SPARC CPU-10
TECHNICAL REFERENCE MANUAL

REVISION NO. 1
FEBRUARY 1995

List of Figures

List of Tables

SECTION 1 Introduction

1. Getting Started

This *SPARC CPU-10 Technical Reference Manual* provides a comprehensive guide to the SPARC CPU-10 board you purchased from FORCE COMPUTERS. In addition, each board delivered by FORCE includes an *Installation Guide.*

Please take a moment to examine the Table of Contents of the *SPARC CPU-10 Technical Reference Manual* to see how this documentation is structured. This will be of value to you when looking for information in the future.

1.1 The SPARC CPU-10 Technical Reference Manual Set

When purchased from FORCE COMPUTERS, this set includes the *SPARC CPU-10 Technical Reference Manual* as well as two additional books. These two books are listed here:

Set of Data Sheets for the SPARC CPU-10

OPEN BOOT PROM 2.0 MANUAL SET

The *Set of Data Sheets for the SPARC CPU-10* contains the following data sheets.

NCR89C100 (MACIO)

TI superSPARC Processor (TMS390Z50)

TI superSPARC Cache Controller (TMS390Z55)

MB86986 VMEbus Interface Chip (MVIC)

The L64861 (SEC)

The L64860 (EMC)

The L64862 (MSI)

T7259 ISDN/Audio Interface (DBRI)

82077SL Floppy Disk Controller

85C30 Serial Controller

SGS-THOMSON MK48T08(B)-10/12/15/20

AMD Flash EPROM (AM28F020)

Intel Flash Memory (28F008SA-L)

The *OPEN BOOT PROM 2.0 MANUAL SET* contains the following three sections.

Open Boot 2.0 Quick Reference

Open Boot 2.0 Command Reference

FCODE Programs

1.2 Summary of the SPARC CPU-10

The SPARC CPU-10 is a VMEbus computer based on the superSPARC CPU chip which is a highly integrated implementation of the SPARC RISC superprocessor.

Performance of the SPARC CPU-10 is scalable through the use of a standard Mbus slot, which allows the installation of one SuperSPARC Mbus module, with or without secondary cache and frequency options. Through this combination of powerful processing power, with a full set of I/O interfaces including fast SCSI, Ethernet, floppy disk, serial I/O, Centronics compliant parallel I/O, keyboard/mouse ports, audio and ISDN interface, the SPARC CPU-10 becomes a high performance solution for embedded applications.

A full 32-bit IEEE 1014 VMEbus interface and two industry standard SBus sockets enable the expansion of memory, I/O and processing performance via a broad range of off-the-shelf solutions.

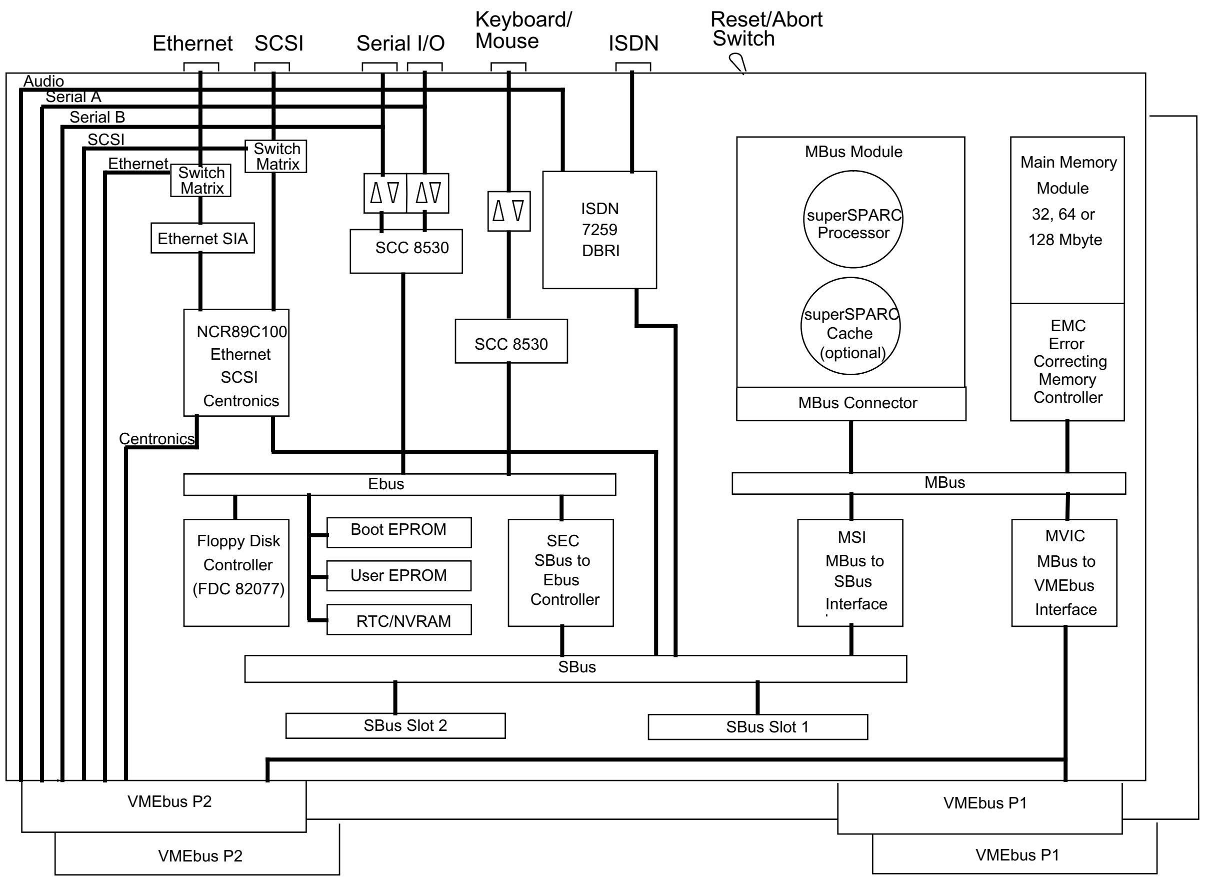
Every SPARC CPU-10 includes an EPROM based monitor/debugger called OpenBoot^{TM}, which provides the functionality of the boot device as well as the setup for the VMEbus interface. The software support for the SPARC CPU-10 ranges from Solaris^{TM}, the most popular implementation of the UNIX operating system, to sophisticated real-time operating systems such as VxWorks.

The SPARC CPU-10 is a two slot VMEbus computer combining workstation performance and functionality with the ruggedness and expandability of the industry standard 6U VMEbus form factor.

1.3 Block Diagram of the SPARC CPU-10

For an overview of your SPARC CPU-10, please examine the block diagram below.

FIGURE 1. Block Diagram of the SPARC CPU-10



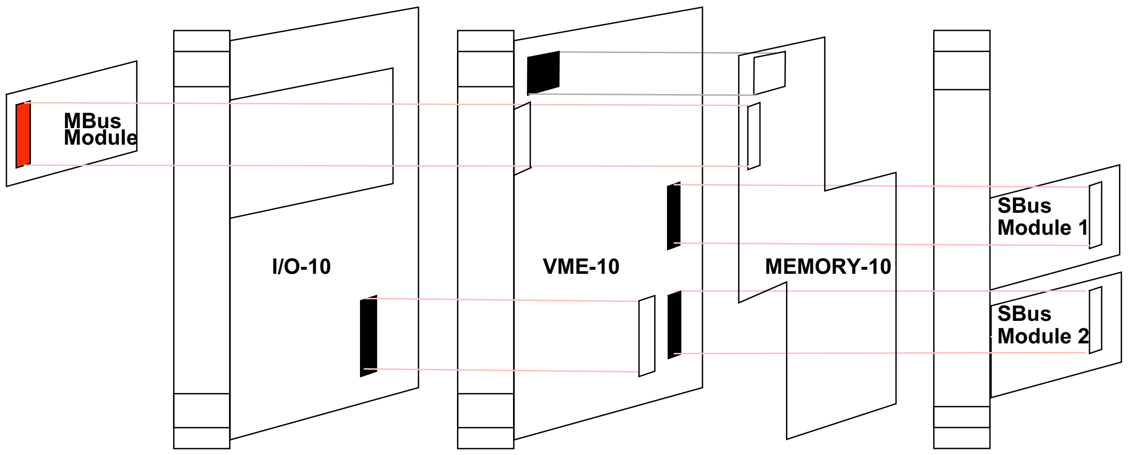
1.4 Mechanical Construction of the SPARC CPU-10

The figure below shows the mechanical construction of the SPARC CPU-10 Board. This is a two slot VMEbus computer in a standard 6U VMEbus form factor.

It consists of up to six board modules:

- SuperSPARC Mbus module
- I/O-10
- VME-10
- Memory-10
- SBus Module 1(optional)
- SBus Module 2 (optional)

FIGURE 2. Mechanical Construction of the SPARC CPU-10



1.5 Specifications

Below is a table outlining the specifications of the SPARC CPU-10 board.

Table 1: Specifications of the SPARC CPU-10

Processor Clock Frequency / Sec. Cache SPECint92 SPECfp92 MIPS MFLOPS	superSPARC 40 MHz / 0 Mbyte 50.2 60.2 109.5 22.9
Processor Clock Frequency / Sec. Cache SPECint92 SPECfp92 MIPS MFLOPS	superSPARC 50 MHz / 1 Mbyte 65.2 83.0 135.5 27.3
Processor Clock Frequency / Sec. Cache SPECint92 SPECfp92 MIPS MFLOPS	superSPARC 60 MHz / 1 Mbyte 77.9 98.0 161.0 31.2
Memory Management Unit	SPARC Reference MMU
Data/Instruction Cache	16Kbyte/20Kbyte
Shared Main Memory	32, 64 or 128 Mbyte DRAM
SBus Slots	2, mechanically compatible to CPU-3CE
SCSI with DMA to SBus	NCR89C100 10 Mbytes/sec 53C90A superset I/O on front panel and P2
Ethernet with DMA to SBus	NCR89C100 10 Mbits/sec AM7990 compatible I/O on front panel or P2
Parallel port with DMA to SBus	NCR89C100 3.3 Mbytes/sec Centronics compatible Uni- or bidirectional I/O on P2

Table 1: Specifications of the SPARC CPU-10 (Continued)

Floppy Disk Interface	82077SL 250, 300, 500 Kbit/sec and 1 Mbit/sec I/O on P2
Serial I/O	8530 2 ports with RS-232 configuration, Optional RS-422/485 configuration via hybrid modules I/O on front panel and P2
Keyboard/Mouse Port	8530 Sun compatible I/O on front panel and P2
Counters/Timers	Five 32-bit, programmable
Boot Flash Memory	512 Kbyte On-board programmable Hardware write protection
User Flash Memory	1 Mbyte (2 Mbyte Option) On-board programmable Hardware write protection
RTC/NVRAM/Battery Usable Memory	MK48T08 6 Kbyte
VMEbus Interface	32-bit master/slave, IEEE-1014
ISDN Interface with DMA to SBus	T7259 Dual Basic Rate (2B&D) CCITT I.430/ANSI T1.605 I/O on front panel (TE/NT)
Audio Interface with DMA to SBus	T7259 16-bit CD-quality audio I/O on P2
Additional Features	Reset / Abort switch Status LEDs on the front panel Watchdog timer
Firmware	OpenBoot with diagnostics
Power consumption	+5V 9.5A (No SBus Modules installled) +5V 13.5A (Two SBus Modules installled) +12V 0.7A -12V 0.2A

Table 1: Specifications of the SPARC CPU-10 (Continued)

Environmental Conditions Temperature (Operating) Temperature (Storage) Humidity	 0^{o} C to $+50^{o}$ C -40^{o} C to $+85^{o}$ C 0% to 95% noncondensing
Board Size	Two Slot 6U VMEbus (No SBus Modules installed) Three Slot 6U VMEbus (SBus Modules installed) 160.00 x 233.35 mm 6.29 x 9.18 inches

1.6 Product Nomenclature

FORCE COMPUTERS' SPARC CPU-10 is available in several memory and speed options. Consult your local sales representative to confirm availability of specific combinations.

The table below explains the product nomenclature.

Table 2: Product Nomenclature

CPU-10/xxx-yy-z		
Main Memory (xxx)	superSPARC Clock Frequency / Secondary Cache Size (yy)	User EPROM (z)
32 = 32 Mbyte 64 = 64 Mbyte 128 =128 Mbyte	40 = 40 MHz / 0 Mbyte 51 = 50 MHz / 1 Mbyte 61 = 60 MHz / 1 Mbyte	0 = 0 Mbyte 1 = 1 Mbyte 2 = 2 Mbyte
Sample Product Nomenclature		
CPU-10/32-40-1	40 MHz superSPARC CPU, no secondary cache, 32-Mbyte DRAM, 1-Mbyte user Flash EPROM, SCSI, Ethernet, floppy disk, parallel and 2 serial I/O ports, 32-bit VMEbus interface, ISDN/audio interface, 2 SBus slots, OpenBoot firmware. User documentation included.	
Other Available Configurations		
CPU-10/64-40-z	as above, except 64 Mbyte DRAM	
CPU-10/32-51-z	as above, except 50 MHz, 1 Mbyte secondary cache	
CPU-10/64-51-z	as above, except 64 Mbyte DRAM, 50 MHz, 1 Mbyte secondary cache	
CPU-10/32-61-z	as above, except 60 MHz, 1 Mbyte secondary cache	
CPU-10/64-61-z	as above, except 64 Mbyte DRAM, 60 MHz, 1 Mbyte secondary cache	
CPU-10/128-61-z	as above, except 128 Mbyte DRAM, 60 MHz, 1 Mbyte secondary cache	

1.6.1 Ordering Information

This page contains a list of the product names and their descriptions.

Table 3: Ordering Information

Catalog Name	Product Description
SBus Modules	
SBus/Mono	Monochrome frame buffer, 1152 x 900, single SBus slot.
SBus/Color	Color frame buffer, 1152 x 900, 8 bits per pixel, single SBus slot.
SBus/GX	Color 2D and 3D wire frame graphics accelerator, 1152 x 900, 8 bits per pixel, single SBus slot.
SBus/GX+	Color 2D and 3D wire frame graphics accelerator, 1280 x 1024, 8 bits per pixel, double buffering, uses two SBus slots.
SBus/FP	Front panel for up to 2 SBus cards for CPU-2CE, CPU-3CE and CPU-10
Accessories	
CPU-10/TM	Technical Reference Manual Set for SPARC CPU-10 including OpenBoot User's Manual, a detailed hardware description and Set of Data Sheets for the SPARC CPU-10.
IOBP-10	I/O back panel on VMEbus P2 with flat cable connectors for SCSI, Floppy, Centronics, Serial, Audio and a micro-DSUB connector for Ethernet. For use with CPU-3CE and CPU-10.
Serial-2CE	Serial adapter cable for CPU-2CE, CPU-3CE and CPU-10, 26-pin shielded to 25-pin DSub.
TARGET-32 Cable Set 5	Cable set for 13W3 graphics card connector to 3U back panel with BNC connectors.
FH003/SET	Hybrid modules for RS-422 serial I/O configuration.
FH005/SET	Hybrid modules for RS-485 serial I/O configuration.
Software	
Solaris 2.x/CPU-10	Solaris 2.x two-user client/desktop right-to-use license with VMEbus driver. Includes media. Please contact your local sales representative for current version information.
Solaris 2.x/CPU-10/ Client -RTU	Solaris 2.x client/desktop right-to-use license. Without media. Please contact your local sales representative for current version information.

Table 3: Ordering Information (Continued)

Catalog Name	Product Description
Solaris 2.x/CPU-10/ Server -RTU	Solaris 2.x server right-to-use license. Without media. Please contact your local sales representative for current version information.
Solaris 2.x/UM	Solaris 2.x operating system user manual. Please contact your local sales representative for current version information.
Solaris 1.1/CPU-10	Solaris 1.1 two-user right-to-use license with VMEbus driver. Includes media.
Solaris 1.1/CPU-10/2U-RTU	Solaris 1.1 two-user right-to-use license. Without media.
Solaris 1.1/CPU-10/UU-RTU	Solaris 1.1 unlimited-user right-to-use license. Without media.
Solaris 1.1/UM	Solaris 1.1 operating system user manual.
VxWorks/DEV SPARC Products	VxWorks development package for SPARC host and target.
VxWorks/BSP CPU-10	VxWorks board support package for CPU-10
VxWorks/Solaris Driver CPU-10	VxWorks/Solaris CPU-10 back plane driver.

1.7 History of the Manual Publication

Below is a description of the publication history of this *SPARC CPU-10 Technical Reference Manual.*

Table 4: History of Manual

Revision No.	Description	Date
0	Preliminary Release	August 1993
1	First Print	January 1994
2	(Next print we describe the error on page 114) It has been corrected in June with a "READ ME FIRST"	?

SECTION 2 Installation

2. Installation

This Installation Section provides guidelines for powering up the SPARC CPU-10 board. The Installation Section, which you have in your hand now, appears both as Section 2 of the *SPARC CPU-10 Technical Reference Manual* and as a stand-alone *Installation Guide*. The stand-alone Installation Guide is delivered by FORCE COMPUTERS with every board. *The SPARC CPU-10 Technical Reference Manual* provides a comprehensive hardware and software guide to your board and is intended for those persons who require complete information.

2.1 Caution

Please read the Installation Section before installing the board. Take a moment to examine the Table of Contents to see how this section of the documentation is structured. This will be of value to you when looking for specific information in the future.

2.2 Diagrams of the SPARC CPU-10 Board

The SPARC CPU-10 board consists of up to six board modules: the VME-10, the I/O-10, the MEMORY-10, the MBus module and the optional SBus modules. The construction of the SPARC CPU-10 board can be seen in the drawing below. Highlighted diagrams showing the components of the VME-10, the I/O-10 and the MEMORY-10 appear on the next few pages. The purpose of these highlighted diagrams is to bring to the user's attention the position of important components on the board.

FIGURE 3. Mechanical Construction of the SPARC CPU-10

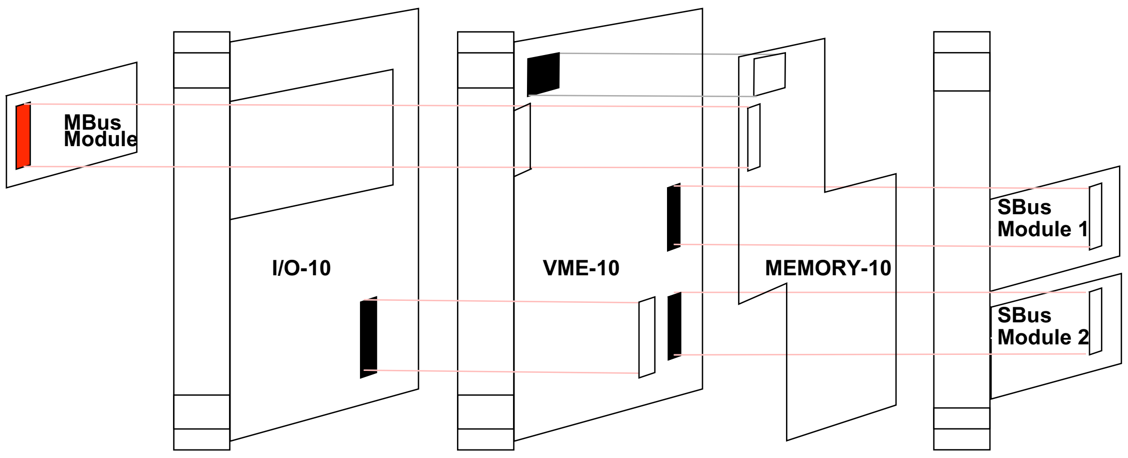


FIGURE 4. Location Diagram of the VME-10

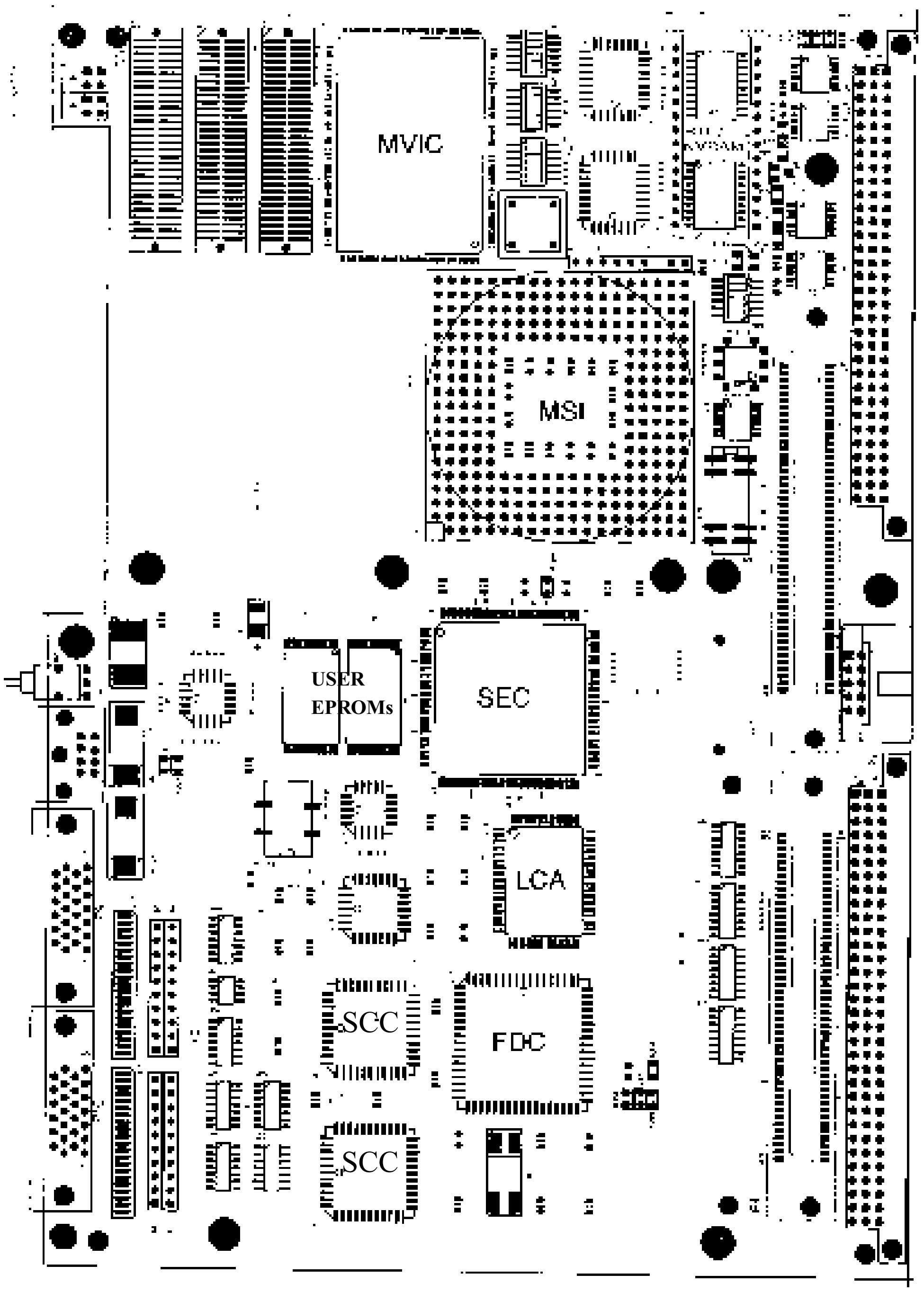


FIGURE 5. Highlighted Location Diagram of the VME-10

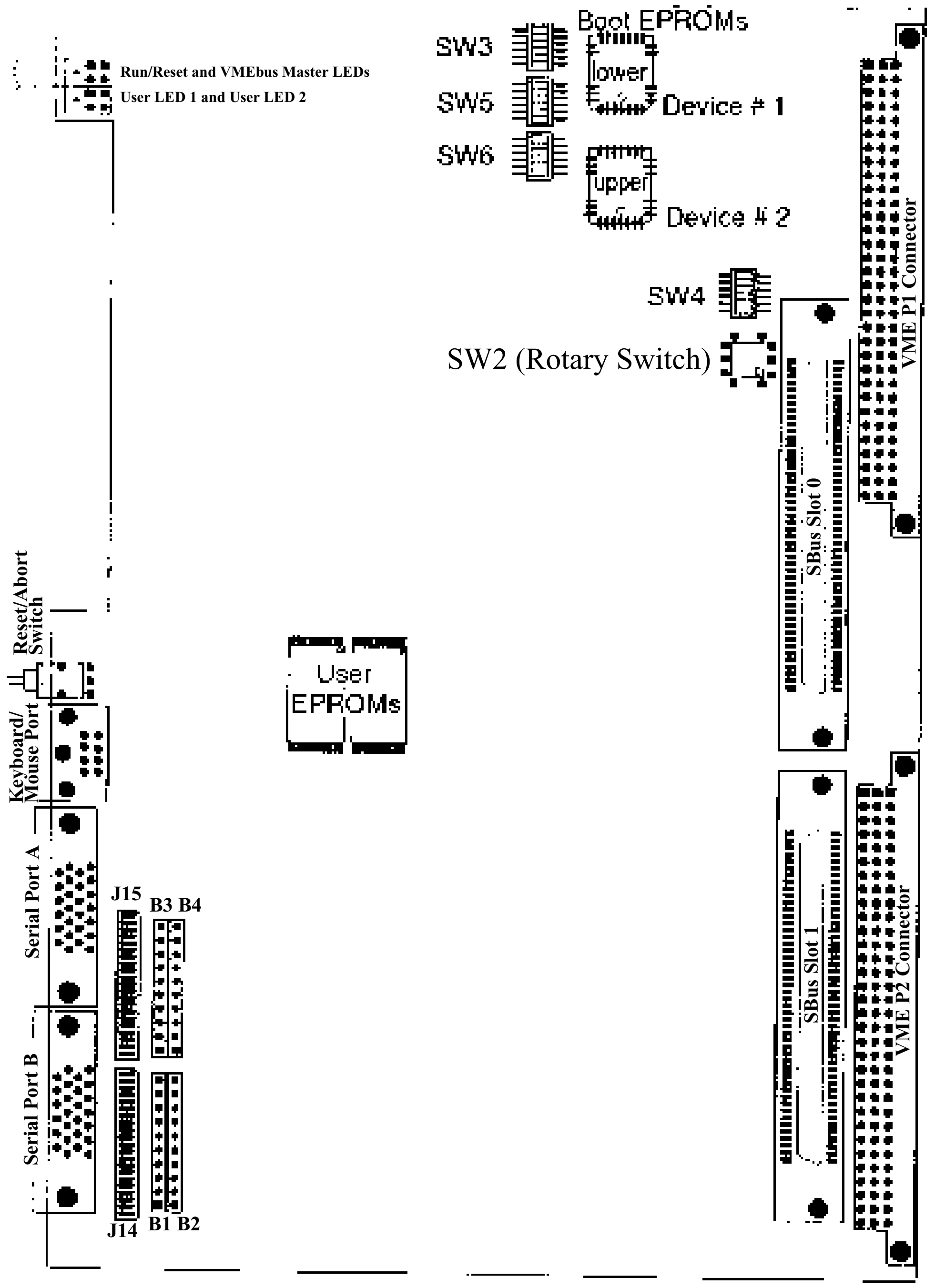


FIGURE 6. Location Diagram of the I/O-10

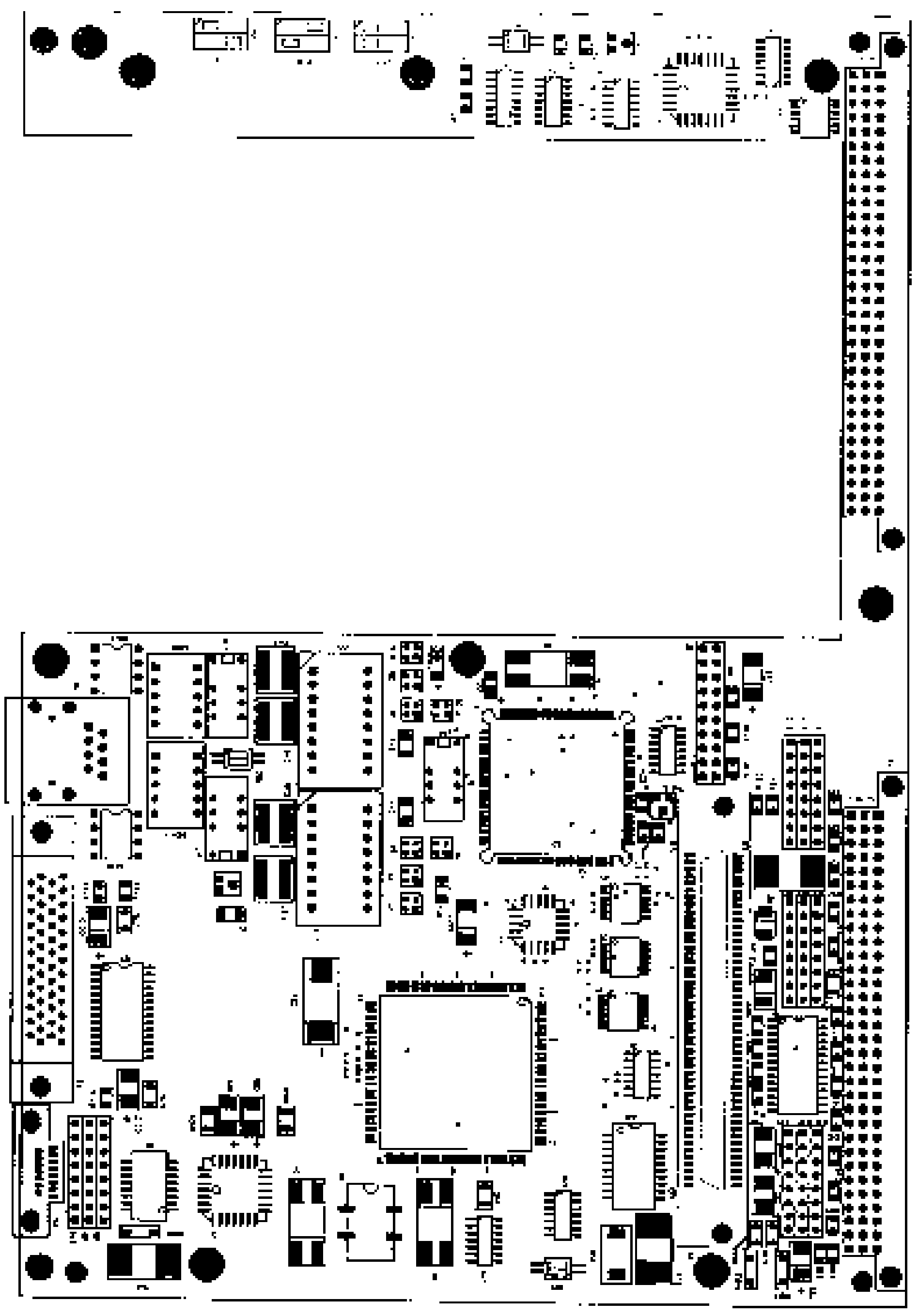


FIGURE 7. Highlighted Location Diagram of the I/O-10

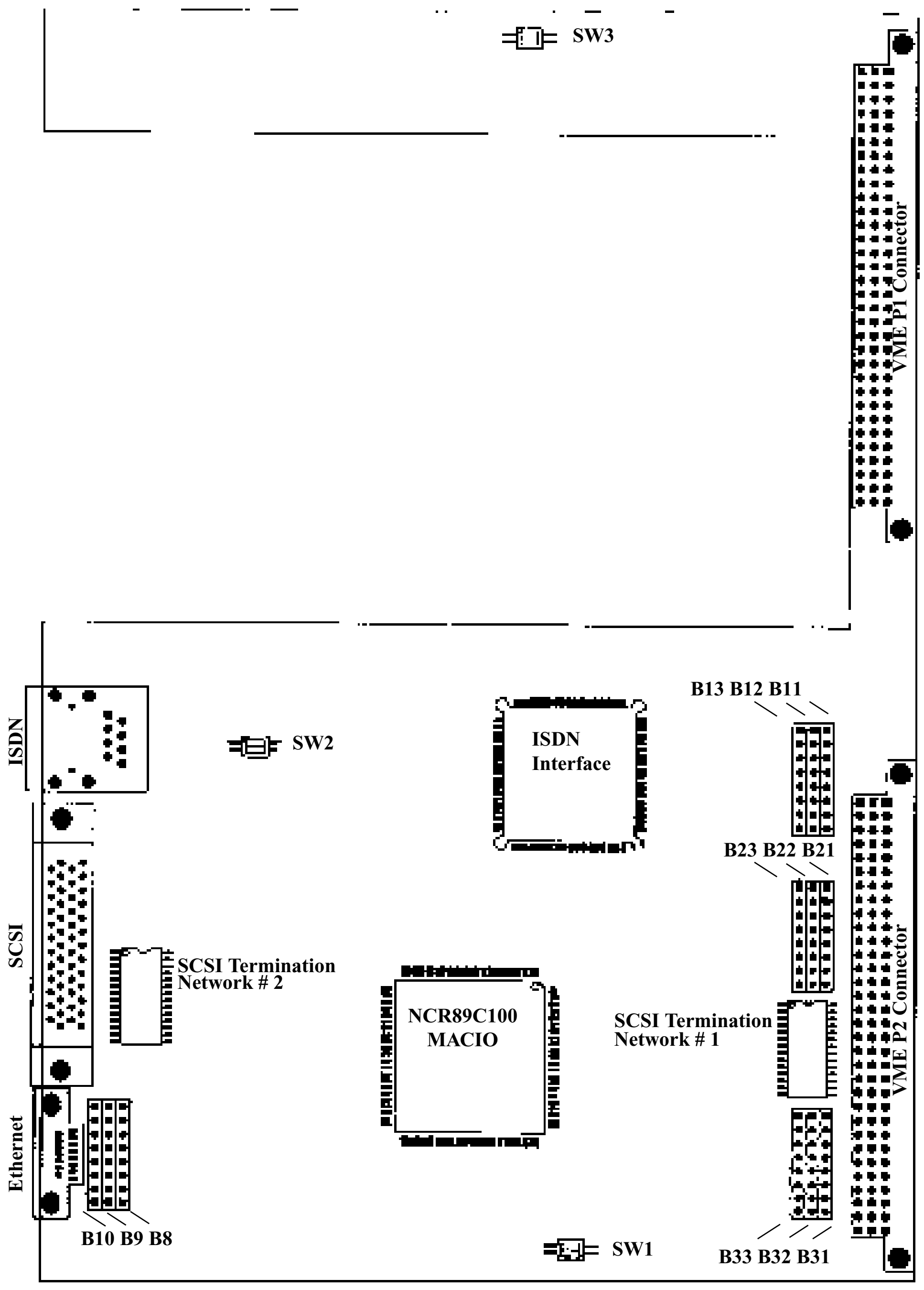
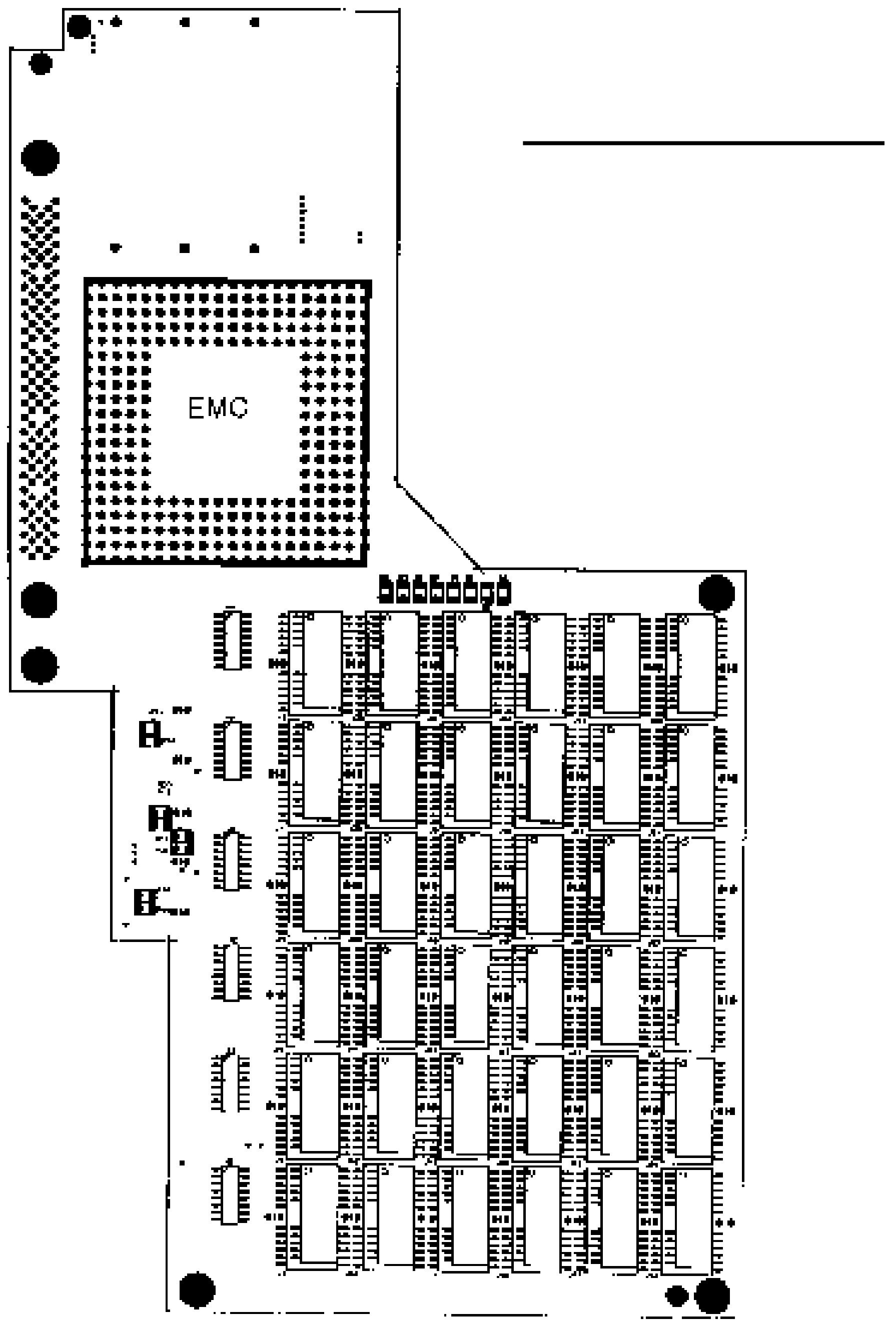


