C
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 * AUTHOR
 *      RSS
 *
 * MODIFICATIONS:
 *
 *****/

/*****************************************************************************
 *       Bug.h:  This file is intended to provide standard constants and
 *               data structures common to all files independent of
 *               processor compiler and board model.
 ***/

/*****************************************************************************
 *      Define the constants for TRUE, FALSE, NULL and ERROR.
 ***/

#define NULL    0
#define TRUE    1
#define FALSE   0
#define ERROR   -1

#define FAILED  0
#define PASSED  1

#define   READ          0
#define   WRITE         1
#define   READ_PROBE    2
#define   WRITE_PROBE   3

/*****************************************************************************
 *      Define the constants for BYTE, WORD, and LONG.
 ***/

#define BYTE  1
#define WORD  2
#define LONG  4
```



```
/*****************************************************************************
 *      Define the constants for DECIMAL, HEX, UPPER and LOWER case.
 ***/

#define DECIMAL   0x8
#define HEX       0x4
#define UPPER     0x2
#define LOWER     0x1
#define ALPHA     0x3

/*****************************************************************************
 *      MAXLN is the character limit of the command line editor.
 ***/

#define MAXLN   80

/*****************************************************************************
 *      Character definitions
 ***/

#define   EOF  0
#define   DEL  0x7F
#define   ESC  0x1B
#define   SP   ' '
#define   BS   '\b'
#define   CR   '\r'
#define   LF   '\n'
#define   TAB  '\t'

/*****************************************************************************
 *      Argument structure argc - argv
 ***/

#define MAXARGS 20

struct args {
        char argcount;
        char *argv[MAXARGS];
};


/*****************************************************************************
 *      UNIX style time structure
 ***/

struct tm {
    unsigned long tm_fsec;      /* fractions of seconds (0 - 99) */
    unsigned long tm_sec;       /* seconds (0 - 59) */
    unsigned long tm_min;       /* minutes (0 - 59) */
    unsigned long tm_hour;      /* hours   (0 - 23) */
    unsigned long tm_mday;      /* day of month (1 - 31) */
    unsigned long tm_mon;       /* month of year (0 - 11) */
    unsigned long tm_year;      /* Year - 1900 */
    unsigned long tm_wday;      /* day of week (sunday = 0) */
};

typedef struct tm tm;
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"

/*****************************************************************************
 * CIO.c:  This file contains the functions necessary to read, write and
 *         configure the Z85C36 Counter Timer / parallel port chip.
 *         The functions defined in this module are listed below:
 *
 *              InitCio()          ConfigCio()        StartTimer()
 *              ReadCioPortA()     ReadCioPortB()     ReadCioPortC()
 *              CioIntr()          WriteCioPortB()
 ***/

/*****************************************************************************
 * This file contains all the CIO specific subroutines necessary to reset,
 * initialize, read and write to the CIO ports and counter timers.
 *
 * InitCio():       Sets the CIO to the hardware reset state.
 *
 * ConfigCio():   This is the default state of the CIO and it should be set
 *                  to this state at reset.
 *
 * WriteCioPortA()
 * WriteCioPortB()
 * WriteCioPortC():  These are the routines used to write to ports A-C of
 *                   the CIO.
 *
 */


/*****************************************************************************
 * InitCio(): This function resets the counter timer regardless
 *             what state the chip might be in.
```



```
 ***/

InitCio()
{
    volatile unsigned char *p, c;

    p = CIO_CTRL;
    c = *p;             /* make sure we're waiting for a reg ptr  */
    *p = 0x00;          /* master int ctl reg ptr                 */
    c = *p;             /* (must be a good reason to do it again) */
    *p = 0x00;
    *p = 0x01;          /* reset bit on, off                      */
    *p = 0x00;
}


/*****************************************************************************
 * ConfigCio(): This function initializes the counter timer to the
 *                 state expected by the monitor. The configuration sets
 *                 the parallel ports as bit output ports so that the
 *                 VME slave comparison addresses can be written to ports
 *                 A, B and C.
 ***/

ConfigCio()
{
    static unsigned char ciotable[] = {
                                    /* Port A Initialization */
        0x20,   0x06,               /* bit port, pri encoded vector */
        0x22,   0x9e,               /* Invert neg true bits         */
        0x23,   0xff,               /* all bits inputs */
        0x24,   0x10,               /* ones catcher */
        0x25,   0x10,               /* pattern polarity register */
        0x26,   0x00,               /* all levels */
        0x27,   0x10,               /* pattern mask, enable mailbox */
        0x02,   CIO_VECTOR,         /* base interrupt vector */
        0x08,   0xc0,               /* set int enable, no int on err */
                                    /* Port B Initialization */
        0x28,   0x06,               /* bit port, pri encoded vector */
        0x2a,   0x80,               /* Don't invert inputs */
        0x2b,   0x80,               /* one input bit */
        0x2c,   0x00,               /* normal input (no ones catchers) */
        0x2d,   0x00,               /* bit interrupt on a one */
        0x2e,   0x00,               /* no transition */
        0x2f,   0x00,               /* no interrupts */
        0x03,   CIO_VECTOR          /* set interrupt vector */
        0x09,   0xc0,               /* set int enable, no int on err */
        0x0e,   0xff,               /* set int enable, no int on err */
                                    /* Timer 3 and other CIO initialization */
        0x1e,   0x80,               /* Set mode to auto reload */
        0x1a,   0xff,               /* High byte delay constant */
        0x1b,   0xff,               /* Low byte delay constant */
        0x04,   CIO_VECTOR,         /* Interrupt vector */
        0x08,   0x20,               /* Clear any port A ints */

        0x01,   0x84,   /* enable ports A & B */
        0x00,   0x82    /* enable interrupts */
    };
    register int cnt;
    volatile unsigned char *p;

    InitCio();
    p = CIO_CTRL;
    for(cnt = 0; cnt < sizeof(ciotable); cnt++) {
        *p = ciotable[cnt];
```

```
    }
}

/*************************************************************************
 * WriteCioPortA():
 * WriteCioPortB():
 * WriteCioPortC():  These functions provide the ability to write to the
 *                   CIO output ports. Ports A, B and C are used for the
 *                   VMEbus slave maps for the Extended, Short and Standard
 *                   spaces, respectively.
 ***/

WriteCioPortB(Data)
register unsigned char Data;
{
    *CIO_BData = Data;
}

ReadCioPortA(Data)
register unsigned char Data;
{
    return (*CIO_AData);
}

ReadCioPortB(Data)
register unsigned char Data;
{
    return (*CIO_BData);
}

ReadCioPortC(Data)
register unsigned char Data;
{
    return (*CIO_CData);
}


/*************************************************************************
 * StartTimer():  This function is intended to provide an example of how
 *                to initialize the CIO counter timers. Here the CIO is
 *                initialized, the interrupt handler is attached, and then
 *                the counter is started. In this example the location
 *                'NumTicks' is incremented for every interrupt received
 *                and a dot is printed every second. This function is
 *                turned off by calling ConfigCio() and disconnecting
 *                the interrupt handler.
 ***/

volatile int NumTicks;

StartTimer()
{
    register int cnt;
    register int CioIntr();
    static unsigned char ctitable[] = {
        0x00,    0x82,                  /* Enable master interrupt VIS     */
        0x1E,    0x80,                  /* Channel 3 Continuous           */
        0x1A,    0x82,   0x1B,   0x35,  /* Channel 3 Count (1/60th sec)    */
        0x0C,    0x20,                  /* Clear IP and IUS for channel 3  */
        0x1D,    0x80,                  /* Channel 2 Continuous           */
        0x18,    0x50,   0x19,   0x8A,  /* Channel 2 Count (1/97th sec)    */
        0x0B,    0x20,                  /* Clear IP and IUS for channel 2  */

        0x1C,    0x80,                  /* Channel 1 Continuous           */
        0x16,    0x31,   0x17,   0xC3,  /* Channel 1 Count (1/157th sec)   */
        0x0A,    0x20,                  /* Clear IP and IUS for channel 1  */
```



```
        0x05,    0x00,                  /* Set up port 3   */
        0x06,    0xFF,
        0x07,    0x00,
        0x01,    0x40,                  /* Enable counters 1, 2, and 3     */
        0x0C,    0xC6,                  /* Enable Interrupts, start count  */
        0x0B,    0xC6,
        0x0A,    0xC6
    };

    xprintf("NumTicks loaded at 0x%x\n", &NumTicks);
    ConnectHandler(CIO_VECTOR, CioIntr);
    NumTicks = 0;
    *CIO_CTRL = 0x04;
    *CIO_CTRL = CIO_VECTOR;
    for(cnt = 0; cnt < sizeof(ctitable); cnt++)
        *CIO_CTRL = ctitable[cnt];
    UnMaskInts();
}


/*************************************************************************
 * CioIntr():  This is the interrupt handler for the counter timer.
 *             This function removes the interrupt in the device and
 *             then clears the interrupt in the processor.
 ***/

static CioIntr()
{
    register unsigned char Vector, Status;
    register int i;

    for( i = 0 ; i < 0x1000 ; i++);
    Vector = *CIO_CTRL;
    *CIO_CTRL = 0x04;
    Vector = *CIO_CTRL;

    *CIO_CTRL = 0x0A;
    Status = *CIO_CTRL;
    *CIO_CTRL = 0x0A;
    if ((NumTicks++ % 157) == 0) {
        PutC('.');
    }
    *CIO_CTRL = 0x24;
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 * AUTHOR
 *    RSS
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Proc.h"

/*****************************************************************************
 * DOCSEC:      Exceptions  1   MC68030   Processor
 *
 * SYNOPSIS:    VectInit()
 *
 *              unsigned long *VecToVecAddr(Vector)
 *              unsigned long Vector;
 *
 *              ConnectHandler(Vector, Handler)
 *              unsigned long Vector;
 *              int (*Handler)();
 *
 *              DisConnectHandler(Vector)
 *              unsigned long Vector;
 *
 *              Probe(DirFlag, SizeFlag, Address, Data)
 *              char DirFlag, SizeFlag;
 *              unsigned long Address;
 *              unsigned long Data;
 *
 * DESCRIPTION: These functions are the 68030 processor specific functions
 *              which provide interrupt and exception handling support.
 *
 *              The function VectInit initializes the entire interrupt
 *              table to reference the unexpected interrupt handler. This
 *              assures that the board will not hang when unexpected interrupts
 *              are received. The unexpected interrupt handler saves the state
 *              of the processor at the point the interrupt was detected and
```



```
 *              then calls the IntrErr function, which displays the
 *              error and restarts the monitor.
 *
 *              The function VectToVectAddr converts the argument
 *              Vector to the vector address contained in the
 *              interrupt table associated with the vector. This allows
 *              modification of vectors without knowing where the
 *              interrupt table is located in memory.
 *
 *              The function ConnectHandler allocates an interrupt
 *              wrapper, links the wrapper into the interrupt table and then
 *              initializes the wrapper to call the Handler address. The
 *              argument Vector indicates the vector number to be
 *              connected and the argument Handler should be the
 *              address of the function that will handle the interrupts.
 *              The Interrupt Wrapper is a relocatable assembly language
 *              module that can be placed in free memory and linked into
 *              the interrupt table. This allows the programmer to avoid
 *              using assembly language programming for interrupts.
 *
 *              The function DisConnectHandler modifies the interrupt
 *              table entry associated with Vector to use the unexpected
 *              interrupt handler and then de-allocates the memory used for
 *              the interrupt wrapper allocated by ConnectHandler.
 *              Because both ConnectHandler and DisConnectHandler
 *              use the Malloc and Free facilities it is necessary
 *              for memory management to be initialized.
 *
 *              The function Probe should be used to access memory
 *              locations that may or may not result in a watchdog timeout
 *              or bus error. This function returns TRUE if the location
 *              was accessed and FALSE if the access resulted in a bus
 *              error. The argument DirFlag indicates whether a
 *              read (0) or a write (1) should be attempted. The argument
 *              SizeFlag indicates whether a byte access (1), a
 *              word access (2) or a long access (4) should be attempted.
 *              The argument Address indicates the address to be
 *              accessed and the argument Data is a pointer to
 *              where the read or write data is.
 * SEE ALSO:
 *******/

extern unsigned long int_table[];   /* Address of interrupt table     */

unsigned long *VecToVecAddr(Vector)
unsigned long Vector;
{
   return((unsigned long *) (int_table + Vector));
}


VectInit()
{
  int i, UnExpIntr();
  unsigned long *VectPtr;

