HK68/V3D

68030-based VMEbus Single-board Computer
HK68/V3D
VMEbus 68030-based Single Board Computer

USER'S MANUAL
Revision A (Preliminary)
June 1991
The information in this manual has been checked and is believed to be accurate and reliable. HOWEVER, NO RESPONSIBILITY IS ASSUMED BY HEURIKON FOR ITS USE OR FOR ANY INACCURACIES. Specifications are subject to change without notice. HEURIKON DOES NOT ASSUME ANY LIABILITY ARISING OUT OF USE OR OTHER APPLICATION OF ANY PRODUCT, CIRCUIT OR PROGRAM DESCRIBED HEREIN. This document does not convey any license under Heurikon's patents or the rights of others.

Heurikon and HK68/V are trademarks of Heurikon Corporation.

Intel is a trademark of Intel Corporation. Ethernet is a trademark of Xerox Corporation. UNIX is a registered trademark of AT&T. VxWorks is a trademark of Wind River Systems, Inc.

REVISION HISTORY

<table>
<thead>
<tr>
<th>Revision Level</th>
<th>Principal Changes</th>
<th>Date of Publication</th>
<th>Board Revision Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (Preliminary)</td>
<td>First publication</td>
<td>June 1991</td>
<td>EP0</td>
</tr>
</tbody>
</table>

Copyright 1991 Heurikon Corporation. All rights reserved.
Contents

1 — Overview
1.1 Introduction ....................................................................................... 1-1
1.2 Components and Features ................................................................. 1-1
1.3 Functional Description ..................................................................... 1-4
1.4 Jumpers, Connectors, and Switches ................................................. 1-6
1.4.1 Jumpers ...................................................................................... 1-6
1.4.2 Connectors .................................................................................... 1-6
1.4.3 Interrupt or Reset Switch ................................................................ 1-7
1.5 Overview of the Manual .................................................................... 1-8
1.5.1 Terminology and Notation .......................................................... 1-8
1.5.2 Additional Technical Information ................................................. 1-8

2 — Setup and Installation
2.1 Introduction ....................................................................................... 2-1
2.2 Unpacking .......................................................................................... 2-1
2.3 Recording Serial Numbers .................................................................. 2-1
2.4 Providing Power .................................................................................. 2-2
2.5 Reserving Space .................................................................................. 2-2
2.6 Providing Air Flow ............................................................................... 2-2
2.7 Checking Operation ............................................................................. 2-3
2.8 Troubleshooting and Service Information ........................................ 2-10
2.9 Monitor Summary ............................................................................... 2-12

3 — Microprocessor Unit
3.1 Introduction ....................................................................................... 3-1
3.2 MPU Interrupts ................................................................................... 3-1
3.3 MPU Exception Vectors ..................................................................... 3-2
3.4 Status LEDs ......................................................................................... 3-6
3.5 Monitoring MPU Status from the Front Panel Interface .................. 3-6
3.6 MPU Cache Control .......................................................................... 3-7
3.7 Coprocessors ...................................................................................... 3-7

4 — Optional Floating Point Coprocessor
4.1 Feature Summary ............................................................................... 4-1
4.2 Bypassing the Floating Point Coprocessor ......................................... 4-2
5 — System Error Handling

6 — On-card Memory Configuration
6.1 Memory Configuration ................................................................. 6-1
6.2 ROM ...................................................................................... 6-1
6.3 On-card RAM ............................................................................. 6-4
6.4 On-card Memory Sizing .............................................................. 6-4
6.5 Bus Memory .............................................................................. 6-4
6.6 Physical Memory Map .............................................................. 6-4
6.7 Memory Timing ........................................................................ 6-6
6.8 Nonvolatile RAM ...................................................................... 6-7

7 — VMEbus Control
7.1 Introduction .............................................................................. 7-1
7.2 Bus Control Signals ..................................................................... 7-1
7.3 Bus Arbitration and Release ...................................................... 7-4
7.4 Accesses from the VMEbus (Slave Mode) .................................. 7-6
7.5 VMEbus Interrupts ...................................................................... 7-12
  7.5.1 Interrupter Module Operation ........................................... 7-12
  7.5.2 Interrupt Handler Operation ............................................. 7-13
7.6 SYSFAIL Control ...................................................................... 7-14
7.7 Bus Addressing (Master Mode) .................................................. 7-14
7.8 Mailbox Interface ...................................................................... 7-15
7.9 Watchdog and Bus Timer .......................................................... 7-16
  7.9.1 On-card Watchdog Timer .................................................. 7-16
  7.9.2 VMEbus Timer .................................................................... 7-16
7.10 Bus Control Jumpers ............................................................... 7-17
7.11 VMEbus Interface ..................................................................... 7-17
7.12 VMEbus Pin Assignments, P1 .................................................. 7-18
7.13 VMEbus Pin Assignments, P2 .................................................. 7-19

8 — 7-segment Display

9 — CIO Implementation
9.1 Introduction .............................................................................. 9-1
9.2 Port A Bit Definition ................................................................. 9-1
9.3 Port B Bit Definition ................................................................. 9-2
9.4 Port C Bit Definition ................................................................. 9-2
9.5 Counter/Timers ......................................................................... 9-3
9.6 Register Address Summary (CIO) .......................................... 9-3
9.7 CIO Initialization ...................................................................... 9-3
9.8 CIO Programming Hints .......................................................... 9-5
10 — Serial I/O
10.1 Introduction ..................................................................................... 10-1
10.2 RS-232 Pin Assignments ................................................................. 10-1
10.3 Signal Naming Conventions (RS-232) .............................................. 10-3
10.4 Connector Conventions..................................................................... 10-4
10.5 SCC Initialization Sequence .............................................................. 10-6
10.6 Port Address Summary .................................................................... 10-6
10.7 Baud Rate Constants ...................................................................... 10-6
10.8 RS-422 Operation .......................................................................... 10-7
10.9 Relevant Jumpers (Serial I/O) ............................................................ 10-7
10.10 Serial I/O Cable ........................................................................... 10-8

11 Optional SCSI Port
11.1 Introduction ..................................................................................... 11-1
11.2 SCSI Implementation Notes .............................................................. 11-1
11.3 Register Address Summary (SCSI) .................................................. 11-2
11.4 SCSI Port Pinouts ........................................................................... 11-2
11.5 SCSI Bus Termination ...................................................................... 11-4

12 — Optional Ethernet Interface
12.1 Introduction ..................................................................................... 12-1
12.1.1 Network Interface Controller (82596CA) .................................... 12-1
12.1.2 Serial Network Interface (82C501AD) ........................................ 12-1
12.2 Ethernet Address ............................................................................ 12-2
12.2.1 Verifying the Ethernet Address ................................................... 12-2
12.2.2 Ethernet Address on the HK68/V3D .......................................... 12-3
12.3 82596CA Implementation on the HK68/V3D ................................... 12-3
12.3.1 82596CA Configuration on the HK68/V3D ............................... 12-3
12.3.2 82596CA Parameter Selections .................................................. 12-3
12.4 Byte Ordering .................................................................................. 12-5
12.5 Ethernet Access .............................................................................. 12-5
12.5.1 Port Access ............................................................................... 12-6
12.5.2 Channel Attention ...................................................................... 12-7
12.6 SYSBUS Byte of the System Configuration Pointer ....................... 12-8
12.7 Recommended Initialization ............................................................. 12-9
12.8 Addresses of Ethernet Functions .................................................... 12-9
12.8.1 Interrupts ................................................................................. 12-11
12.9 Exception Conditions ..................................................................... 12-13
12.10 Ethernet Jumper .......................................................................... 12-14
12.11 Ethernet Port Pin Assignments, P4 ................................................. 12-14
Figures

Figure 1-1 Component Map ................................................................. 1-3
Figure 1-2 HK68/V3D Block Diagram ............................................. 1-5
Figure 1-3 Front Panel .................................................................. 1-7
Figure 2-1 Location of Serial Numbers ........................................... 2-1
Figure 2-2 Guide to Jumper Locations ............................................ 2-4
Figure 2-3 The HK68/V3D Configured as
VMEbus System Controller via Jumpers J14-J18 ..................... 2-9
Figure 2-4 The HK68/V3D Not Configured as
VMEbus System Controller Via Jumpers J14-J18 ................. 2-9
Figure 6-1 ROM Jumpers ............................................................... 6-2
Figure 6-2 ROM position for 28-pin ROMs .................................... 6-3
Figure 6-3 Physical Memory Map .................................................. 6-5
Figure 7-1 Bus Request Jumper Settings, J16 ............................ 7-5
Figure 7-2 Bus Grant Level Jumper Settings, J14, J15, J17, J18 ....... 7-5
Figure 7-3 Slave Window Size Jumper Settings, J21-24 .............. 7-7
Figure 7-4 Bus Control Latch ....................................................... 7-10
Figure 7-5 Memory Accesses from the VMEbus ......................... 7-11
Figure 7-6 Interrupt Signal Routing ............................................. 7-14
Figure 7-7 VMEbus Connectors, P1 and P2 ................................. 7-17
Figure 8-1 7-segment Display ..................................................... 8-1
Figure 10-1 Serial Connector, P3 .................................................. 10-1
Figure 10-2 Serial Cable ............................................................. 10-8
Figure 11-1 SCSI Connector, P2 .................................................. 11-2
Figure 11-2 Location of SCSI Terminating Resistor Networks
and Fuse F6 ........................................................................... 11-4
Figure 12-1 Ethernet Address Format ........................................... 12-2
Figure 12-2 Required Settings of the System Configuration
Pointer SYSBUS Byte .............................................................. 12-8
Figure 12-3 Ethernet Connector, P4 ............................................... 12-14
Figure 13-1 Real-time Clock Socket ............................................ 13-1
Figure 13-2 RTC Comparison Register Definition..................... 13-5
Figure 13-3 RTC Register Definition .......................................... 13-7
Figure 14-1 Jumper Locations ..................................................... 14-5

Tables

Table 1-1 Technical References ..................................................... 1-8
Table 2-1 Power Requirements ..................................................... 2-2
Table 2-2 Standard Jumper Settings ............................................. 2-4
Table 2-3 ROM Size Options ....................................................... 2-6
Table 2-4 Monitor Command Summary ....................................... 2-13
Table 3-1  MPU Interrupt Levels..................................................... 3-1
Table 3-2  MPU Exception Vectors............................................. 3-3
Table 3-3  Suggested Interrupt Vectors...................................... 3-4
Table 3-4  Device Interrupt Vector Values (Suggested).............. 3-5
Table 3-5  Status LEDs................................................................. 3-6
Table 3-6a Front Panel Interface, P5, Output Signals ............. 3-6
Table 3-6b Front Panel Interface, P5, Input Signals............... 3-7
Table 3-7  MPU Cache Control................................................... 3-7

Table 6-1  ROM Address Summary........................................... 6-1
Table 6-2  ROM Capacity and Jumper Positions ..................... 6-2
Table 6-3  Access Time Required for No Wait States .............. 6-6
Table 6-4  Inserting Wait States into RAM Cycles ................. 6-6
Table 6-5  Nonvolatile RAM Addresses................................. 6-7
Table 6-6  NV-RAM Contents (partial)................................. 6-8

Table 7-1  System Controller Functions ................................... 7-4
Table 7-2  Bus Control Bits ....................................................... 7-6
Table 7-3  Slave Mode Control................................................ 7-6
Table 7-4  Slave Window Size Jumpers .................................... 7-7
Table 7-5  Bus Control Latch (VMEbus Slave Logic) ............. 7-9
Table 7-6  Slave Address Modifiers ........................................... 7-10
Table 7-7  VMEbus Interrupter Addresses ............................. 7-12
Table 7-8  Interrupt Acknowledge Port Summary ..................... 7-13
Table 7-9  VMEbus Regions......................................................... 7-15
Table 7-10 Mailbox Control ..................................................... 7-15
Table 7-11 Mailbox Functions ................................................ 7-15
Table 7-12 Bus Control Jumpers ............................................. 7-17
Table 7-13 VMEbus Pin Assignments, P1 .................... 7-18
Table 7-14 VMEbus Pin Assignments, P2 .................... 7-20

Table 8-1  Addresses for the 7-segment Display ..................... 8-1
Table 9-1  CIO Port A Bit Definitions ...................................... 9-1
Table 9-2  CIO Port B Bit Definitions ...................................... 9-2
Table 9-3  CIO Port C Bit Definitions ...................................... 9-2
Table 9-4  CIO Register Addresses .......................................... 9-3

Table 10-1a Port A Serial Port Pin Assignments, P3 .......... 10-2
Table 10-1b Port B Serial Port Pin Assignments, P3 .......... 10-2
Table 10-2  Signal Naming Conventions .............................. 10-3
Table 10-3  RS-232 Cable Reversal ........................................... 10-4
Table 10-4  SCC Initialization Sequence .............................. 10-5
Table 10-5  SCC Register Addresses ...................................... 10-6
Table 10-6  Baud Rate Constants ............................................. 10-7
Table 10-7  Serial I/O Jumpers ............................................... 10-7
Table 11-1  SCSI Register Address Summary ............................................ 11-2
Table 11-2  SCSI Pin Assignments, P2 .................................................. 11-3
Table 12-1  Ethernet Accesses .......................................................... 12-5
Table 12-2  Port Access Definition ..................................................... 12-6
Table 12-3  Port Accesses ................................................................. 12-7
Table 12-4  SYSBUS Byte Selections .................................................. 12-8
Table 12-5  Ethernet Peripheral Addresses ....................................... 12-10
Table 12-6  Transmit Differential Line Configuration (J1) .................... 12-14
Table 12-7  Ethernet Connector Pin Assignments, P4 ...................... 12-15
Table 13-1  Effect of RTC Installation on Board Height ...................... 13-2
Table 13-2  Pin Assignments, Real-Time Clock .................................. 13-4
Table 14-1  Address Summary ........................................................... 14-2
Table 14-2  Standard Jumper Settings ............................................... 14-3
Table 14-3  ROM Size Options .......................................................... 14-4
Table 14-4  Power Requirements ....................................................... 14-6
Table 14-5  Mechanical Specifications ............................................... 14-6
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

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPU</td>
<td>Motorola 68030 or 68EC030 microprocessor chip, running at 32 MHz</td>
</tr>
<tr>
<td></td>
<td>32-bit internal architecture</td>
</tr>
<tr>
<td></td>
<td>32-bit address and data paths</td>
</tr>
<tr>
<td></td>
<td>32 address lines</td>
</tr>
<tr>
<td></td>
<td>4-gigabyte addressing range</td>
</tr>
<tr>
<td></td>
<td>256-byte data cache</td>
</tr>
<tr>
<td></td>
<td>256-byte instruction cache</td>
</tr>
<tr>
<td></td>
<td>MMU standard (option for 68EC030 MPU without MMU)</td>
</tr>
<tr>
<td>FPU option</td>
<td>Optional 68882 floating point coprocessor</td>
</tr>
<tr>
<td></td>
<td>Uses the IEEE-P754 Binary Floating Point Standard</td>
</tr>
<tr>
<td>RAM</td>
<td>2-, 4-, or 16-megabyte capacity</td>
</tr>
<tr>
<td></td>
<td>One parity bit per byte</td>
</tr>
<tr>
<td></td>
<td>Uses 256K × 4 or 1024K × 4 DRAMs.</td>
</tr>
<tr>
<td></td>
<td>Hardware refresh</td>
</tr>
<tr>
<td>EPROM</td>
<td>Two ROM sockets</td>
</tr>
<tr>
<td></td>
<td>2-megabyte total capacity</td>
</tr>
<tr>
<td></td>
<td>Page-addressable ROM and EEPROM capability</td>
</tr>
</tbody>
</table>
NV-RAM  Nonvolatile static RAM for programmable functions
        256 x 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
- 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 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.

![Diagram of front panel interface](image-url)
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

<table>
<thead>
<tr>
<th>Device</th>
<th>Number</th>
<th>Document</th>
<th>HK68/V3D User's Manual Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI</td>
<td>WD33C93</td>
<td><em>WD33C93 Technical Specification.</em> This document is also available from Heurikon (part number 001M209).</td>
<td>11</td>
</tr>
<tr>
<td>Ethernet Interface</td>
<td>82596CA</td>
<td><em>Intel 82596CA User's Manual</em> (Intel publication number 296443-001) and <em>Intel 82C501AD Data Sheet.</em> (Not currently available from Heurikon.)</td>
<td>12</td>
</tr>
<tr>
<td>Real-Time Clock</td>
<td>DS1216F</td>
<td><em>Dallas Semiconductor 1990-91 Product Data Book</em> (Dallas, TX: Dallas Semiconductor). (Not currently available from Heurikon.)</td>
<td>13</td>
</tr>
</tbody>
</table>

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.
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.

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**

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Current</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5</td>
<td>7.0 A, max</td>
<td>All logic</td>
</tr>
<tr>
<td>+12</td>
<td>20 mA, max</td>
<td>Reset timing, RS-232 interface</td>
</tr>
<tr>
<td>-12</td>
<td>20 mA, max</td>
<td>RS-232 interface</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Standard Configuration</th>
<th>Options</th>
<th>Function</th>
<th>HK68/V3D Manual Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Installed</td>
<td>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).</td>
<td>Selects Ethernet differential voltage</td>
<td>12</td>
</tr>
<tr>
<td>J3</td>
<td>J3:1-2 Ring Indicator</td>
<td>J3:1-2 Ring Indicator</td>
<td>Selects Ring Indicator or Data Carrier Detect for SCC Port A.</td>
<td>10</td>
</tr>
<tr>
<td>J5-J8</td>
<td>Matches ROM0 size. See Table 2-3.</td>
<td>2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27513 paged, 2864 R/W EEPROM, 2817 R/W EEPROM</td>
<td>Selects ROM 0 size (default is 2764)</td>
<td>5</td>
</tr>
<tr>
<td>J9-J12</td>
<td>Matches ROM1 size. See Table 2-3.</td>
<td>2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27513 paged, 2864 R/W EEPROM, 2817 R/W EEPROM</td>
<td>Selects ROM 1 size (default is 2764)</td>
<td>5</td>
</tr>
<tr>
<td>J14, J15, J17, J18</td>
<td>Bus Grant Level 3</td>
<td>Bus Grant Level 3</td>
<td>Selects VMEbus Bus Grant level</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus Grant Level 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus Grant Level 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus Grant Level 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J16</td>
<td>Bus Request Level 3)</td>
<td>Bus Request Level 3</td>
<td>VMEbus arbitration (bus request level 3, not system controller)</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus Request Level 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus Request Level 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus Request Level 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>J19:2-3 output to VMEbus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J20</td>
<td>Removed</td>
<td>J20 installed: Allows HK68/V3D to respond to ACFAIL* interrupt. J21 removed: HK68/V3D does not respond to ACFAIL* interrupt.</td>
<td>ACFAIL* connects to VMEbus</td>
<td>7</td>
</tr>
<tr>
<td>J21-J24</td>
<td>Matches memory size.</td>
<td>1, 2, 4, 8, or 16 megabytes</td>
<td>VMEbus slave window size</td>
<td>7</td>
</tr>
<tr>
<td>J25</td>
<td>Removed</td>
<td>J25 installed: drives SYSCLK</td>
<td>Disables SYSCLK</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J25 removed: does not drive SYSCLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J26</td>
<td>Removed</td>
<td>J26 installed: HK68/V3D can drive BCLR*. J26 removed: HK68/V3D cannot drive BCLR*.</td>
<td>Disables BCLR*</td>
<td>7</td>
</tr>
<tr>
<td>J91</td>
<td>Factory set for memory configuration. Do not alter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J92</td>
<td>Factory set for memory configuration. Do not alter.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 2-3
**ROM size options**

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

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.