FIGURE 8. Location Diagram of the MEMORY-10



2.3 Before Powering Up

Before powering up, please make sure that the default switch settings are all set according to the table below. Please also make sure that the switches on the I/O-10 board are set according to the table “Default Switch Settings on the I/O-10 Board” on page 22. Check these switch settings *before* powering up the SPARC CPU-10 because the board is configured for power up according to these default settings. The figure “Highlighted Location Diagram of the VME-10” on page 15 shows the position of the switches on the VME-10 board and the figure “Highlighted Location Diagram of the I/O-10” on page 17 shows the position of the switches on the IO-10 board. Now is an excellent time to examine the switches to confirm that they are correctly set.

NOTE: Please note that pin # 1 is always located near the small diagonal line on the switch. This is shown in the diagrams of the switches below. When examining the board, you can always identify pin # 1 by finding this small diagonal line on the switch.

2.3.1 Default Switch Settings on the VME-10 Board

Table 5: Default Switch Settings on the VME-10 Board

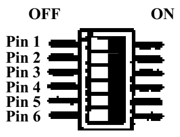
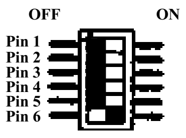
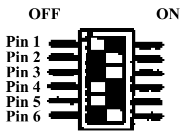
Diagram of the Switch on the VME-10 Board	Switches	Default Setting	Function
SWITCH 3 (Controls Reset)			
Switch 3  OFF ON Pin 1 Pin 2 Pin 3 Pin 4 Pin 5 Pin 6	SW3-1	ON	VME Slot-1 Device ON=Yes OFF=No
	SW3-2	ON	Front Panel Reset Key Configuration
	SW3-3	ON	VME Reset generates Board Reset ON=Yes OFF=No
	SW3-4	ON	Power up Reset generates VME Reset ON=Yes OFF=No
	SW3-5	ON	Trigger Voltage for Power up Reset ON=4.85V OFF=4.0V
	SW3-6	ON	Front Panel Reset Key Configuration

Table 5: Default Switch Settings on the VME-10 Board (Continued)

Diagram of the Switch on the VME-10 Board	Switches	Default Setting	Function
SWITCH 4 (Controls Boot EPROM and User EPROM)			

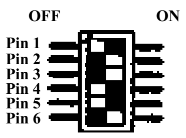
Switch 4  OFF
ON
Pin 1
Pin 2
Pin 3
Pin 4
Pin 5
Pin 6
	SW4-1	OFF	Boot EPROM Device Selection
			ON=Standard PROM/EPROM
			OFF=Flash Memory

	SW4-2	OFF	Boot EPROM Write Protection ON=Not Protected OFF= Protected
	SW4-3	OFF	User EPROM Write Protection ON=Not Protected OFF= Protected
	SW4-4	OFF	Unused
	SW4-5	OFF	Unused
	SW4-6	ON	Reserved for Test Purposes
SWITCH 5 (Controls Serial Channel A)			
Switch 5  OFF ON Pin 1 Pin 2 Pin 3 Pin 4 Pin 5 Pin 6	SW5-1	ON	TRXC on Front Panel Connector for RS-232 ON=Available, OFF=Not Available

	SW5-2	OFF	RTS functions as TEN for RS-485
			ON=TEN function enabled
			OFF=TEN function disabled

	SW5-3	OFF	TRXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW5-4	ON	RTS on Front Panel Connector for RS-232 OR RTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW5-5	ON	CTS on Front Panel Connector for RS-232 OR CTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW5-6	OFF	RTXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available

Table 5: Default Switch Settings on the VME-10 Board (Continued)

Diagram of the Switch on the VME-10 Board	Switches	Default Setting	Function
SWITCH 6 (Controls Serial Channel B)			
Switch 6  OFF ON Pin 1 Pin 2 Pin 3 Pin 4 Pin 5 Pin 6	SW6-1	ON	TRXC on Front Panel Connector for RS-232 ON=Available, OFF=Not Available

	SW6-2	OFF	RTS functions as TEN for RS-485
			ON=TEN function enabled
			OFF=TEN function disabled

	SW6-3	OFF	TRXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW6-4	ON	RTS on Front Panel Connector for RS-232 OR RTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW6-5	ON	CTS on Front Panel Connector for RS-232 OR CTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW6-6	OFF	RTXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available

CAUTION: To avoid damaging the serial ports, please consider the following regarding Switch 5 and Switch 6. Do not set the switches (SW5-1 and SW5-2), or (SW5-3 and SW5-4), or (SW5-5 and SW5-6) to ON at the same time and do not set the switches (SW6-1 and SW6-2), or (SW6-3 and SW6-4), or (SW6-5 and SW6-6) to ON at the same time!

2.3.2 Default Switch Settings on the I/O-10 Board

Before powering up, please make sure that the default switch settings are all set according to the table below and the table “Default Switch Settings on the I/O-10 Board” on page 22. Check all the switch settings *before* powering up the SPARC CPU-10 because the board is configured for power up according to these default settings. For the position of the switches on the I/O-10 board, please see the “Highlighted Location Diagram of the I/O-10” on page 17.

NOTE: Please note that pin # 1 is always located near the small diagonal line on the switch. This is shown in the diagrams of the switches below. When examining the board, you can always identify pin # 1 by finding this small diagonal line on the switch.

Table 6: Default Switch Settings on the I/O-10 Board

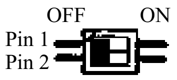
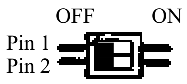
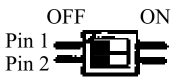
	Switches	Default Setting	Function
SWITCH 1 (Controls SCSI Termination)			
Switch 1 	SW1-1	OFF	SCSI Termination Network # 1 ON=Enabled OFF=Disabled
	SW1-2	OFF	Unused
SWITCH 2 (Controls ISDN*) * ISDN is implemented on the SPARC CPU-10 for use in future applications.			
Switch 2 	SW2-1	OFF	ISDN Loop ON=Enabled OFF=Disabled
	SW2-2	OFF	ISDN Loop ON=Enabled OFF=Disabled

Table 6: Default Switch Settings on the I/O-10 Board (Continued)

	Switches	Default Setting	Function
SWITCH 3 (VMEbus Arbiter Mode)			
Switch 3 	SW3-1	OFF	**SW3-1** **SW3-2** **Arbiter Mode** OFF OFF Round Robin OFF ON Prioritized ON OFF Prioritized Round Robin ON ON Single Level
	SW3-2	OFF	

2.4 Powering Up

The initial power up can easily be done by connecting a terminal to ttya (serial port A). The advantage of using a terminal is that no frame buffer, monitor, or keyboard is used for initial power up, which facilitates a simple start up.

Please see the chapter “OpenBoot Firmware” on page 33 for more detailed information on booting the system.

2.4.1 VME Slot 1 Device

The SPARC CPU-10 can be plugged into any VMEbus slot; however, the default configuration sets the board as a VME slot 1 device, which functions as VME system controller. To configure your CPU-10 as a non-VME slot 1 device, the default configuration must be changed. The following table shows the switch setting of SW3-1 on the VME-10 board for the VME slot 1 configuration.

Table 7: VME Slot 1 Device

SW3-1	VME Slot 1 Device	Default
OFF	No	
ON	Yes	*

2.4.2 Serial Ports

By default, both serial ports are configured as RS-232 interfaces. It is also possible to configure both ports as RS-422 or RS-485 interfaces. This optional configuration is achieved with the special FORCE Hybrids FH-003 and FH-005.

The chapter “Default Switch Settings on the VME-10 Board” on page 19 shows the necessary switch settings for RS-232 operation, where SW5 controls serial port A and SW6 controls serial port B. Please check that the switches are set accordingly.

2.4.3 RESET Key Configuration

When toggled **up**, the RESET / ABORT Key on the front panel can trigger two types of resets. The first reset is called POWERUP_RESET and resets the board and reconfigures the LCA on the CPU-10. The second reset is called BOARD_RESET and does **not** reconfigure the LCA on the CPU-10.

The following table shows the switch settings of SW3-2 and SW3-6 on the VME-10 board for the possible reset configurations.

Table 8: RESET Key Configuration

SW3-2	SW3-6	Reset	Default
OFF	OFF	No Reset Generated	
OFF	ON	BOARD_RESET **without** LCA reconfiguration	
ON*	OFF	POWERUP_RESET **with** LCA reconfiguration	
ON*	ON	POWERUP_RESET **with** LCA reconfiguration	*

* When SW3-2 is set to ON, the setting of SW3-6 does not matter; in both cases, the reset occurs with FPGA reconfiguration.

2.4.4 Boot EPROM Device Selection

By default, the Boot EPROM consists of two flash memory devices which are located in sockets on the VME-10 board. Additionally, the user may install standard PROMs/EPROMs instead of the flash memory devices. These are standard CMOS EPROM devices, for example, 27C020 or 27C040 from Advanced Mircro Devices. The following table shows the necessary switch setting of SW4-1 for the two types of devices. Switch SW4-1 is located on the VME-10 board.

Table 9: Boot EPROM Device Selection

SW4-1	Boot EPROM Devices	Default
OFF	Flash Memory Devices	*
ON	Standard PROM/EPROMs	

2.4.5 Boot EPROM Write Protection

The boot EPROMs (flash memory devices) can be write protected via SW4-2 on the VME-10 board. The following table shows the switch setting of SW4-2 for write protection of the boot EPROMs.

Table 10: Boot EPROM Write Protection

SW4-2	Write Protection	Default
OFF	Yes	*
ON	No	

2.4.6 User EPROM Write Protection

The user EPROMs can be write protected via SW4-3 on the VME-10 board. The following table shows the switch setting of SW4-3 for the write protection of the user EPROMs.

Table 11: User EPROM Write Protection

SW4-3	Write Protection	Default
OFF	Yes	*
ON	No	

2.4.7 SCSI Interface on the CPU-10

The SCSI interface is located on the I/O-10 board, which occupies the first of the two VMEbus slots that SPARC CPU-10 uses. By default, all signals of the SCSI interface are routed to the VMEbus P2 connector. This connection is compatible to the CPU-2CE and the CPU-3CE.

NOTE: For an understanding of the usage of the SCSI interface, it is important to read **all** of the chapters here which describe the SCSI interface. Do not forget to check the following important factors: where the SCSI signals are located (for example, endpoint on SCSI bus), the configuration of SCSI termination networks # 1 and # 2, and the position of the SCSI switch matrices.

2.4.7.1 SCSI Termination Networks # 1 and # 2

There are two SCSI termination networks on the I/O-10 board of the SPARC CPU-10. **SCSI Termination Network # 1** is located near the VMEbus P2 connector and **SCSI Termination Network # 2** is located near the front panel SCSI connector. For the position of SCSI termination networks on the board, please see the figure "Location of Devices Used to Configure SCSI" on page 29.

The **SCSI Termination Network # 1** can be enabled and disabled via switch SW1-1 on the I/O-10 board. When SW1-1 is ON, the termination network # 1 is enabled. When SW1-1 is OFF, the termination network # 1 is disabled.

The following table shows the configurations of SCSI Termination Network # 1.

Table 12: SCSI Termination Network # 1

SW1-1	SCSI Termination Network # 1	Default
OFF	**Disabl**ed	*
ON	**Enabled**	

The **SCSI Termination Network # 2** is enabled and disabled automatically as described here. The termination network is enabled when there is no SCSI cable plugged into the front panel connector. The termination network is disabled when there is a SCSI cable plugged into the front panel connector.

The following table shows the configurations of SCSI Termination Network # 2.

Table 13: SCSI Termination Network # 2

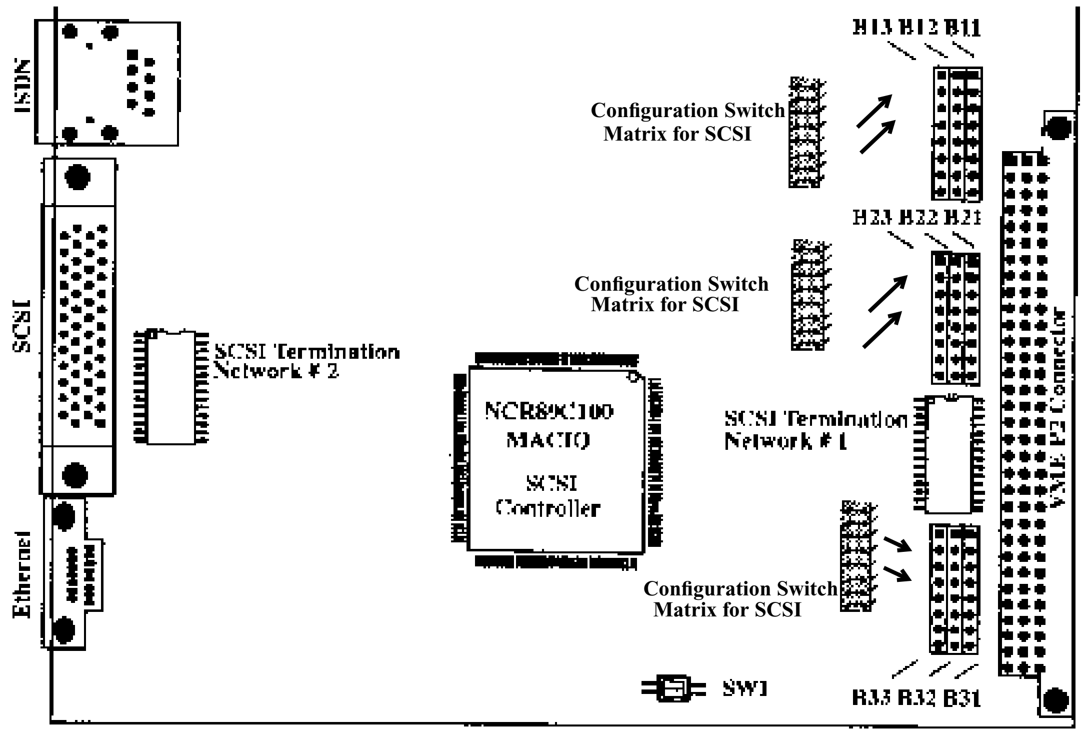
SCSI Cable Plugged into Front Panel Connector	SCSI Termination Network # 2	Default
No	**Enabled** (Automatically Enabled When No SCSI Cable Is Plugged Into Front Panel Connector)	*
Yes	**Disabled** (Automatically Disabled When A SCSI Cable Is Plugged Into Front Panel Connector)	

2.4.7.2 SCSI via Front Panel / VME P2 Connector

All signals of the SCSI interface are routed to the VMEbus P2 connector, which is pin compatible to the CPU-2CE and the CPU-3CE. There are several SCSI configurations possible through the use of three configuration switch matrices. In the table "SCSI Interface Configuration" on page 30, the different SCSI configurations are presented. It is important to check this table in order to correctly configure your SCSI.

The following figure shows the location of the 9 sockets which are used to configure the SCSI interface. Together with the three configuration switch matrices, the sockets used to configure the SCSI interface are B13, B12, B11 and B23, B22, B21 and B33, B32, B31.

FIGURE 9. Location of Devices Used to Configure SCSI


ISDN
SCSI
Ethernet
SCSI Termination Network # 2
NCR89C100 MACIO SCSI Controller
Configuration Switch Matrix for SCSI
Configuration Switch Matrix for SCSI
Configuration Switch Matrix for SCSI
B13 B12 B11
B23 B22 B21
B33 B32 B31
SCSI Termination Network # 1
VME P2 Connector
SW1

2.4.7.3 Possible SCSI Configurations

The SCSI interface may be configured in the following ways.

- SCSI via the VMEbus P2 connector at an endpoint of the SCSI bus (default configuration).
- SCSI via the VMEbus P2 connector but not at an endpoint of the SCSI bus.
- SCSI via the front panel connector at an endpoint of the SCSI bus.
- SCSI via the VMEbus P2 connector and via the front panel connector.
- SCSI not used.

The following table shows the different SCSI configurations, the respective positions of the switch matrices, and the settings of the termination networks #1 and #2. The table also shows the settings when the SCSI interface is not used.

Table 14: SCSI Interface Configuration

Default	SCSI on VMEbus P2	SCSI on Front Panel	Switch Matrices are plugged into the following sockets	SCSI Termination #1 (Via Switch SW1-1)	SCSI Termination #2 (Automatically disabled or enabled)
*	Yes (endpoint on SCSI bus)	No	B12 and B11 B22 and B21 B32 and B31	Disabled (SW1-1 = OFF)	Enabled (No cable plugged into front panel connector)
	Yes (not endpoint on SCSI bus)	No	B13 and B12 B23 and B22 B33 and B32	Disabled (SW1-1 = OFF)	Enabled (No cable plugged into front panel connector). NOTE: In this case, SCSI termination # 2 is not connected with the SCSI interface on VMEbus P2.
	No	Yes (endpoint on SCSI bus)	B12 and B11 B22 and B21 B32 and B31	Enabled (SW1-1 = ON)	Disabled (Cable plugged into front panel connector)
	Yes (not endpoint) on SCSI bus)	Yes (not endpoint on SCSI bus)	B12 and B11 B22 and B21 B32 and B31	Disabled (SW1-1 = OFF)	Disabled (Cable plugged into front panel connector)
	No	No	B12 and B11 B22 and B21 B32 and B31	Enabled (SW1-1 = ON)	Enabled (No cable plugged into front panel connector)

2.4.7.4 Ethernet via Front Panel / VME P2 Connector

The Ethernet interface is located on the I/O-10 board, which occupies the first of the two VMEbus slots that SPARC CPU-10 uses. The default configuration provides the Ethernet on the front panel connector. Using an 8-pin configuration switch matrix, the default configuration may be changed so that the Ethernet interface is available via the VME P2 connector. This is done by changing the position of the configuration switch matrix as shown in the table below.

FIGURE 10. Location of Devices Used to Configure Ethernet

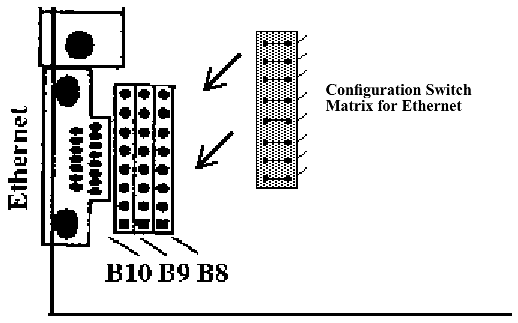


Table 15: Ethernet Configuration

Switch Matrix is plugged into the Following Sockets	Ethernet Signals Via VME P2 Connector	Ethernet Signals Via Front Panel Connector	Default
B9 and B8	Yes	No	
B10 and B9	No	Yes	*

WARNING

When the Ethernet interface is configured via VME P2, do not connect the Ethernet cable at the front panel connector.

2.4.8 Reserved and Unused Switches

Please note that on the VME-10 board, SW4-4 and SW4-5 are unused, and SW4-6 is reserved for test purposes. Switches SW4-4 and SW4-5 should always be OFF and SW4-6 should always be ON.

Please also note that on the I/O-10 board, SW1-2 is reserved for test purposes and should always be OFF.

2.5 OpenBoot Firmware

This chapter describes the use of OpenBoot firmware. Specifically, you will read how to perform the following tasks.

- Boot the System
- Run Diagnostics
- Display System Information
- Reset the System
- OpenBoot Help

For more detailed information about OpenBoot firmware, please see the *OPEN BOOT PROM 2.0 MANUAL SET* .

2.5.1 Boot the System

The most important function of OpenBoot firmware is booting the system. Booting is the process of loading and executing a stand-alone program such as the operating system. After it is powered on, the system usually boots automatically after it has passed the Power On Self Test (POST). This occurs without user intervention.

If necessary, you can explicitly initiate the boot process from the OpenBoot command interpreter. Automatic booting uses the default boot device specified in nonvolatile RAM (NVRAM); user initiated booting uses either the default boot device or one specified by the user.

To boot the system from the default boot device, type the following command at the Forth monitor prompt.

```
ok boot
```

or, if you are at the Restricted monitor prompt, you have to type the following:

```
> b
```

The boot command has the following format:

```
boot [device-specifier] [filename] [-ah]
```

The optional parameters are described as follows.

[device-specifier]	The name (full path or alias) of the boot device. Typical values are cdrom, disk, floppy, net or tape.
[filename]	The name of the program to be booted. *filename* is relative to the root of the selected device. If no filename is specified, the boot command uses the value of *boot-file* NVRAM parameter. The NVRAM parameters used for booting are described in the following chapter.
[-a]	-a prompt interactively for the device and name of the boot file.
[-h]	-h halt after loading the program.

NOTE: These options are specific to the operating system and may differ from system to system.

To explicitly boot from the internal disk, type:

```
ok boot disk
```

or at the Restricted Monitor prompt:

```
> b disk
```

To retrieve a list of all device alias definitions, type *devalias* at the Forth Monitor command prompt. The following table lists some typical device aliases:

Table 16: Device Alias Definitions

Alias	Boot Path	Description
disk	/iommu/sbus/espdma/esp/sd@3,0	Default disk (1st internal) SCSI-ID 3
disk3	/iommu/sbus/espdma/esp/sd@3,0	First internal disk SCSI-ID 3
disk2	/iommu/sbus/espdma/esp/sd@2,0	Additional internal disk SCSI-ID 2
disk1	/iommu/sbus/espdma/esp/sd@1,0	External disk SCSI-ID 1
disk0	/iommu/sbus/espdma/esp/sd@0,0	External disk SCSI-ID 0
tape	/iommu/sbus/espdma/esp/st@4,0	First tape drive SCSI-ID 4
tape0	/iommu/sbus/espdma/esp/st@4,0	First tape drive SCSI-ID 4
tape1	/iommu/sbus/espdma/esp/st@5,0	Second tape drive SCSI-ID 5
cdrom	/iommu/sbus/espdma/esp/sd@6,0:d	CD-ROM partition d, SCSI-ID 6
net	/iommu/sbus/ledma/le	Ethernet
floppy	/obio/SUNW,fdtwo	Floppy drive

2.5.2 NVRAM Boot Parameters

The OpenBoot firmware holds configuration parameters in NVRAM. At the Forth Monitor prompt, type *printenv* to see a list of all available configuration parameters. The OpenBoot command *setenv* may be used to set these parameters.

setenv [configuration parameter] [value]

This information refers only to those configuration parameters which are involved in the boot process. The following table lists these parameters.

Table 17: Setting Configuration Parameters

Parameter	Default Value	Description
auto-boot?	true	If true, boot automatically after power on or reset
boot-device	disk	Device from which to boot
boot-file	empty string	File to boot
diag-switch?	false	If true, run in diagnostic mode
diag-device	net	Device from which to boot in diagnostic mode
diag-file	empty string	File to boot in diagnostic mode

When booting an operating system or another stand-alone program, and neither a boot device nor a filename is supplied, the boot command of the Forth Monitor takes the omitted values form the NVRAM configuration parameters. If the parameter diag-switch? is false, boot-device and boot-file are used. Otherwise, the OpenBoot firmware uses diag-device and diag-file for booting.

For a detailed description of all NVRAM configuration parameters, please refer to the *OPEN BOOT PROM 2.0 MANUAL SET.*

2.5.3 Diagnostics

At power on or after reset, the OpenBoot firmware executes POST. If the NVRAM configuration parameter diag-switch? is true for each test, a message is displayed on a terminal connected to the first serial port. In case the system is not working correctly, error messages indicating the problem are displayed. After POST, the OpenBoot firmware boots an operating system or enters the Forth Monitor if the NVRAM configuration parameter auto-boot? is false.

The Forth Monitor includes several diagnostic routines. These on-board tests let you check devices such as network controller, SCSI devices, floppy disk system, memory, clock and installed SBus cards. User installed devices can be tested if their firmware includes a selftest routine.

The table below lists several diagnostic routines.

Table 18: Diagnostic Routines

Command	Description
probe-scsi	Identify devices connected to the on-board SCSI bus
probe-scsi-all [*device-path*]	Perform probe-scsi on all SCSI buses installed in the system below the specified device tree node. (If *device-path* is omitted, the root node is used.)
test *device-specifier*	Execute the specified device's selftest method. device-specifier may be a device path name or a device alias. For example: test net - test network connection test /memory - test number of megabytes specified in the selftest-#megs NVRAM parameter or test all of memory if diag-switch? is true
test-all [*device-specifier*]	Test all devices (that have a built-in selftest method) below the specified device tree node. (If *device-path* is omitted, the root node is used.)
watch-clock	Monitor the clock function
watch-net	Monitor network connection

To check the on-board SCSI bus for connected devices, type:

```
ok probe-scsi
Target 3
        Unit 0 Disk superP 1684-07MB1036511AS0C1684
ok
```

To test all the SCSI buses installed in the system, type

```
ok probe-scsi-all
/iommu@0,10000000/sbus@0,10001000/esp@2,100000
Target 6
        Unit 0 Disk Removable Read Only Device SONY CD-ROM CDU-8012 3.1a

/iommu@0,10000000/sbus@0,10001000/espdma@4,8400000/esp@4,8800000
Target 3
        Unit 0 Disk superP 1684-07MB1036511AS0C1684

ok
```

The actual response depends on the devices on the SCSI buses.

To test a single installed device, type:

```
ok test device-specifier
```

This executes the device method name selftest of the specified device node. device-specifier may be a device path name or a device alias as described in Table 16, “Device Alias Definitions,” on page 35.The response depends on the selftest of the device node.

To test a group of installed devices, type:

```
ok test-all
```

All devices below the root node of the device tree are tested. The response depends on the devices that have a selftest routine. If a device specifier option is supplied at the command line, all devices below the specified device tree node are tested.

When you use the memory testing routine, the system tests the number of megabytes of memory specified in the NVRAM configuration parameter selftest-#megs. If the NVRAM configuration parameter diag-switch? is true, all memory is tested.

```
ok test /memory
testing 32 megs of memory at addr 0 27
ok
```

The command **test-memory** is equivalent to **test /memory**. In the example above, the first number (0) is the base address of the memory bank to be tested, the second number (27) is the number of megabytes remaining. If the CPU board is working correctly, the memory is erased and tested and you will receive the **ok** prompt. If the PROM or the on-board memory is not working, you receive one of a number of possible error messages indicating the problem.

To test the clock function, type

```
ok watch-clock
Watching the ‘seconds’ register of the real time clock chip.
It should be ‘ticking’ once a second.
Type any key to stop.
22
ok
```

The system responds by incrementing a number once a second. Press any key to stop the test.

To monitor the network connection, type:

```
ok watch-net
Using AUI Ethernet Interface
Lance register test -- succeeded.
Internal loopback test -- succeeded.
External loopback test -- succeeded.
Looking for Ethernet packets.
‘.’ is a good packet. ‘X’ is a bad packet.
Type any key to stop.
...........X...........................X..............
ok
```

The system monitors the network traffic, displaying “.” each time it receives a valid packet and displaying “X” each time it receives a packet with an error that can be detected by the network hardware interface.

2.5.4 Display System Information

The Forth Monitor provides several commands to display system information. These commands let you display the system banner, the Ethernet address for the Ethernet controller, the contents of the ID PROM, and the version number of the OpenBoot firmware.

The ID PROM contains information specific to each individual machine, including the serial number, date of manufacture, and assigned Ethernet address.

The following table lists these commands.

Table 19: Commands to Display System Information

Command	Description
banner	Display system banner
show-sbus	Display list of installed and probed SBus devices
.enet-addr	Display current Ethernet address
.idprom	Display ID PROM contents, formatted
.traps	Display a list of SPARC trap types
.version	Display version and date of the boot PROM
show-devs	Display a list of all device tree nodes
devalias	Display a list of all device aliases

2.5.5 Reset the System

If your system needs to be reset, you either press the reset button on the front panel or, if you are in the Forth Monitor, type **reset** on the command line.

```
ok reset
```

The system immediately begins executing the Power On Self Test (POST) and initialization procedures. Once the POST completes, the system either boots automatically or enters the Forth Monitor, just as it would have done after a power on cycle.

2.5.6 OpenBoot Help

The Forth Monitor contains an online help. To get this, type:

```
ok help
Enter ‘help command-name’ or ‘help category-name’ for more help
(Use ONLY the first word of a category description)
Examples: help select -or- help line
Main categories are:
File download and boot
Resume execution
Diag (diagnostic routines)
Power on reset
>-prompt
Floppy eject
Select I/O devices
Ethernet
System and boot configuration parameters
Line editor
Tools (memory,numbers,new commands,loops)
Assembly debugging (breakpoints,registers,disassembly,symbolic)
Sync (synchronize disk data)
Nvramrc (making new commands permanent)
ok
```

A list of all available help categories is displayed. These categories may also contain subcategories. To get help for special forth words or subcategories just type help [name]. An example is shown on the next page.

An example of how to get help for special forth words or subcategories.

