ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Emerson / Motorola / Force Computers CPU-5CE/32* at our website: ***Click HERE***



SPARC/CPU-5CE
Technical Reference Manual

Edition No. 3.0
October1996

P/N 203509
FORCE COMPUTERS Inc./GmbH
All Rights Reserved

This document shall not be duplicated, nor its contents used
for any purpose, unless express permission has been granted.

Copyright by FORCE COMPUTERS

SECTION 1 INTRODUCTION 1

1. Getting Started 1
1.1. The SPARC CPU-5CE Technical Reference Manual Set 1
1.2. Summary of the SPARC CPU-5CE 2
1.3. Board Components 3
1.4. Specifications 4
1.5. Product Nomenclature 6
1.5.1. Ordering Information 6
1.6. History of the Manual 9

SECTION 2 INSTALLATION 11

2. Introduction 11
2.1. Caution 11
2.2. Location Diagram of the SPARC CPU-5CE Board 12
2.3. Before Powering Up 15
2.3.1. Default Switch Settings 15
2.4. Powering Up 18
2.4.1. VME Slot-1 Device 18
2.4.2. VMEbus SYSRESET Switches 18
2.4.3. Serial Ports 18
2.4.4. RESET and ABORT Key Enable 19
2.4.5. SCSI Termination 19
2.4.6. Boot Flash EPROM Write Protection 19
2.4.7. User Flash EPROM Write Protection 19
2.4.8. Reserved Switches 19
2.4.9. Parallel Port or Floppy Interface via VME P2 Connector 20
2.4.10. Ethernet via Front Panel or VME P2 Connector 21
2.5. OpenBoot Firmware 22
2.5.1. Boot the System 22
2.5.2. NVRAM Boot Parameters 25
2.5.3. Diagnostics 26
2.5.4. Display System Information 29
2.5.5. Reset the System 30
2.5.6. OpenBoot Help 30
2.5.7. How to Install an OpenBoot ROM 32
2.6. Front Panel 33
2.6.1. Features of the Front Panel 34
2.7. SPARC CPU-5CE Connectors 35
2.7.1. Ethernet Connector Pinout 36
2.7.2. Serial Ports A and B Connector Pinout 37
2.7.3. Keyboard/Mouse Connector Pinout 38
2.7.4. VME P2 Connector Pinout 39
2.7.5. The IOBP-10 Connectors 41
2.8. How to Determine the Ethernet Address and Host ID 48

SECTION 3 HARDWARE DESCRIPTION 49

3. Board Components 49

3.1. The microSPARC-II Processor 52
3.1.1. Features of the microSPARC-II Processor 52
3.1.2. Address Mapping for microSPARC-II 53
3.2. The Shared Memory 54
3.3. SBus Participants 55
3.3.1. Address Mapping for SBus Slots on the SPARC CPU-5CE 55
3.4. NCR89C100 (MACIO) 56
3.4.1. Features of the NCR89C100 on the SPARC CPU-5CE 57
3.4.2. SCSI 58
3.4.3. SCSI Termination 58
3.4.4. Ethernet 59
3.4.5. Parallel Port 60
3.5. NCR89C105 (SLAVIO) 61
3.5.1. Features of the NCR89C105 on the SPARC CPU-5CE 61
3.5.2. Address Map of Local I/O Devices on SPARC CPU-5CE 62
3.5.3. Serial I/O Ports 63
3.5.4. RS-232, RS-422 or RS-485 Configuration 63
3.5.5. RS-232 Hardware Configuration 64
3.5.6. RS-422 Hardware Configuration 65
3.5.7. RS-485 Hardware Configuration 67
3.5.8. Transmission Line Termination 68
3.5.9. Keyboard and Mouse Port 72
3.5.10. Floppy Disk Interface 73
3.5.11. 8-Bit Local I/O Devices 74
3.5.12. Boot EPROM 75
3.5.13. User Flash EPROM 76
3.5.14. Programming the On-board Flash Memories 77
3.5.15. Programming Control Bits for Flash Memory Devices 78
3.5.16. RTC/NVRAM 80
3.6. VMEbus Interface 80
3.6.1. Master Interface 80
3.6.1.1. VMEbus Master Address Implementation 81
3.6.1.2. Data Bus Sizes 84
3.6.2. Slave Interface 84
3.6.2.1. VMEbus Slave Address Modes 86
3.6.2.2. VMEbus Default Slave Mode 87
3.6.2.3. VMEbus Enhanced Slave Mode 87
3.6.3. VMEbus Interrupt Handler and MailBox Interrupt Function 89
3.6.4. VMEbus System Controller 90
3.6.5. Register Accesses to the S4-VME Chip 91
3.6.6. VMEbus Utility Functions 91
3.6.7. VMEbus SYSRESET Enable/Disable 95

3.6.8. VMEbus Bus Timer ... 95
3.7. Front Panel ... 96
3.7.1. RESET and ABORT Keys ... 98
3.7.1.1. The RESET Key ... 98
3.7.1.2. The ABORT Key ... 98
3.7.2. Front Panel Status LEDs ... 100
3.7.3. Diagnostic LED (Hex Display) ... 101
3.8. Additional features ... 102
3.8.1. Hardware Watchdog Timer ... 102
3.8.2. Rotary Switch ... 105
3.9. Additional Registers ... 107
3.9.1. vme_slavebase1 Register ... 107
3.9.2. vme_slavebase2 Register ... 108
3.9.3. How to Program the VMEbus Slave Base Address ... 109
3.9.4. vme_slavebase3 Register ... 110
3.9.5. vme_ctl Register ... 111
3.9.6. vme_a32map Register ... 113
3.9.7. gen_purpose1 Register ... 115
3.9.8. led_display Register ... 117
3.9.9. gen_purpose2 Register ... 118

SECTION 4 CIRCUIT SCHEMATICS ... 121

4. General Index to the CPU-5CE Schematics ... 121
4.1. Signal and Unit Cross References ... 122
4.2. IOBP-10 Schematics ... 123

SECTION 5 OPENBOOT ENHANCEMENTS ... 125

5. OpenBoot ... 125
5.1. Controlling the VMEbus Master and Slave Interface ... 126
5.1.1. VMEbus addressing ... 126
5.1.2. VMEbus Master Interface ... 127
5.1.2.1. SPARC/CPU-5CE ... 127
5.1.3. VMEbus Slave Interface ... 130
5.1.3.1. SPARC/CPU-5CE ... 130
5.2. VMEbus Interface ... 132
5.2.1. Generic Information ... 132
5.2.2. Register Addresses ... 132
5.2.3. Register Accesses ... 134
5.2.4. VMEbus Interrupt Handler ... 137
5.2.5. VMEbus Arbiter ... 139
5.2.6. VMEbus Requester ... 139
5.2.7. VMEbus Status Signals ... 141
5.2.8. VMEbus Master Interface ... 143

5.2.9. VMEbus Slave Interface 145
5.2.10. VMEbus Device Node 148
5.2.11. Mailboxes 153
5.3. System Configuration 154
5.3.1. Watchdog Timer 154
5.3.2. Abort Switch 156
5.3.3. Seven Segment LED Display and Rotary Switch 156
5.3.4. Miscellanea 157
5.4. Flash Memory Support 158
5.4.1. Flash Memory Programming 158
5.4.2. Loading and Executing Programs from USER Flash Memory 163
5.4.3. Controlling the Flash Memory Interface 164
5.5. On-board Interrupts 165
5.5.1. VMEbus Interrupts 165
5.5.2. SYSFAIL Interrupt 166
5.5.3. ACFAIL Interrupt 167
5.5.4. ABORT Interrupt 168
5.5.5. Watchdog Timer Interrupt 168
5.6. Further Commands 169
5.7. Frequently Asked Questions 169
5.8. BusNet Support 170
5.8.1. Limitations 170
5.8.2. Loading Programs 170
5.8.3. The BusNet Device 171
5.8.3.1. Device Properties 171
5.8.3.2. Device Methods 173
5.8.3.3. NVRAM Configuration Parameters 174
5.8.4. Device Operation 178
5.8.5. How to Use BusNet 179
5.8.6. Using `bn-dload` to Load from the Backplane 181
5.8.7. Booting from a Solaris/SunOS BusNet Server 182
5.8.8. Booting from a VxWorks BusNet Server 183
5.8.9. Setting NVRAM Configuration Parameters 185

SECTION 6 SUN OPEN BOOT DOCUMENTATION 187

6. Insert your OPEN BOOT 2.0 PROM MANUAL SET here 187

List of Figures

Figure 1. Block Diagram of the SPARC CPU-5CE 3
Figure 2. Diagram of the CPU-5CE 13
Figure 3. Highlighted Diagram of the CPU-5CE 14
Figure 4. Parallel Port Interface Via VME P2 Connector 20
Figure 5. Ethernet Interface Availability 21
Figure 6. Diagram of the Front Panel 33
Figure 7. Pinout of theEthernet Cable Connector 36
Figure 8. Serial Ports A and B Connector Pinout 37
Figure 9. Keyboard/Mouse Connector 38
Figure 10. The IOBP-10 41
Figure 11. The 48-bit (6-byte) Ethernet address 48
Figure 12. The 32-bit (4-byte) host ID 48
Figure 13. Block Diagram of the SPARC CPU-5CE 49
Figure 14. Diagram of the SPARC CPU-5CE Board 50
Figure 15. Highlighted Diagram of the CPU-5CE 51
Figure 16. Ethernet Interface Availability 59
Figure 17. Parallel Port Configuration Via P2 Connector 60
Figure 18. Method A Line Termination & Resistor Network 69
Figure 19. Method B Line Termination & Resistor Network 70
Figure 20. Method C Line Termination & Resistor Network 71
Figure 21. Floppy Disk Interface Via P2 Connector 73
Figure 22. Front Panel 97
Figure 23. Segments of the Hex Display 101
Figure 24. Address translation (master): microSPARC – SBus – VMEbus 128
Figure 25. Mapping a VMEbus area to the processor’s virtualaddress space 129
Figure 26. Address translation (slave): VMEbus – SBus – microSPARC 130

List of Tables

Table 1. Specifications of the SPARC CPU-5CE 4
Table 2. Product Nomenclature 6
Table 3. Ordering Information 7
Table 4. History of Manual 9
Table 5. Default Switch Settings 15
Table 6. Device Alias Definitions 24
Table 7. Setting Configuration Parameters 25
Table 8. Diagnostic Routines 26
Table 9. Commands to Display System Information 29
Table 10. Features of the Front Panel 34
Table 11. SPARC CPU-5CE Connectors 35
Table 12. Ethernet the Connector Pinout 36
Table 13. Serial Ports A and B Connector Pinout for RS-232 37
Table 14. Keyboard/Mouse Connector Pinout 38
Table 15. VME P2 Connector Pinout 39
Table 16. IOBP-10 P1 Pinout 42
Table 17. IOBP-10 P2 Pinout (SCSI) 44
Table 18. IOBP-10 P3 Pinout (Floppy) 45
Table 19. IOBP-10 P4 Pinout (Centronics) 46
Table 20. IOBP-10 P5 Pinout (Serial) 47
Table 21. IOBP-10 Pinout (Ethernet) 47
Table 22. Physical Memory Map of microSPARC-II 53
Table 23. CPU-5CE Memory Banks 54
Table 24. Physical Memory Map of SBus on SPARC CPU-5CE 55
Table 25. NCR89C105 Chip Address Map 62
Table 26. RS-232, RS-422 or RS-485 Configuration 63
Table 27. Serial Ports A and B Pinout List (RS-232) 64
Table 28. Switch Settings for Ports A and B (RS-232) 64
Table 29. Serial Ports A and B Pinout List (RS-422) 65
Table 30. Switch Settings for Ports A and B (RS-422) 66
Table 31. Serial Ports A and B Pinout List (RS-485) 67
Table 32. Switch Settings for Ports A and B (RS-485) 67
Table 33. Transmission Line Termination 68
Table 34. Signal Resistor Termination (Method C) for RS-422 72
Table 35. Signal Resistor Termination (Method C) for RS-485 72
Table 36. 8-Bit Local I/O Devices 74
Table 37. Boot EPROM Capacity 75
Table 38. User Flash EPROM Capacity 76
Table 39. +12V Programming Voltage Control Bit 77

Table 40. Flash Memory Programming Control Bits 79
Table 41. VMEbus Master Interface Physical Address Map..... 81
Table 42. VMEbus Address Ranges 82
Table 43. Supported Address Modifier Codes..... 82
Table 44. Address Modifier Supervisory Bit..... 83
Table 45. VMEbus Master Interface Transfer Cycles 84
Table 46. Slave Mode Bit 85
Table 47. DVMA Enable Bit 85
Table 48. Slave Address Mode Bit 86
Table 49. Window Size Bits 88
Table 50. S4-VME Chip Physical Address Map 91
Table 51. SYSFAIL Non-Maskable Interrupt Pending Bit 92
Table 52. SYSFAIL to VMEbus Bit..... 93
Table 53. SYSFAIL Status Bit 93
Table 54. ACFAIL Non-Maskable Interrupt Pending Bit 94
Table 55. ACFAIL Status Bit 94
Table 56. VMEbus Bus Timer..... 95
Table 57. Features of the Front Panel..... 96
Table 58. Abort Non-Maskable Interrupt Pending Bit 99
Table 59. Rotary Switch Settings 106
Table 60. Interrupt Mapping..... 153

SECTION 1 INTRODUCTION

1. Getting Started

This *SPARC CPU-5CE Technical Reference Manual* provides a comprehensive guide to the SPARC CPU-5CE 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-5CE 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-5CE Technical Reference Manual Set

When purchased from FORCE, this set includes the *SPARC CPU-5CE Technical Reference Manual* as well as two additional books. These two books are listed here:

Set of Data Sheets for the SPARC CPU-5CE

OPEN BOOT PROM 2.0 MANUAL SET

The *Set of Data Sheets for the SPARC CPU-5CE* contains the following data sheets.

NCR SBus I/O Chipset Data Manual	Sun Microsystems S4 Chip Set (Rev.4)
microSPARC-II Data Sheet	AMD Flash EPROM (AM28F020)
SGS-THOMSON MK48T08(B)-10/12/15/20	Intel Flash Memory (28F008SA-L)

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

Open Boot 2.0 Quick Reference	FCODE Programs
Open Boot 2.0 Command Reference	

1.2 Summary of the SPARC CPU-5CE

The SPARC CPU-5CE is a VMEbus board based on the microSPARC-II CPU chip which is a highly integrated implementation of the SPARC RISC microprocessor.

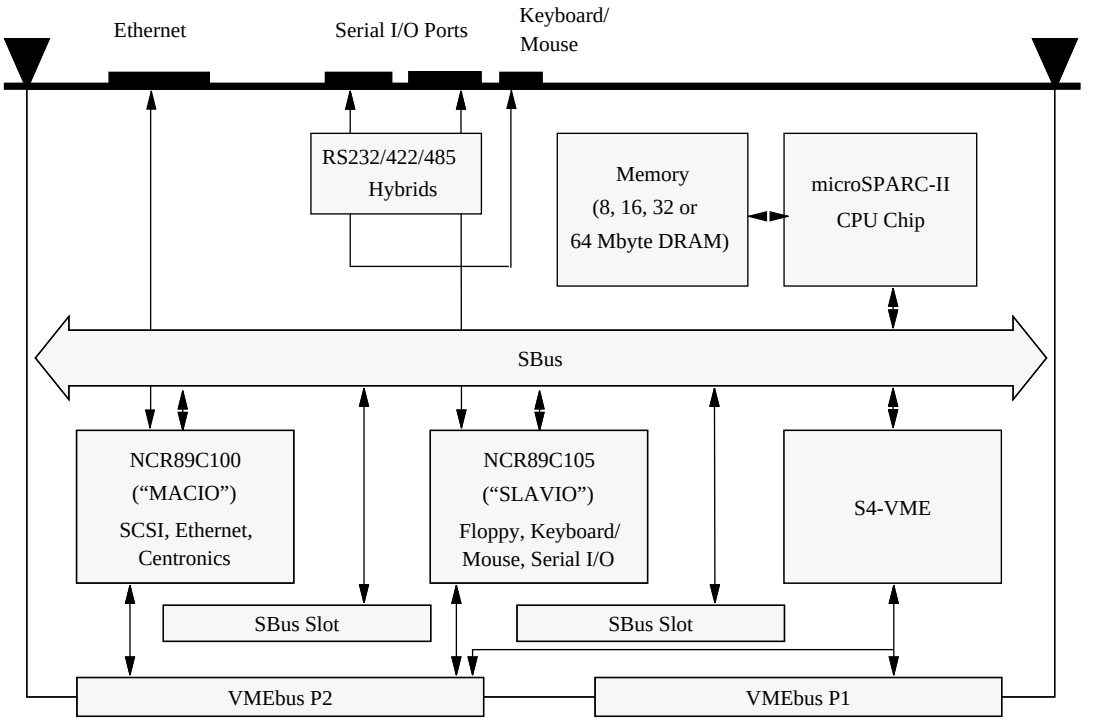
Through a combination of powerful processing power with a full set of I/O interfaces including fast SCSI, Ethernet, floppy disk, serial I/O, Centronics compliant parallel I/O and keyboard/ mouse ports, the SPARC CPU-5CE becomes a high performance cost effective solution for embedded applications.

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-5CE includes an EPROM based monitor/debugger called OpenBoot™, which provides the functionality of the boot device as well as the setup for the VMEbus interface. The software support for the SPARC CPU-5CE ranges from Solaris™, the most popular implementation of the UNIX operating system, to sophisticated hard real-time operating systems such as VxWorks.

The SPARC CPU-5CE is a single board computer combining workstation performance and functionality with the ruggedness and expandability of an industry standard 6U VMEbus card.

FIGURE 1. Block Diagram of the SPARC CPU-5CE



1.3 Board Components

As is shown in the above diagram, the microSPARC-II chip interfaces directly to a 64-bit wide DRAM on the one side and to the SBus on the other side. The SPARC CPU-5CE is available with 8, 16, 32, or 64 Mbytes of on-board DRAM. The shared DRAM is 64-bit wide with 2 bit parity.

The SPARC CPU-5CE utilizes the Sun S4-VME chip to provide a complete 32-bit VMEbus interface. Using SBus modules, the board becomes a VMEbus two-slot solution.

The SCSI interface, the Ethernet interface, and a parallel port are realized via the NCR89C100 (MACIO).

The floppy disk interface, two serial I/O ports, the keyboard/mouse interface are provided by the NCR89C105 chip (SLAVIO), which additionally controls the boot EPROM, the RTC and NVRAM, and a user EPROM via its 8-bit expansion port.

1.4 Specifications

Below is a table outlining the specifications of the SPARC CPU-5CE board.

Table 1: Specifications of the SPARC CPU-5CE

Processor Clock Frequency SPECint92 SPECfp92 MIPS MFLOPS	microSPARC-II 85 MHz 64.0 54.6 112.5 14.9
Memory Management Unit	SPARC Reference MMU
Data/Instruction Cache	8 Kbyte/16 Kbyte
Shared Main Memory	8, 16, 32 or 64 Mbyte DRAM
SBus Slots	2, mechanically compatible to CPU-2CE and CPU-3CE
SCSI with DMA to SBus	NCR89C100 10 Mbytes/sec 53C90A superset I/O on 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 3.4 Mbytes/sec Centronics compatible Uni- or bidirectional I/O on P2 via switch matrix
Floppy Disk Interface	NCR89C105 250, 300, 500 Kbytes/sec and 1 Mbytes/sec 82077AA-1 compatible I/O on P2 via switch matrix
Serial I/O	NCR89C105 Two ports with RS-232 configuration, Optional RS-422/485 via hybrid modules 8530 compatible I/O on front panel and P2
Keyboard/Mouse Port	Sun compatible, on front panel
Counters/Timers	Two 22-bit, programmable

Table 1: Specifications of the SPARC CPU-5CE (Continued)

Boot Flash Memory	512 Kbyte (1 Mbyte Option) On-board programmable Hardware write protection
User Flash Memory	2 Mbyte (optional) On-board programmable Hardware write protection
RTC/NVRAM/Battery Usable Memory	MK48T08 6 Kbyte
VMEbus Interface	32-bit master/slave, IEEE-1014
Additional Features	Reset and Abort switches Status LEDs on the front panel HEX display on the front panel Watchdog timer
Firmware	OpenBoot with diagnostics
Power consumption	+5V 5.0 A (No SBus Module installled) +12V 0.7A -12V 0.2A
Environmental Conditions Temperature (Operating) Temperature (Storage) Humidity	 0° C to +50° C -40° C to +85° C -20° C to +70° C (with installed RTC) 0% to 95% noncondensing
Board Size	Single Slot 6U VMEbus 160.00 x 233.35 mm 6.29 x 9.18 inches

1.5 Product Nomenclature

FORCE COMPUTERS' SPARC CPU-5CE 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-5CE/xx-yy-z		
Main Memory (xx)	microSPARC-II Clock Frequency (yy)	User EPROM (z)
8 = 8 Mbyte 16 = 16 Mbyte 32 = 32 Mbyte 64 = 64 Mbyte	85 = 85MHz	0 = 0 Mbyte 1 = 1 Mbyte 2 = 2 Mbyte
Sample Product Nomenclature		
CPU-5CE/16-85-0	85 MHz microSPARC-II CPU board with 16-Mbyte DRAM, 0-Mbyte user flash EPROM, SCSI, Ethernet, floppy disk, parallel and 2 serial I/O ports, 32-bit VMEbus interface, 2 SBus slots, OpenBoot firmware. User documentation included.	
Other Available Configurations		
CPU-5CE/16-85-2	16 Mbyte, (as above, except 2 Mbyte user flash EPROM)	
CPU-5CE/32-85-2	32 Mbyte, (as above, except 32 Mbyte DRAM)	
CPU-5CE/64-85-2	64 Mbyte, (as above, except 64 Mbyte DRAM)	

1.5.1 Ordering Information

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

Table 3: Ordering Information

Catalog Name	Product Description
SBus Modules	
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/TGX+	Color 2D and 3D wire frame high perfomance graphics accelerator, up to 1600 x 1280, 8 bits per pixel, double buffering, single SBus slot.
SBus/FP	Front panel for up to 2 SBus cards for CPU-2CE, CPU-3CE, CPU-5CE and CPU-10.
Accessories	
CPU-5CE/TM	Technical Reference Manual Set for CPU-5CE including OpenBoot User's Manual and a Set of Data Sheets.
IOBP-10	I/O back panel on VMEbus P2 with flat cable connectors for Ethernet, SCSI, serial I/O and Centronics/floppy interface. For use with CPU-3CE, CPU-5CE and CPU-10.
Serial-2CE	Serial adapter cable for CPU-2CE, CPU-3CE, CPU-5CE and CPU-10, 26-pin shielded to 25-pin D-Sub.
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-5CE	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-5CE/ Client -RTU	Solaris 2.x client/desktop right-to-use license. Without media. Please contact your local sales representative for current version information.
Solaris 2.x/CPU-5CE/ 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.

Table 3: Ordering Information (Continued)

Catalog Name	Product Description
Solaris 1.1/CPU-5CE	Solaris 1.1 two-user right-to-use license with VMEbus driver. Includes media.
Solaris 1.1/CPU-5CE/2U-RTU	Solaris 1.1 two-user right-to-use license. Without media.
Solaris 1.1/CPU-5CE/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-5CE	VxWorks board support package for CPU-5CE
VxWorks/Solaris Driver CPU-5CE	VxWorks/Solaris CPU-5CE back plane driver.

1.6 History of the Manual

Below is a description of the publication history of this *SPARC CPU-5CE Technical Reference Manual.*

Table 4: History of Manual

Revision No.	Description	Date of Last Change
1	First Print	July 1994
2	Editorial Changes	September 1995
3	BusNet, Controlling the VMEbus Master and Slave Interface, and How to Install the OpenBoot ROM sections have been added. The Ethernet and Host ID chapter has been updated.	October 1996

SECTION 2 INSTALLATION

2. Introduction

This Installation Section provides guidelines for powering up the SPARC CPU-5CE board. The Installation Section, which you have in your hand now, appears both as Section 2 of the *SPARC CPU-5CE Technical Reference Manual* and as a stand-alone *Installation Guide*. This stand-alone Installation Guide is delivered by FORCE COMPUTERS with every board. *The SPARC CPU-5CE 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



Read the following safety note before handling the board.

To ensure proper functioning of the product over its usual lifetime, take the following precautions before handling the board.

Electrostatic discharge and incorrect board installation and uninstallation can damage circuits or shorten their lifetime.

- Before installing or uninstalling the board, read this *Installation* section.
- Before installing or uninstalling the board in a VME rack:
  - Check all installed boards for steps that you have to take before turning off the power.
  - Take those steps.
  - Finally turn off the power.
- Before touching integrated circuits, ensure that you are working in an electrostatic free environment.
- Ensure that the board is connected to the VMEbus via both connectors, the P1 and the P2 and that power is available on both.
- When operating the board in areas of strong electro-magnetic radiation, ensure that the board
  - is bolted on the VME rack
  - and shielded by closed housing.

2.2 Location Diagram of the SPARC CPU-5CE Board

A location diagram showing all the components of the CPU-5CE appears on the next page. On the page next to it, there is a location diagram of the CPU-5CE which highlights components that are of particular interest to the user.

FIGURE 2. Diagram of the CPU-5CE

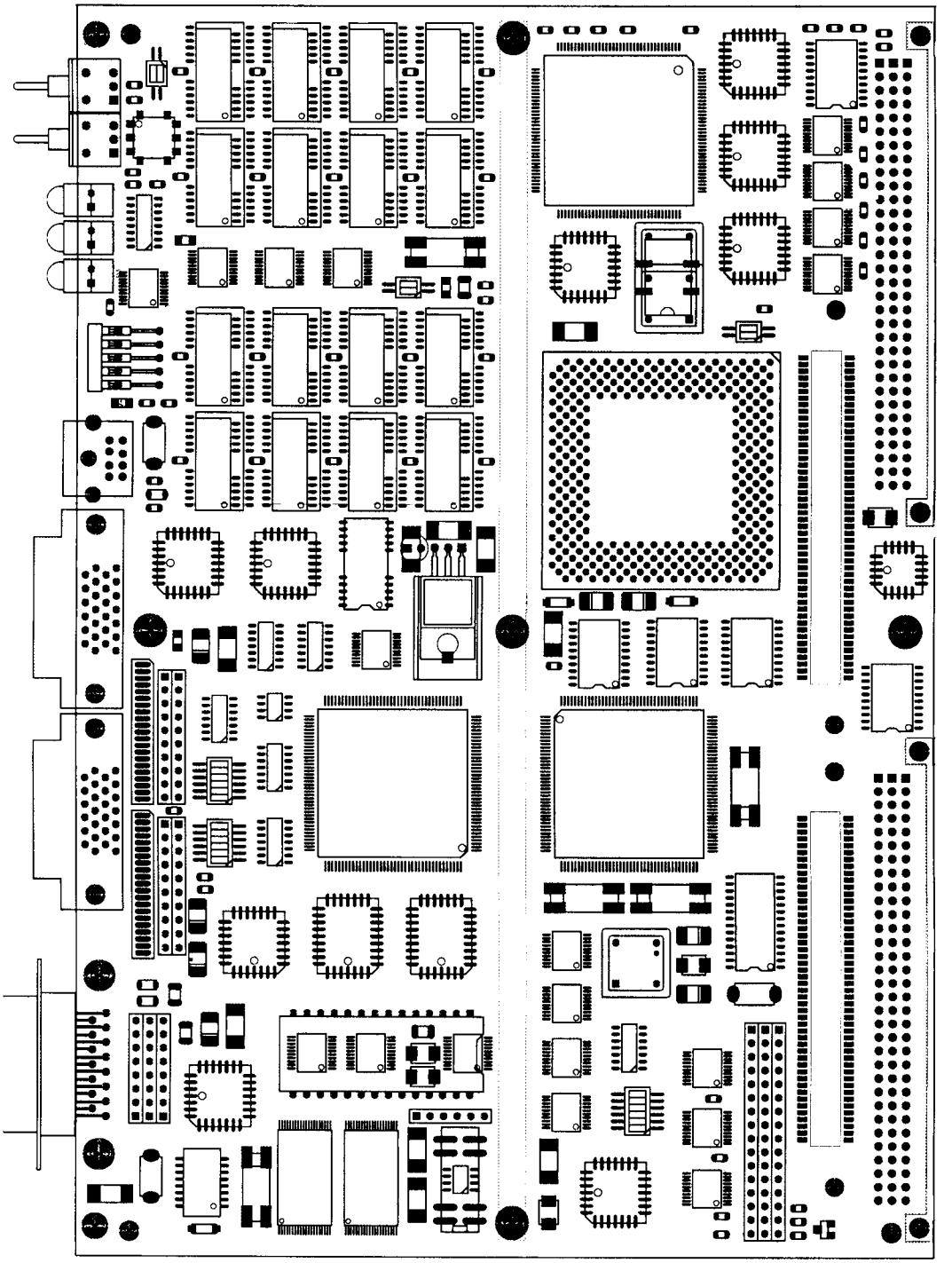
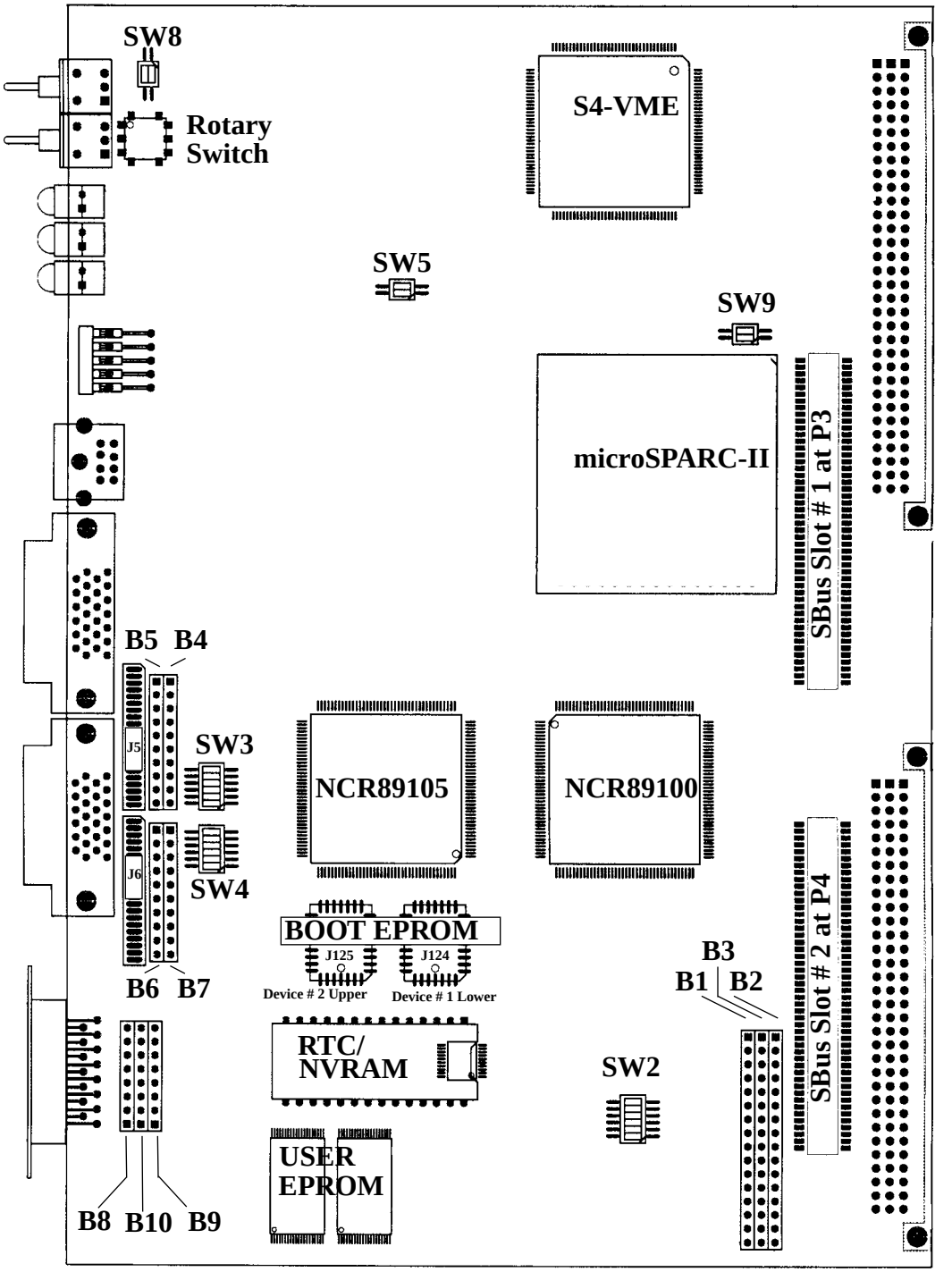


FIGURE 3. Highlighted Diagram of the CPU-5CE



2.3 Before Powering Up

Before powering up, please make sure that the default switch settings are all set according to the table below. Check these switch settings *before* powering up the SPARC CPU-5CE because the board is configured for power up according to these default settings. For the position of the switches on the board, please see the "Highlighted Diagram of the CPU-5CE" on page 14. Now is an excellent time to examine the switches to confirm that they are correctly set.