Turn the system on.

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.
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 the 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 ACFFAIL.

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 controlled 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.
In this example, J21-J24 are configured for 16-megabyte slave window size.

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

In this example, J14-J18 are configured for VMEbus request level 3 and the HK68/V3D not the system controller.

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

In this example, J21-J24 are configured for 16-megabyte slave window size.
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 HK68N3D is available as an option. The monitor allows you to access almost all of the hardware registers on the HK68N3D. 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 and 0001,0000 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 online 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 HK68N3D.

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

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

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>
<thead>
<tr>
<th>Vector</th>
<th>Address Offset</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>Reset: Initial SSP (Supervisor Stack Pointer)</td>
</tr>
<tr>
<td>1</td>
<td>004</td>
<td>Reset: Initial PC (Supervisor Program Counter)</td>
</tr>
<tr>
<td>2</td>
<td>008</td>
<td>Bus Error (Watchdog Timer, MMU Fault)</td>
</tr>
<tr>
<td>3</td>
<td>00C</td>
<td>Address Error</td>
</tr>
<tr>
<td>4</td>
<td>010</td>
<td>Illegal Instruction</td>
</tr>
<tr>
<td>5</td>
<td>014</td>
<td>Divide by Zero</td>
</tr>
<tr>
<td>6</td>
<td>018</td>
<td>CHK Instruction (register bounds)</td>
</tr>
<tr>
<td>7</td>
<td>01C</td>
<td>TRAPV Instruction (overflow)</td>
</tr>
<tr>
<td>8</td>
<td>020</td>
<td>Privilege Violation (STOP, RESET, RTE, etc)</td>
</tr>
<tr>
<td>9</td>
<td>024</td>
<td>Trace (Program development tool)</td>
</tr>
<tr>
<td>10</td>
<td>028</td>
<td>Instruction Group 1010 Emulator</td>
</tr>
<tr>
<td>11</td>
<td>02C</td>
<td>FPP Coprocessor not present</td>
</tr>
<tr>
<td>12</td>
<td>030</td>
<td>(reserved)</td>
</tr>
<tr>
<td>13</td>
<td>034</td>
<td>FPP Coprocessor Protocol Violation</td>
</tr>
<tr>
<td>14</td>
<td>038</td>
<td>Format Error</td>
</tr>
<tr>
<td>15</td>
<td>03C</td>
<td>Uninitialized Interrupt</td>
</tr>
<tr>
<td>16-23</td>
<td>040-05F</td>
<td>(reserved-8)</td>
</tr>
<tr>
<td>24</td>
<td>060</td>
<td>Spurious Interrupt, not used</td>
</tr>
<tr>
<td>25</td>
<td>064</td>
<td>Level 1 autovector, VSB</td>
</tr>
<tr>
<td>26</td>
<td>068</td>
<td>Level 2 autovector, not used</td>
</tr>
<tr>
<td>27</td>
<td>06C</td>
<td>Level 3 autovector, not used</td>
</tr>
<tr>
<td>28</td>
<td>070</td>
<td>Level 4 autovector, SCSI Interrupt</td>
</tr>
<tr>
<td>29</td>
<td>074</td>
<td>Level 5 autovector, not used</td>
</tr>
<tr>
<td>30</td>
<td>078</td>
<td>Level 6 autovector, not used</td>
</tr>
<tr>
<td>31</td>
<td>07C</td>
<td>Level 7 autovector, parity error, ACFAIL</td>
</tr>
<tr>
<td>32-47</td>
<td>080-0BF</td>
<td>TRAP Instruction Vectors (16)</td>
</tr>
<tr>
<td>48-54</td>
<td>0C0-0DB</td>
<td>FPP Exceptions (8)</td>
</tr>
<tr>
<td>55-63</td>
<td>0DC-0FF</td>
<td>(reserved-8)</td>
</tr>
<tr>
<td>64-255</td>
<td>100-3FF</td>
<td>User Interrupt Vectors (192)</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Level</th>
<th>Vector</th>
<th>Device</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>31</td>
<td>CIO</td>
<td>Parity err./ACFAIL autovectored interrupt</td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>CIO</td>
<td>Timer 3</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>CIO</td>
<td>External Interrupt (P6-11)</td>
</tr>
<tr>
<td></td>
<td>77</td>
<td>CIO</td>
<td>EEPROM 1 Ready</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>CIO</td>
<td>EEPROM 0 Ready</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td></td>
<td>Mailbox Interrupt</td>
</tr>
<tr>
<td>69</td>
<td></td>
<td></td>
<td>VME Interrupt in Progress</td>
</tr>
<tr>
<td>67, 65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
<td>CIO</td>
<td>Timer 2</td>
</tr>
<tr>
<td>76, 74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72, 70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68, 66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>CIO</td>
<td>Timer 1</td>
</tr>
<tr>
<td>102</td>
<td></td>
<td>CIO</td>
<td>Timer, error</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>SCSI</td>
<td>SCSI Interface (autovectored)</td>
</tr>
<tr>
<td>2</td>
<td>92</td>
<td>SCC</td>
<td>Port A, Receive character available</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td></td>
<td>Port A, Special receive condition</td>
</tr>
<tr>
<td>88</td>
<td></td>
<td></td>
<td>Port A, Transmit buffer empty</td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td>Port A, External/Status change</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Device</th>
<th>Hexadecimal Value</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCC 1 (Ports A &amp; B):</td>
<td>50&lt;sub&gt;16&lt;/sub&gt;</td>
<td>80</td>
</tr>
<tr>
<td>CIO, Port A:</td>
<td>41&lt;sub&gt;16&lt;/sub&gt;</td>
<td>65</td>
</tr>
<tr>
<td>CIO, Port B:</td>
<td>40&lt;sub&gt;16&lt;/sub&gt;</td>
<td>64</td>
</tr>
<tr>
<td>CIO, C/T vector:</td>
<td>60&lt;sub&gt;16&lt;/sub&gt;</td>
<td>96</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>LED</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Fail</td>
<td>The SYSFAIL line is being driven active by this board.</td>
</tr>
<tr>
<td>M</td>
<td>Master</td>
<td>The HK68N3D is the master on the VMEbus. It owns the VMEbus.</td>
</tr>
<tr>
<td>B</td>
<td>Slave (bus grant acknowledge, BGACK*)</td>
<td>The HK68N3D is not the VMEbus master. It has given up the local bus, which might be either VMEbus or Ethernet.</td>
</tr>
</tbody>
</table>

3.5 MONITORING MPU STATUS FROM THE FRONT PANEL INTERFACE

Four status outputs allow remote monitoring of the HK68N3D processor. Connections are made through a 14-pin connector, P5.

<table>
<thead>
<tr>
<th>P5 pin</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Supr</td>
<td>The MPU is in the supervisor state.</td>
</tr>
<tr>
<td>4</td>
<td>User</td>
<td>The MPU is in the user state.</td>
</tr>
<tr>
<td>6</td>
<td>not connected</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Halt</td>
<td>The MPU has halted. (Double bus fault, odd stack address or the system reset line is active.)</td>
</tr>
<tr>
<td>10</td>
<td>Bus grant acknowledge (BGACK*)</td>
<td>The HK68N3D is being accessed as a slave on the VMEbus.</td>
</tr>
<tr>
<td>1,3,5,7,9</td>
<td>Vcc</td>
<td>Vcc (+5) volts</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>P5 pin</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>INTR*</td>
<td>Connected to CIO bit A7, and pull-up Refer to section 9.2</td>
</tr>
<tr>
<td>12</td>
<td>Gnd</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>RESET*</td>
<td>When low, causes a local reset</td>
</tr>
<tr>
<td>14</td>
<td>Gnd</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Function (write-only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>02B0,0002_{hex}</td>
<td>MPU Cache Control</td>
</tr>
<tr>
<td></td>
<td>D0 = 0, cache disabled (default)</td>
</tr>
<tr>
<td></td>
<td>D0 = 1, caches enabled</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM Parity</strong></td>
<td>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 00007C₁₆. Parity checking cannot be disabled.</td>
</tr>
<tr>
<td><strong>Watchdog Timeout</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
**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.
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>
<thead>
<tr>
<th>Base Address</th>
<th>ROM</th>
<th>Component</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000,0000₁₆</td>
<td>0</td>
<td>U70</td>
<td></td>
</tr>
<tr>
<td>0040,0000₁₆</td>
<td>1</td>
<td>U80</td>
<td></td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>PROM Type</th>
<th>ROM Capacity</th>
<th>Total Board Capacity</th>
<th>Jumper Positions (for U70 and U80)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>J5 or J9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>J6, J7, J8, J10, J11, J12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2764</td>
<td>8 kilobytes</td>
<td>16 kilobytes</td>
<td>C</td>
</tr>
<tr>
<td>27128</td>
<td>16 kilobytes</td>
<td>32 kilobytes</td>
<td>C</td>
</tr>
<tr>
<td>27256</td>
<td>32 kilobytes</td>
<td>64 kilobytes</td>
<td>C</td>
</tr>
<tr>
<td>27512</td>
<td>64 kilobytes</td>
<td>128 kilobytes</td>
<td>B</td>
</tr>
<tr>
<td>27010</td>
<td>128 kilobytes</td>
<td>256 kilobytes</td>
<td>B</td>
</tr>
<tr>
<td>27020</td>
<td>256 kilobytes</td>
<td>512 kilobytes</td>
<td>B</td>
</tr>
<tr>
<td>27040</td>
<td>512 kilobytes</td>
<td>1 megabyte</td>
<td>B</td>
</tr>
<tr>
<td>27080</td>
<td>1 megabyte</td>
<td>2 megabytes</td>
<td>B</td>
</tr>
<tr>
<td>27513 paged</td>
<td>64 kilobytes</td>
<td>128 kilobytes</td>
<td>C</td>
</tr>
<tr>
<td>2864 R/W EEPROM</td>
<td>8 kilobytes</td>
<td>16 kilobytes</td>
<td>x</td>
</tr>
<tr>
<td>2817 R/W EEPROM</td>
<td>2 kilobytes</td>
<td>4 kilobytes</td>
<td>A</td>
</tr>
</tbody>
</table>
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.

32-pin socket

![Diagram of 32-pin socket with 28-pin device justification](image)

**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}. 

---

Revision A (Preliminary) / June 1991
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 0300,000016.

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 0300,000016.

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 at 0310,000016:
   a. Write the long word pattern 1234,5678 at the current boundary as 4-byte accesses (12 at 0310,000016; 23 at 0310,000116, 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 0320,000016; 0330,000016 ... 0340,000016 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

<table>
<thead>
<tr>
<th>CPU Speed (MHz)</th>
<th>Read Cycle (ns)</th>
<th>Write Cycle (ns)</th>
<th>Burst Cycle (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>15</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>100-nanosecond DRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Speed</td>
</tr>
<tr>
<td>32 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>80-nanosecond DRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Speed</td>
</tr>
<tr>
<td>32 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>60-nanosecond DRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Speed</td>
</tr>
<tr>
<td>32 MHz</td>
</tr>
</tbody>
</table>

60-nanosecond, 80-nanosecond and 100-nanosecond times are estimates based on existing 256K x 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

<table>
<thead>
<tr>
<th>Address Mode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0270,000016</td>
<td>Read/Write RAM contents (4 bits).</td>
</tr>
<tr>
<td>0270,000016</td>
<td>Recall RAM contents from nonvolatile memory.</td>
</tr>
<tr>
<td>0270,000016</td>
<td>Store RAM contents in nonvolatile memory. The 68030 \texttt{tas} (test and set) instruction must be used for this operation.</td>
</tr>
</tbody>
</table>

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 \texttt{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>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic number</td>
</tr>
<tr>
<td>Checksum</td>
</tr>
<tr>
<td>Accumulated number of writes</td>
</tr>
<tr>
<td>Board type, serial number and revision level</td>
</tr>
<tr>
<td>Hardware configuration information</td>
</tr>
<tr>
<td>Software configuration information</td>
</tr>
<tr>
<td>System configuration information</td>
</tr>
</tbody>
</table>
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**

LONG WORD. 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 02B0,000E1.
### 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**

<table>
<thead>
<tr>
<th>Function</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Clock (SYSCLK*)</td>
<td>J25 (install)</td>
</tr>
<tr>
<td>System Reset (SYSRESET*)</td>
<td>J19:2–3 (output)</td>
</tr>
<tr>
<td>Bus Clear (BCLR*)</td>
<td>J26 (install)</td>
</tr>
</tbody>
</table>

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.

Note: The Bus Request Level must match the Bus Grant Level.

**FIGURE 7-1. Bus request jumper settings, J16**

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

Revision A (Preliminary) / June 1991
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**

<table>
<thead>
<tr>
<th>Bus control bits</th>
<th>Bus Release Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC1</td>
<td>BC0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Slave mode control</th>
<th>Function (write-only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>02B0,000C₁₆</td>
<td>Slave mode enable</td>
</tr>
<tr>
<td>D0 = 0, slave disable</td>
<td></td>
</tr>
<tr>
<td>D0 = 1, slave enable</td>
<td></td>
</tr>
</tbody>
</table>

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.
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<sub>16</sub>.

2. Write a 32-bit long word to the bus control latch at address 02A0,0000<sub>16</sub>. 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<sub>16</sub>.
EXAMPLE 7-1. Bus control latch loading routine

```c
#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` 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 BCO and BC1
- Set the replacement address to 0 (base of RAM)
- Set the slave address to `40000000`

```
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>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>(reserved)</td>
</tr>
<tr>
<td>22</td>
<td>Indivisible Read Modify Writes</td>
</tr>
<tr>
<td>21</td>
<td>Slave Address Modifier 2</td>
</tr>
<tr>
<td>20</td>
<td>Slave Address Modifier 1</td>
</tr>
<tr>
<td>19</td>
<td>Slave Address Modifier 0</td>
</tr>
<tr>
<td>18</td>
<td>VMEbus Slave Release Without Hold</td>
</tr>
<tr>
<td>17</td>
<td>Bus Control BC 1</td>
</tr>
<tr>
<td>16</td>
<td>Bus Control BC 0</td>
</tr>
<tr>
<td>15</td>
<td>Replacement Address 23</td>
</tr>
<tr>
<td>14</td>
<td>Replacement Address 22</td>
</tr>
<tr>
<td>13</td>
<td>Replacement Address 21</td>
</tr>
<tr>
<td>12</td>
<td>Replacement Address 20</td>
</tr>
<tr>
<td>11</td>
<td>Slave Compare Address 23</td>
</tr>
<tr>
<td>10</td>
<td>Slave Compare Address 22</td>
</tr>
<tr>
<td>9</td>
<td>Slave Compare Address 21</td>
</tr>
<tr>
<td>8</td>
<td>Slave Compare Address 20</td>
</tr>
<tr>
<td>7</td>
<td>Slave Compare Address 31</td>
</tr>
<tr>
<td>6</td>
<td>Slave Compare Address 30</td>
</tr>
<tr>
<td>5</td>
<td>Slave Compare Address 29</td>
</tr>
<tr>
<td>4</td>
<td>Slave Compare Address 28</td>
</tr>
<tr>
<td>3</td>
<td>Slave Compare Address 27</td>
</tr>
<tr>
<td>2</td>
<td>Slave Compare Address 26</td>
</tr>
<tr>
<td>1</td>
<td>Slave Compare Address 25</td>
</tr>
<tr>
<td>0</td>
<td>Slave Compare Address 24</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>SAM2</th>
<th>SAM1</th>
<th>SAM0</th>
<th>Slave Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No slave access allowed (disable)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Standard data</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>No slave access allowed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Standard (all)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Extended supervisor data</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Extended data</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>No slave access allowed</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Extended (all)</td>
</tr>
</tbody>
</table>

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:
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>
<thead>
<tr>
<th>Table 7-7</th>
<th>VMEbus interrupter addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Vector Size</td>
</tr>
<tr>
<td>0290,0004₁₆</td>
<td>8</td>
</tr>
<tr>
<td>0290,0008₁₆</td>
<td>8</td>
</tr>
<tr>
<td>0290,000C₁₆</td>
<td>8</td>
</tr>
<tr>
<td>0290,0010₁₆</td>
<td>8</td>
</tr>
<tr>
<td>0290,0014₁₆</td>
<td>8</td>
</tr>
<tr>
<td>0290,0018₁₆</td>
<td>8</td>
</tr>
<tr>
<td>0290,001C₁₆</td>
<td>8</td>
</tr>
</tbody>
</table>

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 is 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**

<table>
<thead>
<tr>
<th>Interrupt acknowledge port summary</th>
<th>Address (read-only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority Level</td>
<td></td>
</tr>
<tr>
<td>IRQ1</td>
<td>0080,0003&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>IRQ2</td>
<td>0080,0005&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>IRQ3</td>
<td>0080,0007&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>IRQ4</td>
<td>0080,0009&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>IRQ5</td>
<td>0080,000B&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>IRQ6</td>
<td>0080,000D&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>IRQ7</td>
<td>0080,000F&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>On-card addresses</th>
<th>VMEbus Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C0,0000\textsubscript{16} through 00C0,FFFF\textsubscript{16}</td>
<td>VMEbus Short Address (0000\textsubscript{16} through FFFF\textsubscript{16})</td>
</tr>
<tr>
<td>0100,0000\textsubscript{16} through 01FF,FFFF\textsubscript{16}</td>
<td>VMEbus Standard (00,0000\textsubscript{16} through FF,FFFF\textsubscript{16})</td>
</tr>
<tr>
<td>0400,0000\textsubscript{16} through FFFF,FFFF\textsubscript{16}</td>
<td>VMEbus Extended (0400,0000\textsubscript{16} and up)</td>
</tr>
</tbody>
</table>

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\textsuperscript{*} to AM0\textsuperscript{*}) are 2D\textsubscript{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\textsubscript{16}.