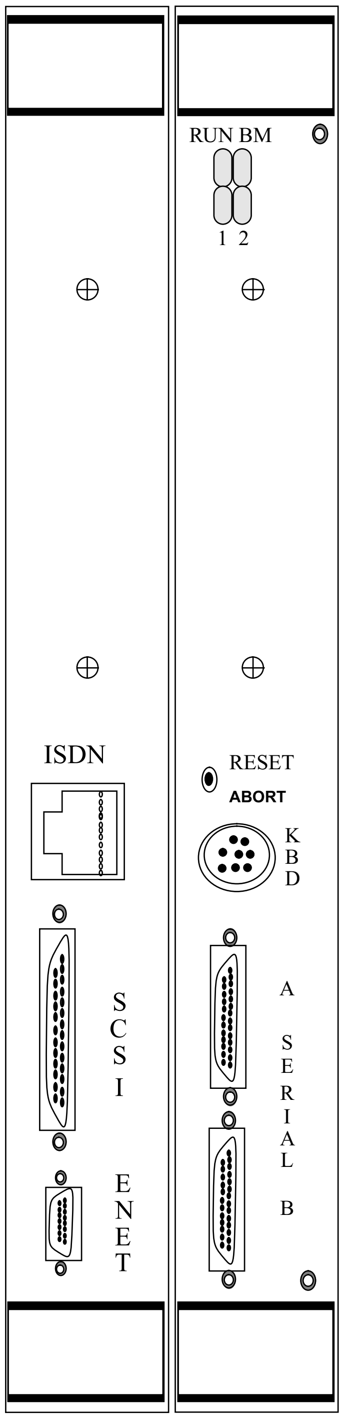
```
ok help tools
 Category: Tools (memory,numbers,new commands,loops)
 Subcategories are:
Memory access
Arithmetic
Radix (number base conversions)
Numeric output
Defining new commands
Repeated loops
ok
ok help memory
 Category: Memory access
dump ( addr length -- ) display memory at addr for length bytes
fill ( addr length byte -- ) fill memory starting at addr with byte
move ( src dest length -- ) copy length bytes from src to dest address
map? ( vaddr -- ) show memory map information for the virtual address
l? ( addr -- ) display the 32-bit number from location addr
w? ( addr -- ) display the 16-bit number from location addr
c? ( addr -- ) display the 8-bit number from location addr
l@ ( addr -- n ) place on the stack the 32-bit data at location addr
w@ ( addr -- n ) place on the stack the 16-bit data at location addr
c@ ( addr -- n ) place on the stack the 8-bit data at location addr
l! ( n addr -- ) store the 32-bit value n at location addr
w! ( n addr -- ) store the 16-bit value n at location addr
c! ( n addr -- ) store the 8-bit value n at location addr
ok
```

The online help shows you the forth word, the parameter stack before and after execution of the forth word (before -- after), and a short description.

The online help of the Forth Monitor is located in the boot PROM, so there is not an online help for all forth words.

2.6 Front Panel

FIGURE 11. Diagram of the Front Panel



2.6.1 Features of the Front Panel

- Reset / Abort Key
- Status LEDs
- Keyboard / Mouse Connector
- Serial A Connector
- Serial B Connector
- ISDN Connector
- SCSI Connector
- Ethernet Connector

These features are described in detail in Section 3 of the *SPARC CPU-10 Technical Reference Manual*.

Table 20: Features of the Front Panel

Device	Function	Front Panel Name

Switch	Reset / Abort
	UP=Reset
	DOWN=Abort
		RESET
		
		ABORT

LED	RUN/RESET	RUN
LED	VMEbus Bus Master	BM
LED	User LED 1	1
LED	User LED 2	2
MiniDin Connector	Keyboard/Mouse	KBD
Serial Connector	Serial Interface A	A SERIAL
Serial Connector	Serial Interface B	B
ISDN Connector	ISDN Interface	ISDN
SCSI Connector	SCSI Interface	SCSI
Micro D-Sub Connector	Ethernet Interface	ENET

2.7 SPARC CPU-10 Connectors

The connectors on the SPARC CPU-10 are listed in the following table.

Table 21: SPARC CPU-10 Connectors

Function	Location	Connector Type	Manufacturer Part Number
Keyboard/Mouse	Front Panel	8-Pin Mini DIN	AMP 749232-1
Serial Port A	Front Panel	26-Pin Fine Pitch	AMP 749831-2
Serial Port B	Front Panel	26-Pin Fine Pitch	AMP 749831-2
ISDN	Front Panel	RJ-45	AMP 555131-1
SCSI	Front Panel	50-Pin Fine Pitch	AMP 749831-5
Ethernet	Front Panel	15-Pin Micro DSub	ITT CANNON MDSM15PE-Z10
SBus Slot 0	P3	96-Pin SMD	FUJITSU FCN-234J096-G/V
SBus Slot 1	P4	96-Pin SMD	FUJITSU FCN-234J096-G/V
VMEbus P1	P1 on VME-10 and I/O-10	96-Pin VG	ITT CANNON G60M096P3BEBM3D
VMEbus P2	P2 on VME-10 and I/O-10	96-Pin VG	ITT CANNON G60M096P3BEBM3D

The following pages show the pinouts for the connectors.

2.7.1 Serial Ports A and B Connector Pinout

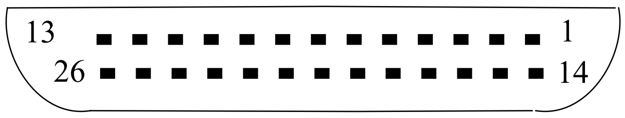
The two serial I/O ports are available on the front panel via two 26-pin shielded connectors which are compatible to the CPU-2CE and CPU-3CE.

Both channels are available via the VMEbus P2 connector of the VME-10, each with four signals (RXD, TXD, RTS, CTS). Each of the two serial I/O ports are independent full-duplex channels. The table below shows the pinout of both of the 26-pin shielded connectors. The table is valid for both serial I/O connectors A and B.

Table 22: Serial Ports A and B Connector Pinout for RS-232

Pin	Transmitted Signals	Pin	Received Signals
2	TxD-Transmit Data	3	RxD-Receive Data
4	RTS-Request To Send	5	CTS-Clear To Send
7	GND	6	SYNC
20	DTR-Data Terminal Ready	8	DCD-Data Carrier Detect
24	TRXC-DTE Terminal Clock	15	TRXC-DCE Terminal Clock
		17	RTXC-DCE Terminal Clock

FIGURE 12. Pinout of Serial Ports A and B Connectors



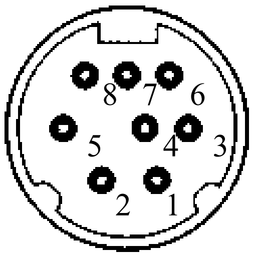
2.7.2 Keyboard/Mouse Connector Pinout

The keyboard and mouse port is available on the front panel via a mini DIN connector.

Table 23: Keyboard/Mouse Connector Pinout

Pin	Function
1	GND
2	GND
3	+5VDC
4	Mouse In
5	Keyboard Out
6	Keyboard In
7	Mouse Out
8	+5VDC

FIGURE 13. Pinout of Keyboard/Mouse Connector



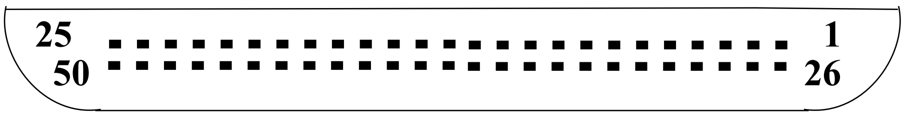
2.7.3 SCSI Connector Pinout

The following table is a pinout of the SCSI connector. The figure on the next page shows the SCSI connector and location of the pin numbers.

Table 24: SCSI Connector Pinout

Pin No.	Signal	Pin No.	Signal
1	GND	26	SCSI Data 0
2	GND	27	SCSI Data 1
3	GND	28	SCSI Data 2
4	GND	29	SCSI Data 3
5	GND	30	SCSI Data 4
6	GND	31	SCSI Data 5
7	GND	32	SCSI Data 6
8	GND	33	SCSI Data 7
9	GND	34	SCSI DP
10	GND	35	GND
11	TERMDIS	36	GND
12	N.C.	37	N.C.
13	N.C.	38	TERMPWR
14	N.C.	39	N.C.
15	GND	40	GND
16	GND	41	SCSI ATN
17	GND	42	GND
18	GND	43	SCSI BSY
19	GND	44	SCSI ACK
20	GND	45	SCSI RST
21	GND	46	SCSI MSG
22	GND	47	SCSI SEL
23	GND	48	SCSI CD
24	GND	49	SCSI REQ
25	GND	50	SCSI IO

FIGURE 14. Pinout of SCSI Connector



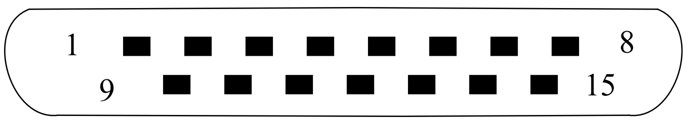
2.7.4 Ethernet Connector Pinout

The following table is a pinout of the Ethernet connector. The figure below shows the Ethernet connector and pin numbers.

Table 25: Ethernet Connector Pinout

Pin	Function
1	GND
2	Collision+
3	Transmit Data+
4	GND
5	Receive Data+
6	GND
7	N.C.
8	N.C.
9	Collision-
10	Transmit Data-
11	GND
12	Receive Data-
13	+12VDC
14	GND
15	N.C.

FIGURE 15. Pinout of Ethernet Connector



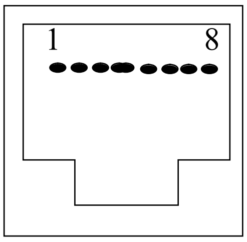
2.7.5 ISDN Connector Pinout

The following table is a pinout of the ISDN connector. The figure below shows the ISDN connector and pin numbers. The IDSN interface is implemented for use in future applications.

Table 26: ISDN Connector Pinout

Pin	Function
1	NT IN +
2	NT OUT +
3	TE OUT +
4	TE IN +
5	TE IN -
6	TE OUT -
7	NT OUT -
8	NT IN -

FIGURE 16. Pinout of ISDN Connector



WARNING

The ISDN interface must be used together with the ISDN-10 Adapter Set purchased from FORCE COMPUTERS. Do not plug in an ISDN compatible cable into the front panel connector; this can cause damage to your system.

2.7.6 VME P2 Connector Pinout of the VME-10

The following is a pinout of the VME P2 connector of the VME-10. The table is continued on the next page.

Table 27: VME P2 Connector Pinout of the VME-10

ROWA	Signal	ROW B	Signal	ROW C	Signal
1	N.C.	1	+5VDC	1	FPY DENSEL
2	N.C.	2	GND	2	FPY DENSENSE
3	N.C.	3	RESERVED	3	FPY DRVSEL3
4	N.C.	4	VME A24	4	FPY INDEX
5	N.C.	5	VME A25	5	FPY DRVSEL0
6	N.C.	6	VME A26	6	FPY DRVSEL1
7	N.C.	7	VME A27	7	FPY DRVSEL2
8	N.C.	8	VME A28	8	FPY MOTEN
9	N.C.	9	VME A29	9	FPY DIR
10	N.C.	10	VME A30	10	FPY STEP
11	N.C.	11	VME A31	11	FPY WRDATA
12	N.C.	12	GND	12	FPY WRGATE
13	N.C.	13	+5VDC	13	FPY TRACK0
14	N.C.	14	VME D16	14	FPY WRPROT
15	N.C.	15	VME D17	15	FPY RDDATA
16	N.C.	16	VME D18	16	FPY HEADSEL
17	N.C.	17	VME D19	17	FPY DISKCHG
18	N.C.	18	VME D20	18	FPY EJECT
19	N.C.	19	VME D21	19	N.C.
20	N.C.	20	VME D22	20	GND
21	N.C.	21	VME D23	21	GND
22	N.C.	22	GND	22	N.C.
23	N.C.	23	VME D24	23	N.C.
24	POWEROFF	24	VME D25	24	N.C.
25	MOUSEOUT	25	VME D26	25	N.C.
26	MOUSEIN	26	VME D27	26	N.C.
27	KBDOUT	27	VME D28	27	N.C.

Table 27: VME P2 Connector Pinout of the VME-10 (Continued)

ROWA	Signal	ROW B	Signal	ROW C	Signal
28	KBDIN	28	VME D29	28	N.C.
29	TxD Port A	29	VME D30	29	TxD Port B
30	RxD Port A	30	VME D31	30	RxD Port B
31	RTS Port A	31	GND	31	RTS Port B
32	CTS Port A	32	+5VDC	32	CTS Port B

2.7.7 VME P2 Connector Pinout of the I/O-10

The following is the pinout of the VME P2 connector of the I/O-10. The table is continued on the next page.

Table 28: VME P2 Connector Pinout of the I/O-10

ROW A	Signal	ROW B	Signal	ROW C	Signal
1	SCSI Data 0	1	+5VDC	1	CENTR DS
2	SCSI Data 1	2	GND	2	CENTR Data 0
3	SCSI Data 2	3	N.C.	3	CENTR Data 1
4	SCSI Data 3	4	N.C.	4	CENTR Data 2
5	SCSI Data 4	5	N.C.	5	CENTR Data 3
6	SCSI Data 5	6	N.C.	6	CENTR Data 4
7	SCSI Data 6	7	N.C.	7	CENTR Data 5
8	SCSI Data 7	8	N.C.	8	CENTR Data 6
9	SCSI DP	9	N.C.	9	CENTR Data 7
10	GND	10	N.C.	10	CENTR ACK
11	GND	11	N.C.	11	CENTR BSY
12	GND	12	GND	12	CENTR PE
13	TERMPWR	13	+5VDC	13	CENTR AF
14	GND	14	N.C.	14	CENTR INIT
15	GND	15	N.C.	15	CENTR ERR
16	SCSI ATN	16	N.C.	16	CENTR SLCT IN
17	GND	17	N.C.	17	CENTR SLCT
18	SCSI BSY	18	N.C.	18	N.C.
19	SCSI ACK	19	N.C.	19	+12VDC
20	SCSI RST	20	N.C.	20	GND
21	SCSI MSG	21	N.C.	21	GND
22	SCSI SEL	22	GND	22	ETH REC+
23	SCSI CD	23	N.C.	23	ETH REC-
24	SCSI REQ	24	N.C.	24	ETH TRA+
25	SCSI IO	25	N.C.	25	ETH TRA-

Table 28: VME P2 Connector Pinout of the I/O-10 (Continued)

ROW A	Signal	ROW B	Signal	ROW C	Signal
26	N.C.	26	N.C.	26	ETH COL+
27	AUD DX	27	N.C.	27	ETH COL-
28	AUD DR	28	N.C.	28	AUD EPDOWN
29	AUD CLK	29	N.C.	29	AUD RESET
30	AUD FS	30	N.C.	30	AUD EMCTL
31	AUD DFSYNC	31	GND	31	AUD DC
32	AUD VCC	32	+5VDC	32	AUD GND

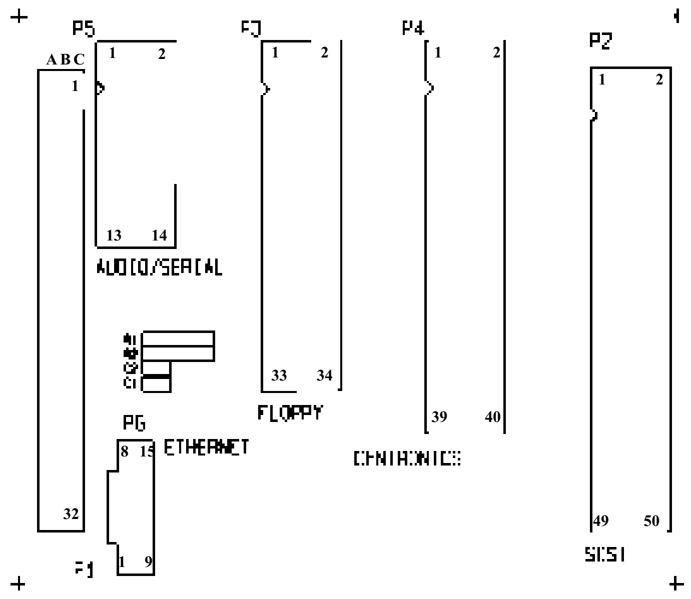
2.7.8 The IOBP-10 Connectors

The IOBP-10 is an I/O back panel on VMEbus P2 with flat cable connectors for SCSI, Serial/ Audio interface, Centronics compliant interface and Floppy interface, and the IOBP-10 also has a micro D-Sub connector for an Ethernet interface.
Two of the IOBP-10 back panels can be used together with the CPU-10. One back panel can be plugged into the VME P2 connector of the I/O-10 board (first VME slot), the other one can be plugged into the VME P2 connector of the VME-10 board (second VME slot). On the back panel in the first VME slot, the SCSI, Ethernet, Centronics and Audio interface are available. On the back panel in the second VME slot, the Floppy, Serial and Keyboard/Mouse interface are available.

Note that the Audio interface and the Serial interface use the same flat cable connector on the IOBP-10 back panel.

The diagram below shows all the connectors.

FIGURE 17. The IOBP-10



The pinouts of the connectors (P1) ... (P6) are shown in the following tables.

CAUTION

This IOBP-10 back panel is especially designed for the SPARC CPU-10. Do not use any other I/O back panels on the SPARC CPU-10, for example, the IOBP-1.

Table 29: IOBP-10 P1 Pinout

ROW A	Signal when used on VME-10	Signal when used on I/O-10	ROW C	Signal when used on VME-10	Signal when used on I/O-10
1	N.C.	SCSI Data 0	1	FPY DENSEL	CENTR DS
2	N.C.	SCSI Data 1	2	FPY DENSENSE	CENTR Data 0
3	N.C.	SCSI Data 2	3	FPY DRVSEL3	CENTR Data 1
4	N.C.	SCSI Data 3	4	FPY INDEX	CENTR Data 2
5	N.C.	SCSI Data 4	5	FPY DRVSEL0	CENTR Data 3
6	N.C.	SCSI Data 5	6	FPY DRVSEL1	CENTR Data 4
7	N.C.	SCSI Data 6	7	FPY DRVSEL2	CENTR Data 5
8	N.C.	SCSI Data 7	8	FPY MOTEN	CENTR Data 6
9	N.C.	SCSI DP	9	FPY DIR	CENTR Data 7
10	N.C.	GND	10	FPY STEP	CENTR ACK
11	N.C.	GND	11	FPY WRDATA	CENTR BSY
12	N.C.	GND	12	FPY WRGATE	CENTR PE
13	N.C.	TERMPWR	13	FPY TRACK0	CENTR AF
14	N.C.	GND	14	FPY WRPROT	CENTR INIT
15	N.C.	GND	15	FPY RDDATA	CENTR ERR
16	N.C.	SCSI ATN	16	FPY HEADSEL	CENTR SLCT IN
17	N.C.	GND	17	FPY DISKCHG	CENTR SLCT
18	N.C.	SCSI BSY	18	FPY EJECT	N.C.
19	N.C.	SCSI ACK	19	N.C.	+12VDC
20	N.C.	SCSI RST	20	GND	GND
21	N.C.	SCSI MSG	21	GND	GND
22	N.C.	SCSI SEL	22	N.C.	ETH REC+
23	N.C.	SCSI CD	23	N.C.	ETH REC-
24	POWEROFF	SCSI REQ	24	N.C.	ETH TRA+
25	MOUSEOUT	SCSI IO	25	N.C.	ETH TRA-
26	MOUSEIN	N.C.	26	N.C.	ETH COL+
27	KBDOUT	AUD DX	27	N.C.	ETH COL-
28	KBDIN	AUD DR	28	N.C.	AUD EPDOWN
29	TxD Port A	AUD CLK	29	TxD Port B	AUD RESET

Table 29: IOBP-10 P1 Pinout (Continued)

ROW A	Signal when used on VME-10	Signal when used on I/O-10	ROW C	Signal when used on VME-10	Signal when used on I/O-10
30	RxD Port A	AUD FS	30	RxD Port B	AUD EMCTL
31	RTS Port A	AUD DFSYNC	31	RTS Port B	AUD DC
32	CTS Port A	AUD VCC	32	CTS Port B	AUD GND

Table 30: IOBP-10 P2 Pinout (SCSI)

Pin No.	Signal	Pin No.	Signal
1	GND	2	SCSI Data 0
3	GND	4	SCSI Data 1
5	GND	6	SCSI Data 2
7	GND	8	SCSI Data 3
9	GND	10	SCSI Data 4
11	GND	12	SCSI Data 5
13	GND	14	SCSI Data 6
15	GND	16	SCSI Data 7
17	GND	18	SCSI DP
19	GND	20	GND
21	GND	22	GND
23	GND	24	GND
25	N.C.	26	TERMPWR
27	GND	28	GND
29	GND	30	GND
31	GND	32	SCSI ATN
33	GND	34	GND
35	GND	36	SCSI BSY
37	GND	38	SCSI ACK
39	GND	40	SCSI RST
41	GND	42	SCSI MSG
43	GND	44	SCSI SEL
45	GND	46	SCSI CD
47	GND	48	SCSI REQ
49	GND	50	SCSI IO

Table 31: IOBP-10 P3 (Floppy)

Pin No.	Signal	Pin No.	Signal
1	FPY EJECT	2	FPY DENSEL
3	GND	4	FPY DENSENSE
5	GND	6	FPY DRVSEL3
7	GND	8	FPY INDEX
9	GND	10	FPY DRVSEL0
11	GND	12	FPY DRVSEL1
13	GND	14	FPY DRVSEL2
15	GND	16	FPY MOTEN
17	GND	18	FPY DIR
19	GND	20	FPY STEP
21	GND	22	FPY WRDATA
23	GND	24	FPY WRGATE
25	GND	26	FPY TRACK0
27	GND	28	FPY WRPROT
29	GND	30	FPY RDDATA
31	GND	32	FPY HEADSEL
33	GND	34	FPY DISKCHG

Table 32: IOBP-10 P4 Pinout (Centronics)

Pin No.	Signal	Pin No.	Signal
1	CENTR DS	2	GND
3	CENTR Data 0	4	GND
5	CENTR Data 1	6	GND
7	CENTR Data 2	8	GND
9	CENTR Data 3	10	GND
11	CENTR Data 4	12	GND
13	CENTR Data 5	14	GND
15	CENTR Data 6	16	GND
17	CENTR Data 7	18	GND
19	CENTR ACK	20	GND
21	CENTR BSY	22	GND
23	CENTR PE	24	GND
25	CENTR SLCT	26	CENTR INIT
27	CENTR AF	28	CENTR ERR
29	N.C.	30	GND
31	GND	32	N.C.
33	N.C.	34	N.C.
35	N.C.	36	CENTR SLCT IN
37	N.C.	38	N.C.
39	N.C.	40	N.C.

Table 33: IOBP-10 P5 Pinout (Serial / Audio)

Pin No.	Signal when used on VME-10	Signal when used on I/O-10	Pin No.	Signal when used on VME-10	Signal when used on I/O-10
1	MOUSEOUT	AUD EPDOWN	2	KBDOUT	AUD DX
3	MOUSEIN	N.C.	4	KBDIN	AUD DR
5	TxD Port B	AUD RESET	6	TxD Port A	AUD CLK
7	RxD Port B	AUD EMCTL	8	RxD Port A	AUD FS
9	RTS Port B	AUD DC	10	RTS Port A	AUD DFSYNC
11	CTS Port B	AUD GND	12	CTS Port A	AUD VCC
13	GND	GND	14	GND	GND

Table 34: IOBP-10 P6 Pinout (Ethernet)

Pin	Function
1	GND
2	Collision+
3	Transmit Data+
4	GND
5	Receive Data+
6	GND
7	N.C.
8	N.C.
9	Collision-
10	Transmit Data-
11	GND
12	Receive Data-
13	+12VDC
14	GND
15	N.C.

SECTION 3 Hardware Description

3. Overview of the SPARC CPU-10

This section of the *SPARC CPU-10 Technical Reference Manual* provides a description of the CPU-10 from the hardware perspective. Please take a moment to examine the table of contents in order to see how this section is structured. This will be of value to you when looking for information in the future.

The block diagram below gives a functional overview of the SPARC CPU-10 board.

FIGURE 18. Block Diagram of the SPARC CPU-10

3.1 The Four Buses on the SPARC CPU-10

As is shown in the block diagram on the previous page, there are four separate buses on the SPARC CPU-10: the Mbus and the SBus, the Ebus and the VMEbus.

The **Mbus** is the 64-bit wide processor bus which connects the Mbus slot, the memory controller L64860 (EMC), the VME interface MB86986 (MVIC) and the Mbus-SBus interface L64862 (MSI). The Mbus slot can hold a SuperSPARC Mbus module, which is a standard product. SuperSPARC Mbus modules can hold up to two processors, are available with or without external cache, and in several frequency variants. The MVIC provides a complete 32-bit IEEE-1014 VMEbus interface. The EMC is the error correcting memory controller which interfaces the Mbus directly to the 128-bit wide DRAM. The SPARC CPU-10 is available with 32, 64 or 128 Mbytes of DRAM. The MSI is the interface chip between the Mbus and the SBus

The **SBus** is the 32-bit wide I/O bus which connects the MSI, the two I/O interface chips L64861 (SEC) and NCR89C100 (MACIO chip), the ISDN/audio interface T7259 (DBRI) and the two SBus slots for standard SBus modules. The SEC provides the interface between the SBus and the 8-bit local I/O bus (Ebus). The MACIO chip provides the interface between the SBus and the SCSI interface, the Ethernet interface, and the Centronics compliant parallel port. The DBRI provides an ISDN interface (CCITT 1.430 and ANSI T1.605 standard) and a 16-bit CD-quality audio port.

The **Ebus** is an 8-bit local I/O bus which connects the floppy disk interface, two serial I/O ports, the keyboard/mouse interface, the boot EPROM, the user EPROM, the RTC/NVRAM, and an additional FPGA (LCA) for general board control.

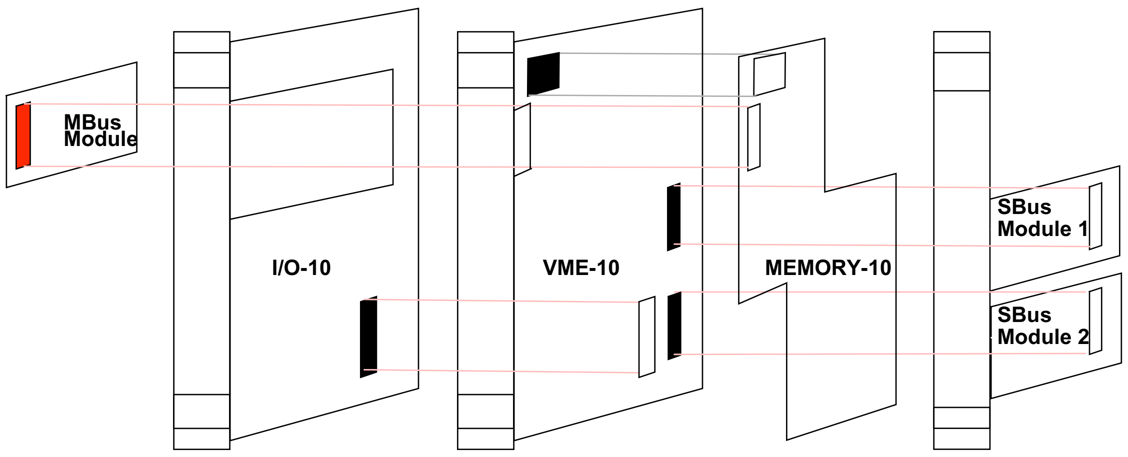
The **VMEbus** is the 32-bit backplane bus and is connected to the MVIC, which is the interface between the Mbus and the VMEbus.

3.1.1 Mechanical Construction of the SPARC CPU-10

It is useful to have an understanding of the mechanical construction of the CPU-10 since there are components on the board which can be modified by the user. These components include DIP switches, configuration switch matrices, EPROMS and an RTC/NVRAM. Highlighted diagrams showing the components of the VME-10, the I/O-10 and the MEMORY-10 appear at the beginning of the Installation section of this manual. By looking at these diagrams, you can identify the important components on the board such as the switches, EPROMs, etc.

The figure on the next page shows the mechanical construction of the SPARC CPU-10 together with a description of the board modules.

FIGURE 19. Mechanical Construction of the SPARC CPU-10



The SPARC CPU-10 is a two slot VMEbus computer in standard 6U VMEbus form factor. It consists of up to six board modules. The electrical connection of the six board modules is realized with the help of special module connectors. The mechanical connection is realized with several standoffs, mounting blocks and stiffeners to ensure maximum mechanical stability and ruggedness of the SPARC CPU-10.

3.1.2 Board Components

The six board modules which make up the SPARC CPU-10 are as follows.

- The SuperSPARC Mbus Module
- The I/O-10 Board
- The VME-10 Board
- The MEMORY-10 Board
- The SBus module 1 (optional)
- The SBus module 2 (optional)

The **SuperSPARC Mbus Module** occupies the first VME slot together with the I/O-10 board. It contains the SuperSPARC processor and the optional secondary cache including the cache controller.

The **I/O-10 Board** also occupies the first one of the two VME slots. It holds the MACIO chip with SCSI, Ethernet and parallel port interfaces, the DBRI with ISDN and audio interface, and the four-level arbiter and the SYSCLK driver for the VMEbus.

The **VME-10 Board** is the main board and occupies the second one of the two VME slots. It holds the Mbus-SBus Interface (MSI) and the VMEbus Interface (MVIC). It also holds the SEC with the floppy disk interface, the two serial I/O ports, the keyboard/mouse interface, the boot EPROM, the user EPROM, the RTC / NVRAM and the additional FPGA (LCA) for general board control.

The **MEMORY-10 Board** also occupies the second VME slot together with the VME-10. It holds the memory controller (EMC) and the memory itself.

In addition, the SPARC CPU-10 has two SBus slots, both of which can hold standard SBus modules: **SBus Module** 1 and **SBus Module 2.** When the SBus modules are assembled, the SPARC CPU-10 will then occupy a third VMEbus slot.

3.2 Address Map of the SPARC CPU-10

The table below lists the physical address map of the SPARC CPU-10. For the detailed address map of the local I/O devices, please refer to the “Physical Address of the NCR89C100” on page 132.

Table 35: Physical Address Map of the SPARC CPU-10

Physical Address Range	Usage
$0 0000 0000 ... $0 03FF FFFF	Main Memory Bank 1 (64 Mbyte max.)
$0 0400 0000 ... $0 0FFF FFFF	Unused Main Memory
$0 1000 0000 ... $0 13FF FFFF	Main Memory Bank 2 (64 Mbyte max.)
$0 1400 0000 ... $0 1FFF FFFF	Unused Main Memory
$0 2000 0000 ... $0 FFFF FFFF	Reserved
$1 0000 0000 ... $9 FFFF FFFF	36 Gbyte range for mapping 4 Gbyte of VME memory
$A 0000 0000 ... $D FFFF FFFF	Reserved
$E 0000 0000 ... $E 0FFF FFFF	SBus Slot 0

Table 35: Physical Address Map of the SPARC CPU-10 (Continued)

Physical Address Range	Usage
$E 1000 0000 ... $E 1FFF FFFF	SBus Slot 1
$E 2000 0000 ... $E 2FFF FFFF	Unused (SBus Slot 2)
$E 3000 0000 ... $E 3FFF FFFF	Unused (SBus Slot 3)
$ E 4000 0000 ... $ E EFFF FFFF	Reserved
$E F000 0000 ... $E F7FF FFFF	NCR89C100 (MACIO) Ethernet, SCSI, Parallel Port
$E F800 0000 ... $E FFFF FFFF	T7259 (DBRI) ISDN, Audio
$F 0000 0000 ... $F 0000 001B	EMC
$F 0000 001C ... $F DFFF FFFF	Reserved
$F E000 0000 ... $F EFFF FFFF	MSI
$F F000 0000 ... $F F1FF FFFF	SEC Boot and User EPROM, RTC, Local I/O (Serial Ports, Floppy Interface and LCA)
$F F200 0000 ... $F FCFF EFFF	Reserved
$F FCFF F000 ... $F FCFF FFFF	MVIC
$F FD00 0000 ... $F FFFF FFFF	Reserved

3.3 The L64862 Mbus to SBus Interface (MSI)

The L64862 (MSI) is the interface between the Mbus and the SBus. The physical Mbus address range for register accesses to the MSI is $F E000 0000 ... $F FEFF FFFF. For a detailed description of the MSI registers, please refer to the MSI data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*. The main functions of the MSI are outlined here.

Mbus arbitration

The MSI supports up to five Mbus masters including the MSI itself. There are only four Mbus devices having master capabilities on the CPU-10. These can be up to two processors on the Mbus module (depending on the type of Mbus module installed), the MVIC (Mbus to VMEbus Interface Chip) and the MSI itself. A parallel arbitration scheme is used by the arbiter to reduce arbitration overhead.

SBus arbitration

The MSI supports up to seven SBus masters including the MSI itself. There are only five SBus devices having master capabilities on the CPU-10. These can be up to two masters on two SBus modules (depending on the type of SBus modules installed in the SBus slots), the NCR89C100 (MACIO), the T7259 (DBRI) and the MSI itself. A round-robin scheme is used for the arbitration of the SBus.

Mbus to SBus Protocol Conversion

The MSI controls accesses from Mbus masters to SBus devices or from SBus masters to the main memory connected to the Mbus via the EMC memory controller. Each bus uses different protocols and the MSI takes care of the communication between the two buses. If an Mbus operation takes more than 200 microseconds to complete, the MSI assumes the Mbus master is trying to access a non-existent slave. It forces a termination of the operation and signals an error to the Mbus master.
If an SBus operation takes more than 256 SBus clock cycles, the MSI terminates the operation and signals an error to the SBus master.

Data Buffering

The MSI interfaces the 64-bit wide data path of the Mbus and the 32-bit data path of the SBus. It contains three sets of data buffers: one Mbus read / write buffer and two DVMA (direct virtual memory access) read / write buffers to match the bandwidths between the Mbus and the SBus.

The Mbus read / write buffer allows an Mbus to SBus write operation to complete before the SBus cycle is finished (“write posting”). It also accumulates data from the SBus during SBus to Mbus read operations.
The two DVMA read / write buffers are used during operations where the IOMMU of the MSI is involved.