2.3.1 Default Switch Settings

Table 5: Default Switch Settings

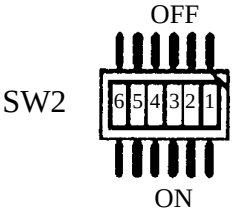
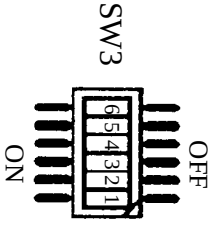
Diagram of Switch	Switches	Default Setting	Function
SWITCH 2			
	SW2-6	ON	**SCSI termination** ON = enable, OFF = disable
	SW2-5	OFF	**Test Switch**, must be OFF
	SW2-4	OFF	**User Flash EPROM write protection** ON = disable, OFF = enable
	SW2-3	OFF	**Boot Flash EPROM write protection** ON = disable, OFF = enable
	SW2-2	ON	**Test Switch**, must be ON
	SW2-1	OFF	**Test Switch**, must be OFF
SWITCH 3 (Controls Serial Channel A)			
	SW3-1	ON	TRXC on Front Panel Connector for RS-232 ON=Available, OFF=Not Available
	SW3-2	OFF	RTS functions as TEN for RS-485 ON=TEN function enabled OFF=TEN function disabled
	SW3-3	OFF	TRXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW3-4	ON	RTS on Front Panel Connector for RS-232 or RTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW3-5	ON	CTS on Front Panel Connector for RS-232 or CTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW3-6	OFF	RTXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available

Table 5: Default Switch Settings (Continued)

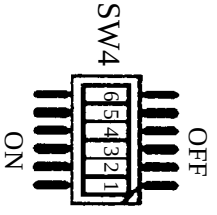
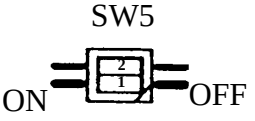
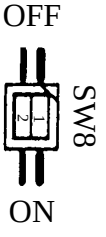
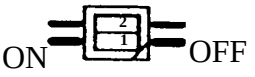
Diagram of Switch	Switches	Default Setting	Function
SWITCH 4 (Controls Serial Channel B)			
	SW4-1	ON	TRXC on Front Panel Connector for RS-232 ON=Available, OFF=Not Available
	SW4-2	OFF	RTS functions as TEN for RS-485 ON=TEN function enabled OFF=TEN function disabled
	SW4-3	OFF	TRXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW4-4	ON	RTS on Front Panel Connector for RS-232 or RTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW4-5	ON	CTS on Front Panel Connector for RS-232 or CTS +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
	SW4-6	OFF	RTXC +/- on Front Panel Connector for RS-422 ON=Available, OFF=Not Available
SWITCH 5			
	SW5-2	OFF	**Test Switch**, must be OFF
	SW5-1	ON	**VMEbus Slot1 Device** ON = Slot-1 Device, OFF = Not Slot-1 Device
SWITCH 8			
	SW8-2	ON	**Abort Key Control** ON=Abort Key enable, OFF=Abort Key disable
	SW8-1	ON	**Reset Key Control** ON=Reset Key enable, OFF=Reset Key disable

Table 5: Default Switch Settings (Continued)

Diagram of Switch	Switches	Default Setting	Function
SWITCH 9			
SW9  	SW9-2 SW9-1	ON ON	**External VMEbus SYSRESET** ON = VMEbus SYSRESET generates on-board RESET OFF = VMEbus SYSRESET does not generate on-board RESET **VMEbus SYSRESET generation** ON = SYSRESET is driven to VMEbus, if board is Slot-1 Device OFF = SYSRESET is not driven to VMEbus

CAUTION: To avoid damaging the serial ports, please consider the following regarding Switch 3 and Switch 4. Do not set the switches (SW3-1 and SW3-2), or (SW3-3 and SW3-4), or (SW3-5 and SW3-6) to ON at the same time and do not set the switches (SW4-1 and SW4-2), or (SW4-3 and SW4-4), or (SW4-5 and SW4-6) to ON at the same time!

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 “Boot the System” on page 22 for more detailed information on booting the system.

2.4.1 VME Slot-1 Device

The SPARC CPU-5CE 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-5CE so it is not a VME slot-1 device, the default configuration must be changed so that SW5-1 is OFF.

CAUTION

Before installing the SPARC CPU-5CE in a miniforce chassis, please first disable the VMEbus System Controller function by setting switch SW5-1 to OFF.

2.4.2 VMEbus SYSRESET Switches

When the SPARC CPU-5CE is a VMEbus slot-1 device, it generates the SYSRESET signal to the VMEbus. This can be disabled by setting the switch SW9-1 to OFF.
An external SYSRESET generates an on-board RESET in the default switch setting, i.e., SW9-2 is ON. When SW9-2 is OFF, the external SYSRESET does not generate an on-board RESET.

2.4.3 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 page 15 shows the necessary switch settings for RS-232 operation, where SW3 controls serial port A and SW4 controls serial port B. Please check that the switches are set accordingly.

2.4.4 RESET and ABORT Key Enable

To enable the RESET and the ABORT functions on the front panel, set switches SW8-1 (RESET) and SW8-2 (ABORT) to ON.

2.4.5 SCSI Termination

Termination for the SCSI interface is enabled when SW2-6 is ON. This is the default setting.

CAUTION

Before installing the SPARC CPU-5CE in a microforce chassis, please first disable the SCSI termination by setting switch SW2-6 to OFF.

2.4.6 Boot Flash EPROM Write Protection

Both Boot Flash EPROMs are write protected via the switch SW2-3. When SW2-3 is OFF, the devices are write protected.

2.4.7 User Flash EPROM Write Protection

The optional User Flash EPROMs are write protected via SW2-4. When SW2-4 is OFF, the User Flash EPROMs are write protected.

2.4.8 Reserved Switches

SW2-1, SW2-2 , SW2-5 and SW5-2 are reserved for test purposes. SW2-1, SW2-5 and SW5-2 should always be OFF. SW2-2 should always be ON.

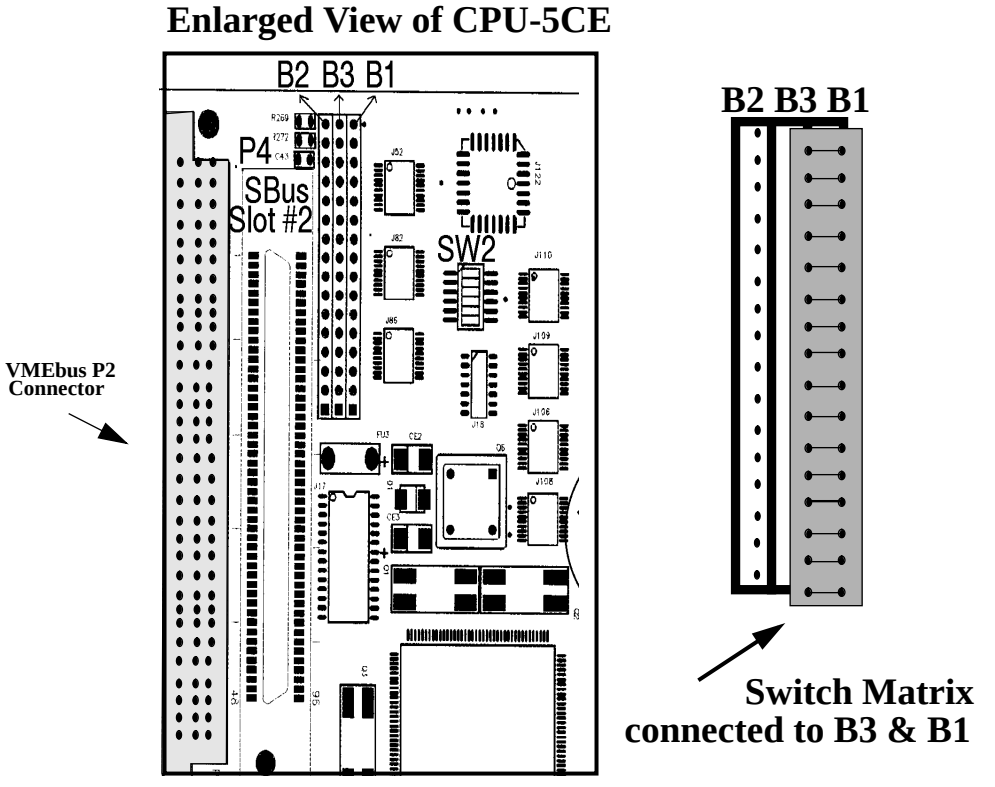
2.4.9 Parallel Port or Floppy Interface via VME P2 Connector

Via a 16-pin configuration switch matrix, it is possible for either the parallel port interface or the floppy interface to be available on the VME P2 connector.

The default setting enables the floppy interface via the VME P2 connector, with the configuration switch matrix plugged into B2 and B3. This means, of course, that by default the parallel port interface is not available via the VMEbus P2 connector.

To enable the parallel port interface via the VME P2 connector, plug the configuration switch matrix in sockets B1 and B3. The following drawing shows this configuration.

FIGURE 4. Parallel Port Interface Via VME P2 Connector



The switch matrix plugs into connectors B1 and B3 and the parallel port is accessible through the VMEbus P2 connector.

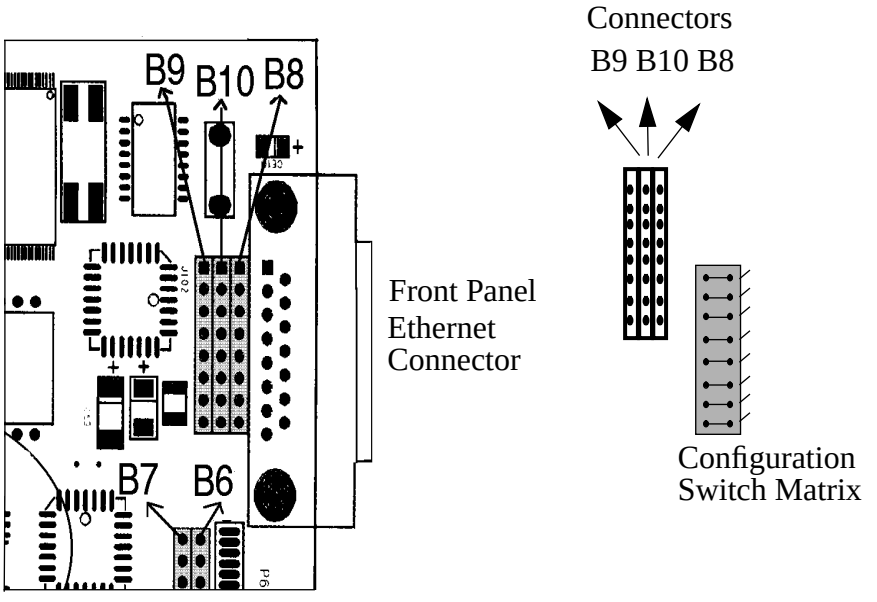
2.4.10 Ethernet via Front Panel or VME P2 Connector

Via an 8-pin configuration switch matrix, it is either possible for the Ethernet interface to be available via the front panel or the VME P2 connector. The default configuration provides the Ethernet through the front panel connector.

In order to have the Ethernet interface accessible via the VME P2 connector, the default configuration must be changed.

Take a moment to examine the diagram to see how one achieves the desired configuration.

FIGURE 5. Ethernet Interface Availability



By default, the Ethernet interface is available through the front panel with the configuration switch matrix plugged into connectors B9 and B10.

To configure the Ethernet interface to be accessible from the VMEbus P2 connector, the configuration switch matrix must be plugged into connectors B8 and B10.

WARNING

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

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 detailed information concerning OpenBoot, please see the *OPEN BOOT PROM 2.0 MANUAL SET*. This manual is included in the *SPARC CPU-5CE Technical Reference 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 SelfTest (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 non-volatile 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 6: 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 7: 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 8: 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 MICROP 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 MICROP 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 6, “Device Alias Definitions,” on page 24.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 9: 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 SelfTest (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 on-line 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)
Sub-categories 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 on-line help shows you the forth word, the parameter stack before and after execution of the forth word (before -- after), and a short description.

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

2.5.7 How to Install an OpenBoot ROM

This section describes how to install the OpenBoot ROM delivered to you by FORCE COMPUPTERS. This information is useful in the case that OpenBoot ROM is exchanged or upgraded.

The latest version of OpenBoot for your SPARC CPU-5CE includes BusNet Support. Here follows a step by step description of how to replace the OpenBoot ROMs.

For the location of the OpenBoot ROMs on the board, see “Highlighted Diagram of the CPU-5CE” on page 14.

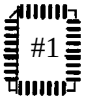
Caution



Avoid touching integrated circuits except in an electrostatic free environment. Electrostatic discharge can damage circuits or shorten their lifetime. Turn off the power before handling the board or its components. Turn off the power before installing the board in a VMEbus rack.

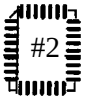
1 Replacing the ROMs

1. Remove the existing OpenBoot ROM from socket J124 and store it in a safe place.



2 Insert the new OpenBoot ROM labeled ROM #1 (2.15.2) in socket J124. Note the position of the diagonally cut edge on the ROM and place the new ROM on the board accordingly.

3. Remove the existing OpenBoot ROM from socket J125 and store it in a safe place.

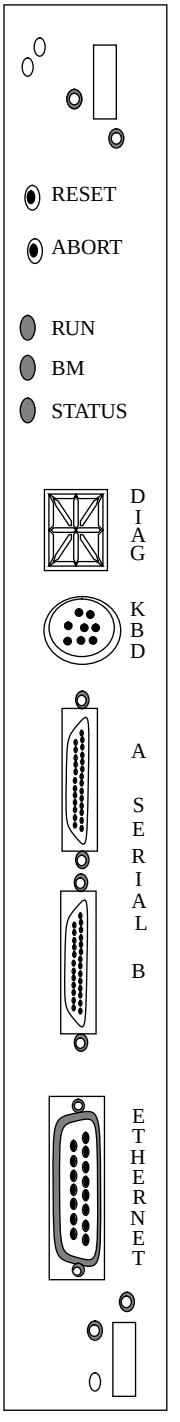


4. Insert the new OpenBoot ROM labeled ROM #2 (2.15.2) in socket J125. Note the position of the diagonally cut edge on the ROM and place the new ROM on the board accordingly.

5. Install the SPARC CPU-5CE in the VMEbus rack. You are now ready to power up.

2.6 Front Panel

FIGURE 6. Diagram of the Front Panel



2.6.1 Features of the Front Panel

- Reset and Abort key
- Status LEDs on the front panel
- Hex display on the front panel

These features are described in detail in Section 3 of the *SPARC CPU-5CE Technical Reference Manual*.

Table 10: Features of the Front Panel

Device	Function	Name
Switch	Reset	RESET
Switch	Abort	ABORT
LED	RUN/RESET	RUN
LED	VMEbus Bus Master	BM
LED	Status	STATUS
HEX Display	Diagnostic	DIAG
Mini DIN Connector	Keyboard/Mouse	KBD
Serial Connector	Serial Interface	SERIAL A
Serial Connector	Serial Interface	SERIAL B
D-Sub Connector	Ethernet Interface	ETHERNET

2.7 SPARC CPU-5CE Connectors

The connectors on the SPARC CPU-5CE are listed in the following table.

Table 11: SPARC CPU-5CE Connectors

Function	Location	Type	Manufacturer Part Number
Ethernet	Front Panel	15-pin D-Sub	AMP 747845-4
Serial Port A	Front Panel	26-pin Fine Pitch	AMP 749831-2
Serial Port B	Front Panel	26-pin Fine Pitch	AMP 749831-2
Keyboard/Mouse	Front Panel	8-pin Mini DIN	AMP 749232-1
SBus Slot1	P3	96-pin SMD	FUJITSU FCN-234J096-G/V
SBus Slot2	P4	96-pin SMD	FUJITSU FCN-234J096-G/V
VMEbus P1	P1	96-pin VGA	Various
VMEbus P2	P2	96-pin VGA	Various

The following pages show the pinouts of the connectors.

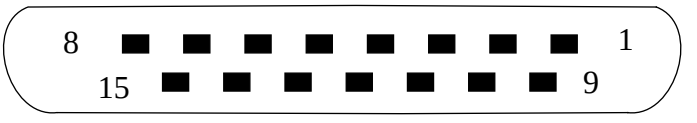
2.7.1 Ethernet Connector Pinout

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

Table 12: Ethernet the Connector Pinout

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

FIGURE 7. Pinout of theEthernet Cable Connector



2.7.2 Serial Ports A and B Connector Pinout

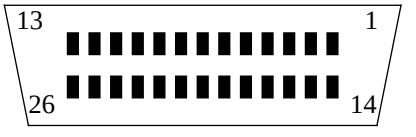
The two serial I/O ports are available on the front panel via two 26-pin shielded connectors which are compatible to the CPU-2CE and to the CPU-3CE.

Both channels are available via the VMEbus P2 connector, 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 serial ports A and B. The table is valid for both serial I/O connectors A and B.

Table 13: 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 8. Serial Ports A and B Connector Pinout



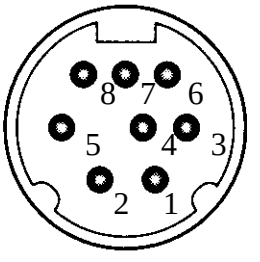
2.7.3 Keyboard/Mouse Connector Pinout

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

Table 14: 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 9. Keyboard/Mouse Connector



2.7.4 VME P2 Connector Pinout

Table 15: VME P2 Connector Pinout

ROW A	Signal	ROW B	Signal	ROW C	Signal for Floppy Interface [1]	Signal for Parallel Port Interface [1]
1	SCSI Data 0	1	+5VDC	1	FPY DENSEL	CENTR DS
2	SCSI Data 1	2	GND	2	FPY DENSENS	CENTR Data 0
3	SCSI Data 2	3	RESERVED	3	N.C.	CENTR Data 1
4	SCSI Data 3	4	VME A24	4	FPY INDEX	CENTR Data 2
5	SCSI Data 4	5	VME A25	5	FPY DRVSEL	CENTR Data 3
6	SCSI Data 5	6	VME A26	6	N.C.	CENTR Data 4
7	SCSI Data 6	7	VME A27	7	N.C.	CENTR Data 5
8	SCSI Data 7	8	VME A28	8	FPY MOTEN	CENTR Data 6
9	SCSI DP	9	VME A29	9	FPY DIR	CENTR Data 7
10	GND	10	VME A30	10	FPY STEP	CENTR ACK
11	GND	11	VME A31	11	FPY WRDATA	CENTR BSY
12	GND	12	GND	12	FPY WRGATE	CENTR PE
13	TERMPWR	13	+5VDC	13	FPY TRACK0	CENTR AF
14	GND	14	VME D16	14	FPY WRPROT	CENTR INIT
15	GND	15	VME D17	15	FPY RDDATA	CENTR ERR
16	SCSI ATN	16	VME D18	16	FPY HEADSEL	CENTR SLCT IN
17	GND	17	VME D19	17	FPY DISKCHG	CENTR SLCT
18	SCSI BSY	18	VME D20	18	FPY EJECT	RESERVED
19	SCSI ACK	19	VME D21	19	+12VDC [2]	+12VDC [2]
20	SCSI RST	20	VME D22	20	GND	GND
21	SCSI MSG	21	VME D23	21	GND	GND
22	SCSI SEL	22	GND	22	ETH REC+ [2]	ETH REC+ [2]
23	SCSI CD	23	VME D24	23	ETH REC- [2]	ETH REC- [2]
24	SCSI REQ	24	VME D25	24	ETH TRA+ [2]	ETH TRA+ [2]

Table 15: VME P2 Connector Pinout (Continued)

ROW A	Signal	ROW B	Signal	ROW C	Signal for Floppy Interface [1]	Signal for Parallel Port Interface [1]
25	SCSI IO	25	VME D26	25	ETH TRA- [2]	ETH TRA- [2]
26	RESERVED	26	VME D27	26	ETH COL+ [2]	ETH COL+ [2]
27	RESERVED	27	VME D28	27	ETH COL- [2]	ETH COL- [2]
28	RESERVED	28	VME D29	28	GND	GND
29	TxD Port A	29	VME D30	29	TxD Port B	TxD Port B
30	RxD Port A	30	VME D31	30	RxD Port B	RxD Port B
31	RTS Port A	31	GND	31	RTS Port B	RTS Port B
32	CTS Port A	32	+5VDC	32	CTS Port B	CTS Port B

Notes:

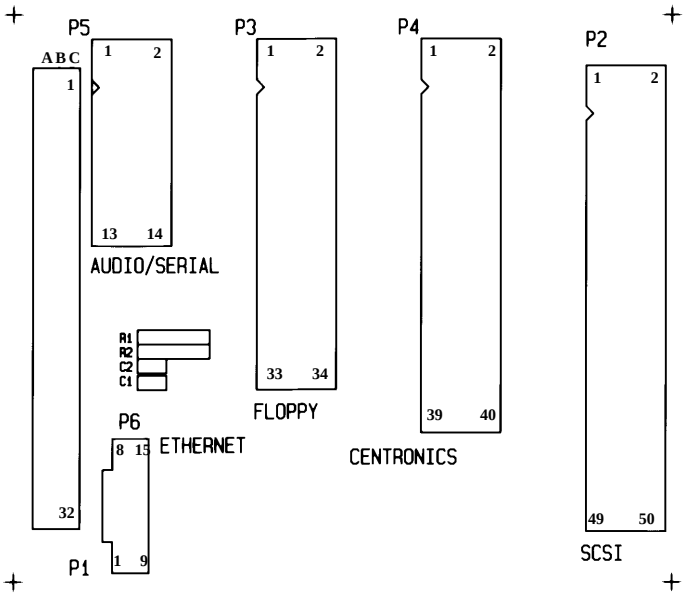
1) For further information, see “Parallel Port Interface Via VME P2 Connector” on page 20.

2) For further information, please see “Ethernet Interface Availability” on page 21.

2.7.5 The IOBP-10 Connectors

The IOBP-10 is an I/O back panel on VMEbus P2 with flat cable connectors for SCSI, serial I/O, Centronics/floppy interface, and a micro D-Sub connector for an Ethernet interface. This back panel can be plugged into the VMEbus P2 connector. The diagram below shows all the connectors. This IOBP-10 back panel is especially designed for the SPARC CPU-5CE. Do not use any other I/O back panels on the SPARC CPU-5CE, for example, the IOBP-1.

FIGURE 10. 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-5CE. Do not use any other I/O back panels on the SPARC CPU-5CE, for example, the IOBP-1.

Table 16: IOBP-10 P1 Pinout

ROW A	Signal	ROW B	Signal	ROW C	Signal for Floppy Interface [1]	Signal for Parallel Port Interface [1]
1	SCSI Data 0	1	N.C.	1	FPY DENSEL	CENTR DS
2	SCSI Data 1	2	GND	2	FPY DENSENS	CENTR Data 0
3	SCSI Data 2	3	N.C..	3	N.C.	CENTR Data 1
4	SCSI Data 3	4	N.C.	4	FPY INDEX	CENTR Data 2
5	SCSI Data 4	5	N.C.	5	FPY DRVSEL	CENTR Data 3
6	SCSI Data 5	6	N.C.	6	N.C.	CENTR Data 4
7	SCSI Data 6	7	N.C.	7	N.C.	CENTR Data 5
8	SCSI Data 7	8	N.C.	8	FPY MOTEN	CENTR Data 6
9	SCSI DP	9	N.C.	9	FPY DIR	CENTR Data 7
10	GND	10	N.C.	10	FPY STEP	CENTR ACK
11	GND	11	N.C.	11	FPY WRDATA	CENTR BSY
12	GND	12	GND	12	FPY WRGATE	CENTR PE
13	TERMPWR	13	N.C.	13	FPY TRACK0	CENTR AF
14	GND	14	N.C.	14	FPY WRPROT	CENTR INIT
15	GND	15	N.C.	15	FPY RDDATA	CENTR ERR
16	SCSI ATN	16	N.C.	16	FPY HEADSEL	CENTR SLCT IN
17	GND	17	N.C.	17	FPY DISKCHG	CENTR SLCT
18	SCSI BSY	18	N.C.	18	FPY EJECT	RESERVED
19	SCSI ACK	19	N.C.	19	+12VDC [2]	+12VDC [2]
20	SCSI RST	20	N.C.	20	GND	GND
21	SCSI MSG	21	N.C.	21	GND	GND
22	SCSI SEL	22	GND	22	ETH REC+ [2]	ETH REC+ [2]
23	SCSI CD	23	N.C.	23	ETH REC- [2]	ETH REC- [2]
24	SCSI REQ	24	N.C.	24	ETH TRA+ [2]	ETH TRA+ [2]
25	SCSI IO	25	N.C.	25	ETH TRA- [2]	ETH TRA- [2]
26	RESERVED	26	N.C.	26	ETH COL+ [2]	ETH COL+ [2]

Table 16: IOBP-10 P1 Pinout (Continued)

ROW A	Signal	ROW B	Signal	ROW C	Signal for Floppy Interface [1]	Signal for Parallel Port Interface [1]
27	RESERVED	27	N.C.	27	ETH COL- [2]	ETH COL- [2]
28	RESERVED	28	N.C.	28	GND	GND
29	TxD Port A	29	N.C.	29	TxD Port B	TxD Port B
30	RxD Port A	30	N.C.	30	RxD Port B	RxD Port B
31	RTS Port A	31	GND	31	RTS Port B	RTS Port B
32	CTS Port A	32	N.C.	32	CTS Port B	CTS Port B

Notes:

1) For further information, see “Parallel Port Interface Via VME P2 Connector” on page 20.

2) For further information, please see “Ethernet Interface Availability” on page 21.

Table 17: 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	GND	26	TERMPWR
27	N.C.	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 18: IOBP-10 P3 Pinout (Floppy)

Pin No.	Signal	Pin No.	Signal
1	FPY EJECT	2	FPY DENSEL
3	GND	4	FPY DENSENS
5	GND	6	N.C.
7	GND	8	FPY INDEX
9	GND	10	FPY DRVSEL
11	GND	12	N.C.
13	GND	14	N.C.
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	N.C.	28	FPY WRPROT
29	GND	30	FPY RDDATA
31	GND	32	FPY HEADSEL
33	GND	34	FPY DISKCHG

Table 19: 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 20: IOBP-10 P5 Pinout (Serial)

Pin No.	Signal	Pin No.	Signal
1	GND	2	RESERVED
3	RESERVED	4	RESERVED
5	TxD Port B	6	TxD Port A
7	RxD Port B	8	RxD Port A
9	RTS Port B	10	RTS Port A
11	CTS Port B	12	CTS Port A
13	GND	14	GND

Table 21: IOBP-10 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.