TABLE 7-10
Mailbox control

<table>
<thead>
<tr>
<th>Address</th>
<th>Function (write-only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>02B0,0004\textsubscript{16}</td>
<td>Mailbox control</td>
</tr>
<tr>
<td></td>
<td>D0 = 0, disable (default)</td>
</tr>
<tr>
<td></td>
<td>D0 = 1, enable</td>
</tr>
</tbody>
</table>

TABLE 7-11
Mailbox functions

<table>
<thead>
<tr>
<th>Address</th>
<th>Function (Slave Mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbase + 0</td>
<td>ClO input D4 (see section 9.2) (mailbox interrupt)</td>
</tr>
<tr>
<td>Mbase + 2</td>
<td>HK68/V3D reset</td>
</tr>
<tr>
<td>Mbase + 4</td>
<td>On-card bus lock on</td>
</tr>
<tr>
<td>Mbase + 5</td>
<td>On-card bus lock off</td>
</tr>
<tr>
<td>Mbase + 6</td>
<td>MPU halt on</td>
</tr>
<tr>
<td>Mbase + 7</td>
<td>MPU halt off</td>
</tr>
</tbody>
</table>

The Mbase value is specified by 13 mailbox base bits in the mailbox address latch at address 02C0,0000\textsubscript{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\text{16}. The timer is enabled by writing a zero to address 02B0,0030\text{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\text{16}. The default state is disabled.
7.10 BUS CONTROL JUMPERS

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Function</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>J14</td>
<td>Bus Arbitration Level</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J15</td>
<td>Bus Arbitration Level</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J16</td>
<td>Bus Request Level</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J17</td>
<td>Bus Arbitration Level</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J18</td>
<td>Bus Arbitration Level</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J19</td>
<td>SYSRESET*</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J21</td>
<td>VMEbus Slave Window Size</td>
<td>See section 7.4</td>
</tr>
<tr>
<td>J22</td>
<td>VMEbus Slave Window Size</td>
<td>See section 7.4</td>
</tr>
<tr>
<td>J23</td>
<td>VMEbus Slave Window Size</td>
<td>See section 7.4</td>
</tr>
<tr>
<td>J24</td>
<td>VMEbus Slave Window Size</td>
<td>See section 7.4</td>
</tr>
<tr>
<td>J25</td>
<td>SYSCLK*</td>
<td>See section 7.3</td>
</tr>
<tr>
<td>J26</td>
<td>BCLR*</td>
<td>See section 7.3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>P1 Pin Number</th>
<th>Row A Signal Mnemonic</th>
<th>Row B Signal Mnemonic</th>
<th>Row C Signal Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D00</td>
<td>BBSY*</td>
<td>D08</td>
</tr>
<tr>
<td>2</td>
<td>D01</td>
<td>BCLR*</td>
<td>D09</td>
</tr>
<tr>
<td>3</td>
<td>D02</td>
<td>ACFAIL*</td>
<td>D10</td>
</tr>
<tr>
<td>4</td>
<td>D03</td>
<td>BG0IN*</td>
<td>D11</td>
</tr>
<tr>
<td>5</td>
<td>D04</td>
<td>BG0OUT*</td>
<td>D12</td>
</tr>
<tr>
<td>6</td>
<td>D05</td>
<td>BG1IN*</td>
<td>D13</td>
</tr>
<tr>
<td>7</td>
<td>D06</td>
<td>BG1OUT*</td>
<td>D14</td>
</tr>
<tr>
<td>8</td>
<td>D07</td>
<td>BG2IN*</td>
<td>D15</td>
</tr>
<tr>
<td>9</td>
<td>Gnd</td>
<td>BG2OUT*</td>
<td>Gnd</td>
</tr>
<tr>
<td>10</td>
<td>SYSCLK</td>
<td>BG3IN*</td>
<td>SYSFAIL*</td>
</tr>
<tr>
<td>11</td>
<td>Gnd</td>
<td>BG3OUT</td>
<td>BERR*</td>
</tr>
<tr>
<td>12</td>
<td>DS1*</td>
<td>BR0*</td>
<td>SYSRESET*</td>
</tr>
<tr>
<td>13</td>
<td>DS0*</td>
<td>BR1*</td>
<td>LWORD*</td>
</tr>
<tr>
<td>14</td>
<td>WRITE*</td>
<td>BR2*</td>
<td>AM5</td>
</tr>
<tr>
<td>15</td>
<td>Gnd</td>
<td>BR3*</td>
<td>A23</td>
</tr>
<tr>
<td>16</td>
<td>DTACK*</td>
<td>AM0</td>
<td>A22</td>
</tr>
<tr>
<td>17</td>
<td>Gnd</td>
<td>AM1</td>
<td>A21</td>
</tr>
<tr>
<td>18</td>
<td>AS*</td>
<td>AM2</td>
<td>A20</td>
</tr>
<tr>
<td>19</td>
<td>Gnd</td>
<td>AM3</td>
<td>A19</td>
</tr>
<tr>
<td>20</td>
<td>IACK*</td>
<td>Gnd</td>
<td>A18</td>
</tr>
<tr>
<td>21</td>
<td>IACKIN*</td>
<td>SERCLK</td>
<td>A17</td>
</tr>
<tr>
<td>22</td>
<td>IACKOUT*</td>
<td>SERDAT*</td>
<td>A16</td>
</tr>
<tr>
<td>23</td>
<td>AM4</td>
<td>Gnd</td>
<td>A15</td>
</tr>
<tr>
<td>24</td>
<td>A07</td>
<td>IRQ7*</td>
<td>A14</td>
</tr>
<tr>
<td>25</td>
<td>A06</td>
<td>IRQ6*</td>
<td>A13</td>
</tr>
<tr>
<td>26</td>
<td>A05</td>
<td>IRQ5*</td>
<td>A12</td>
</tr>
<tr>
<td>27</td>
<td>A04</td>
<td>IRQ4*</td>
<td>A11</td>
</tr>
<tr>
<td>28</td>
<td>A03</td>
<td>IRQ3*</td>
<td>A10</td>
</tr>
<tr>
<td>29</td>
<td>A02</td>
<td>IRQ2*</td>
<td>A09</td>
</tr>
<tr>
<td>30</td>
<td>A01</td>
<td>IRQ1*</td>
<td>A08</td>
</tr>
<tr>
<td>31</td>
<td>-12V</td>
<td>+5V STDBY</td>
<td>+12V</td>
</tr>
<tr>
<td>32</td>
<td>+5V</td>
<td>+5V</td>
<td>+5V</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>P2 Pin Number</th>
<th>Row A Signal Mnemonic</th>
<th>Row B Signal Mnemonic</th>
<th>Row C Signal Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AD00</td>
<td>+5</td>
<td>AD01</td>
</tr>
<tr>
<td>2</td>
<td>AD02</td>
<td>Gnd</td>
<td>AD03</td>
</tr>
<tr>
<td>3</td>
<td>AD04</td>
<td>(reserved)</td>
<td>AD05</td>
</tr>
<tr>
<td>4</td>
<td>AD06</td>
<td>A24</td>
<td>AD07</td>
</tr>
<tr>
<td>5</td>
<td>AD08</td>
<td>A25</td>
<td>AD09</td>
</tr>
<tr>
<td>6</td>
<td>AD10</td>
<td>A26</td>
<td>AD11</td>
</tr>
<tr>
<td>7</td>
<td>AD12</td>
<td>A27</td>
<td>AD13</td>
</tr>
<tr>
<td>8</td>
<td>AD14</td>
<td>A28</td>
<td>AD15</td>
</tr>
<tr>
<td>9</td>
<td>AD16</td>
<td>A29</td>
<td>AD17</td>
</tr>
<tr>
<td>10</td>
<td>AD18</td>
<td>A30</td>
<td>AD19</td>
</tr>
<tr>
<td>11</td>
<td>AD20</td>
<td>A31</td>
<td>AD21</td>
</tr>
<tr>
<td>12</td>
<td>AD22</td>
<td>Gnd</td>
<td>AD23</td>
</tr>
<tr>
<td>13</td>
<td>AD24</td>
<td>+5</td>
<td>AD25</td>
</tr>
<tr>
<td>14</td>
<td>AD26</td>
<td>D16</td>
<td>AD27</td>
</tr>
<tr>
<td>15</td>
<td>AD28</td>
<td>D17</td>
<td>AD29</td>
</tr>
<tr>
<td>16</td>
<td>AD30</td>
<td>D18</td>
<td>AD31</td>
</tr>
<tr>
<td>17</td>
<td>Gnd</td>
<td>D19</td>
<td>Gnd</td>
</tr>
<tr>
<td>18</td>
<td>IRQ*</td>
<td>D20</td>
<td>Gnd</td>
</tr>
<tr>
<td>19</td>
<td>DS*</td>
<td>D21</td>
<td>Gnd</td>
</tr>
<tr>
<td>20</td>
<td>WR*</td>
<td>D22</td>
<td>Gnd</td>
</tr>
<tr>
<td>21</td>
<td>SPACE0</td>
<td>D23</td>
<td>SIZE0</td>
</tr>
<tr>
<td>22</td>
<td>SPACE1</td>
<td>Gnd</td>
<td>PAS*</td>
</tr>
<tr>
<td>23</td>
<td>LOCK*</td>
<td>D24</td>
<td>SIZE1</td>
</tr>
<tr>
<td>24</td>
<td>ERR*</td>
<td>D25</td>
<td>Gnd</td>
</tr>
<tr>
<td>25</td>
<td>Gnd</td>
<td>D26</td>
<td>ACK*</td>
</tr>
<tr>
<td>26</td>
<td>Gnd</td>
<td>D27</td>
<td>AC</td>
</tr>
<tr>
<td>27</td>
<td>Gnd</td>
<td>D28</td>
<td>ASACK1*</td>
</tr>
<tr>
<td>28</td>
<td>Gnd</td>
<td>D29</td>
<td>ASACK0*</td>
</tr>
<tr>
<td>29</td>
<td>Gnd</td>
<td>D30</td>
<td>CACHE*</td>
</tr>
<tr>
<td>30</td>
<td>Gnd</td>
<td>D31</td>
<td>WAIT*</td>
</tr>
<tr>
<td>31</td>
<td>BGIN*</td>
<td>Gnd</td>
<td>BUSY*</td>
</tr>
<tr>
<td>32</td>
<td>BREQ*</td>
<td>+5</td>
<td>BGOUT*</td>
</tr>
</tbody>
</table>
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).

![7-segment display diagram]

**FIGURE 8-1. 7-segment display**

**TABLE 8-1**  
Addresses for the 7-segment display

<table>
<thead>
<tr>
<th>Segment</th>
<th>Address (write-only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>02B0,0010&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>b</td>
<td>02B0,0020&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>c</td>
<td>02B0,0030&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>d</td>
<td>02B0,0040&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>e</td>
<td>02B0,0050&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>f</td>
<td>02B0,0060&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>g</td>
<td>02B0,0070&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
<th>Data Path Polarity</th>
<th>Interface</th>
<th>HK68/V3D User's Manual Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>External Interrupt</td>
<td>Negative True</td>
<td>P5-11</td>
<td>3.3</td>
</tr>
<tr>
<td>D6</td>
<td>EEPROM 1 Ready (U80)</td>
<td>Positive True</td>
<td>U80-1</td>
<td>6.2</td>
</tr>
<tr>
<td>D5</td>
<td>EEPROM 0 Ready (U70)</td>
<td>Positive True</td>
<td>U70-16</td>
<td>6.2</td>
</tr>
<tr>
<td>D4</td>
<td>Mailbox Interrupt</td>
<td>Negative True</td>
<td>—</td>
<td>7.8</td>
</tr>
<tr>
<td>D3</td>
<td>unused</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>D2</td>
<td>VME Interrupt in Progress</td>
<td>Negative True</td>
<td>—</td>
<td>7.5</td>
</tr>
<tr>
<td>D1</td>
<td>SCSI Reset</td>
<td>Positive True</td>
<td>P2-A20</td>
<td>11</td>
</tr>
<tr>
<td>D0</td>
<td>Mailbox Halt</td>
<td>Positive True</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
<th>Data Path Polarity</th>
<th>Interface</th>
<th>HK68/V3D User's Manual Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>Software interrupt</td>
<td>Negative True</td>
<td>Interrupt Switch on front panel</td>
<td>12</td>
</tr>
<tr>
<td>D6</td>
<td>IRQ7 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
<tr>
<td>D5</td>
<td>IRQ6 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
<tr>
<td>D4</td>
<td>IRQ5 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
<tr>
<td>D3</td>
<td>IRQ4 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
<tr>
<td>D2</td>
<td>IRQ3 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
<tr>
<td>D1</td>
<td>IRQ2 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
<tr>
<td>D0</td>
<td>IRQ1 enable</td>
<td>Negative True</td>
<td>P1</td>
<td>3.2</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
<th>Data Path Polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3</td>
<td>VMEbus ACFAIL</td>
<td>Positive True</td>
</tr>
<tr>
<td>D2</td>
<td>VMEbus SYSFAIL*</td>
<td>Negative True</td>
</tr>
<tr>
<td>D1</td>
<td>VMEbus SYSOK* utility bit</td>
<td>Positive True</td>
</tr>
<tr>
<td>D0</td>
<td>Port A Ring Indicator</td>
<td>Negative True</td>
</tr>
</tbody>
</table>
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.

$$\text{TC} = \frac{2,457,600}{\text{interrupt rate (in Hz)}}$$

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>
<thead>
<tr>
<th>TABLE 9-4</th>
<th>CIO register addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Address</td>
</tr>
<tr>
<td>Port C, Data</td>
<td>02D0,0001&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>Port B, Data</td>
<td>02D0,0003&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>Port A, Data</td>
<td>02D0,0005&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>Control Registers</td>
<td>02D0,0007&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

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.

Revision A (Preliminary) / June 1991
EXAMPLE 9-1. CiO program (C portion)

```c
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 = 0x00; 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 &OxFFFF,-(%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>
<thead>
<tr>
<th>TABLE 10-1A</th>
<th>Port A serial port pin assignments, P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3 Pin Number</td>
<td>&quot;D&quot; Pin Number</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 10-1B</th>
<th>Port B serial port pin assignments, P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3 Pin Number</td>
<td>&quot;D&quot; Pin Number</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>20</td>
</tr>
</tbody>
</table>

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 CI0. 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

<table>
<thead>
<tr>
<th>SCC Signal</th>
<th>Interface Signal</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx Data</td>
<td>Rcv Data</td>
<td>to device</td>
</tr>
<tr>
<td>Rcv Data</td>
<td>Tx Data</td>
<td>from device</td>
</tr>
<tr>
<td>Tx Clock</td>
<td>Rcv Clock</td>
<td>from device (port A)</td>
</tr>
<tr>
<td>Tx Clock</td>
<td>Rcv Clock</td>
<td>to device (port B)</td>
</tr>
<tr>
<td>Rcv Clock</td>
<td>Tx Clock</td>
<td>from device</td>
</tr>
<tr>
<td>RTS</td>
<td>DSR</td>
<td>to device</td>
</tr>
<tr>
<td>CTS</td>
<td>DTR</td>
<td>from device</td>
</tr>
<tr>
<td>DTR</td>
<td>CTS</td>
<td>to device</td>
</tr>
<tr>
<td>DCD</td>
<td>RTS</td>
<td>from device</td>
</tr>
<tr>
<td>—</td>
<td>Ring Indicator</td>
<td>from device</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SCC Signal</th>
<th>P3 Pin #</th>
<th>&quot;D&quot; Pin # at HK68/V3D</th>
<th>&quot;D&quot; Pin # at modem</th>
<th>RS-232 Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>Protective Ground</td>
</tr>
<tr>
<td>Rcv Data</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Rcv Data</td>
</tr>
<tr>
<td>Tx Data</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>Tx Data</td>
</tr>
<tr>
<td>DCD</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>DSR</td>
</tr>
<tr>
<td>RTS</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>RTS</td>
</tr>
<tr>
<td>DTR</td>
<td>7</td>
<td>5</td>
<td>20</td>
<td>DTR</td>
</tr>
<tr>
<td>CTS</td>
<td>12</td>
<td>20</td>
<td>5</td>
<td>CTS</td>
</tr>
<tr>
<td>Ring Indicator</td>
<td>8</td>
<td>18</td>
<td>22</td>
<td>Ring Indicator</td>
</tr>
<tr>
<td>Signal Ground</td>
<td>11</td>
<td>7</td>
<td>7</td>
<td>Signal Ground</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Data</th>
<th>Data Register Address (write)</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>02F0,0003_16</td>
<td>Reset SCC register counter</td>
</tr>
<tr>
<td>09,C0</td>
<td>02F0,0003_16</td>
<td>Force reset (for port A only)</td>
</tr>
<tr>
<td>04,4C</td>
<td>02F0,0003_16</td>
<td>Async mode, x16 clock, 2 stop bits</td>
</tr>
<tr>
<td>05,EA</td>
<td>02F0,0003_16</td>
<td>Tx: RTS, Enable, 8 data bits</td>
</tr>
<tr>
<td>03,E1</td>
<td>02F0,0003_16</td>
<td>Rcv: Enable, 8 data bits</td>
</tr>
<tr>
<td>01,00</td>
<td>02F0,0003_16</td>
<td>No Interrupt, Update status</td>
</tr>
<tr>
<td>0B,56</td>
<td>02F0,0003_16</td>
<td>No Xtal, Tx &amp; Rcv clk internal, BR out</td>
</tr>
<tr>
<td>0C,baudL</td>
<td>02F0,0003_16</td>
<td>Set Low half of baud rate constant</td>
</tr>
<tr>
<td>0D,baudH</td>
<td>02F0,0003_16</td>
<td>Set high half of baud rate constant</td>
</tr>
<tr>
<td>0E,03</td>
<td>02F0,0003_16</td>
<td>Null, BR enable</td>
</tr>
</tbody>
</table>

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 \texttt{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 \texttt{clr.b}. Use a variable that is set to zero, or output the values from a lookup table. For example, this is correct:

\begin{verbatim}
zero = 0;
*SCCctrl = 0x20;
*SCCctrl = zero;
\end{verbatim}

### 10.6 PORT ADDRESS SUMMARY

#### Table 10-5
**SCC register addresses**

<table>
<thead>
<tr>
<th>Register</th>
<th>Port A</th>
<th>Port B</th>
<th>Port C</th>
<th>Port D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>$02\text{F}0,0003_{16}$</td>
<td>$02\text{F}0,0001_{16}$</td>
<td>$02\text{E}0,0003_{16}$</td>
<td>$02\text{E}0,0001_{16}$</td>
</tr>
<tr>
<td>Data</td>
<td>$02\text{F}0,0007_{16}$</td>
<td>$02\text{F}0,0005_{16}$</td>
<td>$02\text{E}0,0007_{16}$</td>
<td>$02\text{E}0,0005_{16}$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>x1 clock rate</th>
<th>x16 clock rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>22,340</td>
<td>1,394</td>
</tr>
<tr>
<td>300</td>
<td>8,190</td>
<td>510</td>
</tr>
<tr>
<td>1200</td>
<td>2,046</td>
<td>126</td>
</tr>
<tr>
<td>2400</td>
<td>1,022</td>
<td>62</td>
</tr>
<tr>
<td>4800</td>
<td>510</td>
<td>30</td>
</tr>
<tr>
<td>9600</td>
<td>254</td>
<td>14</td>
</tr>
<tr>
<td>19,200</td>
<td>126</td>
<td>6</td>
</tr>
<tr>
<td>38,400</td>
<td>62</td>
<td>2</td>
</tr>
</tbody>
</table>

The time constant values listed above are computed as follows:

\[ TC = \frac{4,915,200}{(2 \times \text{baud} \times \text{factor})} - 2 \]

The x16 mode will obtain better results with asynchronous protocols because the receiver can search for the middle of the start bit. (In fact, the x1 mode will probably produce frequent receiver errors.)

The maximum SCC data speed is one megabit per second, using the x1 clock and synchronous mode. For asynchronous transmission, the maximum practical rate using the x16 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 |

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Function</th>
<th>Options</th>
<th>Standard Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>J2</td>
<td>RS-232 ports A and B status default</td>
<td>J2-A (True)  J2-B (False)</td>
<td>J2-A (True)</td>
</tr>
<tr>
<td>J3</td>
<td>Selects Ring Indicator of Data Carrier</td>
<td>J3:1–2 (RI)  J3:2–3 (DCD)</td>
<td>J3:2–3 (DCD)</td>
</tr>
</tbody>
</table>
FIGURE 10-2. Serial I/O cable
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

<table>
<thead>
<tr>
<th>Address</th>
<th>R/W</th>
<th>Bits</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0230,0001&lt;sub&gt;16&lt;/sub&gt;</td>
<td>W</td>
<td>8</td>
<td>Set Controller Address Register</td>
</tr>
<tr>
<td>0230,0001&lt;sub&gt;16&lt;/sub&gt;</td>
<td>R</td>
<td>8</td>
<td>Read Auxiliary Register</td>
</tr>
<tr>
<td>0230,0003&lt;sub&gt;16&lt;/sub&gt;</td>
<td>RW</td>
<td>8</td>
<td>SCSI Controller Registers</td>
</tr>
<tr>
<td>0240,0000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>RW</td>
<td>8</td>
<td>SCSI Data Register (pseudo-DMA)</td>
</tr>
<tr>
<td>02B0,0006&lt;sub&gt;16&lt;/sub&gt;</td>
<td>W</td>
<td>1</td>
<td>SCSI Bus Reset (1=reset, 0=release)</td>
</tr>
<tr>
<td>02B0,0020&lt;sub&gt;16&lt;/sub&gt;</td>
<td>W</td>
<td>1</td>
<td>SCSI Interrupt Enable (1=enable)</td>
</tr>
</tbody>
</table>

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**

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

![SCSI BUS TERMINATION Diagram]

**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.
Optional Ethernet Interface

12.1 INTRODUCTION

The HK68/V3D can be ordered 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).

![Ethernet Address Diagram](image)

**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 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 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,FFF4'6.

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, 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.

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>
<thead>
<tr>
<th>Access</th>
<th>R/W</th>
<th>Address</th>
<th>D19-D16</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT</td>
<td>W</td>
<td>02E0,0000₁₆</td>
<td>0</td>
<td>Reset the 82596CA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Perform a self test on the 82596CA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Write a new SCP address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>Dump the 82596CA registers.</td>
</tr>
<tr>
<td>CA</td>
<td>W</td>
<td>02E0,0004₁₆</td>
<td>X</td>
<td>Channel Attention</td>
</tr>
</tbody>
</table>

Revision A (Preliminary) / June 1991
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.

PORT accesses require four writes on the HK68/V3D. Section 5.3.6.3 of the 82596 User's Manual says that all PORT accesses must be 16-bit accesses. Thus, a 32-bit Port command requires two writes to the 82596CA's 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 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>
<thead>
<tr>
<th>TABLE 12-2</th>
<th>Port access definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Access</strong></td>
<td><strong>Second Access</strong></td>
</tr>
<tr>
<td>Big endian</td>
<td>D15-D0 &gt; Lower Command Word</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>D31 ............ D20</th>
<th>D19</th>
<th>D18</th>
<th>D17</th>
<th>D16</th>
<th>D15 ............ D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>A15... Don't care ...A4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A31... Don't care ...A16</td>
</tr>
<tr>
<td>Self-test</td>
<td>A15... Self-test results address ...A4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A31... Self-test results address ...A16</td>
<td></td>
</tr>
<tr>
<td>New SCP</td>
<td>A15... Alternate SCP address ...A4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A31... Alternate SCP address ...A16</td>
</tr>
<tr>
<td>Dump</td>
<td>A15... Dump area pointer ...A4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A31... Dump area pointer ...A16</td>
</tr>
</tbody>
</table>

12.5.2 Channel Attention (CA)

Accessing address 02E0,0000₁₆ 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,FFF4₁₆ or an alternate SCP address written to the 82596CA using the 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,FFF4₁₆) 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.