I/O Reference MMU

SBus masters send out virtual addresses to access the memory via the EMC on the Mbus. The Mbus is based on physical addresses. The IOMMU of the MSI translates virtual addresses into physical addresses. The IOMMU has a TLB (Translation Look-aside Buffer) with 16 entries to speed up the translation.

3.4 The Mbus Participants

The Mbus is the processor bus of the CPU-10 and connects the Mbus slot, the memory controller L64860 (EMC), the VME interface MB86986 (MVIC) and the Mbus to SBus Interface L64862 (MSI), which is described in chapter “The L64862 Mbus to SBus Interface (MSI)” on page 70. The other Mbus participants are described below in the following chapters.

3.5 The Mbus Modules

Mbus modules are processor modules built with the SuperSPARC CPU chip, which is a highly integrated implementation of the SPARC RISC superprocessor. Mbus modules can have up to two processors and are available in different frequencies. Additionally, the Mbus modules are available with or without secondary cache. Performance of the SPARC CPU-10 is therefore scalable simply through the use of different Mbus modules.

3.5.1 The superSPARC TMS390Z50 Processor

The SuperSPARC TMS390Z50 is a high performance superscalar SPARC microprocessor which conforms to the popular SPARC RISC architecture. The main features of the TMS390Z50 are described below. For detailed information, please refer to the TMS390Z50 data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

- High Performance SPARC Processor (3 instructions per cycle)
- SPARC Integer Unit
- SPARC Reference MMU
- Floating Point Unit
- 20 Kbyte Instruction Cache, 16 Kbyte Data Cache
- Support for External Cache via Highly Pipelined, Non-multiplexed Interface
- Multiprocessor Cache Coherence Support

3.5.2 The superSPARC TMS390Z55 Cache Controller

The SuperSPARC TMS390Z55 is an optional external cache controller for the SuperSPARC TMS390Z50 processor. It supports up to 1 Mbyte of external cache using synchronous 128K*8 or 128K*9 SRAM chips which allow pipelined operation. The TMS390Z55 cache controller is connected to the TMS390Z50 processor via the Vbus, which decouples it from the Mbus. The external cache memory provides significant performance improvement and decreases bus traffic on the Mbus. The main features of the TMS390Z55 are described below. For detailed information, please refer to TMS390Z55 Data Sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

- Cache Controller for the TMS390Z50 SuperSPARC Processor
- Support of 1 Mbyte of external cache
- Support for Cache Coherent Multiprocessing
- Memory Block Copy and Block Clear
- Data Prefetching
- Cache Performance Monitor
- Built-In Self Test (BIST)

3.6 The Main Memory

The SPARC CPU-10 is available with one of two different memory module types called MEMORY-10 and MEMORY-10A. MEMORY-10 has a capacity of 32, 64 and 128 Mbytes and MEMORY-10A has a capacity of 64, 128 and 192 Mbytes .
The memory data path on both modules is 128 bit wide and has additional 16 bits for Error Correction Code (ECC).
The Main Memory is controlled by the L64860 Error Correcting Memory Controller (EMC).
For the board geometry please also see the "Mechanical Construction of the SPARC CPU-10" on page 67.

3.6.1 The MEMORY-10 Module

The Main Memory on the MEMORY-10 consists of up to 72 devices of 1M*4 or 4M*4 (400 mil) DRAMs in one or two memory banks.
The following table shows the memory capacities available on the CPU-10 and the corresponding physical address ranges when using the MEMORY-10 module.

Table 36: Memory Capacities using MEMORY-10

Memory Capacity	Physical Address Ranges	Chip Count	Bank 1 assembled	Bank 2 assembled	DRAM Devices
32 MBytes	$0 0000 0000 ... $0 00FF FFFF and $0 1000 0000 ... $0 10FF FFFF	72	*	*	1M*4
64 MBytes	$0 0000 0000 ... $0 03FF FFFF	36	*		4M*4 (400 mil)
128 Mbytes	$0 0000 0000 ... $0 03FF FFFF and $0 1000 0000 ... $0 13FF FFFF	72	*	*	4M*4 (400 mil)

3.6.2 The MEMORY-10A Module

The Main Memory on the MEMORY-10A consists of up to 108 devices of 4M*4 (300 mil) DRAMs in one, two or three memory banks.
The following table shows the memory capacities available on the CPU-10 and the corresponding physical address ranges when using the MEMORY-10A module.

Table 37: Memory Capacities using MEMORY-10A

Memory Capacity	Physical Address Ranges	Chip Count	Bank 1 assembled	Bank 2 assembled	Bank 3 assembled	DRAM Devices
64 MBytes	$0 0000 0000 ... $0 03FF FFFF	36	*			4M*4 (300 mil)
128 MBytes	$0 0000 0000 ... $0 07FF FFFF	72	*	*		4M*4 (300 mil)
192 Mbytes	$0 0000 0000 ... $0 0BFF FFFF	108	*	*	*	4M*4 (300 mil)

3.6.3 The L64860 Error Correcting Memory Controller (EMC)

The L64860 Error Correcting Memory Controller (EMC) directly interfaces the 128 bit wide Main Memory to the 64 bit wide Mbus. The EMC operates as an Mbus slave device only. It provides full DRAM control including refresh cycles. It uses two RAS lines to select the two DRAM banks. Two CAS lines are used to select the upper 64 data bits or the lower 64 data bits. Partial write accesses (less than 8 bytes) result in a Read-Modify-Write-Cycle (RMC).

The physical Mbus address range for register accesses to the EMC is $F 0000 0000 ... $F 0000 001B. For the detailed description of the EMC registers please refer to the EMC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

The following outlines the main features of the EMC.

Features of the EMC

- Supports the Full Level-2 Mbus Protocol
- Supports 40 MHz Mbus Frequency
- Single Bit Error Correction and Multi Bit Error Detection
- Supports Accesses from Single Byte up to 128 Bytes Burst Transfers
- 64 Byte Data Buffer for queueing two 32 Byte Mbus Writes

3.7 The VMEbus Interface

The VMEbus Interface of the SPARC CPU-10 is based upon the MB86986 Mbus to VMEbus Interface Controller (MVIC). In addition to the MVIC, there is an FPGA (LCA) which monitors the VMEbus interrupts and maps them to the onboard interrupt logic. Besides this interrupt mapping logic, the FPGA contains a second VMEbus timer, which is in addition to the bus timer implemented in the MVIC. Furthermore, the SPARC CPU-10 contains system controller functions such as single-level / four-level-arbiter, IACK daisy chain driver and SYSCLK driver.

Features of the VMEbus Interface

- Complete 32-bit Master and Slave Interface compatible to IEEE-1014
- Four Level VMEbus Requester
- VMEbus Interrupter for all VMEbus IRQs providing 8-bit Vector
- Support of VMEbus Interrupt Acknowledge Cycles for all VMEbus IRQs
- VMEbus Timer with Programmable Timeout Value
- System Controller Functions
    - Single Level Arbiter or Four Level Arbiter in Prioritized Mode (PRI), Round Robin Mode (RRS) and Prioritized Round Robin Mode (PRR)
    - SYSCLK Driver
- IACK Daisy Chain Driver
- SYSRESET Driver
- ACFAIL Driver
- SYSFAIL Driver

3.7.1 The MB86986 Mbus to VMEbus Interface Controller (MVIC)

The MB86986 Mbus to VMEbus Interface Controller (MVIC) directly interfaces the 64-bit Mbus to the 32-bit VMEbus. The MVIC has full VMEbus master/slave capabilities and supports VMEbus block mode burst transfers. The MVIC fully supports Level 1 Mbus master and Level 2 Mbus slave interfaces. The MVIC also contains a general purpose Mbus DMA controller using Mbus burst transfers.

The physical Mbus address range for register accesses to the MVIC is as follows: $F FCFF F000 ... $F FCFF FFFF. For the detailed description of the MVIC registers please refer to the MVIC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

Features of the MVIC on Mbus Side

- Supports the Full Level 1 Mbus Master and Level 2 Mbus Slave Protocol
- Supports 40 MHz Mbus Frequency
- Supervisor Bit when Accessed by VMEbus Supervisory Cycle
- General Purpose Mbus DMA Controller
- 32-Byte Mbus Master Burst Transfers
- 64-Byte Mbus Slave Burst Transfers

Features of the MVIC on VMEbus Side

- Completely Asynchronous VMEbus Operation
- Fully Supports VMEbus Master and Slave Interfaces
- A32 and A24 Access in Normal and Block Mode
- A16 Access in Normal Mode
- Does not Support Unaligned Transfers
- Normal, Block and User Definable Address Modifier
- 25 MByte/sec VME Block Mode and 10 MByte/sec VME Normal Mode Transfer Rate
- Supports VMEbus Interrupt Acknowledge Cycles for all VMEbus IRQs

3.7.2 The VMEbus Transfer Modes

The VMEbus master interface and slave interface allow A32 and A24 accesses in Normal Mode and Block Mode and allow A16 accesses only in Normal Mode. D8, D16 and D32 transfers are supported. Unaligned VMEbus transfers are not supported. The following table shows the transfers and modes that are supported by the VMEbus interface and the corresponding VMEbus address ranges.

Table 38: VMEbus Interface Transfers

Transfer	Mode	VMEbus Address Range	Resulting Size
A32 / D32	Normal & Block	$ 0000 0000 ... $ FEFF FFFF	4 Gbyte minus 16 Mbyte
A32 / D16	Normal		
A32 / D8	Normal		
A24 / D32	Normal & Block	$ FF00 0000 ... $ FFFE FFFF	16 Mbyte minus 64 Kbyte
A24 / D16	Normal		
A24 / D8	Normal		
A16 / D32	Normal	$ FFFF 0000 ... $ FFFF FFFF	64 Kbyte
A16 / D16	Normal		
A16 / D8	Normal		

3.7.3 The VMEbus Address Modifiers

The address modifier combinations supported by the VMEbus master and slave interface are shown in the next table. In addition, the VMEbus master interface supports the generation of user defined address modifiers.

During an MVIC's VMEbus master access, the Mbus Supervisor Bit MAD[59] of the initiating Mbus cycle controls whether the VMEbus cycle which follows is Non-Privileged or Supervisory.

During an MVIC's VMEbus slave access, the Mbus Supervisor Bit MAD[59] of the Mbus cycle is controlled by the VMEbus address modifiers. The bit is "one" if the initiating VMEbus access is Supervisory.

For detailed information please refer to the MVIC Data Sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10.*

Table 39: Supported Address Modifier Codes

Address Modifier	Address Mode	Transfer Mode	Mbus Supervisor Bit MAD[59]
$29	A16	Short Non-Privileged Access	0
$2D	A16	Short Supervisory Access	1
$39	A24	Standard Non-Privileged Data Access	0
$3B	A24	Standard Non-Privileged Block Transfer	0
$3D	A24	Standard Supervisory Data Access	1
$3F	A24	Standard Supervisory Block Transfer	1
$09	A32	Extended Non-Privileged Data Access	0
$0B	A32	Extended Non-Privileged Block Transfer	0
$0D	A32	Extended Supervisory Data Access	1
$0F	A32	Extended Supervisory Block Access	1

3.7.4 The VMEbus Master Interface Address Map

The full 4 Gbyte VMEbus address range is available for the VMEbus Master Interface. It can be located on any 4 Gbyte boundary within the unused 36 Mbyte Mbus address range from - $1 0000 0000 to $9 FFFF FFFF. The location within this 36 Mbyte address range is controlled by the MVIC "Mbus Base Address Register" bits MBAR[31:28] for Normal Mode, bits MBAR[27:24] for Block Mode and bits MBAR[23:20] for User Mode.

The table on the next page shows the register bit settings for the possible Mbus base addresses.

NOTE: During an A32 VMEbus Master Access the Mbus address bits MAD[31:1] are passed through the MVIC to the VMEbus address bits VME_A[31:1].
During A24 the bits MAD[23:1] are passed through the MVIC to the VMEbus address bits VME_A[23:1].
During A16 the bits MAD[15:1] are passed through the MVIC to the VMEbus address bits VME_A[15:1].

Table 40: The VMEbus Master Interface Address Map

Physical Mbus Base Address	MBAR[31:28] (Normal Mode) MBAR[27:24] (Block Mode) MBAR[23:20] (User Mode)
reserved	$0
$1 0000 0000	$1
$2 0000 0000	$2
$3 0000 0000	$3
$4 0000 0000	$4
$5 0000 0000	$5
$6 0000 0000	$6
$7 0000 0000	$7
$8 0000 0000	$8
$9 0000 0000	$9
reserved	$A
reserved	$B
reserved	$C
reserved	$D
reserved	$E
reserved	$F

NOTE: The register bits MBAR[31:28], MBAR[27:24] and MBAR[23:20] for the three VMEbus modes must always have different contents (except when the modes are not used and the contents are zero). That means that VMEbus accesses in the different modes are initiated via different physical Mbus base addresses.

3.7.5 The VMEbus Slave Interface Address Map

The MVIC's VMEbus Slave Interface can access all of the onboard DRAM available on the CPU-10. The table below shows the physical Mbus address ranges for the onboard DRAM depending on the capapcity installed on the CPU-10.

Table 41: Physical Address Ranges for the Onboard DRAM

Memory Capacity	Physical Address Ranges
32 MBytes	$0 0000 0000 ... $0 00FF FFFF and $0 1000 0000 ... $0 10FF FFFF
64 MBytes	$0 0000 0000 ... $0 03FF FFFF
128 Mbytes	$0 0000 0000 ... $0 03FF FFFF and $0 1000 0000 ... $0 13FF FFFF

3.7.6 The VMEbus Slave Window Size

The VMEbus Slave Interface supports A32, A24 and A16 VMEbus address modes. The address modes can be enabled/disabled separately. The following table shows the slave window size for the possible VMEbus address modes. The slave window size for A32 address modes is programmable inside the MVIC and the slave window sizes for A24 and A16 are 1 Mbtyte and 32 Byte respectively and cannot be changed inside the MVIC. For detailed a description of the MVIC, please refer to the MVIC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

Table 42: VMEbus Slave Window Size

Address Mode	Slave Window Size
A32	16 Mbyte ... 4096 Mbyte
A24	1 Mbyte
A16	32 Byte

3.7.7 VMEbus System Controller

By default, the SPARC CPU-10 is a VMEbus slot 1 device functioning as VMEbus system controller with the VMEbus arbiter and SYSCLK driver. The arbiter and SYSCLK driver can be enabled/disabled via SW3-1 on the VME-10 board. By default, the system controller functions are **enabled**, which means that the CPU-10 arbitrates on the VMEbus and drives SYSCLK. The following table shows the switch setting of SW3-1 for the VME slot 1 configurations of the CPU-10.

Table 43: VME Slot 1 Functions

SW3-1 on VME-10	VME Slot 1 Functions	Default
OFF	Disabled	
ON	Enabled	*

3.7.7.1 The VMEbus Arbiter

The four level VMEbus arbiter inside the MVIC is **not** in use on the SPARC CPU-10. Therefore, another VMEbus arbiter has been built with additional logic. This VMEbus arbiter is located on the I/O-10 board, which occupies the first one of the two VME slots of the CPU-10. The VMEbus arbiter is capable of single level arbitration or four level arbitration in prioritized mode (PRI), round robin mode (RRS) and prioritized round robin mode (PRR). The following table shows the switch settings of switches SW3-1 and SW3-2 on the I/O-10 board for the possible arbiter modes.

The arbiter can be **enabled / disabled** via switch SW3-1 on the VME-10 board. Note that SW3-1 also enables / disables the VMEbus SYSCLK driver. For the correct setting of SW3-1, please see the “VME Slot 1 Functions” on page 83.

Table 44: VMEbus Arbiter Modes

SW3-1 on I/O-10	SW3-2 on I/O-10	Arbiter Mode	Default
OFF	OFF	Round Robin (RRS)	*
OFF	ON	Prioritized (PRI)	
ON	OFF	Prioritized Round Robin (PRR)	
ON	ON	Single Level (SL)	

WARNING: The four level VMEbus arbiter inside the MVIC is not in use. It is disabled after powerup and must not be enabled by software. That means that bit VMCR[0] in the MVIC’s "VMEbus Master Configuration Register" always must be "0". Note that bit VMCR[1] for the MVICs’ arbitration mode will then not have any effect on the arbitration mode of the CPU-10.

3.7.7.2 The VMEbus SYSCLK

The SPARC CPU-10 has a VMEbus SYSCLK driver. It is located on the I/O-10 board, which occupies the first one of the two VME slots of the CPU-10.

The VMEbus SYSCLK can be **enabled / disabled** via switch SW3-1 on the VME-10 board. Note that SW3-1 also enables / disables the VMEbus arbiter. For the switch setting of SW3-1 see“VME Slot 1 Functions” on page 83.

3.7.8 The VMEbus Timer

The SPARC CPU-10 has implemented two VMEbus timers. Their function is to prevent system deadlock. In the case where system deadlock occurs, the timer (if enabled) asserts the VME BERR signal after the programmed timeout value has expired, so the system deadlock can be resolved. One of the VMEbus timers is located inside the MVIC and the second one is located inside the LCA (which is a field programmable gate array).

The VMEbus timer inside the MVIC is only relevant when the MVIC is acting as VMEbus master. When the MVIC, acting as VMEbus master, is accessing a non-responding slave on the VMEbus, then its internal bus timer will assert VMEBERR after the programmed timeout value has expired. For detailed information about the MVIC's VMEbus timer, please refer to the MVIC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

The VMEbus timer inside the LCA is a general bus timer which monitors each VMEbus access regardless of whether the MVIC is VMEbus master, VMEbus slave or is not involved in the VMEbus cycle at all.

3.7.8.1 Control of the VMEbus Timer inside the LCA

The VMEbus timer is controlled with the BUS_TIMEOUT_CTRL register.

The **BUSTIME_ENA** bit enables / disables the VMEbus timer.

The **BUSTIME[2:0]** bits select the timeout value for the VMEbus timer.

The tables below show the register layout and the bit settings. The table on the next page shows the minimum and maximum values for the VMEbus timer and their respective bit settings.

BUS_TIMEOUT_CTRL (Address: $F F124 0012)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	BUSTIME ENA	BUSTIME[2:0]		
r	r	r	r	r/w	r/w		

BUSTIME_ENA	**VMEbus Timer**	**Default**
0	Disabled	*
1	Enabled	

The table below shows the minimum and maximum values for the VMEbus timer.

BUSTIME[2:0]	Minimum Time until VME BERR is Asserted	Maximum Time until VME BERR is Asserted	Default
000	4.0 us	6 us	*
001	8.0 us	12 us	
010	16 us	23 us	
011	64 us	92 us	
100	250 us	370 us	
101	1.0 ms	1.4 ms	
110	4.0 ms	5.6 ms	
111	500 ms	700 ms	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

WARNING: The VMEbus timer must NOT be enabled while setting the timeout value. Set the timeout value first and then enable the timer.

3.7.9 The VMEbus ACFAIL and SYSFAIL

The two VMEbus signals SYSFAIL and ACFAIL can be controlled by software via two registers, which are located in the LCA.

NOTE: The SYSFAIL signal will be asserted by hardware after powerup, according to the VME Specification, and has to be deasserted by software to enable any VMEbus transfer.

3.7.9.1 Control of VMEbus ACFAIL

ACFAIL_ASSERT in the ACFAIL_CTRL Register is used to control the VMEbus ACFAIL signal. The following tables show the register layout and the bit settings.

ACFAIL_CTRL (Address: $F F124 000A)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	ACFAIL ASSERT
r	r	r	r	r	r	r	r/w

ACFAIL_ASSERT	VME ACFAIL signal
0	Negated (high)
1	Asserted (low)

3.7.9.2 Control of VMEbus SYSFAIL

SYSFAIL_ASSERT in the SYSFAIL_CTRL Register is used to control the VMEbus SYSFAIL signal.

The following tables show the register layout and the bit settings.

SYSFAIL_CTRL (Address: $F F124 000B)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	SYSFAIL ASSERT
r	r	r	r	r	r	r	r/w

SYSFAIL_ASSERT	VME SYSFAIL signal
0	Negated (high)
1	Asserted (low)

3.7.10 The VMEbus SYSRESET

The DIP switches SW3-3 and SW3-4 configure the VMEbus SYSRESET. SW3-3 is used for monitoring the SYSRESET on the CPU-10 and SW3-4 is used for driving the SYSRESET to the VMEbus.

For the position of SW3 on the VME-10 board, please see the location diagrams which appear at the beginning of the installation section of this manual.

3.7.10.1 Monitoring the VMEbus SYSRESET

Monitoring of the VMEbus SYSRESET on the CPU-10 is enabled / disabled via DIP switch SW3-3 on the VME-10 board. When SW3-3 is set to ON, a VMEbus SYSRESET will be sensed by the LCA, then the LCA generates a reset of the CPU-10. This reset is called BOARD_RESET and does **NOT** reconfigure the LCA.

Table 45: Enable Monitoring of VMEbus SYSRESET

SW3-3 on VME-10	Monitoring of VMEbus SYSRESET	Default
OFF	Disabled	
ON	Enabled	*

3.7.10.2 Driving the VMEbus SYSRESET

Driving the VMEbus SYSRESET on the CPU-10 is enabled / disabled via DIP switch SW3-4 on the VME-10 board. When SW3-4 is set to ON, the CPU-10 drives the SYSRESET to the VMEbus. This is done either during a POWERUP_RESET of the CPU-10 or during the SYSRESET programmed inside the LCA.

A POWERUP_RESET of the CPU-10 reconfigures the LCA and will be generated when one of the following conditions occurs:

- Powerup of the supply voltage
- Toggling the front panel reset key (and SW3-2 on the VME-10 board is ON)
- Timeout of the watchdog timer

Programming the VMEbus SYSRESET inside the LCA is described in chapter “Programming the VMEbus SYSRESET” on page 92.

The table on the next page shows the settings of SW3-4 for driving the VMEbus SYSRESET.

Table 46: Enable Driving of VMEbus SYSRESET

SW3-4 on VME-10	Driving of VMEbus SYSRESET	Default
OFF	Disabled	
ON	Enabled	*

3.7.10.3 Programming the VMEbus SYSRESET

The SYSRESET_CTRL register inside the LCA is used to generate a VMEbus SYSRESET. Any write access to this register will generate a VMEbus SYSRESET with a minimum length of 200 ms according to the VME specification. Reading the SYSRESET_CTRL Register will always return $FF. The VMEbus SYSRESET is only driven to the VMEbus by the CPU-10 when DIP switch SW3-4 on the VME-10 is set to ON.

See also chapter “Driving the VMEbus SYSRESET” on page 90.

SYSRESET_CTRL (Address: $F F124 001C)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	1
w	w	w	w	w	w	w	w

3.7.11 The VMEbus Interrupts

The MVIC on the CPU-10 is not able to monitor VMEbus interrupts. Additional logic was therefore built for monitoring VMEbus interrupts and for linking the VMEbus interrupts with the local interrupt structure. This additional logic is implemented in an LCA, which is an FPGA (field programmable gate array).

The LCA monitors the VMEbus interrupts and allows a flexible way of mapping them to the local SBus interrupts. The SBus interrupts themselves are transformed by the SEC into processor interrupts causing software exceptions.

The mapping of the VMEbus interrupts is realized with a set of seven 4-bit wide registers, called **VME_IRQ1_MAP... VME_IRQ7_MAP registers**. Each of the VMEbus interrupts VME_IRQ1... VME_IRQ7 has its own VME_IRQx_MAP register. With the help of this register, it is possible to select one out of seven SBus interrupts which becomes active upon the assertion of the VME_IRQ. This allows maximum flexibility for the onboard interrupt handling of the VMEbus interrupts.

NOTE: Several VME_IRQs can be mapped to the same SBus Interrupt. Software then has to find out the status of the pending VME IRQs, which software does with the help of the pending bits of the VME_IRQ_MAP registers.

3.7.11.1 The VME_IRQ1_MAP ... VME_IRQ7_MAP Registers

The registers VME_IRQ1_MAP ... VME_IRQ7_MAP control the mapping of the VMEbus interrupts VME_IRQ1 ... VME_IRQ7 to the SBus Interrupts SBus_IRQ1 ... SBus_IRQ7. The following tables show the register layout and the possible bit settings. **IRQxP** bit is the pending bit of VME_IRQx. **IRQxMAP[2:0]** bits control which SBus Interrupt line is asserted upon VME_IRQx.

VME_IRQx_MAP (Address: $F F124 0001 ... $ F F124 0007)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	IRQxP	IRQxMAP[2:0]		
r	r	r	r	r	r/w		

IRQxP	VME_IRQx pending
0	Yes
1	No

IRQxMAP[2:0]	Interrupt asserted upon VME_IRQx	Default
000	None	*
001	SBus IRQ1	
010	SBus IRQ2	
011	SBus IRQ3	
100	SBus IRQ4	
101	SBus IRQ5	
110	SBus IRQ6	
111	SBus IRQ7	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.7.12 The MVIC Interrupts

The MVIC has two interrupt lines called the MVIC_DMA_IRQ and the MVIC_ERR_IRQ. Both interrupts are routed to the local interrupt logic via the LCA. The MVIC_DMA_IRQ can trigger one of the seven SBus interrupts and has the same mapping functionality as the VMEbus interrupts. The MVIC_ERR_IRQ passes directly through the LCA and generates a Level-15 NMI.

3.7.12.1 The MVIC DMA Interrupt

The MVIC_DMA_IRQ is asserted whenever a DMA is completed. The LCA monitors the MVIC_DMA_IRQ and allows a flexible way of mapping it to the local SBus interrupts. The SBus interrupts themselves are transformed by the SEC into processor interrupts causing software exceptions.
The 4-bit wide MVIC_DMA_IRQ_MAP register in the LCA controls the mapping of the MVIC_DMA_IRQ to the SBus Interrupts. With the help of this register it is possible to select one out of seven SBus interrupts which become active upon the assertion of the MVIC_DMA_IRQ. This allows maximum flexibility for the onboard interrupt handling of the MVIC_DMA_IRQ.

The detailed register layout and setting are described on the next page.

DMAIP bit is the MVIC DMA Interrupt pending bit. **DMAIRQMAP[2:0]** bits control which SBus IRQ line is asserted upon an MVIC DMA Interrupt.

The following tables show the register layout and the possible bit settings.

MVIC_DMA_IRQ_MAP (Address: $F F124 0000)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	DMAIP	DMAIRQMAP[2:0]		
r	r	r	r	r	r/w		

DMAIP	MVIC DMA Interrupt pending
0	Yes
1	No

DMAIRQMAP[2:0]	Interrupt Asserted	Default
000	None	*
001	SBus IRQ1	
010	SBus IRQ2	
011	SBus IRQ3	
100	SBus IRQ4	
101	SBus IRQ5	
110	SBus IRQ6	
111	SBus IRQ7	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.7.12.2 The MVIC Error Interrupt

The MVIC_ERR_IRQ is asserted when one of the following conditions occurs:

- A16 NMI
- DMA Mbus Error
- VME Slave Buffered Write Error
- VME Master Buffered Write Error

The MVIC_ERR_IRQ passes directly through the LCA and generates a Level-15 NMI. The status of the MVIC_ERR_IRQ can be read in the MVIC Interrupt Status Register. For details please refer to the MVIC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

3.8 SBus Participants

On the SPARC CPU-10 the 32-bit wide **SBus** has the following participants: the MSI, which is described in “The L64862 Mbus to SBus Interface (MSI)” on page 70, the two SBus slots for standard SBus modules, the two I/O interface chips L64861 (SEC) and the NCR89C100 (MACIO) and the ISDN/audio interface T7259 (DBRI). These SBus participants are described in the following chapters.

3.9 The SBus Slots

The SPARC CPU-10 provides two SBus slots for industry-standard SBus modules. SBus modules enable the expansion of I/O interfaces, memory and processing performance with a wide range of solutions. The two SBus slots fully support SBus DMA transfers of SBus modules having DMA capabilities.

When the SBus modules are assembled, the SPARC CPU-10 occupies a third VMEbus slot and has an additional front panel, which is provided by FORCE Computers.

3.10 The L64861 SBus to Ebus Controller (SEC)

The L64861 (SEC) is the interface between the 32-bit wide SBus and the 8-bit wide local I/O bus (Ebus). The SEC is an SBus slave device which provides all the control signals for the 8-bit wide devices on the Ebus. The 8-bit wide devices are as follows: two dual channel serial controllers (85C30), a high speed floppy disk controller (82077SL), the RTC/NVRAM, two boot PROM devices, two flash memory devices for user defined purposes and an LCA (FPGA) for general board control.

In addition, the SEC contains the counters / timers and the interrupt logic which is necessary for the Mbus multiprocessor system. The SEC also contains part of the system reset logic of the SPARC CPU-10.

Features of the SEC on the SPARC CPU-10

- Interface between 32-bit SBus and 8-bit Ebus
- Control of all the onboard 8-bit devices
- Interrupt controller
- System reset control
- Programmable 22-bit counters & timers
- Packaged in 160 pin PQFP

For further information about the L64861 (SEC) please refer to the respective data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10.*

3.10.1 The SEC Interrupt Controller

The SEC contains the interrupt controller of the SPARC CPU-10. The interrupt controller fully supports the multiprocessor architecture of the Mbus. It provides software interrupts on all interrupt levels. In addition, the SEC monitors the hardware interrupts and translates them into the Mbus processor interrupts according to a hardwired encoding scheme. The devices having interrupt capabilities are as follows:

- Mbus Module
- SBus Modules
- LCA
- MSI
- EMC
- MVIC
- MACIO
- Floppy Disk Controller
- Serial Controller 1 and 2
- ISDN Controller

The SEC has four groups of outgoing interrupts for up to four Mbus processors. Each Mbus Module can hold two processors, which means that two Mbus modules with up to four processors can be supported by the SEC.

The SPARC CPU-10 has only one Mbus module installed with a maximum of two processors. This means that only two groups of interrupts are needed and the other two groups remain unused.

Each of the two groups of interrupts consists of four lines which are used for encoding interrupt levels between 0 and 15. Level 15 is the highest interrupt level whereas level 0 indicates that no interrupt is pending. The table on the next page shows the processor interrupt levels generated upon the respective hardware and software interrupts. The interrupts can be subdivided into directed, undirected and broadcast interrupts.

For further information about the interrupt controller on the SPARC CPU-10 please refer to the SEC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10.*

3.10.1.1 SEC Interrupt Mapping

The following table shows the processor interrupt levels generated upon the respective hardware and software interrupts.

Table 47: SEC Interrupt Mapping

IRQ Level	Interrupt Sources
0	No Interrupt pending
1	Soft Interrupt<1>,
2	Soft Interrupt<2>, SBus IRQ<1>, VME IRQ<7..1>, MVIC DMA IRQ
3	Soft Interrupt<3>, SBus IRQ<2>, VME IRQ<7..1>, MVIC DMA IRQ, Centronics IRQ
4	Soft Interrupt<4>, SCSI IRQ
5	Soft Interrupt<5>, SBus IRQ<3>, VME IRQ<7..1>, MVIC DMA IRQ
6	Soft Interrupt<6>, Ethernet IRQ
7	Soft Interrupt<7>, SBus IRQ<4>, VME IRQ<7..1>, MVIC DMA IRQ
8	Soft Interrupt<8>
9	Soft Interrupt<9>, SBus IRQ<5>, VME IRQ<7..1>, MVIC DMA IRQ, Mbus Module IRQ, ISDN/Audio IRQ
10	Soft Interrupt<10>, System Timer IRQ
11	Soft Interrupt<11>, SBus IRQ<6>, VME IRQ<7..1>, MVIC DMA IRQ, Floppy IRQ
12	Soft Interrupt<12>, Serial IRQ, Keyboard/Mouse IRQ
13	Soft Interrupt<13>, SBus IRQ<7>, VME IRQ<7..1>, MVIC DMA IRQ
14	Soft Interrupt<14>, Processor Timer IRQ
15	Soft Interrupt<15>, Mbus Aerr IRQ, MSI IRQ, EMC IRQ, MVIC ERR IRQ, Abort Key IRQ, Watchdog IRQ, VME SYSFAIL IRQ, VME ACFAIL IRQ

NOTE: The following interrupts are controlled via the LCA. The VME IRQ<7..1> and the MVIC DMA IRQ both resulting in **SBus IRQ<7..1>.** The MVIC ERR IRQ, the Abort Key IRQ, the Watchdog IRQ, the VME SYSFAIL IRQ and the VME ACFAIL IRQ all resulting in an **Mbus Aerr IRQ** which generates a **Level 15 Interrupt.** For detailed information about the control of these interrupts. please refer to the respective chapters in this manual. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10.*

3.10.1.2 SEC System Control Registers

The following table shows the physical addresses for the SEC timer and interrupt register sets. Please note that only the register sets for the processors 0 and 1 and for the system are important. Processors 2 and 3 are not available on the CPU-10.

For detailed information about the register layout, please refer to the respective data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

Table 48: SEC System Control Registers

Physical Address	Device
$F F130 0000	Processor 0 Counter Registers
$F F130 1000	Processor 1 Counter Registers
$F F131 0000	System Counter Registers
$F F140 0000	Processor 0 Interrupt Registers
$F F140 1000	Processor 1 Interrupt Registers
$F F141 0000	System Interrupt Registers

3.10.2 The NMI (Non Maskable Interrupt)

There are several sources which can trigger a non maskable interrupt (NMI) on the CPU-10. One source is the Mbus Module itself, the other sources are the SYSFAIL and the ACFAIL signals of the VMEbus, the watchdog timer and the ABORT Key on the front panel of the CPU-10. The various NMI sources can be enabled / disabled with the help of Interrupt Enable (IE) bits.