2.8 How to Determine the Ethernet Address and Host ID

In order to see the Ethernet address and host ID, type the following command at the prompt:

```
ok banner
```

The information below explains how the SPARC/CPU-5CE Ethernet address and the host ID are determined.

FIGURE 11. The 48-bit (6-byte) Ethernet address

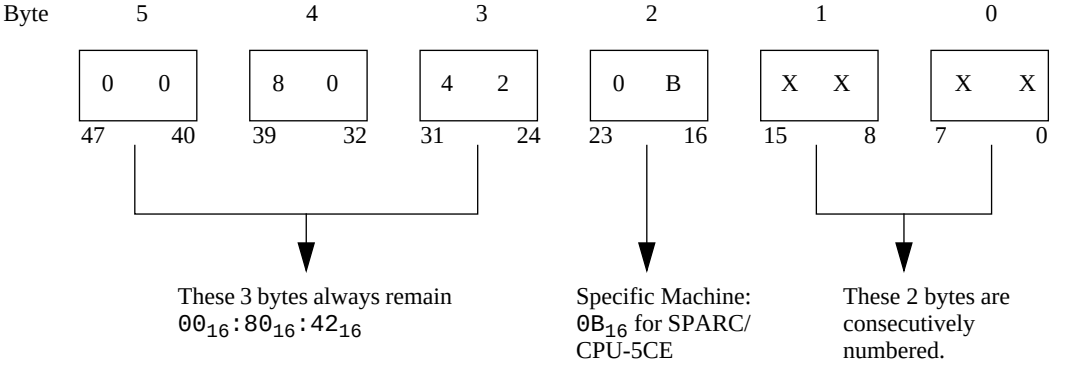
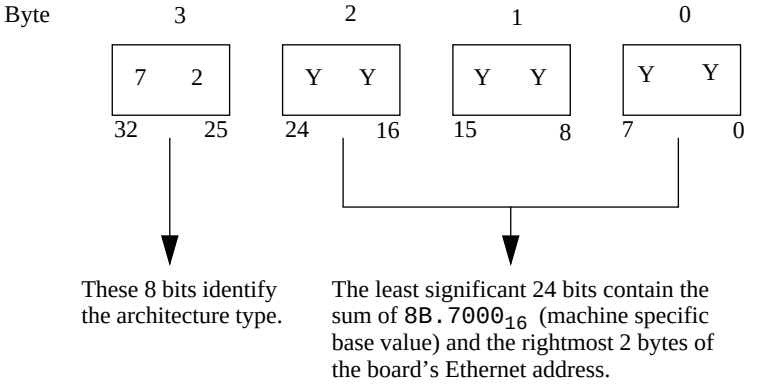


FIGURE 12. The 32-bit (4-byte) host ID



SECTION 3 HARDWARE DESCRIPTION

3. Board Components

As is shown in the block diagram, the microSPARC-II chip interfaces directly to a 64-bit wide DRAM on the one side and to the SBus on the other side. The SPARC CPU-5CE is available with 8, 16, 32, or 64 Mbytes of on-board DRAM. The shared DRAM is 64-bit wide with 2 bit parity.

The SPARC CPU-5CE utilizes the Sun S4-VME chip to provide a complete 32-bit VMEbus interface. Using SBus modules, the board becomes a VMEbus two-slot solution.

The SCSI interface, the Ethernet interface, and a parallel port are realized via the NCR89C100 (MACIO).

The floppy disk interface, two serial I/O ports, the keyboard/mouse interface are provided by the NCR89C105 chip (SLAVIO), which additionally controls the Boot EPROM, the RTC and NVRAM, and a user EPROM via its 8-bit expansion port.

FIGURE 13. Block Diagram of the SPARC CPU-5CE

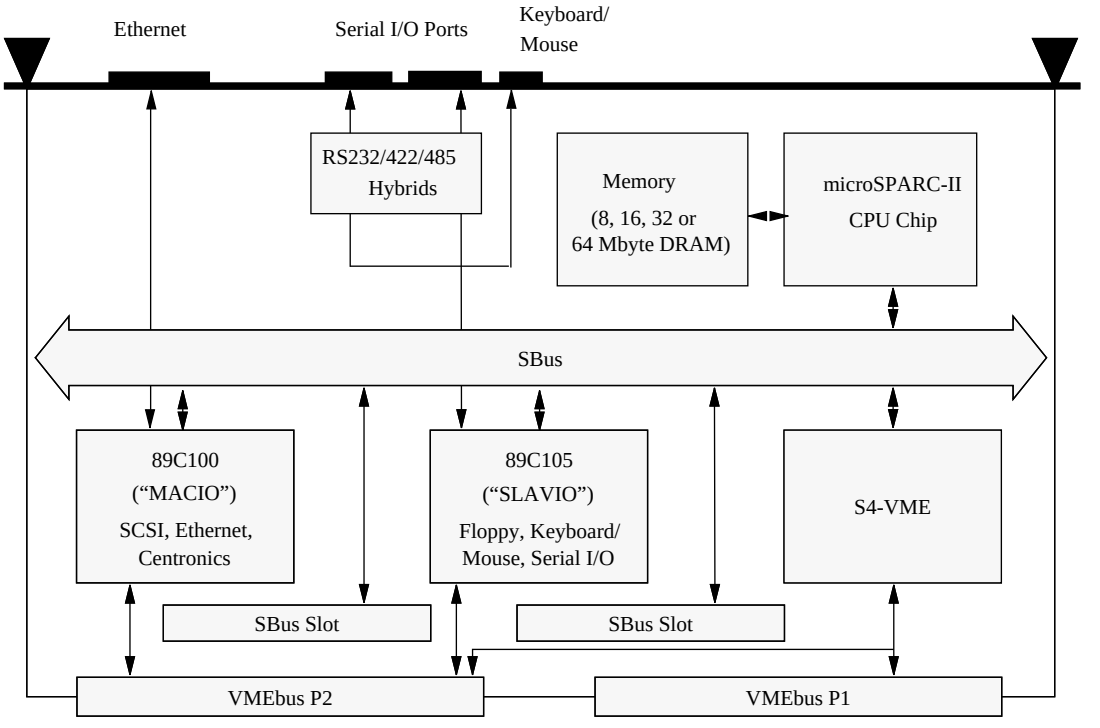


FIGURE 14. Diagram of the SPARC CPU-5CE Board

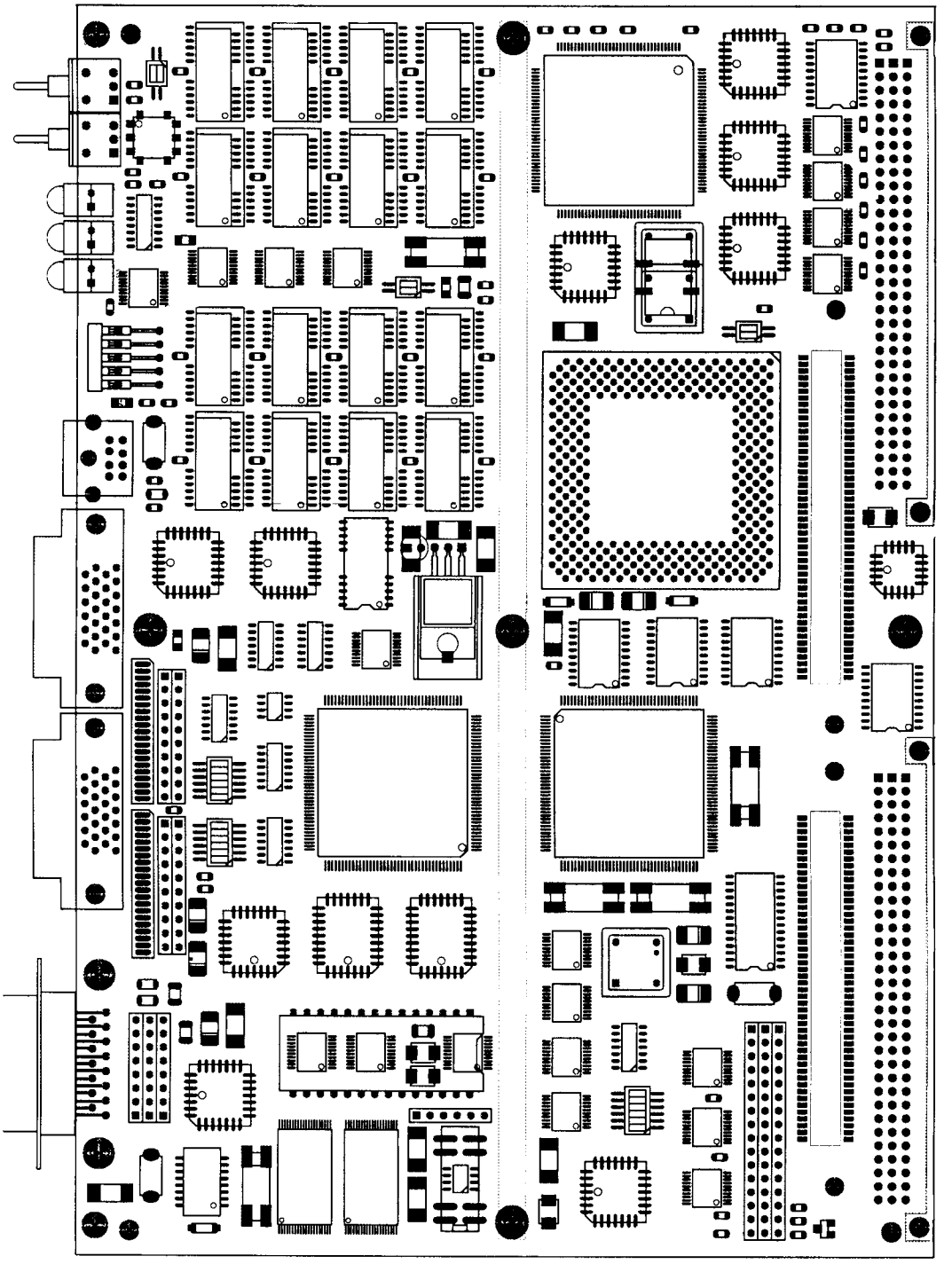
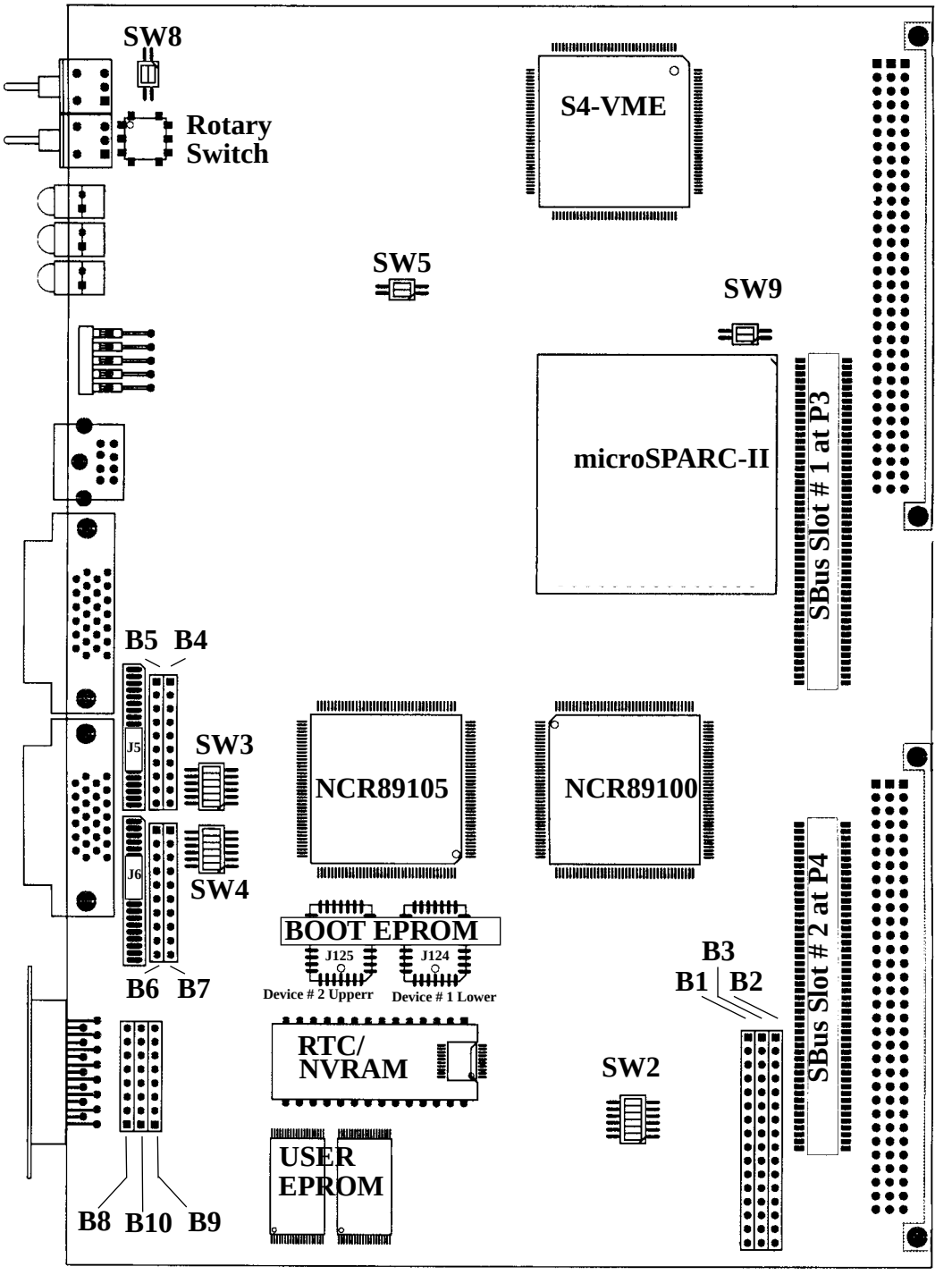


FIGURE 15. Highlighted Diagram of the CPU-5CE



3.1 The microSPARC-II Processor

The microSPARC-II CPU chip is at the core of the SPARC CPU-5CE. This chip is realized in a 321-pin CPGA package . A Floating Point Unit, an Integer Unit, an MMU, an Instruction Cache, and a Data Cache are integrated in the microSPARC-II processor. Please see the microSPARC-II Data Sheet (STP1012) for further information.

3.1.1 Features of the microSPARC-II Processor

- microSPARC-II chip running at 85 MHz
- Integer Unit with 5-stage pipeline
- Floating Point Unit
- SPARC Reference Memory Management Unit
- A 16 Kbyte instruction cache and a 8 Kbyte data cache, directly mapped
- Memory interface which supports up to 256 Mbyte DRAM
- SBus controller supports up to five SBus Slots plus one "master-only" slot

3.1.2 Address Mapping for microSPARC-II

The table below lists the physical addresses of the microSPARC-II processor.

Table 22: Physical Memory Map of microSPARC-II

Address	Function
0000 0000 -> 0FFF FFFF	User Memory
1000 0000 -> 1FFF FFFF	Control Space
2000 0000 -> 2FFF FFFF	AFX Frame buffer
3000 0000 -> 3FFF FFFF	SBus Slave Select 0
4000 0000 -> 4FFF FFFF	SBus Slave Select 1
5000 0000 -> 5FFF FFFF	SBus Slave Select 2
6000 0000 -> 6FFF FFFF	SBus Slave Select 3
7000 0000 -> 7FFF FFFF	SBus Slave Select 4

3.2 The Shared Memory

The microSPARC-II chip interfaces directly to a 64-bit wide DRAM on one side and to the SBus on the other side. The shared DRAM is 64-bit wide with one parity bit for 32 bit data.

The SPARC CPU-5CE provides 8, 16, 32, or 64 Mbyte DRAM which is assembled on the board itself.

There are 4-Mbit devices used to realize 8 and 16 Mbytes and there are 16-Mbit devices to realize 32 and 64 Mbytes.

The microSPARC-II chip supports up to eight memory banks. Two of the eight memory banks are used on the SPARC CPU-5CE board. The table below shows the relationship between the board memory capacity and the memory banks used on the microSPARC-II.

Table 23: CPU-5CE Memory Banks

CPU-5CE Memory Capacity	Memory Bank 0 Used	Memory Bank 1 Used
8 Mbytes	X	
16 Mbytes	X	X
32 Mbytes	X	
64 Mbytes	X	X

3.3 SBus Participants

There are two SBus slots located on the component side of the board. SBus Slot #1 is located at connector P3 and SBus Slot #2 is located at connector P4. A diagram of the board is located in “Highlighted Diagram of the CPU-5CE” on page 51.

The microSPARC-II chip supports up to 5 SBus slots plus an additional "master-only" slot. The SBus controller is inside the microSPARC-II chip.

The following table shows the microSPARC-II physical address map including all of its SBus slots and their functions on the SPARC CPU-5CE.

3.3.1 Address Mapping for SBus Slots on the SPARC CPU-5CE

Table 24: Physical Memory Map of SBus on SPARC CPU-5CE

Addresses	SBus Slave Select Number	Function
3000 0000 -> 3FFF FFFF	SBus slave select 0	S4-VME Chip Registers
4000 0000 -> 4FFF FFFF	SBus slave select 1	SBus Socket # 1
5000 0000 -> 5FFF FFFF	SBus slave select 2	SBus Socket # 2
6000 0000 -> 6FFF FFFF	SBus slave select 3	VMEbus Interface
7000 0000 -> 77FF FFFF	SBus slave select 4	NCR89C105 (SLAVIO) chip
7800 0000 -> 7FFF FFFF	SBus slave select 4	NCR89C100 (MACIO) chip

3.4 NCR89C100 (MACIO)

The NCR89C100 is located on SBus Slave Select 4 at physical address $7800 0000. This chip drives the SCSI, Ethernet and Centronics parallel port.

The NCR89C100 SBus master 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.

3.4.1 Features of the NCR89C100 on the SPARC CPU-5CE

- 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 SBus Interface clocked with 21.25 MHz @ 85 MHz processor frequency
- Concurrently supports:
  - 10 MB/sec SCSI transfers
  - 3.4 MB/sec Parallel port transfers
  - 1.25 MB/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 *NCR SBus I/O Chipset Data Manual.*

3.4.2 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-5CE 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.

All signals of the SCSI interface are routed to the VME P2 connector. This connection is compatible to the CPU-3CE and the CPU-2CE. Please see the "VME P2 Connector Pinout" on page 39 where the SCSI signals on the VME P2 connector shown.

3.4.3 SCSI Termination

The termination of the SCSI interface can be enabled or disabled on-board via the switch SW2-6. If SW2-6 is ON, the termination is enabled. If SW2-6 is OFF, the termination is disabled.

Please see "Highlighted Diagram of the CPU-5CE" on page 51 for the location of the switches on the board.

CAUTION

When installing the SPARC CPU-5CE in a MICROFORCE chassis, please first disable the SCSI termination by switching SW2-6 to OFF.

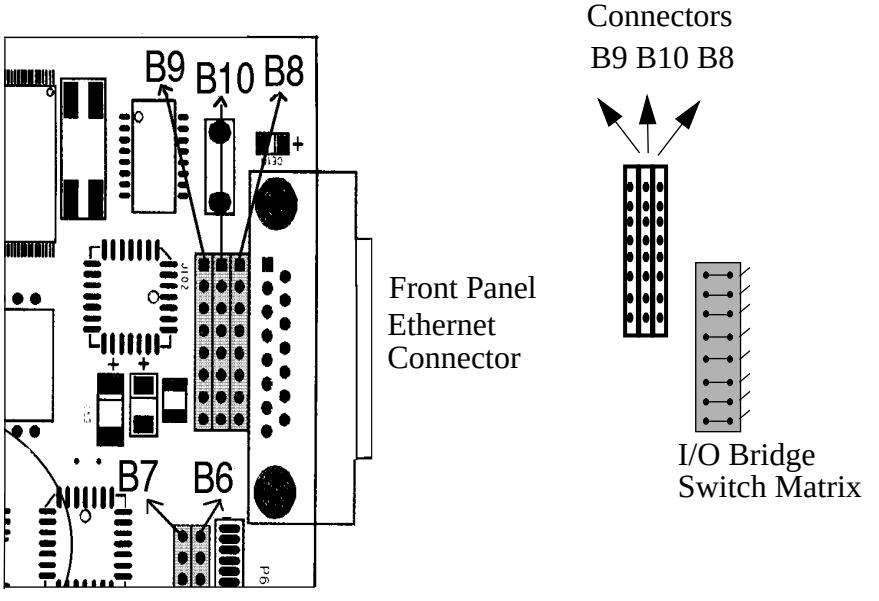
3.4.4 Ethernet

The NCR89C100 DMA controller enables the Ethernet interface to transfer data to and from the shared 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.

An 8-pin configuration switch matrix selects whether the Ethernet interface is available via the front panel or the VME P2 connector. The default configuration provides the Ethernet through the front panel connector.

It is possible to have the Ethernet interface accessible on the VME P2 connector by changing the default configuration as explained below. Take a moment to examine the diagram to understand how one achieves the desired configuration.

FIGURE 16. Ethernet Interface Availability



By default, the Ethernet Interface is available through the front panel connector with the I/O bridge plugged into connectors B9 and B10.

To configure the Ethernet interface to be accessible from the VMEbus P2 connector, the I/O bridge array must be plugged into connectors B8 and B10.

CAUTION

When the Ethernet is configured via P2, do not connect the Ethernet at the front panel.

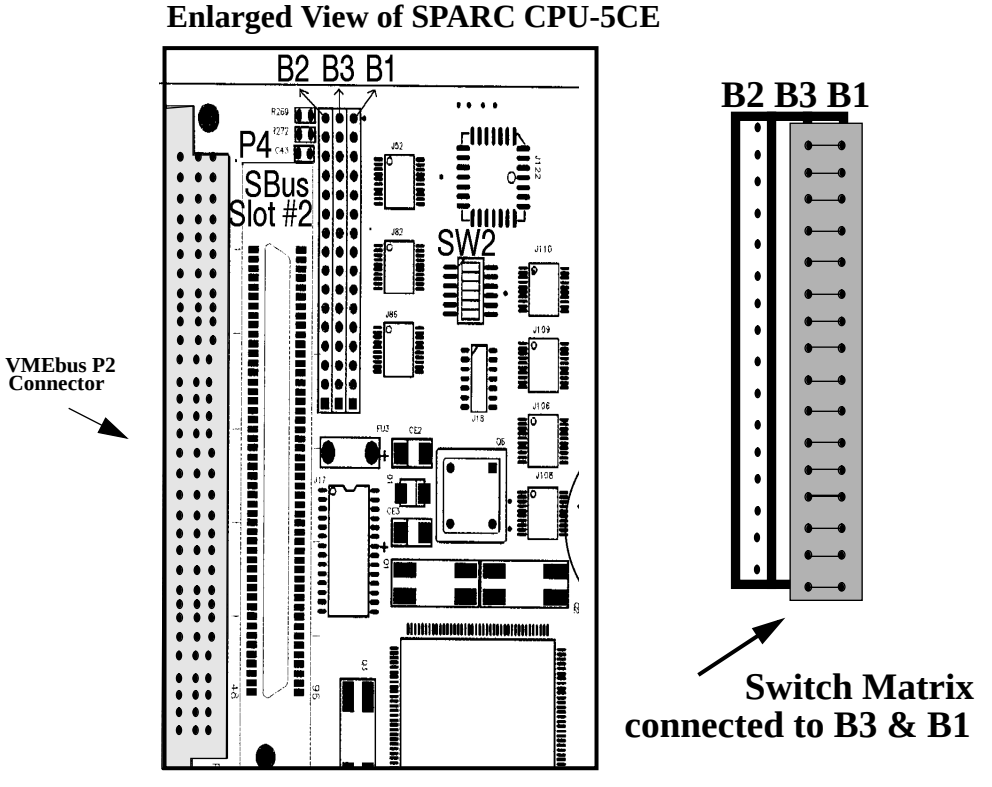
3.4.5 Parallel Port

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

By default, the parallel port is not available on the VMEbus P2 connector. It is the floppy disk interface which is available on the VMEbus P2 connector by default.

In order to configure the parallel port to be accessible from the VMEbus P2 connector, the switch matrix must be plugged into connectors B1 to B3. The following drawing shows this configuration.

FIGURE 17. Parallel Port Configuration Via P2 Connector



The configuration switch matrix plugs into connectors B1 and B3 and the parallel port is accessible through the VMEbus P2 connector.

3.5 NCR89C105 (SLAVIO)

The NCR89C105 SBus slave integrates most of the 8-bit system I/O functions including two dual channel 8530-compatible serial controllers, a high speed 8277AA-1 compatible floppy disk controller, counter/timers, interrupt controllers, and system reset logic. It also provides an SBus interface for several other byte-wide peripherals through an external expansion bus.

The primary serial controller is 8530-compatible and can be used as two general purpose serial ports.

The second serial controller is subset of the 8530 standard and is dedicated for the keyboard/ mouse connection.

The 8277AA-1 compatible floppy disk controller supports up to 1 Mbit/sec data transfer rate.

To reduce part count and system cost, a glueless interface to the SBus is provided. The slave I/O also includes an 8-bit expansion bus with control to support RTC/NVRAM, EPROM and generic 8-bit devices externally.

3.5.1 Features of the NCR89C105 on the SPARC CPU-5CE

- Dual channel serial ports (8530-compatible)
- Keyboard/ mouse port
- 82077AA-1 floppy disk controller
- 8-bit expansion bus for EPROM/TOD/NVRAM
- Glueless SBus interface clocked with 21.25 MHz @ 85 MHz processor frequency
- Interrupt controller
- System reset control
- Programmable 22-bit counters & timers
- Auxiliary I/O registers
- Packaged in 160 pin PQFP

For further information about the NCR89100, please refer to the *NCR SBus I/O Chipset Data Manual.*

3.5.2 Address Map of Local I/O Devices on SPARC CPU-5CE

The following table lists the physical addresses for all local I/O devices and the accesses permitted ((B)yte, (H)alf Word and (W)ord).

Table 25: NCR89C105 Chip Address Map

Physical Address	Device	Access
7000 0000 -> 70FF FFFF	Boot EPROM and User EPROM	B,H,W
7100 0000 -> 711F FFFF	Keyboard, Mouse, and Serial Ports	B
7100 0000	Mouse Control Port	
7100 0002	Mouse Data Port	
7100 0004	Keyboard Control Port	
7100 0006	Keyboard Data Port	
7110 0000	TTYB Control Port	
7110 0002	TTYB Data Port	
7110 0004	TTYA Control Port	
7110 0006	TTYA DATA Port	
7120 0000 -> 712F FFFF	RTC/NVRAM	B,H,W
7130 0000 -> 7137 FFFF	Boot EPROM and User EPROM Programming	B
7138 0000 -> 713F FFFF	Additional Registers	B
7140 0000 -> 714F FFFF	Floppy Controller	B
7140 0002	Digital Output Register (DOR)	
7140 0004	Main Status Register (MSR, Read Only)	
7140 0004	Datarate Select Register (DSR, Write Only)	
7140 0005	FIFO	
7140 0006	Reserved (Test mode select)	
7140 0007	Digital Input Register (DIR, Read Only)	
7140 0007	Configuration Control Register (CCR, Write Only)	
7150 0000 -> 7170 0000	Reserved	
7180 0000	89C105 Configuration Register	B
7190 0000 -> 719F FFFF	Auxiliary I/O Registers	B
7190 0000	Aux 1 Register (Miscellaneous System Functions)	
7191 0000	Aux 2 Register (Software Powerdown Control)	

3.5.3 Serial I/O Ports

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

Both of the two ports are available via the VMEbus P2 connector, each with four signals (RXD, TXD, RTS, CTS). Each of the two serial I/O ports are independent full-duplex ports.

The 8530 SCC block is functionally compatible with the standard NMOS 8530 and therefore provides two fully independent full-duplex ports.

The physical address map for the serial ports is shown in “NCR89C105 Chip Address Map” on page 62.

3.5.4 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 FH-002 hybrid module 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 J5. To change the configuration of serial port B, insert the respective hybrid in socket J6. For the position of the sockets on the board, please see “Highlighted Diagram of the CPU-5CE” on page 51.

Table 26: RS-232, RS-422 or RS-485 Configuration

Hybrid	Configuration	Socket for Serial Port A	Socket for Serial Port B
FH-002	RS-232	J5	J6
FH-003	RS-422	J5	J6
FH-005	RS-485	J5	J6

3.5.5 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 27: 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 28: Switch Settings for Ports A and B (RS-232)

Port A	Port B	Default	Function for RS-232
SW3-1	SW4-1	ON	TRXC is available on front panel connectors, pin 24
SW3-2	SW4-2	OFF	Off for RS-232
SW3-3	SW4-3	OFF	Off for RS-232
SW3-4	SW4-4	ON	RTS is available on front panel connectors, pin 4
SW3-5	SW4-5	ON	CTS is available on front panel connectors, pin 5
SW3-6	SW4-6	OFF	Off for RS-232

CAUTION: To avoid damaging the serial ports, please consider the following regarding Switch 3 and Switch 4. Do not set the switches (SW3-1 and SW3-2), or (SW3-3 and SW3-4), or (SW3-5 and SW3-6) to ON at the same time and do not set the switches (SW4-1 and SW4-2), or (SW4-3 and SW4-4), or (SW4-5 and SW4-6) to ON at the same time!
Please see the “Highlighted Diagram of the CPU-5CE” on page 51 for the location of the switches on the board.

3.5.6 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 29: 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 SW3 for port A and by switch SW4 for port B.