```
    7 6 5 4 3 2 1 0
 0 1 0 1 0 1 0 0
```

- Not used. Intel requires setting to 0.
- Intel requires setting to 1.
- Must be set to 0.
- Address mode select
- Bus Throttle Timer setting
- Must be set to 0.

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

**TABLE 12-4**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
<th>Selections</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not used</td>
<td>—</td>
<td>This bit must be set to 0₂, according to Intel documentation.</td>
</tr>
<tr>
<td>2 and 1</td>
<td>Address Mode select</td>
<td>00₂ = 82586 mode 01₂ = 32-bit segmented mode 10₂ = linear mode 11₂ = reserved</td>
<td>The HK68/V3D supports linear mode (bit 2:1 = 10₂). 32-bit segmented mode should also work but is not supported by Heurikon. 82586 mode cannot be used.</td>
</tr>
<tr>
<td>3</td>
<td>Bus Throttle Timer triggering</td>
<td>0₂ = internal 1₂ = external</td>
<td>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 0₂.</td>
</tr>
<tr>
<td>4</td>
<td>Locked cycles enable</td>
<td>0₂ = enable 1₂ = disable</td>
<td>Locked cycles are an option that can be used for updating the Ethernet statistics counter. Both selections are supported on the HK68/V3D.</td>
</tr>
<tr>
<td>5</td>
<td>82596CA interrupt</td>
<td>0₂ = active high 1₂ = active low</td>
<td>Logic on the board expects the 82596CA's INT signal to be active high. Bit 5 must be 0₂.</td>
</tr>
<tr>
<td>6</td>
<td>—</td>
<td>1₂</td>
<td>The default must be used, according to an erratum from Intel.</td>
</tr>
<tr>
<td>7</td>
<td>Not used</td>
<td>—</td>
<td>This bit must be set to 0₂, according to Intel documentation.</td>
</tr>
</tbody>
</table>
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 44₁₆ or 54₁₆

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,XX00₁₆.

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>02E0,0000(_{16})</td>
<td>Write: 82596CA Port command</td>
<td>This address MUST be accessed as a long word. Writing to this address generates the 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.</td>
</tr>
<tr>
<td></td>
<td>Read: Not used.</td>
<td></td>
</tr>
<tr>
<td>02E0,0004(_{16})</td>
<td>Write: 82596CA Channel Attention command</td>
<td>Writing to this address generates the CA signal to the 82596CA. The data for the write is of no consequence.</td>
</tr>
<tr>
<td></td>
<td>Read: Not used.</td>
<td></td>
</tr>
<tr>
<td>02E0,0008(_{16})</td>
<td>Write: Not used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read: Not used.</td>
<td></td>
</tr>
<tr>
<td>02E0,000C(_{16})</td>
<td>Write: Ethernet section interrupt clear.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Read: Not used.</td>
<td></td>
</tr>
<tr>
<td>02E0,0010(_{16})</td>
<td>Write: 82596CA interrupt enable/disable</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Read: 82596CA interrupt enable/disable</td>
<td></td>
</tr>
<tr>
<td>02E0,0014(_{16})</td>
<td>Write: Abort interrupt enable/disable</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Read: Abort interrupt enable/disable</td>
<td></td>
</tr>
<tr>
<td>02E0,0018(_{16})</td>
<td>Write: Not used.</td>
<td>Reading from this address returns the state of the 82596CA interrupt status as the least significant bit. A 1 bit indicates the interrupt is asserted; 0 indicates not asserted. The other bits are undefined.</td>
</tr>
<tr>
<td></td>
<td>Read: 82596CA interrupt status</td>
<td></td>
</tr>
<tr>
<td>02E0,001C(_{16})</td>
<td>Write: Clear abort interrupt.</td>
<td>Writing to this address clears the 82596CA abort condition. The data for the write is of no consequence.</td>
</tr>
<tr>
<td></td>
<td>Read: Abort interrupt status</td>
<td>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.</td>
</tr>
<tr>
<td>02E0,0020(_{16})</td>
<td>Not used.</td>
<td></td>
</tr>
<tr>
<td>02E0,0037(_{16})</td>
<td>Not used.</td>
<td></td>
</tr>
</tbody>
</table>

Continues
TABLE 12-5 — Continued
Ethernet addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>02E0,0038&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Write: Hardware trigger point #1</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Read: Not used.</td>
<td></td>
</tr>
<tr>
<td>02E0,003C&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Write: Hardware trigger point #2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read: Not used</td>
<td></td>
</tr>
</tbody>
</table>

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 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 BOFF signal and keeps it asserted until the ABORT condition is cleared. Asserting the 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 BOFF is asserted and will take precedence over any transfers that were in progress when 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 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>
<thead>
<tr>
<th>Position</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1 installed</td>
<td>+ (positive) idle differential voltage on TX lines full-step mode</td>
</tr>
<tr>
<td></td>
<td>(for example, for Ethernet 1.0-type transceivers)</td>
</tr>
<tr>
<td>J1 not installed</td>
<td>0 idle differential voltage on TX lines half-step mode</td>
</tr>
<tr>
<td></td>
<td>(for example, for IEEE-802.3-type transceivers)</td>
</tr>
</tbody>
</table>

12.11 ETHERNET PORT PIN ASSIGNMENTS

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

![Ethernet connector, P4](image-url)

**FIGURE 12-3. Ethernet connector, P4**
### TABLE 12-7
**Ethernet connector pin assignments, P4**

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Name</th>
<th>Description</th>
<th>Direction</th>
<th>Transceiver Cable D Connector Pin Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLSNShld</td>
<td>Control In circuit Shield</td>
<td>In</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CLSN-</td>
<td>Control In circuit -</td>
<td>In</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>CLSN+</td>
<td>Control In circuit +</td>
<td>In</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>TX-</td>
<td>Data Out circuit -</td>
<td>Out</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>TX+</td>
<td>Data Out circuit +</td>
<td>Out</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>TXShld</td>
<td>Transmit Shield</td>
<td>In</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>RXShld</td>
<td>Data In circuit Shield</td>
<td>In</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>RX-</td>
<td>Data In circuit -</td>
<td>In</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>RX+</td>
<td>Data In circuit +</td>
<td>In</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>VPLUS</td>
<td>Voltage Plus</td>
<td>Out</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>VCMN</td>
<td>Voltage Common</td>
<td>In</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>VShld</td>
<td>Voltage Shield</td>
<td>In</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>CTLO+</td>
<td>Control Out circuit +</td>
<td>Not connected</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>CTLO-</td>
<td>Control Out circuit -</td>
<td>Not connected</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>CTLOShld</td>
<td>Control Out circuit Shield</td>
<td>In</td>
<td>8</td>
</tr>
</tbody>
</table>
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 VCC 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.

![Diagram of RTC option and socket placement](image_url)
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 installation on board height**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Component Height Above Board</th>
<th>Minimum Board Spacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC module plugged into existing ROM socket:</td>
<td>.75 in.</td>
<td>.85 in. (2 slots)</td>
</tr>
</tbody>
</table>

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

```c
#define WATCHBASE (unsigned char *) 0x00000000 /* ROM socket */
#define WRO_WATCH (unsigned char *) (WATCHBASE+2) /* write 0 */
#define WR1_WATCH (unsigned char *) (WATCHBASE+3) /* write 1 */
#define RD_WATCH (unsigned char *) (WATCHBASE+4) /* read */

#include <stdio.h>

struct rtc_data { /* D7 D6 D5 D4 D3 D2 D1 D0 range */
    unsigned char dotsec; /* 0.01 sec: 00-99 */
    unsigned char sec; /* 00-59 */
    unsigned char min; /* 00-59 */
    unsigned char hour; /* A 0 0 B Hr: hours: 00-23 */
    unsigned char day; /* 0 0 0 0 1-day--; 01-07 */
    unsigned char date; /* 00-31 */
    unsigned char month; /* 01-12 */
    unsigned char 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 AM or "1" for AM (if 01-12 hour mode) */