An additional NMI source is an MVIC interrupt signal called MVIC_ERR_IRQ. The MVIC_ERR_IRQ passes directly through the LCA and cannot be masked with an enable bit inside the LCA. The status of the MVIC_ERR_IRQ can be read in the register MVIC Interrupt Status Register. For further details, please refer to the MVIC data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

When an NMI source is activated (and the enable bit set to "1"), the LCA drives the "open collector" Mbus signal AERR. The AERR signal is monitored by the SEC which will send the NMI to the processor(s) on the Mbus module by encoding Level 15 (NMI) on the IRL[3..0] lines. In the interrupt routine, software has to identify the source which triggered the NMI with the help of the interrupt pending (IP) bits. To clear a pending NMI, software has to clear the Interrupt Enable (IE) bit of the identified NMI source. When subsequent NMIs have to be enabled, the IE bit must be set again.

The following registers contain the NMI enable bits of the possible NMI sources.

- SYSFAIL_IRQ_CTRL1 register
- SYSFAIL_IRQ_CTRL2 register
- ACFAIL_IRQ_CTRL register
- ABORT_IRQ_CTRL register
- WD_CTRL2 register

The registers are described in detail in the following chapters.

3.10.2.1 The NMI on SYSFAIL Negation

The SYSFAIL_IRQ_CTRL1 Register controls the generation of a non maskable interrupt (NMI) upon the negation (rising edge) of the VMEbus SYSFAIL signal.

SYSFAIL_NEGATE_IE bit is used to enable / disable the generation of the NMI. To clear a pending NMI triggered by negation of SYSFAIL, this bit must be cleared. It must be set again to enable further NMIs.

The following tables show the register layout and the bit settings.

SYSFAIL_IRQ_CTRL1 (Address: $F F124 000C)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	SYSFAIL NEGATE IE
r	r	r	r	r	r	r	r/w

SYSFAIL_NEGATE_IE	NMI on Negation of VME SYSFAIL	Default
0	Disabled / Cleared	*
1	Enabled	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.10.2.2 The NMI on SYSFAIL Assertion

The SYSFAIL_IRQ_CTRL2 Register controls the generation of a non maskable interrupt (NMI) upon the assertion (falling edge) of the VMEbus SYSFAIL signal.

SYSFAIL_ASSERT_IE bit is used to enable / disable the generation of the NMI. To clear a pending NMI triggered by assertion of SYSFAIL, this bit must be cleared. It must be set again to enable further NMIs.

The following tables show the register layout and the bit settings.

SYSFAIL_IRQ_CTRL2 (Address: $F F124 000D)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	SYSFAIL ASSERT IE
r	r	r	r	r	r	r	r/w

SYSFAIL_ASSERT_IE	**NMI on Assertion of VME SYSFAIL**	**Default**
0	Disabled / Cleared	*
1	Enabled	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.10.2.3 The NMI on ACFAIL Assertion

The ACFAIL_IRQ_CTRL Register controls the generation of a non maskable interrupt (NMI) upon the assertion (falling edge) of the VMEbus ACFAIL signal.

ACFAIL_IE bit is used to enable / disable the generation of the NMI. To clear a pending NMI triggered by assertion of ACFAIL, this bit must be cleared. It must be set again to enable further NMIs.

The following tables show the register layout and the bit settings.

ACFAIL_IRQ_CTRL (Address: $F F124 000E)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	ACFAIL IE
r	r	r	r	r	r	r	r/w

ACFAIL_IE	**NMI on Assertion of VME ACFAIL**	**Default**
0	Disabled / Cleared	*
1	Enabled	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.10.2.4 The NMI on Abort Key

The ABORT_IRQ_CTRL Register controls the generation of a non maskable interrupt (NMI) upon toggling down the RESET / ABORT Key on the front panel.

ABORT_IE is used to enable / disable the generation of the NMI. To clear a pending NMI caused by the ABORT Key, this bit must be cleared. It must be set again to enable further NMIs.

The following tables show the register layout and the bit settings

ABORT_IRQ_CTRL (Address: $F F124 000F)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	ABORT IE
r	r	r	r	r	r	r	r/w

ABORT_IE	**NMI on Toggling ABORT Key**	**Default**
0	Disabled / Cleared	*
1	Enabled	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.10.2.5 The NMI on Watchdog

The WD_CTRL2 Register controls the generation of a non maskable interrupt (NMI) upon a timeout by the watchdog timer.

WD_IE bit is used to enable / disable an NMI that is generated upon a watchdog timeout. To clear a pending NMI caused by a watchdog timeout, this bit must be cleared. To enable subsequent NMIs the bit has to be set again.

The following tables show the register layout and the bit settings

WD_CTRL2 (Address: $F F124 0013)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	WD IE	WD PROT	POWER UP RESET
r	r	r	r	r	r/w	r/w	r/w

WD_IE	**NMI on Watchdog Timeout**	**Default**
0	Disabled / Cleared	*
1	Enabled	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.10.2.6 The NMI Pending Bits

There are two registers which contain the pending bits of the possible NMI sources: the NMI_PEND register and the SIGNAL_STAT register. Both are described here.

The NMI_PEND Register shows the status of the following pending NMIs: the NMI upon toggling the ABORT Key, the NMI upon assertion of ACFAIL, the NMI upon assertion of SYSFAIL, the NMI upon negation of SYSFAIL and the NMI upon watchdog timeout.

The following tables show the register layout and the bit settings.

NMI_PEND (Address: $F F124 0014)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	ABORT IP	ACFAIL IP	SYSFAIL ASSERT IP	SYSFAIL NEGATE IP
r	r	r	r	r	r	r	r

ABORT_IP	**ABORT NMI Pending**
0	Yes
1	No

ACFAIL_IP	**ACFAIL NMI Pending**
0	Yes
1	No

SYSFAIL_ASSERT_IP	**SYSFAIL_ASSERT NMI Pending**
0	Yes
1	No

SYSFAIL_NEGATE_IP	SYSFAIL_NEGATE NMI Pending
0	Yes
1	No

The SIGNAL_STAT register contains the interrupt pending bit WD_IP for the NMI caused by watchdog timeout.

SIGNAL_STAT (Address: $F F124 0015)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	WD IP	ABORT STATUS	ACFAIL STATUS	SYSFAIL STATUS
r	r	r	r	r	r	r	r

WD_IP	WD NMI Pending
0	Yes
1	No

3.11 The Local I/O Devices on the SPARC CPU-10

On the SPARC CPU-10 there are several 8-bit wide devices for local I/O which are controlled by the SEC. These are two dual channel serial controllers 85C30, a high speed 82077SL floppy disk controller, an RTC/NVRAM device, two boot PROM devices, two devices of flash memory for user defined purposes and an LCA (FPGA) for general board control. All the 8-bit wide I/O devices are located on the VME-10 board, which occupies the second of the two VMEbus slots the SPARC CPU-10 uses. All the devices are described in detail in the following chapters.

3.11.1 Address Map of Local I/O Devices

The table below lists the physical addresses for all local I/O devices.

Table 49: Address Map of Local I/O Devices

Physical Address	Device

$F F000 0000 ...
$F F003 FFFF
 	
$F F004 0000 ...
$F F007 FFFF
	Boot EPROM Device 1 (256 Kbyte):
	Read Operation Only
	
	Boot EPROM Device 2 (256 Kbyte):
	Read Operation Only

$F F100 0000 $F F100 0002 $F F100 0004 $F F100 0006 $F F110 0000 $F F110 0002 $F F110 0004 $F F110 0006	Mouse Control Port Mouse Data Port Keyboard Control Port Keyboard Data Port Serial B Control Port Serial B Data Port Serial A Control Port Serial A Data Port
$F F120 0000	RTC / NVRAM
$F F124 0000	LCA
$F F128 0000 ... $F F12F FFFF	Boot EPROM (512 Kbyte) and User EPROM (2 Mbyte max): Read Operation and Programming Operation
$F F170 0000	Floppy Disk Controller
$F F180 0000	Auxiliary 1 Register (Floppy TC and Floppy Density Sense Control)
$F F1A0 1000	Auxiliary 2 Register (Software Powerdown Control)

3.11.2 LCA for Interrupt and General Control

On the SPARC CPU-10 there is an LCA (logic cell array). This LCA is an FPGA and is configured during powerup of the board with the serial bit stream out of a PROM. The LCA serves several purposes:

- Monitoring and Mapping Logic for VME Interrupts and MVICs DMA Interrupt
- VMEbus Timer
- Watchdog Timer
- NMI (Non Maskable Interrupt) Control
- Flash Memory Control
- Front Panel LED Control
- VME SYSFAIL, ACFAIL and SYSRESET Control

The LCAs functions are described in detail in separate chapters. Please refer to the table of contents of this manual to find the respective chapters.

3.11.2.1 Register Map of the LCA

The following table shows the register layout of the LCA.

Table 50: Register Map of the LCA

Register	Physical Address	Reset Value	Access
MVIC_DMA_IRQ_MAP	$F F124 0000	$F8	r/w
VME_IRQ1_MAP	$F F124 0001	$F8	r/w
VME_IRQ2_MAP	$F F124 0002	$F8	r/w
VME_IRQ3_MAP	$F F124 0003	$F8	r/w
VME_IRQ4_MAP	$F F124 0004	$F8	r/w
VME_IRQ5_MAP	$F F124 0005	$F8	r/w
VME_IRQ6_MAP	$F F124 0006	$F8	r/w
VME_IRQ7_MAP	$F F124 0007	$F8	r/w
LED1_CTRL	$F F124 0008	$FE	r/w
LED2_CTRL	$F F124 0009	$FE	r/w

Table 50: Register Map of the LCA (Continued)

Register	Physical Address	Reset Value	Access
ACFAIL_CTRL	$F F124 000A	$FE	r/w
SYSFAIL_CTRL	$F F124 000B	$FF	r/w
SYSFAIL_IRQ_CTRL1	$F F124 000C	$FE	r/w
SYSFAIL_IRQ_CTRL2	$F F124 000D	$FE	r/w
ACFAIL_IRQ_CTRL	$F F124 000E	$FE	r/w
ABORT_IRQ_CTRL	$F F124 000F	$FE	r/w
FLASH_CTRL1	$F F124 0010	$F0	r/w
WD_CTRL1	$F F124 0011	$F0	r/w
BUS_TIMEOUT_CTRL	$F F124 0012	$F0	r/w
WD_CTRL2	$F F124 0013	$F8	r/w
NMI_PEND	$F F124 0014	$FF	r
SIGNAL_STAT	$F F124 0015	$FE	r
RESET_STAT	$F F124 0016	$F0	r
ROTARY_SWITCH_STAT	$F F124 0017	$F0	r
BOOT_ROM_SIZE_CTRL	$F F124 0018	$FE	r/w
FLASH_CTRL2	$F F124 0019	$FE	r/w
FLASH_VPP_CTRL	$F F124 001A	$FE	r/w
WD_RESET_CTRL	$F F124 001B	$FF	w
SYSRESET_CTRL	$F F124 001C	$FF	w
RESERVED2	$F F124 001D	not affected	not affected
RESERVED3	$F F124 001E	not affected	not affected
LCA_ID	$F F124 001F	$Fx	r

3.11.3 Boot EPROM

The boot EPROM consists of two 2-Mbit (256K*8) flash EPROM devices or one 4-Mbit (512K*8) flash EPROM device. The devices are installed in sockets J6 and J7, wherein J6 holds boot device # 1 and J7 holds boot device # 2. The default assembly is two 2-Mbit devices. For the location of the boot EPROM devices on the board, please see “Highlighted Location Diagram of the VME-10” on page 15.

The boot EPROM can be accessed in two separate address ranges: $F F000 0000 ... $F F007 FFFF for read operation and $F F128 0000 ... $F F12F FFFF for read and write operation.

Table 51: Physical Address of Boot EPROM

Physical Address	Device

$F F000 0000 ...
$F F003 FFFF
 	
$F F004 0000 ...
$F F007 FFFF
	Boot EPROM Device 1 (256 Kbyte):
	Read Operation Only
	
	Boot EPROM Device 2 (256 Kbyte):
	Read Operation Only

$F F128 0000 ... $F F12F FFFF	Boot EPROM (512 Kbyte) and User EPROM (2 Mbyte max): Read Operation and Programming Operation

Table 52: Boot EPROM Capacity

Devices	Count	Capacity	Default
256 K * 8	2	512 Kbyte	*
512 K* 8	1	512 Kbyte	

3.11.3.1 Using Standard PROMs instead of Flash EPROM Devices

In some applications reprogramming the boot EPROM is not necessary. In that case, the user may choose to install standard PROMs / EPROMs instead of the flash EPROM devices. To install standard PROMs / EPROMs, the default switch setting of SW4-1 on the VME-10 board has to be modified. The table below shows the switch setting for the two types of devices.

Table 53: Boot EPROM Device Selection

SW4-1	Boot EPROM Devices	Default
OFF	Flash EPROMs	*
ON	Standard PROMs/EPROMs	

WARNING: You must set SW4-1 to ON and SW4-2 to OFF when using standard PROMs instead of the flash memory devices. If these switches are not correctly set, the chip can be damaged.

3.11.3.2 Boot EPROM Write Protetion

When flash EPROM devices are installed for the boot EPROM, the devices can be reprogrammed on-board and can be write protected via SW4-2 on the VME-10 board. The following table shows the switch setting of SW4-2 for the write protection of the boot EPROMs.

Table 54: Boot EPROM Write Protection

SW4-2	Write Protection	Default
OFF	Yes	*
ON	No	

The boot EPROM devices can easily be removed from the board since they are located in sockets. This permits programming them in a standard EPROM programmer, which may be necessary if the power fails during onboard reprogramming. In that case, the contents of the boot EPROM would be lost, the board would not be able to boot, and the boot EPROM devices would have to be reprogrammed on an EPROM programmer.

3.11.4 User EPROM

The user flash EPROM area consists of a maximum of two 8-Mbit (1M*8) flash memory devices. One device is assembled in the default configuration. This area can be used to store ROMable operating systems as well as application specific code. The user EPROM can be read and written in the following address range: $F F128 0000 .. $F F12F FFFF.

Table 55: Address Map of User EPROM

Physical Address	Device
$F F128 0000 ... $F F12F FFFF	Boot EPROM (512 Kbyte) and User EPROM (2 Mbyte max): Read Operation and Programming Operation

Table 56: User EPROM Capacity

Devices	Count	Capacity	Default
1M*8	1	1 Mbyte	*
1M*8	2	2 Mbyte	

The user EPROM devices can be reprogrammed on-board and can be write protected via SW4-3 on the VME-10. The following table shows the switch setting of SW4-3 for the write protection of the user EPROMs.

Table 57: User EPROM Write Protection

SW4-3	Write Protection	Default
OFF	yes	*
ON	no	

3.11.5 Control of Flash Memory (Boot EPROM and User EPROM)

On the CPU-10 there is a maximum of 2.5 Mbyte flash memory available (512 Kbyte boot EPROM and 2 Mbyte user EPROM). The 2.5 Mbyte flash memory can be accessed in 512 Kbyte pages (programming window) for read and write operation. Additionally, the boot EPROM is accessible in the boot EPROM address range for read only operations. The size of the programming window is limited to 512 Kbyte, which means that only 512 Kbyte of the whole 2.5 Mbyte flash memory area can be accessed contiguously.

The BOOT_ROM_SIZE_CTRL register controls the decoding of the boot EPROMs. The FLASH_CTRL1 and FLASH_CTRL2 registers are used to map 512 Kbyte of the whole 2.5 Mbyte flash memory area into the programming window. The FLASH_VPP_CTRL register is used to control the +12V programming voltage.

The following tables show the register layout and describe the function of the register bits. For mapping 512 Kbyte out of the whole 2.5 Mbyte flash memory area into the programming window, the register bits have to be set according to table “Flash Memory Selection Control Bits” on page 117. This table provides a description of the bit settings which are relevant to programming the flash memories.

The FLASH_CTRL1 Register

Bits **A[21:19]** of the FLASH_CTRL1 register are used to control the address lines A21 ... A19 of the user EPROMs. With the help of these address lines, a specific 512 Kbyte page is selected from the user EPROM area and mapped into the 512 Kbyte programming window. This allows addressing user EPROM of up to 4 Mbyte per device.

Bit **SELECT_ROM** of the FLASH_CTRL1 register chooses between device #1 and device #2 of either the two boot EPROM devices or the two user EPROM devices. When SELECT_ROM is set to "0" then device #1 is selected for programming. When SELECT_ROM is set to "1" then device #2 is selected for programming.

FLASH_CTRL1 (Address: $F F124 0010)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	A[21:19]			SELECT ROM
r	r	r	r	r/w			r/w

The FLASH_CTRL2 Register

Bit **BOOT_ROM_SELECT** of the FLASH_CTRL2 register chooses between the boot EPROM and the user EPROM. When BOOT_ROM_SELECT is set to "1" then the boot EPROM devices are selected for programming. When BOOT_ROM_SELECT is set to "0" then the user EPROM devices are selected for programming.

FLASH_CTRL2 (Address: $F F124 0019)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	BOOT ROM SELECT
r	r	r	r	r	r	r	r/w

The BOOT_ROM_SIZE_CTRL Register

Bit **256KSEL** of the BOOT_ROM_SIZE_CTRL register controls the decoding of the boot EPROMs. When two 256 Kbyte devices are installed (which is the default configuration of the boot EPROM), the bit 256KSEL has to be set to "1". When one 512 Kbyte device is installed, bit 256KSEL has to be set to "0".

Note: The setting of bit 256KSEL of the BOOT_ROM_SIZE_CTRL register is not only relevant for the programming address range $F F128 0000 ... $F F12F FFFF, but also for the boot address range $F F000 0000 ... $F F007 FFFF. For proper operation of the boot EPROM, the bit 256KSEL always has to be set to the right value.

BOOT_ROM_SIZE_CTRL (Address: $F F124 0018)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	256KSEL
r	r	r	r	r	r	r	r/w

3.11.5.1 Flash Memory Selection Control Bits

The table below provides a description of bit settings of the register bits described on the previous pages. It shows the possible bit settings for programming the flash memory on the CPU-10.

Table 58: Flash Memory Selection Control Bits

256KSEL	BOOT ROM SELECT	SELECT ROM	A21	A20	A19	Programs
1	1	0	X	X	X	Boot EPROM Dev#1 with 256 Kbyte
1	1	1	X	X	X	Boot EPROM Dev#2 with 256 Kbyte
0	1	0	X	X	X	Boot EPROM Dev#1 with 512 Kbyte
X	0	0	X	X	0	User EPROM first 512 Kbyte of Dev#1
X	0	0	X	X	1	User EPROM second 512 Kbyte of Dev#1
X	0	1	X	X	0	User EPROM first 512 Kbyte of Dev#2
X	0	1	X	X	1	User EPROM second 512 Kbyte of Dev#2
0	0	0	0	0	0	reset setting

X = Don’t Care

By default only one 1 Mbyte device of user EPROM is installed, which means that only bit A19 is relevant in the selection of one of two 512 Kbyte pages. Bits A20 and A21 are don’t care bits. A20 and A21 are for future use when 2 Mbyte or 4 Mbyte devices are installed.

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.11.5.2 Control of the Programming Voltage

VPPON bit in the FLASH_VPP_CTRL register controls the +12V programming voltage for the flash memories. The following tables show the register layout and the bit settings.

FLASH_VPP_CTRL (Address: $F F124 001A)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	VPPON
r	r	r	r	r	r	r	r/w

VPPON	Programming Voltage	Default
0	Off	*
1	On	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.11.6 RTC / NVRAM

The MK48T08 combines an 8K * 8 full CMOS SRAM, a bytewide accessible Real Time Clock, a crystal, and a long life lithium carbon monofluoride battery, all in a single plastic DIP package. The MK48T08 is a nonvolatile 8K * 8 SRAM, which is pin and functionally equivalent to any Jedec standard. Approximately 2 Kbyte of the SRAM is used by OpenBoot for storing system parameters. The remaining 6 Kbyte are available for user purposes.

For a detailed description of the RTC / NVRAM, please see the respective data sheet.

Table 59: Physical Address of RTC / NVRAM

Physical Address	Device
$F F120 0000	RTC / NVRAM

3.11.7 Serial I/O Ports

The serial I/O ports of the SPARC CPU-10 are located on the VME-10 board, which occupies the second VMEbus slot the SPARC CPU-10 uses. The two serial I/O ports are available on the front panel via two 26-pin shielded connectors, which are compatible to the CPU-2CE and CPU-3CE. The pinout of the keyboard and mouse connector on the front panel is described in the chapter “Keyboard/Mouse Connector Pinout” on page 47.

The ports are controlled with one of the two dual channel serial controllers (85C30) installed on the CPU-10. The physical address map for the serial ports is shown in the following table. For detailed information about programming the serial ports please refer to the respective data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10.*

Table 60: Physical Address of Serial I/O Ports

Physical Address	Device
$F F110 0000	Serial B Control Port
$F F110 0002	Serial B Data Port
$F F110 0004	Serial A Control Port
$F F110 0006	Serial A Data Port

Both serial ports are available with four signals (RXD, TXD, RTS, CTS) per port via the VMEbus P2 connector on the VME-10 board The two serial I/O ports are independent full-duplex ports. With FORCE Computers’ back panel IOBP-10 installed on the VMEbus P2 connector of the VME-10 board, the two serial I/O ports are available on a 14-pin flat cable connector.

The IOBP-10 and the pin assignment of the serial ports are described in “The IOBP-10 Connectors” on page 56 and in “IOBP-10 P5 Pinout (Serial / Audio)” on page 62.

3.11.7.1 RS-232, RS-422 or RS-485 Configuration

Both serial ports can be configured as RS-232, RS-422 or RS-485. By default, the serial port is installed for RS-232 operation. RS-422 and RS-485 can be configured with termination resistors.

In order to simplify changing the serial interfaces, FORCE COMPUTERS has developed RS-232, RS-422 and RS-485 hybrid modules: the FH-002, FH-003 and FH-005. These 21-pin SIL modules are installed in sockets so that they may be easily changed to meet specific application needs.

To change the configuration of serial port A, insert the respective hybrid in socket J14. To change the configuration of serial port B, insert the respective hybrid in socket J15. For the position of the sockets on the board, please see“Highlighted Location Diagram of the VME-10” on page 15.

Table 61: RS-232, RS-422 or RS-485 Configuration

Hybrid	Configuration	Socket for Serial Port A	Socket for Serial Port B
FH-002	RS-232	J14	J15
FH-003	RS-422	J14	J15
FH-005	RS-485	J14	J15

3.11.7.2 RS-232 Hardware Configuration

The serial ports A and B are configured by default for RS-232 operation. The following individual I/O signals are available for serial ports A and B on the front panel connectors.

Table 62: Serial Ports A and B Pinout List (RS-232)

Pin	Transmitted Signals	Pin	Received Signals
2	TXD-Transmit Data	3	RXD-Receive Data
4	RTS-Request to Send	5	CTS-Clear to Send
7	Ground	6	SYNC
20	DTR-Data Terminal Ready	8	DCD-Data Carrier Detect
24	TRXC-DTE Transmit Clock	15	TRXD-DCE Transmit Clock
		17	RTXC-DCE Receive Clock

The table below shows the switch settings for each port.

Table 63: Switch Settings for Ports A and B (RS-232)

Port A	Port B	Default	Function for RS-232
SW5-1	SW6-1	ON	TRXC is available on front panel connectors, pin 24
SW5-2	SW6-2	OFF	Off for RS-232
SW5-3	SW6-3	OFF	Off for RS-232
SW5-4	SW6-4	ON	RTS is available on front panel connectors, pin 4
SW5-5	SW6-5	ON	CTS is available on front panel connectors, pin 5
SW5-6	SW6-6	OFF	Off for RS-232

CAUTION: To avoid damaging the serial ports, please consider the following regarding Switch 5 and Switch 6. Do not set the switches (SW5-1 and SW5-2), or (SW5-3 and SW5-4), or (SW5-5 and SW5-6) to ON at the same time and do not set the switches (SW6-1 and SW6-2), or (SW6-3 and SW6-4), or (SW6-5 and SW6-6) to ON at the same time! Please see "Highlighted Location Diagram of the VME-10" on page 15 for the location of the switches on the board.

3.11.7.3 RS-422 Hardware Configuration

It is possible to reconfigure serial ports A and B for RS-422 operation. In order to configure the serial ports to RS-422, the hybrid module FH-003 must be used. Termination resistors can be installed to adapt various cable lengths and reduce reflections.

Table 64: Serial Ports A and B Pinout List (RS-422)

Pin	Transmitted Signals	Pin	Received Signals
24	TXD+ Transmit Data	20	RXD+ Receive Data
8	TXD- Transmit Data	7	RXD- Receive Data
4*	RTS+ Request to Send	2*	CTS+ Clear to Send
3*	RTS- Request to Send	5*	CTS- Clear to Send
4*	TRXC+ Transmit Clock	2*	RTXC+ Receive Clock
3*	TRXC- Transmit Clock	5*	RTXC- Receive Clock

* Signals RTS and TRXC can be switched so that they are available on connector pins 3 and 4. Signals CTS and RTXC can also be switched so that they are available on connector pins 2 and 5. This is done by switch SW5 for port A and by switch SW6 for port B.

The table on the next page shows the corresponding switch settings.

Table 65: Switch Settings for Ports A and B (RS-422)

Port A	Port B	Default	Function for RS-422
SW5-1	SW6-1	ON	ON for RS-422
SW5-2	SW6-2	OFF	OFF for RS-422
SW5-3	SW6-3	OFF	TRXC +/- on front panel connectors, pins 3 and 4 ON = available OFF =Not available
SW5-4	SW6-4	ON	RTS +/- on front panel connectors, pins 3 and 4 ON = available OFF =Not available
SW5-5	SW6-5	ON	CTS +/- on front panel connectors, pins 2 and 5 ON = available OFF =Not available
SW5-6	SW6-6	OFF	RTXC +/- on front panel connectors, pins 2 and 5 ON = available OFF =Not available

CAUTION: To avoid damaging the serial ports, please consider the following regarding Switch 5 and Switch 6. Do not set the switches (SW5-1 and SW5-2), or (SW5-3 and SW5-4), or (SW5-5 and SW5-6) to ON at the same time and do not set the switches (SW6-1 and SW6-2), or (SW6-3 and SW6-4), or (SW6-5 and SW6-6) to ON at the same time!

Please see “Highlighted Location Diagram of the VME-10” on page 15 for the location of the switches on the board.

3.11.7.4 RS-485 Hardware Configuration

It is possible to reconfigure serial ports A and B to be RS-485 compatible by using the hybrid module FH-005. Termination resistors can be installed to adapt various cable lengths and reduce reflections.

The following I/O signals are available on the front panel connectors of both serial ports.

Table 66: Serial Ports A and B Pinout List (RS-485)

Pin	Signals
7 20	RXTX+ Receive/Transmit Data RXTX- Receive/Transmit Data

The Receive-Enable (REN) and Transmit-Enable (TEN) of the hybrid module FH-005 are controlled via the serial I/O signals DTR (REN) and RTS(TEN).

The following table shows the corresponding switch settings

Table 67: Switch Settings for Ports A and B (RS-485)

Port A	Port B	Configuration	Function for RS-485
SW5-1	SW6-1	OFF	OFF for RS-485
SW5-2	SW6-2	ON	RTS functions as TEN
SW5-3	SW6-3	OFF	No function for RS-485
SW5-4	SW6-4	OFF	No function for RS-485
SW5-5	SW6-5	OFF	No function for RS-485
SW5-6	SW6-6	OFF	No function for RS-485

Please see “Highlighted Location Diagram of the VME-10” on page 15 for the location of the switches on the board.

3.11.7.5 Transmission Line Termination

For the termination of the RS-422/RS-485 transmission lines, the SPARC CPU-10 provides SIL sockets in which resistor networks or single RGU resistors (2,54 mm grid) can be installed.

The line termination is designed for networks in a 10-pin package with one common pin (pin #1) for all resistors and separate pins for the single resistors of the network.

The table below shows the relationship between the serial ports and their termination sockets.

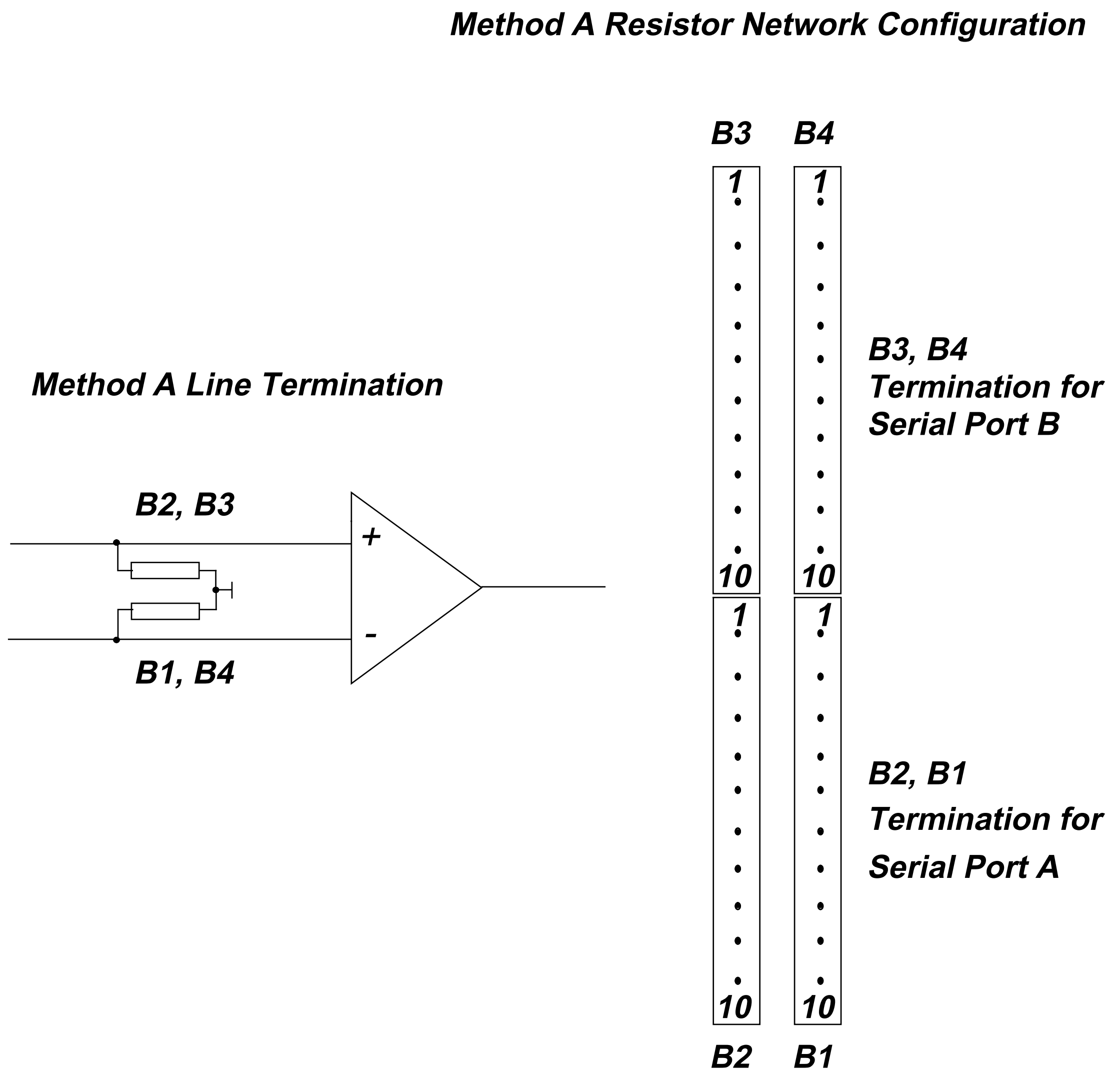
Table 68: Transmission Line Termination

Serial Port	Socket	Terminates
Serial Port A	B1	Inverted Signals
Serial Port A	B2	Non-inverted Signals
Serial Port B	B4	Inverted Signals
Serial Port B	B3	Non-inverted Signals

There are three methods for line termination and resistor network configuration. These three methods, labeled Method A, B, and C, are depicted on the following pages.

For the position of sockets B1, B2, B3, and B4, please see “Highlighted Location Diagram of the VME-10” on page 15.