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

Table 30: Switch Settings for Ports A and B (RS-422)

Port A	Port B	Default	Function for RS-422
SW3-1	SW4-1	ON	ON for RS-422
SW3-2	SW4-2	OFF	OFF for RS-422
SW3-3	SW4-3	OFF	TRXC +/- on front panel connectors, pins 3 and 4 ON = available OFF =Not available
SW3-4	SW4-4	ON	RTS +/- on front panel connectors, pins 3 and 4 ON = available OFF =Not available
SW3-5	SW4-5	ON	CTS +/- on front panel connectors, pins 2 and 5 ON = available OFF =Not available
SW3-6	SW4-6	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 3 and Switch 4. Do not set the switches (SW3-1 and SW3-2), or (SW3-3 and SW3-4), or (SW3-5 and SW3-6) to ON at the same time and do not set the switches (SW4-1 and SW4-2), or (SW4-3 and SW4-4), or (SW4-5 and SW4-6) to ON at the same time!

Please see the “Highlighted Diagram of the CPU-5CE” on page 51 for the location of the switches on the board.

3.5.7 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 31: 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 NCR89C105 serial I/O signals DTR (REN) and RTS(TEN).

The following table shows the corresponding switch settings

Table 32: Switch Settings for Ports A and B (RS-485)

Port A	Port B	Configuration	Function for RS-485
SW3-1	SW4-1	OFF	OFF for RS-485
SW3-2	SW4-2	ON	RTS functions as TEN
SW3-3	SW4-3	OFF	No function for RS-485
SW3-4	SW4-4	OFF	No function for RS-485
SW3-5	SW4-5	OFF	No function for RS-485
SW3-6	SW4-6	OFF	No function for RS-485

Please see the “Highlighted Diagram of the CPU-5CE” on page 51 for the location of the switches on the board.

3.5.8 Transmission Line Termination

For the termination of the RS-422/RS-485 transmission lines, the SPARC CPU-5CE provides SIL sockets in which resistor networks or single resistors (2,54 mm grid) can be installed. When a serial port is not being used in a RS-422/RS-485 application, it should be terminated in a proper manner to avoid floating input signals.

The line termination is designed for networks in a 10-pin package.

The type of resistor network being used must be the bused version, which means that it has a common pin (pin #1) for all resistors and separate pins for the single resistors.

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

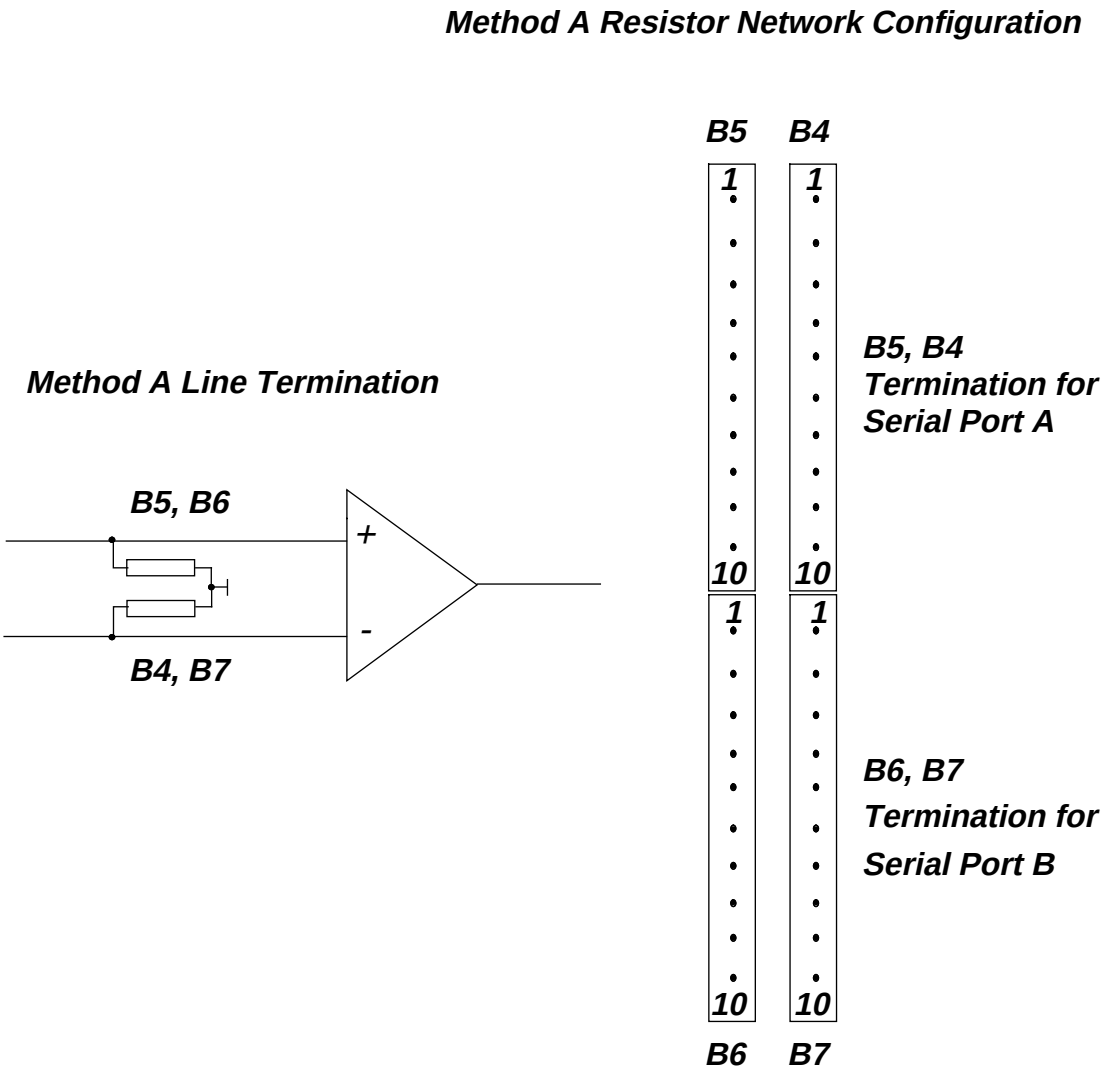
Table 33: Transmission Line Termination

Serial Port	Socket	Terminates
Serial Port A	B4	Inverted Signals
Serial Port A	B5	Non-inverted Signals
Serial Port B	B7	Inverted Signals
Serial Port B	B6	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 B4, B5, B6, and B7, please see “Highlighted Diagram of the CPU-5CE” on page 51.

FIGURE 18. Method A Line Termination & Resistor Network



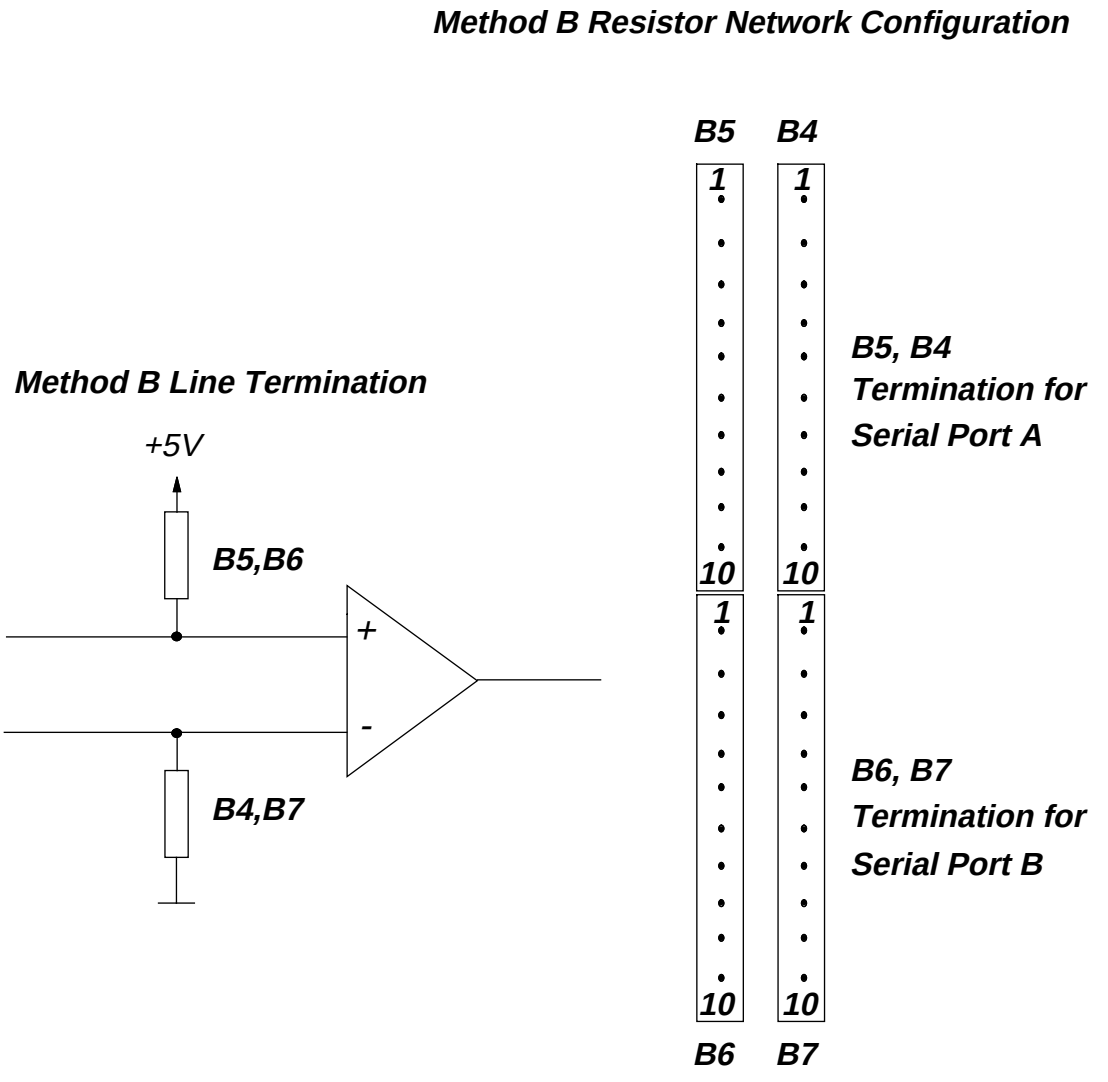
This **method A** requires resistor networks to be used for the transmission line termination.

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 B5 and B4, terminate the serial port A. Serial port B is terminated by using the socket locations B6 and B7.

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

FIGURE 19. Method B Line Termination & Resistor Network

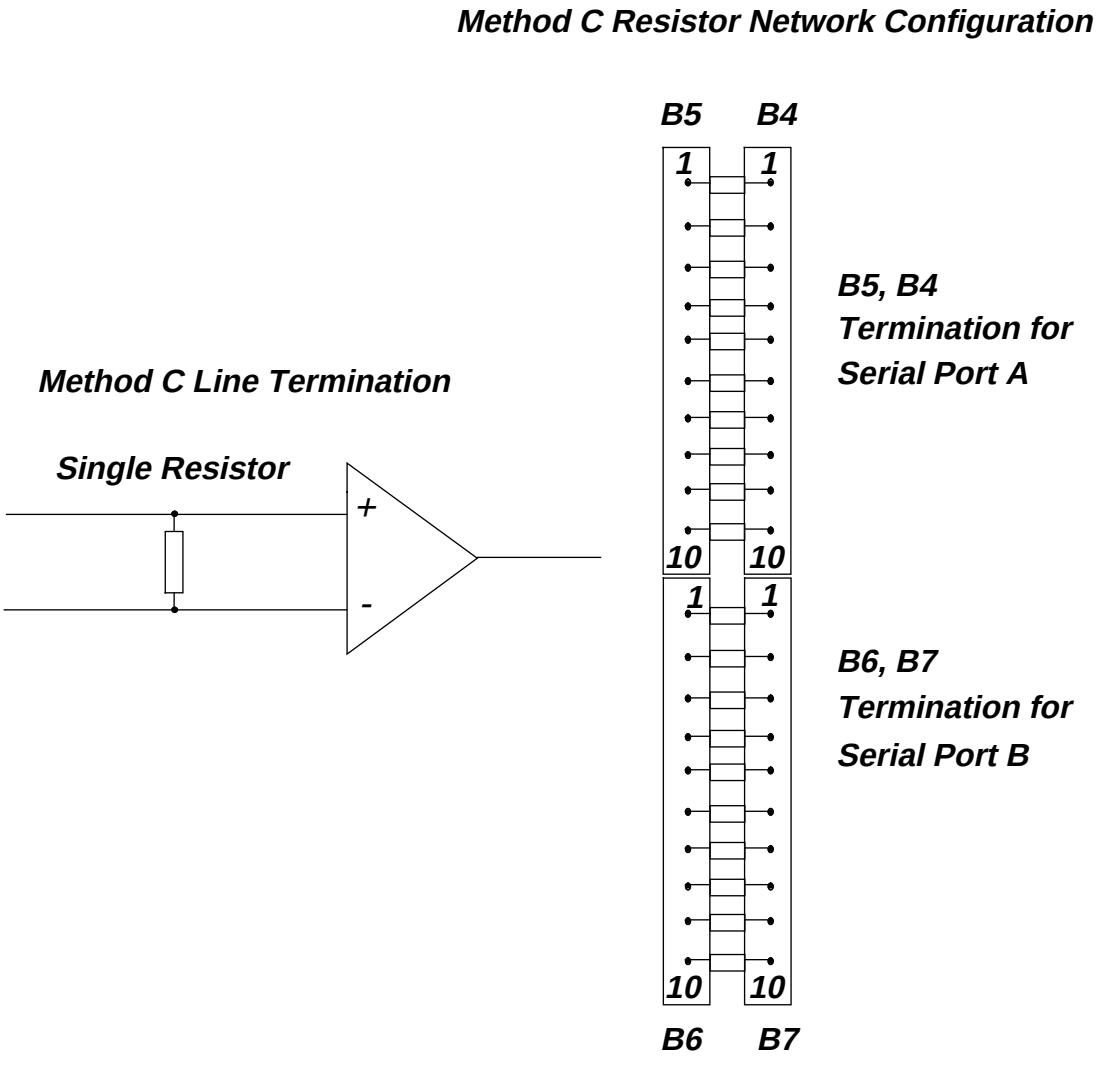


This **method B** requires resistor networks to be used for the transmission line termination. Using this method requires installing the networks at location B5 and B6 **reversed**. This means that the networks must be plugged into the socket so that their pin #1 is inserted into pin #10 of the socket.

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 B5 and B4. Serial port B is also terminated by using the socket locations B6 and B7. This allows terminating the serial ports A and B with different termination resistors.

FIGURE 20. Method C Line Termination & Resistor Network



This **method C** uses single 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 34: Signal Resistor Termination (Method C) for RS-422

Signal Transmission Line	Serial Port	Resistor inserted between B4-B5	Signal Transmission Line	Serial Port	Resistor inserted between B6-B7
RxD+/-	A	2-2	RxD+/-	B	2-2
TxD+/-	A	3-3	TxD+/-	B	3-3
RTS+/-* TRXC+/-*	A	4-4	RTS+/-* TRXC+/-*	B	4-4
CTS+/-* RTXC+/-*	A	5-5	CTS+/-* RTXC+/-*	B	5-5

Note: The signals marked with * are alternatively available on the network resistor sockets. It depends 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 35: Signal Resistor Termination (Method C) for RS-485

Signal Transmission Line	Serial Port	Resistor inserted between B4-B5	Signal Transmission Line	Serial Port	Resistor inserted between B6-B7
RXTX+/-	A	2-2	RXTX+/-	B	2-2

3.5.9 Keyboard and Mouse Port

The keyboard and mouse port is available on the front panel via an 8-pin mini DIN connector.

The serial port controller used for the keyboard and mouse port is compatible with the NMOS 8530 controller.

The pinout of the keyboard and mouse port are described in Section 2, Installation.

The physical address for the keyboard and mouse port is shown in “NCR89C105 Chip Address Map” on page 62.

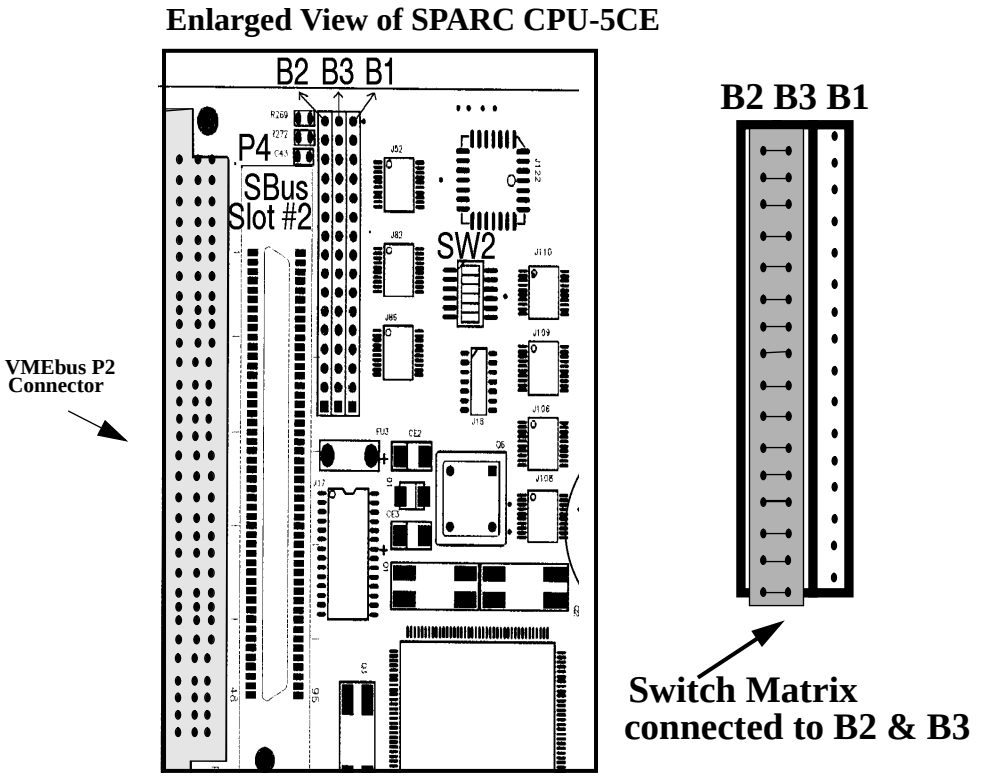
3.5.10 Floppy Disk Interface

The floppy disk interface is available on the P2 connector by default, with the configuration switch matrix plugged into B2 and B3.

The floppy disk interface is 82077AA-1 compatible. It is able to transfer data rates of 250, 300, 500 Kbytes/sec, and 1 Mbyte/sec.

The floppy disk controller block is functionally compatible with the Intel 82077AA-1. It integrates drivers, receivers, data separator, and a 16-byte bidirectional FIFO. The floppy disk controller supports all standard disk formats (typically 720 K and 1.44 M floppies). It is also compatible with the 2.88 MB floppy format.

FIGURE 21. Floppy Disk Interface Via P2 Connector



When switch matrix plugs into connectors B2 and B3, the floppy interface is accessible through the VMEbus P2 connector.

3.5.11 8-Bit Local I/O Devices

The following local I/O devices are interfaced via the NCR89C105

Table 36: 8-Bit Local I/O Devices

Function	IRQ	Physical Base Address
Boot Flash EPROM Device # 1 256 Kbyte (default)	No	$7000 0000->$7003 FFFF
Boot Flash EPROM Device # 2 256 Kbyte (default)	No	$7004 0000 ->$7007 FFFF
User Flash EPROM 1 Mbyte Device # 1	No	$7010 0000 ->$701F FFFF
User Flash EPROM 1 Mbyte Device # 2	No	$7020 0000 ->$702F FFFF
RTC/NVRAM	No	$7120 0000 ->$712F FFFF
Flash EPROM Programming Area	No	$7130 0000 ->$7137 FFFF
Additional Registers	No	$7138 0000 ->$713F FFFF

3.5.12 Boot EPROM

The Boot EPROM consists of two 2-Mbit or 4-Mbit flash memory devices. In the default configuration, there are two 2-Mbit devices installed. The 4-Mbit devices are an additional assembly option.

The Boot EPROM devices can be reprogrammed on-board and can also be write protected via hardware switch SW2-3.

When SW2-3 is ON, write accesses are possible. The devices are write protected when SW2-3 is OFF.

The Boot EPROM devices are installed in sockets at location J124 and J125. This permits programming them in a standard programmer. This may be necessary if the power fails during reprogramming. In this case, the contents of the Boot EPROM would be lost and the board would not be able to boot.

Table 37: Boot EPROM Capacity

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

The on-board programming of the Boot EPROM devices requires setting some bits in the *vme_a32map* register and *gen_purpose2* register. This is described in the chapter “Programming the On-board Flash Memories” on page 77.

For the position of the Boot EPROM on the board, please see “Highlighted Diagram of the CPU-5CE” on page 51.

3.5.13 User Flash EPROM

The User Flash EPROM area consists of a maximum of two 8-Mbit flash memory devices, providing a capacity of 2 Mbytes. The capacity of user flash EPROMS is outlined in the product nomenclature, which can be seen in the table “Product Nomenclature” on page 6.
This area can be used to store ROMable operating systems as well as application specific code.

Table 38: User Flash EPROM Capacity

Devices	Count	Capacity
1M*8	0	0 Mbyte
1M*8	1	1 Mbyte
1M*8	2	2 Mbyte

The User Flash EPROM devices can be reprogrammed on-board and can also be write protected via hardware switch SW2-4. When SW2-4 is ON, write accesses are possible. When SW2-4 is OFF, the devices are write protected.

The on-board programming of the User Flash EPROM devices requires setting some bits in the *vme_a32map* register and the *gen_purpose2* register. This is described in the chapter “Programming the On-board Flash Memories” on page 77.

For the position of the user flash EPROM on the board, please see “Highlighted Diagram of the CPU-5CE” on page 51.

3.5.14 Programming the On-board Flash Memories

Both areas of flash memories, the Boot EPROM area and the User EPROM area, can be reprogrammed on-board.

To enable the programming of the flash memory devices, the +12V programming voltage must be switched ON. This is done by setting bit VPPCTL in the *vme_a32map* register.

Physical Address	Register Name	Read/Write	Access
7138 0004	vme_a32map	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FL_PAG	VPPCTL	WTENA	WNMIP	VME _A[31..28]			

Table 39: +12V Programming Voltage Control Bit

Setting	Function
VPPCTL = 0	No flash EPROM programming possible
VPPCTL = 1	Flash EPROM programming possible

Initialization: VPPCTL is cleared on reset. This inhibits the programming of the flash EPROMs.

3.5.15 Programming Control Bits for Flash Memory Devices

The address range in which the flash EPROMs can be programmed is located in a 512 Kbyte page (programming window) of the Generic Port area of the NCR89C105 (SLAVIO). The physical address range is $7130 0000 .. $7137 FFFF.

On the CPU-5CE there is a maximum of 3 Mbyte flash memory available (1 Mbyte Boot EPROM and 2 Mbyte User EPROM). To program these areas, they have to be divided into 2 or 4 512 Kbyte pages, which will fit into the programming window. The flash memories can only be programmed in the programming window.

In order to decide which area is to be mapped to the programming window, the following three bits are used to control this. The relevant three bits are: FL_PAG in the *vme_a32map* register, BT_US and US_DEV in the *gen_purpose2* register.

Physical Address	Register Name	Read/Write	Access
$7138 0007	gen_purpose2	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WNMIR	ACSTAT	ACNMIR	ACNMIP	DVMA_ ENA	IRQ15_ ENA	US_DEV	BT_US

The next page contains a list of the flash memory programming control bits.

Table 40: Flash Memory Programming Control Bits

BT_US	FL_PAG	US_DEV	Programs
1	0	don’t care	Boot EPROM Dev#1 with 256 KB
1	1	don’t care	Boot EPROM Dev#2 with 256 KB
1	0	don’t care	Boot EPROM Dev#1 with 512 KB
1	1	don’t care	Boot EPROM Dev#2 with 512 KB
0	0	0	User EPROM first 512 KB of Dev#1
0	1	0	User EPROM second 512 KB of Dev#1
0	0	1	User EPROM first 512 KB of Dev#2
0	1	1	User EPROM second 512 KB of Dev#2

Initialization: BT_US, FL_PAG and US_DEV are all cleared to 0s after reset.

For the detailed description of all additional register bits on the CPU-5CE, please also read the chapter “Additional Registers” on page 107.

3.5.16 RTC/NVRAM

The MK48T08 combines an 8 K x 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 pin and functionally equivalent to any Jedec standard 8 K x 8 SRAM

For a detailed description of the RTC/NVRAM, please see the respective Data Sheet in Section 4 of this manual.

3.6 VMEbus Interface

The CPU-5CE utilizes the Sun S4-VME chip to provide a complete 32-bit VMEbus interface. Supported functions include master and slave data transfer capabilities, VMEbus interrupt handling and arbitration functions. Additional VMEbus utility functions and a special loop-back cycle for stand-alone testing of the interface are provided.

Features of the SPARC CPU-5CE VMEbus Interface

- A32/A24/A16 Master and A24/A32 Slave DVMA device as SBus Master device
- Full 4 Gigabyte VME addressing with mapping register
- Enhanced Slave Mode with programmable slave base address and slave window size
- System Controller Functions:
    - Single-level or round-robin arbitration with bus arbiter timer
    - IACK Daisy Chain Driver
    - SYSCLK Clock driver
    - SYSRESET driver
- VME Interrupt Handler
- Programmable Mailbox Interrupt Level
- VMEbus Bus Timer

3.6.1 Master Interface

The VME master interface allows A32, A24 and A16 mode addressing with D8(even/odd), D16 and D32 mode data transfers. A full 4 Gbyte address range is available by mapping a 256 Mbyte SBus slot window. Unaligned transfers and block mode transfers are not supported.

3.6.1.1 VMEbus Master Address Implementation

The VMEbus master interface is physically located in the SBus Slave Select 3 address range.

Table 41: VMEbus Master Interface Physical Address Map

Physical Address	Name	Function
$6000 0000.. $6FFF FFFF	SBus Slave Select 3	VMEbus Master Interface

The 4 bit base address for the 256 Mbyte boundary is set in the *vme_a32map* register. The 4 upper VMEbus address lines A31..A28 can be programmed in that register.

Physical Address	Register Name	Read/Write	Access
$7138 0004	vme_a32map	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FL_PAG	VPPCTL	WTENA	WNMIP	VME _A[31..28]			

Initialization: At reset VME_A[31..28] are cleared to 0s.

CAUTION

The *vme_a32map* register is not identical with the *A32_map* register of the S4-VME chip. For the programming of the VMEbus master address, the *A32_map* register of the S4-VME chip is not used.

The VMEbus master interface allows the following three address ranges:

Table 42: VMEbus Address Ranges

Mode	Address Lines Used	Short Form	Physical Address Offset
Extended Addressing	A01..A31	A32	$0000 0000
Standard Addressing	A01..A24	A24	$FF00 0000
Short Addressing	A01..A15	A16	$FFFF 0000

CAUTION

In order to access the A16 or the A24 VMEbus address range, the VME_A[28] bit of the above described vme_a32map register must be set.

All supported Address Modifier combinations are shown in the next table.

Table 43: Supported Address Modifier Codes

Address Modifier	Address Mode	Transfer Mode
$2D	A16	Short Supervisory Access
$29	A16	Short Non-Privileged Access
$3D	A24	Standard Supervisory Data Access
$39	A24	Standard Non-Privileged Data Access
$0D	A32	Extended Supervisory Data Access
$09	A32	Extended Non-Privileged Data Access

To choose the access mode “Supervisory” or “Non-Privileged”, a software controlled bit, called SUPV (SUPerVisory mode) in the *vme_ctl* register, directly reflects the Address Modifier Bit 2.

Physical Address	Register Name	Read/Write	Access
$7138 0003	vme_ctl	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
reserved	SMODE	SUPV	SAMODE	reserved	WIN2	WIN1	WIN0

Table 44: Address Modifier Supervisory Bit

Setting	Function
SUPV = 0	Non-Privileged Address Mode
SUPV = 1	Supervisory Address Mode

Initialization: At reset the SUPV is cleared to 0.

3.6.1.2 Data Bus Sizes

The SPARC CPU-5CE VMEbus master interface supports the data size modes D8(even/odd), D16 and D32. Block mode transfers and unaligned transfers are not supported. The following table illustrates the supported VMEbus transfer cycles.

Table 45: VMEbus Master Interface Transfer Cycles

Transfer Type	D31..D24	D23..D16	D15..D08	D07..D00
8-bit				X
8-bit			X	
16-bit			X	X
32-bit	X	X	X	X

NOTE: The “X” shows the active byte portion.

Please also refer to Section 5 “OpenBoot Enhancements” and Section 6 “VMEbus Driver” for VMEbus master interface accessing methods.

3.6.2 Slave Interface

Access to the SPARC CPU-5CE on-board DRAM is allowed to a 1 Mbyte page within a 16 Mbyte area in the Slave Default Mode. In the Slave Enhanced Mode the accessible window is not restricted to the 1 Mbyte page and the base address is variable. Eight different window sizes are possible, from 1 Mbyte to 64 Mbyte. In both modes the VMEbus address space is always mapped to the upper virtual address space. There is a software controlled bit to select one of these modes. This bit is called SMODE (Slave MODE) in the *vme_ctl* register.

Physical Address	Register Name	Read/Write	Access
$7138 0003	vme_ctl	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
reserved	SMODE	SUPV	SAMODE	reserved	WIN2	WIN1	WIN0

Table 46: Slave Mode Bit

Setting	Function
SMODE = 0	VMEbus Slave Enhanced Mode
SMODE = 1	VMEbus Slave Default Mode

Initialization: At reset the SMODE bit is cleared to 0.