#define 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 */
        0x00, 0x3A, 0xA3, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    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 : *WRO_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++ )
        for( bit = 1; bit & 0xff; bit <<= 1 )
            *data |= (*RD_WATCH & 1) ? bit : 0;
}
```

Revision A (Preliminary) / June 1991
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**

<table>
<thead>
<tr>
<th>32-pin RTC Pin Number</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RST\</td>
<td>RESET</td>
</tr>
<tr>
<td>10</td>
<td>A2</td>
<td>Address Bit 2 (READ/WRITE)</td>
</tr>
<tr>
<td>12</td>
<td>A0</td>
<td>Address Bit 0 (Data Input)</td>
</tr>
<tr>
<td>13</td>
<td>DQ0</td>
<td>I/O₀ (Data Output)</td>
</tr>
<tr>
<td>16</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>22</td>
<td>CE\</td>
<td>Conditioned Chip Enable</td>
</tr>
<tr>
<td>24</td>
<td>OE\</td>
<td>Output Enable</td>
</tr>
<tr>
<td>32</td>
<td>VCC</td>
<td>+5 VDC to the socket</td>
</tr>
</tbody>
</table>

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.
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

---

**FIGURE 13-2. RTC comparison register definition**

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>7 6 5 4 3 2 1 0</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 1 0 0 0 1 0 1</td>
<td>C5</td>
</tr>
<tr>
<td>Byte 1</td>
<td>0 0 1 1 1 0 1 0</td>
<td>3A</td>
</tr>
<tr>
<td>Byte 2</td>
<td>1 0 1 0 0 0 1 1</td>
<td>A3</td>
</tr>
<tr>
<td>Byte 3</td>
<td>0 1 0 1 1 1 0 0</td>
<td>5C</td>
</tr>
<tr>
<td>Byte 4</td>
<td>1 1 0 0 0 1 0 1</td>
<td>C5</td>
</tr>
<tr>
<td>Byte 5</td>
<td>0 0 1 1 1 0 1 0</td>
<td>3A</td>
</tr>
<tr>
<td>Byte 6</td>
<td>1 0 1 0 0 0 1 1</td>
<td>A3</td>
</tr>
<tr>
<td>Byte 7</td>
<td>0 1 0 1 1 1 0 0</td>
<td>5C</td>
</tr>
</tbody>
</table>
occur during pattern recognition or while accessing the RTC registers. \texttt{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 \texttt{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 \texttt{RESET} and oscillator functions. Bit 4 controls the \texttt{RESET} (pin 1). When the \texttt{RESET} bit is set to logic 1, the \texttt{RESET} input pin is ignored. When the \texttt{RESET} bit is set to logic 0, a low input on the \texttt{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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Range (BCD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1 second</td>
<td>00-99</td>
</tr>
<tr>
<td>1</td>
<td>0.01 second</td>
<td>00-99</td>
</tr>
<tr>
<td>2</td>
<td>10 seconds</td>
<td>00-59</td>
</tr>
<tr>
<td>3</td>
<td>10 minutes</td>
<td>00-59</td>
</tr>
<tr>
<td>4</td>
<td>10 hours</td>
<td>00-23</td>
</tr>
<tr>
<td>5</td>
<td>10 days</td>
<td>01-07</td>
</tr>
<tr>
<td>6</td>
<td>10 dates</td>
<td>01-31</td>
</tr>
<tr>
<td>7</td>
<td>10 months</td>
<td>01-12</td>
</tr>
<tr>
<td>8</td>
<td>10 years</td>
<td>00-99</td>
</tr>
</tbody>
</table>

**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.
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

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

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Standard Configuration</th>
<th>Options</th>
<th>Function</th>
<th>HK68/V3D Manual Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Installed</td>
<td>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)</td>
<td>Selects Ethernet differential voltage</td>
<td>12</td>
</tr>
<tr>
<td>J3</td>
<td>J3:1-2 Ring Indicator</td>
<td>J3:1-2 Ring Indicator J3:2-3 Data Carrier Detect</td>
<td>Selects Ring Indicator or Data Carrier Detect for SCC Port A.</td>
<td>10</td>
</tr>
<tr>
<td>J5–J8</td>
<td>Matches ROM0 size. See Table 14-3.</td>
<td>2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27153 paged, 2864 R/W EEPROM, 2817 R/W EEPROM</td>
<td>Selects ROM 0 size (default is 2764)</td>
<td>5</td>
</tr>
<tr>
<td>J9–J12</td>
<td>Matches ROM1 size. See Table 14-3.</td>
<td>2764, 27128, 27256, 27512, 27010, 27020, 27040, 27080, 27153 paged, 2864 R/W EEPROM, 2817 R/W EEPROM</td>
<td>Selects ROM 1 size (default is 2764)</td>
<td>5</td>
</tr>
<tr>
<td>J14, J15, J17, J18</td>
<td>Bus Grant Level 3</td>
<td>Bus Grant Level 3 Bus Grant Level 2 Bus Grant Level 1 Bus Grant Level 0</td>
<td>Selects VMEbus Bus Grant level</td>
<td>7</td>
</tr>
<tr>
<td>J16</td>
<td>Bus Request Level 3)</td>
<td>Bus Request Level 3 Bus Request Level 2 Bus Request Level 1 Bus Request Level 0</td>
<td>VMEbus arbitration (bus request level 3, not system controller)</td>
<td>7</td>
</tr>
</tbody>
</table>
**J19**
- J19:1-2 input from VMEbus
- J19:2-3 output to VMEbus
- Enables VMEbus SYSRESET*

**J20**
- Removed
- J20 installed: Allows HK68/V3D to respond to ACFAIL* interrupt.
- J21 removed: HK68/V3D does not respond to ACFAIL* interrupt.

**J21-J24**
- Matches memory size.
- 1, 2, 4, 8, or 16 megabytes
- VMEbus slave window size

**J25**
- Removed
- J25 installed: drives SYSCLK
- J25 removed: does not drive SYSCLK

**J26**
- Removed
- J26 installed: HK68/V3D can drive BCLR*.
- J26 removed: HK68/V3D cannot drive BCLR*.

**J91**
- Factory set for memory configuration.
- Do not alter.

**J92**
- Factory set for memory configuration.
- Do not alter.

---

**TABLE 14-3**

<table>
<thead>
<tr>
<th>ROM Type</th>
<th>ROM Capacity</th>
<th>Jumper Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>2764</td>
<td>64 Kbits (8K x 8)</td>
<td>J5 or J9</td>
</tr>
<tr>
<td>27128</td>
<td>128 Kbits (16K x 8)</td>
<td>J6 or J10</td>
</tr>
<tr>
<td>27513</td>
<td>256 Kbits (32K x 8)</td>
<td>J7 or J11 (either A or B)</td>
</tr>
<tr>
<td>27256</td>
<td>512 Kbits (64K x 8)</td>
<td>J5 or J9</td>
</tr>
<tr>
<td>27512</td>
<td>512 Kbits (64K x 8)</td>
<td>J6 or J10</td>
</tr>
<tr>
<td>27010</td>
<td>1 Mbits (128K x 8)</td>
<td>J7 or J11 (either A or B)</td>
</tr>
<tr>
<td>27020</td>
<td>2 Mbits (256K x 8)</td>
<td>J5 or J9</td>
</tr>
<tr>
<td>27040</td>
<td>4 Mbits (512K x 8)</td>
<td>J6 or J10</td>
</tr>
</tbody>
</table>

Revision A (Preliminary) / June 1991
**Figure 14-1. Jumper locations**
14.4 POWER REQUIREMENTS

**TABLE 14-4**  
*Power requirements*

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Current</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5</td>
<td>7.0A, max</td>
<td>All logic</td>
</tr>
<tr>
<td>+12</td>
<td>20ma, max</td>
<td>RS-232 interface</td>
</tr>
<tr>
<td>-12</td>
<td>20ma, max</td>
<td>RS-232 interface</td>
</tr>
</tbody>
</table>

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*

<table>
<thead>
<tr>
<th>Width</th>
<th>Depth</th>
<th>Height (above board)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.187 in.</td>
<td>6.299 in.</td>
<td>0.6 in. (0.8 in.)</td>
</tr>
<tr>
<td>233.35 mm</td>
<td>160 mm</td>
<td>15.25 mm (20.35 mm)</td>
</tr>
</tbody>
</table>

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.

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

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:

<table>
<thead>
<tr>
<th>Device Address</th>
<th>Byte Offsets</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0270,0000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>0 - 15FF&lt;sub&gt;16&lt;/sub&gt;</td>
<td>User-defined data area</td>
</tr>
<tr>
<td>0270,B000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>1600&lt;sub&gt;16&lt;/sub&gt; - 17FF&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Monitor/board initialization</td>
</tr>
<tr>
<td>0270,C000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>1800&lt;sub&gt;16&lt;/sub&gt; - 1FF&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Manufacturing/service hardware information (write protected)</td>
</tr>
</tbody>
</table>

Revision A (Preliminary) / June 1991
MONITOR INSTALLATION AND SETUP

J2 - Ports A and B defaults

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A = True (+12V)
B = False (-12V)

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<sub>16</sub>.
- Slave standard space mapped to 000000<sub>16</sub>.
- Address modifier select is “ExAll”.
- Slave standard space replacement address is 0000,0000<sub>16</sub>.
- 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<sub>16</sub>. ROM base is at 00400000<sub>16</sub>. ROM size is 00020000<sub>16</sub>. 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 report
HK68/V3D Power Up Serial Test PASSED
HK68/V3D Power Up Memory Test PASSED
Copyright Heurikon Corp., 1991
Created: Fri Mar 8 08:54:12 1991
Hit 'H' to skip auto-boot

Memory Test report

Prompt to cancel nonvolatile configuration and nonvolatile autoboot

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 0000,0000'6 and 0001,0000'6 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
   Baud   9600
   Parity None
   Data   8-bits
   StopBits 2-bits
   XonXoff On
   ChBaudOnBreak False
   RstOnBreak False

[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.

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 \texttt{nvdisplay}, namely, Hardware, Manufacturing, and Service, are write-protected. Attempts to modify these fields result in the display of an error message:

\begin{center}
Warning, protected region was not modified.
\end{center}

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

READING AND WRITING MEMORY

Use \texttt{readmem} or \texttt{displaymem} to read memory, and \texttt{writemem} or \texttt{setmem} to write memory.

\begin{description}
\item[Required flags] \texttt{readmem}, \texttt{writemem} and \texttt{setmem} require one of the following flags, which determine the data size:
\begin{itemize}
\item [-b] indicates the data is in bytes.
\item [-w] indicates the data is in words.
\item [-l] indicates the data is in long words.
\end{itemize}
\item[Number bases] All arguments default to hexadecimal. Specify other bases by typing a colon (:) and the base after the value.
\end{description}

For example, type 52:10 for decimal 52.

\begin{center}
\texttt{displaymem\ startaddr\ lines}
\end{center}
displays \textit{lines} of memory starting at \textit{startaddr}. If the \textit{lines} argument is not specified, 16 lines are displayed. After you type this command, pressing \textless cr\textgreater displays the next block of memory. Access size is bytes.

\begin{center}
\texttt{readmem\ -[b,w,l]\ address}
\end{center}
reads a memory location specified by \textit{address}. This command displays the data in hexadecimal, decimal, octal, binary, or string format.

\begin{center}
\texttt{setmem\ -[b,w,l]\ address}
\end{center}
allows memory locations to be modified starting at \textit{address}. \texttt{setmem} first displays the value that was read. Then you can type new data for the value. If you press \textless cr\textgreater after the data, the address counts up. If you press \textless ESC\textgreater after the data, the address counts down.

\begin{center}
\texttt{writemem\ -[b,w,l]\ address\ value}
\end{center}
writes \textit{value} to a memory location specified by \textit{address}.

\begin{center}
Revision A (Preliminary) / June 1991
\end{center}
CONFIRGURING 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.

3. Press E to edit the group.

4. Press <cr> until the "BootDev" field is displayed.

5. Type the new value "ROM".

Revision A (Preliminary) / June 1991
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 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.

Revision A (Preliminary) / June 1991
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 DBO 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-In.tel records. In practice, records are usually separated by a convenient number of carriage returns, line feeds, 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-In.tel Format**

Hex-In.tel 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:

\[
\text{address} = (\text{segment} \ll 4) + \text{offset};
\]

or

\[
\begin{align*}
\text{segment} & \quad \text{ssss} \\
+ & \quad \\
\text{offset} & \quad \text{0000} \\
\text{address} & \quad \text{aaaaa}
\end{align*}
\]

For addresses with fewer than 16 bits, the segment portion of the address is unnecessary. The hex-In.tel 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-In.tel format — extended address record, data record, optional start address record, and end-of-file record. A file composed of hex-In.tel records must end with a single end-of-file record.

**Extended Address Record**

\[
:02000002\text{ssss}\text{cs}
\]

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>is the record start character.</td>
</tr>
<tr>
<td>02</td>
<td>is the record length.</td>
</tr>
<tr>
<td>0000</td>
<td>is the load address field, always 0000.</td>
</tr>
<tr>
<td>02</td>
<td>is the record type.</td>
</tr>
<tr>
<td>sssss</td>
<td>is the segment address field.</td>
</tr>
<tr>
<td>cs</td>
<td>is the checksum.</td>
</tr>
</tbody>
</table>
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<sub>16</sub>. This means that all subsequent data record addresses should have 200<sub>16</sub> added to their addresses to determine the absolute load address.

---

**Data Record**

:11aaaa00d1d2d3...dncs

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>11</td>
<td>aaaa</td>
<td>00</td>
<td>d1...dn</td>
</tr>
</tbody>
</table>

: 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<sub>16</sub>; if any segment value was previously specified, it is added to the address. 50<sub>16</sub> is loaded to address 10<sub>16</sub>, D5<sub>16</sub> to address 11<sub>16</sub>, 5A<sub>16</sub> to address 12<sub>16</sub>, and DF<sub>16</sub> to address 13<sub>16</sub>. The checksum is 8E<sub>16</sub>.

---

Revision A (Preliminary) / June 1991
Start Address Record

:04000003ssssooooocs

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.
oooo0 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 loads:
byte 20\textsubscript{16} to address 00\textsubscript{16}
byte 82\textsubscript{16} to address 01\textsubscript{16}
byte E4\textsubscript{16} to address 02\textsubscript{16}
byte 46\textsubscript{16} to address 03\textsubscript{16}
byte A8\textsubscript{16} to address 04\textsubscript{16}
byte 0A\textsubscript{16} to address 05\textsubscript{16}
byte 6C\textsubscript{16} to address 06\textsubscript{16}
byte CE\textsubscript{16} to address 07\textsubscript{16}

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

:0800000000EDE0A2744617EFFE7 loads:
byte DO\textsubscript{16} to address 10\textsubscript{16}
byte ED\textsubscript{16} to address 11\textsubscript{16}
byte 0A\textsubscript{16} to address 12\textsubscript{16}
byte 27\textsubscript{16} to address 13\textsubscript{16}
byte 44\textsubscript{16} to address 14\textsubscript{16}
byte 61\textsubscript{16} to address 15\textsubscript{16}
byte 7E\textsubscript{16} to address 16\textsubscript{16}
byte FF\textsubscript{16} to address 17\textsubscript{16}

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

:04003000902BB4FD5F loads:
byte 90\textsubscript{16} to address 40\textsubscript{16}
byte 2B\textsubscript{16} to address 41\textsubscript{16}
byte B4\textsubscript{16} to address 42\textsubscript{16}
byte FD\textsubscript{16} to address 43\textsubscript{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)

S0nnnd1d2d3...dncs

S0 indicates the record type.

nn indicates 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 ASCII representation of "v30bugs." The checksum is 6D16.

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

S1nnaaaad1d2d3...dncs

S2nnaaaaaad1d2d3...dncs

S3nnaaaaaaad1d2d3...dncs

S1 indicates the record type.

nn indicates 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, 30, FF, DC, and 95 are loaded into memory starting at address 01A0.

S30B30000000FFFF5555AAAAAD3
In this example, the bytes FF, FF, 55, 55, AA, and AA are loaded into memory starting at address 3000. 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)**

\[ S5nnn\_dl\_dn...dncs \]

S5 indicates the record type.

nn is the count of data and checksum bytes.

dl 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, and the count of the S1-records, S2-records, and S3-records in the file is 343.

**S7-, S8-, and S9-records**

*(Termination and Start Address Records)*

\[ S705nnaaacs \]
\[ S804nnaaaaaacs \]
\[ S903nnaaaaaacacss \]

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
S10F000000010000000000084EFAFFFE93
S5030001FB
S9030008F4

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

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

S10F000000010000000000084EFAFFFE93 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 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:

```bash
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

```bash
download -b
```

downloads a binary file.

Executing a Downloaded Program — call

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

```bash
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

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:

```c
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:

```c
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`

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 byteword

clears byteword bytes starting at address source.

cmpmem

cmpmem source destination byteword

compares byteword 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 byteword

copies byteword 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 -1 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

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:

<table>
<thead>
<tr>
<th>Group</th>
<th>Fields</th>
<th>Purpose</th>
<th>Heurikon Default</th>
<th>Optional Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Console and Download</td>
<td>Port</td>
<td>Selects communications port.</td>
<td>A (Download)</td>
<td>(A, B, C, D)</td>
</tr>
<tr>
<td></td>
<td>Baud</td>
<td>Selects baud rate.</td>
<td>B (Console)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parity</td>
<td>Selects parity type.</td>
<td>None</td>
<td>(Even, Odd, None, Force)</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>Selects the number of data bits for transfer.</td>
<td>8-Bits</td>
<td>(5-Bits, 6-Bits, 7-Bits, 8-Bits)</td>
</tr>
<tr>
<td></td>
<td>StopBits</td>
<td>Selects the number of stop bits for transfer.</td>
<td>2-Bits</td>
<td>(1-Bit, 2-Bits)</td>
</tr>
<tr>
<td></td>
<td>XOnXOff</td>
<td>Selects XOnXOff protocol.</td>
<td>On</td>
<td>(Off, On)</td>
</tr>
<tr>
<td></td>
<td>ChBaudOnBreak</td>
<td>Break character causes baud rate change.</td>
<td>False</td>
<td>(False, True)</td>
</tr>
<tr>
<td></td>
<td>RstOnBreak</td>
<td>Break character causes reset.</td>
<td>False</td>
<td>(False, True)</td>
</tr>
<tr>
<td>VmeBus</td>
<td>ExtSlaveMap</td>
<td>Address to map slave extended space.</td>
<td>0x80000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>StdSlaveMap</td>
<td>Address to map slave standard space.</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>AddrModSel</td>
<td>Address mode selection</td>
<td>ExAll</td>
<td>(None, StDat, StAll, ExSuDat, ExDat, ExAll)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Notes: The abbreviations stand for: no slave access allowed (disable), standard data, all standard, extended supervisor data, extended data, all extended (see section 7.4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplaceAddr</td>
<td>Standard space replacement address</td>
<td>0x00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableSlave</td>
<td>Enable/disable slave standard space</td>
<td>True</td>
<td>(False, True)</td>
<td></td>
</tr>
<tr>
<td>MastRelModes</td>
<td>Select master release modes.</td>
<td>OnRequest</td>
<td>(WhenDone, OnRequest, OnClear, Never)</td>
<td></td>
</tr>
<tr>
<td>SlaveRelOnReq</td>
<td>Enable/disable slave release-on-request</td>
<td>On</td>
<td>(Off, On)</td>
<td></td>
</tr>
<tr>
<td>LocalBusTimer</td>
<td>Select duration of on-card bus timer</td>
<td>32μ</td>
<td>(4μs, 16μs, 32μs, 64μs, 128μs, 256μs, 512μs, Off)</td>
<td></td>
</tr>
<tr>
<td>VmeBusTimer</td>
<td>Select duration of VMEbus timer</td>
<td>64μ</td>
<td>(4μs, 16μs, 32μs, 64μs, 128μs, 256μs, 512μs, Off)</td>
<td></td>
</tr>
<tr>
<td>Sysfail</td>
<td>Turn SYSFAIL* on or off.</td>
<td>Off</td>
<td>(Off, On)</td>
<td></td>
</tr>
<tr>
<td>IndivRMC</td>
<td>Turn indivisible read-modify-write on or off.</td>
<td>Off</td>
<td>(Off, On)</td>
<td></td>
</tr>
<tr>
<td>Mailbox</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShtSlaveMap</td>
<td>Address to map slave short space</td>
<td>0xffff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableSht</td>
<td>Enable/disable short space.</td>
<td>False</td>
<td>(False, True)</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InstrCache</td>
<td>Turn instruction cache on or off.</td>
<td>On</td>
<td>(Off, On)</td>
<td></td>
</tr>
<tr>
<td>DataCache</td>
<td>Turn data cache on or off.</td>
<td>Off</td>
<td>(Off, On)</td>
<td></td>
</tr>
<tr>
<td>Misc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerUpMemClr</td>
<td>Clear memory on power-up.</td>
<td>True</td>
<td>(False, True)</td>
<td></td>
</tr>
<tr>
<td>ClrMemOnReset</td>
<td>Clear memory on reset.</td>
<td>True</td>
<td>(False, True)</td>
<td></td>
</tr>
<tr>
<td>PowerUpDiag</td>
<td>Use power-up diagnostics.</td>
<td>On</td>
<td>(Off, On)</td>
<td></td>
</tr>
<tr>
<td>CountValue</td>
<td>Choose shortest (0) to longest (7) duration for autoboot countdown</td>
<td>7</td>
<td>(0, 1, 2, 3, 4, 5, 6, 7)</td>
<td></td>
</tr>
<tr>
<td>BootParams</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BootDev</td>
<td>Select boot device.</td>
<td>None</td>
<td>(None, Disk, Floppy, Tape, Serial, Ethernet, ROM, Bus)</td>
<td></td>
</tr>
<tr>
<td>LoadAddress</td>
<td>Define load address.</td>
<td>0x03010000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RomBase</td>
<td>Define ROM base.</td>
<td>0x00040000</td>
<td>This field is used only when BootDev is defined as ROM.</td>
<td></td>
</tr>
<tr>
<td>RomSize</td>
<td>Define ROM size.</td>
<td>0x00020000</td>
<td>This field is used only when BootDev is defined as ROM.</td>
<td></td>
</tr>
</tbody>
</table>
### 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

```plaintext
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

```plaintext
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

```plaintext
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

```plaintext
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

```plaintext
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:

<table>
<thead>
<tr>
<th>MPU Trace Configuration Display:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleStep</td>
</tr>
<tr>
<td>Branch</td>
</tr>
<tr>
<td>Call</td>
</tr>
<tr>
<td>Return</td>
</tr>
<tr>
<td>Prereturn</td>
</tr>
<tr>
<td>Breakpoints</td>
</tr>
<tr>
<td>BreakPoint1</td>
</tr>
<tr>
<td>BreakPoint2</td>
</tr>
<tr>
<td>BreakPoint3</td>
</tr>
<tr>
<td>BreakPoint4</td>
</tr>
<tr>
<td>BreakPoint5</td>
</tr>
<tr>
<td>BreakPoint6</td>
</tr>
<tr>
<td>BreakPoint7</td>
</tr>
<tr>
<td>BreakPoint8</td>
</tr>
</tbody>
</table>

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.
The HK68/V3D Monitor

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 8216.

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 (" ").
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
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.

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.
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 HK681V3D.

NV-RAM COMMANDS

- nvdlsplay 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

- checksummem source bytcount reads bytcount bytes starting at address source, indicates the checksum for that region of memory.

- clearmem source bytcount clears bytcount bytes starting at address source.

- cmpmem source destination bytcount compares bytcount bytes at source with those at destination. Any differences are displayed.

- copymem [-b,w,l] source destination bytcount copies bytcount bytes from source to destination.

- displaymem startaddr lines displays lines of memory starting at startaddr. Lines defaults to 16.

- 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 bytcount swaps bytcount 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.
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 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 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.

UTILITIES

disassemble startaddr lines

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 stop timer 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.
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 HK68V3D.

NV-RAM COMMANDS

nvdlsplay
  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.

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)
  xxFFxxxx (standard space)
  xxxxFFxx (short space)

slaved [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.