FIGURE 20. Method A Line Termination & Resistor Network



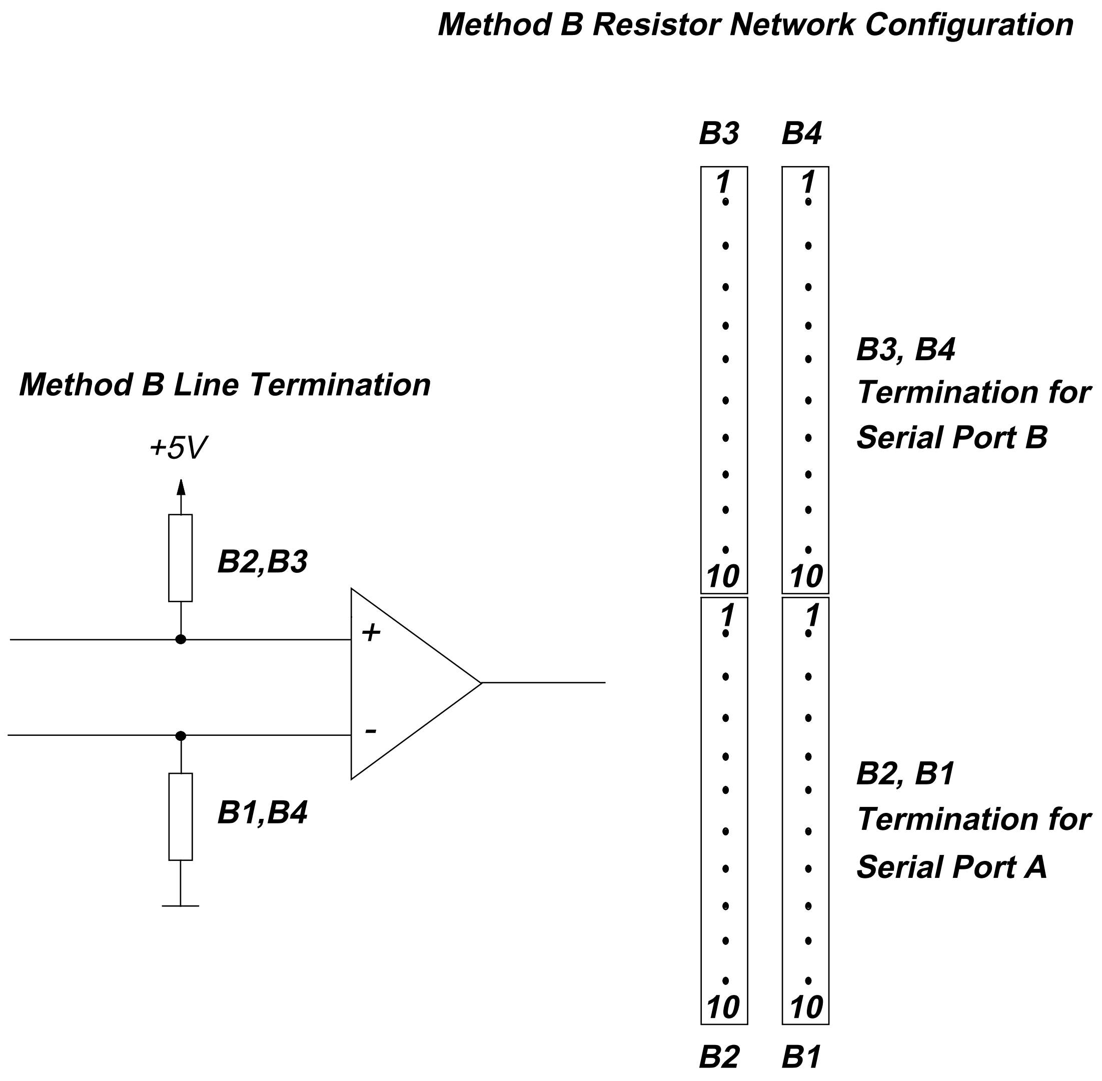
This **method A** requires resistor networks to be used for the transmission line termination.

All networks must be installed in the same direction with network pin #1 matching pin #1 of the socket.

Resistor networks, which are installed at location B1 and B2, terminate the serial port A. Serial port B is terminated by using the socket locations B3 and B4.

This allows terminating the serial ports A and B with different termination resistors.

FIGURE 21. Method B Line Termination & Resistor Network

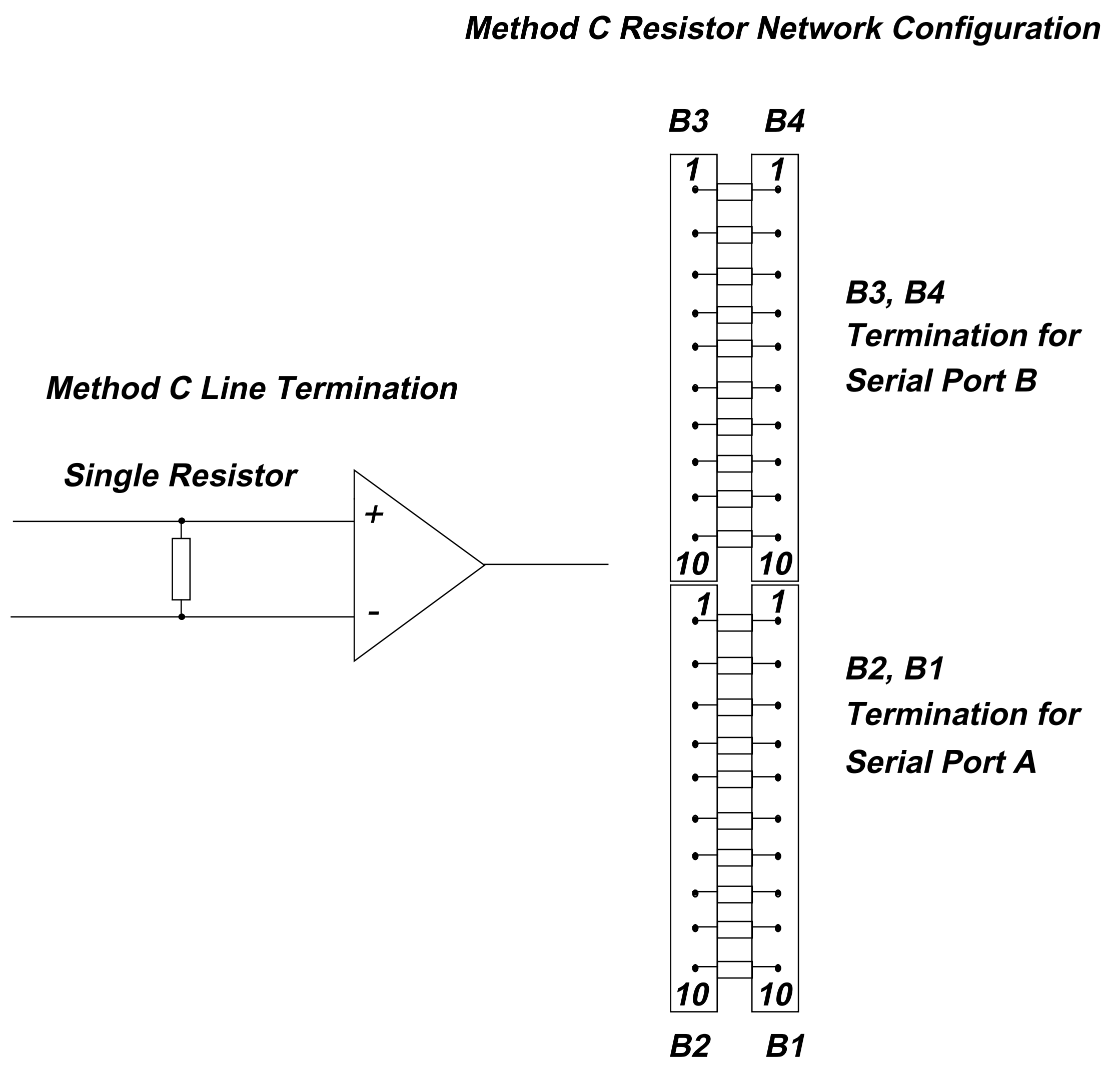

Method B Resistor Network Configuration
B3
B4
1
10
B3, B4 Termination for Serial Port B
Method B Line Termination
+5V
B2,B3
+
-
B1,B4
B2, B1 Termination for Serial Port A
B2
B1

This **method B** requires resistor networks to be used for the transmission line termination. Using this method requires installing the networks at location B2 and B3 **reversed**. This means that the networks must be plugged into the socket so that their pin #1 is inserted into pin #10 of the socket.

Installed in this way, the non-inverting inputs of the receiver circuitry are terminated to +5V and the inverted inputs are terminated to GND.

The serial port A is terminated with resistor networks installed at location B1 and B2. Serial port B is terminated by using the socket locations B3 and B4. This allows terminating the serial ports A and B with different termination resistors.

FIGURE 22. Method C Line Termination & Resistor Network



This **method C** uses single RGU resistors with 2.54 mm (0.100”) grid. This method allows to terminate each serial port individually.

The resistors must be installed into the sockets as shown in the diagram. The pins #1 and #10 of the sockets need not be assembled since they have GND and + 5V connected.

The table on the next page shows in which pins of neighboring sockets a single resistor has to be plugged in to terminate the corresponding interface signal according to method C.

Table 69: Signal Resistor Termination (Method C) for RS-422

Signal Transmission Line	Serial Port	Resistor inserted between B2-B1	Signal Transmission Line	Serial Port	Resistor inserted between B3-B4
RxD+/-	A	Pin2-Pin2	RxD+/-	B	Pin2-Pin2
TxD+/-	A	Pin3-Pin3	TxD+/-	B	Pin3-Pin3
RTS+/-* TRXC+/-*	A	Pin4-Pin4	RTS+/-* TRXC+/-*	B	Pin4-Pin4
CTS+/-* RTXC+/-*	A	Pin5-Pin5	CTS+/-* RTXC+/-*	B	Pin5-Pin5

Note: The signals marked with * are alternately available on the network resistor sockets depending on on the switch setting of serial ports A and B. The switch settings for serial ports A and B are outlined under their respective chapters.

Table 70: Signal Resistor Termination (Method C) for RS-485

Signal Transmission Line	Serial Port	Resistor inserted between B2-B1	Signal Transmission Line	Serial Port	Resistor inserted between B3-B4
RXTX+/-	A	Pin2-Pin2	RXTX+/-	B	Pin2-Pin2

3.11.8 Keyboard and Mouse Port

The keyboard and mouse port is available on the front panel via an 8-pin mini DIN connector. The pinout of the keyboard and mouse connector on the front panel is described in “Keyboard/ Mouse Connector Pinout” on page 47.

The keyboard and mouse port is controlled via one of the two dual channel serial controllers (85C30) installed on the CPU-10, where channel A is for keyboard control and channel B for mouse control. The physical address for the keyboard and mouse port is shown in the following table. For detailed information about programming the port, please refer to the respective data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

Table 71: Physical Address of Keyboard / Mouse Port

Physical Address	Device
$F F100 0000	Mouse Control Port
$F F100 0002	Mouse Data Port
$F F100 0004	Keyboard Control Port
$F F100 0006	Keyboard Data Port

Both the keyboard and mouse port is available with four signals (KEYBOARD_IN, KEYBOARD_OUT, MOUSE_IN, MOUSE_OUT) via the VMEbus P2 connector on the VME-10 board. The four signals are routed to the VMEbus P2 connector via 0-Ohm resistors. By default, the resistors are **not** assembled. With the resistors assembled and with FORCE COMPUTERS’ back panel IOBP-10 installed on the VMEbus P2 connector of the VME-10 board, the port is available on a 14-pin flat cable connector.

The IOBP-10 is described in detail in “The IOBP-10 Connectors” on page 56 and the pin assignment of the connector is described in “Keyboard/Mouse Connector Pinout” on page 47.

3.11.9 Floppy Disk Interface

On the SPARC CPU-10 the floppy disk interface is realized with the 82077SL floppy disk controller. The 82077SL is able to transfer data with data rates of 250, 300, 500 Kbit/sec and 1 Mbit/sec. It integrates drivers, receivers, a data separator, and a 16-byte bidirectional FIFO. The floppy disk controller supports all standard disk formats (typically 720 Kbyte and 1.44 Mbyte floppies) and also the 2.88 MByte and 4 MByte floppy format.

The floppy disk interface on the CPU-10 has two additional control pins. The **FLOPPY_TC** pin, which is an input pin on the 82077SL, is used to terminate the current disk transfer. The pin is controlled via bit D[2] of the SEC's Auxiliary 1 register. The **FLOPPY_DENSENSE** pin, which is an output pin on the floppy drive, is used to indicate whether the drive is high density or low density. The status of the signal can be read via bit D[5] of the SEC's Auxiliary 1 register.

The physical address for the floppy disk interface is shown in the following table. For detailed information about programming the interface, please refer to the respective data sheet.

Table 72: Physical Address of Floppy Disk Interface

Physical Address	Device
$F F170 0000	Floppy Disk Controller 82077SL
$F F180 0000	Auxiliary 1 Register (Floppy TC and Floppy Density Sense Control)

AUXILIARY 1 REGISTER (Address: $F F180 0000)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	FLOPPY DEN SENSE	1	UNUSED ON CPU-10	FLOPPY TC	UNUSED ON CPU-10	UNUSED ON CPU-10
r	r	r	r	r/w	r/w	r/w	r/w

The floppy disk interface is available on the VME P2 connector on the VME-10 board of the CPU-10. With FORCE Computers back panel IOBP-10 installed on the VMEbus P2 connector of the VME-10 board the floppy disk interface is available on a standard 34-pin flat cable connector. The IOBP-10 and the pin assignment of the floppy connector are described in "The IOBP-10 Connectors" on page 56.

3.12 The NCR89C100 (MACIO)

The NCR89C100 is an SBus I/O chip providing SCSI, Ethernet and Centronics parallel interfaces. It is located on the I/O-10 board which occupies the first one of the two VME slots the SPARC CPU-10 uses. The NCR89C100 has full SBus master capabilities. It can be addressed as an SBus slave at address $E F000 0000 physically.

The NCR89C100 integrates high performance I/O macrocells and logic including an Ethernet controller core, a fast 53C9X SCSI core, a high-speed parallel port, a DMA2 controller and an SBus interface.

The Ethernet core is compatible with the industry standard 7990 Ethernet controller. The SCSI core is a superset of the industry standard NCR53C90A which has been modified to support fast SCSI. The uni/bi-directional parallel port is Centronics compliant and can operate in either programmed I/O or DMA mode.

The DMA2 block comprises the logic used to interface each of these functions to the SBus. It provides buffering for each of the functions. Buffering takes the form of a 64-byte data cache and 16-bit wide buffer for the Ethernet channel, and a 64-byte FIFO for both the SCSI channel and the parallel port. The DMA2 incorporates as improved cache and FIFO draining algorithm which allows better SBus utilization than previous DMA implementations.

Table 73: Physical Address of the NCR89C100

Physical Address	Device
$E F000 0000	NCR89C100 (MACIO) Ethernet, SCSI, Centronics

Features of the NCR89C100 on the SPARC CPU-10

- Fast 8-bit SCSI
    - Supports fast SCSI mode
    - Backward compatible to 53C90A
- 7990-compatible Ethernet
- Parallel Port
    - I/O or DMA programmable modes
    - Centronics compatibility
- LS64854-compatible DMA2 Controller
- Glueless 20 MHz SBus Interface
- Concurrently supports:
    - 10 Mbyte/sec SCSI transfers
    - 3.3 Mbyte/sec Parallel port transfers
    - 1.25 MByte/sec Ethernet transfers
- 64-byte FIFO for SCSI and Parallel Port data
- Supports SBus burst modes
    - 4-word, 8-word and "no/burst"
- Packaged in 160 pin PQFP

For further information about the NCR89100, please see The NCR89C100 (MACIO) data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

3.12.1 SCSI

The SCSI interface provides a standard interface to a wide variety of mass storage devices, such as hard disks, tapes and CD-ROMs. The SCSI transfers up to 10 Mbytes per second.

The SPARC CPU-10 board's SCSI is realized via the NCR89C100. The NCR89C100 has on-chip 48 mA drivers and therefore provides direct drive of single ended SCSI bus. The SCSI core is a superset of the industry standard NCR53C90A which has been modified to support fast SCSI.

The SCSI interface is single ended and supports "TERMPWR". The NCR89C100 DMA2 core is able to transfer the data to and from the shared main memory.

The SCSI interface is located on the I/O-10 board, which occupies the first of the two VMEbus slots that SPARC CPU-10 uses. By default, all signals of the SCSI interface are routed to the VMEbus P2 connector. This connection is compatible to the CPU-2CE and the CPU-3CE.

NOTE: For an understanding of the usage of the SCSI interface, it is important to read **all** of the chapters here which describe the SCSI interface. Do not forget to check the following important factors: where the SCSI signals are located (for example, endpoint on SCSI bus), the configuration of SCSI termination networks # 1 and # 2, and the position of the SCSI switch matrices.

3.12.1.1 SCSI Termination Networks # 1 and # 2

There are two SCSI termination networks on the I/O-10 board of the SPARC CPU-10. **SCSI Termination Network # 1** is located near the VMEbus P2 connector and **SCSI Termination Network # 2** is located near the front panel SCSI connector. For the position of SCSI termination networks on the board, please see the figure "Location of Devices Used to Configure SCSI" on page 136.

The **SCSI Termination Network # 1** can be enabled and disabled via switch SW1-1 on the I/O-10 board. When SW1-1 is ON, the termination network # 1 is enabled. When SW1-1 is OFF, the termination network # 1 is disabled.

The following table shows the configurations of SCSI Termination Network # 1.

Table 74: SCSI Termination Network # 1

SW1-1	SCSI Termination Network # 1	Default
OFF	**Disabled**	*
ON	**Enabled**	

The **SCSI Termination Network # 2** is enabled and disabled automatically as described here. The termination network is enabled when there is no SCSI cable plugged into the front panel connector. The termination network is disabled when there is a SCSI cable plugged into the front panel connector.

The following table shows the configurations of SCSI Termination Network # 2.

Table 75: SCSI Termination Network # 2

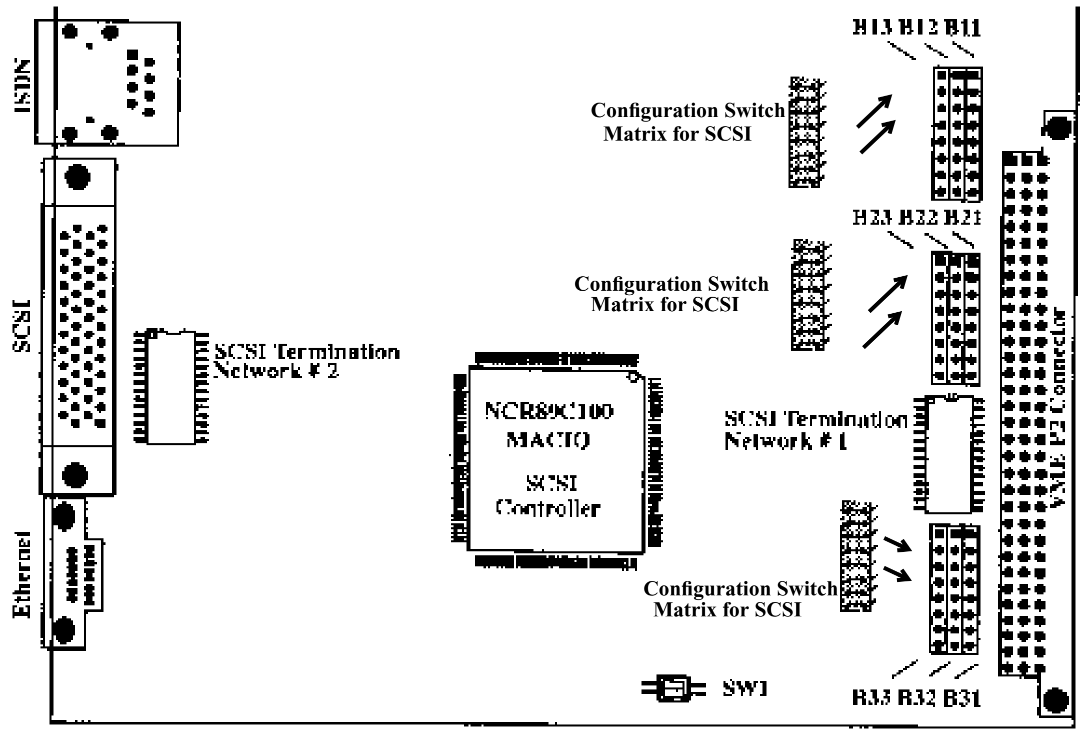
SCSI Cable Plugged into Front Panel Connector	SCSI Termination Network # 2	Default
No	**Enabled** (Automatically Enabled When No SCSI Cable Is Plugged Into Front Panel Connector)	*
Yes	**Disabled** (Automatically Disabled When A SCSI Cable Is Plugged Into Front Panel Connector)	

3.12.1.2 SCSI via Front Panel / VME P2 Connector

All signals of the SCSI interface are routed to the VMEbus P2 connector, which is pin compatible to the CPU-2CE and the CPU-3CE. There are several SCSI configurations possible through the use of three configuration switch matrices. In the table "SCSI Interface Configuration" on page 137, the different SCSI configurations are presented. It is important to check this table in order to correctly configure your SCSI.

The following figure shows the location of the 9 sockets which are used to configure the SCSI interface. Together with the three configuration switch matrices, the sockets used to configure the SCSI interface are B13, B12, B11 and B23, B22, B21 and B33, B32, B31.

FIGURE 23. Location of Devices Used to Configure SCSI


ISDN
SCSI
Ethernet
SCSI Termination Network # 2
NCR89C100 MACIO SCSI Controller
B13 B12 B11
Configuration Switch Matrix for SCSI
B23 B22 B21
Configuration Switch Matrix for SCSI
SCSI Termination Network # 1
Configuration Switch Matrix for SCSI
SW1
B33 B32 B31
VME P2 Connector

3.12.1.3 Possible SCSI Configurations

The SCSI interface may be configured in the following ways.

- SCSI via the VMEbus P2 connector at an endpoint of the SCSI bus (default configuration).
- SCSI via the VMEbus P2 connector but not at an endpoint of the SCSI bus.
- SCSI via the front panel connector at an endpoint of the SCSI bus.
- SCSI via the VMEbus P2 connector and via the front panel connector.
- SCSI not used.

The following table shows the different SCSI configurations, the respective positions of the switch matrices, and the settings of the termination networks #1 and #2. The table also shows the settings when the SCSI interface is not used.

Table 76: SCSI Interface Configuration

Default	SCSI on VMEbus P2	SCSI on Front Panel	Switch Matrices are plugged into the following sockets	SCSI Termination #1 (Via Switch SW1-1)	SCSI Termination #2 (Automatically disabled or enabled)
*	Yes (endpoint on SCSI bus)	No	B12 and B11 B22 and B21 B32 and B31	Disabled (SW1-1 = OFF)	Enabled (No cable plugged into front panel connector)
	Yes (not endpoint on SCSI bus)	No	B13 and B12 B23 and B22 B33 and B32	Disabled (SW1-1 = OFF)	Enabled (No cable plugged into front panel connector). NOTE: In this case, SCSI termination # 2 is not connected with the SCSI interface on VMEbus P2.
	No	Yes (endpoint on SCSI bus)	B12 and B11 B22 and B21 B32 and B31	Enabled (SW1-1 = ON)	Disabled (Cable plugged into front panel connector)
	Yes (not endpoint) on SCSI bus)	Yes (not endpoint on SCSI bus)	B12 and B11 B22 and B21 B32 and B31	Disabled (SW1-1 = OFF)	Disabled (Cable plugged into front panel connector)
	No	No	B12 and B11 B22 and B21 B32 and B31	Enabled (SW1-1 = ON)	Enabled (No cable plugged into front panel connector)

3.12.2 Ethernet

The NCR89C100 DMA controller enables the Ethernet interface to transfer data to and from the main memory. The Ethernet core is register level compatible with the AMD Am7990, Revision F, standard Ethernet controller, which is capable of transferring Ethernet data up to 10 Mbit/sec resulting in 1.25 Mbyte/sec.

3.12.2.1 Ethernet via Front Panel / VME P2 Connector

The Ethernet interface is located on the I/O-10 board, which occupies the first of the two VMEbus slots that SPARC CPU-10 uses. The default configuration provides the Ethernet on the front panel connector. Using an 8-pin configuration switch matrix, the default configuration may be changed so that the Ethernet interface is available via the VME P2 connector. This is done by changing the position of the configuration switch matrix as shown in the table below.

FIGURE 24. Location of Devices Used to Configure Ethernet

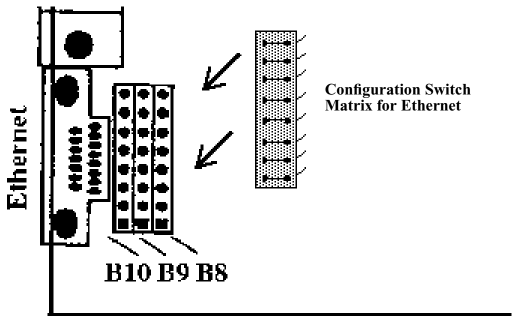


Table 77: Ethernet Configuration

Switch Matrix is plugged into the Following Sockets	Ethernet Signals Via VME P2 Connector	Ethernet Signals Via Front Panel Connector	Default
B9 and B8	Yes	No	
B10 and B9	No	Yes	*

WARNING
When the Ethernet interface is configured via VME P2, do not connect the Ethernet cable at the front panel connector.

3.12.3 The Parallel Port

The parallel port is centronics compliant and provides uni/bidirectional communication. It operates in either programmed I/O or DMA mode.

The parallel port is available via VME P2 connector on the I/O-10 board of the CPU-10. With the FORCE Computers' IOBP-10 back panel installed on the VMEbus P2 connector of the I/O-10 board, the parallel port is available on a standard 40-pin flat cable connector.

For a description of the IOBP-10 back panel, together with its connectors, please see "The IOBP-10 Connectors" on page 56.

For detailed information about the parallel port of the NCR89100, please see the NCR89C100 (MACIO) data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

3.13 T7259 SBus Dual Basic Rate ISDN (DBRI) Transceiver

The T7259 is an SBus I/O chip providing ISDN and audio interfaces. The T7259 is located on the I/O-10 board, which occupies the first one of the two VMEbus slots the CPU-10 uses. The T7259 has full SBus master capabilities. It can be physically addressed as an SBus slave at address $E F800 0000.

The T7259 integrates a dual basic rate ISDN interface, a 16-bit CD quality audio interface and a complete SBus interface with DMA capabilities.

Table 78: Physical Address of the T7259

Physical Address	Device
$E F800 0000	T7259 (DBRI), ISDN, Audio

Features of the DBRI on the SPARC CPU-10

- Complete SBus Interface
- 16-Channel DMA Memory Address Generator and Buffer Manager
- 80 Byte FIFOs per DMA Channel
- Interrupt Queue in System Memory
- Simultaneous operation as both an ISDN TE or an ISDN NT
- Automatic Synchronization of ISDN Interfaces
- Flexible Loopback and Test Modes
- Full Support of AT&T Concentration Highway Interface as an Audio Interface
- Four additional I/O Pins for Control of Audio Interface

3.13.1 The ISDN Interface

The ISDN interface on the SPARC CPU-10 is a T7259 Dual Basic Rate Interface (DBRI). It is compatible to CCITT I.430 and ANSI T1.605 standards for four-wire ISDN 2B+D basic access at the S/T reference point. The two ISDN B channels have a transfer rate of 64 Kbit/sec and the ISDN D channel has a transfer rate of 16 Kbit/sec. The ISDN interface operates as both a terminal endpoint (TE) and as network termination (NT).

The ISDN interface is routed to the RJ-45 connector on the front panel, which holds both the TE and the NT signal lines. Since the pinout of the front panel connector is not compatible with a standard ISDN connector, it is necessary to use the FORCE Computers' ISDN Adapter Kit, which makes the ISDN interface available on two standard ISDN RJ-45 connectors. The ISDN Adapter Kit separates the TE signals from the NT signals and provides both signals on ISDN compatible connectors.

For detailed information about the ISDN interface of the T7259, please refer to the respective data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10*.

Warning: Do not use the ISDN interface without the ISDN Adapter Kit ! The front panel RJ-45 connector pinout is not ISDN compatible. Plugging in an ISDN compatible cable can cause damage to the CPU-10 and the other ISDN participants.

3.13.2 The Audio Interface

The audio interface on the SPARC CPU-10 is included in the T7259 DBRI chip. It provides a 16-bit CD-quality audio interface with up to 48 KHz sampling rate. The audio interface is a digital time-division-multiplexed bus called CHI (Concentration Highway Interface). The CHI is capable of simultaneous input and output of 16-bit stereo audio.

The audio interface is available on the VME P2 connector of the I/O-10 board of the CPU-10. With FORCE Computers' IOBP-10 back panel installed on the VMEbus P2 connector, the audio interface is available on a standard 14-pin flat cable connector. The IOBP-10 and the pin assignment of the audio connector are described in "The IOBP-10 Connectors" on page 56 and in "IOBP-10 P5 Pinout (Serial / Audio)" on page 62.

The audio interface is designed for direct connection to the speakerbox from SUN Microsystems for the SUN SPARC Station 10. An additional cable is required which adapts the speakerbox connector to the 14-pin flat cable connector of the IOBP-10.

For detailed information about the audio interface of the T7259, please refer to the respective data sheet. The data sheets relevant to the SPARC CPU-10 are packaged in the book *Set of Data Sheets for the SPARC CPU-10.*

3.14 Additional Features of the CPU-10

The SPARC CPU-10 provides additional features for diagnostic and control purposes. These features include a hardware watchdog timer, a RESET / ABORT Key and programmable LEDs on the front panel and a hexadecimal rotary switch. The following chapters describe the features in detail.

3.14.1 The Watchdog Timer

The CPU-10 contains a hardware watchdog timer, in addition to the five programmable 32-bit counters/timers located in the L64861 (SEC), which are used for system control functions. The hardware watchdog timer is located in the LCA, which is an FPGA (field programmable gate array). The function of the hardware watchdog timer is to prevent system deadlock.

Under normal conditions when the watchdog timer is enabled, software periodically retriggers the watchdog timer so that no timeout occurs. If software is hung up, then the watchdog timer is not retriggered and, as a result, the watchdog timer generates a non maskable interrupt (NMI) after a specified timeout period. This gives software a chance to react to the NMI by solving the problem and retriggering the watchdog timer. If the watchdog timer is not retriggered after the NMI, a system reset is generated by the watchdog timer after another timeout period.

The following registers, which are described on the next pages, are used for the control of the watchdog timer.

- WD_CTRL1 register
- WD_CTRL2 register
- WD_RESET_CTRL register

3.14.1.1 The Watchdog Enable / Timeout Control

WDENA bit enables / disables the watchdog timer.

WDTIME[2:0] bits select the timeout value for the watchdog timer.

The following tables show the register layout of WD_CTRL1 Register and the possible bit settings. Please also see the description of the WD_CTRL2 and WD_RESET_CTRL registers.

WD_CTRL1 (Address: $F F124 0011)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	WDENA	WDTIME[2:0]		
r	r	r	r	r/w	r/w		

WDENA	Watchdog Timer	Default
0	Disabled	*
1	Enabled	

The table on the next page shows the timeout values for the watchdog timer.

The table below shows the timeout values for the watchdog timer. Note that the timeout values specified are minimum values and the maximum value can be twice the specified timeout value.

WDTIME[2:0]	**Minimum Time until NMI is Asserted**	**Minimum Time until Reset is Asserted**	**NMI to Reset**	**Default**
000	50 ms	75 ms	25 ms	*
001	200 ms	300 ms	100 ms	
010	800 ms	1200 ms	400 ms	
011	3.2 s	4.8 s	1.6 s	
100	12 s	19 s	7 s	
101	50 s	75 s	25 s	
110	3.3 min	5 min	1.7 min	
111	13 min	20 min	7 min	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

WARNING: The VMEbus timer must NOT be enabled while setting the timeout value. Set the timeout value first and then enable the timer.

3.14.1.2 The Watchdog NMI / Watchdog Reset Control

WD_IE bit enables / disables the non maskable interrupt (NMI) that the watchdog timer generates when a timeout occurs. WD_IE is also used to clear a pending watchdog NMI by clearing the WD_IE bit. To enable subsequent watchdog NMIs, the bit has to be set again.

POWERUP_RESET bit controls whether a powerup reset (**with** LCA reconfiguration) or a board reset (**without** LCA reconfiguration) is generated by a watchdog timeout.

WD_PROT is a write protection bit which allows / inhibits write accesses to the WD_CTRL1 and WD_CTRL2 Registers. The purpose of the WD_PROT bit is to prevent the two registers from being modified unintentionally after the watchdog timer has been initialized and enabled. If this bit has been set once, it cannot be modified unless a reset occurs.

The following tables show the register layout of WD_CTRL2 Register and the possible bit settings. Please see also description of the WD_CTRL1 and WD_RESET_CTRL registers.

WD_CTRL2 (Address: $F F124 0013)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	WD IE	WD PROT	POWER UP RESET
r	r	r	r	r	r/w	r/w	r/w

WD_IE	**Watchdog NMI**	**Default**
0	Disabled / Cleared	*
1	Enabled	

WD_PROT	**WD_CTRL1 and WD_CTRL2 Write Protected**	**Default**
0	No	*
1	Yes	

POWERUP_RESET	Reset Generated upon Watchdog Timeout	Default
0	BOARD_RESET (**without** LCA reconfiguration)	*
1	POWERUP_RESET (**with** LCA reconfiguration)	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.14.1.3 The Watchdog Timer Trigger

The WD_RESET_CTRL register is used to retrigger the watchdog timer. Any write access to this register resets the watchdog timer and it starts a new count cycle according to the timeout value specified by the WD_TIME[2:0] bits in the WD_CTRL1 register. Reading the WD_RESET_CTRL Register will always return $FF.

See also description of the WD_CTRL1 and WD_CTRL2 registers.

WD_RESET_CTRL (Address: $F F124 001B)							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	1	1	1	1	1	1	1
w	w	w	w	w	w	w	w

3.14.2 The RESET / ABORT Key

The front panel on the SPARC CPU-10 has a mechanical switch which directly influences the system. When it is toggled **up**, a reset of the sytem can be generated and when it is toggled **down**, a non-maskable interrupt (NMI) to the processor(s) will be generated.

3.14.2.1 The RESET Key Function

When toggled **up**, the RESET / ABORT Key on the front panel can trigger two types of resets. The first reset is called POWERUP_RESET and resets the board and reconfigures the LCA on the CPU-10. The second reset is called BOARD_RESET and does **not** reconfigure the LCA on the CPU-10.

The following table shows the switch settings of SW3-2 and SW3-6 on the VME-10 board for the possible reset configurations.

Table 79: RESET Key Configuration

SW3-2	SW3-6	Reset	Default
OFF	OFF	No Reset Generated	
OFF	ON	BOARD_RESET **without** LCA reconfiguration	
ON*	OFF	POWERUP_RESET **with** LCA reconfiguration	
ON*	ON	POWERUP_RESET **with** LCA reconfiguration	*

* When SW3-2 is set to ON, the setting of SW3-6 does not matter; in both cases, the reset occurs with FPGA reconfiguration.

Besides the front panel switch there are several other devices which are able to trigger a reset. Please see the chapter “The RESET_STAT Register” on page 152.

3.14.2.2 The ABORT Key Function

When toggled **down**, the RESET / ABORT Key on the front panel can trigger a non-maskable interrupt (NMI) to the processor(s). The ABORT key function is controlled via the LCA on the CPU-10. The LCA provides a masking bit and a pending bit for the NMI caused by toggling the ABORT key.