The VMEbus slave interface only works if it is enabled via the DVMA_ENA (DVMA ENAble) bit in the *gen_purpose2* register.

Physical Address	Register Name	Read/Write	Access
$7138 0007	gen_purpose2	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WNMIR	ACSTAT	ACNMIR	ACNMIP	DVMA_ ENA	IRQ15_ ENA	US_DEV	BT_US

Table 47: DVMA Enable Bit

Setting	Function
DVMA_ENA= 0	VMEbus Slave Mode disabled
DVMA_ENA= 1	VMEbus Slave Mode enabled

Initialization: At reset the DVMA_ENA bit is cleared to 0.

NOTE: This bit is not identical with the DVMA bit in the *slave-map* register of the S4-VME chip. The function is the same, however the DVMA bit of the S4-VME chip enables the slave mode after reset. The DVMA_ENA bit in the *gen_purpose2* register overwrites the DVMA S4-VME bit. This means that when DVMA_ENA = 0 the slave mode is always disabled, independent of the S4-VME bit.
Addressing is recognized for both 32-bit extended and 24-bit standard accesses, with 16-bit accesses reserved for the mailbox interrupt.Unaligned slave accesses and block mode transfers are not supported.

3.6.2.1 VMEbus Slave Address Modes

The CPU-5CE VMEbus slave interface can handle A32, A24, and A16 mode. The A16 address space is only acknowledged for the mailbox interrupt functions, which are described below in “VMEbus Interrupt Handler and MailBox Interrupt Function” on page 89. To distinguish A32 and A24 addressing mode, a software controlled bit called SAMODE (Slave Address MODE), is provided in the *vme_ctl* register.

Physical Address	Register Name	Read/Write	Access
$7138 0003	vme_ctl	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
reserved	SMODE	SUPV	SAMODE	reserved	WIN2	WIN1	WIN0

Table 48: Slave Address Mode Bit

Setting	Function
SAMODE = 0	VME Extended A32 Mode
SAMODE = 1	VME Standard A24 Mode

Initialization: At reset the SAMODE bit is cleared to 0.

3.6.2.2 VMEbus Default Slave Mode

The default slave mode allows access to a 1 Mbyte page within a 16 Mbyte area of the SPARC CPU-5CE on-board memory. This 1 Mbyte window is always mapped to the upper virtual address space, resulting in a DVMA address space of $FFF0 0000 .. $FFFF FFFF.

The 1 Mbyte window within the 16 Mbyte area is selected in the *slave-map* register of the S4-VME chip. Please refer to “Register Accesses to the S4-VME Chip” on page 91, and to the Data Sheet of the S4-VME chip in Section 4, for a detailed description of the S4-VME registers.

3.6.2.3 VMEbus Enhanced Slave Mode

In the enhanced slave mode the DVMA window is not restricted to the 1 Mbyte size and to the 16 Mbyte area. There are eight different windows supported, which are from 1 Mbyte to 64 Mbyte. The window size can be programmed in the *vme_ctl* register.

Physical Address	Register Name	Read/Write	Access
$7138 0003	vme_ctl	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
reserved	SMODE	SUPV	SAMODE	reserved	WIN2	WIN1	WIN0

The table on the next page shows the setting for WIN[2..0].

Table 49: Window Size Bits

WIN2	WIN1	WIN0	Window Size	Virtual Base Address
0	0	0	1 Mbyte	$FFF0 0000
0	0	1	2 Mbyte	$FFE0 0000
0	1	0	4 Mbyte	$FFC0 0000
0	1	1	8 Mbyte	$FF80 0000
1	0	0	16 Mbyte	$FF00 0000
1	0	1	32 Mbyte	$FE00 0000
1	1	0	64 Mbyte	$FC00 0000
1	1	1	No function	

Initialization: At reset, WIN[2..0] are set to 0s.

Depending on the programmed window size, the DVMA window is mapped to the upper virtual address space. Additionally, the upper table shows the virtual base addresses for the different window sizes.

Note that the virtual base address gets an offset if the programmed VMEbus slave base address does not have the selected window size boundary.

For example: The programmed window size is 64 Mbyte. You choose a VMEbus slave base address of $8010 0000, which is not in a 64 Mbyte boundary. This would lead to a virtual base address of $FC00 0000 + $10 0000 = $FC10 0000.

The VMEbus slave base address can be programmed anywhere in the 4 Gbyte VMEbus address space. The VMEbus slave base address is that address where the VMEbus master can access on-board memory of the CPU-5CE. For the programming of the VMEbus slave base address, please refer to“Additional Registers” on page 107.

3.6.3 VMEbus Interrupt Handler and MailBox Interrupt Function

A VMEbus interrupt handler supports all interrupt levels. These are enabled via bit 4 of the gen_purpose2 register and also via the interrupt enable register within the S4-VME chip. Bit 4 in the gen_purpose2 register always overwrites the interrupt enable register if the interrupts should be disabled. Writing a zero to bit 4 of the gen_purpose2 register disables the VMEbus interrupts, regardless of the contents in the respective S4-VME register. This is the value after reset. To enable the VMEbus interrupts, both registers must be set to enable. For a detailed description of the gen_purpose 2 register, please see “Additional Registers” on page 107.

A mailbox interrupt function allows other VMEbus participants to interrupt the CPU-5CE. This mailbox interrupt can be generated with accesses to the specific A16 address space. The *mailbox control* register and the *mailbox interrupt level* register in the S4-VME chip controls this interrupt feature.

The mailbox interrupt can be set in the *mailbox interrupt level* register to generate either any level of SBus interrupts or an interrupt at the MB_IRQ pin of the S4-VME chip. If the mailbox interrupt is gated to the MB_IRQ pin of the S4-VME chip, a **board reset** would result.

Please refer to “Register Accesses to the S4-VME Chip” on page 91, and to the Data Sheet of the S4-VME chip in Section 4, for the detailed description of the interrupt handler and the mailbox interrupt function.

3.6.4 VMEbus System Controller

The SPARC CPU-5CE VMEbus interface is configured by default as a slot-1 device which functions as VMEbus system controller.

The VMEbus system controller feature is selected by the on-board SW5-1. When SW5-1 is ON, the SPARC CPU-5CE is a VMEbus slot-1 device, when SW5-1 is OFF, the SPARC CPU-5CE is not system controller. Please see the “Highlighted Diagram of the CPU-5CE” on page 51 for the position of the switches on the board.

Features of the VMEbus System Controller

- Single-level or round-robin arbitration with bus arbiter timer
- IACK Daisy Chain Driver
- SYSCLK Clock Driver
- SYSRESET Driver

Please refer to the Data Sheet of the S4-VME chip in the Section 4 for additional information about the S4-VME system controller.

CAUTION

Before installing a SPARC CPU-5CE in a MINIFORCE chassis, please first disable the VMEbus System Controller function by setting switch SW5-1 to OFF.

3.6.5 Register Accesses to the S4-VME Chip

All registers of the S4-VME chip are located in the SBus Slave Select 0 address range starting with the physical base address shown in the next table.

Table 50: S4-VME Chip Physical Address Map

Physical Base Address	Name	Function
$3FE0 0000	SBus Slave Selcect 0	S4-VME Chip Registers

For the offsets of all S4-VME chip registers, please refer to the S4-VME chip Data Sheet in Section 4.

Note: The physical address described in the S4-VME chip is the address seen by the chip itself, but the user sees the chip internal registers in the above described SBus Slave Select 0 range.

3.6.6 VMEbus Utility Functions

The CPU-5CE handles the VMEbus signals SYSFAIL and ACFAIL. SYSFAIL and ACFAIL are monitored and their high-to-low edges are capable of generating a level 15 non-maskable interrupt. A non-maskable interrupt is only generated if the IRQ15_ENA bit in the *gen_purpose2* register is set to enable.

The SYSFAIL signal can also be driven by the SPARC CPU-5CE.

The pending non-maskable interrupt generated by the VMEbus SYSFAIL is readable in the *gen_purpose1* register. This bit is called SYSNMIP (SYSfail Non-Maskable Interrupt Pending).

The *gen_purpose1* register and relevant bits are shown on the next page.

Physical Address	Register Name	Read/Write	Access
$7138 0005	gen_purpose1	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SYSSTAT	SYSVME	SYSNMIP	ABNMIP	ROT[3..0]			

Table 51: SYSFAIL Non-Maskable Interrupt Pending Bit

Setting	Function
SYSNMIP= 0	No SYSFAIL NMI Pending
SYSNMIP= 1	SYSFAIL NMI Pending

NOTE: SYSNMIP is a read only bit.

Initialization: At reset, SYSNMIP is cleared to 0.

The appropriate interrupt handler has to reset the pending non-maskable interrupt. This can be done with a write access to the *vme_slavebase*2 register. To reset the SYSNMIP bit, write any data to that register. In addition, the Power Fail Detect bit in the Aux 2 Register of the NCR89C105 must be set to clear the interrupt. Please refer to the *NCR SBus I/O Chipset Data Manual* for the Aux 2 Register.

In order to drive the VMEbus SYSFAIL, bit SYSVME in the *gen_purpose1* register has to be manipulated.

Table 52: SYSFAIL to VMEbus Bit

Setting	Function
SYSVME= 0	Active SYSFAIL to VMEbus
SYSVME= 1	No SYSFAIL to VMEbus

Initialization: At reset, SYSVME is cleared to 0.

The actual status of the VMEbus SYSFAIL signal can be read in the *gen_purpose1* register in bit SYSSTAT.

Table 53: SYSFAIL Status Bit

Setting	Function
SYSSTAT= 0	SYSFAIL active on VMEbus
SYSSTAT= 1	SYSFAIL inactive on VMEbus

NOTE: SYSSTAT is a read only bit.

The pending non-maskable interrupt generated by the VMEbus ACFAIL is readable in the *gen_purpose2* register. This bit is called ACNMIP (ACFAIL Non-Maskable Interrupt Pending).

Physical Address	Register Name	Read/Write	Access
$7138 0007	gen_purpose2	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WNMIR	ACSTAT	ACNMIR	ACNMIP	DVMA_ENA	IRQ15_ENA	US_DEV	BT_US

The relevant bits are described on the next page.

Table 54: ACFAIL Non-Maskable Interrupt Pending Bit

Setting	Function
ACNMIP= 0	No ACFAIL NMI Pending
ACNMIP= 1	ACFAIL NMI Pending

NOTE: ACNMIP is a read only bit for the ACFAIL interrupt function. Writing that bit enables/disables the VMEbus interrupts. This is described in the chapter “VMEbus Interrupt Handler and MailBox Interrupt Function” on page 89.

The appropriate interrupt handler has to reset the pending non-maskable interrupt. This can be done with writing a one to the ACNMIR (ACFAIL Non-Maskable Interrupt Reset) bit in the *gen_purpose2* register. This bit is a write only.
In addition, the Power Fail Detect bit in the Aux 2 Register of the NCR89C105 must be set to clear the interrupt. Please refer to the *NCR SBus I/O Chipset Data Manual* for the Aux 2 Register.
The actual status of the VMEbus ACFAIL signal can be read in the *gen_purpose*2 register in bit ACSTAT.

Table 55: ACFAIL Status Bit

Setting	Function
ACSTAT= 0	ACFAIL active on VMEbus
ACSTAT= 1	ACFAIL inactive on VMEbus

3.6.7 VMEbus SYSRESET Enable/Disable

When the SPARC CPU-5CE is a VMEbus slot-1 device, it generates the SYSRESET signal to VMEbus. This can be disabled by setting the switch SW9-1 to OFF.

An external SYSRESET generates an onboard RESET in the default switch setting, i.e., SW9-2 is ON. When SW9-2 is OFF, the external SYSRESET does not generate an onboard RESET.

Please see the “Highlighted Diagram of the CPU-5CE” on page 51 to see the position of the switches on the board.

3.6.8 VMEbus Bus Timer

The VMEbus bus timer monitors all bus activities on the VMEbus and generates a BERR signal if the current cycle is not terminated in time. The bus timer works regardless of whether or not the CPU-5CE is a VMEbus slot-1 device. The table below shows the timeout values of the bus timer.

Table 56: VMEbus Bus Timer

Bus Timer Generates	Min.	Typical	Max.
BERR	1.0 ms	1.5 ms	2.0 ms

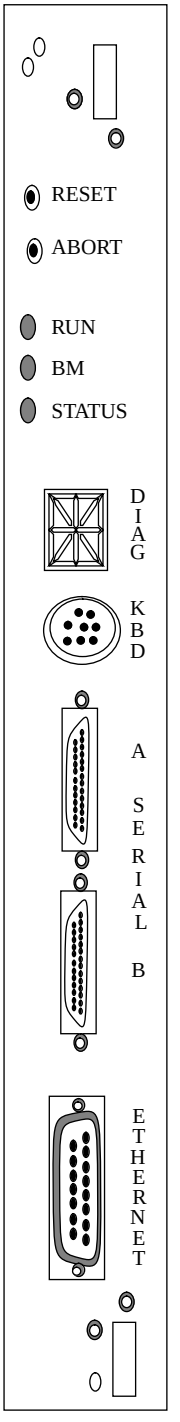
3.7 Front Panel

The figure on the next page shows a diagram of the front panel. The connectors which are listed below are described in Section 2, Installation. The Reset and Abort functions are described on the following pages.

Table 57: Features of the Front Panel

Device	Function	Name
Switch	Reset	RESET
Switch	Abort	ABORT
LED	RUN/RESET	RUN
LED	VMEbus Bus Master	BM
LED	Status	STATUS
HEX. Display	Diagnostic	DIAG
MiniDin Connector	Keyboard/Mouse	KBD
Serial Connector	Serial Interface	SERIAL A
Serial Connector	Serial Interface	SERIAL B
D-Sub Connector	Ethernet Interface	ETHERNET

FIGURE 22. Front Panel



3.7.1 RESET and ABORT Keys

The front panel on the SPARC CPU-5CE has two mechanical switches which directly influence the system. Please see “Highlighted Diagram of the CPU-5CE” on page 51 for the position of the switches.

3.7.1.1 The RESET Key

The **RESET** key enables the user to reset the whole board. If the board is VMEbus system controller (slot-1 device), the SYSRESET signal of the VMEbus also becomes active with the RESET key. This resets the complete VMEbus system. With on-board switch SW8-1, it is possible to deactivate the RESET key. When SW8-1 is ON, the RESET key works and when SW8-1 is OFF, toggling the RESET key has no effect.

3.7.1.2 The ABORT Key

The **ABORT** key on the front panel can be used to generate a non-maskable interrupt (level 15). The ABORT key function is controlled by switch SW8-2. When SW8-2 is ON, the key works and when SW8-2 is OFF, toggling the ABORT key has no effect. If the ABORT key produces a non-maskable interrupt, the pending signal can be read in the *gen_purpose1* register.

The tables on the next page show the relevant bits in the the *gen_purpose1* register.

Physical Address	Register Name	Read/Write	Access
$7138 0005	gen_purpose1	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SYSSTAT	SYSVME	SYSNMIP	ABNMIP	ROT[3..0]			

NOTE: ABNMIP is a read only bit.

Initialization: At reset, ABNMIP is cleared to 0.

The appropriate interrupt handler has to reset the pending non-maskable interrupt. This is done with a write access to the *vme_slavebase*1 register. Please refer to the chapter "Additional Registers" on page 107.

In addition, the Power Fail Detect bit in the Aux 2 Register of the NCR89C105 must be set to clear the interrupt. Please refer to the *NCR SBus I/O Chipset Data Manual* for the Aux 2 Register.

To reset the ABNMIP bit, you can write any data to that register.

Table 58: Abort Non-Maskable Interrupt Pending Bit

Setting	Function
ABNMIP = 0	No Abort NMI Pending
ABNMIP = 1	Abort NMI Pending

3.7.2 Front Panel Status LEDs

There are three single LEDs on the front panel.

- The RUN/RESET LED
- The VME BM LED (Bus Master)
- A STATUS LED

The RUN/RESET LED is either red or green. This LED is red when any reset signal on the board is active. In all other cases, this LED is green.

The BM LED reflects all VMEbus master activities on the CPU-5CE. When the board accesses the VMEbus, the BM LED lights up green.

An additional STATUS LED is a freely programmable LED, which is controlled by accessing a register in the NCR89C105 (SLAVIO). Bit 0 (LED) of the *Aux 1* register controls the STATUS LED with following settings.

Please refer to the *NCR SBus I/O Chipset Data Manual* for the Aux 1 Register.

Physical Address	Register Name	Read/Write	Access
$7190 0000	Aux 1 Register	r/w	8 bit

Setting	Function
LED = 0	STATUS LED = OFF
LED = 1	STATUS LED = ON

3.7.3 Diagnostic LED (Hex Display)

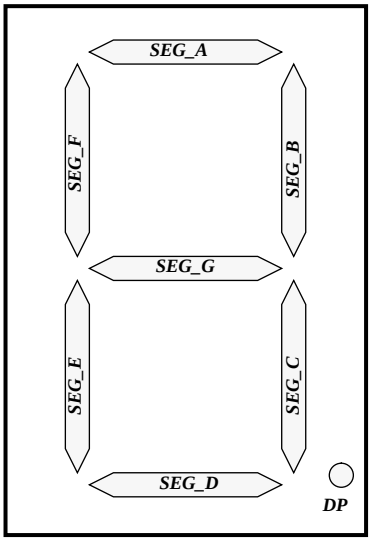
A freely programmable LED display on the front panel provides diagnostic features. It can be accessed via the *led_display* register.

Physical Address	Register Name	Read/Write	Access
$7138 0006	led_display	w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DP	SEG_G	SEG_F	SEG_E	SEG_D	SEG_C	SEG_B	SEG_A

The following figure shows the hex display with the segments named in accordance to their bits in the *led_display* register. To switch a specific segment on, the corresponding bit must be set to one.

FIGURE 23. Segments of the Hex Display



3.8 Additional features

3.8.1 Hardware Watchdog Timer

In addition to the two programmable 22-bit counters/timers in the NCR89C105 (SLAVIO), there is a hardware watchdog timer for system control functions. It is used to inhibit system deadlock.

In such system deadlock cases, the timer (if enabled) will first generate a non-maskable interrupt (WNMIP) to give software a chance to react by retriggering the timer.

There will be a board reset if the timer is not retriggered during the times indicated in the table below (RESET times).

Signal	min. time	typ. time	max. time
WNMIP	435 ms	670 ms	905 ms
RESET	2.00 s	3.08 s	4.16 s

The hardware watchdog timer is enabled with the WTENA bit in the *vme_a32map* register. This bit is also used to retrigger the timer. To start the timer again, you have to write a one to that bit.

Physical Address	Register Name	Read/Write	Access
$7138 0004	vme_a32map	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FL_PAG	VPPCTL	WTENA	WNMIP	VME _A[31..28]			

Setting	Function
WTENA= 0	Watchdog disabled
WTENA= 1	Watchdog enabled (to retrigger write a one!)

NOTE: Once the timer is enabled, it can't be disabled anymore.

Initialization: At reset, the WTENA bit is cleared which disables the watchdog timer.

Description:

The WNMIP bit in the *vme_a32map* register is active when there is a watchdog NMI pending, that is, when the first timeout of the watchdog timer occurs. WNMIP directly generates a non-maskable interrupt (level 15 interrupt). This bit is read only.

Setting	Function
WNMIP= 0	No Watchdog NMI pending
WNMIP= 1	Watchdog NMI pending

The appropriate interrupt handler has to reset the pending non-maskable interrupt. This is done with the WNMIR bit (Watchdog NMI Reset), in the *gen_purpose2* register. It must be set to one to reset the Watchdog NMI.

In addition, the Power Fail Detect bit in the Aux 2 Register of the NCR89C105 must be set to clear the interrupt. Please refer to the *NCR SBus I/O Chipset Data Manual* for the Aux 2 Register.

Physical Address	Register Name	Read/Write	Access
$7138 0007	gen_purpose2	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WNMIR	ACSTAT	ACNMIR	ACNMIP	DVMA_ ENA	IRQ15_ ENA	US_DEV	BT_US

NOTE: WNMIR is a write only bit.

3.8.2 Rotary Switch

The CPU-5CE provides an additional rotary switch for user selectable settings. See “Highlighted Diagram of the CPU-5CE” on page 51 for the position of the rotary switch on the board. It is a hexadecimal rotary switch, decoded with 4 bits. The status of the rotary switch can be read in the *gen_purpose*1 register.

Physical Address	Register Name	Read/Write	Access
$7138 0005	gen_purpose1	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SYSSTAT	SYSVME	SYSNMIP	ABNMIP	ROT[3..0]			

The table on the next page shows the rotary switch settings and the corresponding values of the bits ROT[3..0], which you can read from the *gen_purpose*1 register.

Table 59: Rotary Switch Settings

Setting	ROT[3]	ROT[2]	ROT[1]	ROT[0]
$0	1	1	1	1
$1	1	1	1	0
$2	1	1	0	1
$3	1	1	0	0
$4	1	0	1	1
$5	1	0	1	0
$6	1	0	0	1
$7	1	0	0	0
$8	0	1	1	1
$9	0	1	1	0
$A	0	1	0	1
$B	0	1	0	0
$C	0	0	1	1
$D	0	0	1	0
$E	0	0	0	1
$F	0	0	0	0

3.9 Additional Registers

The following additional registers are provided on the CPU-5CE to increase functionality. They are used to control the VMEbus interface, the diagnostic LED on the front panel, the hardware watchdog timer and flash memories. These additional registers are also used to handle level 15 interrupts caused by the abort key, VME SYSFAIL and VME ACFAIL.

This information on the following pages gives a summary of all additional registers on the CPU-5CE.

3.9.1 *vme_slavebase1* Register

The *vme_slavebase1* register is for serial loading of the VMEbus slave base address in the Enhanced Slave Mode. If you read the 8-bit value from that register, you get the (U)pper boundary of the set VMEbus slave base address, bits A27_U..A20_U.

Physical Address	Register Name	Read/Write	Access
$7138 0000	vme_slavebase1	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
A27_U	A26_U	A25_U	A24_U	A23_U	A22_U	A21_U	A20_U

NOTE: The bits A27_U..A20_U are read only.

3.9.2 *vme_slavebase2* Register

If you read the 8-bit value of the *vme-slavebase2* register, you get the (L)ower boundary of the VMEbus slave base address, bits A27_L..A20_L.

Writing any data to that register clears the pending non-maskable interrupt of the ABORT switch.

Physical Address	Register Name	Read/Write	Access
$7138 0001	vme_slavebase2	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
A27_L	A26_L	A25_L	A24_L	A23_L	A22_L	A21_L	A20_L

NOTE: The bits A27_L..A20_L are read only.

3.9.3 How to Program the VMEbus Slave Base Address

The complete VMEbus slave base address in the enhanced slave mode consists of the following three parts:

- The 4-bit (B)ase address, this is the address lines A31..A28
- The 8-bit (U)pper boundary, this is the address lines A27..A20
- The 8-bit (L)ower boundary, this is also the address lines A27..A20

To program that complete slave base address, exactly(!) 8 bytes must be written to the *vme_slavebase1* register, each containing the following data:

Byte#	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte#1	-	-	-	-	-	A28_B	A20_L	A20_U
Byte#2	-	-	-	-	-	A29_B	A21_L	A21_U
Byte#3	-	-	-	-	-	A30_B	A22_L	A22_U
Byte#4	-	-	-	-	-	A31_B	A23_L	A23_U
Byte#5	-	-	-	-	-	1	A24_L	A24_U
Byte#6	-	-	-	-	-	1	A25_L	A25_U
Byte#7	-	-	-	-	-	1	A26_L	A26_U
Byte#8	-	-	-	-	-	1	A27_L	A27_U

NOTE: “-” means don’t care

3.9.4 *vme_slavebase*3 Register

If you read the 8-bit value of the *vme-slavebase*3 register, you get the 256 Mbyte boundary (B)ase address of the VMEbus slave base address, bits A31_B..A28_B.

Writing any data to that register clears the pending non-maskable interrupt of the VME SYSFAIL signal.

Physical Address	Register Name	Read/Write	Access
$7138 0002	vme_slavebase3	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	1	1	1	A31_B	A30_B	A29_B	A28_B

NOTE: The bits A31_B..A28_B are read only. Bits [7..4] are always read as ones if the program algorithm for the serial loading of the VMEbus slave base address is used. This is described in the chapter "*vme_slavebase1* register"

3.9.5 *vme_ctl* Register

The *vme_ctl* register is used for settings regarding the VMEbus master and slave interface.

Physical Address	Register Name	Read/Write	Access
$7138 0003	vme_ctl	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Reserved	SMODE	SUPV	SAMODE	Reserved	WIN2	WIN1	WIN0

Initialization: All bits are cleared to 0s at reset.

Description:

WIN2..WIN0 set the VMEbus slave window size if the enhanced slave mode is selected.

WIN2	WIN1	WIN0	Window Size
0	0	0	1 Mbyte
0	0	1	2 Mbyte
0	1	0	4 Mbyte
0	1	1	8 Mbyte
1	0	0	16 Mbyte
1	0	1	32 Mbyte
1	1	0	64 Mbyte
1	1	1	no function

SAMODE sets the VMEbus Slave Address Mode

Setting	Function
SAMODE = 0	VME Extended A32 Mode
SAMODE = 1	VME Standard A24 Mode

SUPV distinguishes between non-privileged and supervisory mode in the VMEbus master interface.

Setting	Function
SUPV = 0	Non-privileged Address Mode
SUPV = 1	Supervisory Address Mode

SMODE sets the VMEbus Slave Mode. There is a “default” mode, which is a compatible mode to the SPARC CPU-2CE and there is an “enhanced” mode, which has improved features regarding the VMEbus slave window.

Setting	Function
SMODE = 0	Enhanced Mode
SMODE = 1	Default Mode

3.9.6 *vme_a32map* Register

The *vme_a32map* register integrates the programming of the four upper VMEbus master address bits as well as other functions.

Physical Address	Register Name	Read/Write	Access
$7138 0004	vme_a32map	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
FL_PAG	VPPCTL	WTENA	WNMIP	VME_ A[31..28]			

Initialization: All bits are cleared to 0s at reset.

Description:

VME_A[31..28] are the four upper VMEbus address lines at a VME master access.

WNMIP reflects the watchdog non-maskable interrupt pending signal, which is active if the hardware watchdog timer reaches its first timeout. WNMIP directly generates a non-maskable interrupt (level 15 interrupt), if the IRQ15_ENA bit in the *gen_purpose2* register is set to enable. WNMIP is a read only bit.

Setting	Function
WNMIP= 0	No Watchdog NMI pending
WNMIP= 1	Watchdog NMI pending

WTENA enables the hardware watchdog timer. This bit is also used to retrigger the watchdog timer, so that it can not reach the timeout.

Note: When the watchdog timer is enabled once, it can't be disabled anymore.

Setting	Function
WTENA= 0	Watchdog disabled
WTENA= 1	Watchdog enabled (to retrigger write a one!)

VPPCTL controls the +12V programming voltage for all flash memory devices, that is for the Boot EPROM area and the User EPROM area.

Setting	Function
VPPCTL = 0	No flash EPROM programming possible
VPPCTL = 1	Flash EPROM programming possible

FL_PAG supports, in conjuction with BT_US and US_DEV of the *gen_purpose2* register, the programming of the flash memory devices. It selects a 512 Kbyte page within the Boot EPROM area or the User EPROM area. Please refer to chapter "Programming the On-board Flash Memories" on page 77 for the settings of FL_PAG.

3.9.7 *gen_purpose1* Register

The *gen_purpose1* register combines various functions such as reading the rotary switch setting, VME SYSFAIL control and ABORT key support.

Physical Address	Register Name	Read/Write	Access
$7138 0005	gen_purpose1	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SYSSTAT	SYSVME	SYSNMIP	ABNMIP	ROT[3..0]			

Initialization: Bits [6..4] are cleared to 0s at reset.

Description:

ROT[3..0] reflect the status of the rotary switch setting. Please refer to “Rotary Switch” on page 105 for all possible bit variations.

ABNMIP reflects the abort key non-maskable interrupt pending signal, which is active if the abort key was toggled. ABNMIP directly generates a non-maskable interrupt (level 15 interrupt), if the IRQ15_ENA bit in the *gen_purpose2* register is set to enable. ABNMIP is a read only bit.

Setting	Function
ABNMIP= 0	No Abort Key NMI pending
ABNMIP= 1	Abort Key NMI pending

SYSNMIP reflects the VMEbus SYSFAIL non-maskable interrupt signal, which becomes active if either the CPU-5CE generates a SYSFAIL signal, or another VMEbus partner causes it to be active. It only becomes active with a high-to-low rising edge of SYSFAIL. The SYSNMIP directly generates a non-maskable interrupt (level 15 interrupt) if the IRQ15_ENA bit in the *gen_purpose2* register is set to enable. SYSNMIP is a read only bit.

Setting	Function
SYSNMIP= 0	No SYSFAIL NMI pending
SYSNMIP= 1	SYSFAIL NMI pending

SYSVME is directly an output to the VMEbus SYSFAIL signal and therefore can assert/negate the VMEbus SYSFAIL.

Setting	Function
SYSVME= 0	Asserts SYSFAIL on VMEbus
SYSVME= 1	Negates SYSFAIL on VMEbus

SYSSTAT reflects the status of the VMEbus SYSFAIL signal. It is a read only bit.

Setting	Function
SYSSTAT= 0	SYSFAIL active on VMEbus
SYSSTAT= 1	SYSFAIL inactive on VMEbus

3.9.8 *led_display* Register

The *led_display* register directly controls the front panel diagnostic LED display. Please refer to the chapter “Diagnostic LED (Hex Display)” on page 101 for the settings of that register.