<table>
<thead>
<tr>
<th>Message</th>
<th>Source and suggested solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error while clearing NV memory.</td>
<td>NV memory has become corrupted. Type nvinit to restore defaults. If the problem persists, call a Heurikon customer representative.</td>
</tr>
<tr>
<td>Error while reading NV memory.</td>
<td></td>
</tr>
<tr>
<td>Error while storing NV memory.</td>
<td></td>
</tr>
<tr>
<td>Hit 'H' to skip bus auto-boot</td>
<td>Consult the introduction to this appendix for information about power-up conditions.</td>
</tr>
<tr>
<td>No help for ___.</td>
<td>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.</td>
</tr>
<tr>
<td>Power-up Memory Test FAILED.</td>
<td>A failed Memory Test or Serial Test could mean a hardware malfunction. Report the error to Heurikon Customer Support.</td>
</tr>
<tr>
<td>Power-up Serial Test FAILED.</td>
<td></td>
</tr>
<tr>
<td>Unable to change ID.</td>
<td>The Module ID can be changed only by Heurikon.</td>
</tr>
<tr>
<td>Unknown ___</td>
<td>The Module ID is incorrect. Report the error to Heurikon Customer Support.</td>
</tr>
<tr>
<td>Unknown boot device</td>
<td>The boot device is invalid. Use nvdisplay to check and edit the &quot;BootParams&quot; group, &quot;BootDev&quot; field. Save a new value with nvupdate.</td>
</tr>
<tr>
<td>Unexpected ____ Exception at ____ .</td>
<td>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.</td>
</tr>
<tr>
<td>Warning NV memory board initialization skipped.</td>
<td>Only minimum configuration has been completed. The configuration data structures are invalid.</td>
</tr>
<tr>
<td>Warning NV memory is invalid - using defaults.</td>
<td>Consult the introduction to this appendix for information about reset conditions.</td>
</tr>
<tr>
<td>Warning</td>
<td>Action</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Warning protected region of NV memory cannot be initialized.</td>
<td>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 <code>nvopen</code> command, or return any fields you tried to edit to their original values.</td>
</tr>
<tr>
<td>Warning protected region of NV memory was not modified.</td>
<td></td>
</tr>
<tr>
<td>Warning protected region of NV memory is corrupt.</td>
<td></td>
</tr>
<tr>
<td>Warning: Real Time clock is invalid.</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

MaskInts()
UnMaskInts()
VecToVecAddr(Vector)
VectInit()
ConnectHandler(Vector)
DisConnectHandler(Vector)

Register and cache functions:

FlushCache()

Configuring the board by using NV memory parameters:

ConfigVmeBus()
ConfigVsbBus()
ConfigCio()
ConfigScsi()
ConfigSerDevs()
Initializing the board to the default conditions:

InitBoardO
InitCioO
InitScsiO

Controlling the serial ports:

PutC(Char)
RPutC(Char)
GetCO
RGetCO
KBHitO
RKBHitO
TxMTO
RTxMTO
RChBaud(BaudRate)
ChBaud(BaudRate)

Writing to CIO data ports:

ReadCIOPortAO
ReadCIOPortBO
ReadCIOPortCO
WriteCIOPortBO

Unmasking VMEbus interrupts:

UnMaskVMEIntO

Writing to the LED display:

SetLedDisplayO

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:

SetNvDefaultsO

Booting a program from the specified drive:

BootUpO
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)
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page Title</th>
<th>Section Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add()</td>
<td>Add</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>atob()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>atod()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>atoh()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>atoo()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>atox()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>BinToHex()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>BootBus()</td>
<td>BootBus</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>BootRom()</td>
<td>BootRom</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>BootSerial()</td>
<td>BootSerial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>BootUp()</td>
<td>BootUp</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Call()</td>
<td>Call</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Calloc()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>CFree()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>ChBaud()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>CheckSumMem()</td>
<td>BlockMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>ClearMem()</td>
<td>BlockMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>CmpMem()</td>
<td>BlockMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>CmpStr()</td>
<td>Strings</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>ConnectHandler()</td>
<td>Exceptions</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>CopyMem()</td>
<td>BlockMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Date()</td>
<td>Date</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>DisAssemble()</td>
<td>DisAssemble</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>DisConnectHandler()</td>
<td>Exceptions</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>DisDataCache()</td>
<td>Cache</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>DisInstCache()</td>
<td>Cache</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>DispGroup()</td>
<td>NVSupport</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>DisplayMem()</td>
<td>DisplayMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Div()</td>
<td>Add</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>DownLoad()</td>
<td>DownLoad</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>DumpRegs()</td>
<td>DumpRegs</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>EnbDataCache()</td>
<td>Cache</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>EnbInstCache()</td>
<td>Cache</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>ExecTrace()</td>
<td>Trace</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>FastFillMem()</td>
<td>FastFillMem</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>FillMem()</td>
<td>BlockMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>FindBitSet()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>FindMem()</td>
<td>FindMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>FindNotMem()</td>
<td>FindMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>FindStr()</td>
<td>FindMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>FlushCache()</td>
<td>Cache</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>Free()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>FromFifo()</td>
<td>InitFifo</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>GetC()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Help()</td>
<td>Help</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>HexToBin()</td>
<td>atoh</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>InitFifo()</td>
<td>InitFifo</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>InitTrace()</td>
<td>Trace</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>IsLegal()</td>
<td>IsLegal</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>KBHit()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Malloc()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Function</td>
<td>Category</td>
<td>Module</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>MaskInts()</td>
<td>Interrupts</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>MemAdd()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>MemReset()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>MemStats()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Mul()</td>
<td>Add</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>NVDisplay()</td>
<td>NVMemory</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>NVInit()</td>
<td>NVMemory</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>NVOp()</td>
<td>NVSupport</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>NVOpen()</td>
<td>NVMemory</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>NVSet()</td>
<td>NVMemory</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>NVUpdate()</td>
<td>NVMemory</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Probe()</td>
<td>Exceptions</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>PutC()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Rand()</td>
<td>Add</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>RChBaud()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>ReAlloc()</td>
<td>MemMng</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>RGetC()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>RKBHit()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>RPutC()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>RTxMT()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Seed()</td>
<td>Add</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>SetDate()</td>
<td>Date</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>SetMem()</td>
<td>DisplayMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>SetNVdefaults()</td>
<td>NVSupport</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>SetTrace()</td>
<td>Trace</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>Step()</td>
<td>Trace</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>StrCat()</td>
<td>Strings</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>StrCmp()</td>
<td>Strings</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>StrCpy()</td>
<td>Strings</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>StrLen()</td>
<td>Strings</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>Sub()</td>
<td>Add</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>SwapMem()</td>
<td>BlockMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>TestMem()</td>
<td>TestMem</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>ToFifo()</td>
<td>InitFifo</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>TransMode()</td>
<td>TransMode</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>TxMT()</td>
<td>Serial</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>UnMaskInts()</td>
<td>Interrupts</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>VectInit()</td>
<td>Exceptions</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>VecToVecAddr()</td>
<td>Exceptions</td>
<td>Processor (MC68030)</td>
</tr>
<tr>
<td>xprintf()</td>
<td>xprintf</td>
<td>Monitor (Std)</td>
</tr>
<tr>
<td>xsprintf()</td>
<td>xprintf</td>
<td>Monitor (Std)</td>
</tr>
</tbody>
</table>
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 Ox496d4f6b (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 Ox596f4f6b (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 Ox42796521 (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:

<table>
<thead>
<tr>
<th>DEVICE NUMBER</th>
<th>DOWNLOAD FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_MCS86</td>
<td>Intel MCS-86 Hexadecimal Format</td>
</tr>
<tr>
<td>MOT_EXORMAT</td>
<td>Motorola Exormax Format (S0-S3,S7-S9 Records).</td>
</tr>
<tr>
<td>HK_BINARY</td>
<td>Heurikon Binary Format</td>
</tr>
</tbody>
</table>

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()
EnblInstCache()
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 EnblInstCache 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

`DownLoadO`, `TransModeO`. 
SYNOPSIS

Date()

SetDate(DayOfWeek, Month, DayOfMonth, Year, Hour, Min, 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.

```c
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

RtcAcc().
DisplayMem(1) Monitor (Std) DisplayMem(1)

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 equivalent 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 language equivalent 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 lose synchronization with an assembly program which can result in an error in the display. This usually 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.

```c
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 VectToVecAddr 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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Legal Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL</td>
<td>0x8</td>
<td>0 - 9</td>
</tr>
<tr>
<td>HEX</td>
<td>0x4</td>
<td>0 - 9, A - F, a - f</td>
</tr>
<tr>
<td>UPPER</td>
<td>0x2</td>
<td>A - Z, 0 - 9</td>
</tr>
<tr>
<td>LOWER</td>
<td>0x1</td>
<td>a - z</td>
</tr>
<tr>
<td>ALPHA</td>
<td>0x3</td>
<td>A - Z, a - z, 0 - 9</td>
</tr>
</tbody>
</table>

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 \textit{MemAddr} of size specified by \textit{MemSize}. This function currently allows for only one contiguous memory pool and must be preceded by a function call to \textit{MemReset} whenever called.

The function \textit{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.

\textbf{SEE ALSO}

\texttt{MemTop()}, \texttt{MemBase()}.  

June 21, 1991
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV_OP_FIX</td>
<td>0</td>
<td>Fix NV section checksum</td>
</tr>
<tr>
<td>NV_OP_CLEAR</td>
<td>1</td>
<td>Clear NV section</td>
</tr>
<tr>
<td>NV_OP_CK</td>
<td>2</td>
<td>Check if NV section is valid</td>
</tr>
<tr>
<td>NV_OP_OPEN</td>
<td>3</td>
<td>Open NV Section</td>
</tr>
<tr>
<td>NV_OP_SAVE</td>
<td>4</td>
<td>Save NV Section</td>
</tr>
<tr>
<td>NV_OP_CMP</td>
<td>5</td>
<td>Compare NV Section data</td>
</tr>
</tbody>
</table>

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.

```c
NVOp(NV_OP_CLEAR, &NVExt, sizeof(NVExt), 0);
NVOp(NV_OP_SAVE, &NVExt, sizeof(NVExt), 0);
NVOp(NV_OP_OPEN, &NVExt, sizeof(NVExt), 0);
NVOp(NV_OP_FIX, &NVExt, sizeof(NVExt), 0);
NVOp(NV_OP_SAVE, &NVExt, sizeof(NVExt), 0);
```

The clear, save, and open operations cause the NV device to be cleared and filled with the NVExt 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.

<table>
<thead>
<tr>
<th>Error number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVE_NONE</td>
<td>No errors.</td>
</tr>
<tr>
<td>NVE_OVERFLOW</td>
<td>NV device write count exceeded.</td>
</tr>
<tr>
<td>NVE_MAGIC</td>
<td>Bad magic number read from NV device.</td>
</tr>
<tr>
<td>NVE_CKSUM</td>
<td>Bad checksum read from NV device.</td>
</tr>
<tr>
<td>NVE_STORE</td>
<td>Write to NV device failed.</td>
</tr>
<tr>
<td>NVE_CMD</td>
<td>Unknown operation requested.</td>
</tr>
<tr>
<td>NVE_CMP</td>
<td>Data does not compare to NV device.</td>
</tr>
</tbody>
</table>
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 occurred. 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 fSetTrace/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 SetNVDefault 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

```c
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 defined HK_FIELDS;
extern NV MonDefs NvMonDefs;
char BoardModel[1] = "V3D";

DOCSEC: ConfigBoard 1 V3D Board

SYNOPSIS: InitBoard()

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

DOCSEC: Misc 1 V3D Board

SYNOPSIS: PrStatus()
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 monitor's 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.

```c
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;
}
```

```c
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 monitor's 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.

    ************************************************************************
    ** DOCSEC: IntErr 1 V3F Board
    ** SYNOPSIS: IntErr(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.

static char ExcErrStr[] = "Unexpected % % % Exception at 0x%.8x (Acc at 0x%.8x)\n";
static char DevIntStr[] = "Unexpected % % % Interrupt at 0x%.8x\n";
static char UnkIntStr[] = "Unexpected Interrupt at 0x%.8x (%x) Vector 0x%.8x\n"

IntrErr(AccAddr, Addr, Vector)
register long AccAddr,
register char *Addr;
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);
}
Board.h  Page 1
/****************************

* 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:
* 2B5C36 CIO Counter Timer.
* 2B5C30 SCC Serial Controller, Ports A-B
* WD33C93 SCSI Controller.
* DM1216F Realtime clock.
* NMI Status Latch.
* 82518CA Ethernet Controller.
* 82C54 EEPROM
**/**
#define MON_REV_LEVEL "1.1" /* define monitor revision level */
/****************************

* Interrupt Vector assignments for v3d (68030).
**/**
#define BUS_ERROR 0x02
#define ADDR_ERROR 0x03
#define ILLEGAL_INSTR 0x04
#define ZERO_DIVIDE 0x05
#define PRIV_VIOLATION 0x06
#define TRACF_FAULT 0x09
#define EMULATOR10 0x0A
#define EMULATOR11 0x0B
#define SPURIOUS_INTR 0x18
#define V86 VECTOR 0x19
#define SCSI VECTOR 0x1C
#define PARITY_ERROR 0x1F

Board.h  Page 2
# define C10_VECTOR 0x90
# define SCC_ABVECTOR 0xA0
# define SCC_CDVECTOR 0x80

/****************************

* DRAM
**/**
#define RAM_BASE 0x03000000
/****************************

* C10: Definitions for the 2B5C36 CIO Counter Timer and parallel ports
**/**
#define C10PORT 0x02D00001
#define C10_ADDR ((volatile unsigned char *) (C10PORT + 0x04))
#define C10_BDATA ((volatile unsigned char *) (C10PORT + 0x02))
#define C10_CDATA ((volatile unsigned char *) (C10PORT + 0x00))
#define C10_CTRL ((volatile unsigned char *) (C10PORT + 0x06))

/****************************

* SCC: Definition for the 2B5C30 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 {
  unsigned char Control;
  unsigned char Dummy[SCC_REG_SPREAD];
  unsigned char Data;
}; /* Define port addresses */
#define SCC_PORTA (struct SCCPort *) 0x02000000
#define SCC_PORTB (struct SCCPort *) 0x02000001
#define SCC_PORTC (struct SCCPort *) 0x02000002
#define SCC_POR TD (struct SCCPort *) 0x02000003
#define SCC_POR TC (struct SCCPort *) 0x02000004
#define SCC_PORTD (struct SCCPort *) 0x02000005

/****************************

* SCSI: Definition for the WD33C93 SCSI interface.
**/**
#define SCSI_ADDR 0x02300001 /* Base Address of SCSI chip */
#define SCSI_ENABLE (unsigned char *) 0x02300020
#define SC8_RESET (unsigned char *) 0x02300056 /* Bus reset */

struct SC8Chip {
  unsigned char SC_AddrPtr;
  unsigned char SC_Dummy[1];
  unsigned char SC_Register;
}; /* Define macros to read and write */
#define SC8 (struct SC8Chip *) 0x02300000
#define SCWriteReg(Reg, Val) SC8->SC_AddrPtr = Reg 
SC8->SC_Register = Val
#define SCreadReg(Reg, Val) SC8->SC_AddrPtr = Reg 
Val = SC8->SC_Register
/**
 * SCSI bus interface controller registers
 */
define SREG OWNID Ox00
define SREG_CTRL Ox01
define SREG_TIMEOUT Ox02
define SREG_TSECT Ox03
define SREG_THREAD Ox04
define SREG_TCYLL Ox05
define SREG_TCYLL Ox06
define SREG_RN_LADR Ox07
define SREG_RN_LADR Ox08
define SREG_LM_LADR Ox09
define SREG_TM_LADR Ox0A
#define SREG_SECT Ox0B
define SREG_READ Ox0C
define SREG_CYLL Ox0D
define SREG_CYLL Ox0E
define SREG_TLUN Ox0F
#define SREG_9PBASE Ox10
#define SREG_SINT Ox11
define SREG_MTCN Ox12
define SREG_MCNT Ox13
define SREG_DEST_ID Ox14
define SREG_SRC ID Ox15
define SREG_SEC_STK Ox16
define SREG_CMD Ox17
define SREG_DATA Ox18
#define SREG_DB1 Ox19
define SREG_DB2 Ox1A
define SREG_DB3 Ox1B
define SREG_DB4 Ox1C
define SREG_DB5 Ox1D
define SREG_DB6 Ox1E
define SREG_DB7 Ox1F
define SREG_DB8 Ox20
define SREG_DB9 Ox21
#define SREG_DB10 Ox22
define SREG_DB11 Ox23
define SREG_DB12 Ox24
#define SCOMA_ADDRESS Ox240000 /* DMA Acknowledge address */
/**
 * RTC: Data structures and addresses for the real time clock
 */
define WATCHBASE (volatile unsigned char *) Ox00000000
#define MRD WATCH (volatile unsigned char *) (WATCHBASE + 2)
define WRL WATCH (volatile unsigned char *) (WATCHBASE + 3)
define RD WATCH (volatile unsigned char *) (WATCHBASE + 4)
struct rtc data { /* */
  unsigned char dotsec; /* -- 0.01 sec --- */
  unsigned char sec; /* -- 0.1 sec ----- */
  unsigned char min; /* -- 10 min ------- */
  unsigned char hour; /* -- A 0 B Hr ------ */
  unsigned char weekday; /* -- 0 0 0 1 ------ */
  unsigned char date; /* -- 10 date ------- */
  unsigned char month; /* -- 10 Month ---- */
};
/**
 * VME: Must Write this
 */
define MBOX BASE (unsigned short *) Ox02000000
#define ENBL_OOG (unsigned char *) Ox02000001
#define VME_TIMER (unsigned char *) Ox02000002
#define SYSFAIL (unsigned char *) Ox02000003
#define ENBL_MBOX (unsigned char *) Ox02000004
#define BUS LATCH (unsigned long *) Ox02000005
#define SLAVE_ENABLE (unsigned char *) Ox02000006
/**
 * X212 NVRAM: Definition for the NV Memory Interface
 */
define NV_BASE Ox02000000 /* Base address of NV memory */
define NV_SIZE Ox00000000 /* Size in bytes of NV memory */
define NV_PROTECTED Ox00000000 /* Beginning of protected NV memory */
define NV_MON_DEFS Ox00000000 /* Beginning of monitor NV defs. */
define NV_MAX_BB_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 *) Ox02060000
#define NV_RECALL (unsigned char *) Ox02070000
This file contains much of the 6B030-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"

---

ColdStart: mov.l $0x0000, %d0
mov.l %d0, %lo
mov.l %lo, 0x10(%a0)
mov.l %d0, 0x20(%a0)

start_ip: mov.l $0x0000, %d0
mov.l %d0, %lo
mov.l %lo, 0x10(%a0)
mov.l %d0, 0x20(%a0)

MonEntryPt: mov.l $0x07000000, %a7 # Clear LED's
movm.l $0x00000000, %d0
movm.l %d0, %lo
movm.l %lo, 0x10(%a0)

Pause 500 mSec for RAM and then do 8 RAS/CAS cycles to initialize memory.

MonEntryPt:
ColdStart: mov.l $0x0000, %d0
mov.l %d0, %lo
mov.l %lo, 0x10(%a0)
mov.l %d0, 0x20(%a0)

IsPowerUp:

IsPowerUp:

SetState:
mov.l %sup, %a7 # New supervisory stack
mov.l %sup, %a7
mov.l %sup, %a7
mov.l %sup, %a7

StartMon:
jar VectorInit # Initialize Vector Table.
mov.l %lo, %lo
mov.l %lo, %lo
mov.l %lo, %lo
mov.l %lo, %lo

 STACK DEFINITIONS: The following data definitions define the stacks for the 6B030. 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 6B030 manual. The initialization of these structures is performed by other functions.
even

global int_table
global sup_stack
global PwrUpLoc
lcomm int_table, Ox0400
lcomm top_stack, Ox4000
lcomm sup_stack, Ox40
lcomm PwrUpLoc, Ox04
lcomm Reserved, Ox0C
/* ******************************************************************************
   * 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 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 characters definitions */
#define EOF 0
#define DEL 0x7F
#define ESC 0x1B
#define SP ' '
#define BS '\b'
#define CR '\r'
#define LF '\n'
#define TAB '\t'
/* Define the character definitions */
#define MAXARGS 20
struct args {
    char argv[MAXARGS];
};
/* Define the UNIX style time structure */
struct tm {
    unsigned long tm_msec; /* 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;
# Source Code: CIO.c

**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 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

- **InitCio():** This function resets the counter timer regardless of 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():** This function initializes the counter timer to the state expected by the monitor. The configuration sets

```c
/**
 * InitCio()
 * volatile unsigned char *p, *c;
 * p = CIO_CTRL;
 * c = "p";
 */
```
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.

```c
WriteCioPortB(Data)
register unsigned char Data;
{ return (*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); }
```

The `StartTimer()` 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.

```c
volatile int NumTicks;
StartTimer()
{ register int cnt;
  register int CioIntr();
  static unsigned char ctitable[] = {
  0x00, 0x02,
  0x16, 0x80,
  0x19, 0x02, 0x1B, 0x35, /* Channel 3 Count (1/60th sec) */
  0x0C, 0x20, /* Clear IP and IUS for channel 3 */
  0x1D, 0x00,
  0x18, 0x20, /* Channel 2 Count (1/97th sec) */
  0x08, 0x08, /* Channel 1 Count (1/157th sec) */
  0x00, 0x00,
  0x04, 0x04, /* Enable master interrupt VIS */
  0x00, 0x00, /* Enable counters 1, 2, and 3 */
  0x01, 0x00,
  0x0C, 0x06, /* Enable interrupts, start count */
  0x0B, 0x06, /* Set up port 3 */
  0x0A, 0x06; /* Enabl`e Interrupts, start count */
};

sprint("NumTicks loaded at 0x%x", &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];
UnMaskIntrs();
}
```

The `CioIntr()` function is the interrupt handler for the counter timer. This function removes the interrupt in the device and then clears the interrupt in the processor.

```c
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;
    Status = *CIO_CTRL;
    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"

DOCSRC:
Exceptions 1 MC68030 Processor

SYNOPSIS:
 VectInit()
 unsigned long *VectToVecAddr(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 VectToVecAddr 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
handler, 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 wrapper and then deallocates 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 *VectToVecAddr(Vector)
unsigned long Vector;
{ return((unsigned long *)&int_table + Vector)); }

VectInit()
{ int i, UnExpIntr();
 unsigned long *VecPtr;
 VecPtr = int_table;
 for(i = 0; i < 256; i++) { 
 *VecPtr++ = (unsigned long) UnExpIntr;
 }

struct IntWrapper IntCode = {
 0x4be7ffff, 0x302f0046, 0x2f014eb9, 0x00000090, 0x00000000, 0x60000010, 0x2f7c0000,
 0x00000000, 0x2f7c0000, 0x00000000, 0x00000000, 0x0046e48b, 0x02800000, 0x0046e48b, 0x00422f00,
 0x00403780};
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 & OxDF) {
        case 'R':
            if (!sav_env()) {
                *(unsigned char *) Address = *(unsigned char *) Data;
                BusError = TRUE;
            } else {
                BusError = TRUE;
            }
            break;
        case 'W':
            if (!sav_env()) {
                *(unsigned short *) Address = *(unsigned short *) Data;
                BusError = TRUE;
            } else {
                BusError = TRUE;
            }
            break;
    }
}

switch (SizeFlag & OxDF) {
    case 'B':
        if (!sav_env()) {
            *(unsigned char *) Address = *(unsigned char *) Data;
            BusError = TRUE;
        } else {
            BusError = TRUE;
        }
        break;
    case 'W':
        if (!sav_env()) {
            *(unsigned short *) Address = *(unsigned short *) Data;
            BusError = TRUE;
        } else {
            BusError = TRUE;
        }
        break;
    case 'L':
        if (!sav_env()) {
            *(unsigned long *) Address = *(unsigned long *) Data;
            BusError = TRUE;
        } else {
            BusError = TRUE;
        }
        break;
    default:
        xprintf("error: argument 2 must be -b, -w or -l\n");
        break;
}
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 */
**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 respectively by turning on the enables in the CACR register.
- The functions DisInstCache and DisDataCache disable the instruction and data caches respectively by turning off the enables in the CACR register. Before a cache is disabled it is flushed.

**SEE ALSO:**

---

**DOCSEC:** 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:

DESCRIPTION:

SYNOPSIS:

global ProcRegs

the command line editor is re-entered.

The FastFillMem function provides a fast method

for filling memory with the Value specified.

The FastFillMem 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.1 40x0FFF,-(hp) # Save registers

movm.1 0x44(hp),%d0 # Get 'FillValue' off stack

movm.1 0x48(hp),%a1 # Get 'Base' off stack

movm.1 0x4C(hp),%a2 # Get 'Top' off stack

mov.1 %d0, %d1 # Copy FillValue to other

mov.1 %d0, %d2 # registers.

mov.1 %a0, %d6 # Copy Top

sub.l %d1, %d6 # Count = (Top - Base)

sub.l %d1, %d6 # Count = Count / 32;

FillLoop:

dbra %d6, FillLoop # Branch till done

CleanUp:

mov.1 %d0,-(%a0)

cmp.l %a1,%a0

bit CleanUp

movm.1 (%a0)+,40x0FF

rts

j

# sav_env (env)

# jmp_buf (env);

# res_env (env, retval)

# jmp_buf *env;

# int retval;

# Recover from anticipated bus error

****

* 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 FastFillMem 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: *****/
lcomm EnvBuffer, 0x50 # Bas Area to save environment
--- Page 1 ---

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:

```assembly
Wrapper:
  NotBusErr:
    movm.l &0xffff, -(hp)
    mov.w &0x0000, %d0
    cmp.w &d0, &0x0000
    bne.s
    mov.l &0,144
    mov.w &0x46(%sp), %d0
    lsr.l &2
    and.l &255
    mov.l %d0,-(hp)
    jsr IntHdl
    add.l &8,%sp
    movm.l &0x0000, -(hp)

  IntHdl
```

--- Page 2 ---

```c
struct IntWrapper {  
  unsigned long CodeSeg0[14];  
  unsigned long CallAddr;  
  unsigned long CodeSeg1[2];  
  unsigned long DataSeg0[3];  
};
```

-- Register File definitions for 68030: --

```c
typedef struct RegFile {
  unsigned long DataRegs[8];
  unsigned long AddrRegs[8];
  unsigned long CtrlRegs[16];
  RegFile, *RegFilePtr;
} RegFile, *RegFilePtr;
```

```c
typedef struct TraceStackFrame {
  unsigned short StatusReg;
  unsigned long ProgCtr;
  unsigned short Vector;
  unsigned long InstrAddr;
} TrStkFrame, *TrStkFramePtr;
```

--- Page 1 ---

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,-(hp)
 4: 302f 0046  mov.w 0x46(%sp), %d0
 8: 0340 f000  and.w &0x4000, %d0
 c: 0c40 b000  cmp.w &b000, &0c40
 0: 6600 0010  bne.w 0x10 <22>
 9: d7f7 0000 0000 0000
 1: 02e0 f7ff  mov.w 0x0000, 0x20
 2: 302f 0046  mov.w 0x46(%sp), %d0
 6: 4e88  mov.l %d0, %d0
 8: 2b80 0000 0000
 a: 22f2 0042  mov.l 0x42, %d0
 3: 2f00  mov.l &d0,-(hp)
 4: 2fd1  mov.l &d0,-(hp)
 6: 4eb9 0000 0000
 3: 508f  add.l &f, %sp
 4: 4cdff ffff  movm.l &0xffff,-(hp)
 4: 4e73  rte
 4: 4e71  nop
 4: 4e71  nop
 4: 4e71  nop
 4: 4e71  nop
 4: 4e71  nop
```
# RTC.c

```c
/* ***************************************************************************/
static unsigned char InitKey[];
#include "Board.h"
/* ***************************************************************************/

static rtc acc(data, Type)
int Type;
#include
register
1;
/* All Copyright (c) Heurikon Corporation */
accuracy, reliability, currentness, or otherwise; and you rely appears in all copies and that both the copyright notice and this permission appear in the 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[] = {
  0x05, 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 1, bit;
  volatile unsigned char temp;
  temp = *RD_WATCH;
  for(i = 0; i < 8; i++){
    for(bit = 1; bit & 0xFF; bit <<= 1){
      temp = (data[i] & bit) ? *WR1_WATCH : *WRO_WATCH;
    }
  }
  if (Type)
    for(i = 0; i < 8; i++){
      for(bit = 1; bit & 0xFF; bit <<= 1){
        temp = (data[i] & bit) ? *WR1_WATCH : *WRO_WATCH;
      }
    }
    /* Copy function to memory. */
    if (Type)
      for(i = 0; i < 8; i++){
        for(bit = 1; bit & 0xFF; bit <<= 1){
          temp = (data[i] & bit) ? *WR1_WATCH : *WRO_WATCH;
        }
      }
      /* Copy function to memory. */
      #ifdef RAM_MON
        rtc acc(RtcData, Flag);
      /* If RAM based monitor */
      #else
        Size = (int) rtc acc - (int) InitKey;
        Funct = (int (*)(void)) malloc(sizeof(int));
        FlushCache();
        CopyMem(rtc acc, Funct, Size);
        Funct(RtcData, Flag);
        Free(Funct);
      #endif
```
if (Flag == READ) {
    Time->tm_nsec = 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;
}
/* SCC.c: This file contains the functions necessary to read, write and configure the ZSS(30-l6 Serial Controller.

The functions defined in this module are listed below:

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,
(unsigned long) SCC_PORTB,
};

/****************************************************
* SCC.c: This file contains the functions necessary to read, write and configure the ZSS(30-l6 Serial Controller.

The functions defined in this module are listed below:

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,
(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 (!)
if (Port->Control & OxOl)
Data = Port->Data;
if (Port->Control & Ox80)
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 Ox04);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;
register char c;
{ Port->Control = 0;
return(Port->Control & Ox01);
}

/****************************************************************************
* 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 & Ox04) ? 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 < Ox1000; tc++);
tc = BaudToTimeConst(Baud);
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 = Ox09;
    Port->Control = Ox00;
}

/* ConfigSerDevs(): This function uses the current definitions in the NV structure 'NvMonDefs' to configure the serial ports. The 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. This function hard resets both ports associated with 'Port' because it's too clumsy to reset individual ports. */

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[] = {
        Ox09, Ox00, /* No Reset */
        Ox0A, Ox00, /* NRI */
    }
    if (((unsigned) Port == ConDev) ||
        Conf == &NvMonDefs.Console) {
        if (((unsigned long) Port == ConDev) ||
            Conf == &NvMonDefs.DownLoad) {
            return;
        } else {
            return;
        }
    }
    // Other code for ConfigPort...
}

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) {
        if (Conf == &NvMonDefs.Console) {
            if (((unsigned long) Port == ModDev) {
                if (Conf == &NvMonDefs.DownLoad) {
                    return;
                }
            }
        }
    }
    // Other code for FoundBreak...
}
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(): Sets the state of the SCSI according to the NV
* definitions.
* *****/

#define SC_RESET 0x00 /* Issues a RESET Command to WD33C93 */
#define FREQ_SEL 0x80 /* Select Frequency for Divisor of 4 */

extern NV_MonDefs NvMonDefs; /* Monitor-defined configuration */

#define SC_RESET 0x00 /* Issues a RESET Command to WD33C93 */
#define FREQ_SEL 0x80 /* Select Frequency for Divisor of 4 */

InitScsi()
{
    register unsigned char Stat;

    register NV_MonDefPtr Conf = &NvMonDefs;

    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 */
    {
        *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"

Redux
#include "VME_defs.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.
 */

extern NVMonDefs NVMonDefs; /* NV Monitor definitions */

ConfigVmeBus() {
    register NVU.BusConfig *Conf = &NVMonDefs.VmeBus;

    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;
    }

    if (EnblSlave(Conf)) {
        *SLAVE_ENABLE = 1;
    } else {
        *SLAVE_ENABLE = 0;
    }

    BusVal = ((ExtSlaveMap(Conf) >> 24) & 0x000000FF) + ((StdSlaveMap(Conf) >> 12) & 0x000000FF) + ((ReplaceAddr(Conf) >> 8) & 0x000000FF) + ((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 1;
    for (i = 0; i < 3; i++) {
        *BUS_LATCH = (value >> 1);
    }
Appendix C

NVDRAM Information

The NVDRAM 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>
<thead>
<tr>
<th>Device Address</th>
<th>Byte Offsets</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0270,0000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>0-15FF&lt;sub&gt;16&lt;/sub&gt;</td>
<td>User-defined data area</td>
</tr>
<tr>
<td>0270,B000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>1600&lt;sub&gt;16&lt;/sub&gt;-17FF&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Monitor/board initialization</td>
</tr>
<tr>
<td>0270,C000&lt;sub&gt;16&lt;/sub&gt;</td>
<td>1800&lt;sub&gt;16&lt;/sub&gt;-1FFF&lt;sub&gt;16&lt;/sub&gt;</td>
<td>Manufacturing/service hardware information</td>
</tr>
</tbody>
</table>

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.
/* THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF HEURIKON CORPORATION. 
   * Copyright (c) 1990 Heurikon Corporation. All Rights Reserved. 
   * 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.
   */

#include "Bug.h"
#include "Board.h"
#include "NvMonDefs.h"

extern NV HkDefined HKFields;
extern NV MonDefs NvMonDefs;

nv_recall() {
    Delay(20);
    return (*NV_RECALL);
}
	nv_store() {
    Delay(100);
    return (*NV_STORE);
}

NVRMaxNbrWrites() { /* Returns limit of write count */
    return (NV_MAX_NBR_WRITES);
}
**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.

```c
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;
{ }
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/EESROM, 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 short and long bounded 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 nonvolatile sections assume that this is the first structure in the section and will not function.
typedef struct NV_HKDefined { /* HK struct = 40/444 bytes */
    NV_Internal Internal; /* Internal definitions */
    NVH_Hardware Hardware; /* Hardware definitions */
    NVH_Manufacturing Manuf; /* Manuf definitions */
} NV_HKDefined;
#endif
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_CHKSUM 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 */
#define NV_OP_FIX 0 /* On Board Non Volatile memorycmds. */
#define NV_OP_CLEAR 1 /* NVOp Command to fix checksum */
#define NV_OP_CHK 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 */
#define NV_OP_MCS_FIX 10 /* Module configuration space commands */
#define NV_OP_MCS_CLEAR 11 /* NVOp Command to fix checksum */
#define NV_OP_MCS_CHK 12 /* NVOp to checksum NV sections */
#define NV_OP_MCS_OPEN 13 /* NVOp to Open NV Section */
extern NVGroup NVGroups[]; /* NV memory groupings structure */
extern NV_HkDefined HKFields; /* Heurikon defined structure */

/* * NV Error Strings(); */

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. */
as hex, decimal, or a list of legal values. An example of
the display is shown below:

<table>
<thead>
<tr>
<th>Group 'Console'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port A</td>
</tr>
<tr>
<td>Baud 9600</td>
</tr>
<tr>
<td>Parity None</td>
</tr>
<tr>
<td>Data 8-Bits</td>
</tr>
<tr>
<td>StopBits 2-Bits</td>
</tr>
</tbody>
</table>

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;
  NVInternal *NvMon = (NV_Internal *) NvMonAddr();
  unsigned long NvMonSiz = NvMonSize();
  Err = NVOp(NV_OP_CB, NvMon, NvMonSiz);
  if (Err != NVE_NONE)
    xprintf(NVSupStrl, "reading", NVErrTable[Err]);
    return;

  xprintf("Non-Volatile Memory Configuration Display");
  xprintf("----------------------------------------");
  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("\n\n\n");
      DispGroup(NVGroups[i], TRUE);
    }
  }
}

NVUpdate()
{
  register int Err;
  NVInternal *NvMon = (NV_Internal *) NvMonAddr();
  unsigned long NvMonSiz = NvMonSize();
  unsigned long NvMonOff = NvMonOffset();
  unsigned long NvHkOff = NvHkOffset();
  if ((GroupName == NULL) || (FieldName == NULL)) {
    xprintf("Both a Group and Field must be specified\n");
    return;
  }
  if (NVGroupPtr Group, FindGroup();
    NVOp(NV_OP_SAVE, NvMon, NvMonSiz, NvMonOff);
    if (Err != NVE_NONE)
      xprintf(Err, "was not modified");
  
}

NVSet(GroupName, FieldName, Value)
char *GroupName, *FieldName,
{
  int Err;
  NVGroupPtr Group, FindGroup();
  NVOp(NV_OP_CB, NvMon, NvMonSiz);
  if (Err != NVE_NONE)
    xprintf("Both a Group and Field must be specified\n");
  return;
```c
if (Err = NVOp(NV_OP_FIX, NvMon, NvMonSize)) != NVE_NONE) {
    xprintf(NVSupStr1, "reading", NVErrTable[Err]);
    return;
}
SetField(GroupName, FieldName, Value, TRUE);
NVOp(NV_OP_WRITE, NvMon, NvMonSize);
NVOp(NV_OP_READ, NvMon, NvMonSize);
NVOp(NV_OP_SAVE, NvMon, NvMonSize);
NVOp(NV_OP_CLEAR, NvHkFields, sizeof(NV_HkDefined));
NVOp(NV_OP_OPEN, NvMon, NvMonSize);
SetNvDefaults(NVGroupPtr, NumGroups);
HKFields.Manuf.Revision = RevLev;
HKFields.Manuf.SerialNumber = SerNum;
HKFields.Manuf.ECOLevel = EColv;
HKFields.Internal.WriteCnt = Writes;
NvMon->WriteCnt = Writes;
NVOp(NV_OP_FIX, NvHkFields, sizeof(NV_HkDefined));
NVOp(NV_OP_SAVE, NvMon, NvMonSize);
if (Err != NVOp(NV_OP_WRITE, NvMon, NvMonSize)) {
    if (Err != NVE_NONE) {
        xprintf(NVSupStr2, "writing", NVErrTable[Err]);
        return;
    }
    if (Err != NVE_NONE) {
        xprintf(NVSupStr2, "cannot be initialized");
        return;
    }
}

// Supported data structures for NvMon
NVInternal "NvMon = (NV_Internal *) NvMonAddr();
unsigned long NvMonSize = NvMonSize();
unsigned long NvMonOff = NvMonOffset();
unsigned long NvMonSize = NvMonSize();
NVOp(NV_OP_CLEAR, NvHkFields, sizeof(NV_HkDefined));
NVOp(NV_OP_OPEN, NvMon, NvMonSize);
NVOp(NV_OP_SAVE, NvMon, NvMonSize, NvMonOff);
NVOp(NV_OP_CLEAR, NvHkFields, sizeof(NV_HkDefined), NVHkOff);
NVOp(NV_OP_OPEN, NvMon, NvMonSize, NVMonOff);
NVOp(NV_OP_SAVE, NvMon, NvMonSize, NvMonOff);
NVOp(NV_OP_OPEN, NvMon, NvMonSize, NvMonOff);
NVOp(NV_OP_CLEAR, NvHkFields, sizeof(NV_HkDefined));
NVOp(NV_OP_OPEN, NvMon, NvMonSize);
SetNvDefaults(NVGroups, NumGroups);
HKFields.Manuf.Revision = RevLev;
HKFields.Manuf.SerialNumber = SerNum;
HKFields.Manuf.ECOLevel = EColv;
HKFields.Internal.WriteCnt = Writes;
NvMon->WriteCnt = Writes;
NVOp(NV_OP_FIX, NvHkFields, sizeof(NV_HkDefined));
NVOp(NV_OP_SAVE, NvMon, NvMonSize);
Err = NVOp(NV_OP_WRITE, NvMon, NvMonSize, NvMonOff);
if (Err != NVE_NONE) {
    printf(NVSupStr1, "writing", NVErrTable[Err]);
    return;
}
Err = NVOp(NV_OP_WRITE, NvHkFields, sizeof(NV_HkDefined), NVHkOff);
if (Err != NVE_NONE) {
    printf(NVSupStr2, "cannot be initialized");
    return;
}
```

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:

- **NV_OP_WRITE**: Write data to NV memory.
- **NV_OP_READ**: Read data from NV memory.
- **NV_OP_SAVE**: Save data to NV memory.
- **NV_OP_CLEAR**: Clear data from NV memory.
- **NV_OP_OPEN**: Open NV memory for reading or writing.
- **NV_OP_CLOSE**: Close NV memory.
- **NV_OP_NOCLOSE**: No close operation.
- **NV_OP_EXIT**: Exit NV memory.
- **NV_OP_TIMEOUT**: Handle timeout.
- **NV_OP_RESERVED**: Reserved command.
- **NV_OP_FAULT**: Fault operation.
- **NV_OPtility**: NvOpCmd specific to the implementation.

These commands are used to help manage the NV memory section and provide a way to store and retrieve data in a structured manner.
SEE ALSO: *****/

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 < 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:
                    Value = FieldRead(Field->Address, Field->Size);
                    Value = Value | (Field->InitVal << Temp);
                    FieldWrite(Field->Address, Field->Size, Value);
                    break;

                case NV_TYPE_STRING:
                    StrCpy(Field->Address, Field->InitVal);
                    break;
            }
        }
    }
}

DispGroup(Group, EditFlag)
NVGroupPtr Group;
unsigned long EditFlag;
{
    int NumFields = Group->NumFields;
    unsigned long Value, Temp;
    char Buffer[80], RetVal;
    NVFieldPtr Field;

    for (j = 0; j < NumFields; j++) {
        Field = &Group->Fields[j];
        xprintf("%-14s %s", Field->Name, Field->Address);
        switch (Field->Type) {
            case NV_TYPE_HEX:
                if (Field->Aux) {
                    Value = FieldRead(Field->Address, Field->Size);
                    Value = Value | Field->Aux;
                    printf("%x", Value);
                    break;
                }
                break;

            case NV_TYPE_DECIMAL:
                printf("%d", Value);
                break;

            case NV_TYPE_STRING:
                printf("\n", Field->Address);
                break;
        }
    }
}
break;
    case NV_TYPE_BITFIELD:
        Temp = FindBitSet(Field->Aux);
        Value = (Value & Field->Aux) >> Temp;
        DispFieldName(Field->Vals, Temp, Value);
        break;
    case NV_TYPE_VAL_LIST:
        DispFieldName(Field->Vals, Field->Aux, Value);
        break;
    case NV_TYPE_VAL_LIST:
    }

   _DISPFieldName(Field->Vals, Field->Aux, Value);
    }

    if (EditFlag)
    {
        xprintf(":
        
        RetVal = GetString(Buffer);
        xprintf("%.
        
        if (RetVal == CR)
        {
            if (!SetField(Group, Field->Name, Buffer, FALSE))
                --j;
        }

    case NV_OP_OPEN:
    }

    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[0];
        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 Op on module configuration space */
            if (Internals->Magic != NV_MAGIC)
                return(NVE_MAGIC);
            if (Internals->ChkSum != Sum)
                return(NVE_CHKSUM);
            if (Internals->WriteCnt > NVMaxNbrWrites())
                return(NVE_OVERFLOW);
            return(NVE_NONE);
        case NV_OP_OPEN:
        }

        if (Internals->Magic != NV_MAGIC)
            return(NVE_MAGIC);
        if (Internals->ChkSum != Sum)
            return(NVE_CHKSUM);
        if (Internals->WriteCnt > NVMaxNbrWrites())
            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 = 0; ByteNum < Size; ByteNum++)
            {
                NVFunct(WRITE, Offset + ByteNum, &Base[ByteNum]);
                nv_store();
            }
            return(NVE_NONE);
        }
        case NV_OP_CMP:
        {
            Offset += (Size - DataSize); /* Skip Header */
            for (ByteNum = 0; ByteNum < DataSize; ByteNum++)
            {
                if (Base[ByteNum] != NVFunct(READ, Offset + ByteNum))
                    return(NVE_CMP);
            }
            return(NVE_NONE);
        }
        default:
        {
            return(NVE_CMD);
        }
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).

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

#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_82595CA 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 PPU_NONE 0 /* Floating Point is optional */
#define PPU_68881 1
#define PPU_68882 2

define

typedef struct NVU Boot {
    unsigned long RomBase;
    unsigned long RomSize;
    unsigned char Number;
    unsigned long LoadAddress;
    unsigned char AutoBootDev;
    unsigned char PortNum;
    unsigned short PortFlags;
    unsigned long Baud;
    unsigned char Reserved;
    unsigned long PortFlags

    PortFlags = (x->PortFlags & Ox0020)
    PortFlags = (x->PortFlags & Ox0010)
    PortFlags = (x->PortFlags & Ox0008)
    PortFlags = (x->PortFlags & Ox0004)
    PortFlags = (x->PortFlags & Ox0002)
    PortFlags = (x->PortFlags & Ox0001)
    PortFlags = (x->PortFlags & Ox0000)
}

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 */
    unsigned char -Reserved;
}

typedef NvU Port;

/* Warning: These macros only work with pointers */
#define Parity(x) (x->PortFlags & Ox0000)
#define DataBits(x) (x->PortFlags & Ox0000) >> 2)
#define XOnXOff(x) (x->PortFlags & Ox0010)
#define ControlPort(x) (x->PortFlags & Ox0008)
#define StatusPort(x) (x->PortFlags & Ox0004)
#define DataPort(x) (x->PortFlags & Ox0002)
#define ControlPort(x) (x->PortFlags & Ox0001)
#define StatusPort(x) (x->PortFlags & Ox0000)

typedef SP APORT 0 /* Serial Port Assignments */
typedef SP COMPORT 1
typedef SP CPORT 2
typedef SP COMPORT 3
typedef SP PARITY EVEN 0 /* Parity Type Assignments */
typedef SP PARITY ODD 1
typedef SP PARITY NONE 2
typedef SP PARITY FORCE 3
typedef SP DATA 8BITS 0 /* Data Bits Assignments */
typedef SP DATA 8BITS 1
typedef SP DATA 8BITS 2
typedef SP DATA 8BITS 3
typedef SP STOP 8BITS 0
typedef SP STOP 8BITS 1

****** BOOT DEFINITIONS ********************

* This section defines the boot parameters for loading the application
* from a device, and the boot parameters. 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 Device 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 */
}
define NV SMALL
# VME BUS DEFINITIONS

* This structure defines the VMEbus configuration of the slave interface
  * andVIC configuration registers. This structure should be loaded in
    * the user-defined section of the NV memory.

```c
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 */
} NVU_BusConfig;
```

```c
#define ExtSlaveMap(x) (x->SlaveBusMap & 0xFFFF0000)
#define StdSlaveMap(x) (x->SlaveBusMap & 0x0000FFFF)
#define ShlSlaveMap(x) (x->SlaveBusMap & 0x000000FF)
```

### 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.

```c
typedef struct NV_MonDefs {
    /* MonDefs struct = 76/48 */
    NVU_INTERNAL * Internal;
    /* Internal definitions */
    unsigned long MiscFlags; /* Misc monitor flags */
    NVU_PORT Console; /* Console Port Configuration */
    NVU_PORT Debug; /* Debug Port Configuration */
    NVU_BOOT Boot; /* Boot Definitions */
    NVU_BUSCONF VmeBus; /* Bus Configuration Definitions */
} NV_MonDefs;
```
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
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, IA, 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
  Comware 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
Index

7-segment display 8-1
82596CA 12-1
82C501AD 12-1

A
abbreviations in monitor commands A-9
add, monitor command A-30
address error 5-2
address modes, bus 7-14
addresses, summary 14-2
addressing mode, Ethernet controller 12-4
ambiguous command, monitor error A-43
arbitration, bus 7-4
arguments to monitor memory commands A-29
autoboot cancellation A-44
autosources 3-3

B
baud rates 10-6
big-endian byte ordering, Ethernet 12-3
binary arguments, monitor A-29
binary format records, downloading with monitor A-25
Board.c B-1
Board.h B-1
boot devices, configuration with monitor A-14
Boot group, monitor display A-30, 31
boot-up A-46
bootbus, monitor command A-30
BootParams Group, monitor A-37
boostrom, monitor command A-31
bootserial, monitor command A-31
breakpoints A-40
Bug.h B-1
bus address modes 7-14
bus arbitration 7-4
bus control 7-1
bus control latch, with monitor commands A-15
bus error (MPU) 5-1
bus grant jumpers, J14, J15, J17, J18 7-5
bus interface 7-17
bus interrupts 7-3, 12
bus memory 6-4
bus priorities 7-2, 4
bus request jumper, J16 7-5
bus throttle timers 12-4
bus timer 7-11, 16
by-pass, FPP 4-2
byte ordering 12-5

C
cache control 3-7
Cache Group, monitor A-37
call, monitor command A-31
Centronics interrupt 9-2
clock, CIO 9-3
clear mem, monitor command A-32
clock, CIO 9-3
clearmem, monitor command A-32
clearmem, monitor command A-31
CIO usage 9-1
clearmem, monitor command A-32
clock, CIO 9-3
clearmem, monitor command A-32
clearmem, monitor command A-8
clearmem, monitor command A-8
clearmem, monitor command A-2
configboard, monitor command A-32
configuration jumpers 14-3
configuration, memory 6-1
connectors 1-6
connectors, overview 1-6
Console Group, monitor A-38
counter/timers (CIO) 9-2, 3
current, monitor command A-32
cRC terminal, setup 2-7

D
date, monitor command A-32
debugging applications, with monitor A-27
defaults, monitor A-46
device I/O execution A-46
disassembler, monitor command A-33
displaymem, monitor command A-33
div, monitor command A-33
divide by zero error 5-2
double bus fault error 5-1
Download Group, monitor A-38
download, monitor command A-17, 16, 34
dumpregs, monitor command A-34

E
editor, monitor commands A-8
EPROM 2-10
equipment for setup 2-3
error, timer frequency 9-3
errors A-43
errors, system response 5-1
ESD prevention 2-3
Ethernet address 12-2
Ethernet byte ordering 12-5
Ethernet exception conditions 12-13
Ethernet peripheral address 12-10
exception conditions, Ethernet 12-13
exception vectors 3-2
exception vectors, MPU 3-3
exceptions, FPP 5-2
exectrace, monitor command A-34

F
feature summary 1-1
fillmem, monitor command A-34
findmem, monitor command A-35
findnotmem, monitor command A-35
findstr, monitor command A-35
flags required for monitor memory commands A-29
flags, for monitor commands A-9
floating point processor (FPP) 4-1
format for monitor commands A-9
format for monitor memory commands A-29
FPP exceptions 5-2
free memory A-47
function summary, monitor A-45
functional description 1-4

H
HALT state 3-6; 5-1; 7-15
help, monitor command A-35
hex-Intel file, example A-21
hex-Intel records A-17, 18, 34
hexadecimal arguments, monitor A-29
history of commands, monitor A-8

I
illegal instruction error 5-2
indicators, status LEDs 3-6
initialization error, nonvolatile memory A-43
initialization, CIO 9-3
initialization, Ethernet controller 12-9
initialization, software 14-1
initializing memory, from the monitor A-11
initializing nonvolatile memory, caution A-38
initializing the board to defaults A-46
installation 2-3
installation and setup, monitor A-5
interrupt acknowledge 7-13
interrupt enable 12-3
interrupt handler, bus 7-13
interrupt levels, MPU 3-1
interrupt pending 7-13; 9-1
interrupt polarity 12-4
interrupt vector values 3-5
interrupt, mailbox 7-15
interrupt, mailbox 9-1
interrupter module, bus 7-12
interrupts 12-11
interrupts, bus 7-3
interrupts, VMEbus, unmasking A-45, 46
IRQ interrupt 3-3

J
J14, J15, J17, J18, bus grant jumpers 7-5
J16, bus request jumper 7-5
jumper summary 14-3
jumpers 1-6
jumpers, bus control 7-17
jumpers, ROM type 6-2
jumpers, serial I/O 10-7
jumpers, watchdog 7-17

L
LEDs, status 3-6
LoadAddress field, monitor A-30
locked bus cycles 12-4

M
Mailbox Group, monitor A-37
mailbox interrupt 7-15; 9-1
mechanical specifications 14-6
memory configuration 6-1
memory management A-47
memory map 6-4
memory sizing 6-4
Memory Test error A-44
memory test, monitor A-7
memory timing 6-6
memory, bus 6-4
Misc Group, monitor A-37
monitor commands, errors A-44
monitor defaults, nonvolatile memory
configuration A-6
monitor entry point A-45
monitor operation A-1
monitor program 2-12
MPU status 3-6
MPU summary 3-1
mul, monitor command A-35

N
network interface controller (82596CA) 12-1
nonvolatile configuration and nonvolatile
autoboot A-7
nonvolatile memory configuration,
monitor defaults A-6
nonvolatile RAM 6-7; C-1
number base definitions, monitor A-29
number bases, for monitor arguments A-9
numeric format A-9
NVRAM C-1
nvdisplay, monitor command A-11, 36
nvinit, monitor command A-38
nvopen, monitor command A-38
nvset, monitor command, caution A-39
nvupdate A-11
nvupdate, monitor command A-39

O
octal arguments, monitor A-29
operating temperature 2-3
oscillator and reset bits, RTC 13-6

P
P1 pin assignments, VMEbus 7-18
P1 signal descriptions 7-1
P2 pin assignments (SCSI) 11-2
P2 pin assignments (VMEbus) 7-19
P3 pinout (serial) 10-1
P4 pin assignments 12-14
P5 pinout (status) 3-6
paged ROMs, 3
parity, RAM 3-1; 5-1; 6-4
physical memory map 6-4
pin assignments, real-time clock 13-4
pin assignments, VMEbus, P1 7-18
pinouts, SCSI and VMEbus 7-19
pinouts, VMEbus 7-18
port access 12-5, 6
port addresses, CIO 9-3
port addresses, SCC 10-6
port addresses, SCSI 11-2
port addresses, summary 14-2
ports, overview 1-6
power requirements 14-6
power-up errors A-44
power-up memory configuration 6-1
precautions 2-3
privileged violation error 5-2
Proc.c B-1
Proc.h B-1
ProcAsm.s B-1
protected fields, nonvolatile memory A-13
prstatus, monitor command A-39

R
radix, monitor A-29
RAM parity 6-4
RAM, bus 6-4
RAM, nonvolatile 6-7; C-1
RAM, on-card 6-4
rand, monitor command A-39
reading memory, monitor command A-13
readmem, monitor command A-39
real time clock 13-1
real time clock error A-44
real-time clock, pin assignments 13-4
register definition, RTC 13-7
register summary 14-2
registers, RTC 13-6
Release-without-hold bit 7-11
reset 7-15
reset sequence, monitor A-5
reset switch 1-6, 7
reset vector 3-3; 14-1
ROM 2-10; 6-1
RPACKs, SCSI 11-4
RS-232 conventions 10-3
RS-232 pinouts 10-1
RS-422 operation 10-7
RTC nonvolatile controller 13-6
RTC registers 13-6
RTC, AM-PM/12/24 mode, 6
RTC, description of operation 13-4
RTC.c B-1

S
S-record file, example A-24
S-records A-17, 22, 34
S0 records A-22
S1 data records A-22
S2 data records A-22
S3 data records A-22
S5 data count records A-23
S7 termination and start address records A-23
S8 termination and start address records A-23
S9 termination and start address records A-23
SCC.c B-1
screen messages A-43
SCSI bus termination 11-4
SCSI pin assignments, P2 7-19
SCSI port 11-1
serial I/O 10-1
serial network interface (82C501AD) 12-1
serial numbers 2-1
serial port control, from the monitor A-46
Serial Test error A-44
serial test, monitor A-7
service 2-10
setdate, monitor command A-40
setmem, monitor command A-40
settrace, monitor command A-40
setup 2-3
setup, CRT terminal 2-7
seven-segment display 8-1
short addresses, bus mode 7-14
sizing memory 6-4
slavedis, monitor command A-41
slaveenable, monitor command A-41
software initialization 14-1
standard addresses, bus mode 7-14
start-up display, monitor A-7
starttimer, monitor command A-41
status LEDs 3-6
status, MPU 3-6
status/ID byte 7-12
step, monitor command A-41
stop timer, monitor command A-41
string format A-9
sub, monitor command A-42
summary, features 1-1
swapmem, monitor command A-42
symbol format A-9
syntax for monitor commands A-9
Index

SYSBUS byte 12-3, 8
SYSFAIL bus signal 7-3, 14
System Configuration Pointer 12-3
System Configuration Pointer address 12-4
System Control Block 12-3
system controller board 7-4, 17
system errors 5-1

T
technical documents 1-8
terminators, SCSI 11-4
testmem, monitor command A-42
Timer.c B-1
timers (CIO) 9-3
timing, memory 6-6
tracing A-40
transmode, monitor command A-42
transparent mode, monitor A-25, 26
troubleshooting 2-10

U
unmasking VMEbus interrupts A-46

V
vectors, exception 3-2
VME.c B-1
VMEbus connectors, P1 and P2 7-17
VMEbus description 7-1
VmeBus Group, monitor A-38
VMEbus interrupts 7-12
VMEbus interrupts, unmasking A-45, 46
VMEbus pin assignments, P2 7-19
VMEbus pinouts (P1) 7-17

W
watchdog, vector 3-3; 5-1; 7-11, 16
writemem, monitor command A-42
writestr, monitor command A-42
writing memory, monitor command A-13

Z
zero bits, RTC 13-7