For the detailed description of the ABORT key NMI please refer to chapter “The NMI (Non Maskable Interrupt)” on page 101

3.14.3 The Front Panel LEDs

There is a four-segment LED array on the front panel of the CPU-10. The lower two LEDs can be controlled by software for diagnostic purposes. LED1 is the lower left one of the four-segment LED array and LED2 is the lower right one. The LEDs are marked on the front panel with "1" and "2". The LEDs on the top are marked RUN and BM. The RUN LED turns red during a BOARD RESET and green afterwards. The BM LED indicates that the board is acting as VMEbus master.

3.14.3.1 Control of LED1

LED1 is controlled with LED1_CTRL Register.

LED1 bit in the LED1_CTRL Register is used to turn the LED1 on the front panel ON or OFF. The following tables show the register layout and the bit settings.

LED1_CTRL (Address: $F F124 0008)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	LED1
r	r	r	r	r	r	r	r/w

LED1	**LED status**	**Default**
0	Off	*
1	On	

NOTE: Whenever the BOARD_RESET signal is asserted, the default values in the LCA registers return just as they are shown in the table above. In order to change the values, please see the respective chapters found in Section 5, OpenBoot Enhancements, of this manual.

3.14.3.2 Control of LED2

LED2 is controlled with LED2_CTRL Register.

LED2 bit of the LED2_CTRL Register is used to turn the LED2 on the front panel ON or OFF. The following tables show the register layout and the bit settings.

LED2_CTRL (Address: $F F124 0009)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	1	1	1	LED2
r	r	r	r	r	r	r	r/w

LED2	**LED status**
0	Off
1	On

3.14.4 The Rotary Switch

There is a rotary switch on the VME-10 board of the SPARC CPU-10 and its setting can be read via status register ROTARY_SWITCH_STAT. For the position of the rotary switch on the board, please see the "Highlighted Location Diagram of the VME-10" on page 15.

ROT_SWI[3:0] bits of the ROTARY_SWITCH_STAT resgister reflect the status of the rotary switch on the VME-10 board.

The following tables show the register layout and the register contents

ROTARY_SWITCH_STAT (Address: $F F124 0017)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	ROT_SWI[3:0]			
r	r	r	r	r			

Switch Setting	ROT_SWI[3:0]	Switch Setting	ROT_SWI[3:0]
0	1111	8	0111
1	1110	9	0110
2	1101	A	0101
3	1100	B	0100
4	1011	C	0011
5	1010	D	0010
6	1001	E	0001
7	1000	F	0000

Note: The bit setting is inverted to the values printed on the rotary switch ("0" will be read as 1111 and "F" will be read as 0000).

3.14.5 Additional LCA Status Registers

The LCA contains three additional status registers, the RESET_STAT register, the SIGNAL_STAT register and the LCA_ID register, which are described here.

3.14.5.1 The RESET_STAT Register

Besides the front panel switch there are several other devices which are able to trigger a reset. In the LCA there is a register called the RESET_STAT register. The RESET_STAT register stores the name of the device which triggered the reset. This enables the source of the reset to be identified by reading the register contents after the reset occurs. Bits MVIC_RESET, VMEBUS_RESET, SWITCH_RESET and WD_RESET will be set when the corresponding reset is activated. After the reset source has been identified by software, the bits have to be cleared by writing any value to the register.

The following tables show the register layout and the register contents after the different types of reset.

RESET_STAT (Address: $F F124 0016)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	MVIC RESET	VMEBUS RESET	SWITCH RESET	WD RESET
r	r	r	r	r	r	r	r

Register Contents	Reset Type
$F0	Powerup Reset or Switch Reset (only when VME-10 SW3-2=ON)
$F1	Watchdog Reset
$F2	Switch Reset (only when VME-10 SW3-2=OFF and SW3-6=ON)
$F4	VMEbus SYSRESET
$F8	MVIC Reset

3.14.5.2 The SIGNAL_STAT Register

The SIGNAL_STAT register contains the status bits of the ABORT signal, the VMEbus ACFAIL signal and the VMEbus SYSFAIL signal. These bits directly reflect the current status of the corresponding signals. The SIGNAL_STAT register also contains the pending bit for the NMI caused by watchdog timeout. For information about the watchdog timer, please refer to chapter "Additional Features of the CPU-10" on page 143.

The following tables show the register layout and the register contents.

SIGNAL_STAT (Address: $F F124 0015)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	WD IP	ABORT STATUS	ACFAIL STATUS	SYSFAIL STATUS
r	r	r	r	r	r	r	r

3.14.5.3 The LCA_ID Register

The LCA_ID register provides the 4-bit wide hexadecimal code of the LCA revision on the SPARC CPU-10. The first LCA revision is 1111 and will be counted downwards for LCA revisions released in the future.

LCA_ID (Address: $F F124 001F)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
1	1	1	1	REV[3:0]			
r	r	r	r	r			

SECTION 4 Circuit Schematics

4. Circuit Schematics for the SPARC CPU-10

Copies of the circuit schematics listed below appear on the following pages.

4.1 Circuit Schematics for the I/O-10

4.2 Circuit Schematics for the VME-10

4.3 Circuit Schematics for the MEMORY-10

Circuit Schematics for the I/O-10

Circuit Schematics for the VME-10

Circuit Schematics for the MEMORY-10

SECTION 5 FORCE OpenBoot Enhancements

5. Introduction

This section describes the enhancements to the standard OpenBoot firmware that have been done for the SPARC CPU-10. For a description of standard OpenBoot firmware features, please see the *OPEN BOOT PROM 2.0 MANUAL SET*.

Besides the FORTH words already provided by the standard OpenBoot firmware, the OpenBoot firmware available on the SPARC CPU-10 includes further words for

- accessing and controlling the VMEbus Interface
- accessing and programming available flash memories
- controlling the operating mode of the Watchdog Timer
- making use of the Diagnostics

The following subsections describe these words in detail, and examples are given when it seems necessary to convey the usage of a particular or a group of words. In general, each word is described using the notation stated below:

`name` (*stack-comment*) *description*

The `name` field identifies the name of the word being described.

The stack parameters passed to and returned from a word are described by the *stack-comment* notation — enclosed in parentheses —, and shows the effect of the word on the evaluation stack. The notation used is:

parameters before execution — parameters after execution

The parameters passed and returned to the word are separated by the "—".

The *description* body describes the semantics of the word and conveys the purpose and effect of the particular word.

5.1 VMEbus Interface

The VMEbus Interface on the SPARC CPU-10 consists of two parts: 1.) the MBus to VMEbus Interface Controller (MVIC) MB86986 from Fujitsu Microelectronics; and 2.) various control and status registers included in a LCA.

5.1.1 Generic Information

The FORTH words described below are used to retrieve generic information about the VMEbus interface.

`vmectl` (— *vaddr*) returns the virtual base address of the LCA, which contains further control logic of the VMEbus interface.

`vmectl-pa` (— *paddr*) returns the physical base address of the LCA, which contains further control logic of the VMEbus interface.

`mvic` (— *vaddr*) returns the virtual base address of the MBus to VMEbus Interface Controller (MVIC) MB86986.

`mvic-pa` (— *paddr*) returns the physical base address of the MBus to VMEbus Interface Controller (MVIC) MB86986.

The FORTH words below are used to access various registers within the VMEbus Interface:

`lca-id` (— *vaddr*) returns the virtual address *vaddr* of the LCA's **ID Register**.

`lca-id@` (— *id*) reads the LCA's **ID Register** and returns the *id*.

`midr` (— *vaddr*) returns the virtual address *vaddr* of the **Mbus ID Register**.

`midr@` (— *data*) reads the MVIC's **Mbus ID Register** and returns the *data*.

`mbar` (— *vaddr*) returns the virtual address *vaddr* of the **MBus Base Address Register**.

`mbar@` (— *data*) reads the MVIC's **MBus Base Address Register** and returns the *data*.

`mbar!` (*data* —) stores the *data* in the MVIC's **MBus Base Address Register**.

`vmcr` (— *vaddr*) returns the virtual address *vaddr* of the **VMEbus Master Configuration Register**.

`vmcr@` (— *data*) reads the MVIC's **VMEbus Master Configuration Register** and returns the *data*.

`vmcr!` (*data* —) stores the *data* in the MVIC's **VMEbus Master Configuration Register**.

`vscr` (— *vaddr*) returns the virtual address *vaddr* of the **VMEbus Slave Configuration Register**.

`vscr@` (— *data*) reads the MVIC's **VMEbus Slave Configuration Register** and returns the *data*.

`vscr!` (*data* —) stores the *data* in the MVIC's **VMEbus Slave Configuration Register**.

`vasmbar` (— *vaddr*) returns the virtual address *vaddr* of the **VMEbus A16 Slave MBus Base Address Register**.

`vasmbar@` (— *data*) reads the MVIC's **VMEbus A16 Slave MBus Base Address Register** and returns the *data*.

`vasmbar!` (*data* —) stores the *data* in the MVIC's **VMEbus A16 Slave MBus Base Address** Register.

`va16nmi` (— *vaddr*) returns the virtual address *vaddr* of the **VMEbus A16 Reset and NMI Register**.

`va16nmi@` (— *data*) reads the MVIC's **VMEbus A16 Reset and NMI Register** and returns the *data*.

`va16nmi!` (*data* —) stores the *data* in the **VMEbus A16 Reset and NMI Register**.

`misr` (— *vaddr*) returns the virtual address *vaddr* of the **Interrupt Status Register**.

`misr@` (— *data*) reads the MVIC's **Interrupt Status Register** and returns the *data*.

`.misr` (—) displays the contents of the MVIC's **Interrupt Status Register**.

`vsendi` (— *vaddr*) returns the virtual address *vaddr* of the **VMEbus Send Interrupt Register**.

`vsendi@` (— *data*) reads the MVIC's **VMEbus Send Interrupt Register** and returns the *data*.

`vsendi!` (*data* —) stores the *data* in the MVIC's **VMEbus Send Interrupt Register**.

`vdma-entry` (*page* — *vaddr*) returns the virtual address *vaddr* of a **VDMA Translation Base Address Register** identified by its *page* number. The pages are numbered beginning from zero to 255.

`vdma-entry@` (*page* — *data*) reads a the contents of a **VDMA Translation Base Address Register** identified by its VDMA *page* number and returns the *data*. The pages are numbered beginning from zero to 255.

`vdma-entry!` (*data page* —) stores the *data* in **VDMA Translation Base Address Register** identified by its VDMA *page* number.

5.1.2 VMEbus Interrupter and Interrupt Handler

The FORTH words described below are available to access and control the VMEbus interrupter and interrupt generator.

`vme-iack@` (*level* — *vector*) initiates an interrupt acknowledge cycle at the given VMEbus interrupt request *level* and returns the obtained 8-bit *vector*. Typically, the *vector* returned is within the range 0 through 255, but when no interrupt is pending, and therefore no interrupt has to be acknowledged, the value -1 is returned.

`vme-intr-pending?` (*level* — *true* | *false*) checks whether an interrupt is pending on a given interrupt request *level* and returns a *flag*. When an interrupt is pending the *flag* is *true*; otherwise it is *false*.

`vme-irq-level@` (— *level*) returns the interrupt request *level* currently in effect and is used by the MVIC when it has to generate a VMEbus interrupt.

`vme-irq-level!` (*level* —) selects the interrupt request *level* to be used when the MVIC has generate a VMEbus interrupt.

`vme-intr!` (*vector level* —) generates a VMEbus interrupt at a given interrupt request *level* accompanied by a specified 8-bit *vector*.

`vme-intr-ena` (*mapping level* —) enables the interrupt to be generated upon the receipt of a VMEbus interrupt at *level*. The parameter *mapping* defines the SBus interrupt asserted when the certain VMEbus interrupt request level is asserted. Table "VMEbus Interrupt to SBus Interrupt Level Mapping" on page 166 lists all allowed mappings.

mapping	SBus Interrupt Level
0	Interrupts are disabled
1	SBus IRQ1
2	SBus IRQ2
3	SBus IRQ3
4	SBus IRQ4
5	SBus IRQ5
6	SBus IRQ6
7	SBus IRQ7

Table 80: VMEbus Interrupt to SBus Interrupt Level Mapping

`vme-intr-dis` (*level* —) disables the interrupt to be generated when the specified VMEbus interrupt request at *level* is asserted.

`install-vme-intr-handler` (*mapping level* —) installs the interrupt service routine dealing with the given VMEbus interrupt *level*. The parameter *mapping* defines the SBus interrupt asserted when the certain VMEbus interrupt request level is asserted. Table "VMEbus Interrupt to SBus Interrupt Level Mapping" on page 166 lists all allowed mappings. The address of the interrupt service routine currently in effect is preserved.

`uninstall-vme-intr-handler` (*level* —) removes the interrupt service routine dealing with the given VMEbus interrupt *level* and installs the *old* interrupt service routine.

`.vme-vectors` (—) displays the VMEbus interrupt vectors received during an interrupt acknowledge cycle.

5.1.3 VMEbus Requester

The FORTH words listed below are available to access and control the VMEbus

`vme-bus-request-level@` (— *level*) returns the VMEbus request *level* in use when the VMEbus interface tries to gain the ownership of the VMEbus.

`vme-bus-request-level!` (*level* —) selects the bus-request *level* to be used when the VMEbus is being accessed.

5.1.4 VMEbus Status Signals and Bus Timer

The FORTH words listed below are available to access and control the VMEbus status signals and the bus watcher.

`vme-sysfail-set` (—) asserts (sets) the VMEbus SYSFAIL* signal.

`vme-sysfail-clear` (—) negates (clears) the VMEbus SYSFAIL* signal.

`vme-sysfail?` (— *true* | *false*) determines the state of the VMEbus SYSFAIL* signal and returns a *flag* set according to the signal's state. When the SYSFAIL* signal is asserted the *flag* returned is *true*; otherwise its value is *false*.

`vme-sysfail-assert-nmiena` (—) allows the VMEbus interface to generate a non-maskable interrupt upon the assertion of the VMEbus SYSFAIL* signal.

`vme-sysfail-assert-nmidis` (—) disables the non-maskable interrupt to be generated upon the assertion of the VMEbus SYSFAIL signal. (This word may be used to clear the corresponding pending interrupt.)

`vme-sysfail-assert-ip?` (— *true* | *false*) checks whether an interrupt is pending due to the assertion of the VMEbus SYSFAIL* signal and returns a *flag* set according to the appropriate interrupt pending flag. The *flag* is *true* when the interrupt is pending; otherwise its value is *false*.

`vme-sysfail-negate-nmiena` (—) allows the VMEbus interface to generate a non-maskable interrupt upon the negation of the VMEbus SYSFAIL* signal.

`vme-sysfail-negate-nmidis` (—) disables the non-maskable interrupt to be generated upon the negation of the VMEbus SYSFAIL signal. (This word may be used to clear the corresponding pending interrupt.)

`vme-sysfail-negate-ip?` (— *true* | *false*) checks whether an interrupt is pending due to the negation of the VMEbus SYSFAIL* signal and returns a *flag* set according to the appropriate interrupt pending flag. The *flag* is *true* when the interrupt is pending; otherwise its value is *false*.

`vme-acfail-set` (—) asserts (sets) the VMEbus ACFAIL* signal.

`vme-acfail-clear` (—) negates (clears) the VMEbus ACFAIL* signal.

`vme-acfail?` (— *true* | *false*) determines the state of the VMEbus ACFAIL* signal and returns a *flag* set according to the signal's state. When the ACFAIL* signal is asserted the *flag* returned is *true*; otherwise it is *false*.

`vme-acfail-assert-nmiena` (—) allows the VMEbus interface to generate a non-maskable interrupt upon the assertion of the VMEbus ACFAIL* signal.

`vme-acfail-assert-nmidis` (—) disables the non-maskable interrupt to be generated upon the assertion of the VMEbus ACFAIL signal. (This word may be used to clear the corresponding pending interrupt.)

`vme-acfail-assert-ip?` (— *true* | *false*) checks whether an interrupt is pending due to the assertion of the VMEbus ACFAIL* signal and returns a *flag* set according to the appropriate interrupt pending flag. The *flag* is *true* when the interrupt is pending; otherwise its value is *false*.

`vme-bus-timer-ena` (—) enables the bus timer to watch any processors's attempt to access the VMEbus.

`vme-bus-timer-dis` (—) disables the bus timer from watching any processors's attempt to access the VMEbus.

`vme-bus-timeout!` (*time-out* —) sets the reference value of the bus timer for timeout according to the given *time-out*. The value of *time-out* may be one of the values in the range 0 through 7. Each value selects a particular time-out period. The table below lists all possible values:

time-out	$t_{bus\text{-}timeout\text{-}min}$
0	4 µs
1	8 µs
2	16 µs
3	64 µs
4	256 µs
5	1 ms
6	4 ms
7	500 ms

Table 81: Bus Timer Timeout Values

5.1.5 VMEbus Master Interface

The FORTH words listed below are available to access and control the VMEbus Interface operating in the master mode. The word *space* represents the most significant four bits of the 36 bit MBus address.

`vme-normal-mbus-addr@` (— *space*) returns the *space* — the most significant four bits of the physical address — where the MVIC responds to a MBus transaction in order to perform a *normal* VMEbus access.

`vme-normal-mbus-addr!` (*space* —) sets the *space* — the most significant four bits of the physical address — where the MVIC will respond to a MBus transaction in order to perform a *normal* VMEbus access.

`vme-block-mbus-addr@` (— *space*) returns the *space* — the most significant four bits of the physical address — where the MVIC responds to a MBus transaction in order to perform a *block* VMEbus access.

`vme-block-mbus-addr!` (*space* —) sets the *space* — the most significant four bits of the physical address — where the MVIC will respond to a MBus transaction in order to perform a *block* VMEbus access.

`vme-user-mbus-addr@` (— *space*) returns the *space* — the most significant four bits of the physical address — where the MVIC responds to a MBus transaction in order to perform a VMEbus access using the user defined address modifier code.

`vme-user-mbus-addr!` (*space* —) sets the *space* — the most significant four bits of the physical address — where the MVIC will respond to a MBus transaction in order to perform a VMEbus access using the user defined address modifier code.

`vme-a16-amcode@` (— *am-code*) returns the user defined address modifier *am-code* emitted by the MVIC during a master VMEbus access within the *short* address space.

`vme-a16-amcode!` (*am-code* —) sets the user defined address modifier *am-code* the MVIC will emit during a master VMEbus access within the *short* address space.

`vme-a24-amcode@` (— *am-code*) returns the user defined address modifier *am-code* emitted by the MVIC during a master VMEbus access within the *standard* address space.

`vme-a24-amcode!` (*am-code* —) sets the user defined address modifier *am-code* the MVIC will emit during a master VMEbus access within the *standard* address space.

`vme-a32-amcode@` (— *am-code*) returns the user defined address modifier *am-code* emitted by the MVIC during a master VMEbus access within the *extended* address space.

`vme-a32-amcode!` (*am-code* —) sets the user defined address modifier *am-code* the MVIC will emit during a master VMEbus access within the *extended* address space.

`vme-a16-c@` (*addr* — *data*) reads an 8-bit *data* from a given address *addr* within the *short* address space of the VMEbus.

`vme-a16-c!` (*data addr* —) stores an 8-bit *data* at a given address *addr* within the *short* address space of the VMEbus.

`vme-a16-w@` (*addr* — *data*) reads a 16-bit *data* from a given address *addr* within the *short* address space of the VMEbus.

`vme-a16-w!` (*data addr* —) stores a 16-bit *data* at a given address *addr* within the *short* address space of the VMEbus.

`vme-a16-l@` (*addr* — *data*) stores a 32-bit *data* at a given address *addr* within the *short* address space of the VMEbus.

`vme-a16-l!` (*data addr* —) reads a 32-bit *data* from a given address *addr* within the *short* address space of the VMEbus.

`vme-a24-c@` (*addr* — *data*) reads an 8-bit *data* from a given address *addr* within the *standard* address space of the VMEbus.

`vme-a24-c!` (*data addr* —) stores an 8-bit *data* at a given address *addr* within the *standard* address space of the VMEbus.

`vme-a24-w@` (*addr* — *data*) reads a 16-bit *data* from a given address *addr* within the *standard* address space of the VMEbus.

`vme-a24-w!` (*data addr* —) stores a 16-bit *data* at a given address *addr* within the *standard* address space of the VMEbus.

`vme-a24-l@` (*addr* —*data*) reads a 32-bit *data* from a given address *addr* within the *standard* address space of the VMEbus.

`vme-a24-l!` (*data addr* —) stores a 32-bit *data* at a given address *addr* within the *standard* address space of the VMEbus.

`vme-a32-c@` (*addr* — *data*) reads an 8-bit *data* from a given address *addr* within the *extended* address space of the VMEbus.

`vme-a32-c!` (*data addr* —) stores an 8-bit *data* at a given address *addr* within the *extended* address space of the VMEbus.

`vme-a32-w@` (*addr* — *data*) reads a 16-bit *data* from a given address *addr* within the *extended* address space of the VMEbus.

`vme-a32-w!` (*data addr* —) stores a 16-bit *data* at a given address *addr* within the *extended* address space of the VMEbus.

`vme-a32-l@` (*addr* — *data*) reads a 32-bit *data* from a given address *addr* within the *extended* address space of the VMEbus.

`vme-a32-l!` (*data addr* —) stores a 32-bit *data* at a given address *addr* within the *extended* address space of the VMEbus.

The VMEbus interface is prepared for subsequent *normal* master accesses by:

```
ok 2 vme-normal-mbus-addr!
ok
```

This command sets the *space* the MVIC will respond to during a MBus transaction in order to perform the particular VMEbus *master* access.
On the SPARC CPU-10 only the spaces 2_{16} to 9_{16} should be used. The spaces 0_{16} and 1_{16} are reserved for the memory controllers; the spaces A_{16} , B_{16} , C_{16} , and D_{16} are reserved; the SBus is accessed within space E_{16} , and the space F_{16} is used to access the System Control Space.

The words `vme-{a16|a24|a32}-{c|w|l}{@|!}` are available to access data in a certain VMEbus address range. Data in the *short* address range is accessed, either by

```
ok h# 8000 vme-a16-c@
ok
```

or

```
ok h# 55 h# 8002 vme-a16-c!
```

```
ok
```

In the first case a single byte is read from address 8000_{16} within the *short* address range; and in the latter case the value 55_{16} is stored at address 8000_{16} within the *short* address range.
Data in the *standard* address range is accessed, either by

```
ok h# 48.0000 vme-a24-l@
ok
```

or

```
ok h# 55aa1234 h# 28.0000 vme-a24-l!
ok
```

In the former case a word (32 bit) is read from address 480000_{16} within the *standard* address range; and in the latter case the value $55AA1234_{16}$ is stored at address 280000_{16} within the *standard* address range.
Data in the *extended* address range is accessed, either by

```
ok h# 4080.0000 vme-a32-w@
ok
```

or

```
ok h# ff7f h# 8400.1000 vme-a32-w!
ok
```

In the former case a half-word (16 bit) is read from address 84001000_{16} within the *extended* address range; and in the latter case the value $FF7F_{16}$ is stored at address $B8001000_{16}$ in the *extended* address range.
When the words described above are used to access the VMEbus, then these accesses are accompanied by the appropriate address modifier. The address modifier is generated automatically by the MVIC depending on the MBus transaction that accesses the VMEbus through the MVIC. Because OpenBoot is always executed in the *supervisor* mode, the MVIC will emit one of the *supervisory* address modifier codes ($0D_{16}$, $2D_{16}$, $3D_{16}$) to the VMEbus.
When the VMEbus is being accessed by

```
ok h# 33 h# b801.0000 vme-a32-c!
ok
```

the MVIC will generate the *Extended Supervisory Data Access* address modifier ($0D_{16}$).
To perform a master access to the VMEbus with a *user defined* address modifier (10_{16} to $1F_{16}$), the VMEbus interface must be initialised as follows:

```
ok 3 vme-user-mbus-addr!
ok
```

This command sets the *space* the MVIC will respond to during an MBus transaction in order to access the VMEbus. Such an access will be accompanied by an address modifier previously defined by

```
ok h# 3 vme-a16-amcode!
ok h# 8 vme-a24-amcode!
ok h# e vme-a32-amcode!
ok
```

The commands listed above define the address modifiers to be used when the *short-*, *standard-*, and *extended* address ranges are accessed.
The words `vme-{a16|a24|a32}-{c|w|l}{@|!}` only allow to access the VMEbus in *normal* mode. Thus, the word below has to be defined to access data in a certain address range accompanied by the *user defined* address modifier:

```
ok : vme-a32-user-c@ ( address -- data )
] vmea32-base + vme-user-mbus-addr@ h# 20 or spacec@
] ;
ok
```

This word reads a single byte from a given *address* within the *extended* address range and returns the obtained *data*. During the read access the *user defined* address modifier — in the example $1E_{16}$ — is applied, instead of the address modifier usually generated by the MVIC upon the state of the particular MBus transaction.

The words `vme-{a16|a24|a32}-{c|w|l}{@|!}` mentioned above only allow to access one single data at a time. When a command has to be used to, e.g. display the contents of a VMEbus area, the following steps have to be taken:

```
ok 2 vme-normal-mbus-addr!
ok h# 4084.0000 h# 2 pagesize memmap value vme-area
ok vme-area pagesize not-cacheable
ok vme-area pagesize memory-test-suite
   Data lines test -- succeeded.
   Address quick test -- succeeded.
   Data size test -- succeeded.
   Data bits test -- succeeded.
   Address=data test -- succeeded.
ok .
0
ok
```

The example shown above makes an area beginning at address 40840000_{16} within the *extended* address range available, and the command `memory-test-suite` is used to access this area. At first the VMEbus interface is prepared for master accesses by setting the *space* the MVIC will respond to during a MBus transaction to access the VMEbus. In the following step the physical address 40840000_{16} — a location within the *extended* address range of the VMEbus — is mapped to the processor's *virtual* address space. The virtual address returned by the word `memmap` is stored in the variable `vme-area`. Only `pagesize` bytes — typically 4 Kbytes

— are made available. The next command disables the cacheability of this area, and guarantees that any data written to the area is stored immediately in the VMEbus area, instead of the cache only, and vice versa.
A memory test is then performed on this area, and the result returned by the word `memory-test-suite` is discarded with the *dot* command.

When an area within the *standard* address range has to be accessed, then the address given in the second command has to be replaced by FF<*a24-address*>$_{16}$. And in the case of the *short* address range, the address must be replaced by FFFF<*a16-address*>$_{16}$.

5.1.6 VMEbus Slave Interface

The FORTH words listed below are available to access and control the VMEbus Interface operating in the slave mode.

`vme-a16-slave-ena` (—) enables the MVIC to respond to an access within the *short* address space (A16).

`vme-a16-slave-dis` (—) prevents the MVIC from responding to an access within the *short* address space (A16).

`vme-a16-slave-addr@` (— *vme-a16-addr*) returns the base address the MVIC will respond to when a VMEbus master accesses the A16 address space on the VMEbus.

`vme-a16-slave-addr!` (*vme-a16-addr* —) sets the base address the MVIC will respond to when a VMEbus master accesses the A16 address space on the VMEbus.

`vme-a16-mbus-addr@` (— *addr space*) returns the physical base address emitted by the MVIC on the MBus when it responds to an access within the *short* address space of the VMEbus. The physical base address is represented by the *space*, which corresponds to the physical MBus address bits 35 to 32, and the offset within the particular space is identified by *addr*, which corresponds to the address bits 0 through 31.

`vme-a16-mbus-addr!` (*addr space* —) sets the physical base address the MVIC will emit on the MBus when it responds to an access within the *short* address space of the VMEbus. The physical base address is represented by the *space*, which corresponds to the physical MBus address bits 35 to 32, and the offset within the particular space is identified by *addr*, which corresponds to the address bits 0 through 31.

`vme-a16-reset-ena` (—) enables the MVIC to generate a reset when a given VMEbus address within the A16 address space matches the *slave reset address* specified in the VMEbus A16 Reset and NMI register.

`vme-a16-reset-dis` (—) prevents the MVIC from generating a reset when a given VMEbus address within the A16 address space matches the *slave reset address* specified in the VMEbus A16 Reset and NMI register.

`vme-a16-reset-addr@` (— *vme-a16-addr*) returns the address, which are to be accesses within the *short* address space of the VMEbus, causing the MVIC to generate a RESET on the board.

`vme-a16-reset-addr!` (*vme-a16-addr* —) sets the address, which are to be accesses within the *short* address space of the VMEbus, causing the MVIC to generate a RESET on the board.

`vme-a16-nmi-ena` (—) enables the MVIC to generate a non-maskable interrupt when a given VMEbus address within the A16 address space matches the *slave NMI address* specified in the VMEbus A16 Reset and NMI register.

`vme-a16-nmi-dis` (—) prevents the MVIC to generate a non-maskable interrupt when a given VMEbus address within the A16 address space matches the *slave NMI address* specified in the VMEbus A16 Reset and NMI register.

`vme-a16-nmi-addr@` (— *vme-a16-addr*) returns the address, which are to be accesses within the *short* address space of the VMEbus, causing the MVIC to generate a non-maskable interrupt on the board.

`vme-a16-nmi-addr!` (*vme-a16-addr* —) sets the address, which are to be accesses within the *short* address space of the VMEbus, causing the MVIC to generate a non-maskable interrupt on the board.

`vme-a16-nmi-addr@` (— *vme-a16-addr*) returns the base address *vme-a16-addr* where the MVIC will respond to in A16 slave mode.

`vme-a16-nmi-addr!` (*vme-a16-addr* —) sets the base address *vme-a16-addr* where the MVIC responds to in A16 slave mode.

The VMEbus interface is prepared for subsequent A16 *slave* accesses by:

```
ok h# 1000 vme-a16-slave-addr!
ok h# 10.0000 obmem vme-a16-mbus-addr!
ok vme-a16-ena
ok h# 10.0000 obmem h# 10.0000 pagesize map-pages
ok h# 10.0000 pagesize not-cacheable
ok
```

As stated in the example, the VMEbus interface's A16 slave address is set by the first command to 1000_{16} . The size of the A16 slave area is limited by the MVIC to 32 Bytes!
The second command defines that any A16 slave access is translated to an access of the onboard memory beginning at address 100000_{16} . And the third command enables slave accesses to the A16 address range.
The last two commands make the memory available to the processor's virtual address space and disables cacheability of this area. These commands do not influence the VMEbus interface,

but are presented for completeness. They allow one to inspect this area from within OpenBoot using the appropriate command, e.g. `dump, c@`, etc.

The VMEbus interface allows another VMEbus master to generate a reset and a non-maskable interrupt by accessing unique addresses within the *short* address range. These addresses are defined by

```
ok h# c000 vme-a16-reset-addr!
ok vme-a16-reset-ena
ok h# c004 vme-a16-nmi-addr!
ok vme-a16-nmi-ena
ok vme-a16-slave-ena
ok
```

As shown in the example above, the address to be accessed within the *short* address range to generate a reset is $C000_{16}$; and the address to be accessed to generate a non-maskable interrupt is $C004_{16}$. It is not sufficient to enable the generation of a reset and non-maskable interrupt only using the words `vme-a16-reset-ena` and `vme-a16-nmi-ena`, but the A16 slave interface must be enabled at all times.

A variable, called `a16nmi-occurred?`, is available which is incremented by one whenever a non-maskable interrupt has been generated by accessing the appropriate address within the A16 slave interface. The state of this variable is determined by:

```
ok a16nmi-occurred? ?
5
ok
```

and is cleared — set to zero — by

```
ok a16nmi-occurred? off
ok
```

`vme-a24-slave-ena` (—) enables the MVIC to respond to an access within the *standard* address space (A24).

`vme-a24-slave-dis` (—) prevents the MVIC from responding to an access within the *standard* address space (A24).

`vme-a24-slave-addr@` (— *vme-a24-addr*) returns the base address the MVIC will respond to when a VMEbus master accesses the A24 address space on the VMEbus.

`vme-a24-slave-addr!` (*vme-a24-addr* —) sets the base address the MVIC will respond to when a VMEbus master accesses the A24 address space on the VMEbus.

`vme-a24-mbus-addr@` (— *addr space*) returns the physical base address emitted by the MVIC on the MBus when it responds to an access within the *standard* address space of the VMEbus. The physical base address is represented by the *space*, which corresponds to the physical MBus address bits 35 to 32, and the offset within the particular space is identified by *addr*, which corresponds to the address bits 0 through 31.

`vme-a24-mbus-addr!` (*addr space* —) sets the physical base address emitted by the MVIC on the MBus when it responds to an access within the *standard* address space of the VMEbus. The physical base address is represented by the *space*, which corresponds to the physical MBus address bits 35 to 32, and the offset within the particular space is identified by *addr*, which corresponds to the address bits 0 through 31.

`vdma-page-validate` (*page* —) validates an entry in the VDMA Translation Table identified by its *page* number.

`vdma-page-invalidate` (*page* —) invalidates an entry in the VDMA Translation Table identified by its *page* number.

`vdma-table-clear` (—) invalidates all entries within the VDMA Translation Table and sets all VDMA translations to zero.

`vdma-table-set` (*addr size* —) sets the necessary number of entries in the VDMA Translation Table according to a given address range identified by *addr* and *size*.

`mbus-addr>vdma` (*addr* — *vdma-translation*) converts a given physical address *addr* to the corresponding VDMA translation. Only the address bits 12 to 28 of the given address are considered.

`vdma>mbus-addr` (*vdma-translation* — *addr*) converts a given VDMA translation *vdma-translation* to its physical address *addr*.