Physical Address	Register Name	Read/Write	Access
$7138 0006	led_display	w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DP	SEG_G	SEG_F	SEG_E	SEG_D	SEG_C	SEG_B	SEG_A

3.9.9 *gen_purpose2* Register

The *gen_purpose*2 register combines various functions such as controlling the programming of the flash memories, VME ACFAIL control, hardware watchdog support, NMI level 15 enable, VMEbus DVMA enable.

Physical Address	Register Name	Read/Write	Access
$7138 0007	gen_purpose2	r/w	8 bit

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WNMIR	ACSTAT	ACNMIR	ACNMIP	DVMA_ENA	IRQ15_ENA	US_DEV	BT_US

Initialization: Bits [4..0] are cleared to 0s at reset.

Description:

BT_US and **US_DEV** supports in conjuction with FL_PAG of the *gen_purpose*1 register the programming of the flash memory devices. They select a 512 Kbyte page within the Boot EPROM area or the User EPROM area. Please refer to chapter “Programming the On-board Flash Memories” on page 77 for the settings of BT_US and US_DEV.

IRQ15_ENA controls the capability to generate a non-maskable interrupt with level 15 by the Abort Key, SYSFAIL, ACFAIL and the Hardware Watchdog Timer.

Setting	Function
IRQ15_ENA= 0	Disables NMI capability of Abort Key, SYSFAIL, ACFAIL, and the Hardware Watchdog Timer
IRQ15_ENA= 1	Enables NMI capability of Abort Key, SYSFAIL, ACFAIL, and the Hardware Watchdog Timer

DVMA_ENA enables the VMEbus slave accesses.

Setting	Function
DVMA_ENA= 0	Disables VME slave accesses
DVMA_ENA= 1	Enables VME slave accesses

ACNMIP is a double functional bit. Reading the ACNMIP bit reflects the VMEbus ACFAIL non-maskable interrupt signal, which becomes active if the VMEbus ACFAIL signal is asserted. It only becomes active with a high-to-low rising edge of ACFAIL. The ACNMIP directly generates a non-maskable interrupt (level 15 interrupt) if the IRQ15_ENA bit is set to enable. Writing the ACNMIP bit enables or disables the VMEbus interrupt lines treated by the onboard interrupt handler.

Setting	Function
Read ACNMP = 0	No ACFAIL NMI pending
Read ACNMP = 1	ACFAIL NMI pending
Write ACNMIP = 0	Disable all VMEbus interrupts
Write ACNMIP = 1	Enable all VMEbus interrupts

ACNMIR clears the pending non-maskable interrupt, which was initiated by the VMEbus ACFAIL signal. To reset the pending NMI, write a one to that bit. ACNMIR is a write only bit.

ACSTAT reflects the status of the VMEbus ACFAIL signal. It is a read only bit.

Setting	Function
ACSTAT= 0	ACFAIL active on VMEbus
ACSTAT= 1	ACFAIL inactive on VMEbus

WNMIR clears the pending non-maskable interrupt, which was initiated by the hardware watchdog timer. To reset the pending NMI, write a one to that bit. WNMIR is a write only bit.

SECTION 4 CIRCUIT SCHEMATICS

4. General Index to the CPU-5CE Schematics

Copies of the CPU-5CE schematics are found on the next page. The first page of the schematics includes the index.

4.1 Signal and Unit Cross References

Copies of the signal and unit cross references are found on the next page.

4.2 IOBP-10 Schematics

Copies of the IOBP-10 schematics are found on the next page.

SECTION 5 OPENBOOT ENHANCEMENTS

5. OpenBoot

This section describes the enhancements to the standard OpenBoot firmware that have been done for the SPARC CPU-5CE. For a description of standard OpenBoot firmware features, please see the *OPEN BOOT PROM 2.0 MANUAL SET*.

Besides the commands already provided by the standard OpenBoot firmware, the OpenBoot firmware available on the SPARC CPU-5CE includes additional words for the following:

- accessing and controlling the VMEbus interface,
- accessing and programming available flash memories,
- controlling the operating mode of the Watchdog Timer, and
- 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 show 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 dash "—".

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

The OpenBoot ported to the SPARC CPU-5CE is based upon the OpenBoot 2.15 obtained from Sun Microsystems.

5.1 Controlling the VMEbus Master and Slave Interface

5.1.1 VMEbus addressing

The VMEbus has a number of distinct address spaces represented by a subset of the 64 possible values encoded by the 6 address modifier bits. The size of the address space depends on the particular address space, for example the *standard* (A24) address space is limited to 16 MByte, whereas the *extended* (A32) address space allows to address 4 GByte. An additional bit – which corresponds with the VMEbus LWORD* signal – is used to select between 16-bit and 32-bit data.

A *physical* VMEbus address is represented numerically by the pair *phys.high* (also called *space*) and *phys.low* (also called *offset*). The *phys.high* consists of the 6 address modifier bits `AM0` through `AM5` corresponding with bit `0` through `5`; and the data width bit `LWORD*` (`0`= 16-bit data, `1` = 32-bit data) in bit `6`.
OpenBoot provides a number of constants combining the information mentioned above. These constants are called `AML` constants. `AML` is the combination of the first letters of the words Address Modifier and `LWORD*`. Each `AML` constant specifies a unique address space:

`vmea16d16` (— *h# 2d*) returns the `AML` constant `2D16` identifying the privileged **short** (A16) address space with 16-bit data transfers.

`vmea16d32` (— *h# 6d*) returns the `AML` constant `6D16` identifying the privileged **short** (A16) address space with 32-bit data transfers.

`vmea24d16` (— *h# 3d*) returns the `AML` constant `3D16` identifying the privileged **standard** (A24) address space with 16-bit data transfers.

`vmea24d32` (— *h# 7d*) returns the `AML` constant `7D16` identifying the privileged **standard** (A24) address space with 32-bit data transfers.

`vmea32d16` (— *h# 0d*) returns the `AML` constant `0D16` identifying the privileged **extended** (A32) address space with 16-bit data transfers.

`vmea32d32` (— *h# 4d*) returns the `AML` constant `4D16` identifying the privileged **extended** (A32) address space with 32-bit data transfers.
The `AML` modifiers described below are available to modify the `AML` in such a way that additional VMEbus address spaces may be identified:

`burst` (*phys.high-single — phys.high-burst*) converts the numeric representation of any VMEbus `AML` constant in single-transaction form to its burst-transaction (BLT) form.

```
ok vmea24d32 burst.
3f
ok
```

`vme-user` (*phys.high-privileged — phys.high-non-privileged*) converts the numeric representation of any VMEbus `AML` constant in privileged form to its non-privileged (user-mode) form.

```
ok vmea16d32 vme-user .
69
ok
```

`vme-program` (*phys.high-data — phys.high-program*) converts the numeric representation of any VMEbus `AML` constant in data-transaction form to its program form.

```
ok vmea32d16 vme-program .
e
ok
```

The *offset* specifies the VMEbus address of an area within the selected address *space*. The value of the offset depends on the address space. For example the **standard** (A24) address space is limited to 16 MByte (24-bit addresses ranging from `00.000016` to `FF.FFFF16`), whereas the **extended** (A32) address space allows to address 4 GByte (32-bit addresses ranging from `0000.000016` to `FFFF.FFFF16`), and the **short** (A16) address space is limited to 64 KByte (16-bit addresses ranging from `000016` to `FFFF16`).

Example:

The example below shows how to specify the address of a VMEbus board that is accessible within the **extended** (A32) address space (`vmea32d32`) beginning at offset `4080.000016`:

```
ok h# 4080.0000 vmea32d32
```

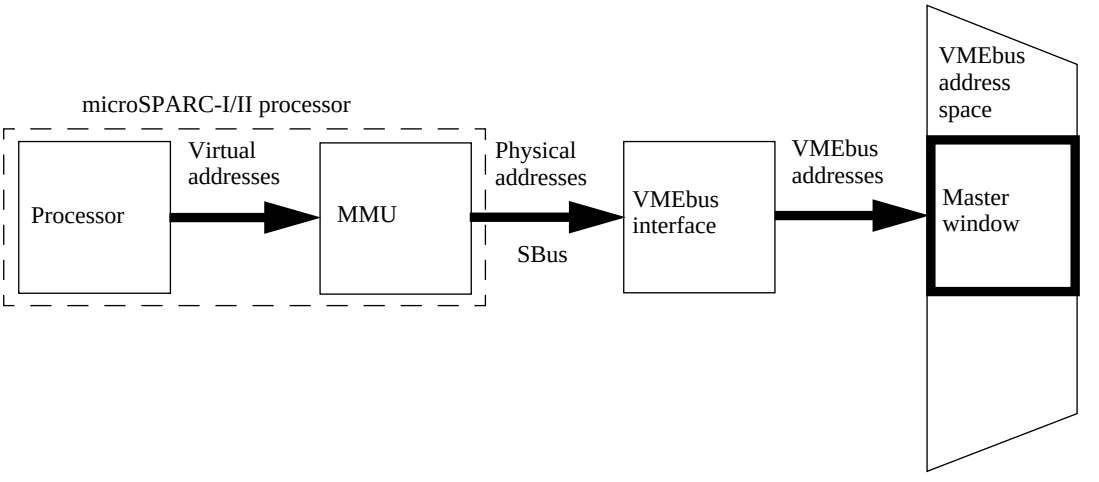
The first part represents the *offset* (*phys.low*) and the second part represents the *space* (*phys.high*).

5.1.2 VMEbus Master Interface

5.1.2.1 SPARC/CPU-5CE

As shown in the figure below the processor emits virtual addresses during a data transfer cycle which are translated to physical addresses by the MMU. Within a microSPARC-I/II environment, the VMEbus is connected with the SBus and the VMEbus interface responds to unique physical SBus addresses and executes the appropriate VMEbus transfer. Depending on the physical addresses and the state of specific registers within the VMEbus interface, the interface addresses a specific VMEbus address space.

FIGURE 24. Address translation (master): microSPARC – SBus – VMEbus



Before the processor may access a specific area within one of the VMEbus address spaces, the steps described below must be taken:

- The VMEbus interface has to be set up to respond to specific physical SBus addresses to forward the access to a certain VMEbus address space.
- The contents of the MMU table are modified to make the SBus address range available to the processor's address range and thus allowing accesses to the specific VMEbus area using virtual addresses. In general, this means that the VMEbus area is made available to the processor's virtual address space.
- The VMEbus interface has to be enabled, in order to allow accesses to the VMEbus address space.

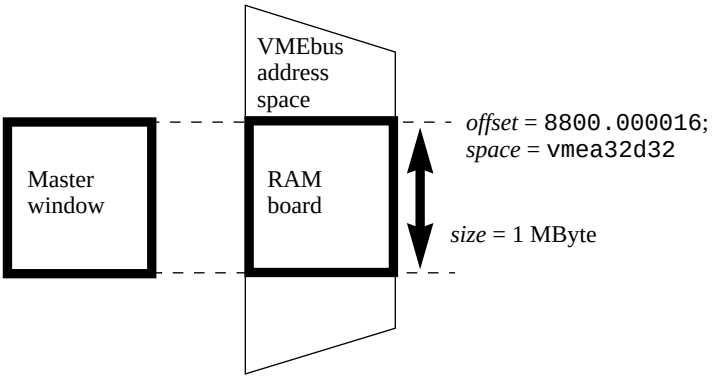
OpenBoot provides commands to make VMEbus areas available to the processor's virtual address space and to remove these VMEbus areas from the processor's virtual address space. The command `vme-memmap` performs all steps to make specified VMEbus areas available to the processor's virtual address space. The command `vme-free-virtual` removes the VMEbus area which has been made available previously from the processor's virtual address space.

`vme-memmap` (*offset space size — vaddr*) initializes the VMEbus master interface according to the parameters *offset* and *space* and returns the virtual address *vaddr* to be used to access the specified VMEbus area.
The parameters *space* and *offset* describe the VMEbus address area in detail: *offset* specifies the physical VMEbus address of the area to be accessed and *space* specifies the address space where the VMEbus area is located in. The size of the VMEbus area is given by *size*.

Example:

Assumed a memory board is accessible within the **extended** (A32) VMEbus address space beginning at address `8800.000016` and ranging to `880F.FFFF16` (1 MByte) as shown in the figure below:

FIGURE 25. Mapping a VMEbus area to the processor's virtual-address space



In order to make this VMEbus area available to the processor's virtual address space, the commands listed below have to be used:

```
ok 0 value vme-ram
ok h# 8800.0000 vmea32d32 1Meg vme-memmap is vme-ram
ok
```

The first command defines a variable `vme-ram` which is later used to store the virtual address of the VMEbus area. The second command listed above makes 1 MByte beginning at physical address `8800.000016` within the **extended** (A32) VMEbus address space available to the processor's virtual address space. The virtual address returned by the command is stored in the variable `vme-ram` which has been defined by the first command `value`. The variable `vme-ram` may be used later to access this VMEbus area.

`vme-free-virtual` (*vaddr size* —) removes the VMEbus area associated with the virtual address *vaddr* from the processor's virtual address space.

The VMEbus area previously made available to the processor's virtual address space is removed from the virtual address space using the `vme-free-virtual` command as shown below:

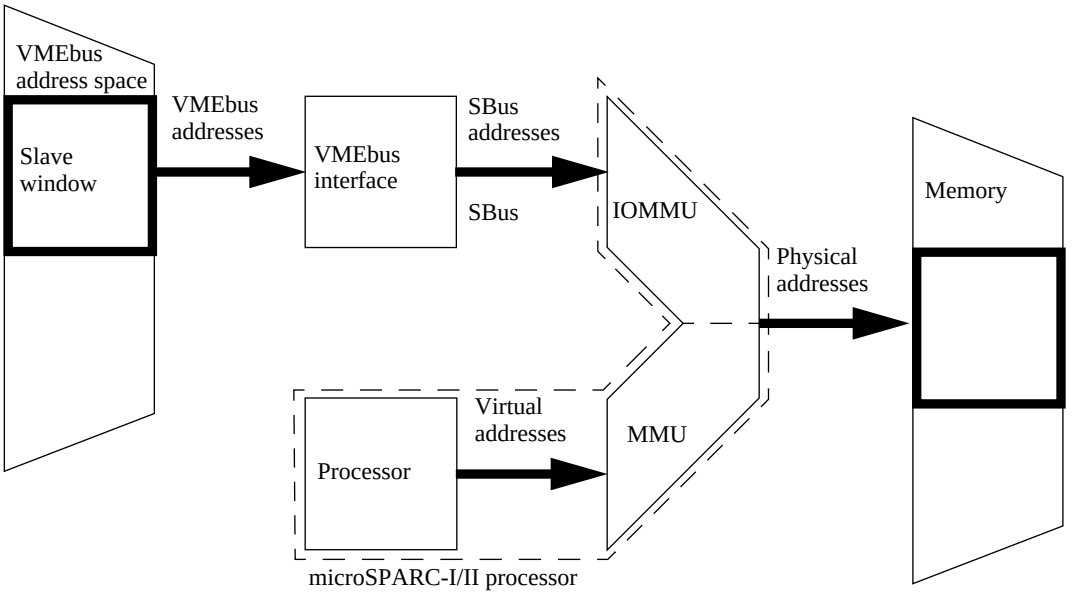
```
ok vme-ram 1Meg vme-free-virtual
ok
```

5.1.3 VMEbus Slave Interface

5.1.3.1 SPARC/CPU-5CE

As shown in the figure below the VMEbus interface responds to unique VMEbus addresses and translates these addresses to virtual SBus addresses. The IOMMU translates these virtual SBus addresses to physical addresses, which address a certain area within the on-board memory.

FIGURE 26. Address translation (slave): VMEbus – SBus – microSPARC



The processor accesses the same on-board memory by applying virtual addresses to the MMU which are translated to the appropriate physical addresses.
Before another VMEbus master may access the on-board memory, the following steps have to be taken to make a certain amount of on-board memory available to one of the VMEbus address spaces, e.g. s*tandard* (A24), or *extended* (A32) address space:

- A certain amount of the available on-board memory has to be allocated to make it available to one of the VMEbus address spaces.
- The VMEbus interface has to be set up to respond to specific addresses within the selected VMEbus address spaces. In general, registers within the VMEbus interface are modified to accomplish this.
- The contents of the IOMMU table are modified to associate the virtual SBus addresses, which are emitted by the VMEbus interface during a slave access, with the physical addresses of the allocated memory. Furthermore, the contents of the MMU table are

modified to associate the virtual addresses, which are emitted by the processor during accesses to the on-board memory, with the physical addresses of the allocated memory.

- The VMEbus interface has to be enabled, in order to allow accesses from the VMEbus to the on-board memory.

OpenBoot provides commands to make the on-board memory available to one of the VMEbus address spaces, and to remove the on-board memory from these VMEbus address spaces. The command `set-vme-slave` performs all steps to make a specified amount of memory available at a specific VMEbus address space. The command `reset-vme-slave` removes the on-board memory from the VMEbus address space.

`set-vme-slave` (*offset space size — vaddr*) initializes the VMEbus slave interface according to the parameters passed to the command and returns the virtual address *vaddr* of the memory which has been made available to the VMEbus. OpenBoot provides all necessary mappings (MMU and IOMMU) to access the memory from the processor and the VMEbus.
The parameters *space* and *offset* specify where the slave interface is accessible within the VMEbus address range. The parameter *offset* specifies the physical base address of the slave interface within the particular address space. The size of the memory that should be made available to the VMEbus is given by *size*.

Example:
Assumed that 1 MByte of on-board memory should be made available to the **extended** (A32) address space of the VMEbus beginning at the VMEbus address `4080.000016`, the commands listed below have to be used.

```
ok 0 value my-mem
ok 4080.0000 vmea32d32 1meg set-vme-slave is my-mem
ok
```

The first command defines a variable `my-mem` which is later used to store the virtual address of the on-board memory which has been made available to the VMEbus. The second command listed above makes 1 MByte beginning at physical address `4080.000016` available within the **extended** (A32) VMEbus address space. The virtual address returned by the command is stored in the variable `my-mem` which has been defined by the first command `value`. The variable `my-mem` may be used later to access the on-board memory.

`reset-vme-slave` (*vaddr size —*) resets the VMEbus slave interface associated with the virtual address *vaddr* and destroys all mappings which were necessary to make the memory available to VMEbus.

```
ok my-mem 1Meg reset-vme-slave
ok
```

5.2 VMEbus Interface

The VMEbus interface on the SPARC CPU-5CE consists of the S4 chip and additional circuitry.

5.2.1 Generic Information

The commands described below are used to retrieve generic information about the VMEbus interface:

`s4-va` (— *vaddr*) returns the virtual base address *vaddr* of the registers included in the S4.

`vmectl-va` (— *vaddr*) returns the virtual base address *vaddr* of additional control and status registers included in the VMEbus interface.

`s4-ctl` (— *vaddr*) returns the virtual base address *vaddr* of the registers included in the S4.

`vmectl` (— *vaddr*) returns the virtual base address *vaddr* of additional control and status registers included in the VMEbus interface.

5.2.2 Register Addresses

The commands described below are used to obtain the virtual addresses of specific registers in the S4 and the additional VMEbus Interface Register.

`s4-bus-locker` (— *vaddr*) returns the virtual address *vaddr* of the S4's **Bus Locker Register**.

`s4-intr-monitor` (— *vaddr*) returns the virtual address *vaddr* of the S4's **Interrupt Monitor Register**.

`s4-mbox-intr-level` (— *vaddr*) returns the virtual address *vaddr* of the S4's **Mailbox Interrupt Level Register**.

`s4-mbox-ctrl` (— *vaddr*) returns the virtual address *vaddr* of the S4's **Mail Box Register**.

`s4-intr-ena` (— *vaddr*) returns the virtual address *vaddr* of the S4's **Interrupt Enable Register**.

`s4-a32map` (— *vaddr*) returns the virtual address *vaddr* of the S4's **A32 Map Register**.

`s4-slavemap` (— *vaddr*) returns the virtual address *vaddr* of the S4's **Slave Map Register**.

`s4-iack-cycle` (*level* — *vaddr*) returns the virtual address *vaddr* of the S4's **IACK Cycle Register** associated with the given *level*. The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels.

Only the least significant three bits of *level* are considered, and when *level* is zero, then the command treats it as if the value "one" has been passed to the command.

`vme-slavebase1` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus Slave Base Register 1**.

`vme-slavebase2` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus Slave Base Register 2**.

`vme-slavebase3` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus Slave Base Register 3**.

`vme-ctrl` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus Control Register**.

`vme-a32-map` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus A32 Map Register**.

`vme-gpr1` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus General Purpose Register 1**.

`vme-gpr2` (— *vaddr*) returns the virtual address *vaddr* of the VMEbus interface's **VMEbus General Purpose Register 2**.

The commands described below are used to obtain the virtual addresses of specific system configuration registers:

`abort-ctrl` (— *vaddr*) returns the virtual address *vaddr* of the **ABORT Control Register.**

`flash-wdt-csr` (— *vaddr*) returns the virtual address *vaddr* of the **FLASH Memory and Watchdog Timer Control and Status Register**.

`led-display` (— *vaddr*) returns the virtual address *vaddr* of the **Seven Segment Display Control Register**.

`gpr1` (— *vaddr*) returns the virtual address *vaddr* of the **General Purpose Register 1**.

`gpr2` (— *vaddr*) returns the virtual address *vaddr* of the **General Purpose Register 2**.

5.2.3 Register Accesses

The FORTH words described below are used to read data from, and to store data in, specific registers of the S4 and VMEbus interface control and status registers:

`s4-bus-locker@` (— *byte*) returns the contents — an 8-bit data — of the S4's Bus Locker Register.

`s4-bus-locker!` (*byte* —) stores the 8-bit data *byte* in the S4's Bus Locker Register.

`s4-intr-monitor@` (— *byte*) returns the contents — an 8-bit data — of the S4's Interrupt Monitor Register.

`s4-intr-monitor!` (*byte* —) stores the 8-bit data *byte* in the S4's Interrupt Monitor Register.

`s4-mbox-intr-level@` (— *byte*) returns the contents — an 8-bit data — of the S4's Mailbox Interrupt Level Register.

`s4-mbox-intr-level!` (*byte* —) stores the 8-bit data *byte* in the S4's Mailbox Interrupt Level Register.

`s4-mbox-ctrl@` (— *byte*) returns the contents — an 8-bit data — of the S4's Mail Box Register.

`s4-mbox-ctrl!` (*byte* —) stores the 8-bit data *byte* in the S4's Mail Box Register.

`s4-intr-ena@` (— *byte*) returns the contents — an 8-bit data — of the S4's Interrupt Enable Register.

`s4-intr-ena!` (*byte* —) stores the 8-bit data *byte* in the S4's Interrupt Enable Register.

`s4-a32map@` (— *byte*) returns the contents — an 8-bit data — of the S4's A32 Map Register.

`s4-a32map!` (*byte* —) stores the 8-bit data *byte* in the S4's A32 Map Register.

`s4-slavemap@` (— *byte*) returns the contents — an 8-bit data — of the S4's Slave Map Register.

`s4-slavemap!` (*byte* —) stores the 8-bit data *byte* in the S4's Slave Map Register.

`s4-iack-cycle@` (*level — vector*) ...
Only the least significant three bits of *level* are considered, and when *level* is zero, then the command treats it as if the value “one” has been passed to the command.

`vme-slavebase1@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s VMEbus Slave Base Register 1.

`vme-slavebase1!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface’s VMEbus Slave Base Register 1.

`vme-slavebase2@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s VMEbus Slave Base Register 2.

`vme-slavebase2!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface’s VMEbus Slave Base Register 2.

`vme-slavebase3@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s VMEbus Slave Base Register 3.

`vme-slavebase3!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface’s VMEbus Slave Base Register 3.

`vme-ctrl@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s Control Register.

`vme-ctrl!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface’s Control Register.
`vme-a32-map@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s VMEbus A32 Map Register.

`vme-a32-map!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface’s VMEbus A32 Map Register.

`vme-gpr1@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s VMEbus General Purpose Register 1.

`vme-gpr1!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface’s VMEbus General Purpose Register 1.

`vme-gpr2@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface’s VMEbus General Purpose Register 2.

`vme-gpr2!` (*byte* —) stores the 8-bit data *byte* in the VMEbus interface's VMEbus General Purpose Register 2.

The commands described below are used to read data from, and to store data in, specific system configuration registers:

`abort-ctrl!` (*byte* —) stores the 8-bit data *byte* in the ABORT Control Register.

`flash-wdt-csr@` (— *byte*) returns the contents — an 8-bit data — of the FLASH Memory and Watchdog Timer Control and Status Register.

`flash-wdt-csr!` (*byte* —) stores the 8-bit data *byte* in the FLASH Memory and Watchdog Timer Control and Status Register.

`led-display!` (*byte* —) stores the 8-bit data *byte* in the Seven Segment Display Control Register.

`gpr1@` (— *byte*) returns the contents — an 8-bit data — of the General Purpose Register 1.

`gpr1!` (*byte* —) stores the 8-bit data *byte* in the General Purpose Register 1.

`gpr2@` (— *byte*) returns the contents — an 8-bit data — of the General Purpose Register 2.

`gpr2!` (*byte* —) stores the 8-bit data *byte* in the General Purpose Register 2.

5.2.4 VMEbus Interrupt Handler

`vme-intr@` (— *byte*) returns the contents — an 8-bit data — of the S4's Interrupt Monitor Register.

`vme-intena@` (— *byte*) returns the contents — an 8-bit data — of the S4's Interrupt Enable Register.

`vme-intena!` (*byte* —) stores the 8-bit data *byte* in the S4's Interrupt Enable Register.

`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*. The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels.

Only the least significant three bits of *level* are considered, and when *level* is zero, then the command treats it as if the value "one" has been passed to the command.

The command verifies the state of the bit in the VMEbus Interrupt Status Register associated with the given *level*. When the corresponding status bit is set, then no VMEbus interrupt is pending and the command returns *false*. Otherwise — the status bit is cleared — the value *true* is returned.

`vme-iack@` (*level* — *vector*) initiates an interrupt acknowledge cycle at the given VMEbus interrupt request *level* and returns the obtained 8-bit *vector*. The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels.

Typically, the *vector* returned is within the range of 0 through 255. However, when no interrupt is pending and consequently no interrupt has to be acknowledged, then the value -1 is returned.

Only the least significant three bits of *level* are considered, and when *level* is zero, then the command treats it as if the value "one" has been passed to the command.

`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 interrupt asserted by the S4 when a certain VMEbus interrupt request level is asserted. The value of *mapping* may be one of the values in the range of one through seven. Each value specifies one of the seven S4 interrupt request lines. The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels.

Only the least significant three bits of *mapping* and *level* are considered, and when *level* is zero, then the command treats it as if the value "one" has been passed to the command.

Because the VMEbus interface on the SPARC CPU-5CE does not allow to *map* a VMEbus interrupt to any SBus interrupt level, the values of *mapping* and *level* passed

to the command must be the same. To enable the VMEbus interrupt request level 5, the parameters listed in the example below have to be passed to the command:

```
ok 5 5 vme-intr-ena
ok
```

`vme-intr-dis` (*level* —) disables the interrupt to be generated when the specified VMEbus interrupt request at *level* is asserted. The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels.
Only the least significant three bits of *level* are considered, and when *level* is zero, then the command treats it as if the value “one” has been passed to the command.

`install-vme-intr-handler` (*mapping level* —) installs the interrupt service routine dealing with the given VMEbus interrupt *level*. The parameter *mapping* defines the interrupt request line asserted by the S4 when a certain VMEbus interrupt request level is asserted. The value of *mapping* may be one of the values in the range of one through seven. Each value specifies one of the seven S4 interrupt request lines.
The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels. The address of the interrupt service routine currently in effect is preserved.
Only the least significant three bits of *mapping* and *level* are considered, and when *level* is zero, then the command treats it as if the value “one” has been passed to the command.
Because the VMEbus interface on the SPARC CPU-5CE does not allow to *map* a VMEbus interrupt to any SBus interrupt level, the values of *mapping* and *level* passed to the command must be the same. To enable the VMEbus interrupt request level 2, the parameters listed in the example below have to be passed to the command:

```
ok 2 2 install-vme-intr-handler
ok
```

`uninstall-vme-intr-handler` (*level* —) removes the interrupt service routine dealing with the given VMEbus interrupt *level* and installs the *old* interrupt service routine. The value of *level* may be one of the values in the range of one through seven. Each value specifies one of the seven VMEbus interrupt request levels.
Only the least significant three bits of *level* are considered, and when *level* is zero, then the command treats it as if the value “one” has been passed to the command.

`.vme-vectors` (—) displays the VMEbus interrupt vectors received during the last interrupt acknowledge cycle.
OpenBoot maintains seven variables called `vme-intr{1|2|3|4|5|6|7}-vec-`

`tor` 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.

5.2.5 VMEbus Arbiter

The commands described below are available to control the VMEbus arbiter as well as to retrieve information about the state of the VMEbus arbiter.

`vme-arb-mode@` (— *mode*) returns the *mode* the arbiter is currently operating in. The value of *mode* may range from zero to one. Each value specifies a particular mode: the value zero indicates that the arbiter is operating in the *priority* mode — which means that the arbiter is operating as a single-level arbiter at level 3. The value one specifies the *round-robin* mode.

`vme-arb-mode!` (*mode* —) selects the arbiter mode specified by *mode*. The value of *mode* may range from zero to one. Each value specifies a particular mode: the value zero indicates that the arbiter operates in the *priority* mode — which means that the arbiter is operating as a single-level arbiter at level 3. The value one specifies the *round-robin* mode.
Two constants are available to specify one of the two arbiter modes: `pri` (0_{10}) and `rro` (1_{10}).

5.2.6 VMEbus Requester

The commands described below are available to control the VMEbus requester as well as to retrieve information about the state of the VMEbus requester.

`vme-buslock@` (— *byte*) returns the contents — an 8-bit data — of the S4's Bus Locker Register.

`vme-buslock!` (*byte* —) stores the 8-bit data *byte* in the S4's Bus Locker Register.