  VectPtr = int_table;
  for(i = 0; i < 256; i++) {
    *VectPtr++ = (unsigned long) UnExpIntr;
  }
}

struct IntWrapper IntCode = {
 0x48e7ffff, 0x302f0046, 0x0240f000, 0x0c40b000, 0x66000010, 0x2f7c0000,
 0x00000090, 0x026ffeff, 0x004a302f, 0x0046e488, 0x02800000, 0x00ff222f,
 0x00422f00, 0x2f014eb9,
```

```
 0x00000000,
 0x508f4cdf, 0xffff4e73,
 0x00000000, 0x00000000, 0x00000000
};

ConnectHandler(Vector, Handler)
unsigned long Vector;
int Handler();
{
    unsigned long *CodePtr, *MemPtr;
    struct IntWrapper *Wrapper;
    int i, UnExpIntr();
    unsigned long *VectPtr, *VecToVecAddr();
    unsigned char *Malloc();

    VectPtr = VecToVecAddr(Vector);
    FlushCache();

    if (*VectPtr != (unsigned long) UnExpIntr) {
       Wrapper = (struct IntWrapper *) *VectPtr;
       Wrapper->CallAddr = (unsigned long) Handler;
       return;
    }

    MemPtr = (unsigned long *) Malloc(sizeof(struct IntWrapper));
    CodePtr = (unsigned long *) &IntCode;
    Wrapper = (struct IntWrapper *) MemPtr;

    for (i = 0; i < (sizeof(struct IntWrapper) / sizeof(unsigned long)); i++) {
      *MemPtr++ = *CodePtr++;
    }
    Wrapper->CallAddr = (unsigned long) Handler;

    *VectPtr = (unsigned long) Wrapper;
    FlushCache();
}


DisConnectHandler(Vector)
unsigned long Vector;
{
    unsigned long OldWrapper, *VecToVecAddr();
    int UnExpIntr();

    OldWrapper = *VecToVecAddr(Vector);
    Free(OldWrapper);
    *VecToVecAddr(Vector) = (unsigned long) UnExpIntr;
}

unsigned long BusError;

Probe(DirFlag, SizeFlag, Address, Data)
char DirFlag, SizeFlag;
unsigned long Address;
unsigned long Data;
{
    int Cnt, buserr();
    unsigned long *VectPtr, *VecToVecAddr();
    unsigned long OldVector;

    BusError = FALSE;
    VectPtr = VecToVecAddr(2);
    OldVector = *VectPtr;
    *VectPtr = (unsigned long) buserr;
    switch (DirFlag & 0xDF) {
        case 'R': {
```



```
            switch (SizeFlag & 0xDF) {
                case 'B': {
                    if (!sav_env()) {
                        *(unsigned char *) Data = *(unsigned char *) Address;
                    } else {
                        BusError = TRUE;
                    }
                    break;
                }
                case 'W': {
                    if (!sav_env()) {
                        *(unsigned short *) Data = *(unsigned short *) Address;
                    } else {
                        BusError = TRUE;
                    }
                    break;
                }
                case 'L': {
                    if (!sav_env()) {
                        *(unsigned long *) Data = *(unsigned long *) Address;
                    } else {
                        BusError = TRUE;
                    }
                    break;
                }
                default: {
                    xprintf("error: argument 2 must be -b, -w or -l\n");
                }
            }
            break;
        }
        case 'W': {
            switch (SizeFlag & 0xDF) {
                case 'B': {
                    if (!sav_env()) {
                        *(unsigned char *) Address = *(unsigned char *) Data;
                    } else {
                        BusError = TRUE;
                    }
                    break;
                }
                case 'W': {
                    if (!sav_env()) {
                        *(unsigned short *) Address = *(unsigned short *) Data;
                    } else {
                        BusError = TRUE;
                    }
                    break;
                }
                case 'L': {
                    if (!sav_env()) {
                        *(unsigned long *) Address = *(unsigned long *) Data;
                    } else {
                        BusError = TRUE;
                    }
                    break;
                }
                default: {
                    xprintf("error: argument 2 must be -b, -w or -l\n");
                }
            }
            break;
        }
        default: {
            xprintf("error: argument 1 must be -r or -w\n");
        }
```

```
    }
    Cnt = 0;
    *VectPtr = (unsigned long) OldVector;
    while(BusError == FALSE) {          /* This is strange but it is        */
        if(Cnt++ > 100)                 /* necessary to allow the           */
            return(TRUE);               /* processor to sync up to          */
    }                                   /* handler. Because things may      */
    return(FALSE);                      /* not happen sequentially anymore  */
                                        /* a simple if would execute while  */
}                                       /* a bus error was taking place     */
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

                text
                even

/*****************************************************************************
 * DOCSEC:      Interrupts  1   MC68030   Processor
 *
 * SYNOPSIS:    UnMaskInts()
 *              MaskInts()
 *
 * DESCRIPTION: The functions UnMaskInts and MaskInts are used
 *              to enable and disable interrupts at the processor. The
 *              function UnMaskInts sets the interrupt level bits
 *              in the processor status register to 0 allowing all levels
 *              to interrupt the processor.
 *              The function MaskInts sets the interrupt level bits
 *              in the processor status register to 7 disabling all
 *              interrupts except the non-maskable level 7 interrupt.
 *
 * SEE ALSO:
 *******/

                global    UnMaskInts, MaskInts

UnMaskInts:     and.w     &0xF8FF,%sr         # Clear interrupt levels
                rts                           # Go home

MaskInts:       mov.w     %sr,%d0
                and.l     &0x00000700,%d0
                eor.w     &0x0700,%d0
                or.w      &0x0700,%sr
                rts
```



```
/*****************************************************************************
 * DOCSEC:      Cache  1   MC68030   Processor
 *
 * SYNOPSIS:    FlushCache()
 *              EnbInstCache()
 *              DisInstCache()
 *              EnbDataCache()
 *              DisDataCache()
 *
 * DESCRIPTION: These functions are used to enable, disable and flush
 *              the instruction and data caches.
 *              The FlushCache function flushes both the
 *              instruction and data caches.
 *
 *              The functions EnbInstCache and EnbDataCache
 *              enable the instruction and data caches respective by
 *              turning on the enables is the CACR register.
 *
 *              The functions DisInstCache and DisDataCache
 *              disable the instruction and data caches respective by
 *              turning off the enables in the CACR register. Before
 *              a cache is disabled it is flushed.
 *
 * SEE ALSO:
 *******/

                global    FlushCache
                global    EnbInstCache
                global    DisInstCache
                global    EnbDataCache
                global    DisDataCache

FlushCache:     mov       %cacr, %d0
                or.l      &0x00000808,%d0     # cacr |= (CD | CI);
                bra       ByeBye

EnbInstCache:   mov       %cacr, %d0
                or.l      &0x00000019,%d0     # cacr |= (IBE | CI | EI);
                bra       ByeBye

DisInstCache:   mov       %cacr, %d0
                and.l     &0xFFFFFFE6,%d0     # cacr &= ~(IBE | CI | EI);
                bra       ByeBye

EnbDataCache:   mov       %cacr, %d0
                or.l      &0x00001900,%d0     # cacr |= (DBE | CD | ED);
                bra       ByeBye

DisDataCache:   mov       %cacr, %d0
                and.l     &0xFFFFE6FF,%d0     # cacr &= ~(DBE | CD | ED);

ByeBye:         mov       %d0, %cacr
                rts

/*****************************************************************************
 * DOCSECP:     UnExpIntr  1   MC68030   Processor
 *
 * SYNOPSIS:    UnExpIntr()
 *
 * DESCRIPTION: This is the bad vector routine for catching unexpected
 *              interrupts. If all unused entries in the vector table are
 *              initialized to reference this function then it is not
 *              likely that an errant program can crash the monitor or
 *              an application.
 *
 *              When an unexpected interrupt occurs this function dumps
```

```
 *              the state of the processor registers to a processor
 *              register data structure. After the registers have been saved
 *              the function IntrErr is called, which prints the
 *              exception error message and the register dump before
 *              the command line editor is re-entered.
 *
 * SEE ALSO:
 *******/

                global    UnExpIntr
                global    ProcRegs              # Imported from DumpRegs

UnExpIntr:      movm.l    &0xffff, ProcRegs     # Dump Standard Registers
                movm.l    &0xffff,-(%sp)        # save up state
                mov.w     70(%sp),%d0           # get vector off of stack
                lsr.l     &2,%d0                # divide by 4 to get vec #
                and.l     &0x00ff,%d0           # get rid of non-vector bits
                mov.l     66(%sp),%d1           # Get address of exception
                mov.l     80(%sp),%d2           # Get access address
                mov.l     %d0,-(%sp)            # store vector result in variable
                mov.l     %d1,-(%sp)            # save PC
                mov.l     %d2,-(%sp)            # save Access Addr

                mov.l     &ProcRegs, %a0          # Pointer into control reg file
                mov.l     %d1, 0x40(%a0)          # Save off PC of exception
                mov.l     %sr, 0x64(%a0)          # Save off PC of exception
                mov.l     0x3C(%a0), 0x58(%a0)    # Save SSP = a7
                mov.l     %sfc, %d1
                mov.l     %d1, 0x44(%a0)          # Save SFC
                mov.l     %dfc, %d1
                mov.l     %d1, 0x48(%a0)          # Save DFC
                mov.l     %vbr, %d1
                mov.l     %d1, 0x4C(%a0)          # Save VBR
                mov.l     %cacr, %d1
                mov.l     %d1, 0x50(%a0)          # Save CACR
                mov.l     %caar, %d1
                mov.l     %d1, 0x54(%a0)          # Save CAAR
                mov.l     %isp, %d1
                mov.l     %d1, 0x5C(%a0)          # Save ISP
                mov.l     %msp, %d1
                mov.l     %d1, 0x60(%a0)          # Save MSP

                jsr       IntrErr               # print error message
                jsr       start_ip              # print error message


/*****************************************************************************
 * DOCSEC:      FastFillMem  1   MC68030   Processor
 *
 * SYNOPSIS:    FastFillMem(Value, StartAddress, EndAddress)
 *              unsigned long Value;
 *              unsigned long *StartAddress, *EndAddress;
 *
 * DESCRIPTION: The FastFillMem function provides a fast method
 *              for filling memory with the Value specified.
 *              The FillMem monitor command is too slow to
 *              clear large amounts of memory (megabytes). This
 *              function takes advantage of the burst ability of the
 *              processor, which can achieve much higher data rates
 *              than single reads and writes.
 *
 *              The parameters StartAddress and EndAddress
 *              indicate the start and end of the block of memory to be
 *              filled. The argument Value is the value used to
 *              fill memory. The value is always assumed to be an unsigned
 *              long value and the start and end pointers are assumed to
```



```
 *              be long word aligned addresses.
 *
 * SEE ALSO:
 *******/

                global    FastFillMem

FastFillMem:    movm.l    &0xFFFF,-(%sp)  # Save registers
                mov.l     0x44(%sp),%d0   # Get 'FillValue' off stack
                mov.l     0x48(%sp),%a1   # Get 'Base' off stack
                mov.l     0x4C(%sp),%a0   # Get 'Top' off stack

                mov.l     %d0, %d1        # Copy FillValue to other
                mov.l     %d0, %d2        # registers.
                mov.l     %d0, %d3
                mov.l     %d0, %a3
                mov.l     %d0, %a4
                mov.l     %d0, %a5
                mov.l     %d0, %a6

                mov.l     %a0, %d6        # Copy Top
                sub.l     %a1,%d6         # Count = (Top - Base)
                lsr.l     &5,%d6          # Count = Count / 32;
                sub.l     &1,%d6          # Count = Count - 1;


FillLoop:       movm.l    &0xF01E,-(%a0)  # Move 8 registers at a time.
                dbra      %d6, FillLoop   # Branch till done

CleanUp:        mov.l     %d0,-(%a0)
                cmp.l     %a1,%a0
                blt       CleanUp

                movm.l    (%sp)+,&0xFFFF
                rts

#*****************************************************************************
#   sav_env (env)
#   jmp_buf *env;
#
#   res_env (env, retval)
#   jmp_buf *env;
#   int retval;
#
#   Recover from anticipated bus error
#***

            even
            global    sav_env, res_env, buserr
            global  EnvBuffer

sav_env:    mov.l     &EnvBuffer,%a0    # get pointer to environment buffer
            mov.l     (%sp),(%a0)       # save the pc
            mov.w     %sr,4(%a0)        # and status
            movm.l    &0xFEFE,8(%a0)    # save %D1-%D7/%A1-%A7
            mov.l     &0,%d0            # return false
            rts

buserr:     or.w      &0x0700,%sr       # disable ints
res_env:    mov.l     &EnvBuffer,%a0
            movm.l    8(%a0),&0xFEFE    # restore %D1-%D7/%A1-%A7
            mov.w     4(%a0),%sr
            mov.l     (%a0),(%sp)       # restore %pc to just after sav_env call
            mov.l     &1,%d0            # return true
            rts                         # We magically return via the new PC
```

```
        lcomm   EnvBuffer, 0x50      # Bss Area to save environment
```

```
/*****************************************************************************
 * The Interrupt Wrapper is a relocatable assembly language module that      *
 * Is allocated on the stack. The Interrupt table vector location is         *
 * initialized to point to the wrapper and the wrapper is initialized to     *
 * point to the interrupt handler. This level of indirection will reduce     *
 * the dependency of the test software on the type of processor by           *
 * removing all assembly code from the tests.                                *
 *                                                                           *
 * The assembly language module is included below:                           *
 *                                                                           *
 *                                                                           *
 * Wrapper:     movm.l   &0xffff,-(%sp)     # save cpu state
 *              mov.w    70(%sp),%d0        # get vector off of stack
 *              and.w    &0xf000,%d0        # mask high bits of vector offset
 *              cmp.w    %d0,&0xb000        # Compare to mask of bus exception
 *              bne.s    NotBusErr          # check if bus error
 *              mov.l    &0,114(%sp)        # data for data input buffer
 *              and.w    &0xfeff,74(%sp)    # clear rerun bus cycle bit
 * NotBusErr:   mov.w    70(%sp),%d0        # get vector off of stack
 *              lsr.l    &2,%d0             # divide by 4 to get vec #
 *              and.l    &0x00ff,%d0        # get rid of non-vector bits
 *              mov.l    66(%sp),%d1        # Get address of exception
 *              mov.l    %d0,-(%sp)         # store result in variable
 *              mov.l    %d1,-(%sp)         # push vector
 *              jsr      _IntHdl            # jump to the test
 *              add.l    &8,%sp             # Adjust stack pointer
 *              movm.l   (%sp)+,&0xffff     # restore cpu state
 *              rte                         # Return from exception
 *              space    4                  # Storage for old Vector
 *              space    4                  # Storage for old Vector
 *              space    4                  # Storage for old Vector
 *
 * The basic operation of connecting an interrupt to the vector is           *
 * accomplished by allocating on the stack memory for the wrapper            *
 * copying the wrapper onto the stack, writing the correct call address      *
 * and finally saving the previous Vector pointer in the data space          *
 * allocated.                                                                *
 *                                                                           *
 *************** disassembly for Interrupt Wrapper ***********************
 *
 *       0:  48e7 ffff                       movm.l  &0xffff,-(%sp)
 *       4:  302f 0046                       mov.w   0x46(%sp),%d0
 *       8:  0240 f000                       andi.w  &-4096,%d0
 *       c:  0c40 b000                       cmpi.w  &-20480,%d0
 *      10:  6600 0010                       bne.w   0x10 <22>
 *      14:  2f7c 0000 0000 0090             mov.l   &0, 144(%sp)
 *      1c:  026f feff 004a                  andi.w  &-257, 74(%sp)
 *      22:  302f 0046                       mov.w   0x46(%sp),%d0
 *      26:  e488                            lsr.l   &2,%d0
 *      28:  0280 0000 00ff                  andi.l  &255,%d0
 *      2e:  222f 0042                       mov.l   0x42(%sp),%d1
 *      32:  2f00                            mov.l   %d0,-(%sp)
 *      34:  2f01                            mov.l   %d1,-(%sp)
 *      36:  4eb9 0000 0000                  jsr     0.l
 *      3c:  508f                            addq.l  &8,%sp
 *      3e:  4cdf ffff                       movm.l  (%sp)+,&0xffff
 *      42:  4e73                            rte
 *      44:  4e71                            nop
 *      46:  4e71                            nop
 *      48:  4e71                            nop
 *      4a:  4e71                            nop
 *      4c:  4e71                            nop
 *      4e:  4e71                            nop
 *
 *****************************************************************************/
```



```
struct IntWrapper {
  unsigned long CodeSeg0[14];
  unsigned long CallAddr;
  unsigned long CodeSeg1[2];
  unsigned long DatSeg0[3];
};


/*****************************************************************************
 * Register File definitions for 68030:
 ***/

typedef struct RegFile {
    unsigned long DataRegs[8];
    unsigned long AddrRegs[8];
    unsigned long CtrlRegs[16];
} RegFile, *RegFilePtr;

typedef struct TraceStackFrame {
    unsigned short StatusReg;          /* Status Register     */
    unsigned long  ProgCtr;            /* Next Instruction    */
    unsigned short Vector;             /* Vector Number       */
    unsigned long  InstrAddr;          /* Instruction Address */
} TrStkFrame, *TrStkFramePtr;
```

```
/*************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"

unsigned char Key[8];                   /* bss and data versions of RTC Key  */
static unsigned char InitKey[] = {
    0xC5, 0x3A, 0xA3, 0x5C, 0xC5, 0x3A, 0xA3, 0x5C
};

/*************************************************************************
 * rtc_acc: This function reads or writes the real-time clock, depending
 *          on 'Type'. The 'data' is received and returned in the format
 *          of the real-time clock (Board.h). This function cannot be
 *          loaded into ROM; because of the way the RTC operates, the
 *          clock would be reset by ROM execution.
 ***/

static rtc_acc(data, Type)
register unsigned char *data;
int Type;
{
    register int i, bit;
    volatile unsigned char temp;

    temp = *RD_WATCH;
    for(i = 0; i < 8; i++){
        for(bit = 1; bit & 0xFF; bit <<= 1){
            temp = (Key[i] & bit) ? *WR1_WATCH : *WR0_WATCH;
        }
    }

    if (Type) {
        for(i = 0; i < 8; i++){
            for(bit = 1; bit & 0xFF; bit <<= 1){
```



```
                temp = (data[i] & bit) ? *WR1_WATCH : *WR0_WATCH;
            }
        }
    } else {
        for(i = 0; i < 8; i++){
            data[i] = 0;
            for(bit = 1; bit & 0xFF; bit <<= 1){
                data[i] |= (*RD_WATCH & 1)? bit : 0;
            }
        }
    }
}

/*************************************************************************
 * RtcAcc: This function accepts the structure 'Time' and either reads
 *         the time into or writes the new time from this structure.
 *         'Flag' indicates whether the function is reading or writing
 *         the time. There are several very strange things that should be
 *         described about this function:
 *
 *         Because the RTC stores the time as packed nibbles internally
 *         it is necessary to convert to packed nibbles when writing
 *         and to binary when reading the RTC.
 *
 *         Because the ROM cannot be accessed when the RTC is being read
 *         it is necessary to copy the function rtc_acc into RAM and then
 *         execute the function. This is also why the 'Key' is located in
 *         the 'bss' section. Great care was taken to assure that the
 *         function rtc_acc was relocatable so be careful !!!.
 ***/

RtcAcc(Time, Flag)
register tm *Time;
int Flag;
{
    int (*Funct)();
    int Size, nibble(), rtc_acc();
    char *Malloc();
    register unsigned long tmp;
    register struct rtc_data RtcData;

    CopyMem(InitKey, Key, sizeof(InitKey));

    if (Flag == WRITE) {
        RtcData.hour    = BinToHex(Time->tm_hour);      /* Write */
        RtcData.min     = BinToHex(Time->tm_min);
        RtcData.month   = BinToHex(Time->tm_mon + 1);
        RtcData.weekday = Time->tm_wday | 0x10;
        if (Time->tm_wday == 0)                         /* Converts sunday to 7 */
            RtcData.weekday = 0x17;
        RtcData.date    = BinToHex(Time->tm_mday);
        RtcData.year    = BinToHex(Time->tm_year);
        RtcData.sec     = 0;
        RtcData.dotsec  = 0;
    }

#ifdef RAM_MON                          /* If RAM based monitor          */
    rtc_acc(&RtcData, Flag);
#else                                   /* If EPROM based monitor        */
    Size = (int) RtcAcc - (int) rtc_acc;    /* Size of function to copy      */
    Funct = (int (*)()) Malloc(Size);       /* Allocate memory for function.*/
    FlushCache();
    CopyMem(rtc_acc, Funct, Size);          /* Copy function to memory.      */
    Funct(&RtcData, Flag);                  /* Call function.                */
    Free(Funct);
#endif
```

```
    if (Flag == READ) {
        Time->tm_fsec  = HexToBin(RtcData.dotsec);      /* Read  */
        Time->tm_sec   = HexToBin(RtcData.sec);
        Time->tm_min   = HexToBin(RtcData.min);
        Time->tm_hour  = HexToBin(RtcData.hour);
        Time->tm_mday  = HexToBin(RtcData.date);
        Time->tm_mon   = HexToBin(RtcData.month - 1);
        Time->tm_year  = HexToBin(RtcData.year);
        Time->tm_wday  = (RtcData.weekday & 0x7);
        if (Time->tm_wday == 7)                 /* Converts sunday to 0 */
            Time->tm_wday = 0;
    }
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"
#include "NvMonDefs.h"

/*****************************************************************************
 * SCC.c:  This file contains the functions necessary to read, write and
 *         configure the Z85C30-16 Serial Controller.
 *         The functions defined in this module are listed below:
 *
 *               GetChar()       PutChar()       KeyHit()
 *               TxEmpty()       ChangeBaud()    ConfigSerDevs()
 *               SCCReset()      FoundBreak()    ConfigPort()
 ***/

extern NV_MonDefs    NvMonDefs;      /* Monitor defined configuration    */

volatile unsigned long ConDev;       /* Console Device                   */
volatile unsigned long ModDev;       /* Modem/Download Device            */

static unsigned long SerDevList[] = {   /* List of port assignments      */
    (unsigned long) SCC_PORTA,            /* Corresponds to NV definitions.*/
    (unsigned long) SCC_PORTB,
};

/*****************************************************************************
 * GetChar():  Get a character from specified device 'Port'. This function
 *             is also set up to check for a 'break' and allows the monitor
 *             to perform functions on break, like reset or baud changes.
 *******/

GetChar(Port)
volatile struct SCCPort *Port;
{
```



```
    register unsigned char Data;

    Port->Control = 0;
    while (1) {
        if (Port->Control & 0x01) {
            Data = Port->Data;
            if (Port->Control & 0x80) {
                Port->Control = 0x10;       /* Reset Ext/Status Ints     */
                Port->Control = 0x10;       /* Only works if done twice  */
                FoundBreak(Port);
            } else {
                return(Data);
            }
        }
    }
}

/*****************************************************************************
 * PutChar():  Put a character 'c' to specified device 'Port'
 *******/

PutChar(Port, c)
volatile struct SCCPort *Port;
register char c;
{
    Port->Control = 0;
    while (!(Port->Control &  0x04));
    Port->Data = c;
}

/*****************************************************************************
 * KeyHit():  Check for character on specified device 'Port'. This is
 *            useful during powerup and transparent mode.
 *******/

KeyHit(Port)
volatile struct SCCPort *Port;
{
    Port->Control = 0;
    return(Port->Control & 0x01);
}

/*****************************************************************************
 * TxEmpty():  Check transmitter if empty on specified device 'Port'. This
 *             function is useful for transparent mode.
 *******/

TxEmpty(Port)
volatile struct SCCPort *Port;
{
    return((Port->Control & 0x04) ? TRUE : FALSE);
}

/*****************************************************************************
 * ChangeBaud():  Change baud rate for specified port 'Port' to rate 'Baud'.
 *******/

ChangeBaud(Baud, Port)
volatile struct SCCPort *Port;
register int Baud;
{
    int tc;
    unsigned short dummy;

    for (tc = 0; tc < 0x1000; tc++);
    tc = BaudToTimeConst(Baud);
```

```
    dummy = Port->Control;
    Port->Control = 0x0C;
    Port->Control = tc;
    Port->Control = 0x0D;
    Port->Control = tc >> 8;
    for (tc = 0; tc < 0x1000; tc++);
}


/*****************************************************************************
 * SCCReset():  This function hard resets both ports associated with 'Port'
 *              because it's too clumsy to reset individual ports.
 *******/

static SCCReset(Port)
volatile struct SCCPort *Port;
{
    Port->Control = 0;
    Port->Control = 0x09;
    Port->Control = 0xC0;
}

/*****************************************************************************
 * ConfigSerDevs():  This function uses the current definitions in the
 *                NV structure 'NvMonDefs' to configure the serial ports.
 *                This function is called once when NvMonDefs contains
 *                the default system configuration and once after the
 *                NV memory has been read with the user's configuration.
 *
 * NOTICE:        It is important that the NvMonDefs be valid when this
 *                function is called!
 **/

ConfigSerDevs()
{
    SCCReset(SCC_PORTB);      /* Reset all serial devices.              */

    ConDev = SerDevList[NvMonDefs.Console.PortNum];  /* Set up Console. */
    ConfigPort(ConDev, &NvMonDefs.Console);
    ChangeBaud(NvMonDefs.Console.Baud,  ConDev);

    ModDev = SerDevList[NvMonDefs.DownLoad.PortNum]; /* Set up Download.*/
    ConfigPort(ModDev, &NvMonDefs.DownLoad);
    ChangeBaud(NvMonDefs.DownLoad.Baud, ModDev);
}


/*****************************************************************************
 * ConfigPort():    Initialize specified port 'Port' to the configuration
 *                  specified by 'Conf'. The configurable portion of this
 *                  function includes:
 *
 *                    Data Bits ... 5,6,7 or 8.
 *                    Stop Bits ... 1,or 2.
 *                    Parity    ... None, Even or Odd.
 *                    XOnXOff   ... On/Off
 **/

static ConfigPort(Port, Conf)
volatile struct SCCPort *Port;
register NVU_Port *Conf;
{
    static unsigned char SCCTabl[] = {
        0x09, 0x00,     /* No Reset                          */
        0x0A, 0x00,     /* NRZ                               */
```



```
        0x0B, 0x56,     /* TxClk = RxClk = Baud Rate Gen  */
        0x0E, 0x02,     /* Baud Rate Generator Source     */
        0x0E, 0x03,     /* Start Baud Rate Generator      */
        0x0F, 0x80,     /* Enable interrupt on break      */
        0x01, 0x00,
    };

    register int Cnt;
    register unsigned char Mask;

    for (Cnt = 0; Cnt < 0x1000; Cnt++);

    Port->Control = 0;
    for(Cnt = 0; Cnt < sizeof(SCCTabl); Cnt++)
        Port->Control = SCCTabl[Cnt];

    Mask = 0x0;
    if (Parity(Conf) == SP_PARITY_EVEN)      /* Determine parity.           */
        Mask = 0x3;
    if (Parity(Conf) == SP_PARITY_ODD)
        Mask = 0x1;

    if (StopBits(Conf))                      /* Determine stop bits.        */
        Mask = Mask | 0x08;

    Port->Control = 0x04;                    /* Write register 4            */
    Port->Control = 0x44 | Mask;             /* 16x clock, parity, stop bits */

    Mask = DataBits(Conf);                   /* Determine data bits.        */
    Mask = ((Mask & 0x1) << 1)
        + ((Mask & 0x2) >> 1);
    Port->Control = 0x05;
    Port->Control = (0x8A | (Mask << 5));    /* Set Tx bit size, enable Tx. */
    Mask = Mask << 6;
    if (XOnXOff(Conf))                       /* Turn on auto enables.       */
        Mask = Mask | 0x20;
    Port->Control = 0x03;
    Port->Control = (0x01 | Mask);           /* Set Rx Bit Size, Enable Rx  */

    Port->Control = 0x38;                    /* Reset highest IUS.          */
    Port->Control = 0x30;                    /* Reset errors.               */
    Port->Control = 0x10;                    /* Reset Ext/Status Ints.      */
    for (Cnt = 0; Cnt < 0x1000; Cnt++);
}


/*****************************************************************************
 * FoundBreak():  This function performs functions defined by the NV memory
 *                configuration when a break is received.  Either the monitor
 *                is reset or the baud rate is changed.
 *******/

static FoundBreak(Port)
volatile struct SCCPort *Port;
{
    register NVU_Port *Conf;

    if ((unsigned long) Port == ConDev) {
        Conf = &NvMonDefs.Console;
    } else if ((unsigned long) Port == ModDev) {
        Conf = &NvMonDefs.DownLoad;
    } else {
        return;
    }
```

```
    if (ResetOnBreak(Conf))            /* If reset on break allowed   */
        MonEntryPt();                  /* Reset monitor               */
    if (ChBaudOnBreak(Conf)) {         /* If baud changes on break    */
        Conf->Baud = GetNextBaud(Conf->Baud);
        ChangeBaud(Conf->Baud,  Port);
        xprintf("\nbaud=%d\n", Conf->Baud);
    }
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"
#include "NvMonDefs.h"

extern NV_MonDefs    NvMonDefs;

/*****************************************************************************
 * SCSI.c:  This file contains the functions necessary to read, write and
 *          configure the WD33C93A SCSI Controller.
 *          The functions defined in this module are listed below:
 *
 * InitScsi():  Sets the SCSI to the hardware reset state and removes
 *              the reset interrupt.
 *
 * ConfigScsi(): This sets the state of the SCSI according to the NV
 *               definitions.
 ***/

extern NV_MonDefs    NvMonDefs;      /* Monitor-defined configuration    */

#define SC_RESET    0x00    /* Issues an RESET Command to WD33C93 */
#define FREQ_SEL    0x80    /* Select Frequency for Divisor of 4  */

InitScsi()
{
    register unsigned char Stat;

    MaskInts();                         /* Disable Interrupts.            */
    SCWriteReg(SREG_OWNID, FREQ_SEL);   /* Initailize for 16MHZ operation.*/
    SCReadReg(SREG_SCSI_STAT, Stat);    /* Read Status register.          */
    SCWriteReg(SREG_CMD, SC_RESET);     /* Generate SCSI Reset.           */
    SCReadReg(SREG_SCSI_STAT, Stat);    /* Remove SCSI Interrupt.         */
}
```



```
ConfigScsi()
{
    register unsigned char Stat;
    register NV_MonDefPtr Conf = &NvMonDefs;

    InitScsi();
    if (ScsiResetEnbl(Conf)) {          /* Reset SCSI on reset ?  */
        *SCSI_RESET = 1;                /* Toggle the reset line. */
        Delay(100);                     /* Leave on ~ 1 second.   */
        *SCSI_RESET = 0;                /* Remove SCSI reset.     */
    }
    if (ScsiIntMask(Conf)) {            /* SCSI interrupt mask ?  */
        *SCSI_ENABLE = 0;               /* Disable SCSI Interrupt */
    } else {
        *SCSI_ENABLE = 1;               /* Enable SCSI Interrupt  */
    }
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"
#include "NvMonDefs.h"

/*****************************************************************************
 * VME.c:  This file contains the functions necessary to initialize the
 *         VMEbus as well as examples of how to perform several basic
 *         VME functions.
 *
 *               ConfigBus()     WrBusLatch()
 ***/

extern NV_MonDefs    NvMonDefs;     /* NV Monitor definitions           */

/*****************************************************************************
 * ConfigVmeBus():  This function uses the current definitions in the
 *                  NV structure 'NvMonDefs' to configure the VME bus.
 *                  This function is called once when NvMonDefs contains
 *                  the defualt system configuration and once after the
 *                  NV memory has been read with the users configuration.
 *
 *                  Configured in the function are the following:
 *
 *                    Extended Space  ... Address and Enable
 *                    Standard Space  ... Address and Enable
 *                    Short    Space  ... Address and Enable
 *                    Bus Req Level   ... BR3, BR2, BR1, BR0
 *                    Bus Rel Modes   ... WhenDone, OnReq, OnClear, Never
 *                    Local Bus Timer ... 4us to Infinite
 *                    VME Bus Timer   ... 4us to Infinite
 *                    Arbiter Mode    ... RoundRobin, Priority
 *                    Write Post Slv  ... On/Off
 *                    Write Post Mst  ... On/Off
```



```
 *                    Turbo mode      ... On/Off
 *                    Sys Fail State  ... On/Off
 *                    Indiv R-Mod-Wr  ... On/Off
 *
 * NOTICE:           It is important that the NvMonDefs be valid when this
 *                   function is called!
 ***/

ConfigVmeBus()
{
    register NVU_BusConfig *Conf = &NvMonDefs.VmeBus;
    register unsigned long BusVal;


    if (EnblSht(Conf)) {
        *MBOX_BASE = ShtSlaveMap(Conf);
        *ENBL_MBOX = 1;
    } else {
        *ENBL_MBOX = 0;
    }
    if (Sysfail(Conf)) {
        *SYSFAIL = 0;
    } else {
        *SYSFAIL = 1;
    }
    if (LocBusTimer(Conf)) {
        *ENBL_DOG = 0;
    } else {
        *ENBL_DOG = 1;
    }
    if (VmeBusTimer(Conf)) {
        *VME_TIMER = 1;
    } else {
        *VME_TIMER = 0;
    }

    *SLAVE_ENABLE = 0;
     BusVal = ((ExtSlaveMap(Conf)   >> 24) & 0x000000FF)
            + ((StdSlaveMap(Conf)   >> 12) & 0x00000F00)
            + ((ReplaceAddr(Conf)   >> 8)  & 0x0000F000)
            + ((MastRelMode(Conf)   << 16) & 0x00030000)
            + ((SlaveRelMode(Conf)  << 16) & 0x00040000)
            + ((Conf->AddrModSel    << 19) & 0x00380000)
            + ((IndivRMC(Conf)      << 16) & 0x00400000);
    WrBusLatch(BusVal);
    if (EnblSlave(Conf)) {
        *SLAVE_ENABLE = 1;
    }
}

WrBusLatch(value)
register unsigned long value;
{
    int i;