The VMEbus Interface is prepared for subsequent A24 *slave* accesses by

```
ok h# 70.0000 vme-a24-slave-addr!
ok h# 10.0000 obmem vme-a24-mbus-addr!
ok h# 10.0000 1meg vdma-table-set
ok vme-a24-slave-ena
ok h# 10.0000 obmem h# 10.0000 1meg map-pages
ok h# 10.0000 1meg not-cacheable
ok
```

In the example described above, the VMEbus interface's A24 *slave* address is set by the first command to 700000_{16} . The second command defines that any A24 slave access is translated to an access of the onboard memory beginning at physical address 100000_{16} . Because the VMEbus interface translates the VMEbus address to an on-board address using the VDMA Translation Table, the third command initialises the VDMA Translation Table in such a way that 1 Mbyte of coherent memory is accessible. Finally, the fourth command enables slave accesses to the A24 address range.
The last two commands make the memory available to the processor's virtual address space and disables cacheability of this area. These commands do not influence the VMEbus interface,

but are presented for completeness. They allow one to inspect this area from within OpenBoot using the appropriate command, e.g. `dump, c@`, etc.

`vme-a32-slave-ena` (—) enables the MVIC to respond to an access within the *extended* address space (A32).

`vme-a32-slave-dis` (—) prevents the MVIC from responding to an access within the *extended* address space (A32).

`vme-a32-slave-addr!` (*window-size vme-a32-addr* —) sets the base address — the most significant eight bits of the VMEbus address — the MVIC will respond to when a VMEbus master accesses the A32 address space on the VMEbus. The value of *window-size* may range from zero to eight. Each value specifies the window size as stated in table .

`vme-a32-slave-addr@` (— *window-size vme-a32-addr*) returns the base address — the most significant eight bits of the VMEbus address — the MVIC will respond to when a VMEbus master accesses the A32 address space on the VMEbus. The value of *window-size* may range from zero to eight. Each value identifies the window size as stated in table .

window-size	VME Slave Window Size
0	16 MB
1	32 MB
2	64 MB
3	128 MB
4	256 MB
5	512 MB
6	1024 MB
7	2048 MB
8	4096 MB

Table 82: Sizes of the A32 Slave Address Space.

`vme-a32-mbus-addr@` (— *addr space*) returns the physical base address emitted by the MVIC on the MBus when it responds to an access within the *extended* address space of the VMEbus. The physical base address is represented by the *space*, which corresponds to the physical MBus address bits 35 to 32, and the offset within the particular space is identified by *addr*, which corresponds to the address bits 0 through 31.

`vme-a32-mbus-addr!` (*addr space* —) sets the physical base address the MVIC will emit on the MBus when it responds to an access within the *extended* address space of the VMEbus. The physical base address is represented by the *space*, which corresponds to the physical MBus address bits 35 to 32, and the offset within the particular space is identified by *addr*, which corresponds to the address bits 0 through 31.

The VMEbus Interface is prepared for subsequent A32 *slave* accesses by

```
ok h# 0 h# C800.0000 vme-a32-slave-addr!
ok h# 1000.0000 obmem vme-a32-mbus-addr!
ok vme-a32-slave-ena
ok h# 1000.0000 obmem h# 10.0000 1meg map-pages
ok h# 10.0000 1meg not-cacheable
ok
```

In the example described above, the VMEbus interface's A32 *slave* address is set by the first command to $C8000000_{16}$. The second command defines that any A32 slave access is translated to an access of the onboard memory beginning at physical address 10000000_{16} . Finally, the third command enables slave accesses to the A32 address range.
The last two commands make the memory — only the first 1 Mbyte — available to the processor's virtual address space and disables cacheability of this area. These commands do not influence the VMEbus interface, but are presented for completeness. They allow one to inspect this area from within OpenBoot using the appropriate command, e.g. `dump, c@`, etc.

5.1.7 VMEbus Device Node

The OpenBoot device tree contains the device node for the VMEbus interface and is called "`VME`". It is a *child* of the device node "`/iommu`" (the full pathname of the VMEbus *device* is "`/iommu@f,e0000000/VME@f,fcfff000`"). The *device alias* **`vme`** is available as a shorthand representation of the VMEbus device-path.
The vocabulary of the VMEbus device includes the standard commands recommended for a *hierarchical* device. The words of this vocabulary are only available when the VMEbus device has been selected as shown below:

```
ok cd vme
ok words
selftest    reset    close    open ...
... list of further methods of the device node
ok selftest .
0
ok device-end
ok
```

The example listed above, selects the VMEbus device and makes it the current node. The word `words` displays the names of the *methods* of the VMEbus device. And the third command calls the method `selftest` and the value return by this method is displayed. The last command

unselects the current device node, leaving no node selected.
The following methods are defined in the vocabulary of the VMEbus device:

`open` (— `true`) prepares the package for subsequent use. The value `true` is always returned.

`close` (—) frees all resources allocated by open.

`reset` (—) puts the VMEbus Interface into *quiet* state.

`selftest` (— *error-number*) performs a test of the VMEbus interface, and returns an *error-number* to report the course of the test. In the case that the device has been tested successfully the value zero is returned; otherwise it returns a specific error number to indicate a certain fail state.

`decode-unit` (*addr len* — *low high*) converts the *addr* and *len*, a text string representation, to *low* and *high* which is a numerical representation of a physical address within the address space defined by the package.

`map-in` (*low high size* — *vaddr*) creates a mapping associating the range of physical address beginning at *low*, extending for *size* bytes, within the package's physical address space, with a processor virtual address *vaddr*.

`map-out` (*vaddr size* —) destroys the mapping set by map-in at the given virtual address *vaddr* of length *size*.

`dma-alloc` (*size* — *vaddr*) allocates a virtual address range of length *size* bytes that is suitable for direct memory access by a bus master device. The memory is allocated according to the most stringent alignment requirements for the bus. The address of the acquired virtual memory *vaddr* is returned via the stack.

`dma-free` (*vaddr size* —) releases a given virtual memory, identified by its address *vaddr* and *size*, previously acquired by `dma-alloc`.

`dma-map-in` (*vaddr size cachable?* — *devaddr*) converts a given virtual address range, specified by *vaddr* and *size*, into an address *devaddr* suitable for direct memory access on the bus. The virtual memory must be allocated already by `dma-alloc`. The SPARC CPU-10 does not support caching. Thus the *cachable?* flag is ignored.

`dma-map-out` (*vaddr devaddr size* —) removes the direct memory access mapping previously created by `dma-map-in`.

`dma-sync` (*vaddr devaddr size* —) synchronizes memory caches associated with a given direct memory access mapping, specified by its virtual address *vaddr*, the *devaddr* and its *size* that has been established by dma-map-in.

The NVRAM configuration parameters listed below are available to control the VMEbus Interface. The current state of these configuration parameters are displayed using the `printenv` command, and are modified using either the `setenv`, or the `set-default` command provided by OpenBoot.

`vme-sysfail-clear?` (default: `true`) when the value of the configuration parameter is `true` the SYSFAIL* signal will be cleared by OpenBoot. In the case that the configuration parameter is `false` OpenBoot will not clear the SYSFAIL* signal, but the operating system which is loaded has to clear it.

`vme-init?` controls whether the VMEbus interface is initialised by OpenBoot. When this flag is `true` the VMEbus interface is initialised according to the state of the NVRAM parameter listed below. In the case that the flag is `false` the VMEbus interface is not initialised. The VMEbus interface is initialised after OpenBoot setup the main memory. (default: `true`)

`vme-mbar` contains a 32 bit value to be stored in the MVIC’s MBus Base Address Register. (default: $67000002_{16} = 1728053250_{10}$)

`vme-vmcr` contains a 32 bit value to be stored in the MVIC’s VMEbus Master Configuration Register. On the SPARC CPU-10, the least significant bit (bit 0) of this value is cleared to avoid the setting of the Slot-1-Enable bit because the MVIC’s internal arbiter is not used. (default: $CC_{16} = 204_{10}$)

`vme-vscr` contains a 32 bit value to be stored in the MVIC’s VMEbus Slave Configuration Register. (default: 0_{10})

`vme-vasmbar` contains a 32 bit value to be stored in the MVIC’s VMEbus A16 Slave MBus Base Address Register. (default: 0_{10})

`vme-a16nmi` contains a 32 bit value to be stored in the MVIC’s VMEbus A16 Reset and NMI Register. (default: 0_{10})

`vme-a24-mem-base` contains the physical MBus address any A24 slave access is translated to. Only the bits 12 to 28 are considered! (default: 0_{10})

`vme-a24-mem-size` contains the size of the address range which is made available to the VMEbus A24 address space. This address range is limited to 1 MByte! (default: 100000_{10} which is 1Mbyte)

`vme-intr1` controls whether the VMEbus interrupt request level 1 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 1 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 1 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 1 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-intr2` controls whether the VMEbus interrupt request level 2 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 2 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 2 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 2 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-intr3` controls whether the VMEbus interrupt request level 3 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 3 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 3 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 3 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-intr4` controls whether the VMEbus interrupt request level 4 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 4 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 4 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 4 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-intr5` controls whether the VMEbus interrupt request level 5 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 5 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 5 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 5 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-intr6` controls whether the VMEbus interrupt request level 6 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 6 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 6 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 6 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-intr7` controls whether the VMEbus interrupt request level 7 has to be enabled. When this flag is zero (`0`) then the VMEbus interrupt request level 7 is not enabled. In the case that the value is within the range 1 to 7, the corresponding interrupt handler is activated and the VMEbus interrupt request level 7 is enabled. The values 1 to 7 specify the SBus interrupt level to be generated when a VMEbus interrupt request level 7 occurs. Only the least significant three bits of this value are considered! (default: 0_{10})

`vme-bus-timer?` controls whether the the *external* bus timer — included in the LCA — is used instead of the bus timer provided by the MVIC. When the flag is `true` the bus timer in the MVIC is disabled and the *external* bus timer is enabled. If the flag is `false` the *external* bus timer is disabled while the MVIC's bus timer is used. (default: `true`)

`vme-bus-timeout` contains the timeout value of the *external* bus timer and is a value in the range 0 to 7. Each value selects a particular time-out period. Independent of the state of the configuration parameter `vme-bus-timer?` the time-out value is stored in the appropriate bus timer register. (default: 1_{10})

`wd-ena?` controls whether the watchdog timer has to be started. When the flag is `true`, then the watchdog timer is started after it has been initialised according to the configuration parameter `wd-timeout`, `wd-protect?`, and `wd-por-ena?`. If the flag is `false` the watchdog timer is not started, but the watchdog timer registers are initialised according to the configuration parameters `wd-timeout` and `wd-por-ena?`. Even when the `wd-protect?` flag is `true` the watchdog registers are not protected against modification when the `wd-ena?` flag is `false`! (default: `false`)

`wd-timeout` contains the time-out value of the watchdog timer and is a value in the range 0 to 7. Each value selects a particular time-out period. Independent of the state of the configuration parameter `wd-ena?` the time-out value is stored in the appropriate watchdog timer register. (default: 7_{10})

`wd-protect?` controls whether the watchdog timer registers have to be protected against modification after the watchdog timer is started. When the flag is `true` the registers are protected against modification; otherwise, — the flag is `false` — the contents of the watchdog timer registers are alterable. In the case that the `wd-ena?` flag is `false` the state of this flag is ignored. (default: `false`)

`wd-por-ena?` controls whether the watchdog timer generates a reset equivalent to the power-on reset when the time has expired. When the flag is `true` the watchdog timer generates a power-on reset. If the flag is `false` the watchdog will not generate a power-on reset, but still generates a *watchdog timer* reset. (default: `false`)

`abort-ena?` controls whether the abort switch has to be enabled. When this flag is `true` the abort switch is enabled and has the same effect as pressing the `L1-A` key on the keyboard. If the flag is `false` then the abort switch is disabled. (default: `false`)

`vme-sysfail-assert?` controls whether a non-maskable interrupt is generated upon the assertion of the VMEbus signal SYSFAIL*. When the flag is `true` an interrupt handler, dealing with this interrupt, is installed and the ability to generate a non-maskable interrupt upon the assertion of the SYSFAIL* signal is enabled. In the case that the flag is `false` neither an interrupt handler is installed nor the ability to generate a non-maskable interrupt upon the assertion of the SYSFAIL* signal is enabled. (default: `false`)

`vme-sysfail-negate?` controls whether a non-maskable interrupt is generated upon the negation of the VMEbus signal SYSFAIL*. When the flag is `true` an interrupt handler, dealing with this interrupt, is installed and the ability to generate a non-maskable interrupt upon the negation of the SYSFAIL* signal is enabled. In the case that the flag is `false` neither an interrupt handler is installed nor the ability to generate a non-maskable interrupt upon the negation of the SYSFAIL* signal is enabled. (default: `false`)

`vme-acfail-assert?` controls whether a non-maskable interrupt is generated upon the assertion of the VMEbus signal ACFAIL*. When the flag is `true` an interrupt handler, dealing with this interrupt, is installed and the ability to generate a non-maskable interrupt upon the assertion of the ACFAIL* signal is enabled. In the case that the flag is `false` neither an interrupt handler is installed nor the ability to generate a non-maskable interrupt upon the assertion of the ACFAIL* signal is enabled. (default: `false`)

The NVRAM configuration parameters described so far are modified by the commands `setenv` and `set-default` available in OpenBoot. These commands allow modifying a specific configuration parameter.
Additionally, OpenBoot for the SPARC CPU-10 provides the following commands to *save* and *restore* the state of the VMEbus interface:

`vme-save-state` (—) *saves* the current state of the VMEbus interface. It sets the NVRAM configuration parameters — described on the previous pages — to reflect the state of the interface.

`vme-restore-state` (—) *restores* the state of the VMEbus interface according to the NVRAM configuration parameters described on the previous pages.

5.2 Diagnostic Control Registers

The FORTH words listed below are available to access and control the diagnostic control registers (rotary switches, status LEDs) within the LCA.

`led1-on` (—) turns the first *user* LED on.

`led1-off` (—) turns the first *user* LED off.

`led1?` (— *true* | *false*) determines the state of the first *user* LED, and returns the *flag* to indicate if the LED is turned on or off. The *flag* is set *true* when the LED is turned on; otherwise the *flag* is *false*.

`led2-on` (—) turns the second *user* LED on.

`led2-off` (—) turns the second *user* LED off.

`led2?` (— *true* | *false*) determines the state of the second *user* LED, and returns the *flag* to indicate if the LED is turned on or off. The *flag* is set *true* when the LED is turned on; otherwise the *flag* is *false*.

`rotary-switch@` (— *value*) returns the current state of the rotary switch.

The following example shows how to benefit from the words listed above: the word `toggle-led1` is defined as that turns the first *user* LED on or off depending on the current state of the LED.
After the word has been defined it is added to the OpenBoot *alarm list* in order to be periodically called every two seconds.

```
ok : toggle-led1 ( -- )
] led1? if led1-off else led1-on then
] ;
```

```
ok ['] toggle-led1 d# 2000 alarm
ok
```

The word is removed from the *alarm list* — and blinking of the LED is ceased — by

```
ok ['] toggle-led1 0 alarm
ok
```

5.3 Sytem Configuration Registers

The FORTH words listed below are available to access and control the system configuration registers (Watchdog timer, flash memories) within the LCA.

`abort?` (— *true* | *false*) determines the current state of the abort switch and returns the *flag* to indicate whether the abort switch is pressed or released. The *flag* is *true* when the abort switch is still pressed; otherwise the value is *false*.

`abort-nmi-ena` (—) allows a non-maskable interrupt to generate when the abort switch is pressed.

`abort-nmi-dis` (—) disables the non-maskable interrupt's ability to generate when the abort switch is being pressed. (This word may be used to clear the corresponding pending interrupt.)

`abort-ip?` (— *true* | *false*) checks whether an interrupt is pending because the abort switch has been pressed and returns the *flag*. The *flag* is *true* when the interrupt is pending; otherwise it's value is *false*.

`abort-nmi-clear` (—) clears a pending interrupt caused by the abort switch.

`flash-vpp?` (— *true* | *false*) determines the state of the programming voltage, and returns a *flag* to indicate whether it is turned on or off. When the programming voltage is turned on the *flag* is *true*; otherwise it is *false*.

`flash-vpp-on` (—) turns the programming voltage on.

`flash-vpp-off` (—) turns the programming voltage off.

`userprom-select-page` (*page* —) makes a *page* (one of a eight possible 512 KB pages) of a USER flash memory available in the *flash memory programming window*.

`bootprom-select-page` (*page* —) makes a *page* (one of a eight possible 512 KB pages) of a BOOT flash memory available in the *flash memory programming window*.

`select-bootprom-1` (—) makes the first BOOT flash memory device available in the *flash memory programming window*.

`select-bootprom-2` (—) makes the second BOOT flash memory device available in the *flash memory programming window*.

`select-bootprom` (*device-number* —) makes a BOOT flash memory device, identified by its *device-number*, available in the *flash memory programming window*. The devices are numbered beginning from zero (0).

`bootprom-size?` (— *true* | *false*) determines the size of the BOOT flash memory devices in use and returns a *flag* to indicate whether 512 Kbyte flash memory devices, or 256 Kbyte flash memory devices are installed. The *flag* is *true* when the 512 Kbyte flash memory devices are selected, otherwise the *flag* is *false*.

`bootprom-size-256k` (—) sets the size of the BOOT flash memory devices on the board to 256 Kbyte.

`bootprom-size-512k` (—) sets the size of the BOOT flash memory devices on the board to 512 Kbyte.

`select-userprom-1` (—) makes the first USER flash memory device available in the *flash memory programming window*.

`select-userprom-2` (—) makes the second USER flash memory device available in the *flash memory programming window*.

`select-userprom` (*device* —) makes a USER flash memory device, identified by its *device-number*, available in the *flash memory programming window*. The devices are numbered beginning from zero (0).

`wd-ena` (—) enables and starts the watchdog timer.

`wd-dis` (—) stops and disables the watchdog timer.

`wd-timeout!` (*time-out* —) sets the watchdog timer's reference value for timeout according to the given *time-out*. The value of *time-out* may be one of the values in the range 0 through 7. The values select a particular time-out period. The table below lists all possible values:

time-out	$t_{wd\text{-}timeout\text{-}min}$
0	75ms
1	300 ms
2	1200 ms

Table 83: Watchdog Timer Timeout Values

time-out	$t_{wd\text{-}timeout\text{-}min}$
3	4.8 s
4	19 s
5	75 s
6	5 min
7	20 min

Table 83: Watchdog Timer Timeout Values

`wd-nmi-ena` (—) allow a non-maskable interrupt to generate when half of the watchdog time has expired.

`wd-nmi-dis` (—) disables the non-maskable interrupt's ability to generate when half of the watchdog time has expired.

`wd-nmi-clear` (—) clears a pending interrupt caused by the watchdog timer when half of the watchdog time has expired.

`wd-protect` (— *true* | *false*) prevents all watchdog timer registers, except the register used to reset the watchdog timer, from being modified. The watchdog timer registers are only protected against modification when the watchdog timer has been already enabled. When the registers have been protected successfully, the *flag* returned is *true*; otherwise it is *false*

`wd-poweron-reset-ena` (—) enables the watchdog timer to generate a reset equivalent to the power-on reset when the time has expired.

`wd-poweron-reset-dis` (—) disables the watchdog timer from generating a reset equivalent to the power-on reset when the time has expired (it still generates a *watchdog timer reset*).

`wd-ip?` (— *true* | *false*) checks whether an interrupt is pending due to a non-maskable interrupt generated by the watchdog timer when half of the watchdog time has expired. The *flag* is *true* when the interrupt is pending; otherwise its value is *false*.

`wd-protected?` (— *true* | *false*) checks whether the watchdog timer has been protected. The *flag* is *true* when the watchdog timer has been protected; otherwise its value is *false*.

`wd-restart` (—) resets the watchdog timer and starts a new time count.

The Watchdog Timer is started by the commands listed below:

```
ok 3 wd-timeout!
ok wd-nmi-ena
ok wd-ena
ok
```

In this example the *watchdog timeout* is set to 8 seconds, and a **non-maskable** interrupt is generated whenever half of the watchdog time has expired. The OpenBoot already contains an interrupt handler dealing with the interrupt generated by the watchdog timer, and this interrupt handler increments an internal variable by one, whenever the watchdog timer emits an interrupt. The state of this variable is determined by:

```
ok wdnmi-occurred? ?
6
ok
```

This variable is cleared — set to zero — by

```
ok wdnmi-occurred? on
ok
```

`mvic-reset?` (— *true* | *false*) determines whether a reset has been generated because the *VMEbus A16 Slave Reset Register* of the MVIC has been accessed. If a reset has been generated by the MVIC, then the *flag* is *true*; otherwise it is *false*.

`vme-sysreset?` (— *true* | *false*) determines whether a reset has been generated because the VMEbus SYSRESET* signal has been asserted. If a reset has been generated due to the assertion of the SYSRESET* signal, then the *flag* is *true*; otherwise it is *false*.

`switch-reset?` (— *true* | *false*) determines whether a reset has been generated because the reset switch has been pressed. If a reset has been generated by pressing the reset switch, then the *flag* is *true*; otherwise it is *false*.

`wd-reset?` (— *true* | *false*) determines whether a reset has been generated because the watchdog timer has expired. If a reset has been generated because the watchdog timer reached the timeout value, then the *flag* is *true*; otherwise it is *false*.

5.4 Flash Memory Support

The FORTH words listed below are available to access and program the flash memories available on the SPARC CPU-10.

`flash-messages` (— *vaddr*) returns the virtual address of the *variable* `flash-messages`. The state of this variable controls whether the words to erase and program the flash memories will display messages while erasing or programming the flash memories. Messages will not be displayed after *turning off* this variable by **`flash-messages off`**, and are displayed after *turning on* this variable by **`flash-message on`**.

`flash-va` (— *vaddr*) returns the virtual base address *vaddr* of the flash memory programming window. The virtual address returned is only valid when the flash memories have been previously prepared for accessing using the `select-flash` word.

`boot-flash-va` (— *vaddr*) returns the virtual base address *vaddr* of the BOOT flash memory.

`user-flash-va` (— *vaddr*) returns the virtual base address *vaddr* of the USER flash memory. When the USER flash memory is not accessible directly, but only through the flash memory programming window, then the address returned is zero. On the SPARC CPU-10 the USER flash memory is accessible only through the flash memory programming window. Thus, the commands described above have to be used to access the USER flash memory.

`select-flash` ("USER<eol>" | "BOOT<eol>" —) prepares either the BOOT flash memories, or the USER flash memories for programming. In detail, the number and size of the available flash memories are determined, as well as the size of the flash memory programming window. The flash memory programming window is mapped and the virtual base address of the window is stored internally, and may be obtained by using the word `flash-va`.

`user-flash?` (— *true* | *false*) checks whether the BOOT flash memory or the USER flash memory is accessible through the flash memory prgramming window. It returns *true* in the case that the USER flash memory is accessible through the programming window; otherwise it returns *false*.

`move>flash` (*source-addr dest-addr count* —) programs the selected flash memory beginning at *dest-addr* with a number of bytes, specified by *count*, stored at *source-addr*.

`flash>move` (*source-addr dest-addr count* —) copies a number of bytes, specified by *count*, from the selected flash memory beginning at *source-addr* to *dest-addr*. The flash memory is accessed through the flash memory programming window for reading data from the memory. Thus, the flash memory has to be prepared for accessing using the command `select-flash`.

`fill-flash` (*dest-addr count pattern* —) fills the selected flash memory beginning at *dest-addr* with a particular *pattern*. The number of bytes to be programmed in the flash memory is given by *count*.

`erase-flash` (*device-number* —) erases a flash memory device identified by its *device-number*. The devices are numbered beginning from zero (0).

`c!-flash` (*byte addr* —) stores the *byte* at the location within the selected flash memory identified by *addr*.

`w!-flash` (*half-word addr* —) stores the *half-word* (16 bits) at the location within selected the flash memory identified by *addr*.

`l!-flash` (*word addr* —) stores the *word* (32 bits) at the location within the selected flash memory identified by *addr*.

The USER flash memory is prepared for programming by:

```
ok select-flash USER
USER flash memory is selected for programming
Flash memory programming window at $ffe98000 size 512 Kbyte
512 Kbyte BOOT flash memory is available at $ffe58000.
2048 Kbyte USER flash memory is available.
ok
```

As shown above, the word `select-flash` informs the user that the USER flash memory has been made accessible through the flash memory programming window. It displays the base address (*virtual* address) of the window and its size.
The total amount of the available BOOT flash memory and USER flash memory is displayed, too. After the USER flash memory has been prepared for programming, all commands described above operate on the USER flash memory. And the BOOT flash memory is only read and programmed by these commands when the BOOT flash memory has been prepared for these operations by:

```
ok select-flash BOOT
BOOT flash memory is selected for programming
Flash memory programming window at $ffe98000 size 512 Kbyte
512 Kbyte BOOT flash memory is available at $ffe58000.
2048 Kbyte USER flash memory is available.
ok
```

To read data from the selected flash memory — in the current context from the USER flash memory — the command `flash>move` is used as follows:

```
ok flash-va h# 10.0000 h# 20.0000 flash>move
ok
```

The contents of the entire USER flash memory is copied to main memory beginning at address 100000_{16} . A specific area within the selected flash memory is read by:

```
ok flash-va h# 6.8000 + h# 10.0000 h# 5.8c00 flash>move
ok
```

and copies 363520 bytes beginning from address `flash-va` + 68000_{16} to main memory beginning at address 100000_{16} .

5.5 Onboard Interrupts

Besides the interrupt handlers already available in the standard OpenBoot, the OpenBoot of the SPARC CPU-10 provides further handlers that deal with the interrupts generated by following:

- one of the VMEbus interrupt levels one to seven;
- the assertion and negation of the SYSFAIL* signal;
- the assertion of the ACFAIL* signal;
- pressing the ABORT switch;
- the Watchdog Timer, when half the time has expired; and
- an A16 *slave* access to the VMEbus interface's A16 Reset and NMI Register.

5.5.1 VMEbus Interrupts

The interrupt handlers for any VMEbus interrupt are not installed automatically by OpenBoot; however, appropriate words are available to *activate* and *deactivate* an interrupt handler serving a specific VMEbus interrupt. Such an interrupt handler is activated by:

```
ok 0 pil!
ok 3 5 install-vme-intr-handler
ok
```

The `pil!` command decreases the processor interrupt level to allow to the processor to respond to all interrupts. By default, OpenBoot sets the mask to 13 and allows the processor to respond to interrupts above interrupt level 13. The second command installs the interrupt handler that deals with the VMEbus interrupt level 5. Furthermore, this command specifies that a SBus interrupt level 3 will be generated upon the occurrence of a VMEbus interrupt 5. Any of the seven SBus interrupt levels may be specified to be generated upon a VMEbus interrupt. OpenBoot maintains seven variables called `vme-intr{1|2|3|4|5|6|7}-vector` which are modified by the VMEbus interrupt handlers. In general, the interrupt handlers store the vector obtained during an interrupt acknowledge cycle in the appropriate variable. The state of these variables is displayed by

```
ok .vme-vectors
1: --   2: --   3: --   4: --   5: 33   6: --   7: --
ok
```

By default, the value -1 (`true`) is assigned to these variables to indicate that no VMEbus interrupt occurred. So, the word `.vme-vectors`, as shown above, will display "`--`" indicating that no interrupt occurred; otherwise it shows the vector obtained (a value in the range 0 to FF_{16}).
Another way to display the state of a variable used to store the interrupt vector is

```
ok vme-intr5-vector ?
33
ok
```

and the variable is set to -1 (`true`) by

```
ok vme-intr5-vector on
ok
```

An interrupt handler is removed and the corresponding interrupt is disabled by

```
ok 5 uninstall-vme-intr-handler
ok
```

All interrupt handlers to serve all VMEbus interrupts are installed by

```
ok 0 pil!
ok 8 1 do i i install-vme-intr-handler loop
ok
```

In this case, all interrupt handlers are installed and the VMEbus interrupt to SBus interrupt mapping is as follows: SBus interrupt level 1 is generated upon the occurrence of a VMEbus interrupt 1; SBus interrupt level2 is generated upon the occurrence of a VMEbus interrupt 2; and so forth.

5.5.2 SYSFAIL Interrupt

OpenBoot for the SPARC CPU-10 already includes an interrupt handler to serve the non-maskable interrupt generated upon the assertion and negation of the SYSFAIL* signal. This handler need not to be installed because it is already installed by OpenBoot.
By default, the interrupts that will be emitted by a status change of the SYSFAIL* signal are disabled and have to be enabled by

```
ok vme-sysfail-assert-nmiena
ok vme-sysfail-negate-nmiena
ok
```

which enable the generation of a non-maskable interrupt whenever the SYSFAIL* signal is asserted and negated.

When an non-maskable interrupt occurred due to the assertion of the SYSFAIL* signal, then the appropriate interrupt handler increments the variable `sysfail-asserted?` by one to report the occurrence of such an interrupt. The variable `sysfail-negated?` is incremented by the interrupt handler when the SYSFAIL* signal has been negated and caused a non-maskable interrupt. The state of both variables are obtained by

```
ok sysfail-asserted? ?
0
ok
```

and

```
ok sysfail-negated? ?
1
ok
```

And these variables are cleared — set to zero — by

```
ok sysfail-asserted? off
ok sysfail-negated? off
ok
```

5.5.3 ACFAIL Interrupt

OpenBoot for the SPARC CPU-10 already includes an interrupt handler to serve the non-maskable interrupt generated upon the assertion of the ACFAIL* signal. This handler need not to be installed because it is already installed when by OpenBoot.
By default, the interrupt that will be emitted by asserting the ACFAIL* signal is disabled and has to be enabled by

```
ok vme-acfail-assert-nmiena
ok
```

which enables the generation of a non-maskable interrupt whenever the ACFAIL* signal is asserted.

When a non-maskable interrupt occurred due to the assertion of the ACFAIL* signal, then the appropriate interrupt handler increments the variable `acfail-asserted?` by one to report the occurrence of such an interrupt. The state of this variable is obtained by

```
ok acfail-asserted? ?
2
ok
```

And the variable is cleared — set to zero — by

```
ok acfail-asserted? off
ok
```

5.5.4 ABORT Interrupt

OpenBoot for the SPARC CPU-10 already includes an interrupt handler to serve the non-maskable interrupt generated by pressing the front panel ABORT switch. This handler need not to be installed because it is already installed by OpenBoot.
By default, the interrupt that will be emitted when the ABORT switch has been pressed is disabled and has to be enabled by

```
ok abort-nmi-ena
ok
```

which enables the generation of a non-maskable interrupt whenever the ABORT switch is pressed.

When a non-maskable interrupt occurred due to pressing the ABORT switch, then the appropriate interrupt handler increments the variable `abort-occurred?` by one to report the occurrence of such an interrupt. The state of both variables are obtained by

```
ok abort-occurred? ?
7
ok
```

And these variables are cleared — set to zero — by

```
ok abort-occurred? off
ok
```

Besides the effects described above, the pressing of the ABORT switch has the same effect as giving the `Stop-A` keyboard command. The program currently running is aborted and the FORTH interpreter is appears immediately.

5.5.5 Watchdog Timer Interrupt

OpenBoot for the SPARC CPU-10 already includes an interrupt handler to serve the non-maskable interrupt generated by the watchdog timer when half of the time has expired. This handler need not to be installed because it is already installed by OpenBoot.
By default, the interrupt that will be emitted by the watchdog timer is disabled — the watchdog timer is disabled at all — and has to be enabled by

```
ok 3 wd-timeout!
ok wd-nmi-ena
ok wd-ena
ok
```

In this example the *watchdog timeout* is set to 8 seconds, and a non-maskable interrupt is generated whenever half of the watchdog time has expired. The interrupt handler included in

OpenBoot restarts the watchdog timer to ensure that the watchdog time will not expire and cause a reset. Additionally the interrupt handler increments the variable `wdnmi-occurred?` by one whenever the watchdog timer emits an interrupt. The state of this variable is determined by

```
ok wdnmi-occurred? ?
6
ok
```

This variable is cleared — set to zero — by

```
ok wdnmi-occurred? on
ok
```

5.5.6 A16 Slave Interrupt

OpenBoot for the SPARC CPU-10 already includes an interrupt handler to serve the non-maskable interrupt generated by a *slave* access to the VMEbus interface's A16 Reset and NMI Register. This handler need not to be installed because it is already installed by OpenBoot. By default, the interrupt that will be emitted by such an access is disabled and has to be enabled by

```
ok h# 4000 vme-a16-nmi-addr!
ok vme-a16-nmi-ena
ok vme-a16-slave-ena
ok
```

As shown in the example above the address to be accessed within the *short* address range to generate a non-maskable interrupt is 4000_{16} . It is not sufficient to enable the generation of a non-maskable interrupt using the words `vme-a16-nmi-ena` only, but the A16 slave interface must be enabled.

A variable, called `a16nmi-occurred?`, is available which is incremented by one whenever a non-maskable interrupt has been generated by accessing the appropriate address within the A16 slave interface. The state of this variable is determined by:

```
ok a16nmi-occurred? ?
5
ok
```

and is cleared — set to zero — by

```
ok a16nmi-occurred? off
ok
```

5.6 Online Help

Some help categories — including information about the peculiarities of the SPARC CPU-10 — have been added to the OpenBoot's *online* help facility. These categories are packed up under the help category **`VMEbus Interface`** and are displayed by

```
ok help VMEbus
    Category: VMEbus Interface
    Sub-categories are:
Generic
Slave Interface
Master Interface
Interrupt Module
ok
```

The contents of a specific sub-category are displayed by

```
ok help generic
The VMEbus Interface consists of ...
...
ok
```

5.7 Further Commands

The FORTH words listed below are available to provide miscellaneous services:

`not-cachable` (*vaddr size* —) disables cacheability of an address range identified by its **virtual** base address *addr* and its *size*.

SECTION 6 Sun Openboot Documentation

6. Insert your OPEN BOOT 2.0 PROM MANUAL SET here.

SECTION 7 Appendix

7. Product Error Report

SECTION 8 User Notes

SECTION 9 Options

SECTION 10 Modifications

SECTION 11 Applications