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

`vme-bus-request-mode!` (*mode* —) selects the bus-request *mode* to be used when the VMEbus is being accessed.
Two constants are available to specify one of the two request modes: `fair` (0_{10}) and `unfair` (1_{10}).

`vme-bus-capture!` (*true* | *false* —) enables or disables the *bus-capture-and-hold* capa-

bility of the S4. If the value *true* is passed to the command, the VMEbus interface starts to capture the bus, and when it gains the ownership of the bus, it holds ownership as long as the bus is released. The bus is released when the command is called and the value *false* is passed to it.

`vme-bus-captured?` (— *true* | *false*) determines whether the VMEbus interface gains the ownership of the bus. The value *true* is returned when the VMEbus interface gains the ownership of the VMEbus. Otherwise, the value *false* is returned to indicate that the VMEbus interface has not gained the ownership of the bus.
In general, this command is called immediately after a *capture-and-hold* cycle has been initiated as shown in the example below:

```
ok true vme-bus-capture!
ok begin vme-bus-captured? until
ok ...

ok false vme-bus-capture!
ok
```

5.2.7 VMEbus Status Signals

The commands listed below are available to access and control the VMEbus status signals.

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

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

`vme-sysfail!` (*true* | *false* —) asserts or negates the VMEbus SYSFAIL* signal. When the value *true* is passed to the command, the VMEbus SYSFAIL* signal is asserted. Otherwise — the value *false* is passed to the command — the VMEbus SYSFAIL* signal is negated.

`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@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus Interface's General Purpose Register 1. The state of the most significant three bits SYSSTAT (bit 7), SYSVME (bit 6), and SYSNMIP (bit 5) are preserved, but the least significant five bits are cleared (0).

`vme-sysfail-assert-irq-ena` (—) allows the VMEbus interface to generate an interrupt upon the assertion of the VMEbus SYSFAIL* signal.

`vme-sysfail-assert-irq-dis` (—) disables the interrupt to be generated upon the assertion of the VMEbus SYSFAIL* signal.

`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*.

`reset-sysfail-irq` (—) clears a pending non-maskable interrupt generated by the assertion of the VMEbus SYSFAIL* 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@` (— *byte*) returns the contents — an 8-bit data — of the VMEbus interface's General Purpose Register 2. The state of the bit ACSTAT (bit 6) and ACNMIP (bit 4) are preserved, but all other bits are cleared (0).

`vme-acfail-assert-irq-ena` (—) allows the VMEbus interface to generate an inter-

rupt upon the assertion of the VMEbus ACFAIL* signal.

`vme-acfail-assert-irq-dis` (—) disables the interrupt to be generated upon the assertion of the VMEbus ACFAIL* signal.

`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*.

`reset-acfail-irq` (—) clears a pending non-maskable interrupt generated by the assertion of the VMEbus ACFAIL* signal.

5.2.8 VMEbus Master Interface

The commands listed below are available to control the VMEbus master interface.

`vme-supervisor!` (*true* | *false* —) selects the mode in which the VMEbus is being accessed. When the value *true* is passed to the command, the VMEbus is accessed in *privileged* mode. Otherwise — the value *false* is passed to the command — the VMEbus is accessed in *non-privileged* mode.

`vme-a32map@` (— *#page*) returns the number of the 256 Mbyte page *#page* which is being addressed when the VMEbus is being accessed within the *extended* address space (A32).

`vme-a32map!` (*#page* —) sets the number of the 256 Mbyte page *#page* which will be addressed when the VMEbus is being accessed within the *extended* address space (A32).
Before the command selects the new page, it reads the actual contents of the VMEbus interface's VMEbus A32 Map Register in order to keep the state of the other bits in this register. But the WTENA- (bit 5) and WNMIP (bit 4) bits are **cleared** to prevent the watchdog timer from being started unintentionally.

The following commands require that the NVRAM configuration parameter `use-new-vme`? is set to false.

`set-vme-master` (*size addr* —) initialises the VMEbus interface to access an address range specified by the *size* and the address *addr* of the VMEbus.

The example below shows how to prepare the VMEbus interface to access the VMEbus in the *extended* address range (A32), beginning at address 40800000_{16} and ranging to address $408FFFFF_{16}$.

```
ok 1Meg h# 4080.0000 set-vme-master
ok
```

The next example shows how to prepare the VMEbus interface to access the VMEbus in the *standard* address range (A24), beginning at address 900000_{16} and ranging to address $91FFFF_{16}$.

```
ok h# 2.0000 h# ff90.0000 set-vme-master
ok
```

To prepare the VMEbus interface to access the address range 8000_{16} through $8FFF_{16}$ within the *short* address range (A16) of the VMEbus, the command listed below has to be used.

```
ok h# 1000 h# ffff.8000 set-vme-master
ok
```

The particular VMEbus area can be accessed using the standard commands available in OpenBoot to read and store data. The virtual base address to access the VMEbus is stored in the variable `vmebus`. The example shown below reads a single byte from the VMEbus through the VMEbus interface which has been prepared for accessing the VMEbus using one of the three examples mentioned above.

```
ok vmebus c@
ok
```

The command `set-vme-master` modifies the contents of the following two variables.

`vme-dpr` (— *vaddr*) returns the virtual address *vaddr* of the memory which has been made available to the VMEbus.

`my-vme-base` (— *paddr*) returns the physical address *paddr* of the memory which is accessible from the VMEbus.

`free-vme-mem` (—) releases all virtual and physical memory allocated by the command-`vme-set-master`, and allocated by the commands to setup the VMEbus slave interface.

5.2.9 VMEbus Slave Interface

The commands listed below are available to control the VMEbus slave interface.

`vme-slavewin@` (— *base lower upper*) returns the current contents of the VMEbus interface Slave Base Registers.
The address of the slave window is represented by the triple *base*, *lower*, and *upper*. The value of *base* represents the VMEbus address bits A31 to A28 (one of 16 possible 256 Mbyte pages). The value of *lower* and *upper* represent the size of the slave window that has been made available to the VMEbus. Thus, *lower* encodes the lower boundary of the slave window (address bits A27 to A20), and *upper* encodes the upper boundary of the slave window (address bits A27 to A20).

`vme-slavewin!` (*base lower upper* —) sets the VMEbus interface's Slave Base Registers. The address of the slave window is represented by the triple *base*, *lower*, and *upper*. The value of *base* represents the VMEbus address bits A31 to A28 (one of 16 possible 256 Mbyte pages). The value of *lower* and *upper* represent the size of the slave window that has been made available to the VMEbus. Thus, *lower* encodes the lower boundary of the slave window (address bits A27 to A20), and *upper* encodes the upper boundary of the slave window (address bits A27 to A20).

`vme-slavewin-size!` (*size-code* —) sets the size of the slave window. The value of *size-code* may be one of the values zero through six. Each value specifies one of the seven slave window sizes, as stated in the table below.

size-code	Window Size
0	1 Mbyte
1	2 Mbyte
2	4 Mbyte
3	8 Mbyte
4	16 Mbyte
5	32 Mbyte
6	64 Mbyte

`vme-slavemap@` (— *byte*) returns the contents — an 8-bit data — of the S4's Slave Map Register.

`vme-slavemap!` (*byte* —) stores the 8-bit data *byte* in the S4's Slave Map Register.

`vme-a24mode!` (*true* | *false* —) selects the address range by which the VMEbus slave interface is accessible from the VMEbus. When the value *true* is passed to the command, the VMEbus slave interface is accessible within the *standard* address range (A24). Otherwise — the value *false* is passed to the command — the VMEbus slave interface is accessible within the *extended* address range (A32).

`vme-enhanced-mode!` (*true* | *false* —) selects the operating mode of the VMEbus interface. When the value *true* is passed to the command, the VMEbus slave interface operates in the *enhanced* mode. Otherwise — the value *false* is passed to the command — the VMEbus slave interface operates in the *default* mode.

`vme-dvma-enable@` (— *true* | *false*) checks whether the VMEbus slave interface is enabled or disabled. When the value *true* is returned, the VMEbus slave interface is enabled. Otherwise — the value *false* is returned — the VMEbus slave interface is disabled.

`vme-dvma-enable!` (*true* | *false* —) enables or disables the VMEbus slave interface. When the value *true* is passed to the command, the VMEbus slave interface is enabled. Otherwise — the value *false* is passed to the command — the VMEbus slave interface is disabled.

The following commands require that the NVRAM configuration parameter `use-new-vme`? is set to false.

`set-vme-slave-def` (*size addr* —) initialises the VMEbus interface to operate in the *default* slave mode. In this mode the VMEbus slave interface is accessible within the *standard* address range (A24) of the VMEbus.
The base address and the size of the slave window are specified by the *size* and the *addr*. The size of the slave window in the *default* mode is limited to one Mbyte. The base address of the slave window is given as an index, rather than an absolute address within the standard address range. The value of *addr* may be one of the values in the range of zero through 15. Each value specifies one of the one Mbyte ranges in the standard address range.

The example below shows how to make one Mbyte available to the *standard* address range of the VMEbus beginning at the physical address 900000_{16} .

```
ok 1Meg 9 set-vme-slave-def
ok
```

`set-vme-slave-enh` (*size a31-a28 a27-a20-lower a27-a20-upper* —) initialises the VMEbus interface to operate in the *enhanced* slave mode. In this mode, the VMEbus slave interface is accessible within the *extended* address range (A32) of the VMEbus.
The base address and the size of the slave window are specified by the *size* and the

address specified by *a31-a28*, *a27-a20-lower*, and *a27-a20-upper*. The size of the slave window in the *enhanced* mode is limited to one Mbyte. The base address of the slave window is specified by the following three values: *a31-a28* specifies the 256 Mbyte page in which the slave window is accessible; *a27-a20-lower* and *a27-a20-upper* specify the boundaries of the slave window within the 256 Mbyte page.
The example below shows how to make one Mbyte available to the *extended* address range of the VMEbus beginning at the physical address 23400000_{16} .

```
ok 1Meg 2 34 35 set-vme-slave-enh
ok
```

The two commands described above modify the contents of the following two variables.

`vme-dpr` (— *vaddr*) returns the virtual address *vaddr* of the memory which has been made available to the VMEbus.

`my-vme-base` (— *paddr*) returns the physical address *paddr* of the memory which is accessible from the VMEbus.

`free-vme-mem` (—) releases all virtual and physical memory allocated by the commands `set-vme-slave-def` and `set-vme-slave-enh`, and allocated by the commands to set up the VMEbus master interface.

5.2.10 VMEbus Device Node

The OpenBoot device tree contains the device node for the VMEbus interface and is called "`VME`". It is a *child device* of the device node "`/iommu`" (The full pathname of the VMEbus interface device node is displayed by the command `show-devs`). The device *alias* **`vme`** is available as an abbreviated representation of the VMEbus interface 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 additional 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 returned 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 that associates 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-5CE 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 specified by 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 is displayed using the `printenv` command, and is modified using either the `setenv`, or the `set-default` command provided by OpenBoot.

`vme-sysfail-clear?` 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. The operating system which is loaded has to clear it. (default: `true`)

`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 set up the main memory. (default: `true`)

`level-15-intr-ena?` controls whether the capabilities to generate a non-maskable interrupt by the watchdog timer, by pressing the abort switch and by the assertion of the VMEbus signals SYSFAIL* and ACFAIL*, are enabled or disabled. In the case that the configuration parameter is `true`, OpenBoot will enable the capability to generate a non-maskable interrupt by the sources mentioned above. This is done after the on-

board memory has been probed and before the SBus is being probed for additional devices. When the configuration parameter is `false`, the capability to generate a non-maskable interrupt by the sources listed above is disabled. (default: `true`)

`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`. If the flag is `false`, the watchdog timer is not started, but the watchdog timer registers are initialised according to the configuration parameter `wd-timeout`. (default: `false`)

`vme-intr1` controls whether the VMEbus interrupt request level 1 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 1 is not enabled. In the case that the value is one (1), the corresponding interrupt handler is activated and the VMEbus interrupt request level 1 is enabled. The value one specifies that the S4 asserts its SBus interrupt request line 1 when a VMEbus interrupt request level 1 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-intr2` controls whether the VMEbus interrupt request level 2 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 2 is not enabled. In the case that the value is two (2), the corresponding interrupt handler is activated and the VMEbus interrupt request level 2 is enabled. The value one specifies that the S4 asserts its SBus interrupt request line 2 when a VMEbus interrupt request level 2 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-intr3` controls whether the VMEbus interrupt request level 3 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 3 is not enabled. In the case that the value is three (3), the corresponding interrupt handler is activated and the VMEbus interrupt request level 3 is enabled. The value one specifies that the S4 asserts its SBus interrupt request line 3 when a VMEbus interrupt request level 3 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-intr4` controls whether the VMEbus interrupt request level 4 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 4 is not enabled. In the case that the value is four (4), the corresponding interrupt handler is activated and the VMEbus interrupt request level 4 is enabled. The value one specifies that the S4 asserts its SBus interrupt request line 4 when a VMEbus interrupt request level 4 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-intr5` controls whether the VMEbus interrupt request level 5 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 5 is not enabled. In the case that the value is five (5), the corresponding interrupt handler is activated and the VMEbus interrupt request level 5 is enabled. The value one specifies that the S4 asserts its

SBus interrupt request line 5 when a VMEbus interrupt request level 5 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-intr6` controls whether the VMEbus interrupt request level 6 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 6 is not enabled. In the case that the value is six (6), the corresponding interrupt handler is activated and the VMEbus interrupt request level 6 is enabled. The value one specifies that the S4 asserts its SBus interrupt request line 6 when a VMEbus interrupt request level 6 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-intr7` controls whether the VMEbus interrupt request level 7 has to be enabled. When this flag is 255, then the VMEbus interrupt request level 7 is not enabled. In the case that the value is seven (7), the corresponding interrupt handler is activated and the VMEbus interrupt request level 7 is enabled. The value one specifies that the S4 asserts its SBus interrupt request line 7 when a VMEbus interrupt request level 7 occurs. Only the least significant three bits of this value are considered! (default: 255_{10})

`vme-control` contains the 8-bit data to be stored in the VMEbus interface's VMEbus Control Register when the VMEbus interface is initialised. (default: 80_{10})

`vme-a32map` contains the 8-bit data to be stored in the VMEbus interface's VMEbus A32 Map Register when the VMEbus interface is initialised. (default: 1_{10})

`vme-slavemap` contains the 8-bit data to be stored in the S4's Slave Map Register when the VMEbus interface is initialised. (default: 0_{10})

`vme-rerun` contains the number of VMEbus rerun cycles to be selected. Only the least significant five bits of the configuration parameter are considered. (default: 0_{10})

`vme-intena` contains the 8-bit data to be stored in the S4's Interrupt Enable Register when the VMEbus interface is initialised. (default: fe_{10})

`vme-mailintr` contains the number of the interrupt level that is asserted when the mailbox is being access from the VMEbus. The value of this configuration parameter may range from zero through seven. The values one through seven correspond to the seven SBus interrupt request levels 1 to 7. The value zero indicates generation of either a reset or a non-maskable interrupt — depending on the state of a hardware switch on the SPARC CPU-5CE. Please see "Default Switch Settings" on page 15 for information about the switches.

`vme-mailbox` contains the 8-bit data to be stored in the S4's Mailbox Control Register when the VMEbus interface is initialised. (default: 0_{10})

`vme-buslock` contains the 8-bit data to be stored in the S4's Bus Locker Register when the VMEbus interface is initialised. (default: 0_{10})

`vme-fair-req?` specifies whether the VMEbus requester operates in the *fair* mode when requesting the VMEbus. When the value of the configuration parameter is `true`, the VMEbus requester operates in the *fair* mode. Otherwise — the value of the configuration parameter is `false` — the requester does **not** operate in the fair mode. (default: `true`)

5.2.11 Mailboxes

The commands described below are available to control the mailbox as well as to retrieve information about the state of the mailbox.

`vme-mailbox@` (— *byte*) returns the contents — an 8-bit data — of the S4's Mail Box Register.

`vme-mailbox!` (*byte* —) stores the 8-bit data *byte* in the S4's Mail Box Register.

`vme-mailintr@` (— *mapping*) returns the value *mapping* that indicates the output pin being asserted when the mailbox is accessed from the VMEbus. The value of *mapping* may be one of the values in the range of zero through seven. Each value specifies one of the eight S4 interrupt request lines.
The table below contains a list of all valid mappings and the associated output pin — interrupt request line.

`vme-mailintr!` (*mapping* —) selects the ouput — specified by the parameter *mapping* — which is asserted by the S4 when the mailbox is accessed from the VMEbus. The value of *mapping* may be one of the values in the range of zero through seven. Each value specifies one of the eight S4 interrupt request lines.
The table below contains a list of all valid mappings and the associated output pin — interrupt request line.

mapping	**Interrupt Generated by S4**
0	MB_IRQ* (connected with *non-maskable* interrupt)
1	B_IRQ1* (connected with SBus IRQ1)
2	B_IRQ2* (connected with SBus IRQ2)
3	B_IRQ3* (connected with SBus IRQ3)
4	B_IRQ4* (connected with SBus IRQ4)
5	B_IRQ5* (connected with SBus IRQ5)
6	B_IRQ6* (connected with SBus IRQ6)
7	B_IRQ7* (connected with SBus IRQ7)

Table 60: Interrupt Mapping

5.3 System Configuration

5.3.1 Watchdog Timer

The commands described below are available to control the watchdog timer as well as to retrieve information about the state of the watchdog timer.

`wd-ena` (—) enables and starts the watchdog timer.

`wd-enable!` (*true* | *false*) starts or stops the watchdog timer. When the value *true* is passed to the command the watchdog timer is started. Otherwise — the value *false* is passed to the command — the watchdog timer is stopped.
Once enabled, the watchdog timer on the SPARC CPU-5CE cannot be stopped.

`wd-nmi-ena` (—) allows an interrupt to be generated when half of the watchdog time has expired.

`wd-nmi-dis` (—) disables the interrupt's ability to be generated 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-ip?` (— *true* | *false*) checks whether an interrupt is pending due to an interrupt generated by the watchdog timer when half of the watchdog time has expired. The value *true* is returned when the interrupt is pending; otherwise the value *false* is returned.

`wd-restart` (—) resets the watchdog timer and starts a new time count.

`reset-wd` (—) resets the watchdog timer and starts a new time count.

The watchdog timer is started by the commands listed below:

```
ok wd-nmi-ena
ok wd-ena
ok
```

In the example above, 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. 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? off
ok
```

5.3.2 Abort Switch

The commands described below are available to control the abort switch as well as to retrieve information about the state of the abort switch.

`abort-nmi-ena` (—) allows an interrupt to be generated when the abort switch is pressed.

`abort-nmi-dis` (—) disables the interrupt's ability to be generated when the abort switch is being pressed.

`abort-ip?` (— *true* | *false*) checks whether an interrupt is pending because the abort switch has been pressed. The value *true* is returned when the interrupt is pending; otherwise, the value *false* is returned.

`abort-irq-pending?` (— *true* | *false*) checks whether an interrupt is pending because the abort switch has been pressed. The value *true* is returned when the interrupt is pending; otherwise the vallue *false* is returned.

`abort-nmi-clear` (—) clears a pending interrupt caused by pressing the abort switch.

`reset-abort-irq` (—) clears a pending interrupt caused by pressing the abort switch.

5.3.3 Seven Segment LED Display and Rotary Switch

The commands described below are available to control the seven segment LED display as well as to retrieve information about the state of the rotary switch.

`diag-led!` (*byte* —) stores the data *byte* passed to the command in the register used to control the seven segment display.

`>7-seg-code` (*u* — *7-seg-code*) converts the value *u* to its corresponding seven segment code *7-seg-code*. Only the least significant four bits of the value *u* are considered.

`led!` (*colour* —) controls the user LED identified. The parameter *colour* defines the colour of the LED. The following constants are defined to specify the *colour*: **`black`**, **`green`**, **`red`**, and **`yellow`**. When the colour **`black`** is specified the LED is turned off.

The following example allows the user LED to shine red:

```
ok red led!
ok
```

`led-on` (—) turns the user LED on. The user LED is shining yellow.
`led-off` (—) turns the user LED off.

`led?` (— *true* | *false*) determines the state of the LED and returns either *true* or *false* to indicate if the LED is turned on or off. When the LED is turned on, then the value *true* is returned; otherwise the value *false* is returned.

`toggle-led` (—) determines the state of the user LED and turns the LED on or off. The LED is turned on when it was turned off before, and vice versa.

`rotary-switch@` (— *byte*) returns the current state of the rotary switch. The value of *byte* may be one of the values in the range of zero through 15. The value zero corresponds to the position 0 of the rotary switch, the value one corresponds to position 1, and so forth.

5.3.4 Miscellanea

`flash-rdy?` (— *true* | *false*) determines the status of the internal Write State Machine of the USER flash memory devices. When the USER flash memories are ready for programming or erasure, the value *true* is returned. In the case that the USER flash memory devices are not ready for additional commands, the value *false* is returned.

5.4 Flash Memory Support

5.4.1 Flash Memory Programming

The commands listed below are available to access and program the flash memories available on the SPARC CPU-5CE.

`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`**. They are displayed after *turning on* this variable by **`flash-messages 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 access 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-5CE 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 programming 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 access 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 the selected 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 address beginning at 100000_{16} .

5.4.2 Flash Memory Device

The device tree of OpenBoot for the SPARC CPU-5CE contains a device node associated with the USER flash memories. Thus, it is possible to load an executable image stored in the available USER flash into memory and start such an executable.

The device is called "`flash-memory@0,71300000`" and is attached to the device node "`/obio`". The device alias **`flash`** is available as an abbreviated representation of the flash memory device path.
The vocabulary of the flash memory device includes the standard commands recommended for a *byte* device. The words of this vocabulary are only available when the flash memory device has been selected as shown below:

```
ok cd flash
ok words
close          open          selftest      reset      load
write-blocks   read-blocks   seek          write      read
max-transfer   block-size
ok selftest .
0
ok device-end
ok
```

The example listed above, selects the flash memory device and makes it the current node. The word **`words`** displays the names of the *methods* of the VMEbus device. The third command calls the method **`selftest`** and the value returned by this method is displayed. The last command *unselects* the current device node, leaving no node selected.
When the command **`select-dev`** is used to select the flash memory device, the NVRAM configuration parameters **`bootflash-#megs`** and **`bootflash-#devices`** have to be set properly before the device can be selected.

The NVRAM configuration parameters listed below are available to control the loading of an image from the USER flash memory. The current state of these configuration parameters is displayed using the `printenv` command. It is modified using either the `setenv` or the `set-default` command provided by OpenBoot.

`bootflash-#megs` specifies the amount of available USER flash memory in megabyte. (default: 0 Megabyte)

`bootflash-#devices` specifies the number of available USER flash memory devices. (default: no devices)

`bootflash-load-base` specifies the address where the data loaded from the available flash memory are stored when the **`load`** or **`boot`** command, provided by OpenBoot,

is used to load an image from the flash memory.
When this parameter is set to -1 — which is the parameter's default value — then the image loaded from the flash memory is stored beginning at the address *addr.* But when the value of the configuration parameter differs from -1, then the image loaded from the flash memory is stored beginning at the address specified by the configuration parameter **`bootflash-load-base`**. The same address is stored in the variable **`load-base`** maintained by OpenBoot.

The methods listed below are available in the vocabulary of the flash memory device:

`open` (— *true*) prepares the package for subsequent use. The value `true` is returned when the device has been opened successfully; otherwise, the value `false` is returned. Usually, the fail state is indicated when the NVRAM configuration parameters **`bootflash-#megs`** and **`bootflash-#devices`** are not consistent.

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

`reset` (—) puts the flash memory device into *quiet* state.

`selftest` (— *error-number*) always returns the value zero.

`read` (*addr lenth — actual*) reads at most *length* bytes from the flash memory device into memory beginning at address *addr.* If *actual* is zero or negative, the read failed. The value of *length* may not always be a multiple of the device's normal block size.

`write` (*addr length — actual*) discards the information passed to the command and always returns zero to indicate that the device does not support this function.

`seek` (*offset file# — error?*) seek to byte *offset* within the file identified by *file#.* The flash memory device package maintains an internal position counter that is updated whenever a method to read data from, or to store data in, the flash memories is called. If *offset* and *file#* are both zero, then the internal position counter is reset to offset zero, otherwise the value of *offset* is assigned to the internal position counter, and a subsequent access to the flash memories starts at the offset selected.
Because the flash memory device does not support any file system, the parameter *file#* is ignored, except in the case mentioned above.
When the seek succeeded the value of *error?* is zero, otherwise the value -1 is returnrned to indicate the fail state.

`read-blocks` (*addr block# #blocks — #read*) reads the number of blocks identified by *#blocks* of length *block-size* bytes, each from the device beginning at the device block *block#*, into memory at address *addr.* It returns the number of blocks actually read (*#read*).

`write-blocks` (*addr block# #blocks — #written*) discards the information passed to the command and always returns zero to indicate that the device does not support this function.

`block-size` (*— bytes*) returns the size in bytes *bytes* of a block which is always the size of the flash memory programming window.

`max-transfer` (*— bytes*) returns the size in bytes *bytes* of the largest single transfer the device can perform. The command returns a multiple of `block-size`.

`load` (*addr — length*) reads a stand-alone program from the flash memory beginning at offset 0_{16} and stores it beginning at address *addr*. It returns the number of bytes *length* read from the flash memory.
This method considers the state of the NVRAM configuration parameter **`boot-flash-load-base`**: when this parameter is set to -1 — which is the parameter's default value — then the image loaded from the flash memory is stored beginning at the address *addr*. But when the value of the configuration parameter differs from -1, then the image loaded from the flash memory is stored beginning at the address specified by the configuration parameter **`bootflash-load-base`**. And the same address is stored in the variable **`load-base`** maintained by OpenBoot.

5.4.3 Loading and Executing Programs from USER Flash Memory

Besides the ability to load and execute an executable image from disk, or via network or other components, the OpenBoot for the SPARC CPU-5CE provides a convenient way to load and execute an executable image from the available USER flash memory. The executable image to be loaded has to be either a **binary** image (a.out format), a **FORTH** program, or a **FCode** program.
As mentioned at the beginning of this section, the device alias **`flash`** is available as an abbreviated representation of the flash memory device. The command listed below is used to explicitly load and execute an image from the flash memory:

```
ok boot flash
```

The following NVRAM configuration parameters can be modified to determine whether or not the system will load an executable image automatically after a power-up cycle or system reset:

- `auto-boot?`
- `boot-device`

Assuming, that the SPARC CPU-5CE is equipped with one USER flash memory device, the

size of which is 1Mbyte, then the following commands have to be used to load and execute an image from the flash memory automatically after a power-up cycle or system reset:

```
ok setenv bootflash-#devices 1
bootflash-#devices =  1
ok setenv bootflash-#megs 1
bootflash-#megs =     1
ok setenv boot-device flash
boot-device =         flash
ok setenv auto-boot? true
auto-boot? =          true
ok reset
```

5.4.4 Controlling the Flash Memory Interface

The commands listed below are available to control the flash memory interface. These commands are used to make a specific flash memory device available in the flash memory programming window and to control the flash memory programming voltage.

`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 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 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).

`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).

5.5 On-board Interrupts

Besides the interrupt handlers already available in the standard OpenBoot, the OpenBoot of the SPARC CPU-5CE provides additional handlers that deal with the interrupts generated by the 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.

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 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 an 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 of 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 serving 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 level 2 is generated upon the occurrence of a VMEbus interrupt 2; and so forth.

5.5.2 SYSFAIL Interrupt

OpenBoot for the SPARC CPU-5CE already includes an interrupt handler to serve the non-maskable interrupt generated upon the assertion and negation of the SYSFAIL* signal. This handler does not need 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-irq-ena
ok
```

which enables the generation of a non-maskable interrupt whenever the SYSFAIL* signal is asserted and negated.

When a non-maskable interrupt occurs 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 state of the variable is obtained by

```
ok sysfail-asserted? ?
0
ok
```

And the variable is cleared — set to zero — by

```
ok sysfail-asserted? off
ok
```

5.5.3 ACFAIL Interrupt

OpenBoot for the SPARC CPU-5CE already includes an interrupt handler to serve the non-maskable interrupt generated upon the assertion of the ACFAIL* signal. This handler does not need to be installed because it is already installed 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-irq-ena
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-5CE already includes an interrupt handler to serve the non-maskable interrupt generated by pressing the front panel abort switch. This handler does not need 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 appears immediately.