    for (i = 0; i < 8; i++){
        *BUS_LATCH = (value >> i);
    }
}
```

Appendix C

NVRAM Information

The NVRAM memory is a 128-byte EEPROM that contains manufacturing, service, and hardware configuration information; monitor and board initialization information; and user-defined information. The start address, size, and description of the device are given in Table C-1:

TABLE C-1
EEPROM addresses

Device Address	Byte Offsets	Data
$0270,0000_{16}$	$0 - 15FF_{16}$	User-defined data area
$0270,B000_{16}$	$1600_{16} - 17FF_{16}$	Monitor/board initialization
$0270,C000_{16}$	$1800_{16} - 1FFF_{16}$	Manufacturing/service hardware information

This appendix contains the following files:

NV.c This file contains the functions necessary to read, write, and configure the EEPROM.

NVAssign.h This header file defines the bit field assignments for the NVRAM/EEPROM, as they are defined by Heurikon.

NVDefs.h This header file includes the basic error codes and the codes passed to NVOp to indicate the type of operations to perform on nonvolatile memory.

NVLib.c This file contains the nonvolatile library functions used to manage NVRAM or EEPROM.

NvMonDefs.h This header file defines the bit field assignments for the NVRAM/EEPROM, as they are defined by the board.

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "Bug.h"
#include "Board.h"
#include "NvMonDefs.h"

extern NV_HkDefined  HKFields;
extern NV_MonDefs    NvMonDefs;

/*****************************************************************************
/*****************************************************************************
 * DOCSECP:      nv_recall  1  V3D  Board
 *
 * SYNOPSIS:     nv_recall()
 *               nv_store()
 *
 * DESCRIPTION:  These functions perform the store and recall operation
 *               for NVRAM devices. On some boards which use EEPROM
 *               as NV memory instead of NVRAM these functions are empty
 *               and must be defined to provide compatibility.
 *
 * ALGORITHM:    On boards which have NVRAM it is necessary to install
 *               software delays after the store and recall operations.
 ***/

nv_recall()
{
    Delay(20);
    return (*NV_RECALL);
}

nv_store()
{
    Delay(100);
    return(*NV_STORE);
```



```
}


/*****************************************************************************
 * DOCSECP:      NVRMaxNbrWrites  1  V3D  Board
 *
 * SYNOPSIS:     NVRMaxNbrWrites()
 *
 * DESCRIPTION:  This function returns the number of writes that the
 *               NV memory device is rated for. This allows the NV
 *               memory libraries to determine the lifetime of a
 *               component without including the board header file.
 ***/

NVRMaxNbrWrites()  {                /* Returns limit of write count */
    return(NV_MAX_NBR_WRITES);
}

/*****************************************************************************
 * DOCSEC:       NvHkOffset  1  V3D  Board
 *
 * SYNOPSIS:     NvHkOffset()
 *               NvMonOffset()
 *               NvMonSize()
 *               NvMonAddr()
 *
 * DESCRIPTION:  These functions allow the NV library functions to operate
 *               on the NV memory sections without actually compiling the
 *               board config files into the library. This is desirable
 *               because they will change from board to board.
 *
 *               The NvHkOffset and NvMonOffset functions
 *               describe where in the NV memory device the Heurikon and
 *               monitor defined data sections begin.
 *               In general the Heurikon defined data section
 *               resides in a hardware write protected region and the monitor
 *               data section resides in the user writable section of the NV
 *               memory device. The returned value is the offset in bytes
 *               from the beginning of the device in which the section
 *               is loaded.
 *
 *               The functions NVMonSize and NVMonAddr return the
 *               size and location of the NV monitor configuration data
 *               structure. This again allows other monitor facilities and
 *               application programs to get at the monitor configuration
 *               structure without having to know too much about the monitor.
 ***/


NvHkOffset()  {
    return(NV_PROTECTED);
}

NvMonOffset()  {
    return(NV_MON_DEFS);
}

NvMonSize()  {
    return(sizeof(NV_MonDefs));
}

NvMonAddr()  {
    return( (int) &NvMonDefs);
}
```

```
/*************************************************************************
 * DOCSEC:      NvRamAcc  1  V3D  Board
 *
 * SYNOPSIS:    unsigned char NVRamAcc(Mode, Cnt, Val)
 *              unsigned long Mode, Cnt;
 *              unsigned char *Val;
 *
 * DESCRIPTION: These functions provide the physical interface to the
 *              board NV memory device and the module configuration space
 *              device. The Mode indicates one of four access types.
 *              The four modes are READ, READ_PROBE, WRITE and WRITE_PROBE.
 *              The probe modes perform reads and writes which can
 *              recover from bus errors. This is necessary because some
 *              boards generate a bus error when attempting to write
 *              a protected data area and a bus error is generated when no
 *              module is installed.
 *
 *              The Cnt indicates the byte location to be modified and
 *              assumes the NV memory is a linear array of memory locations.
 *              If there are gaps between bytes on the physical device they
 *              are dealt with here. The last parameter Val is a pointer
 *              to the character location to be written.
 *
 *              Returned from this function is the number of bytes
 *              written to the device or the value read from the device
 *              depending on Mode  This function supports bursts on
 *              writes to speed the storing of data around 32 times.
 *              The burst size is determined by NV_PAGE_SIZE. Another
 *              optimization is that only bytes that differ are written.
 ***/

unsigned char NVRamAcc(Mode, Cnt, Val)
register unsigned long Mode, Cnt;
register unsigned char *Val;
{
    register unsigned char *NVLoc;
    register unsigned char RamVal;

    NVLoc = (unsigned char *) (NV_BASE + (NV_SPACING * Cnt * 2));

    if (Mode == READ) {
                RamVal = ((NVLoc[0] & 0x0F) << 4) + (NVLoc[1] & 0x0F);
        return(RamVal);
    } else {
        NVLoc[0] = (*Val >> 4);
        NVLoc[1] = *Val;
        return(NV_PAGE_SIZE);
    }
}

unsigned char ModConfAcc(Mode, Cnt, Val)
register unsigned long Mode, Cnt;
register unsigned char *Val;
{
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *        The copyright notice above does not evidence any
 *        actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

/*****************************************************************************
 *  NVAssign.h: This header file defines the bit field assignments
 *              for the NVRAM/EEPROM, as they are defined by Heurikon.
 *              It can be used where a program needs to know which bit fields
 *              are assigned to what.
 *              Note that the memory is divided into two separate sections:
 *              the Heurikon-defined, or write-protected, region and the
 *              user-defined region that can be modified interactively
 *              from the monitor or external programs.
 *
 * NOTICE:      Because different compilers may generate different spacing
 *              between structures and structure elements based on the
 *              alignment it is important to be careful defining structures.
 *              Problems can be avoided by forcing shorts and longs onto
 *              long and short boundaries and padding structures to be
 *              a multiple of long words in size.
 *
 *    NOTE:     The definition 'NV_SMALL' is intended to conserve space
 *              for smaller NV devices, which can be as small as 128 bytes.
 *
 ***/

/******* INTERNAL BIT DEFINITIONS ***********************************
 *  This structure provides the internal structures necessary to
 *  maintain a nonvolatile section of memory. The magic number is
 *  used to quickly determine if the structure has been initialized.
 *  The checksum is used to verify the validity of the data. The
 *  write count indicates the number of times the section has been
 *  written and provides an indicator of the lifetime of the component.
 *
 *  This structure must be the first entry in a nonvolatile section.
 *  Many of the functions that manipulate nonvolative sections assume that
 *  this is the first structure in the section and will not function
```



```
 *  if it is omitted.
 ***/

typedef struct NV_Internal {        /* Internal structure = 8 bytes   */
    unsigned short Magic;           /* Magic number                   */
    unsigned short WriteCnt;        /* Write Count                    */
    unsigned long  ChkSum;          /* CheckSum                       */
} NV_Internal, *NV_InternalPtr;

#define NV_MAGIC        0x57CE      /* Magic number for nv memory     */

/******* BOARD BIT DEFINITIONS ***************************************
 * The Manufacturing structure provides information necessary to
 * track the board's manufacturing history, revision, ship date, etc.
 * This structure is located in the write-protected region of the
 * nonvolatile memory device. Modification should only be done
 * by Heurikon's manufacturing departement.
 ***/

typedef struct NVH_Manufacturing { /* Manuf struct = 88/8 bytes       */
    unsigned char  Revision;        /* Board Revision                 */
    unsigned char  ECOLevel;        /* Board ECO Level                */
    unsigned short SerialNumber;    /* Board Serial Number            */
#ifndef NV_SMALL
    char Model[8];                  /* Board Model                    */
    char ManDate[12];               /* Manufacturing Date             */
    char ManPartNum[12];            /* Manufacturing Part Number      */
    char WorkOrderNum[12];          /* Work Order Number              */
    char Reserved[40];
#endif
} NVH_Manufacturing;

/******* SERVICE DEFINITIONS *****************************************
 * This structure provides the service record of the board. This
 * structure consists of the RMA number, Ship Date, Technician name
 * and a short description of the problem. The last 3 records are
 * allowed to be stored in nonvolatile memory.
 ***/

typedef struct NVH_ServRec {        /* ServRec Struct = 72 bytes      */
    char RecNum[12];                /* Service Record Number          */
    char Date[12];                  /* Service Record Date            */
    char Tech[8];                   /* Service Record Technician      */
    char Problem[40];               /* Service Record Technician      */
} NVH_ServRec;

typedef struct NVH_Service {        /* Service Struct = 232 bytes     */
    NVH_ServRec Rec[3];             /* Storage for the last three     */
    char Reserved[16];              /* service records                */
} NVH_Service;

/******* HARDWARE DEFINITIONS ****************************************
 * Board Hardware definitions are provided by this structure, which
 * describes memory sizes and peripheral configuration.
 ***/

typedef struct NVH_Hardware {       /* Hardware Struct = 36/24 bytes  */
    unsigned char MPUType;          /* Processor Type                 */
    unsigned char MMUType;          /* MMU Type                       */
    unsigned char CacheType;        /* Cache Type                     */
    unsigned char FPUType;          /* Floating Point Type            */
    unsigned char DMAType;          /* DMA Type                       */
    unsigned char MemExpType;       /* Memory Expansion Type          */
    unsigned char DiskType;         /* Hard Disk Controller Type      */
    unsigned char TapeType;         /* Streaming Tape Type            */
#ifndef NV_SMALL
```

```
    unsigned char EthernetType;    /* Ethernet Controller Type          */
    unsigned char Padding[3];
#endif
    unsigned long DRAMSize;        /* Dynamic RAM Size                  */
    unsigned long SRAMSize;        /* Static RAM Size                   */
    unsigned long NVMemSize;       /* Nonvolatile memory size           */
#ifndef NV_SMALL
    char Reserved[12];
#endif
} NVH_Hardware;

#define RAMSIZ_0        0x00000000
#define RAMSIZ_128      0x00000080
#define RAMSIZ_1K       0x00000400
#define RAMSIZ_8K       0x00002000
#define RAMSIZ_16K      0x00004000
#define RAMSIZ_32K      0x00008000
#define RAMSIZ_64K      0x00010000
#define RAMSIZ_128K     0x00020000
#define RAMSIZ_256K     0x00040000
#define RAMSIZ_512K     0x00080000
#define RAMSIZ_1M       0x00100000
#define RAMSIZ_2M       0x00200000
#define RAMSIZ_3M       0x00300000
#define RAMSIZ_4M       0x00400000
#define RAMSIZ_8M       0x00800000
#define RAMSIZ_12M      0x00C00000
#define RAMSIZ_16M      0x01000000
#define RAMSIZ_32M      0x02000000
#define RAMSIZ_64M      0x04000000

/******* COMBINED HEURIKON DEFINED VALUES ********************************
 *  The combination of the hardware, manufacturing record and the service
 *  record are bound together in this structure, which is stored in the
 *  protected region of the nonvolatile memory.
 ***/

typedef struct NV_HkDefined {          /* Hk struct = 40/444 bytes      */
    NV_Internal         Internal;      /* Internal definitions          */
    NVH_Hardware        Hardware;      /* Hardware definitions          */
    NVH_Manufacturing   Manuf;         /* Manuf definitions             */
#ifndef NV_SMALL
    NVH_Service         Service;       /* Service record                */
#endif
} NV_HkDefined;
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved .
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

/*****************************************************************************
 *      NvDefs.h:  This header file includes the basic error codes and the
 *             codes passed to NVOp to indicate the type of operations
 *             to perform on nonvolatile memory.
 ***/

/*****************************************************************************
 *      The Error flags are defined below. Note that these error codes have
 *  been used to construct error tables and must not be modified for
 *  any reason.
 */

#define NVE_NONE            0   /* No error                              */
#define NVE_OVERFLOW        1   /* Warning: Too many writes done         */
#define NVE_MAGIC           2   /* Bad magic number in NVRAM image       */
#define NVE_CKSUM           3   /* Bad checksum in NVRAM image           */
#define NVE_STORE           4   /* Could not write NVRAM to memory       */
#define NVE_CMD             5   /* Unknown command requested             */
#define NVE_CMP             6   /* Data does not compare to NVRAM        */

                                /* On Board Non Volatile memory cmds.   */
#define NV_OP_FIX           0     /* NVOp Command to fix checksum         */
#define NV_OP_CLEAR         1     /* NVOp Command to clear NV section     */
#define NV_OP_CK            2     /* NVOp to checksum NV sections         */
#define NV_OP_OPEN          3     /* NvOp to Open NV Section              */
#define NV_OP_SAVE          4     /* NvOp to Save NV Section              */
#define NV_OP_CMP           5     /* NvOp to Compare NV Section           */

                                /* Module configuration space commands  */
#define NV_OP_MCS_FIX      10     /* NVOp Command to fix checksum         */
#define NV_OP_MCS_CLEAR    11     /* NVOp Command to clear NV section     */
#define NV_OP_MCS_CK       12     /* NVOp to checksum NV sections         */
#define NV_OP_MCS_OPEN     13     /* NvOp to Open NV Section              */
```



```
#define NV_OP_MCS_SAVE     14     /* NvOp to Save NV Section              */
#define NV_OP_MCS_CMP      15     /* NvOp to Compare NV Section           */
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 * AUTHOR
 *    RSS
 *
 * MODIFICATIONS:
 *
 *****/

#include     "Bug.h"
#include     "NVDefs.h"
#include     "NVAssign.h"
#include     "NVFields.h"


extern NVGroup       NVGroups[];    /* NV memory groupings structure     */
extern NV_HkDefined  HKFields;      /* Heurikon defined structure        */

/*****************************************************************************
 * NV Error Strings():
 *
 * NOTE:  The Error table strings are defined according to the
 *        definitions in 'NVDefs.h'. Neither of these files should be
 *        modified without fear of complete disaster.
 ****/

static char NVErr0Str[] = "No error";
static char NVErr1Str[] = "Maximum write count exceeded";
static char NVErr2Str[] = "Bad magic number";
static char NVErr3Str[] = "Illegal checksum";
static char NVErr4Str[] = "Write to NV memory does not verify";
static char NVErr5Str[] = "Unknown command";

char *NVErrTable[] = {
    NVErr0Str,  NVErr1Str,  NVErr2Str,  NVErr3Str,  NVErr4Str,
    NVErr5Str
};

/*****************************************************************************
 *  String definitions for error reporting.
```



```
 ***/

static char NVSupStr1[] = "\nError while %s NV memory. %s.";
static char NVSupStr2[] = "\nWarning protected region of NV memory %s.";

static char NVSetStr1[]    = "\nError: %s '%s' not found\n";
static char NVSetStr2[]    = "\nThis field only accepts %s values";
static char NVSetStr3[]    = "\nIllegal field selection try ...";
static char NVSetStr4[]    = "\nThis field limited to %d characters";

static char GroupStr[]     = "Group";
static char FieldStr[]     = "Field";
static char DecimalStr[]   = "Decimal";
static char HexStr[]       = "Hex";

/*****************************************************************************
 * DOCSEC:       NVMemory  1   Std    Monitor
 *
 * SYNOPSIS:     NVDisplay()
 *
 *               NVUpdate()
 *
 *               NVOpen()
 *
 *               NVSet(GroupName, FieldName, Value)
 *               char *GroupName, *FieldName, *Value;
 *
 *               NVInit(SerNum, RevLev, ECOLev, Writes)
 *               int SerNum, ECOLev, RevLev, Writes;
 *
 * DESCRIPTION: The NV memory support functions provide the interface
 *               to the NV memory. All of these functions deal only
 *               with the monitor- and Heurikon-defined sections of the
 *               NV memory. The monitor-defined sections of NV memory
 *               are read/write and can be modified by the user.
 *               The Heurikon-defined section of NV memory is read only
 *               and cannot be modified. Attempts to modify the Heurikon
 *               defined sections will result in an error message when
 *               the store is done.
 *
 *               The NVOpen function reads and checks the monitor
 *               and Heurikon-defined sections. If the NV sections do
 *               not validate, then an error message is displayed.
 *
 *               The NVUpdate function attempts to write the Heurikon-
 *               and monitor-defined NV sections back to NV memory.
 *               The data are first verified, and then written to the
 *               device. The write is verified and all errors are reported.
 *
 *               The NVInit function is used to initialize the NV memory
 *               to the default state defined by the monitor. It first
 *               clears the memory and then writes the Heurikon and
 *               monitor data back to NV memory. This function accepts
 *               as arguments the serial number, revision level, ECO
 *               level and the number of writes to NV Memory. If the
 *               monitor-defined NV memory section somehow becomes corrupt,
 *               the command sequence NVInit followed by NVUpdate
 *               should result in the monitor-defined NV memory resetting
 *               to the default state. This sequence of commands will result
 *               in error messsages that indicate the Heurikon-defined section
 *               was not changed. These messages can be ignored.
 *
 *               The NVDisplay and NVSet commands are used to
 *               display and modify the Heurikon-defined and monitor-defined
 *               NV sections. The values are displayed in logical groups.
 *               Each group has a number of fields. Fields are displayed
```

```
 *              as hex, decimal, or a list of legal values. An example of
 *              the display is shown below:
 *
 *              Group 'Console'
 *              Port            A           (A, B, C, D)
 *              Baud            9600
 *              Parity          None        (Even, Odd, None)
 *              Data            8-Bits      (5-Bits, 6-Bits, 7-Bits, 8-Bits)
 *              StopBits        2-Bits      (1-Bit, 2-Bits)
 *
 *              After each group is displayed, the user has the option of moving
 *              to the next group display, editing the current group display,
 *              or quitting the display completely. If an edit is requested, all
 *              fields of the group are prompted for modification one-by-one.
 *              An empty line indicates that no modification is necessary.
 *
 *              To modify a field using NVSet, the group and field to be
 *              modified are specified and the new value is provided. This
 *              command allows abbreviation of the field and group names.
 *              The NVDisplay function allows fields to be changed
 *              interactively during the display.
 *
 ***/


NVDisplay()
{
    int RetVal, i, Err;
    NV_Internal  *NvMon     = (NV_Internal *) NvMonAddr();
    unsigned long NvMonSiz  = NvMonSize();

    Err = NVOp(NV_OP_CK, NvMon, NvMonSiz);
    if (Err != NVE_NONE) {
        xprintf(NVSupStr1, "reading", NVErrTable[Err]);
        return;
    }

    xprintf("\nNon-Volatile Memory Configuration Display");
    xprintf("\n----------------------------------------");

    for (i = 0; i < NumGroups() ; i++ ) {
        xprintf("\nGroup '%s'\n", NVGroups[i].GroupName);
        DispGroup(&NVGroups[i], FALSE);
        if ((RetVal = Continue()) == ESC) {
            return;
        }
        if (RetVal != CR) {
            xprintf("\nGroup '%s'\n", NVGroups[i].GroupName);
            DispGroup(&NVGroups[i], TRUE);
            NVOp(NV_OP_FIX, &HKFields, sizeof(NV_HkDefined));
            NVOp(NV_OP_FIX, NvMon, NvMonSiz);
            i--;
        }
    }
}


NVUpdate()
{
    register int Err;
    NV_Internal  *NvMon     = (NV_Internal *) NvMonAddr();
    unsigned long NvMonSiz  = NvMonSize();
    unsigned long NvMonOff  = NvMonOffset();
    unsigned long NvHkOff   = NvHkOffset();
```



```
    NvMon->WriteCnt++;
    Err = NVOp(NV_OP_CK, NvMon, NvMonSiz);
    if (Err != NVE_NONE) {
        xprintf(NVSupStr1, "reading", NVErrTable[Err]);
        return;
    }
    Err = NVOp(NV_OP_SAVE, NvMon, NvMonSiz, NvMonOff);
    if (Err != NVE_NONE) {
        xprintf(NVSupStr1, "storing", NVErrTable[Err]);
        return;
    }


    HKFields.Internal.WriteCnt++;
    NVOp(NV_OP_CK, &HKFields, sizeof(NV_HkDefined));
    Err = NVOp(NV_OP_SAVE, &HKFields, sizeof(NV_HkDefined)  , NvHkOff);
    if (Err != NVE_NONE) {
        HKFields.Internal.WriteCnt--;   /* Maybe write protected */
        Err = NVOp(NV_OP_CMP, &HKFields, sizeof(NV_HkDefined)  , NvHkOff);
        if (Err != NVE_NONE) {
            xprintf(NVSupStr2, "was not modified");
        }
        return;
    }
}


NVOpen()
{
    register int Err;
    NV_Internal  *NvMon     = (NV_Internal *) NvMonAddr();
    unsigned long NvMonSiz  = NvMonSize();
    unsigned long NvMonOff  = NvMonOffset();
    unsigned long NvHkOff   = NvHkOffset();


    NVOp(NV_OP_OPEN , &HKFields,   sizeof(NV_HkDefined)  , NvHkOff);
    NVOp(NV_OP_OPEN , NvMon, NvMonSiz, NvMonOff);
    Err = NVOp(NV_OP_CK, &HKFields, sizeof(NV_HkDefined));
    if (Err != NVE_NONE) {
        xprintf(NVSupStr2, "is corrupt");
    }
    Err = NVOp(NV_OP_CK, NvMon, NvMonSiz);
    if (Err != NVE_NONE) {
        xprintf(NVSupStr1, "reading", NVErrTable[Err]);
    }
    return Err;
}


NVSet(GroupName, FieldName, Value)
char *GroupName, *FieldName, *Value;
{
    int Err;
    NVGroupPtr Group, FindGroup();
    NV_Internal  *NvMon     = (NV_Internal *) NvMonAddr();
    unsigned long NvMonSiz  = NvMonSize();
    unsigned long NvMonOff  = NvMonOffset();

    if ((GroupName == NULL) || (FieldName == NULL)) {
        xprintf("Both a Group and Field must be specified\n");
        return;
    }
    if ((Group = FindGroup(NVGroups, NumGroups(), GroupName)) == NULL) {
        xprintf(NVSetStr1, GroupStr, GroupName);
        return;
```

```
    }

    if ((Err = NVOp(NV_OP_CK, NvMon, NvMonSiz)) != NVE_NONE) {
        xprintf(NVSupStr1, "reading", NVErrTable[Err]);
        return;
    }

    SetField(Group, FieldName, Value, TRUE);

    NVOp(NV_OP_FIX, &HKFields, sizeof(NV_HkDefined));
    NVOp(NV_OP_FIX, NvMon, NvMonSiz);
}


NVInit(SerNum, RevLev, ECOLev, Writes)
int SerNum, ECOLev, RevLev, Writes;
{
    register int Err;
    NV_Internal  *NvMon     = (NV_Internal *) NvMonAddr();
    unsigned long NvMonSiz  = NvMonSize();
    unsigned long NvMonOff  = NvMonOffset();
    unsigned long NvHkOff   = NvHkOffset();

    NVOp(NV_OP_CLEAR, &HKFields,   sizeof(NV_HkDefined)  );
    NVOp(NV_OP_CLEAR, NvMon, NvMonSiz);
    NVOp(NV_OP_SAVE , &HKFields,   sizeof(NV_HkDefined)  , NvHkOff);
    NVOp(NV_OP_SAVE , NvMon, NvMonSiz, NvMonOff);
    NVOp(NV_OP_OPEN , &HKFields,   sizeof(NV_HkDefined)  , NvHkOff);
    NVOp(NV_OP_OPEN , NvMon, NvMonSiz, NvMonOff);

    SetNvDefaults(NVGroups, NumGroups());

    HKFields.Manuf.Revision     = RevLev;
    HKFields.Manuf.SerialNumber = SerNum;
    HKFields.Manuf.ECOLevel     = ECOLev;

    HKFields.Internal.WriteCnt   = Writes;
    NvMon->WriteCnt = Writes;

    NVOp(NV_OP_FIX, &HKFields,   sizeof(NV_HkDefined));
    NVOp(NV_OP_FIX, NvMon, NvMonSiz);
    Err = NVOp(NV_OP_SAVE , NvMon, NvMonSiz, NvMonOff);
    if (Err != NVE_NONE) {
        xprintf(NVSupStr1, "storing", NVErrTable[Err]);
        return;
    }
    Err = NVOp(NV_OP_SAVE , &HKFields,   sizeof(NV_HkDefined)  , NvHkOff);
    if (Err != NVE_NONE) {
        xprintf(NVSupStr2, "cannot be initialized");
        return;
    }
}


/*****************************************************************************
 * DOCSEC:      NVSupport  1   Std    Monitor
 *
 * SYNOPSIS:    SetNvDefaults(Groups, NumGroups)
 *              NVGroupPtr Groups;
 *              int NumGroups;
 *
 *              DispGroup(Group, EditFlag)
 *              NVGroupPtr Group;
 *              unsigned long EditFlag;
 *
 *              NVOp(NVOpCmd, Base, Size, Offset)
```



```
 *              unsigned long NVOpCmd, Size, Offset;
 *              unsigned char *Base;
 *
 * DESCRIPTION: The support functions used for displaying, initializing,
 *              and modifying the NV memory data structures can also be
 *              used to manage other data structures which may or may not
 *              be stored in NV memory.
 *
 *              The method used to create a display of a data structure is to
 *              create a second structure that contains a description
 *              of every field of the first structure. This description is
 *              done using the NVGroup structure.
 *              Each entry in the NVGroup structure describes a field
 *              name, pointer to the field, size of the field, indication
 *              of how the field is to be displayed, and the initial value
 *              of the field.
 *
 *              An example data structure is shown below as well as the
 *              NVGroup data structure necessary to describe the
 *              data structure. This example might describe the
 *              coordinates and depth of a window structure.
 *
 *              struct NVExample {
 *              NV_Internal Internal;
 *              unsigned long  XPos, YPos;
 *              unsigned short Mag;
 *              } NVEx;
 *
 *              NVField ExFields[] = {
 *              { "XPos",    (char *) &NVEx.XPos, sizeof(NVEx.XPos),
 *              NV_TYPE_DECIMAL, 0, 100, NULL},
 *              { "YPos",    (char *) &NVEx.YPos, sizeof(NVEx.YPos),
 *              NV_TYPE_DECIMAL, 0, 200, NULL},
 *              { "Depth"    (char *) &NVEx.Mag, sizeof(NVEx.Mag),
 *              NV_TYPE_DECIMAL, 0, 4, NULL}
 *              }
 *
 *              NVGroup ExGroups[] = {
 *              { "Window",  sizeof(ExFields)/sizeof(NVField), ExFields }
 *              };
 *
 *              If passed a pointer to the ExGroups structure, the function
 *              DispGroup generates the display shown below.
 *              The second parameter EditFlag indicates
 *              whether to allow changes to the data structure after it is
 *              displayed (Same as in the NVDisplay command).
 *
 *              Window Display Configuration
 *              XPos            100
 *              YPos            200
 *              Magnitude       4
 *
 *              The SetNvDefaults function, when called with a pointer
 *              to the ExGroup structure, can be used to initialize the data
 *              structure to those values specified in the NVGroup
 *              structure. The second parameter NumGroups indicates
 *              the number of groups to be initialized.
 *
 *              The NVOp function can be used to store and recover
 *              data structures from NV memory. The only requirement of the
 *              data structure to be stored in NV memory is that the first
 *              field of the structure be NVInternal, which is where
 *              all the bookkeeping for the NV memory section is done.
 *              The first parameter NvOpCmd indicates the command
 *              to be performed. A summary of the commands is shown below:
 *
```

```
 *              Command      Value      Description
 *              -----------  -----      ---------------------------
 *              NV_OP_FIX      0        Fix NV section checksum
 *              NV_OP_CLEAR    1        Clear NV section
 *              NV_OP_CK       2        Check if NV section is valid
 *              NV_OP_OPEN     3        Open NV Section
 *              NV_OP_SAVE     4        Save NV Section
 *              NV_OP_CMP      5        Compare NV Section data
 *
 *              The second parameter, Base, indicates the base
 *              address of the data structure to be operated on, and
 *              the Size parameter indicates the size of the
 *              data structure to be operated on. The Offset
 *              parameter indicates the byte offset in the NV memory
 *              device where the data structure is to be stored. An
 *              example of how to initialize, store, and recall the
 *              example data structure is shown below.
 *
 *              NVOp(NV_OP_CLEAR, &NVEx, sizeof(NVEx), 0);
 *              NVOp(NV_OP_SAVE , &NVEx, sizeof(NVEx), 0);
 *              NVOp(NV_OP_OPEN , &NvEx, sizeof(NVEx), 0);
 *
 *              NVOp(NV_OP_FIX,   &NVEx, sizeof(NVEx), 0);
 *              NVOp(NV_OP_SAVE , &NVEx, sizeof(NVEx), 0);
 *
 *              The clear, save, and open operations cause the NV device
 *              to be cleared and filled with the NVEx data structure;
 *              then the data structure is filled from NV memory.
 *              The fix and save operation are used to modify the NV device,
 *              which updates the internal data structures and then writes
 *              them back to the NV memory device.
 *
 *              If errors are encountered during the check, save or
 *              compare operations, an error message is returned from the
 *              function NvOp. The error codes are listed below.
 *
 *              Error number          Description
 *              ---------------     -------------------------------------
 *              NVE_NONE      0     No errors.
 *              NVE_OVERFLOW  1     NV device write count exceeded.
 *              NVE_MAGIC     2     Bad magic number read from NV device.
 *              NVE_CKSUM     3     Bad checksum read from NV device.
 *              NVE_STORE     4     Write to NV device failed.
 *              NVE_CMD       5     Unknown operation requested.
 *              NVE_CMP       6     Data does not compare to NV device.
 *
 *
 *
 * SEE ALSO:    NVFields.h
 ***/


SetNvDefaults(Groups, NumGroups)
NVGroupPtr Groups;
int NumGroups;
{
    unsigned long Value, Temp;
    register int i, j;
    NVFieldPtr Field;

    for (i = 0; i < NumGroups ; i++ ) {
        for (j = 0; j < Groups[i].NumFields ; j++ ) {
            Field = &Groups[i].Fields[j];
            switch (Field->Type) {
            case NV_TYPE_HEX:
                {
```



```
                    if (Field->Aux) {
                        Temp = FieldRead(Field->Address, Field->Size);
                        Temp = Temp & ~Field->Aux;
                        Temp = Temp | Field->InitVal;
                        FieldWrite(Field->Address, Field->Size, Temp);
                    } else {
                        FieldWrite(Field->Address, Field->Size, Field->InitVal);
                    }
                    break;
                }
            case NV_TYPE_DECIMAL:
            case NV_TYPE_VAL_LIST:
                {
                    FieldWrite(Field->Address, Field->Size, Field->InitVal);
                    break;
                }
            case NV_TYPE_STRING:
                {
                    StrCpy(Field->Address, Field->InitVal);
                    break;
                }
            case NV_TYPE_BITFIELD:
                {
                    Temp  = FindBitSet(Field->Aux);
                    Value = FieldRead(Field->Address, Field->Size);
                    Value &= ~Field->Aux;
                    Value |= (Field->InitVal << Temp);
                    FieldWrite(Field->Address, Field->Size, Value);
                    break;
                }
            }
        }
    }
}


DispGroup(Group, EditFlag)
NVGroupPtr Group;
unsigned long EditFlag;
{
    int NumFields = Group->NumFields;
    unsigned long Value, Temp;
    char Buffer[80], RetVal;
    NVFieldPtr Field;
    int j;

    for (j = 0; j < NumFields ; j++ ) {
        Field = &Group->Fields[j];
        xprintf("  %-14s  ", Field->Name);
        Value = FieldRead(Field->Address,Field->Size);
        switch (Field->Type) {
        case NV_TYPE_HEX:
            {
                if (Field->Aux)
                    Value = Value & Field->Aux;
                xprintf(" 0x%x\n",Value);
                break;
            }
        case NV_TYPE_DECIMAL:
            {
                xprintf(" %d\n",Value);
                break;
            }
        case NV_TYPE_STRING:
            {
                xprintf(" %s\n",Field->Address);
```

```
                break;
            }
        case NV_TYPE_BITFIELD:
            {
                Temp = FindBitSet(Field->Aux);
                Value = (Value & Field->Aux) >> Temp;
                Temp = ((Field->Aux >> Temp) + 1);
                DispFieldName(Field->Vals, Temp, Value);
                break;
            }
        case NV_TYPE_VAL_LIST:
            {
                DispFieldName(Field->Vals, Field->Aux, Value);
            }
        }
        if (EditFlag) {
            xprintf(":");
            RetVal = GetString(Buffer);
            xprintf("\r%75s\r", "");
            if (RetVal == CR) {
                if (!SetField(Group, Field->Name, Buffer, FALSE)) {
                    --j;
                }
            }
        }
    }
}


NVOp(NVOpCmd, Base, Size, Offset)
unsigned long NVOpCmd, Size, Offset;
unsigned char *Base;
{
    int ByteNum, DataSize;
    unsigned char *DataSect;
    unsigned char (*NVFunct)(), NVRamAcc(), ModConfAcc();
    unsigned long Operation, Sum;
    NV_Internal *Internals = (NV_Internal *) Base;

    if (NVOpCmd >= 10) {            /* If Op on module configuration space */
        Operation = NVOpCmd - 10;
        NVFunct   = ModConfAcc;
    } else {                        /* Op is on local NV Memory            */
        Operation = NVOpCmd;
        NVFunct   = NVRamAcc;
    }

    DataSect = (unsigned char *) &Internals[1];
    DataSize = Size - sizeof(NV_Internal);
    Sum = CheckSumMem(DataSect, DataSize);

    switch (Operation) {
    case NV_OP_FIX:
        {
            Internals->Magic  = NV_MAGIC;
            Internals->ChkSum = Sum;
            return(NVE_NONE);
        }
    case NV_OP_CLEAR:
        {
            ClearMem(DataSect, DataSize);
            Internals->Magic    = NV_MAGIC;
            Internals->ChkSum   = 0;
            return(NVE_NONE);
        }
    case NV_OP_CK:
```



```
        {
            if (Internals->Magic != NV_MAGIC)
                return(NVE_MAGIC);
            if (Internals->ChkSum != Sum)
                return(NVE_CKSUM);
            if (Internals->WriteCnt > NVRMaxNbrWrites())
                return(NVE_OVERFLOW);
            return(NVE_NONE);
        }
    case NV_OP_OPEN:
        {
            nv_recall();
            for (ByteNum = 0; ByteNum < Size; ByteNum++) {
                Base[ByteNum] = NVFunct(READ, Offset + ByteNum);
            }
            return(NVE_NONE);
        }
    case NV_OP_SAVE:
        {
            if (NVFunct(WRITE_PROBE, Offset, &Base[0]) == NULL) {
                return(NVE_STORE);
            }
            for (ByteNum = 1; ByteNum < Size; ByteNum++) {
                NVFunct(WRITE, Offset + ByteNum, &Base[ByteNum]);
            }
            nv_store();
            nv_recall();
            for (ByteNum = 0; ByteNum < Size; ByteNum++) {
                if (Base[ByteNum] != NVFunct(READ, Offset + ByteNum)) {
                    return(NVE_STORE);
                }
            }
            return(NVE_NONE);
        }
    case NV_OP_CMP:
        {
            Offset += (Size - DataSize);    /* Skip Header */
            for (ByteNum = 0; ByteNum < DataSize; ByteNum++) {
                if (DataSect[ByteNum] != NVFunct(READ, Offset + ByteNum)) {
                    return(NVE_CMP);
                }
            }
            return(NVE_NONE);
        }
    default:
        {
            return(NVE_CMD);
        }
    }
}
```

```
/*****************************************************************************
 *
 * Copyright (c) 1990 Heurikon Corporation
 * All Rights Reserved .
 *
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION.
 *         The copyright notice above does not evidence any
 *         actual or intended publication of such source code.
 *
 * Heurikon hereby grants you permission to copy and modify
 * this software and its documentation.  Heurikon grants
 * this permission provided that the above copyright notice
 * appears in all copies and that both the copyright notice and
 * this permission notice appear in supporting documentation.  In
 * addition, Heurikon grants this permission provided that you
 * prominently mark as not part of the original any modifications
 * made to this software or documentation, and that the name of
 * Heurikon Corporation not be used in advertising or publicity
 * pertaining to distribution of the software or the documentation
 * without specific, written prior permission.
 *
 * Heurikon Corporation does not warrant, guarantee or make any
 * representations regarding the use of, or the results of the use
 * of, the software and documentation in terms of correctness,
 * accuracy, reliability, currentness, or otherwise; and you rely
 * on the software, documentation and results solely at your own
 * risk.
 *
 *
 * MODIFICATIONS:
 *
 *****/

#include "NVAssign.h"       /* Pull in the Internal data definitions       */

#define MPU_68030       3   /* Fixed Hardware devices                      */
#define MMU_68030       3
#define CACHE_NONE      0
#define DMA_NONE        5
#define MEMEXP_NONE     0
#define STREAM_NONE     0

#define ETH_82596CA     1    /* Ethernet may be optional                   */
#define ETH_NONE        0

#define HDISK_NONE      0    /* SCSI may be optional                       */
#define HDISK_WD33C93   3
#define HDISK_WD33C93A  4

#define FPU_NONE        0    /* Floating Point is optional                 */
#define FPU_68881       1
#define FPU_68882       2

/*****************************************************************************
 *  NvMonDefs.h: This header file defines the bit field assignments
 *               for the NVRAM/EEPROM, as they are defined by the board.
 *               It can be used where a program needs to know which bit fields
 *               are assigned to what.
 *               This section describes the board specifics and includes the
 *               Heurikon-specific structures and internal data structures
 *               necessary to maintain NV memory (NVAssign.h).
 *
 * NOTICE:       Because different compilers may generate different spacing
 *               between structures and structure elements based on the
 *               alignment it is important to define structures carefully.
 *               Problems can be avoided by forcing shorts and longs onto
```



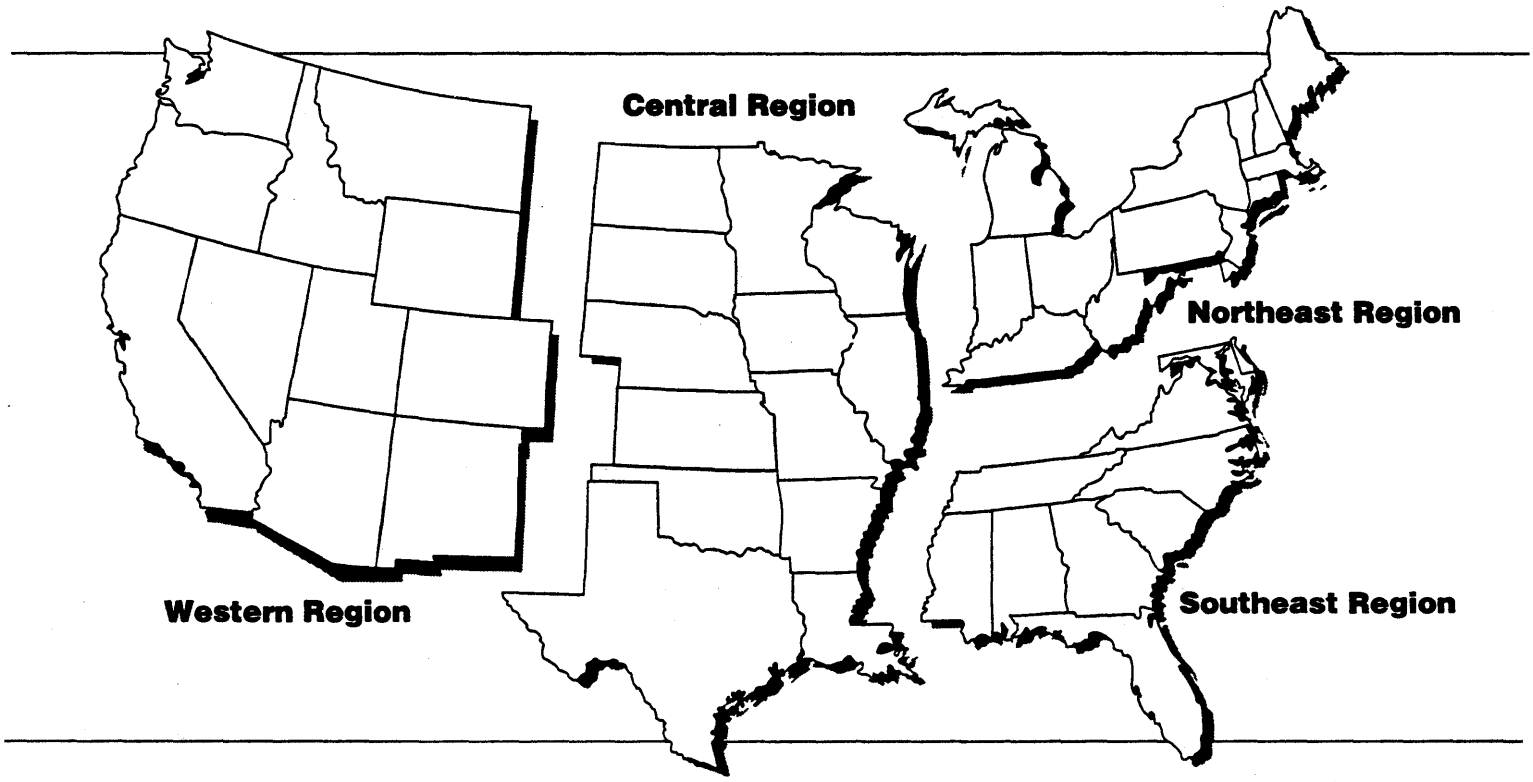
```
 *               long and short boundaries and padding structures to be
 *               a multiple of long words in size.
 *
 *               An early version of the ic960 compiler generated the wrong
 *               structure addresses when the structures were organized as
 *               (long, short, byte) quantities in that order. If the smaller
 *               fields are first in the structure it works much better, so
 *               be careful !!!!.
 ***/

/******* SERIAL DEFINITIONS ***************************************************
 * This structure provides the definitions for a serial port. This
 * includes the port number, baud rate and configuration.
 * This structure should be loaded in the user-configurable portion of
 * the nonvolatile memory array.
 ***/

typedef struct NVU_Port {            /* Port struct = 8/4 bytes          */
    unsigned char  Reserved;
    unsigned char  PortNum;          /* Port number (A,B,C or D)         */
    unsigned short PortFlags;        /* Flags for port                   */
    unsigned long  Baud;             /* Port baud rate                   */
} NVU_Port;

/* Warning: These macros only work with pointers                          */

#define Parity(x)          (x->PortFlags & 0x0003)
#define DataBits(x)       ((x->PortFlags & 0x000C) >> 2)
#define XOnXOff(x)         (x->PortFlags & 0x0010)
#define ChBaudOnBreak(x)   (x->PortFlags & 0x0040)
#define ResetOnBreak(x)    (x->PortFlags & 0x0080)
#define StopBits(x)        (x->PortFlags & 0x0100)

#define SP_APORT           0      /* Serial Port Assignments             */
#define SP_BPORT           1
#define SP_CPORT           2
#define SP_DPORT           3

#define SP_PARITY_EVEN     0      /* Parity Type Assignments             */
#define SP_PARITY_ODD      1
#define SP_PARITY_NONE     2
#define SP_PARITY_FORCE    3

#define SP_DATA_5BITS      0      /* Data Bits Assignments               */
#define SP_DATA_6BITS      1
#define SP_DATA_7BITS      2
#define SP_DATA_8BITS      3

#define SP_STOP_1BITS      0
#define SP_STOP_2BITS      1

/******* BOOT DEFINITIONS *****************************************************
 * This sections defines the boot parameters for loading an application
 * from a device and executing the application. This section should be
 * located in the user section of the nonvolatile memory device.
 ***/

typedef struct NVU_Boot {            /* Boot struct = 32/20 bytes        */
    unsigned char AutoBootDev;       /* Auto Boot Device                 */
    unsigned char Device;            /* Boot Device                      */
    unsigned char Number;            /* Boot Device Number               */
    unsigned char BootFlags;         /* Boot Flags                       */
    unsigned long LoadAddress;       /* Load Address                     */
    unsigned long RomSize;           /* Boot ROM Size                    */
    unsigned long RomBase;           /* Boot ROM Base address            */
#ifndef NV_SMALL
```

```
    char Reserved[16];
#endif
} NVU_Boot;

#define  ClrMemOnBoot(x)     (x->BootFlags & 0x01)     /* Clear on boot    */

#define AB_DONT             0          /* Auto Boot Definitions           */
#define AB_WINCH            1
#define AB_FLOPPY           2
#define AB_TAPE             3
#define AB_SERIAL           4
#define AB_ROM              6
#define AB_ETHERNET         7
#define AB_BUS              8


/******* VME BUS DEFINITIONS ***********************************************
 * This structure defines the VMEbus configuration of the slave interface
 * and Vic configuration registers. This structure should be loaded in
 * the user-defined section of the NV memory.
 ***/

typedef struct NVU_BusConfig {          /* BusConfig struct = 16/4 bytes  */
    unsigned char   Padding[3];         /* Reserved                       */
    unsigned char   AddrModSel;         /* Address Modifier select        */
    unsigned long   MiscBusFlags;       /* Misc bus configuration bits    */
    unsigned long   SlaveBusMap;        /* Slave bus map configuration    */
#ifndef NV_SMALL
    unsigned char   Reserved[4];        /* Reserved                       */
#endif
} NVU_BusConfig;

#define ExtSlaveMap(x)     (x->SlaveBusMap & 0xFFF00000)
#define StdSlaveMap(x)     (x->SlaveBusMap & 0x00F00000)
#define ShtSlaveMap(x)     (x->SlaveBusMap & 0x0000FFF8)
#define EnblSlave(x)       (x->SlaveBusMap & 0x00000004)

#define ReplaceAddr(x)     (x->MiscBusFlags & 0x00F00000)
#define MastRelMode(x)     (x->MiscBusFlags & 0x00000003)
#define SlaveRelMode(x)    (x->MiscBusFlags & 0x00000004)
#define LocBusTimer(x)     (x->MiscBusFlags & 0x00000008)
#define VmeBusTimer(x)     (x->MiscBusFlags & 0x00000010)
#define Sysfail(x)         (x->MiscBusFlags & 0x00000020)
#define IndivRMC(x)        (x->MiscBusFlags & 0x00000040)
#define EnblSht(x)         (x->MiscBusFlags & 0x00000080)


/******* MONITOR DEFINED DEFINITIONS ***************************************
 * This section binds the Monitor-defined data structures into one
 * common structure, which should be loaded into NV memory in the user
 * read/write section.
 ***/

typedef struct NV_MonDefs {             /* Mon Defs struct = 76/48        */
    NV_Internal     Internal;           /* Internal definitions           */
    unsigned long   MiscFlags;          /* Misc monitor flags             */
    NVU_Port        Console;            /* Console Port Configuration     */
    NVU_Port        DownLoad;           /* Download Port Configuration    */
    NVU_Boot        Boot;               /* Boot Definitions               */
    NVU_BusConfig   VmeBus;             /* Bus Configuration Definitions  */
} NV_MonDefs, *NV_MonDefPtr;

#define  ClrMemOnPowerUp(x) (x->MiscFlags & 0x01)  /* Clear on powerup       */
#define  ClrMemOnReset(x)   (x->MiscFlags & 0x02)  /* Clear on reset         */
#define  DoPowerDiag(x)     (x->MiscFlags & 0x04)  /* Do powerup diagnostics */
#define  VsbMasterEnbl(x)   (x->MiscFlags & 0x08)  /* VSB Master enable      */
```



```
#define  VsbReleaseMode(x)  (x->MiscFlags & 0x10)  /* VSB Release modes      */
#define  ScsiResetEnbl(x)   (x->MiscFlags & 0x20)  /* Scsi reset enable      */
#define  ScsiIntMask(x)     (x->MiscFlags & 0x40)  /* Scsi interrupt mask    */

#define  DataCacheEnble(x)  (x->MiscFlags & 0x80)  /* 030 data cache         */
#define  InstCacheEnble(x)  (x->MiscFlags & 0x100) /* Instruction cache Enbl */
#define  CountDownVal(x)    ((x->MiscFlags & 0x00000E00) >> 9)
```

Sales and Customer Service Offices



Heurikon Corporate Office
8000 Excelsior Drive
Madison, WI 53717
Watts: 800-356-9602
Phone: 608-831-0900
Fax: 608-831-4249

Heurikon Customer Support and Factory Service Office
8310 Excelsior Drive
Madison, WI 53717
Watts: 800-327-1251
Phone: 608-831-5500

Heurikon Northeast Regional Office
67 South Bedford, Suite 400 W.
Burlington, MA 01803
Phone: 617-229-5831
Fax: 617-272-9115

Heurikon Southeast Regional Office
2010 Corporate Ridge, Suite 700
McLean, VA 22102
Phone: 703-749-1474
Fax: 703-556-0955

Heurikon Central Regional Office
13100 West 95th Street, Level 4D
Lenexa, KS 66215
Phone: 913-599-1860
Fax: 913-599-1918

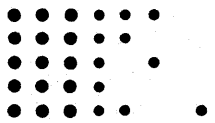
Heurikon Western Regional Office
23121 Plaza Pointe Dr., Suite 109
Laguna Hills, CA 92653
Phone: 714-581-6400
Fax: 714-581-8509

HEURIKON CORP.

OPEN SYSTEMS :: OPEN TOOLS

revised 1.91

Regional Sales Representatives



• NORTHEAST REGION

CT, DE, ME, MA, NH, NJ, NY, Eastern PA, RI and VT

Daner-Hayes, Inc.
62 West Plain Street
Wayland, MA 01778
Tel: (508) 655-0888
Fax: (508) 655-0939

IN, KY, MI, OH, Western PA and WV

Systems Components, Inc.
1327 Jones, Suite 104
Ann Arbor, MI 48105
Tel: (313) 930-1800
Fax: (313) 930-1803

• CENTRAL REGION

AR, LA, OK and TX

Acudata, Inc.
720 Avenue F, Suite 104
Plano, TX 75074
Tel: (214) 424-3567
Fax: (214) 422-7342

MN, ND, SD and Northwest WI

Micro Resources Corp.
4640 W. 77th Street,
Suite 109
Edina, MN 55435
Tel: (612) 830-1454
Fax: (612) 830-1380

IL, IA, KS, MO, NE and Southeast WI

Panatek
2500 West Higgins Road,
Suite 305
Hoffman Estates, IL 60195
Tel: (708) 519-0867
Fax: (708) 519-0897

• SOUTHEAST REGION

MD, VA and Washington, DC

Spectro Associates
1107 Nelson Street, #203
Rockville, MD 20850
Tel: (301) 294-9770
Fax: (301) 294-9772

• WESTERN REGION

AZ, CO, NV, NM and UT

Compware Marketing
100 Arapahoe Ave.
Suite 7
Boulder, CO 80302
Tel: (303) 786-7045
Fax: (303) 786-7047

ID, MT, OR, WA, WY and Canada (Alberta and British Columbia)

Electronic Component Sales
9311 S. E. 36th Street
Mercer Island, WA
98040-3795
Tel: (206) 232-9301
Fax: (206) 232-1095

CA

Qualtech
333 West Maude Avenue,
Suite 108
Sunnyvale, CA 94086
Tel: (408) 732-4800
Fax: (408) 733-7084

HEURIKON CORP.

OPEN SYSTEMS :: OPEN TOOLS

revised 1.91

Index

R

S

T

U

V

W

Z

Heurikon Corporation
8310 Excelsior Drive
Madison, WI 53717

Customer Support: 1•800•327•1251

Sales: 1•800•356•9602