5.5.5 Watchdog Timer Interrupt

OpenBoot for the SPARC CPU-5CE 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 does not need 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 — and has to be enabled by

```
ok wd-nmi-ena
ok wd-ena
ok
```

In this example, 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? off
ok
```

5.6 Further Commands

The command listed below is available to provide miscellaneous services:

`not-cachable` (*vaddr size* —) disables cachability of an address range identified by its **virtual** base address *addr* and its *size*.

5.7 Frequently Asked Questions

Question: Why does the command **abort-nmi-ena** enable the watchdog timer's ability to generate a non-maskable interrupt even when it was previously disabled by using the **wd-nmi-dis** command?

Answer: On the SPARC CPU-5CE the watchdog timer, pressing the abort switch, the assertion of the VMEbus signals SYSFAIL* and ACFAIL* may generate a non-maskable interrupt (level 15 interrupt). But only one bit — the IRQ15_ENA bit (bit 2) in the General Purpose Register 2 — is available to control whether a non maskable interrupt, generated by one of the sources mentioned above, is passed to the processor.
Therefore, the commands **vme-sysfail-assert-irq-ena**, **vme-acfail-assert-irq-ena**, **wd-nmi-ena**, and **abort-nmi-ena** enable the capability to generate an non-maskable interrupt by all sources listed previously.
On the other hand, the the commands **vme-sysfail-assert-irq-dis**, **vme-acfail-assert-irq-dis**, **wd-nmi-dis**, and **abort-nmi-dis** disable the capability to generate an non-maskable interrupt by all sources listed above.

5.8 BusNet Support

In general, the OpenBoot should provide the capability to load and execute (*boot*) an executable image via the VMEbus backplane using the BusNet protocol.

5.8.1 Limitations

Due to the fact that OpenBoot is a simple *booter*, rather than an operating system, the limitations listed below apply to the BusNet protocol implementation:

- The OpenBoot support for the BusNet protocol only allows a participant to operate as a **slave**.
- The network management services are currently **not** supported. Every received packet containing such a request is refused by the BusNet driver.
- The OpenBoot provides only *single-buffering* mode which means that only one buffer is provided for every participant.
- In general, OpenBoot does not use any interrupt mechanism while loading an image from the *boot* device. Therefore, OpenBoot will not enable a mailbox—available on the machine—even if the NVRAM configuration parameters allow the use of a mailbox.

5.8.2 Loading Programs

The OpenBoot provides several methods for loading and executing a program on the machine. These methods load a file from a remote machine across the communication channel into memory, and support execution of FORTH-, FCode- and binary executable programs.

An executable program is loaded across the VMEbus using the BusNet protocol with the following two command provided by OpenBoot:

```
$ boot device-specifier argument
```

or

```
$ load device-specifier argument
```

The parameter *device-specifier* represents the name — full path name or alias — of the BusNet boot device. The OpenBoot provides the following device alias definitions associated with this

device:

Alias	Boot Path	Description
busnet	/iommu/VME/BusNet:tftp	TFTP is used to load program
busnet-tftp	/iommu/VME/BusNet:tftp	TFTP is used to load program
busnet-raw	/iommu/VME/BusNet:raw	pure binary data is loaded (*raw* device)

NOTE: Many commands — like `boot` and `test` — that require a device name, accept either a full device path name or a device alias. In this documentation, the term *device-specifier* is used to indicate that either a device path or a device alias is acceptable for such commands

5.8.3 The BusNet Device

The BusNet device is a *packet* oriented device capable of sending and receiving packets. The BusNet device available in OpenBoot is called BusNet and is attached to the device path /iommu/VME.

5.8.3.1 Device Properties

Device properties identify the characteristics of the package and its associated physical device. The BusNet device is characterized by the properties described below—these properties are static:

name property identifies the package. The BusNet package is identified by the string `busnet`.

device_type declares the type of the device. As the BusNet device is intended for booting across a *network* (VMEbus), its device type is declared as `network`.

address-bits specifies the number of address bits necessary to address this device on its network. Typically, the BusNet address consists of 32 bits, but only the least significant five bits are important. All remaining bits must be cleared (0). Therefore, the property address-bits is set to 32.
The property's size is 32 bits (integer).

reg property describes the VMEbus address ranges which are accessible by the BusNet device driver. The information given by this property is crucial for the operating of the operating system's own BusNet device driver. The register property is declared as follows:

```
h# 0000.0000 vmea16d32 h# 0001.0000     (VMEbus A16 space)
h# 0000.0000 vmea24d32 h# 00ff.0000     (VMEbus A24 space)
h# 0000.0000 vmea32d32 h# ff00.0000     (VMEbus A32 space)
```

The properties listed below are created dynamically whenever the device is opened for subsequent accesses:

bn-packet-size specifies the size of a BusNet packet—including the BusNet packet header.
The value of this property depends on the value of the NVRAM configuration parameter `bn-packet-size`. When the value of the configuration parameter is below the minimum of 2048 bytes, the property's value is set to 2048. In the case that the value of the configuration parameter is not a multiple of 64 bytes, the value of the property is *downsized* to the next 64 byte boundary.
The property's size is 32 bits (integer).

max-frame-size indicates the maximum allowable size of a packet (in bytes). This property is created dynamically when the BusNet device is opened and depends on the property `bn-packet-size`.
The property's size is 32 bits (integer).

bn-master-offset specifies the physical address of the participant designated as master.
The property's size is 32 bits (integer).

bn-master-space specifies the space in which the master's BusNet region is accessible.
The property's size is 32 bits (integer).

bn-master-access specifies the access mode of the master's BusNet region.
The property's size is 32 bits (integer).

bn-p-offset specifies the physical address of the participant's own BusNet region.
The property's size is 32 bits (integer).

bn-p-space specifies the space in which the participant's own BusNet region is accessible.
The property's size is 32 bits (integer).

bn-p-access specifies the access mode of the participant's own BusNet region.
The property's size is 32 bits (integer).

bn-logical-addr specifies the logical address assigned to the participant.
The property's size is 32 bits (integer).

bn-p-mbox-offset specifies the physical address of the participant's mailbox.
The property's size is 32 bits (integer).

bn-p-mbox-space specifies the space in which the mailbox of the participant is accessible.
The property's size is 32 bits (integer).

bn-p-mbox-access specifies the access mode of the participant's mailbox.
The property's size is 32 bits (integer).

bn-p-mbox-intr specifies the interrupt generated when the participant's mailbox is being accessed from the bus.
The property's size is 32 bits (integer).

bn-p-mbox? specifies whether the participant provides a mailbox. When the value of this property is `true` then the participant provides a mailbox. Otherwise, the participant does not provide a mailbox.

5.8.3.2 Device Methods

The BusNet device intended for use by OpenBoot implements the methods described below.

`open` (— *ok?*) prepares the device for subsequent use. The value `true` is returned upon successful completion; otherwise, the value `false` is returned to indicate a failure. When `open` is called, the parent instance chain has already been opened, and this method may call its parent's methods.
Typically, the device builds up its BusNet region, makes this region available to the VMEbus address space, and tries to connect with the BusNet master for registering.

`close` (—) restores the device to its *not-in-use* state. Typically, it informs all known BusNet participants about its intention to withdraw from the protocol, and disables its VMEbus slave interface to prevent it from being accessed by other BusNet participants.

`reset` (—) puts the device into its *quiescent* state, and afterwards starts to register with the master again. In particular, the `reset` method executes the `close` and immediately afterwards the `open` method.

`selftest` (— *error#*) normally tests the package and returns an error number *error#* which identifies a specific failure. But the BusNet device provides this method only for completeness, and returns the value zero when the method is called. The value zero is returned to indicate that no failure has been detected.

`load` (*addr* — *length*) reads the default stand-alone program into memory starting at *addr* using the network booting protocol. The *length* parameter returned specifies the size in bytes of the image loaded.

`read` (*addr length — actual*) receives a network packet and stores at most the first *length* bytes in memory beginning at address *addr*. It returns the *actual* number of bytes received (not the number copied), or it returns zero if no packet is currently available.
The BusNet device driver copies only the data contained in the BusNet packet into memory and discards all information related to the BusNet protocol.

`write` (*addr length — actual*) transmits the network packet of size *length* stored in memory beginning at address *addr*, and returns the number of bytes actually transmitted, or zero if the packet has not been transmitted due to a failure.
The BusNet device driver copies the data into the data field of a BusNet packet and transmits the packet to the specified recipient.

`seek` (*poslow poshigh — -1*) operation is invalid and the method therefore always returns -1 to indicate the failure.

5.8.3.3 NVRAM Configuration Parameters

The OpenBoot provides the NVRAM configuration parameters as defined by the BusNet Protocol Specification 1.4.2. The NVRAM configuration parameters may be modified using the `set-default` or `setenv` commands provided by OpenBoot. The actual state of the NVRAM configuration parameters are displayed by the `printenv` command.

`bn-master-offset` specifies the physical address of the participant designated as master. The default value of this 32-bit configuration parameter is zero.

`bn-master-space` specifies the space in which the master's BusNet region is accessible. Typically, this configuration parameter identifies one of the address spaces available in the address range of the bus. The default value of this 32-bit configuration parameter is $3D_{16}$ (privileged *standard* address space).

`bn-master-access` specifies the access mode of the master’s BusNet region. The default value of this 32-bit configuration parameter is 32_{16} (D32, read/write, no LOCKed cycles are supported).

`bn-p-offset` specifies the physical address of the participant’s own BusNet region. The default value of this 32-bit configuration parameter is zero.

`bn-p-space` specifies the space in which the participant's own BusNet region is accessible. Typically, this configuration parameter identifies one of the address spaces available in the address range of the bus. The default value of this 32-bit configuration parameter is $3D_{16}$ (privileged *standard* address space).

`bn-p-access` specifies the access mode of the participant's own BusNet region. The default value of this 32-bit configuration parameter is 32_{16} (D32, read/write, no LOCKed cycles are supported).

`bn-logical-addr` specifies the logical address assigned to the participant. The value of this configuration parameter may be in the range zero through 31. The default value of this 32-bit configuration parameter is zero.

`bn-p-mbox-offset` specifies the physical address of the participant's mailbox. The default value of this 32-bit configuration parameter depends on the hardware capabilities of the specific machine.

`bn-p-mbox-space` specifies the space in which the participant's mailbox is accessible. Typically, this configuration parameter identifies one of the address spaces available in the address range of the bus.
The default value of this 32-bit configuration parameter depends on the hardware capabilities of the specific machine.

`bn-p-mbox-access` specifies the access mode of the participant's mailbox. The default value of this 32-bit configuration parameter depends on the hardware capabilities of the specific machine.

`bn-p-mbox-intr` specifies the interrupt generated when the participant's mailbox is being accessed from the bus. The default value of this 32-bit configuration parameter depends on the hardware capabilities of the specific machine.

`bn-p-mbox?` specifies whether the participant provides a mailbox. When this configuration parameter is `true` then the participant provides a mailbox. Otherwise, the participant does not provide a mailbox. The default value of this configuration parameter depends on the hardware capabilities of the specific machine.

`bn-packet-size` specifies the size of a BusNet packet. The minimum packet size allowed by the BusNet protocol is 2 Kbytes. The default value of this configuration parameter is 2 Kbytes. If set to another value it must be a multiple of 64 bytes.
The BusNet protocol does not permit participants to use different packet buffer sizes during intitialization.
The default value of this 32-bit configuration parameter is 2048_{10}.

A participant is designated as **master** when the following pairs of configuration parameters `bn-master-space`, `bn-p-space` and `bn-master-offset`, `bn-p-offset` are identical. When these configuration parameters are different, the participant is designated as **slave**. However, OpenBoot does **not** support the master operation of a participant.

NOTE: The default values of some described NVRAM configuration parameters may vary depending on the VMEbus interface of the particualar machine (S4, MVIC, FGA-5000), especially the parameters describing the mailbox of the participant.

The state of the NVRAM configuration parameters listed below are only considered when the Trivial File Transfer Protocol (TFTP) is used to load and execute an image across the network using the BusNet protocol:

`bn-arp?` specifies whether the BusNet driver should scrutinize all outgoing packets and verifies whether an Ethernet frame carries an ARP request.
When the flag is `true`, the BusNet driver checks whether an Ethernet frame contains an ARP request and, if so, it resolves the request and passes the response to the receiving part of the BusNet driver automatically. The Ethernet frame is **not** sent across the network.
The BusNet driver uses the contents of the NVRAM configuration parameters `bn-master-ip-addr`, `bn-p-ip-addr`, `bn-master-en-addr`, and `bn-p-en-addr` to build up the appropriate response.
In the case that the flag is `false`, it sends all Ethernet frames without any further verification across the network. (default: `false`)

`bn-rarp?` specifies whether the BusNet driver should scrutinize all outgoing packets and verifies whether an Ethernet frame carries an RARP request.
When the flag is `true`, the BusNet driver checks whether an Ethernet frame contains a RARP request and, if so, it resolves the request and passes the response to the receiving part of the BusNet driver automatically. The Ethernet frame is **not** send across the network.
The BusNet driver uses the contents of the NVRAM configuration parameters `bn-master-ip-addr`, `bn-p-ip-addr`, `bn-master-en-addr`, and `bn-p-en-addr` to build up the appropriate response.
In the case that the flag is `false`, it sends all Ethernet frames without any further verification across the network. (default: `false`)

`bn-master-ip-addr` specifies the Internet Protocol (IP) Address of the master. The default value of this 32-bit configuration parameter is zero (0). The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-master-ip-addr 0x83030001
```

In the example, the Internet address `131.3.0.1` (83030001_{16}) is assigned to the NVRAM configuration parameter.
This configuration parameter **must** be set when one of the two configuration parameters `bn-arp?` or `bn-rarp?` are set to `true`.

`bn-p-ip-addr` specifies the Internet Protocol (IP) Address of the participant. The default value of this 32-bit configuration parameter is zero (0). The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-p-ip-addr 0x83030002
```

In the example, the Internet address `131.3.0.2` (83030002_{16}) is assigned to the NVRAM configuration parameter.
This configuration parameter **must** be set when one of the two configuration parameters `bn-arp?` or `bn-rarp?` are set to `true`.

`bn-master-en-addr` specifies the Ethernet address of the master. The Ethernet address is represented by an ASCII string in the following format: `XX:XX:XX:XX:XX:XX`—where `XX` is a hexadecimal number. The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-master-en-addr 0:80:42:b:10:ac
```

This configuration parameter **must** be set when one of the configuration parameters `bn-arp?` and `bn-rarp?` are set to `true`.

`bn-p-en-addr` specifies the Ethernet address of the participant. The Ethernet address is represented by an ASCII string in the following format: `XX:XX:XX:XX:XX:XX`—where `XX` is a hexadecimal number. The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-p-en-addr 0:80:42:b:10:ad
```

This configuration parameter **must** be set when one of the configuration parameters `bn-arp?` and `bn-rarp?` are set to `true`.

5.8.4 Device Operation

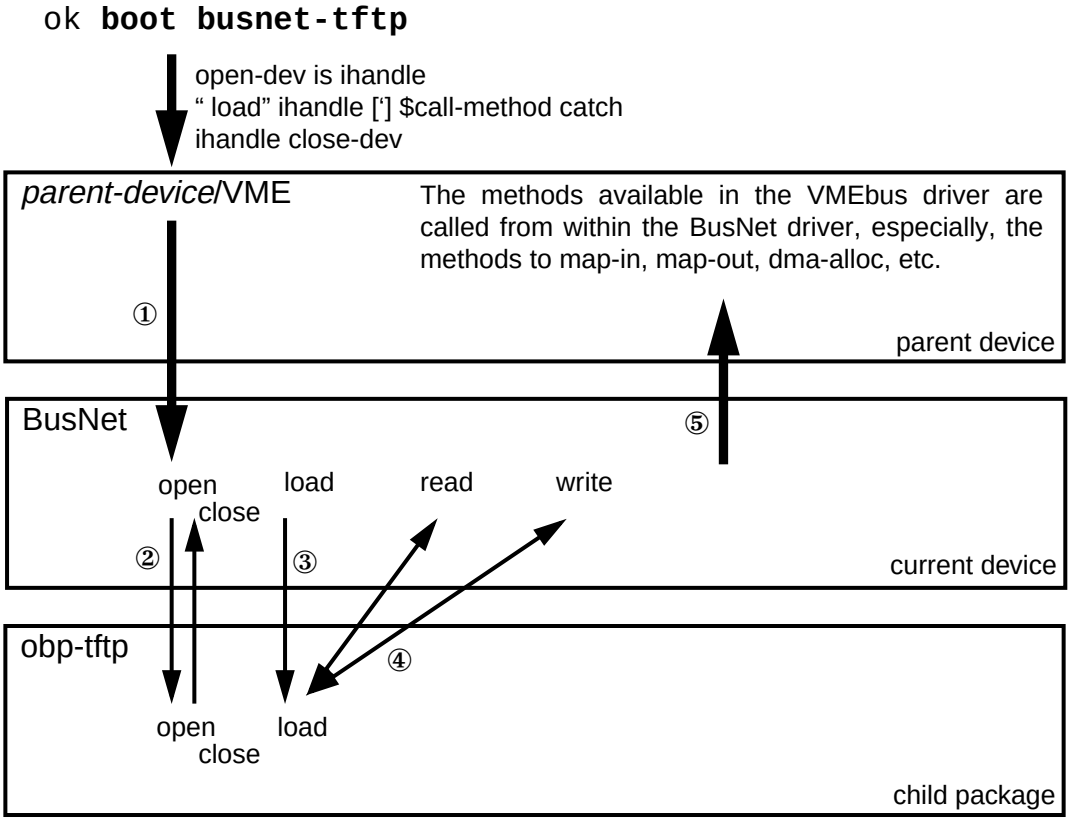
In general, OpenBoot provides the `boot`[1] command to load a program through a communication channel into memory. The *device-specifier* specifies the physical device that is attached to the communication channel. A program is loaded across the VMEbus — using the BusNet protocol — by

```
ok boot busnet
```

or

```
ok boot busnet-tftp
```

The device aliases `busnet` and `busnet-tftp` specify the BusNet device used to load the program. Both aliases contain the argument string `tftp` which informs the BusNet device to use the Trivial File Transfer Protocol TFTP to load the program, and the BusNet driver replaces the medium access layer MAC, which usually is Ethernet.



1. For more and detailed information about the `boot` command and the associated NVRAM configuration parameters refer to the *OpenBoot Command Reference*.

When the `boot` command is called — as shown in the figure above — OpenBoot tries to locate the specified device in its device tree and, opens each node of the device tree in turn, starting at the top until the BusNetBusNet device is reached ①. Assuming the TFTP protocol is used to load the program, the BusNet driver tries to open the package `obp-tftp` provided by OpenBoot and returns control to the `boot` command after the execution of its `open` method is complete ②.
In the next step, the boot command calls the BusNet driver's `load` method, which in turn calls the `load` method of the TFTP package to load the program ③.
During the time the program is loaded, the TFTP package controls operation and calls the methods `read` and `write` of its *parent* device ④— the BusNet device — to receive and transmit packets across the network. Once the program has been loaded, the control is passed back to the BusNet device, and the `boot` command. The latter calls the `close` method of the BusNet device which in turn calls the `close` method of the TFTP package. Finally, control is returned to the `boot` command.
The BusNet device calls the methods of its *parent* device, that is the VMEbus device. Typically, the BusNet driver calls the methods to make its BusNet region available to the VMEbus address space and to map this region to the processor's virtual address space ⑤.

5.8.5 How to Use BusNet

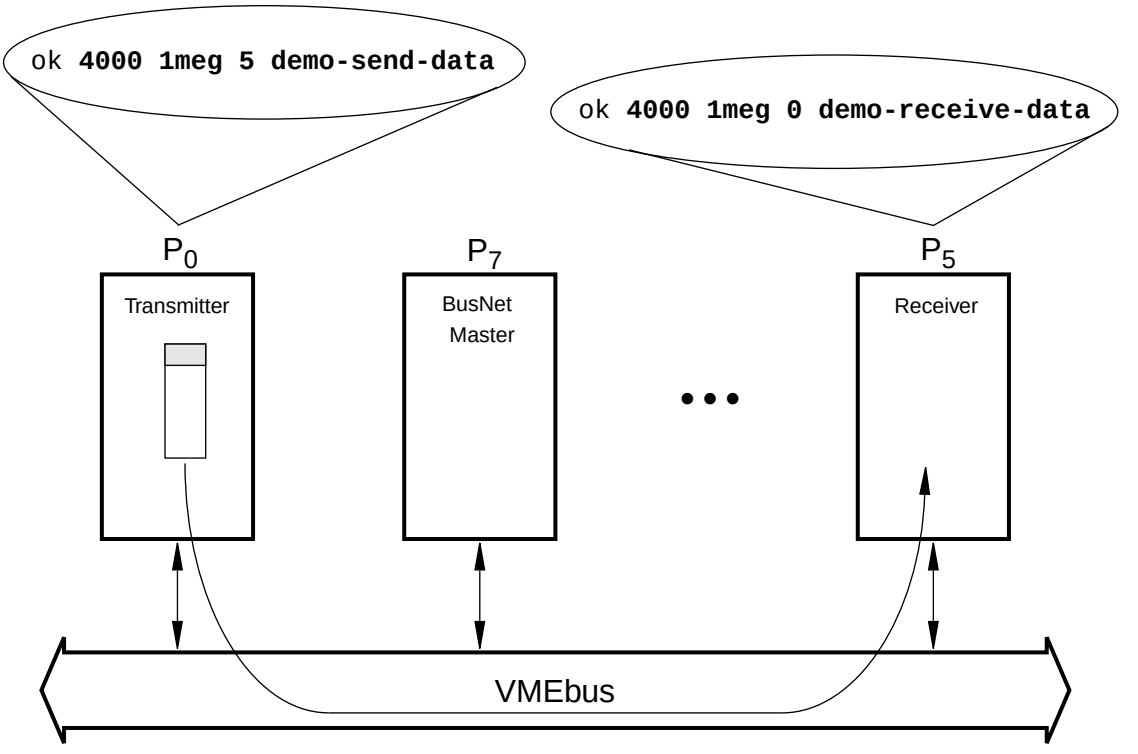
The `/busnet-demo` package is available in OpenBoot to demonstrate how to operate the BusNet driver in the **raw** mode. In this mode pure binary data are sent across the network from one BusNet participant to another participant. The following two definitions are available to initiate the transmission and receipt of data:

`demo-send-data` (*src-addr size dest-p#* —) sends the amount of data specified by *size* and stored beginning at the address *src-addr* to the participant identified by its logical BusNet address *dest-p#*.

`demo-receive-data` (*dest-addr size src-p#* —) receives as much data as specified by *size* from the participant identified by its logical BusNet address *dest-p#* and stores it beginning at the address *dest-addr.*

NOTE: When these commands are used to exchange data between two participants running OpenBoot, then a third participant must be available which provides BusNet master functionality. This is necessary because OpenBoot does not provide BusNet master functionality!

As shown in the figure below, three participants take part in communicating across the network using the BusNet protocol. The logical address of the participants are zero, seven and five. The participants P_0 and P_5 are executing OpenBoot, and the participant P_7 runs an operating system which is capable of providing BusNet master functionality—for example Solaris/SunOS, or VxWorks.



When a certain amount of data located in the on-board memory of the participant zero (P_0)—the transmitter—should be transfered to the participant five (P_5)—the receiver—then the following command must be used on the transmitter:

```
ok 4000 1meg 5 demo-send-data
```

This command initiates a transmission of 1 Mbyte of data located at address 4000_{16} in the transmitter's on-board memory to the receiver. To enable the receiver to receive the data the following command must be used:

```
ok 4000 1meg 0 demo-receive-data
```

This command initiates the receipt of data from the participant zero and stores the data beginning at address 4000_{16} in the receiver's on-board memory.

NOTE: To ensure proper operation of the data exchange, the size applied to the commands on the receiver and transmitter must be the same!

5.8.6 Using `bn-dload` to Load from the Backplane

The command `bn-dload` loads a file across the network and stores it at a specific address, as shown in the example below:

```
ok 4000 bn-dload filename
```

The *filename* must be relative to the server's root, and the contents of the file are stored beginning at address 4000_{16} within the on-board memory. The command `bn-dload` uses the Trivial File Transfer Protocol (TFTP) to load the file.

FORTH Programs. FORTH programs to be loaded with `bn-dload` must be ASCII files beginning with the two characters "\ " (backslash immediately followed by a space). To execute the loaded FORTH program, the `eval` command has to be used as follows:

```
ok 4000 file-size @ eval
```

The variable `file-size` contains the size of the loaded file.

FCode Programs. FCode programs to be loaded with `bn-dload` must be in the `a.out` format. To execute the loaded FORTH program, the `byte-load` command has to be used as follows:

```
ok 4000 1 byte-load
```

The command `byte-load` is used by OpenBoot to interpret FCode programs on expansion boards such as SBus cards. The second argument passed to this command— value one (1) in the example—specifies the separation between FCode byte in general. Because the `bn-dload` command loads the FCode into on-board memory, the spacing is one (1).

Binary Executables. Executable binary programs to be loaded with `bn-dload` must be in the `a.out` format. To execute the binary program, the `go` command has to be used as follows:

```
ok go
```

When the program should be started again, the commands listed below have to be used:

```
ok init-program go
```

5.8.7 Booting from a Solaris/SunOS BusNet Server

When Solaris/SunOS is loaded and executed from a Solaris/SunOS BusNet server, the boot command has to be used as follows:

```
ok boot busnet
```

In this case, OpenBoot will load the appropriate primary booter from the server using the Trivial File Transfer Protocol (TFTP), and start execution of the loaded image.
When the Solaris/SunOS is loaded and executed automatically after each system reset, the NVRAM configuration parameter `auto-boot?` must be set to `true`, and depending on the state of the configuration parameter `diag-switch?`, either `boot-device` or `diag-device` must be set. When the diagnostic mode is disabled, the configuration parameter `boot-device` must be set as follows:

```
ok setenv boot-device busnet
```

And in the case that the diagnostic mode is enabled, the configuration parameter `diag-device` must be set as described in the following:

```
ok setenv diag-device busnet
```

5.8.8 Booting from a VxWorks BusNet Server

Because VxWorks currently is not capable of resolving RARP requests, the NVRAM configuration parameters listed below must be set prior to loading an executable image.

`bn-rarp?` specifies whether the BusNet driver should scrutinize all outgoing packets and verifies whether an Ethernet frame carries an RARP request.
The flag **must** be set to `true`, to enable the BusNet driver to check whether an Ethernet frame contains a RARP request, and if so, it resolves the request and passes the response to the receiving part of the BusNet driver automatically. The Ethernet frame is **not** sent across the network.
The BusNet driver uses the contents of the NVRAM configuration parameters `bn-master-ip-addr`, `bn-p-ip-addr`, `bn-master-en-addr`, and `bn-p-en-addr` to build up the appropriate response.

`bn-master-ip-addr` specifies the Internet Protocol (IP) Address of the master. The default value of this 32-bit configuration parameter is zero (0). The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-master-ip-addr 0x83030001
```

In the example, the Internet address `131.3.0.1` (83030001_{16}) is assigned to the NVRAM configuration parameter.

`bn-p-ip-addr` specifies the Internet Protocol (IP) Address of the participant. The default value of this 32-bit configuration parameter is zero (0). The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-p-ip-addr 0x83030002
```

In the example, the Internet address `131.3.0.2` (83030002_{16}) is assigned to the NVRAM configuration parameter.

`bn-master-en-addr` specifies the Ethernet address of the master. The Ethernet address is represented by an ASCII string in the following format: `XX:XX:XX:XX:XX:XX`— where `XX` is a hexadecimal number. The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-master-en-addr 0:80:42:b:10:ac
```

`bn-p-en-addr` specifies the Ethernet address of the participant. The Ethernet address is represented by an ASCII string in the following format: `XX:XX:XX:XX:XX:XX`—where `XX` is a hexadecimal number. The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-p-en-addr 0:80:42:b:10:ad
```

Assuming the participant's Ethernet- and Internet address are `0:80:42:b:10:ad` and `131.3.0.2`, and the VxWorks server's Ethernet- and Internet address are `0:80:42:b:10:ac` and `131.3.0.1`, then the NVRAM configuration parameters listed above must be set as described below:

```
ok setenv bn-master-en-addr 0:80:42:b:10:ac
ok setenv bn-master-ip-addr 0x83030001
ok setenv bn-p-en-addr 0:80:42:b:10:ad
ok setenv bn-p-ip-addr 0x83030002
ok setenv bn-rarp? true
```

After these NVRAM configuration parameters have been set, the OpenBoot BusNet driver scrutinizes every outgoing packet that carries an Ethernet frame and verifies whether the Ethernet frame contains an RARP request. If so, the BusNet driver resolves the RARP request—using the information contained by the configuration parameters mentioned above—and passes the response internally to the receiving part of the BusNet driver. All other packets are sent across the network.
After this, the `boot`, `load` or `bn-dload` command can be used to load an executable image from the VxWorks server. In case of the first two commands, the name of the image being loaded is always the name of the primary booter (e.g. `83030002.SUN4M`).

5.8.9 Setting NVRAM Configuration Parameters

The CPU-5CE are equipped with the S4 VMEbus Interface Chip which provides a mailbox register located in the *short* address space (A16) of the VMEbus. To enable the mailbox the following NVRAM configuration parameters must be set in addition to the NVRAM configuration parameters listed in the table below:

`vme-ibox-addr` must be set to $Y00Z_{16}$ where Y is one of the values 0, 4, 8, or C_{16} and Z is one of the values 0, 2, 4, ..., C_{16}, or E_{16}.

`vme-ibox-ena?` must be set to `true`

`use-new-vme?` must be set to `true` to operate the VMEbus driver in the *new* mode. This configuration parameter has been added to provide a compatibility with the existing definitions that use the VMEbus device driver methods. To make the BusNet driver portable across a number of systems it was necessary to modify the methods.

The state of this configuration parameter controls the operation of the VMEbus device driver's methods `map-in`, `map-out`, `dma-map-in` and `dma-map-out`. Because these methods are used by the commands `set-vme-slave-def`, `set-vme-slave-enh` and `free-vme-mem` these commands are only executed properly when the configuration parameter is `false`.

NVRAM Configuration Parameter	Default Value	Description

`bn-master-offset`
`bn-master-space`
`bn-master-access`
	00000000_{16}
	$3D_{16}$
	32_{16}
		
		privileged *standard* (A24) address range
		read/write/D32

`bn-p-offset`
`bn-p-space`
`bn-p-access`
	00000000_{16}
	$3D_{16}$
	32_{16}
		
		privileged *standard* (A24) address range
		read/write/D32

`bn-p-mbox?`
`bn-p-mbox-offset`
`bn-p-mbox-space`
`bn-p-mbox-access`
`bn-p-mbox-intr`
	`true`
	0000_{16}
	$2D_{16}$
	10_{16}
	5
		mailbox available (S4 Mailbox)
		same as `vme-ibox-addr`
		privileged *short* (A16) address range
		read/D8
		SBus interrupt level 5 is asserted upon a mailbox

SECTION 6 SUN OPEN BOOT DOCUMENTATION

6. Insert your *OPEN BOOT 2.0 PROM MANUAL SET* here.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP