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 SPARC CPU-20VT* at our website: *Click HERE*



SPARC/CPU-20V,Ve,VT,VTe
Reference Guide

P/N 204223 Edition 8.0
May 2000

Force Computers GmbH
All Rights Reserved

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

Copyright by Force Computers

A SOLECTRON SUBSIDIARY



World Wide Web: www.forcecomputers.com
24-hour access to on-line manuals, driver updates, and application notes is provided via SMART, our SolutionsPLUS customer support program that provides current technical and services information.

Headquarters

The Americas

Force Computers Inc.
5799 Fontanoso Way
San Jose, CA 95138-1015
U.S.A.

Tel.: +1 (408) 369-6000
Fax: +1 (408) 371-3382
Email support@fci.com

Europe

Force Computers GmbH
Prof.-Messerschmitt-Str. 1
D-85579 Neubiberg/München
Germany

Tel.: +49 (89) 608 14-0
Fax: +49 (89) 609 77 93
Email support@force.de

Asia

Force Computers Japan KK
Shiba Daimon MF Building 4F
2-1-16 Shiba Daimon
Minato-ku, Tokyo 105-0012
Japan
Tel.: +81 (03) 3437 3948
Fax: +81 (03) 3437 3968
Email smiyagawa@fci.com

NOTE

The information in this document has been carefully checked and is believed to be entirely reliable. Force Computers makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors which may appear in this document. Force Computers reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance, or design.

Force Computers assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of Force Computers GmbH. Force Computers does not convey to the purchaser of the product described herein any license under the patent rights of Force Computers GmbH nor the rights of others. All product names as mentioned herein are the trademarks or registered trademarks of their respective companies.

Table of Contents

Using This Manual . . . . . **xi**

1 Safety Notes **1**

2 Introduction **5**

3 Installation **11**

3.1 Installation Prerequisites and Requirements **12**

3.1.1 Requirements 12

3.1.2 Terminal Connection 14

3.2 Mechanical Construction **16**

3.2.1 Mechanical Overview of a Completely Assembled SPARC/CPU-20VT 16

3.2.2 Two-, Three-, and Four-Slot SPARC/CPU-20VT Configurations 18

3.2.3 SPARC/CPU-20V 26

3.2.4 Upgrading the SPARC/CPU-20VT 27

3.2.5 Major Components 29

3.2.6 Location Diagrams 30

3.3 Switch Settings **33**

3.3.1 Base Board Switch Setting 33

3.3.2 SBus I/O Board Switch Setting 38

3.4 Front Panel and Connectors **40**

3.4.1 Serial I/O Port RS-232 and RS-422 Connector Pinout 44

3.4.2 Keyboard/Mouse Connector Pinout 46

3.4.3 Audio Connector Pinout (Factory Option) 46

3.4.4 VME P2 Connector Pinout of the Base Board 47

3.4.5 VME P2 Connector Pinout of the SBus I/O Board 48

3.4.6 IOBP-10 49

3.5 SCSI Configuration **56**

3.5.1 SCSI #1 Termination 56
3.5.2 SCSI #2 Termination 59
3.6 Ethernet Configuration 60
3.6.1 Ethernet Address and Host ID 62
3.7 Powering Up 64
3.8 OpenBoot Firmware 65
3.8.1 Boot the System 65
3.8.2 NVRAM Boot Parameters 68
3.8.3 Diagnostics 69
3.8.4 Display System Information 72
3.8.5 Reset the System 73
3.8.6 Using the FGA-5100 Enhanced Feature Set 73
3.8.7 OpenBoot Help 74
3.9 Booting Solaris 2.5 76
3.10 Patching Solaris 2.5 for 200 MHz MBus Modules 80
3.10.1 Obtaining Required Sun Kernel Jumbo Patch 80
3.10.2 Patching the Operating System prior to Installing 200 MHz hyperSPARC Modules 81
3.10.3 Installing Solaris 2.5 from a CD-ROM 81

4 Hardware Description 85
4.1 Address Map of the SPARC/CPU-20VT 89
4.2 MBus Modules 92
4.3 MBus-to-SBus Interface – M2S 93
4.4 Main Memory and Memory Controller RMC 95
4.5 VMEbus Interface – FGA-5x00 97
4.5.1 Adapting the FGA-5000 101
4.5.2 VMEbus SYSRESET Input and Output 102
4.6 Ethernet, SCSI, Parallel I/O – 2 MACIO Devices 104
4.7 SBus-to-EBus Controller – SEC 106
4.7.1 SEC Interrupt Controller 106
4.7.2 NMI (Nonmaskable Interrupt) 107
4.7.3 SEC Interrupt Mapping 107

204223 8 – 0 May 2000

4.8 Local I/O Devices 109
4.8.1 Address Map of Local I/O Devices 110
4.8.2 Boot PROM 111
4.8.3 User Flash 112
4.8.4 Programming the Boot PROM and the User Flash 112
4.8.5 RTC / NVRAM – MK48T18 116
4.8.6 Floppy Disk Interface – FDC 117
4.8.7 Serial I/O Ports A, B, C, and D – SCC 118
4.8.8 Audio Port – Am79C30A (Factory Option) 119
4.8.9 Keyboard and Mouse Port – SCC 120
4.8.10 LCA for General Board Control 121
4.8.11 LCA for Temperature Sensor Control 122
4.9 Front-Panel and Switch Related Features and Registers 124
4.9.1 User LED 1 and 2 Control Registers 125
4.9.2 Seven Segment LED Display 126
4.9.3 Rotary Switch 127
4.9.4 Temperature Sensors 128
4.9.5 Switch Status Registers 132

5 Force OpenBoot Enhancements 135
5.1 Controlling the VMEbus Master and Slave Interface 136
5.1.1 VMEbus Master Interface 137
5.1.2 VMEbus Slave Interface 140
5.2 VMEbus Interface 142
5.2.1 FGA-5x00 Register Accesses 143
5.2.2 VMEbus Interrupt Mapper 152
5.2.3 VMEbus Interrupt Handler and Interrupter 155
5.2.4 VMEbus Arbiter 157
5.2.5 VMEbus Requester 158
5.2.6 VMEbus Status Signals 159
5.2.7 VMEbus Master Interface 159
5.2.8 VMEbus Slave Interface 166
5.2.9 VMEbus Device Node 170

5.2.10 VMEbus NVRAM Configuration Parameters 172
5.2.11 DMA Controller Support 185
5.2.12 Mailboxes and Semaphores 189
5.2.13 Force Message Broadcast 190
5.3 Standard Initialisation of the VMEbus Interface 192
5.4 System Configuration 194
5.4.1 System Configuration Register Accesses 194
5.4.2 Watchdog Timer 196
5.4.3 Abort Switch 198
5.4.4 LEDs, Seven Segment Display and Rotary Switch 198
5.4.5 Reset 200
5.4.6 ID PROM 201
5.5 Flash Memory Support 202
5.5.1 Flash Memory Programming 202
5.5.2 Flash Memory Device 204
5.5.3 Loading and Executing Programs from USER Flash Memory 206
5.5.4 Controlling the Flash Memory Interface 207
5.6 On-board Interrupts 208
5.6.1 VMEbus Interrupts 208
5.6.2 SYSFAIL Interrupt 209
5.6.3 ACFAIL Interrupt 210
5.6.4 ABORT Interrupt 211
5.6.5 Watchdog Timer Interrupt 211
5.7 Viewing the Switch Status and Controlling the Temperature Sensors 212
5.8 BusNet Support 214
5.8.1 Loading Programs 214
5.8.2 The BusNet Device 215
5.8.2.1 Device Properties 215
5.8.2.2 Device Methods 217
5.8.2.3 NVRAM Configuration Parameters 218
5.8.3 Device Operation 221
5.8.4 How to Use BusNet 223
5.8.5 Using `bn-dload` to Load from the Backplane 225

204223 8 — 0 May 2000

5.8.6 Booting from a Solaris/SunOS BusNet Server 226
5.8.7 Booting from a VxWorks BusNet Server 226
5.8.8 Setting NVRAM Configuration Parameters 228
5.9 Additional Serial Ports 229
5.9.1 NVRAM Configuration Parameters 229
5.9.2 Selecting Input and Output Device Options 231

Product Error Report

204223 8 – 0 May 2000

List of Tables and Figures

	Page	Tab./Fig.
History of Manual Publication	xiii	Tab. a
Fonts, Notations and Conventions	xiv	Tab. b
Specifications of the SPARC/CPU-20VT	6	Tab. 1
Product Nomenclature	7	Tab. 2
Excerpt from the Data Sheet's Ordering Information	8	Tab. 3
Cross Section of a Completely Assembled SPARC/CPU-20VT	16	Fig. 1
Components and Connectors of the SPARC/CPU-20VT (Schematic)	17	Fig. 2
Sample Four-Slot Configuration	19	Fig. 3
Sample Three-Slot Configuration	19	Fig. 4
Sample Two-Slot Configuration	19	Fig. 5
SBus I/O Board: Boot PROM and RTC/NVRAM Device Locations	24	Fig. 6
Base Board: Boot PROM and RTC/NVRAM Device Locations	25	Fig. 7
Qualified Memory Module Configurations (All Data in MByte)	27	Tab. 4
Location Diagram of the Base Board (Schematic)	30	Fig. 8
Location Diagram of the SBus I/O Board (Schematic)	31	Fig. 9
Location Diagr. of SBus I/O Board with Audio Factory Opt. (Schem.)	32	Fig. 10
Base Board Switch Settings	34	Tab. 5
SBus I/O Board Switch Settings	38	Tab. 6
Front Panel Features	40	Tab. 7
SPARC/CPU-20VT Connectors	43	Tab. 8
RS-232: Serial A+B and C+D Front-Panel Connector Pinout	44	Fig. 11
RS-422: Serial A+B and C+D Front-Panel Connector Pinout	45	Fig. 12
Serial Ports Signal Naming	45	Tab. 9
Keyboard/Mouse Connector Pinout	46	Tab. 10
Audio Connector Pinout	47	Tab. 11
Base Board P2 Conn. Pinout with Serial I/O Config. for RS-232	47	Fig. 13
SBus I/O Board P2 Conn. Pinout with Serial I/O Config. for RS-232	48	Fig. 14
IOBP-10 (Schematic)	49	Fig. 15
IOBP-10 P1 Pinout	51	Tab. 12
IOBP-10 P2 Pinout (SCSI, Standard 50-pin Flat Cable Conn.)	52	Tab. 13
IOBP-10 P3 Pinout (Floppy, Standard 34-pin Flat Cable Conn.)	53	Tab. 14
IOBP-10 P4 Pinout (Centr., Stand. 40-pin Flat Cable Conn.)	53	Tab. 15
IOBP-10 P5 Pinout (Serial / Audio)	54	Tab. 16
IOBP-10 P6 Pinout (Ethernet, Standard 15-pin Micro D-Sub Conn.)	55	Tab. 17
Ethernet #1 Configuration via the Base-Board Switch Matrix	61	Fig. 16
Ethernet #2 Configuration via the SBus-I/O-Board Switch Matrix	62	Fig. 17
Ethernet Address	63	Fig. 18
Host ID	63	Fig. 19
Device Alias Definitions	66	Tab. 18

	Page	Tab./Fig.
Setting Configuration Parameters	68	Tab. 19
Diagnostic Routines	69	Tab. 20
Commands to Display System Information	72	Tab. 21
Block Diagram of the SPARC/CPU-20VT	88	Fig. 20
FGA-5100^{E}: SPARC/CPU-20VT Physical Address Map	89	Tab. 22
FGA-5000, FGA-5100^{C}: SPARC/CPU-20VT Physical Address Map	91	Tab. 23
Configuration Registers for SBus Participants	94	Tab. 24
Locating MEM-20L 1, MEM-20L 2, MEM-20U 1 and MEM-20U 2	95	Fig. 21
Physical Memory Adresses for MEM-20L 1 and MEM-20U 1	96	Tab. 25
Physical Memory Adresses for MEM-20L 2 and MEM-20U 2	96	Tab. 26
Relating SBus Slots to SBUS_SSEL*x_x+1* Contents	100	Tab. 27
FMB Channel-0 Data Discard Status Register	102	Tab. 28
FMB Channel-1 Data Discard Status Register	102	Tab. 29
SEC Interrupt Mapping	108	Tab. 30
Physical Address Map of Local I/O Devices	110	Tab. 31
Physical Addresses for Boot PROM	111	Tab. 32
Physical Addresses for User Flash	112	Tab. 33
Boot ROM Size Control Register (BOOT_ROM_SIZE_CTRL)	113	Tab. 34
Flash Programming Control Register 2 (FLASH_CTRL2)	113	Tab. 35
Flash Programming Control Register 1 (FLASH_CTRL1)	114	Tab. 36
Flash Memory Selection Control Bits (x = don't care)	114	Tab. 37
Flash Programming Voltage Control Register (FLASH_VPP_CTRL)	115	Tab. 38
Physical Addresses for RTC/NVRAM	116	Tab. 39
Physical Addresses for Floppy Disk Controller	117	Tab. 40
SEC Auxiliary 1 Register	117	Tab. 41
Physical Addresses for Serial I/O A and B	118	Tab. 42
Physical Addresses for Audio Port	119	Tab. 43
Physical Addresses for Keyboard / Mouse	120	Tab. 44
Register Map of the LCA XC4003 – General Board Control	121	Tab. 45
LCA XC4003 Identification Register	122	Tab. 46
Register Map of the LCA XC3030 – Temp. Sensor and Audio	122	Tab. 47
LCA XC3030 Identification Register	123	Tab. 48
User LED 1 and 2 Control Registers	125	Tab. 49
Seven-Segment LED Display Control Register (SEV_SEG_CTRL)	126	Tab. 50
Naming the parts of the hexadecimal display	126	Fig. 22
Rotary Switch Status Register (ROTARY_SWITCH_STAT)	127	Tab. 51
Register Interface of the Temperature Sensors	128	Fig. 23
Temperature Sensor Select Register	129	Tab. 52
Temperature Sensor Communication Register	129	Tab. 53
Temperature Sensor Interrupt Status Register	130	Tab. 54
Temperature Sensor Status Register	130	Tab. 55
Temperature Sensor Interrupt Control Register	131	Tab. 56
Temperature Sensor Interrupt Level Register	132	Tab. 57
B_SW5-*x* and B_SW4-*x* Status Register	132	Tab. 58
B_SW7-1/2 and B_SW6-1/2/3 Status Register	133	Tab. 59
B_SW8-*x*, B_SW7-3, and B_SW6-4 Status Register	133	Tab. 60
I_SW2-*x* and I_SW1-*x* Status Register	133	Tab. 61
Address Translation (Master): Super- or Hypersparc – MBus – VMEbus	138	Fig. 24

204223 8 – 0 May 2000

	Page	Tab./Fig.
Mapping a VMEbus Area to the Processor's Virtual Address Space	139	Fig. 25
Address Translation (Slave): VMEbus – SBus – MBus – Super-/Hypersparc	140	Fig. 26
irq# Values for VME Interrupt Level Select and Enable Registers	148	Tab. 62
irq# Values for Interrupt Generation	152	Tab. 63
Interrupt Mapping	156	Tab. 64
Watchdog Timer Timeout Values	196	Tab. 65
Calling the OpenBoot `boot` Command Using `Busnet-tftp`	222	Fig. 27
Transfering Data Using the BusNet Protocol	224	Fig. 28
NVRAM Configuration Parameters	228	Tab. 66

Page Tab./Fig.

204223 8 – 0 May 2000

Using This Manual

This section does not provide information on the product, but on standard features of the manual itself:

- Its structure
- Special layout conventions
- Related documents

Audience of the Manual

This *Reference Guide* is intended for hard- and software developers installing and integrating the SPARC/CPU-20VT, the SPARC/CPU-20VTe, or the SPARC/CPU-20V into their systems.

- A SPARC/CPU-20VTe is a SPARC/CPU-20VT variant based on the FGA-5100 instead of the FGA-5000. Although of course the full FGA-5100 feature set is available in the FGA-5100 enhanced mode, the SPARC/CPU-20VTe is shipped with the FGA-5100 running in FGA-5000 compatibilty mode by default. Therefore, this manual does not differentiate between a SPARC/CPU-20VT and a SPARC/CPU-20VTe running in FGA-5000 compatibility mode and only highlights differences for the SPARC/CPU-20VTe running in FGA-5100 enhanced mode. This manual uses the term SPARC/CPU-20VT summarily (see the note at the beginning of section 3 “Installation” on page 11) unlessly explicitly stated otherwise.
- A SPARC/CPU-20V is a SPARC/CPU-20VT variant missing the SBus I/O board, being a two-slot configuration. Therefore, this manual does not differentiate between a SPARC/CPU-20VT two-slot configuration and a SPARC/CPU-20V and uses the term SPARC/CPU-20VT summarily (see the note at the beginning of section 3 “Installation” on page 11).

Overview of the Manual

This *Reference Guide* provides a comprehensive hardware and software guide to your board.

Note: **Please take a moment to examine the "Table of Contents" to see how this documentation is structured. This will be of value to you when looking for information in the future.**

It includes:

- Brief overview of the product, the specifications, the ordering information: see section 2 "Introduction" on page 5.
- Installation instructions for powering up the board: see section 3 "Installation" on page 11. It includes the default configuration (switches and the like), initialization, and connector pinouts.

  The installation instructions also appear as the product's installation guide – a separate manual delivered together with each product shipped.
- Detailed hardware description: see section 4 "Hardware Description" on page 85.
- Circuit schematics of the board for reference purposes.

  The circuit schematics are packaged separately to enable easy updating. They themselves are split into two subsections, one for the SPARC/CPU-20VT and one for the SPARC/CPU-20VTe. However, they will always be shipped together with this manual. Therefore:
- A detailed description of OpenBoot which controls the CPU board operations: see section 5 "Force OpenBoot Enhancements" on page 135 and section 6 "Sun OpenBoot (= OPEN BOOT PROM 2.0 MANUAL SET)".

  The Sun OpenBoot section is packaged separately to enable easy updating. It is always shipped together with this manual. Therefore:

☞ **Insert the Sun OpenBoot section now: see section 6 "Sun OpenBoot (= OPEN BOOT PROM 2.0 MANUAL SET)".**

The section 6 "Sun OpenBoot (= OPEN BOOT PROM 2.0 MANUAL SET)" includes the OPEN BOOT PROM 2.0 MANUAL SET, which is a manual in its own with the following sections:

– Open Boot 2.0 Quick Reference

– Open Boot 2.0 Command Reference

204223 8 – 0 May 2000

– FCODE Programs

There is additional space allocated in the manual for user notes, modifications, etc.

Data Sheets

The following data sheets are relevant for the SPARC/CPU-20VT. They contain information relevant for configuring and integrating the board into systems and can be found on the respective company's webpage.

- Audio Controller – Am79C30A (http://www.amd.com)
- Digital Thermometer and Thermostat – DS1620
- Ethernet, SCSI, Parallel-I/O – MACIO STP2000 (NCR89C100)
- Flash Memory – AM28F020 and Intel 28F008SA-L
- hyperSPARC single- and dual-CPU module – RT6224/6226K (http://www.bridgept.com)
- MBus-to-SBus Interface – M2S
- Memory Controller – RMC
- Real-time Clock and NVRAM – RTC/NVRAM MK48T18 (http://www.us.st.com)
- SBus-to-Ebus Interface – SEC STP2014
- Serial I/O and Keyboard / Mouse – SCC AM 85C30

Table a History of Manual Publication

Ed.	Date	Description
1.0	Jul. 1996	First print
2.0	Aug. 1996	Added description of OpenBoot commands for ID PROM
3.0	Nov. 1996	Corrected description of the VMEbus interrupt handler Corrected IOBP-10 P5 pinout Corrected examples in the OpenBoot section *VMEbus Interface* Corrected address map and configuration registers for SBus participants Corrected pinout of the RS-232 serial A+B and C+D front-panel connector Corrected description of `vsi-ibox-irq-map! (byte –)`
4.0	Mar. 1997	Added information for • booting Solaris 2.5, • testing fulfillment of thermal requirements for MBus module operation, • dual 150 MHz hyperSPARC MBus modules, • and SPARC/CPU-20V. Revised limits for temperature sensors

Table a History of Manual Publication

Ed.	Date	Description
5.0	Jun. 1997	Added information for SPARC/CPU-20VTe (FGA-5100 and audio factory option) and revised procedure for separating the base board and the SBus I/O board. Thoroughly revised section 5 “Force OpenBoot Enhancements”.
6.0	Jan. 1998	Integrated description of MBus modules with larger heat sinks. Added safety note for battery maintenance, revised patch information for 200 MHz modules, and corrected `mknod` parameters for serial device support in Solaris.
7.0	Apr. 1998	Added information for SPARC/CPU-20Ve. Description of FH-003 replaced by new hybrid FH-422T.
8.0	May 2000	Added safety notes, removed data sheet section, removed circuit schematics section, editorial changes

Table b Fonts, Notations and Conventions

Notation	Description
	All numbers are decimal numbers except when used with the following notations:
`0000.0000`$_{16}$	Typical notation for hexadecimal numbers (digits are 0 through F), e.g. used for addresses and offsets. Note the dot marking the 4th (to its right) and 5th (to its left) digit.
`0000`$_{8}$	Same for octal numbers (digits are 0 through 7)
`0000`$_{2}$	Same for binary numbers (digits are 0 and 1)
`Program`	Typical character format used for names, values, and the like. It is used to indicate when to type literally the same word. Also used for on-screen output.
Variable	Typical character format for words that represent a part of a command, a programming statement, or the like, and that will be replaced by an applicable value when actually applied.

Naming Convention

FGA-5000, FGA-5100^{C}, FGA-5100^{E}, and FGA-5x00 naming convention:

If applicable, all descriptions in this manual summarily use the term FGA-5x00 for the VMEbus interface device instead of using FGA-5000 and FGA-5100. In case of a description applying to only the FGA-5000,

204223 8 – 0 May 2000

the FGA-5100 in FGA-5000 Compatibility mode, or the FGA-5100 in FGA-5100 Enhanced mode, the respective device is named explicitly:

FGA-5000 • FGA-5000 means the FGA-5000 chip.

FGA-5100 • FGA-5100 means the FGA-5100 chip regardless of its operating mode.

FGA-5100^{C} • FGA-5100^{C} means the FGA-5100 chip being operated in FGA-5000 Compatibility mode.

FGA-5100^{E} • FGA-5100^{E} means the FGA-5100 chip being operated in FGA-5100 Enhanced mode.

FGA-5x00 • As already stated above, FGA-5x00 is used, if a description applies to both the FGA-5000 chip and the FGA-5100 chip.

Therefore, all features described using the terms FGA-5000 or FGA-5100^{C} are available on CPU boards which are based on the:

- FGA-5000 chip (FGA-5000)
- FGA-5100 chip running in FGA-5000 compatibility mode (FGA-5100^{C})

Using the term FGA-5100^{E} in a description implies that:

- An FGA-5100 based CPU board is required for the feature to be available.
- It is required to run the FGA-5100 in the FGA-5100 enhanced mode for the feature to be available.

Using the term FGA-5x00 in a description implies that the description applies to FGA-5100 based CPU boards (running the FGA-5100 in either of its modes) and to FGA-5000 based CPU boards.

Icons for Ease of Use: Safety Notes and Tips & Tricks

There are three levels of safety notes used in this manual which are described below in brief by displaying a typical layout example.

Be sure to always read and follow the safety notes of a section first – before acting as documented in the other parts of the section.

Danger



Dangerous situation: injuries to people and severe damage to objects possible.

Caution



Possibly dangerous situation: no injuries to people but damage to objects possible.

Note: **No danger encountered. Pay attention to important information marked using this layout.**

204223 8 – 0 May 2000



1 Safety Notes

This section provides safety precautions to follow when installing, operating, and maintaining the SPARC/CPU-20VT. For your protection, follow all warnings and instructions found in the following text.

General

This *Reference Guide* provides the necessary information to install and handle the SPARC/CPU-20VT. As the product is complex and its usage manifold, we do not guarantee that the given information is complete. In case you need additional information, ask your Force Computers representative.

The SPARC/CPU-20VT has been designed to meet the standard industrial safety requirements. It must not be used except in its specific area of office telecommunication industry and industrial control.

Only personnel trained by Force Computers or qualified persons in electronics or electrical engineering are authorized to install, uninstall or maintain the SPARC/CPU-20VT. The information given in this manual is meant to complete the knowledge of a specialist and must not be taken as replacement for qualified personnel.

Make sure that contacts and cables of the board cannot be touched while the board is operating.

Installation

Electrostatic discharge and incorrect board installation and uninstallation can damage circuits or shorten their life. Therefore:

- **Before installing the board, check section 3.1.1 “Requirements” on page 12.**
- **Before touching integrated circuits, ensure that you are working in an ESD safe environment.**
- **When plugging the board in or removing it, do not press on the front panel but use the handles.**
- **Before installing or uninstalling the board, read section 3 “Installation” on page 11.**
- **Before installing or uninstalling an additional device or module, read the respective documentation.**
- **Ensure that the board is connected to the CompactPCI backplane via all assembled connectors and that power is available on all power pins.**



- **Before installing or uninstalling the board, read *section 3 "Installation" on page 11*.**
- **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.**
  - **Ensure that the board is connected to the VMEbus via both connectors, the P1 and the P2, and that power is available on all of them.**
- **When operating the board in areas of strong electro-magnetic radiation, ensure that the board is:**
  - **Bolted on the VME rack**
  - **Shielded by closed housing**

Heat Sink Temperature

The heat sink temperature measured must always stay

- **Below 74 °C for a single 125-MHz hyperSPARC module**
- **Below 80 °C for a single 150-MHz hyperSPARC module**
- **Below 85 °C for a dual 125-MHz hyperSPARC module**
- **Below 95 °C for a dual 150-MHz hyperSPARC module**

Operation

- **While operating the board ensure that the requirements as given in section 3.1.1 "Requirements" on page 12 are met.**

EMC

- **If boards are integrated into open systems, always cover empty slots.**
- **The front panel of the SPARC/CPU-20VT provides one cutout for a PMC module. If the board is shipped without the module installed, the front-panel cutout is covered by a blind panel to ensure proper EMC shielding. To ensure proper EMC shielding, always operate the SPARC/CPU-20VT with the blind panel or with a PMC module installed.**

Expansion

- **Check the total power consumption of all components installed (see the technical specification of the respective components). For**

204223.8 – 0 May 2000



the total power consumption of the SPARC/CPU-20VT, see section "Power Supply" on page 12.

- **Ensure that any individual output current of any source stays within its acceptable limits (see the technical specification of the respective source).**
- **Only replace components or system parts with those recommended by Force Computers. In case you use components other than those recommended by Force Computers, you are fully responsible for the impact on EMI and the eventually changed functionality of the product.**

MBus Module

Note that in case of a SPARC/CPU-20VT configuration using two dual 150 MHz hyperSPARC MBus modules (MBus-150H2/*x*) you have to use:

- **MBus-150H2/2 (with broader heat sink) as MBus module 1**
- **MBus-150H2/1 (with narrower heat sink) as MBus module 2**
- **In case of a SPARC/CPU-20VT variant using a 200 MHz hyperSPARC MBus module (e.g. a SPARC/CPU-20V) you might have to patch Solaris as described in section 3.10 "Patching Solaris 2.5 for 200 MHz MBus Modules" on page 80.**

IOBP

The SPARC/IOBP-10 is especially designed for the SPARC/CPU-20VT. Do not use any other I/O panels on the SPARC/CPU-20VT.

SCSI

The SPARC/CPU-20VT provides two distinct SCSI buses, which can easily get mixed up. Therefore, before dealing with a SPARC/CPU-20VT SCSI bus, ensure that you select the correct board:

- **SCSI #1 is located on the base board – associated to the MACIO 1 controller.**
- **SCSI #2 is located on the SBus I/O board – associated to the MACIO 2 controller.**

Ethernet

Before dealing with a SPARC/CPU-20VT Ethernet interface, ensure that you select the correct board:

- **Ethernet #1 is located on the base board.**
- **Ethernet #2 is located on the SBus I/O board.**

- **Never connect the SPARC/CPU-20VT to the same Ethernet network via the Ethernet #1 interface and via the Ethernet #2 interface.**

Booting

This information is relevant for:

- **Use of Solaris 2.5 together with OpenBoot 2.25.1 on a two-slot configuration of a SPARC/CPU-20VT (for example on a SPARC/CPU-20V),**
- **Booting Solaris 2.5 with two or more CPUs (e.g. one dual hyperSPARC module or two single hyperSPARC modules) and an SBus graphic card (framebuffer):**

  –From CD-ROM

  –From disk using the reconfigure option (`-r`) of the `boot` or `reboot` command.
- **Only in these two situations with this special hardware configuration you need to remove the serial devices `ttyc` and `ttyd` to successfully boot Solaris 2.5.**
- **You have to remove `ttyc` and `ttyd` whenever you are in one of the two situations. If necessary for your application and if using a three- or four-slot configuration of a SPARC/CPU-20VT, you always have to add them again as described on the next page.**
- **The following description applies from OpenBoot Version 2.25.2 or greater.**

Environment

Always dispose of used batteries and/or old boards according to your country's legislation.

Battery

The Lithium battery integrated in the RTC/NVRAM provides a data retention of at least 10 years summing up all periods of actual battery use. After this period the Ethernet address, the host ID address, and all other RTC/NVRAM contents may be lost.

If a Lithium battery on the board has to be exchanged, observe the following safety notes:

- **Contact Force Computers before ten years of actual battery use have elapsed.**
- **Never exchange the RTC/NVRAM yourself.**
- **Incorrect exchange of Lithium batteries can result in a hazardous explosion.**

204223 8 – 0 May 2000

2 Introduction

The SPARC/CPU-20VT is a 6U VMEbus computer and is 100% SPARCstation-20 compatible. Providing two MBus slots, it offers scaleability from single to quad processor configurations with the feature of field-upgrades. Through this combination of powerful processing power, with a full set of I/O interfaces including fast SCSI, Ethernet, floppy disk, serial I/O, Centronics compliant parallel I/O, keyboard/mouse ports, the SPARC/CPU-20VT becomes a high performance solution for embedded applications.
A full 32-bit 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-20VT 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-20VT ranges from Solaris, the most popular implementation of the UNIX operating system, to sophisticated real-time operating systems.

The SPARC/CPU-20VT is a VMEbus computer combining workstation performance and functionality with the ruggedness and expandability of the industry standard 6U VMEbus form factor.

Feature Summary

- 100% SPARCstation-20 compatible in a four-slot VMEbus unit
- Two MBus slots for single, dual and quad processor configurations
- 64 to 512 MByte main memory with ECC
- Up to 512 KByte boot flash
- Up to four MByte user flash (factory option)
- VME64 interface based on the FGA-5000 or the FGA-5100 (see “Variants” on page 6)
- I/O interfaces on base board: Ethernet, SCSI, serial A&B, floppy and keyboard/mouse
- Second SCSI and Ethernet, Centronics, two additional serial lines, two SBus connectors, and as factory option one audio port through SBus I/O board – for high-availability and high-performance applications
- 6U VME form factor with scalable slot consumption

Variants

There are two other product families which are closely related to the SPARC/CPU-20VT and which are also documented in this Reference Guide:

- SPARC/CPU-20VTe: the SPARC/CPU-20VTe is based on the high-performance FGA-5100 VMEbus interface chip instead of using the FGA-5000 and always includes the audio factory option. By default, the SPARC/CPU-20VTe is configured to run in the FGA-5000 compatibility mode to be fully software compatible with the FGA-5000 based SPARC/CPU-20VT variants. The FGA-5100 enhanced feature set is available when running the SPARC/CPU-20VTe in the FGA-5100 enhanced mode (see section 3.8.6 “Using the FGA-5100 Enhanced Feature Set” on page 73).
- SPARC/CPU-20V: the SPARC/CPU-20V is a two-slot configuration of the SPARC/CPU-20VT with no SBus I/O board installed.

For a short summary of feature differences, see the description right at the beginning of section 3 “Installation” on page 11.

Table 1 Specifications of the SPARC/CPU-20VT

Processor	HyperSPARC
Shared Main Memory	64 to 512 MByte DRAM with ECC
SBus Slots	Two for 32 & 64 bit SBus modules
SCSI with DMA to SBus	53C90A superset I/O on front panel and P2
Ethernet with DMA to SBus	AM7990 compatible I/O on front panel and P2
Parallel port with DMA to SBus	Centronics compatible, uni- or bidirectional I/O on front panel or P2
Floppy Disk Interface	I/O on P2
Serial I/O	Two ports with RS-232 configuration (optional RS-422) I/O on front panel and P2
Audio Port (factory option)	I/O on front panel (microphone and earpiece) or P2 (loud speaker)
Keyboard/Mouse Port	I/O on front panel and P2
Counters/Timers	Five 32-bit, programmable
Boot PROM	Up to 512 KByte On-board programmable Hardware write protection

204223 8 – 0 May 2000

Table 1 Specifications of the SPARC/CPU-20VT (cont.)

User Flash	Up to 4 MByte (2 MByte in default configuration) On-board programmable Hardware write protection
RTC/NVRAM/Battery	MK48T18; NVRAM reserved if using OpenBoot
VMEbus Interface	FGA-5000 (32-bit master/slave, ANSI/VITA 1-1994)
Additional Features	Reset and abort key, status LEDs, hexadecimal display, rotary switch
Firmware	OpenBoot with diagnostics
Power consumption	See section 3.1 “Installation Prerequisites and Requirements” on page 12
Environm. Conditions Temperature (Operating) Temperature (Storage) Humidity	 0°C to +50 °C -20°C to +75 °C 5% to 95 % noncondensing

The SPARC/CPU-20VT is available in two VMEbus interface variants, with or without SBus I/O board as well as several memory and speed options. For example, the SPARC/CPU-20VTe utilizes the high performance FGA-5100 which also provides additional VMEbus transfer modes such as 2eVME (for a short overview of enhancements see “FGA-5100E Extended Features” on page 98).

Consult your local sales representative to confirm availability of specific combinations. The table below explains the general product nomenclature.

Table 2 Product Nomenclature

SPARC/CPU-20V/xxx-yyyHmn-z		
SPARC/CPU-20VT/xxx-yyyHmn-z		
SPARC/CPU-20VTe/xxx-yyyHmn-z		
xxx = MByte DRAM	yyy = MBus module H = hyperSPARC m = no. of processors on MBus mod. 1 optional: n = no. of processors on MBus mod. 2	z = MByte User Flash
64 =64 MByte 128 =128 MByte	200H1 = single 200 MHz hyperSPARC 200H2 = dual 200 MHz hyperSPARC 200H22 = two dual 200 MHz hyperSPARC	2 = 2 MByte

The following table is an excerpt from the SPARC/CPU-20VT data sheet. Please ask your local Force Computers representative for current information.

Table 3 Excerpt from the Data Sheet's Ordering Information

Product Name	Description
SPARC/CPU-20VTe/…	
…/64-200H1-2	Single 200 MHz hyperSPARC with 64 MByte Main Memory
…/64-200H2-2	Dual 200 MHz hyperSPARC with 64 MByte Main Memory
…/128-200H1-2	Single 200 MHz hyperSPARC with 128 MByte Main Memory
…/128-200H2-2	Dual 200 MHz hyperSPARC with 128 MByte Main Memory
SPARC/CPU-20VT/…	Same as SPARC/CPU-20VTe… but with the FGA-5000 assembled instead of the FGA-5100
SPARC/CPU-20V…	Same as SPARC/CPU-20VT… but without SBus I/O Board
SPARC/MEM-20L/64	64 MB ECC Memory Expansion Module, Lower Position
SPARC/MEM-20L/128	128 MB ECC Memory Expansion Module, Lower Position
SPARC/MEM-20U/64	64 MB ECC Memory Expansion Module, Upper Position
SPARC/MEM-20U/128	128 MB ECC Memory Expansion Module, Upper Position
SPARC/MBus-*xxx*H1	Single hyperSPARC MBus module. *xxx* specifies the processor frequency. For example, SPARC/MBus-150H1 is a single 150 MHz hyperSPARC MBus module.
SPARC/MBus-*xxx*H2	Dual hyperSPARC MBus module. *xxx* specifies the processor frequency. For example, SPARC/MBus-150H1 is a dual 125 MHz hyperSPARC MBus module.
SPARC/MBus-150H2/1	Additional dual 150 MHz hyperSPARC MBus Module
SBus Modules:	
SBus/TGX	Color 2D and 3D wire frame graphics accelerator, 1152 x 900, 8 bit per pixel, single SBus Slot.

204223 8 – 0 May 2000

Table 3 Excerpt from the Data Sheet's Ordering Information (cont.)

Product Name	Description
SBus/TGX+	Color 2D and 3D wire frame graphics accelerator, 1280 x 1024, 8 bit per pixel, double buffering, uses two SBus slots.
Accessories SPARC/CPU-20VT…	
…/TM	Reference Guide Set for SPARC/CPU-20VT including OpenBoot User's Manual, a detailed hardware description and Set of Data Sheets for the SPARC/CPU-20VT.
…/SBus-AccKit	Front Panel and VME Power Module for SBus Expansion
…/FP-AccKit	Cable Set for Front Panel Connectors
…/P2-AccKit	Cable Set for one P2 Connector incl. IOBP-10
Software	Please contact your local sales representative for current version information, VMEbus driver support, and documentation.
Solaris 2.x/CPU-20VT	Current version of Solaris 2.x
Solaris 1.1/CPU-20VT	Current version of Solaris 1.1.x
BusNet	BusNet runtime package for Solaris on SPARC VMEbus boards: available for Solaris 2.x and Solaris 1.1

204223 8 – 0 May 2000

3 Installation

This *Reference Guide* covers four product families:

- SPARC/CPU-20V and SPARC/CPU-20Ve
- SPARC/CPU-20VT and SPARC/CPU-20VTe

SPARC/CPU-20V

Note: **The SPARC/CPU-20V is a two-slot configuration of the SPARC/CPU-20VT. Therefore, this manual does not differentiate between both and uses the term SPARC/CPU-20VT summarily.**

- Note, however, that the following applies to this manual:
  - Not all features described for the SPARC/CPU-20VT are provided by a SPARC/CPU-20V. The major differences are described in section 3.2.3 “SPARC/CPU-20V” on page 26.
  - Only those features are available which are available in a SPARC/CPU-20VT two-slot configuration.
- See section 3.2.2 “Two-, Three-, and Four-Slot SPARC/CPU-20VT Configurations” on page 18 for information on the mechanical differences and restrictions applying to the SPARC/CPU-20V concerning the usage of MBus modules.

SPARC/CPU-20 VTe

Note: **The SPARC/CPU-20VTe is a SPARC/CPU-20VT variant based on the FGA-5100 instead of the FGA-5000. By default, the FGA-5100 of the SPARC/CPU-20VTe runs in FGA-5000 Compatibility mode. Therefore, this manual does not differentiate between both and uses the term SPARC/CPU-20VT summarily.**

- Note, however, that the following applies to this manual:
  - All features described for the SPARC/CPU-20VT are provided by a SPARC/CPU-20VTe.
  - The SPARC/CPU-20VTe comes with the audio factory option installed on the SBus I/O board (see section 3.4.3 “Audio Connector Pinout (Factory Option)” on page 46).
  - The SPARC/CPU-20VTe provides the extended FGA-5100 capabilities only if the FGA-5100 is operated in the FGA-5100 Enhanced mode (FGA-5100^{E}).
- Refer to the *FGA-5x00 Reference Guide* for information on FGA-5100 features provided in addition to the FGA-5000 features and see “Using This Manual” in this manual’s front matter for naming conventions.

SPARC/CPU-20Ve

Note: **:The SPARC/CPU-20Ve is a two-slot configuration of the SPARC/CPU-20VTe. Therefore, analogous differences apply to the SPARC/CPU-20Ve and the SPARC/CPU-20VTe as compared with the differences between the SPARC/CPU-20V and the SPARC/CPU-20VT (see above).**

3.1 Installation Prerequisites and Requirements

Note: **Before powering up check the items described in the following list.**

- Check this section for installation prerequisites and requirements.
- Check the consistency of the current switch settings (see section 3.3 “Switch Settings” on page 33).

3.1.1 Requirements

The installation requires only:

- Power supply meeting the requirements of the power consumptions for the SPARC/CPU-20VT configuration used (see below)
- Fan unit providing an airflow meeting the thermal requirements for the SPARC/CPU-20VT configuration used (see “Thermel Requirements” on page 13)
- VMEbus backplane with P1 and P2 connectors

Power Supply

The power supply needs to provide the following voltages:

- +5 V
- ±12 V (required for RS-232 serial and Ethernet interfaces)

The SPARC/CPU-20VT power consumption depends on the installed MBus and SBus modules. For the SPARC/CPU-20VT MBus modules available from Force Computers the following configurations are valid:

Two Slots

- A two-slot configuration suffices already if only one single or one dual hyperSPARC MBus module is installed, but no SBus module.

Three Slots

- A three-slot configuration suffices if up to two single or one dual hyperSPARC MBus modules, but no SBus modules are installed.

204223.8 – 0 May 2000

Four Slots

- A four-slot configuration using the SPARC/CPU-20VT/SBus-AccKit accessory kit is required:
  - If any SBus module is installed
  - If two dual hyperSPARC MBus modules are installed

  The accessory kit is available from Force Computers and includes an additional VME power module as well as a front panel. The accessory kit's front panel is prepared for the installation of SBus modules on the SBus I/O board.

Typical Power Consumption

A SPARC/CPU-20VT with one single 150 MHz hyperSPARC MBus module typically draws 11.3 A (5V) from the VME backplane. A SPARC/CPU-20VT with one dual 125 MHz hyperSPARC MBus module typically draws 14.2 A (5V) and a SPARC/CPU-20VT with two single 150 MHz hyperSPARC MBus modules typically draws 18.5 A (5V). All values are measured with no SBus module installed.

Thermel Requirements

The operating temperature is 0 ×C to +50 ×C (5 % to 95 % noncondensing humidity). When operating the SPARC/CPU-20VT in systems providing a minimum forced airflow of 300 LFM (linear feet per minute), the typical operating temperature is 0 °C to +40 °C. When exceeding the typical operating temperature range and operating the SPARC/CPU-20VT with two MBus modules, ensure that one of the following condition sets is met:

- Either:
  - The temperature of the incoming airflow at the MBus modules is always below +50 °C.
  - The temperature of the outgoing airflow at the MBus modules is always below +60 °C.
- Or:
  - The temperature of the airflow between the MBus modules is always below +50 °C
  - There is a minimum airflow of 300 LFM between the MBus modules.

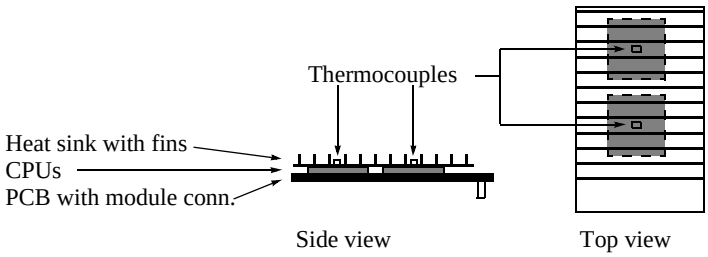
Testing

To test if your system configuration fulfills the thermal requirements concerning the MBus modules on the SPARC/CPU-20VT, do the following:

1. Uninstall the SPARC/CPU-20VT.
2. For each MBus module in your SPARC/CPU-20VT configuration:
   - Locate the CPU(s) on the MBus module (see figure below).
   - Attach a thermocouple to the surface of the heat sink above each CPU.

Note: **Every thermocouple must be located between fins and centered above a CPU.**

The figure below shows the locations for a dual 150-MHz hyperSPARC MBus module (schematic):



3. Install the SPARC/CPU-20VT in your system.
4. Operate your system as usual but in hot environment while measuring the temperature using the installed thermocouples.

   Operating conditions:

   – Maximum temperature

   – With specified airflow

   – According to your application

   – According to system specification

   – According to SPARC/CPU-20VT specification

Caution



The heat sink temperature measured must always stay

- **Below 74 °C for a single 125-MHz hyperSPARC module**
- **Below 80 °C for a single 150-MHz hyperSPARC module**
- **Below 85 °C for a dual 125-MHz hyperSPARC module**
- **Below 95 °C for a dual 150-MHz hyperSPARC module**

3.1.2 Terminal Connection

The SPARC/CPU-20VT provides four serial ports: A and B are implemented on the base board, C and D on the SBus I/O board (for Solaris support of `ttyc` and `ttyd` see “Solaris Support for serial I/O Ports ttyc and ttyd” on page 68). For the initial power up, a terminal can be connected to port A via the front-panel twentysix-pin microconnector SERIAL A+B. Per default, all serial I/O ports provide an RS-232 interface. Alternatively, each of the four ports can be configured as RS-422 interface by using the Force Computers FH-422T hybrid.

204223 8 – 0 May 2000

For the correct switch settings see

- table 5 “Base Board Switch Settings” on page 34: B_SW4 for port A and B_SW5 for port B
- table 6 “SBus I/O Board Switch Settings” on page 38: I_SW1 for port C and I_SW2 for port D.

For information on the serial port connector pinout, see section 3.4.1 “Serial I/O Port RS-232 and RS-422 Connector Pinout” on page 44.

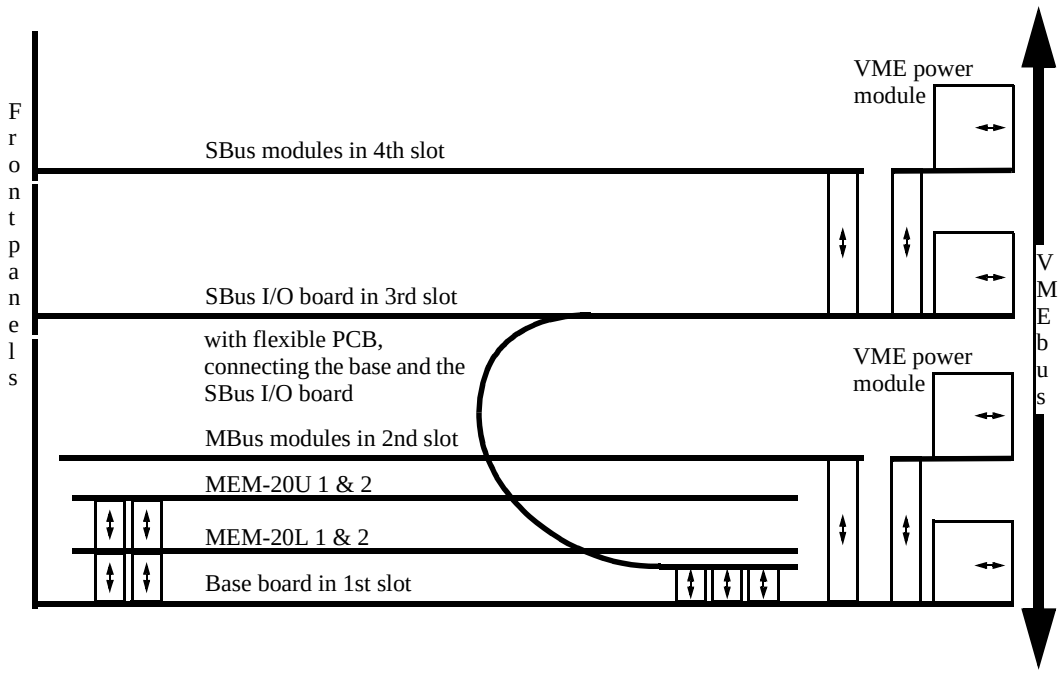
3.2 Mechanical Construction

The SPARC/CPU-20VT is a 6U VMEbus computer. It occupies up to four VMEbus slots and consists of the following major components, which are introduced in section 3.2.5 “Major Components” on page 29:

- Base board with one to four memory modules (MEM-20U/L one or two; U = upper, L = lower) connected to it
- One or two MBus modules with a VME power module installed in the VMEbus slot they occupy
- SBus I/O board
- Up to two SBus modules (single width) with a second power module installed in the VMEbus slot they occupy

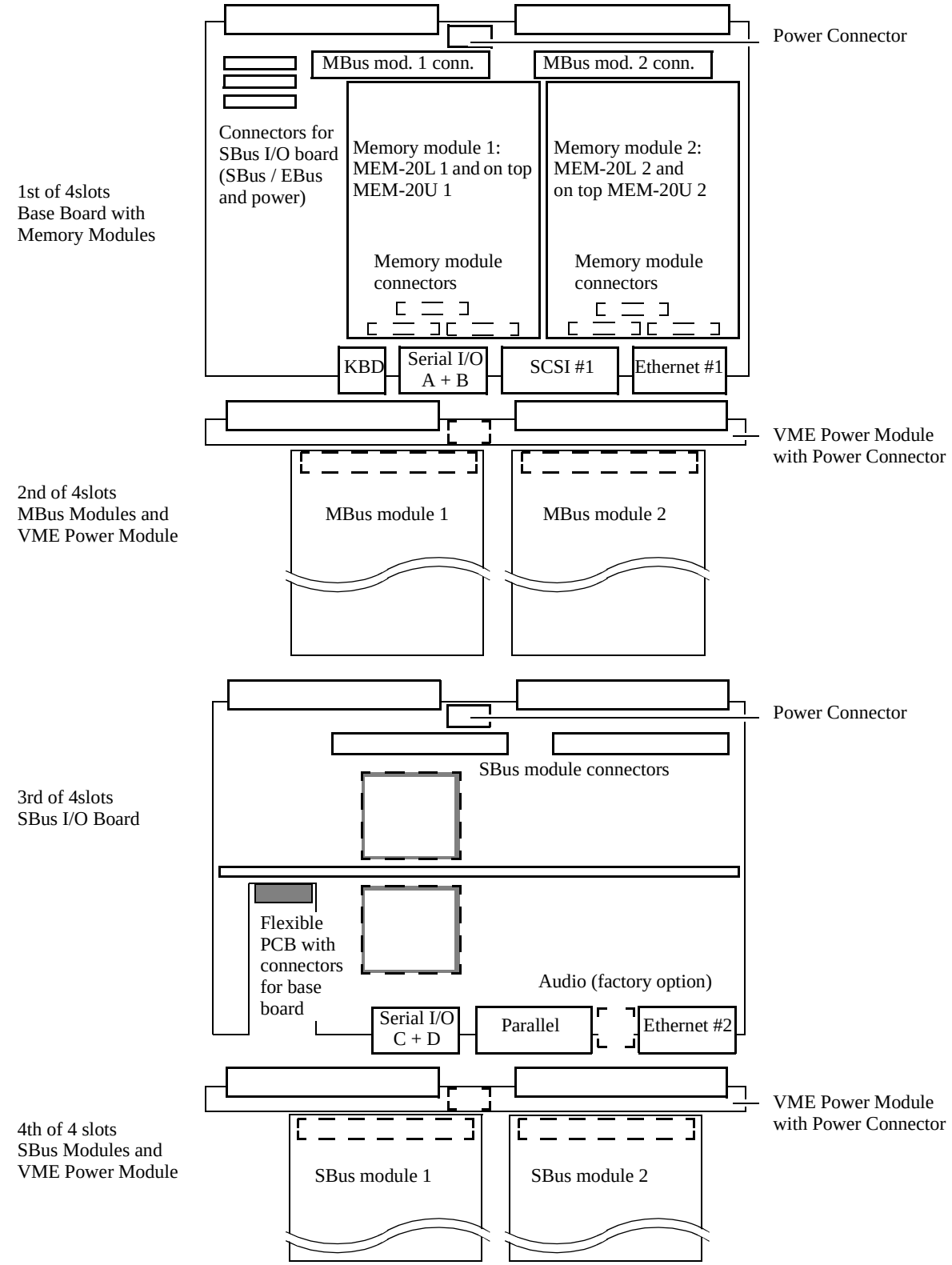
3.2.1 Mechanical Overview of a Completely Assembled SPARC/CPU-20VT

Figure 1 Cross Section of a Completely Assembled SPARC/CPU-20VT



204223 8 – 0 May 2000

Figure 2 Components and Connectors of the SPARC/CPU-20VT (Schematic)



3.2.2 Two-, Three-, and Four-Slot SPARC/CPU-20VT Configurations

Note: **At least the following components are required to be installed (to locate them see figure below):**

- MBus module 1
- Lower memory module 1 (MEM-20L 1)
- VME power module in the 2nd VMEbus slot which is occupied by the SPARC/CPU-20VT

SPARC/CPU-20V

The minimum configuration can be achieved in a two-slot configuration, which is also available as SPARC/CPU-20V.

Note: **The SPARC/CPU-20VT supports a wide range of MBus modules. This allows for adjusting the processor performance to the user's needs. However, not every possible configuration has been qualified by Force Computers. For information on qualified configurations refer to the document *Board Reconfiguration with MBus Modules*. The document is delivered together with every MBus module purchased from Force Computers (also with every such module installed on a CPU or ESP board). If you are not sure whether the configuration you want to use has been qualified, or not, ask your local Force Computers sales representative.**

General Restrictions for Two or Three-Slot Configurations

When operating the SPARC/CPU-20VT in a two- or three-slot configuration the following restrictions apply – mainly due to missing power supply:

- In a two-slot configuration never operate the SPARC/CPU-20VT with a second MBus module installed.
- In a three-slot configuration never operate the SPARC/CPU-20VT with two dual hyperSPARC MBus modules or with any SBus module installed.

The following three figures show examples for four-, three-, and two-slot configurations.

Note: **The base board holds sockets for boot PROM devices and for an RTC/NVRAM. These sockets are only provided for downgrading to a two-slot configuration.**

- Ensure that no devices are assembled in these sockets when operating the SPARC/CPU-20VT in a three- or four-slot configuration.

204223 8 – 0 May 2000

Figure 3 Sample Four-Slot Configuration

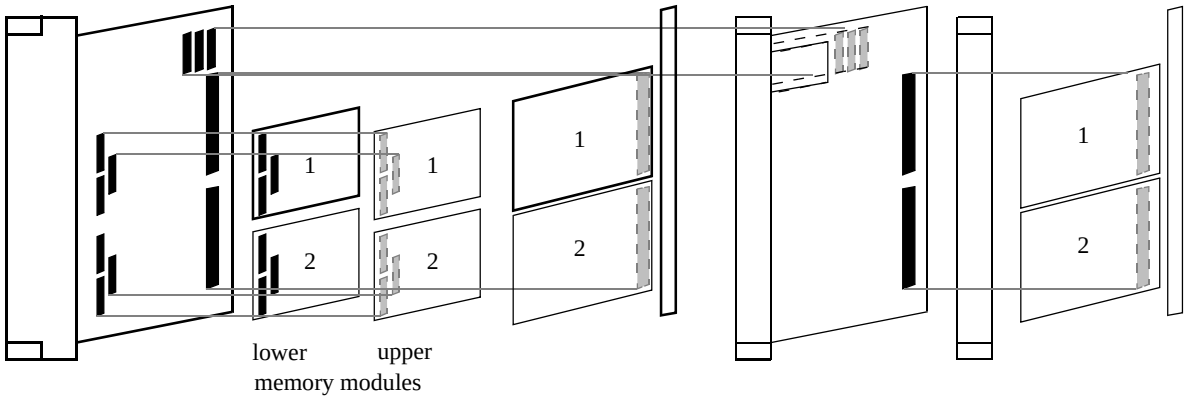


Figure 4 Sample Three-Slot Configuration

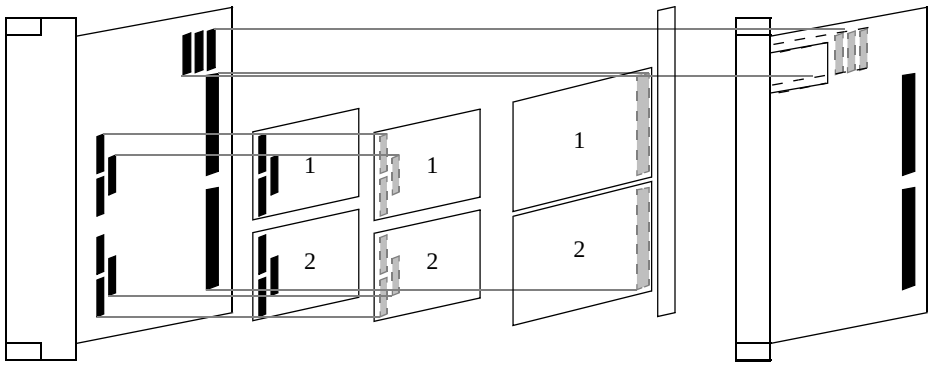
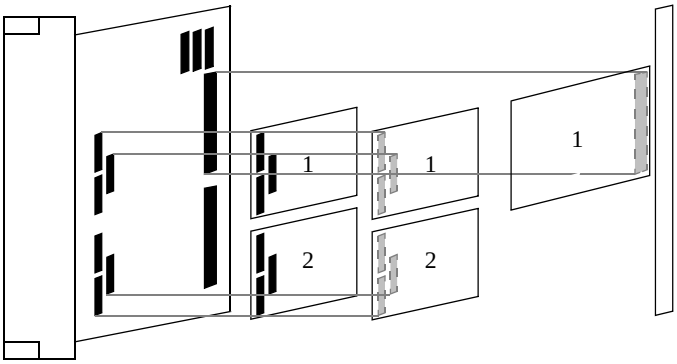


Figure 5 Sample Two-Slot Configuration



From Three to Two Slots

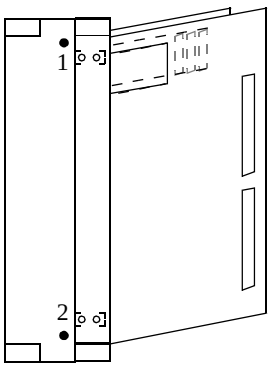
To downgrade the SPARC/CPU-20VT from a three-slot configuration to a two-slot configuration follow the general procedure outlined below:

1. Power off the VME rack and the SPARC/CPU-20VT.
2. Remove the SPARC/CPU-20VT from the VME rack.
3. Separate the base board and the SBus I/O board (see respective section on page 20) and keep the spare parts for later use.
4. Move the two boot PROM devices and the RTC/NVRAM device from the SBus I/O board to the base board (see respective section on page 24).
5. Remove the second MBus module if there is a second one (see respective section on page 25) and keep the spare parts for later use.

Separating the Base Board and the SBus I/O Board

Follow the steps below:

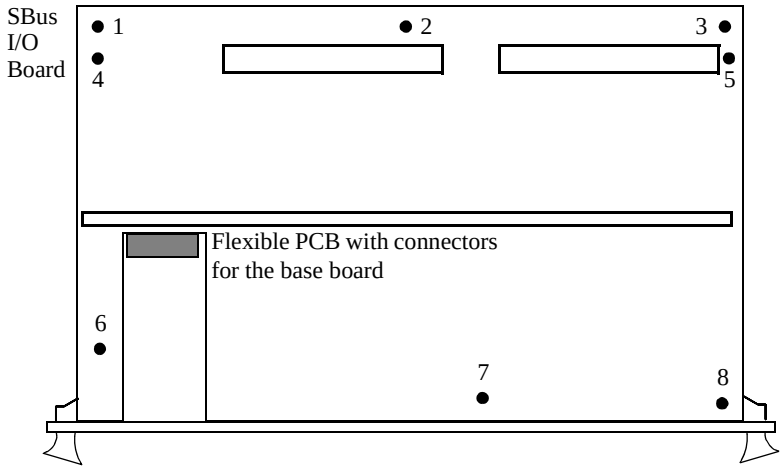
1. Disassemble the two screws on the front panel marked below as 1 and 2:



Base Board and SBus I/O Board

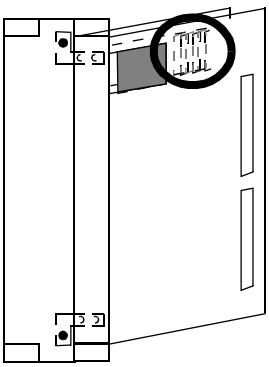
204223 8 – 0 May 2000

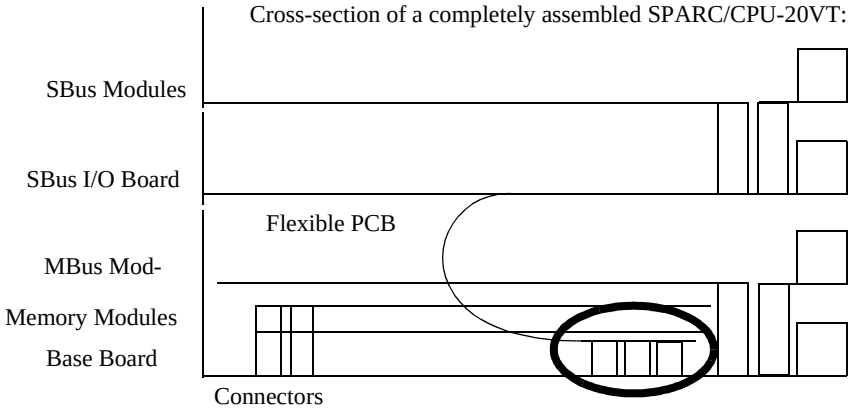
2. Disassemble the eight screws on the SBus I/O board marked below as 1, 2, …, and 8:



Screws 1, 2, and 3 connect the SBus I/O board to the VME power module in the second slot occupied by the SPARC/CPU-20VT. The 3 screws are replaced by standoffs, if the SPARC/CPU-20VT/SBus-AccKit accessory kit is installed.

3. Unplug the connectors on the SBus I/O board's flexible PCB from the base board's connectors.

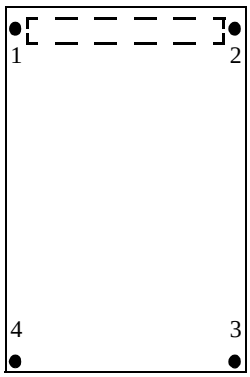




4. This step is only relevant in case of a SPARC/CPU-20VT configuration using hyperSPARC MBus modules with large heat sinks as shown in the figure below.

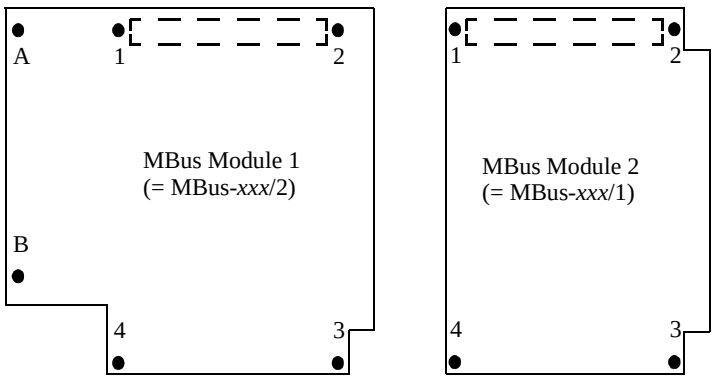
Standard MBus Module

Top View of Standard MBus Module (Mounting Holes marked as 1 … 4)

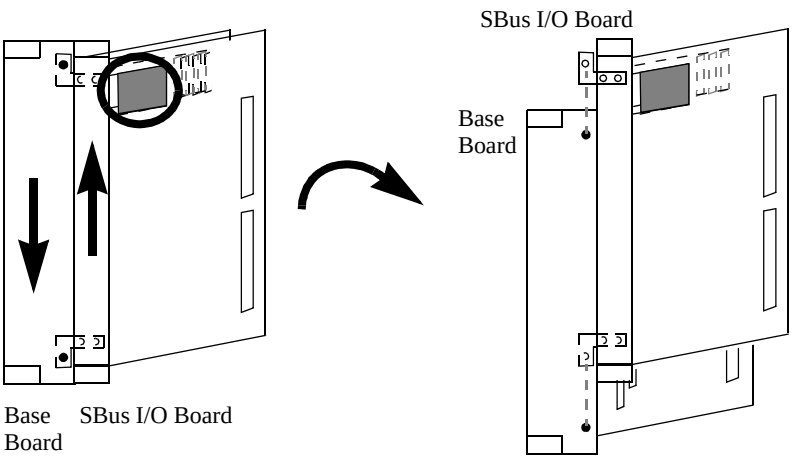


MBus Modules with Large Heat Sinks

Top View of MBus Modules with Larger Heat Sinks (Mounting Holes Marked as 1 … 4, A, B)

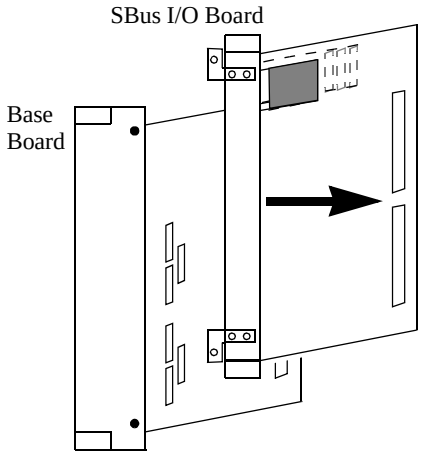


In this case, the SBus I/O board's flexible PCB is carried through a gap in the heat sink of MBus module 1. Therefore, it is necessary to first carefully separate the base board from the SBus I/O board as shown in the figure below before proceeding with the next step:



204223 8 – 0 May 2000

5. Separate the base board from the SBus I/O board carefully as shown below:



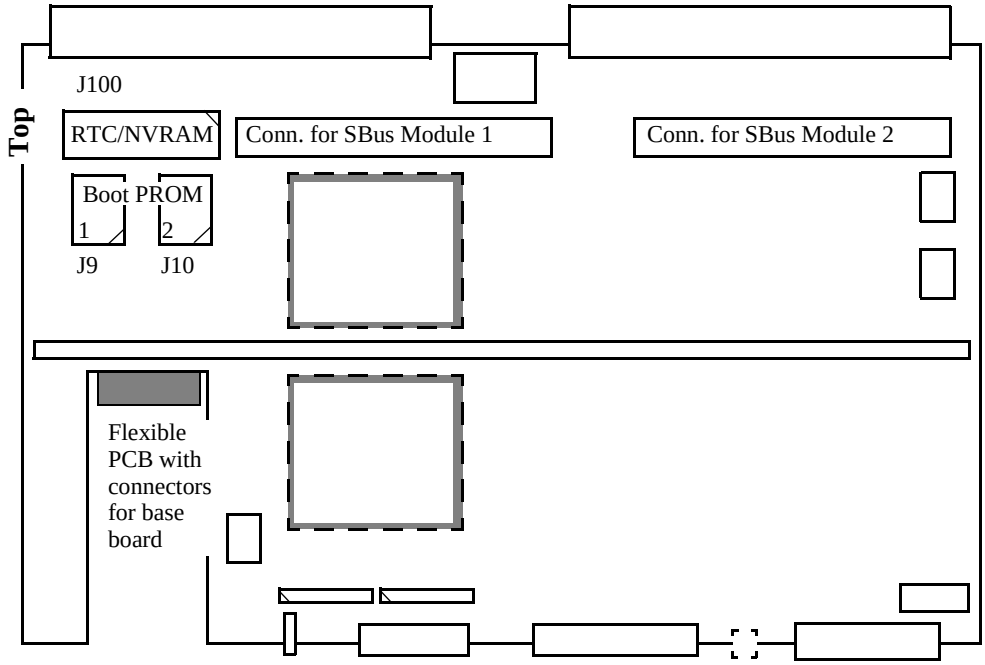
6. Disassemble the standoffs corresponding to the screws at the locations 1, 2, 3, …, and 8 as shown in the figure of step 2 on page 21.
7. Store the standoffs, the screws, and the SBus I/O board in a safe place for later use.

After having taken the steps above, the SPARC/CPU-20VT is separated into its base board and its SBus I/O board.

Moving the Boot PROM and RTC/NVRAM Devices

Note: **Note the position of the diagonally cut edge on each of the three devices and in figure 6 "SBus I/O Board: Boot PROM and RTC/NVRAM Device Locations" on page 24 and figure 7 "Base Board: Boot PROM and RTC/NVRAM Device Locations" on page 25. Assemble each device accordingly.**

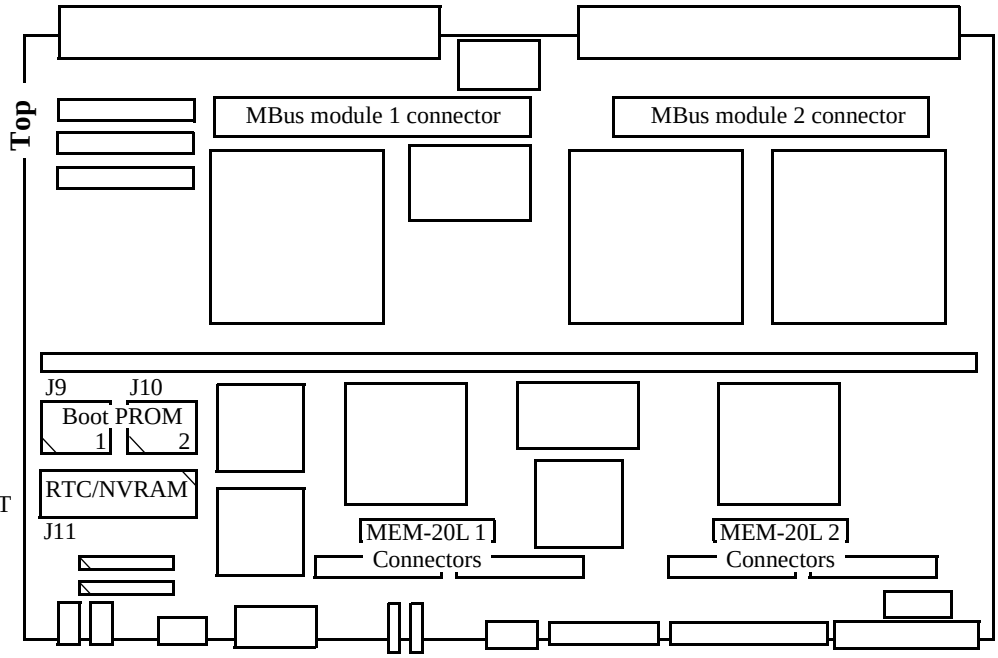
Figure 6 **SBus I/O Board: Boot PROM and RTC/NVRAM Device Locations**



1. Remove the boot PROM device one from socket J9 on the SBus I/O board (see preceding figure).
2. Insert it in socket J9 on the base board (see following figure).
3. Remove the boot PROM device two from socket J10 on the SBus I/O board (see preceding figure).
4. Insert it in socket J10 on the base board (see following figure).
5. Remove the RTC/NVRAM device from socket J100 on the SBus I/O board (see preceding figure).
6. Insert it in socket J11 on the base board (see following figure).

204223 8 – 0 May 2000

Figure 7 Base Board: Boot PROM and RTC/NVRAM Device Locations

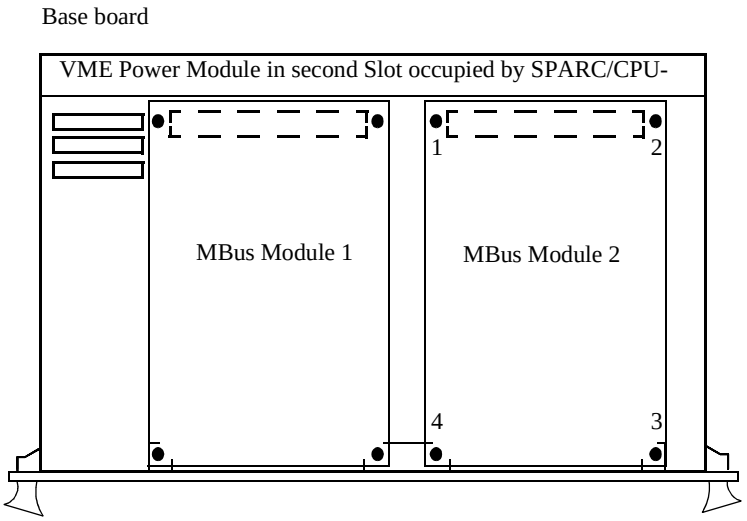


Boot PROM and RTC/NVRAM are only assembled on the base board if operating the SPARC/CPU-20VT in a 2-slot configuration.

Removing the Second MBus Module

Follow the steps below:

1. Disassemble the screws marked as 1, 2, 3, and 4 in the figure below:



2. Disconnect the MBus module 2 from the base board.
3. Re-assemble the screws marked as 1, 2, 3, and 4 in the figure above for later use.

3.2.3 SPARC/CPU-20V

Since the SPARC/CPU-20V is a two-slot configuration of the SPARC/CPU-20VT, the *Installation Guide* and the *Reference Guide* do not differentiate between both in most cases and use the term SPARC/CPU-20VT summarily. However, note the differences described in the remaining parts of this section.

Manuals

The following applies to the manuals:

- Not all features described for the SPARC/CPU-20VT are provided by a SPARC/CPU-20V. Features requiring the SBus I/O board are: serial ports C and D, SCSI #2, Ethernet #2, and parallel port (to name the most prominent).

  As there are no serial ports C and D you do not have to modify `/etc/system`. However, note that in case of OpenBoot 2.25.1 you need to do the steps described in this update concerning Solaris 2.5: see section 3.9 “Booting Solaris 2.5” on page 76.

- Only those features are available which are available in a SPARC/CPU-20VT two-slot configuration.

Configuration Restrictions

The MBus module architecture allows for adjusting the processor performance to the user’s needs. However, not every possible configuration has been qualified by Force Computers.

- See section 3.2.2 “Two-, Three-, and Four-Slot SPARC/CPU-20VT Configurations” on page 18 for general information on the mechanical differences and restrictions applying to the SPARC/CPU-20V concerning the usage of MBus modules.

- For information on qualified configurations, refer to the document *Board Reconfiguration with MBus Modules*. The document is delivered together with every MBus module purchased from Force Computers (also with every such module installed on a CPU or ESP board). If you are not sure whether the configuration you want to use has been qualified, ask your local Force Computers sales representative.

Solaris Support

To operate Solaris 2.5 on a SPARC/CPU-20V, do the following:

1. Remove the additional serial ports as described in see section 3.9 “Booting Solaris 2.5” on page 76.

2. See section 3.10 “Patching Solaris 2.5 for 200 MHz MBus Modules” on page 80 to patch Solaris. When patching and before booting from CD-ROM, you have to enter `d$-off` at the OpenBoot `ok` prompt.

204223 8 – 0 May 2000

3.2.4 Upgrading the SPARC/CPU-20VT

The SPARC/CPU-20VT can be upgraded to the user's needs:

Memory Modules

- The main memory capacity is adjustable via installation of the appropriate memory modules. The base board can hold one to four memory modules providing up to 512 MByte DRAM capacity. At least one memory module (MEM-20L 1) is required.

  There are two types of memory modules: MEM-20L for direct connection to the base board and MEM-20U for connection to a MEM-20L. Therefore, before installing a MEM-20U the respective MEM-20L has to be installed, first.

  Out of the extensive list of possible configurations the following memory module configurations have been qualified (others may be tested and qualified on request):

Table 4 Qualified Memory Module Configurations (All Data in MByte)

Total Capacity	**64**	**128**	**128**	**192**	**192**	**256**	**256**	**384**	**384**	**512**
MEM-20L 1 cap.	64	64	128	64	128	64	128	64	128	128
MEM-20L 2 cap.	–	64	–	64	64	64	128	64	128	128
MEM-20U 1 cap.	–	–	–	64	–	128	–	128	128	128
MEM-20U 2 cap.	–	–	–	–	–	–	–	128	–	128

The upgrading instructions are shipped together with the memory modules: see the *SPARC/MEM-20 Installation Guide*.

MBus Modules

- The processor performance is adjustable via installation of the appropriate MBus modules. The base board can hold one or two MBus modules. At least one is required. If two MBus modules are installed, both must be identical in frequency, number of processors and type, e.g. both being 150 MHz single hyperSPARC MBus modules.

  The upgrading instructions are shipped together with the MBus modules: see the *SPARC/MBus Module Installation Guide*.

Caution



Note that in case of a SPARC/CPU-20VT configuration using two dual 150 MHz hyperSPARC MBus modules (MBus-150H2/*x*) you have to

- **Use MBus-150H2/2 (with broader heat sink) as MBus module 1**
- **And MBus-150H2/1 (with narrower heat sink) as MBus module 2.**

Also note that in case of a SPARC/CPU-20VT variant using a 200 MHz hyperSPARC MBus module (e.g. a SPARC/CPU-20V) you might have to patch Solaris as described in section 3.10 “Patching Solaris 2.5 for 200 MHz MBus Modules” on page 80.

SBus Accessory Kit

- The I/O functionality is adjustable via installation of the appropriate SBus modules. The SBus I/O board can hold up to two industry-standard SBus modules.

  SBus modules enable the expansion of I/O interfaces, memory and processing performance with a wide range of solutions. The two SBus slots fully support SBus DMA transfers of SBus modules having DMA capabilities.

  If any SBus module is assembled, the SPARC/CPU-20VT occupies a fourth VMEbus slot. The SBus accessory kit available from Force Computers provides an additional front panel and a VME power module which may also be required due to the total power consumption (see “Power Supply” on page 12).

  The upgrading instructions are shipped together with the SBus accessory kit: see the *SPARC/CPU-20VT/SBus-AccKit Installation Guide.*

204223.8 – 0 May 2000

3.2.5 Major Components

Base Board

The base board is located in the first of four VMEbus slots the SPARC/CPU-20VT occupies. It provides:

- Two standard single height connectors for MBus modules, one for each module
- Six memory module connectors for up to four memory modules, three for each pair of lower and upper memory modules
- Three connectors for the SBus I/O board
- The following I/O interfaces: SCSI #1, Ethernet #1, floppy, keyboard and mouse as well as the two serial interfaces A+B

MBus Modules

The MBus modules are located in the second of four VMEbus slots the SPARC/CPU-20VT occupies. They hold the processors, including the cache controllers and the second-level cache (depending on the type of MBus module used). They are plugged into the MBus module connectors provided by the base board.

SBus I/O Board

The SBus I/O board is located in the third of four VMEbus slots the SPARC/CPU-20VT occupies. It holds the boot PROM devices and the RTC/NVRAM in three- or four-slot SPARC/CPU-20VT configurations. Additionally, it holds two standard single height SBus module connectors, a second SCSI (called SCSI #2), a second Ethernet (called Ethernet #2), a parallel interface and the two additional serial interfaces C+D. A part of the SBus I/O board is made of a flexible PCB that provides the connection to the base board.

The SBus I/O board also holds two temperature sensors which sense the temperature in the incoming and in the outgoing cooling air for the SPARC/CPU-20VT. The temperature sensors control the status of the two front-panel TEMP LEDs.

– Audio Fact. Opt.

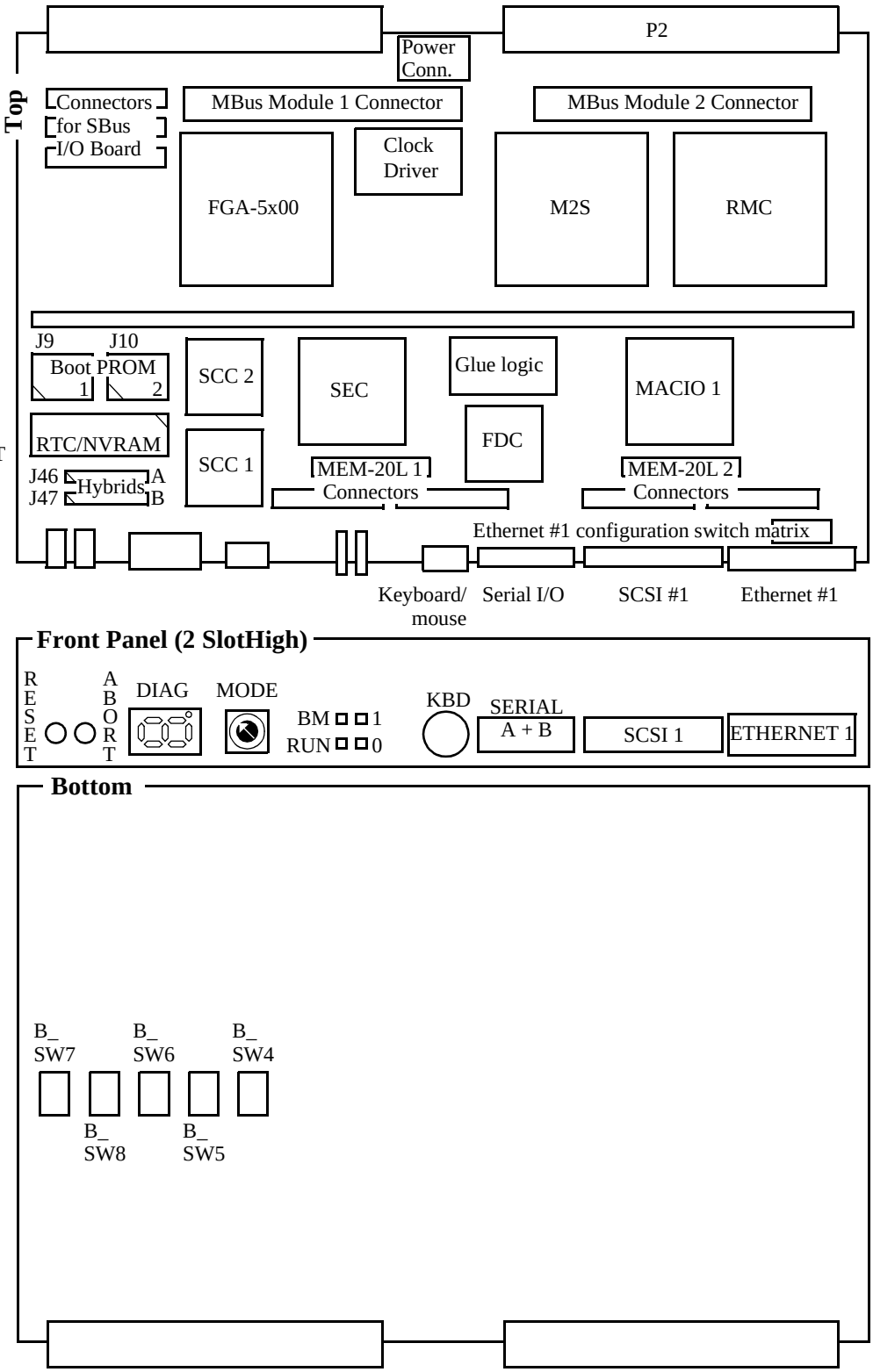
There is an SBus I/O board variant with installed audio factory option. For further information, see figure 10 “Location Diagr. of SBus I/O Board with Audio Factory Opt. (Schem.)” on page 32 and section 3.4.3 “Audio Connector Pinout (Factory Option)” on page 46.

SBus Modules

The SBus modules are located in the fourth of four VMEbus slots the SPARC/CPU-20VT occupies. They are plugged into the SBus module connectors provided by the SBus I/O board.

3.2.6 Location Diagrams

Figure 8 Location Diagram of the Base Board (Schematic)



Boot PROM and RTC/NVRAM are only assembled on the base board if operating the SPARC/CPU-20VT in a two-slot configuration.

204223 8 – 0 May 2000

Figure 9 **Location Diagram of the SBus I/O Board (Schematic)**

For a location diagram of the SBus I/O board variant with installed audio factory option see page 32.

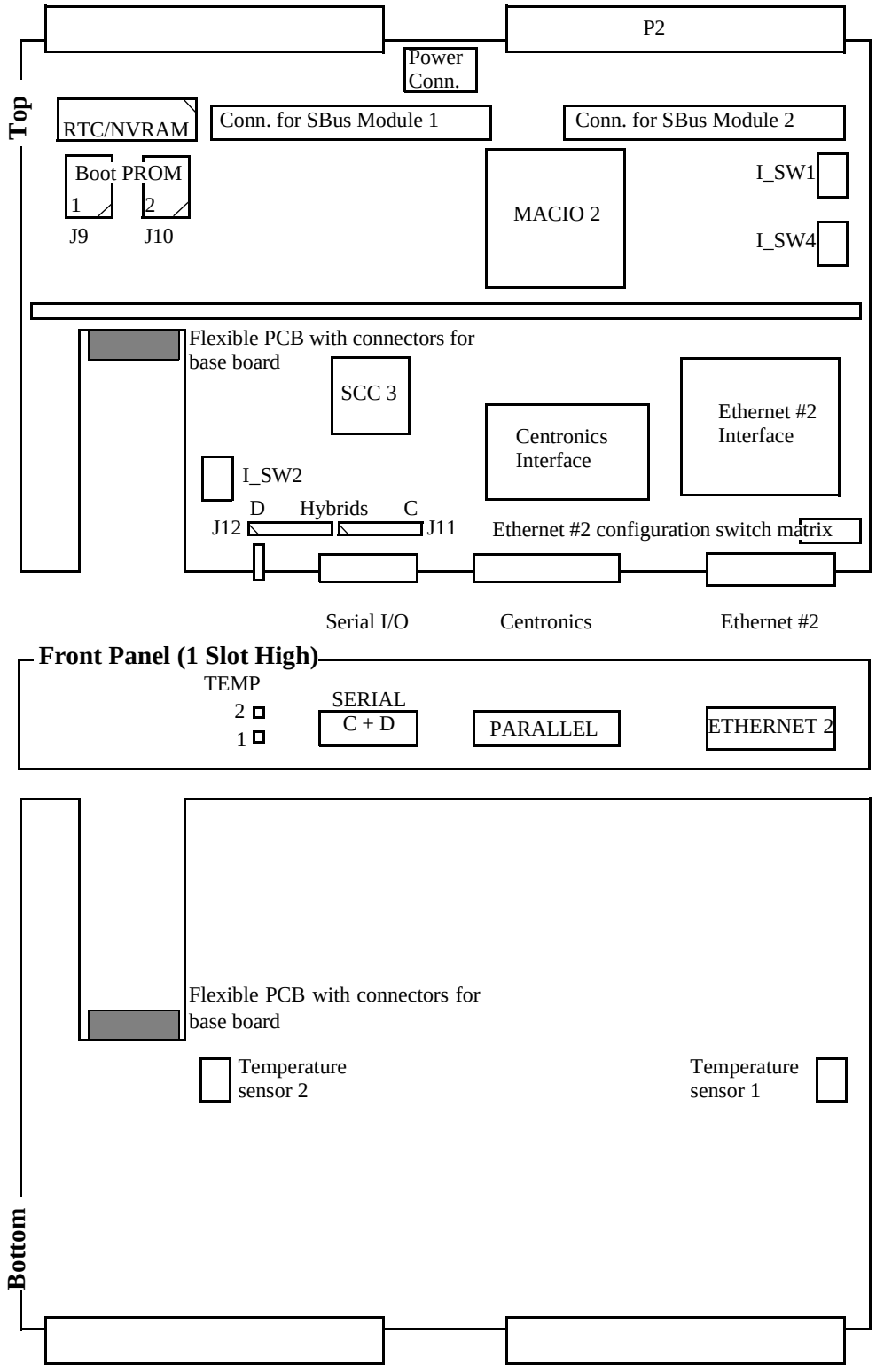
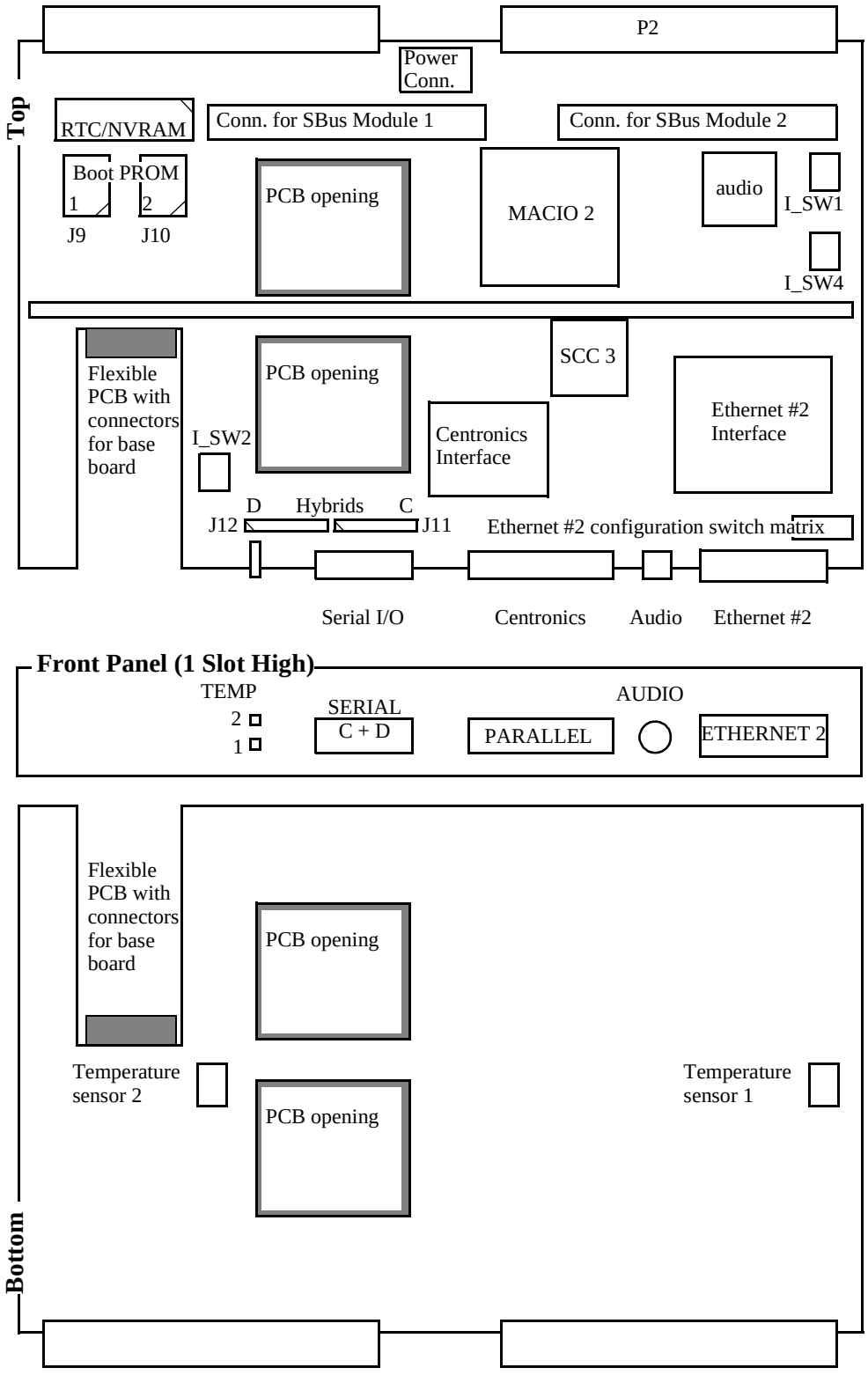


Figure 10 **Location Diagr. of SBus I/O Board with Audio Factory Opt. (Schem.)**
For a location diagram of the standard SBus I/O board see page 31.



204223 8 – 0 May 2000

3.3 Switch Settings

A particular switch name SW*x*-*x* in some cases applies to two switches: one located on the base board and one on the SBus I/O board. Therefore, this manual uses two prefixes: B_ for switches located on the base board and I_ for switches located on the SBus I/O board.

Additionally, the SPARC/CPU-20VT provides two distinct SCSI buses and two distinct Ethernet interfaces which can easily get mixed up. SCSI #1 and Ethernet #1 are located on the base board, whereas SCSI #2 and Ethernet #2 are located on the SBus I/O board.

Note: **Before changing a switch setting, ensure that you select the switch on the correct board by checking with the following two tables and the respective location diagram: see figure 8 "Location Diagram of the Base Board (Schematic)" on page 30 and figure 9 "Location Diagram of the SBus I/O Board (Schematic)" on page 31, respectively.**

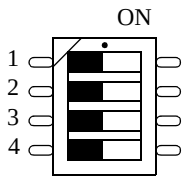
Note: **Before dealing with a SCSI bus or an Ethernet interface, ensure that you select the correct switches on the correct board.**

3.3.1 Base Board Switch Setting

Note: **To ensure correct switch setting follow the notes below:**

- For the correct setting of B_SW6-1 and B_SW6-2 see section 3.5 "SCSI Configuration" on page 56.
- B_SW8-*x* is located between B_SW7-*x* and B_SW6-*x* (see figure 8 "Location Diagram of the Base Board (Schematic)" on page 30).
- The following base board switches are reserved and must always be set to OFF: B_SW4-1, B_SW4-4, B_SW5-1, B_SW5-4, B_SW8-3, and B_SW8-4.

Table 5 Base Board Switch Settings

Name and Default Setting		Description
	B_SW4-1 OFF	reserved: must be OFF.
	B_SW4-2 OFF	SERIAL A configuration – signals of front-panel connector depending • on B_SW4-2 and B_SW4-3 setting • and on the hybrid installed at location J46: FH-002 for RS-232 support or FH-422T for RS-422 support Default: all switches OFF B_SW…Configuration 4-24-3 OFFOFF= RS-232 OFFOFF= RS-422: CTS+/– and RTS+/– available on conn. ONOFF= RS-422: RTXC+/– and RTS+/– available on conn. OFFON= RS-422: CTS+/– and TRXC+/– available on conn. ONON= RS-422: RTXC+/– and TRXC+/– available on conn.
	B_SW4-3 OFF	
	B_SW4-4 OFF	reserved: must be OFF.

204223 8 – 0 May 2000

Table 5 Base Board Switch Settings (cont.)

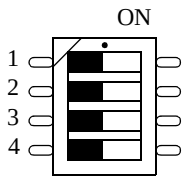
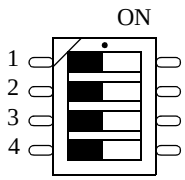
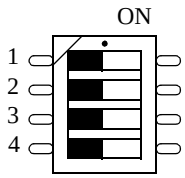
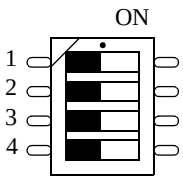
Name and Default Setting		Description
	B_SW5-1 OFF	Reserved: must be OFF.
	B_SW5-2 OFF	SERIAL B configuration – signals of front-panel connector depending • on B_SW5-2 and B_SW5-3 setting • and on the hybrid installed at location J47: FH-002 for RS-232 support or FH-422T for RS-422 support Default: all switches OFF B_SW…Configuration 5-25-3 OFFOFF= RS-232 OFFOFF= RS-422: CTS+/– and RTS+/– available on conn. ONOFF= RS-422: RTXC+/– and RTS+/– available on conn. OFFON= RS-422: CTS+/– and TRXC+/– available on conn. ONON= RS-422: RTXC+/– and TRXC+/– available on conn.
	B_SW5-3 OFF	
	B_SW5-4 OFF	Reserved: must be OFF.

Table 5 Base Board Switch Settings (cont.)

Name and Default Setting		Description
	B_SW6-1 OFF	Termination for front-panel SCSI #1 (see section 3.5 “SCSI Configuration” on page 56) OFF = automatic enabling or disabling of termination by sensing whether a SCSI cable is plugged in ON = disabled
	B_SW6-2 OFF	Termination for P2 SCSI #1 (see section 3.5 “SCSI Configuration” on page 56) OFF = disabled ON = enabled
	B_SW6-3 OFF	Abort key OFF = enabled ON = disabled
	B_SW6-4 OFF	Reset key OFF = enabled ON = disabled
	B_SW7-1 OFF	VMEbus slot 1 auto-detection OFF = enabled ON = disabled (also called manual mode, see B_SW7-2)
	B_SW7-2 OFF	VMEbus slot 1 manual mode: only available when B_SW7-1 = ON OFF = installed in VMEbus slot 1 ON = not installed in VMEbus slot 1
	B_SW7-3 OFF	VMEbus SYSRESET input OFF = enabled, i.e. VMEbus SYSRESET generates on-board reset ON = disabled
	B_SW7-4 OFF	VMEbus SYSRESET output OFF = enabled, i.e. on-board reset is driven to VMEbus ON = disabled

204223 8 – 0 May 2000

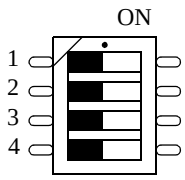
Table 5 Base Board Switch Settings (cont.)

Name and Default Setting		Description
	B_SW8-1 OFF	Boot PROM write protection OFF = write-protected ON = writing enabled
	B_SW8-2 OFF	User flash write protection OFF = write-protected ON = writing enabled
	B_SW8-3 OFF	Reserved: must be OFF.
	B_SW8-4 OFF	Reserved: must be OFF.

3.3.2 SBus I/O Board Switch Setting

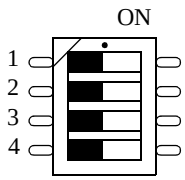
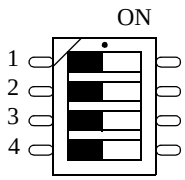
Note: **The following SBus I/O board switches are reserved and must always be set to OFF: I_SW1-1, I_SW2-1, I_SW2-4, I_SW4-1, I_SW4-2, I_SW4-3, I_SW4-4.**

Table 6 SBus I/O Board Switch Settings

Name and Default Setting		Description
	I_SW1-1 OFF	Reserved: must be OFF.
	I_SW1-2 OFF	SERIAL C configuration – signals of front-panel connector depending • on I_SW1-2 and I_SW1-3 setting • and on the hybrid installed in location J11: FH-002 for RS-232 support or FH-422T for RS-422 support Default: all switches OFF I_SW…Configuration 1-21-3 OFFOFF= RS-232 OFFOFF= RS-422: CTS+/– and RTS+/– available on conn. ONOFF= RS-422: RTXC+/– and RTS+/– available on conn. OFFON= RS-422: CTS+/– and TRXC+/– available on conn. ONON= RS-422: RTXC+/– and TRXC+/– available on conn.
	I_SW1-3 OFF	
	I_SW1-4 OFF	Termination for SCSI #2 OFF = enabled ON = disabled

204223 8 – 0 May 2000

Table 6 **SBus I/O Board Switch Settings (cont.)**

Name and Default Setting		Description
	I_SW2-1 OFF	Reserved: must be OFF.
	I_SW2-2 OFF	SERIAL D configuration – signals of front-panel connector depending • On I_SW2-2 and I_SW2-3 setting • On the hybrid installed in location J12: FH-002 for RS-232 support or FH-422T for RS-422 support Default: all switches OFF I_SW…Configuration 2-22-3 OFFOFF= RS-232 OFFOFF= RS-422: CTS+/– and RTS+/– available on conn. ONOFF= RS-422: RTXC+/– and RTS+/– available on conn. OFFON= RS-422: CTS+/– and TRXC+/– available on conn. ONON= RS-422: RTXC+/– and TRXC+/– available on conn.
	I_SW2-3 OFF	
	I_SW2-4 OFF	Reserved: must be OFF.
	I_SW4-1 OFF	Reserved: must be OFF.
	I_SW4-2 OFF	
	I_SW4-3 OFF	
	I_SW4-4 OFF	

3.4 Front Panel and Connectors

Front Panel

The features of the front panel are described in table 7 “Front Panel Features” on page 40. This table lists all front-panel connectors. For a location diagram see figure 8 “Location Diagram of the Base Board (Schematic)” on page 30 and figure 9 “Location Diagram of the SBus I/O Board (Schematic)” on page 31.

Connectors

In addition to the front-panel connectors, the SPARC/CPU-20VT provides on-board connectors for MBus modules (on base board), for SBus modules (on SBus I/O board) and for the VMEbus (on base and SBus I/O board). An overview – including the front-panel connectors – is shown in table 8 “SPARC/CPU-20VT Connectors” on page 43.

Furthermore, the IOBP-10 provides more connectors which provide the SPARC/CPU-20VT signals available on the base board’s or on the SBus I/O board’s VMEbus P2 connector (see section 3.4.6 “IOBP-10” on page 49).

Table 7 Front Panel Features

Device	Description
RESET (base board)	Mechanical reset key: When enabled and toggled it instantaneously affects the SPARC/CPU-20VT by generating a reset. The reset key is activated in UP position. For information on enabling the key, see “B_SW6-4” on page 36.
ABORT (base board)	Mechanical abort key: When enabled and toggled it instantaneously affects the SPARC/CPU-20VT by generating a nonmaskable interrupt request to the processor. The abort key is activated in UP position. The abort key interrupt logic is implemented by the FGA-5x00 (refer to the device’s *Reference Guide)*. In addition to the FGA-5x00 interrupt control logic, the abort key interrupt can be enabled manually (see “B_SW6-3” on page 36).
DIAG (base board)	Software programmable hexadecimal display for diagnostics.
MODE (base board)	Hexadecimal rotary switch, decoded with 4 bit. Default setting: F_{16}.

204223 8 – 0 May 2000

Table 7 Front Panel Features (cont.)

Device	Description
RUN (Base Board)	CPU status: Green Normal operation Red The processor is halted or reset is active; it starts blinking to signal a hangup of the SPARC/CPU-20VT.
BM (Base Board)	VME busmaster and SYSFAIL LED: Green If the SPARC/CPU-20VT accesses the VMEbus as VMEbus master Red If SYSFAIL is asserted from the FGA-5x00 to the VMEbus Off Otherwise
0, 1 (Base Board)	Two software programmable user LEDs. Possible status: off, red, yellow, or green, all colors either permanent or with a blinking frequency of approximately 0.5, 1, or 2 Hz.

Table 7 Front Panel Features (cont.)

Device	Description
TEMP 1, 2 (SBus I/O b.)	Two temperature LEDs showing the status of the two temperature sensors on the SBus I/O board. The two temperature sensors do not sense the temperature of the respective MBus module but rather sense the temperature of the incoming and outgoing airflow. Each of the two temperature sensors may sense a temperature above the upper, below the lower limit or within the acceptable temperature range. Upper and lower limit are predefined but may be altered by the user, for example, by the respective commands from the OpenBoot command set enhanced by Force Computers. The upper and lower limit temperatures can be set independently for each of the two temperature sensors within the range of –55 °C … +125 °C. Possible LED status: Red The measured temperature is above the upper limit. Green The measured temperature is between upper and lower limit. Off The measured temperature is below the lower limit. Predefined limits for temperature sensor 1 Upper limit: 80 °C; lower limit: 10 °C Predefined limits for temperature sensor 2 Upper limit: 90 °C; lower limit: 10 °C
KBD (Base Board)	Standard eight-pin mini DIN connector for keyboard and mouse (see section 3.4.2 “Keyboard/Mouse Connector Pinout” on page 46).
SERIAL A+B (Base Board)	26-pin shielded connector, each for 2 serial interfaces (see section 3.4.1 “Serial I/O Port RS-232 and RS-422 Connector Pinout” on page 44).
SERIAL C+D (SBus I/O b.)	
SCSI 1 (Base Board)	Standard SCSI 50-pin fine pitch connector.
PARALLEL (SBus I/O b.)	Standard 25-pin D-Sub connector for Centronics parallel interface.

204223 8 – 0 May 2000

Table 7 **Front Panel Features (cont.)**

Device	Description
AUDIO (factory option on SBus I/O Board)	Standard eight-pin mini DIN connector for audio interface (see section 3.4.3 “Audio Connector Pinout (Factory Option)” on page 46).
ETHERNET 1 (Base Board)	Standard 15-pin AUI Ethernet connector for thick-wire Ethernet (802.3/10base5).
ETHERNET 2 (SBus I/O b.)	

Table 8 **SPARC/CPU-20VT Connectors**

Connector Location and Function		Connector Type and Sample Manufacturer Part Number
Base-Board Front Panel	Keyboard / mouse	eight-pin mini DIN: AMP 749232-1
	Serial A + B	26-pin fine pitch: AMP 749831-2
	SCSI #1	50-pin fine pitch: AMP 749831-5
	Ethernet #1	15-pin D-Sub: AMP 747845-4
SBus-I/O-Board Front Panel	Serial C + D	26-pin fine pitch: AMP 749831-2
	Parallel	25-pin D-Sub: AMP 747846-4
	Audio	eight-pin mini DIN: AMP 749232-1
	Ethernet #2	15-pin D-Sub: AMP 747845-4
P1 on Base Board and SBus I/O boaRd	VMEbus P1	96-pin VG: ITT CANNON G60M096P3BEBM3D
P2 on Base Board and SBus I/O Board	VMEbus P2	
P7 on Base Board	Mbus mod. 2	Standard connectors for MBus Modules
P8 on Base Board	MBus mod. 1	

Table 8 **SPARC/CPU-20VT Connectors (cont.)**

Connector Location and Function		Connector Type and Sample Manufacturer Part Number
P7 on SBus I/O Board	SBus mod. 1	96-pin through-hole: PSC-96FD2
P8 on SBus I/O Board	SBus mod. 2	

3.4.1 Serial I/O Port RS-232 and RS-422 Connector Pinout

All four serial I/O ports of the SPARC/CPU-20VT are independent full-duplex channels. For each of them the four signals RXD, TXD, RTS, and CTS are also provided via the respective VMEbus P2 connector (for ports A and B see section 3.4.4 “VME P2 Connector Pinout of the Base Board” on page 47 and for ports C and D see section 3.4.5 “VME P2 Connector Pinout of the SBus I/O Board” on page 48).

Additionally, there are two front-panel serial connectors:

SERIAL A+B
- SERIAL A+B on the base board’s front panel holds the signals for the two serial ports A and B

SERIAL C+D
- SERIAL C+D on the SBus I/O board’s front panel holds the signals for the two serial ports C and D

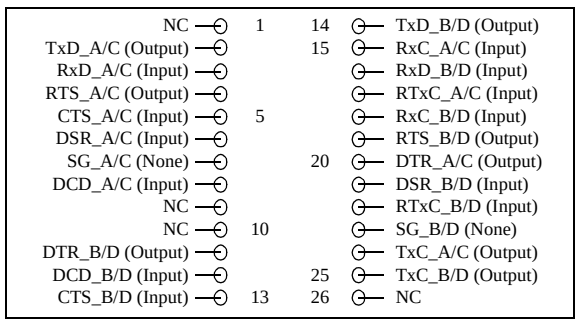
Caution



Both connectors are 26-pin shielded connectors compatible to the CPU-5V and CPU-5VT. Therefore, they have the same pinout if you remembered that

- **Port A on SERIAL A+B and port C on SERIAL C+D use the same pins for the same signals.**
- **Port B on SERIAL A+B and port D on SERIAL C+D use the same pins for the same signals.**

Figure 11 **RS-232: Serial A+B and C+D Front-Panel Connector Pinout**



204223 8 – 0 May 2000

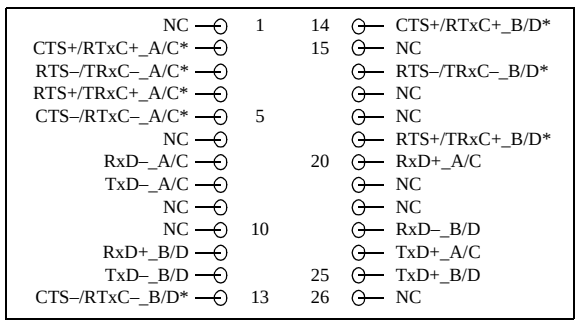
RS-232 (Default) or RS-422

Easy serial I/O port configuration is provided by installing the appropriate hybrid module: FH-002 for RS-232 (factory default) or FH-422T for RS-422. To change the configuration, insert the respective hybrid in the respective socket for:

- Port A in the base board's socket J46
- Port B in the base board's socket J47
- Port C in the SBus I/O board's socket J11
- Port D in the SBus I/O board's socket J12

For information on the location and orientation (pin 1 location) of the hybrids, see figure 8 "Location Diagram of the Base Board (Schematic)" on page 30 and figure 9 "Location Diagram of the SBus I/O Board (Schematic)" on page 31, respectively.

Figure 12 **RS-422: Serial A+B and C+D Front-Panel Connector Pinout**



* signal depending on switch setting: see B_SW4-2, B_SW4-3, B_SW5-2, B_SW5-3, I_SW1-2, I_SW1-3, I_SW2-2, and I_SW2-3 in section 3.3 "Switch Settings" on page 33.

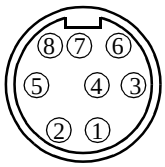
Table 9 **Serial Ports Signal Naming**

Signal	Description	Signal	Description
TxD	Transmit data	RxD	Receive data
RTS	Request to send	CTS	Clear to send
TRxC	Transmit clock	RTxC	Receive clock
DSR	Data set ready	DTR	Data terminal ready
TxC	Transmit clock: DCE source	RxC	Receive clock
SG	Signal ground	DCD	Data carrier detect

3.4.2 Keyboard/Mouse Connector Pinout

The keyboard and mouse port is available on the front panel via an eight-pin mini DIN connector. The signals KBDIN, KBDOUT, and MOUSEIN are additionally available on the base board's P2 connector.

Table 10 Keyboard/Mouse Connector Pinout

Pin		Function
	1	GND
	2	GND
	3	+5VDC
	4	MOUSEIN, Mouse in
	5	KBDOUT, Keyboard out
	6	KBDIN, Keyboard in
	7	MOUSEOUT, Mouse out
	8	+5VDC

3.4.3 Audio Connector Pinout (Factory Option)

There is an SBus I/O board variant with installed audio factory option supporting:

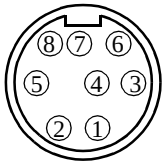
- Two audio inputs for one passive microphone (Microphone+ and Microphone–)
- One dc-free output (> 540 Ω impedance and < 100 pF capacitive load) for one earpiece
- Two differential push-pull outputs (> 40 Ω impedance and < 100 pF capacitive load) for one loud speaker

For a location diagram of the SBus I/O board variant with installed audio factory option see page 32.

The microphone and the earpiece port are available on the front panel via an eight-pin mini DIN connector. The loud speaker signals are available on the P2 connector of the SBus I/O board (see SPKR+/– = SPKR1 and SPKR–/+ = SPKR2 in section 3.4.5 "VME P2 Connector Pinout of the SBus I/O Board" on page 48).

204223 8 – 0 May 2000

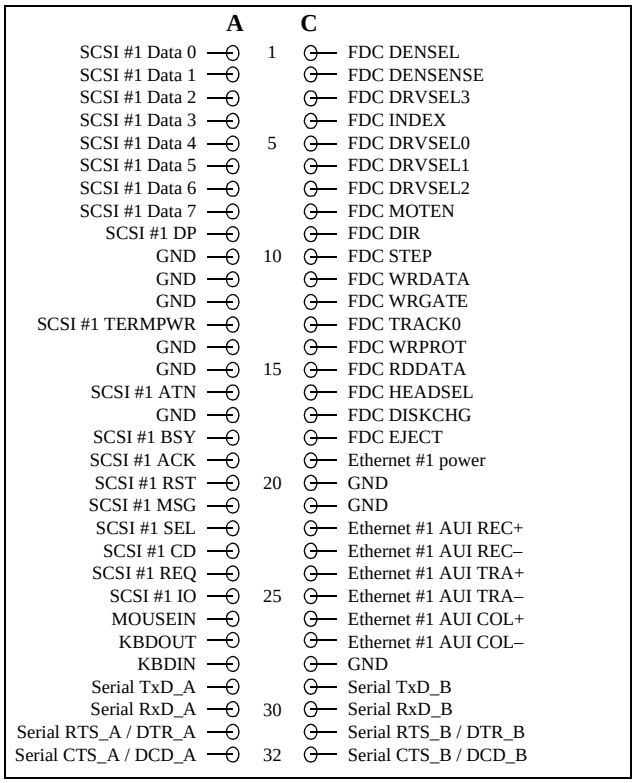
Table 11 Audio Connector Pinout

Pin		Function
	1	GND
	2	GND
	3	Microphone–
	4	NC
	5	NC
	6	Microphone+
	7	GND
	8	Earpiece

3.4.4 VME P2 Connector Pinout of the Base Board

The pinout shown in the figure below applies to RS-232 configuration of the base board's serial I/O ports. Serial I/O ports configured for RS-422 are only available on the front panel.

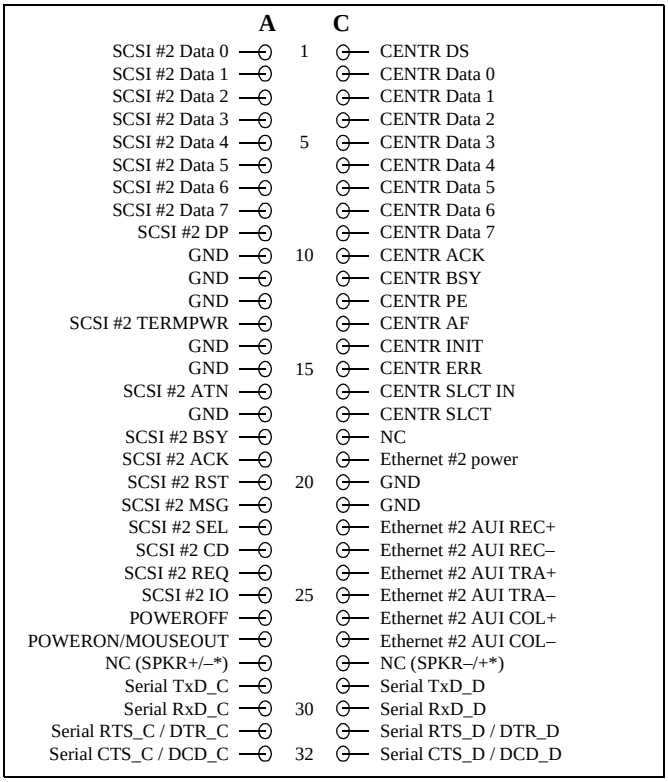
Figure 13 Base Board P2 Conn. Pinout with Serial I/O Config. for RS-232



3.4.5 VME P2 Connector Pinout of the SBus I/O Board

The pinout shown in the figure below applies to RS-232 configuration of the SBus I/O board's serial I/O ports. Serial I/O ports configured for RS-422 are only available on the front panel.

Figure 14 SBus I/O Board P2 Conn. Pinout with Serial I/O Config. for RS-232



* = only available with installed audio factory option (see section 3.4.3 "Audio Connector Pinout (Factory Option)" on page 46)

204223 8 – 0 May 2000

3.4.6 IOBP-10

As a separate price list item Force Computers offers a SPARC/IOBP-10 I/O panel which is plugged into the VMEbus backplane from its rear. The panel enables easy connection to the I/O signals of a SPARC/CPU-20VT which are available on one of the P2 connectors.

Danger



The SPARC/IOBP-10 is especially designed for the SPARC/CPU-20VT. Do not use any other I/O panels on the SPARC/CPU-20VT.

Features of the I/O Panel

The SPARC/IOBP-10 is an I/O panel on VMEbus P2 with flat cable connectors for SCSI, serial, Centronics compliant parallel and floppy interface. The SPARC/IOBP-10 also has a micro D-Sub connector for an Ethernet interface. Two of the I/O panels can be used together with the SPARC/CPU-20VT: one I/O panel to interface to the VMEbus P2 connector of the base board and the other one to the VMEbus P2 connector of the SBus I/O board.

Base Board I/O Panel

The following interfaces are available on the I/O panel connected to the base board via the VMEbus backplane:

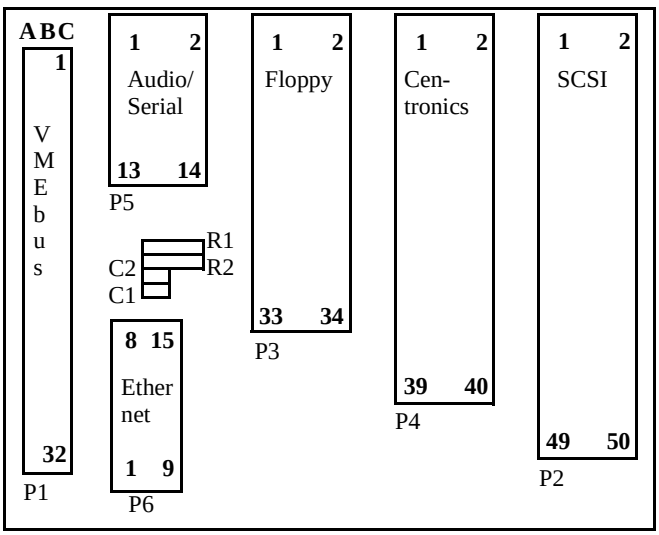
- SCSI #1, Ethernet #1, floppy interface, serial ports A and B as well as keyboard/mouse

SBus I/O Board I/O Panel

The following interfaces are available on the I/O panel connected to the SBus I/O board via the VMEbus backplane:

- SCSI #2, Ethernet #2, Centronics interface, serial ports C and D

Figure 15 **IOBP-10 (Schematic)**



The pinouts of the P1…6 connectors are shown in the following tables.

204223 8 – 0 May 2000

Table 12 IOBP-10 P1 Pinout

ROW A	Signal When IOBP-10 is Used on Base Board	Signal When IOBP-10 is Used on SBus I/O board	ROW C	Signal When Iobp-10 Is Used on Base Board	Signal When IOBP-10 is Used on SBus I/O Board
1	SCSI #1 Data 0	SCSI #2 Data 0	1	FDC DENSEL	CENTR DS
2	SCSI #1 Data 1	SCSI #2 Data 1	2	FDC DENSENSE	CENTR Data 0
3	SCSI #1 Data 2	SCSI #2 Data 2	3	FDC DRVSEL3	CENTR Data 1
4	SCSI #1 Data 3	SCSI #2 Data 3	4	FDC INDEX	CENTR Data 2
5	SCSI #1 Data 4	SCSI #2 Data 4	5	FDC DRVSEL0	CENTR Data 3
6	SCSI #1 Data 5	SCSI #2 Data 5	6	FDC DRVSEL1	CENTR Data 4
7	SCSI #1 Data 6	SCSI #2 Data 6	7	FDC DRVSEL2	CENTR Data 5
8	SCSI #1 Data 7	SCSI #2 Data 7	8	FDC MOTEN	CENTR Data 6
9	SCSI #1 DP	SCSI #2 DP	9	FDC DIR	CENTR Data 7
10	GND	GND	10	FDC STEP	CENTR ACK
11	GND	GND	11	FDC WRDATA	CENTR BSY
12	GND	GND	12	FDC WRGATE	CENTR PE
13	SCSI #1 TERMPWR	SCSI #2 TERMPWR	13	FDC TRACK0	CENTR AF
14	GND	GND	14	FDC WRPROT	CENTR INIT
15	GND	GND	15	FDC RDDATA	CENTR ERR
16	SCSI #1 ATN	SCSI #2 ATN	16	FDC HEADSEL	CENTR SLCT IN
17	GND	GND	17	FDC DISKCHG	CENTR SLCT
18	SCSI #1 BSY	SCSI #2 BSY	18	FDC EJECT	N.C.
19	SCSI #1 ACK	SCSI #2 ACK	19	Ethernet #1 power	Ethernet #2 power
20	SCSI #1 RST	SCSI #2 RST	20	GND	GND
21	SCSI #1 MSG	SCSI #2 MSG	21	GND	GND
22	SCSI #1 SEL	SCSI #2 SEL	22	Ethernet #1 REC+	Ethernet #2 REC+
23	SCSI #1 CD	SCSI #2 CD	23	Ethernet #1 REC–	Ethernet #2 REC–
24	SCSI #1 REQ	SCSI #2 REQ	24	Ethernet #1 TRA+	Ethernet #2 TRA+
25	SCSI #1 IO	SCSI #2 IO	25	Ethernet #1 TRA–	Ethernet #2 TRA–
26	MOUSEIN	POWEROFF	26	Ethernet #1 COL+	Ethernet #2 COL+
27	KBDOUT	POWERON/MOUSEOUT	27	Ethernet #1 COL–	Ethernet #2 COL–
28	KBDIN	N.C. (SPKR+/–*)	28	GND	N.C. (SPKR–/+*)
29	TxD Port A	TxD Port C	29	TxD Port B	TxD Port D
30	RxD Port A	RxD Port C	30	RxD Port B	RxD Port D
31	RTS Port A / DTR Port A	RTS Port C / DTR Port C	31	RTS Port B / DTR Port B	RTS Port D / DTR Port D
32	CTS Port A / DCD Port A	CTS Port C / DCD Port C	32	CTS Port B / DCD Port B	CTS Port D / DCD Port D

Table 12 IOBP-10 P1 Pinout (cont.)

ROW A	Signal When IOBP-10 is Used on Base Board	Signal When IOBP-10 is Used on SBus I/O board	ROW C	Signal When Iobp-10 Is Used on Base Board	Signal When IOBP-10 is Used on SBus I/O Board
* = only available with installed audio factory option (see section 3.4.3 “Audio Connector Pinout (Factory Option)” on page 46)					

Table 13 IOBP-10 P2 Pinout (SCSI, Standard 50-pin Flat Cable Conn.)

Pin	Signal	Pin	Signal
1	GND	2	SCSI Data 0
3	GND	4	SCSI Data 1
5	GND	6	SCSI Data 2
7	GND	8	SCSI Data 3
9	GND	10	SCSI Data 4
11	GND	12	SCSI Data 5
13	GND	14	SCSI Data 6
15	GND	16	SCSI Data 7
17	GND	18	SCSI DP
19	GND	20	GND
21	GND	22	GND
23	GND	24	GND
25	N.C.	26	SCSI TERMPWR
27	GND	28	GND
29	GND	30	GND
31	GND	32	SCSI ATN
33	GND	34	GND
35	GND	36	SCSI BSY
37	GND	38	SCSI ACK
39	GND	40	SCSI RST
41	GND	42	SCSI MSG
43	GND	44	SCSI SEL
45	GND	46	SCSI CD
47	GND	48	SCSI REQ
49	GND	50	SCSI IO

204223 8 – 0 May 2000

Table 14 IOBP-10 P3 Pinout (Floppy, Standard 34-pin Flat Cable Conn.)

Pin	Signal	Pin	Signal
1	FDC EJECT	2	FDC DENSEL
3	GND	4	FDC DENSENSE
5	GND	6	FDC DRVSEL3
7	GND	8	FDC INDEX
9	GND	10	FDC DRVSEL0
11	GND	12	FDC DRVSEL1
13	GND	14	FDC DRVSEL2
15	GND	16	FDC MOTEN
17	GND	18	FDC DIR
19	GND	20	FDC STEP
21	GND	22	FDC WRDATA
23	GND	24	FDC WRGATE
25	GND	26	FDC TRACK0
27	GND	28	FDC WRPROT
29	GND	30	FDC RDDATA
31	GND	32	FDC HEADSEL
33	GND	34	FDC DISKCHG

Table 15 IOBP-10 P4 Pinout (Centr., Stand. 40-pin Flat Cable Conn.)

Pin	Signal	Pin	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

Table 15 IOBP-10 P4 Pinout (Centr., Stand. 40-pin Flat Cable Conn.) (cont.)

Pin	Signal	Pin	Signal
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 16 IOBP-10 P5 Pinout (Serial / Audio)

Pin	Signal When Used on Base Board	Signal When Used on SBus I/O Board	Pin	Signal When used on Base Board	Signal When Used on SBus I/O Board
1	GND	N.C. (SPKR–/+*)	2	KBDOUT	POWERON / MOUSEOUT
3	MOUSEIN	POWEROFF	4	KBDIN	N.C. (SPKR+/–*)
5	TxD Port B	TxD Port D	6	TxD Port A	TxD Port C
7	RxD Port B	RxD Port D	8	RxD Port A	RxD Port C
9	RTS Port B / DTR Port B	RTS Port D / DTR Port D	10	RTS Port A / DTR Port A	RTS Port C / DTR Port C
11	CTS Port B / DCD Port B	CTS Port D / DCD Port D	12	CTS Port A / DCD Port A	CTS Port C / DCD Port C
13	GND	GND	14	GND	GND

* = only available with installed audio factory option
(see section 3.4.3 “Audio Connector Pinout (Factory Option)” on page 46)

204223 8 – 0 May 2000

Table 17 IOBP-10 P6 Pinout (Ethernet, Standard 15-pin Micro D-Sub Conn.)

Pin	Description
1	GND
2	Ethernet COL+
3	Ethernet TRA+
4	GND
5	Ethernet REC+
6	GND
7	N.C.
8	N.C.
9	Ethernet COL-
10	Ethernet TRA-
11	GND
12	Ethernet REC-
13	Ethernet power
14	GND
15	N.C.

3.5 SCSI Configuration

Caution



The SPARC/CPU-20VT provides two distinct SCSI buses, which can easily get mixed up. Therefore:

- **Before dealing with a SPARC/CPU-20VT SCSI bus, ensure that you select the correct board:**
  - **– SCSI #1 is located on the base board – associated to the MACIO 1 controller.**
  - **– SCSI #2 is located on the SBus I/O board – associated to the MACIO 2 controller.**

3.5.1 SCSI #1 Termination

The base board's SCSI #1 bus is accessible via the base board's front-panel SCSI #1 connector and via the base board's P2 connector. Therefore, the base board holds two distinct SCSI bus terminations to enable correct termination of the SCSI #1 bus. Associated to the two terminations there are two switches – B_SW6-1 and B_SW6-2 – which allow easy selection of a valid SCSI #1 bus configuration.

There are four valid base board switch settings corresponding to valid SCSI #1 bus configurations. The following factors differentiate the valid SCSI #1 bus configurations:

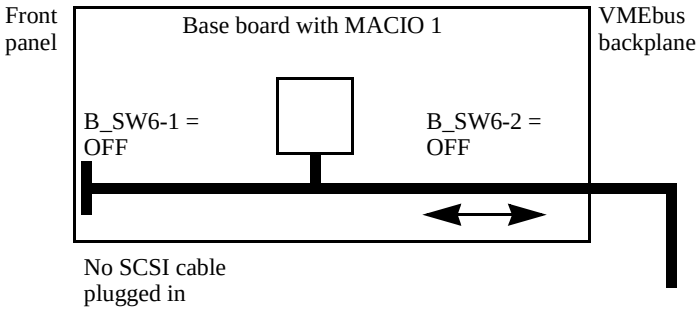
- The base board's location within the SCSI #1 bus: Is the base board located at an endpoint of the SCSI #1 bus?
- The connector(s) being used from the SCSI #1 bus:
  - – Is a SCSI cable plugged into the base board's front-panel SCSI connector?
  - – Is the base board's VMEbus P2 connector used by the SCSI #1 bus?
  - – Are both base-board connectors used by the SCSI #1 bus?

Each of the following configuration descriptions starts with identifying the SCSI #1 bus configuration being covered and ends with defining the correct switch setting corresponding to the configuration under consideration.

204223 8 – 0 May 2000

Default Configuration 1

- The default configuration 1 is covered by the default switch setting: The base board is located at an endpoint of the SCSI #1 bus, the SCSI #1 bus is extended via the VMEbus P2 connector, but no SCSI cable is plugged into the front-panel SCSI connector:



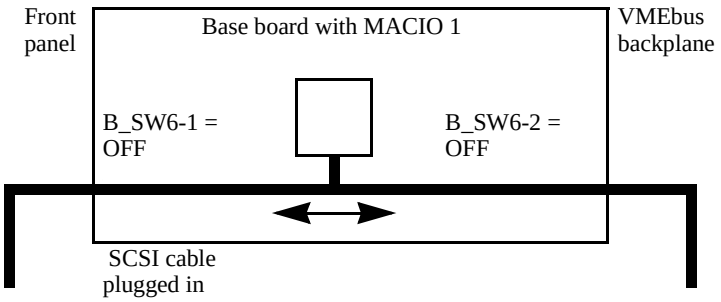
In this configuration:

– B_SW6-1 must be set to OFF = automatic enabling or disabling of termination by sensing whether a SCSI cable is plugged in (default “OFF”, see page 36).

– B_SW6-2 must be set to OFF = disabled (default “OFF”, see page 36).

Default Configuration 2

- The default configuration 2 is also covered by the default switch setting: the base board is not located at an endpoint of the SCSI #1 bus, the SCSI 1 bus is extended via the VMEbus P2 connector and via the front-panel SCSI connector:



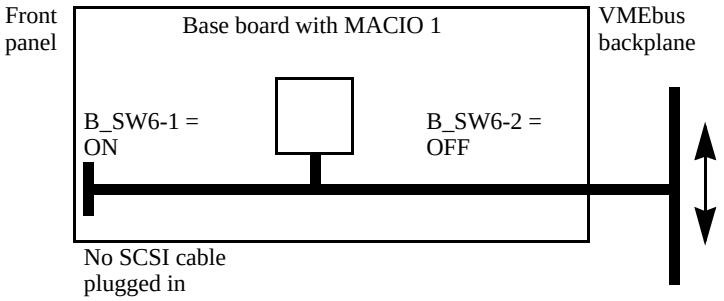
In this configuration:

– B_SW6-1 must be set to OFF = automatic enabling or disabling of termination by sensing whether a SCSI cable is plugged in (default “OFF”, see page 36).

– B_SW6-2 must be set to OFF = disabled (default “OFF”, see page 36).

Alternative 1

- Alternative configuration 1: the base board is not located at an endpoint of the SCSI #1 bus, the SCSI #1 bus is accessed via the VMEbus P2 connector, but no SCSI cable is plugged into the front-panel SCSI connector:



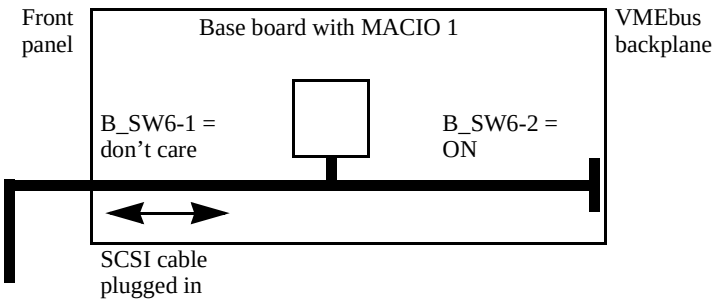
In this configuration

– B_SW6-1 must be set to ON = disabled termination (default “OFF”, see page 36).

– B_SW6-2 must be set to OFF = disabled termination (default “OFF”, see page 36).

Alternative 2

- Alternative configuration 2: the base board is located at an endpoint of the SCSI #1 bus and the VMEbus P2 connector is not used for SCSI #1 bus signalling, but the SCSI #1 bus is extended via the front-panel connector:



In this configuration

– Both settings of B_SW6-1 are valid.

– B_SW6-2 must be set to ON = enabled (default “OFF”, see page 36).

Since in this configuration the SCSI #1 bus is extended via the front-panel connector setting B_SW6-1 to ON = disabled termination reflects this configuration explicitly.

204223 8 – 0 May 2000

3.5.2 SCSI #2 Termination

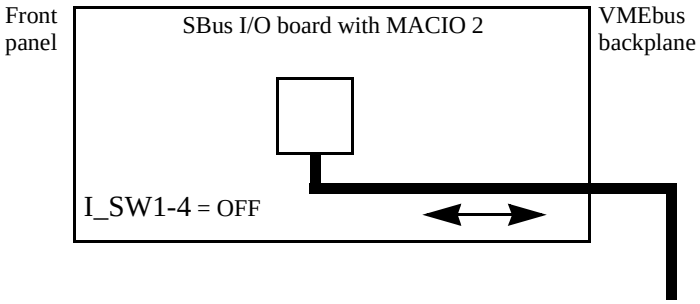
The SBus I/O board's SCSI #2 bus is only available at the SBus I/O board's VMEbus P2 connector. It is terminated – i.e. the SBus I/O board is at one endpoint of the SCSI #2 bus – if I_SW1-4 is set appropriately: OFF = enabled termination (default "OFF", see page 38).

Valid Configurations

There are two valid SBus I/O board switch settings corresponding to valid SCSI #2 bus configurations:

Default Configuration

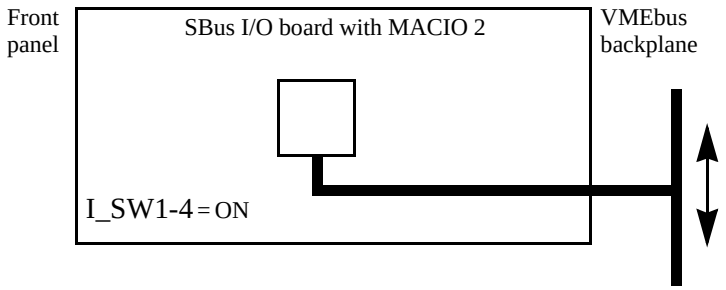
- The default configuration is covered by the default switch setting: The SBus I/O board is located at an endpoint of the SCSI #2 bus, i.e., the SCSI #2 bus is extended via the VMEbus P2 connector:



In this configuration I_SW1-4 must be set to OFF = enabled termination (default "OFF", see page 38).

Alternative

- Alternative configuration: the SBus I/O board is not located at an endpoint of the SCSI #2 bus, i.e., the SCSI #2 bus is accessed via the VMEbus P2 connector:



In this configuration I_SW1-4 must be set to ON = disabled termination (default "OFF", see page 38).

3.6 Ethernet Configuration

The SPARC/CPU-20VT provides two distinct Ethernet interfaces, which use the same Ethernet address and which can easily get mixed up. Therefore:

Caution



- **Before dealing with a SPARC/CPU-20VT Ethernet interface, ensure that you select the correct board:**
  - **Ethernet #1 is located on the base board.**
  - **Ethernet #2 is located on the SBus I/O board.**
- **Never connect the SPARC/CPU-20VT to the same Ethernet network via the Ethernet #1 interface and via the Ethernet #2 interface.**

Either Front Panel or P2

Each of the Ethernet interfaces can be configured to be available:

- Via the front-panel 15-pin D-Sub connector
- Via the VMEbus P2 connector

Note: **Never use the front-panel Ethernet connector, if the Ethernet interface is configured to be available via the VMEbus P2 connector – regardless of whether you use the Ethernet #1 or the Ethernet #2 interface.**

Configuration

Two eight-pin configuration switch matrices control if the Ethernet interface is available via the front-panel Ethernet connector or via the VMEbus P2 connector:

Ethernet #1

- The base board's switch matrix controls the Ethernet #1 configuration:
  - Ethernet #1 is available on the base board's front-panel Ethernet connector if the switch matrix located on the base board is plugged into B5 and B6 – this is the default configuration.
  - Ethernet #1 is available on the base board's VMEbus P2 connector if the switch matrix is plugged into B4 and B5.

Ethernet #2

- The SBus I/O board's switch matrix controls the Ethernet #2 configuration:
  - Ethernet #2 is available on the SBus I/O board's front-panel Ethernet connector if the switch matrix is plugged into B5 and B6 – this is the default configuration.

204223 8 – 0 May 2000

– Ethernet #2 is available on the SBus I/O board's VMEbus P2 connector if the switch matrix is plugged into B6 and B7.

Figure 16 Ethernet #1 Configuration via the Base-Board Switch Matrix

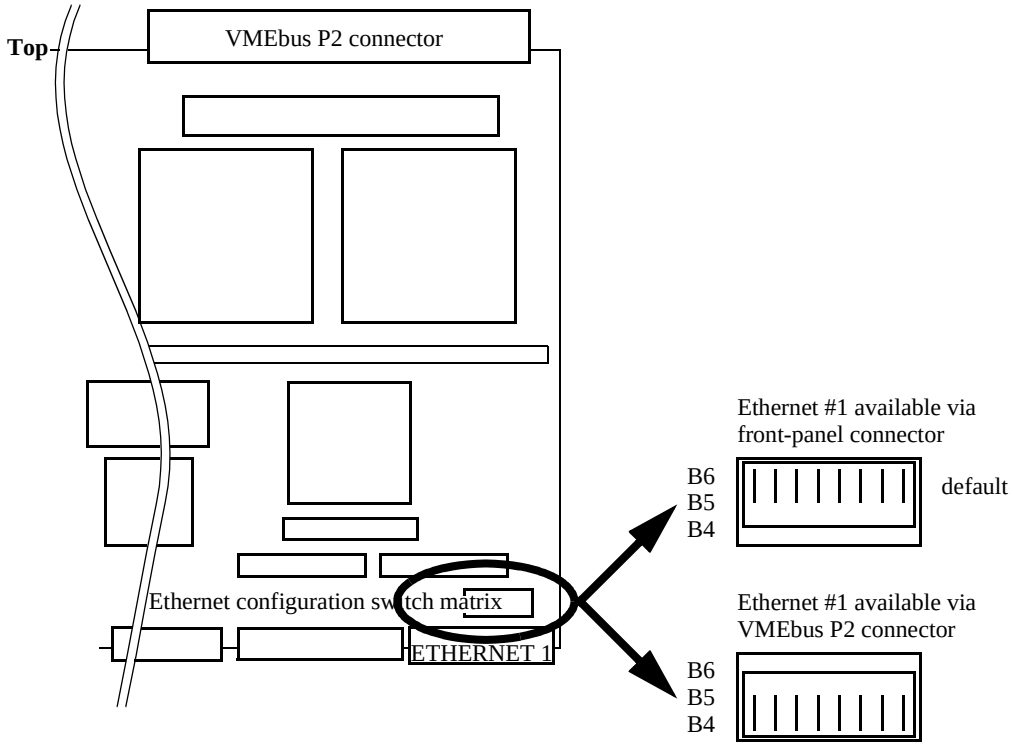
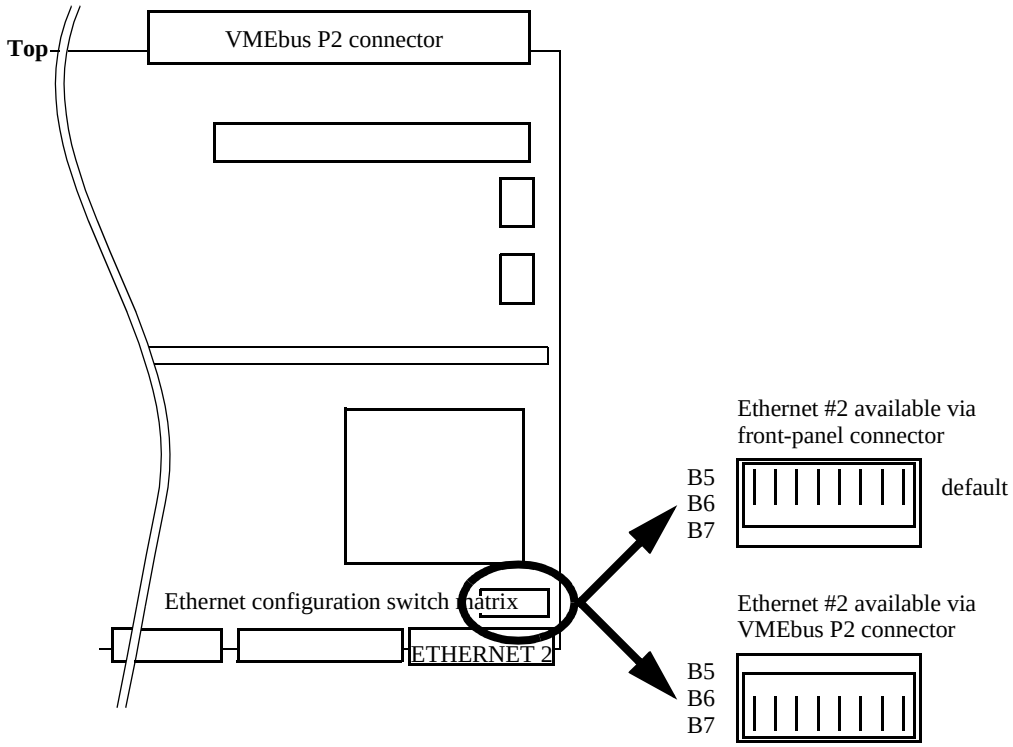


Figure 17 Ethernet #2 Configuration via the SBus-I/O-Board Switch Matrix



3.6.1 Ethernet Address and Host ID

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

```
ok banner
```


204223 8 – 0 May 2000

This section explains how to determine the SPARC/CPU-20VT Ethernet address and its host ID.

Figure 18 Ethernet Address

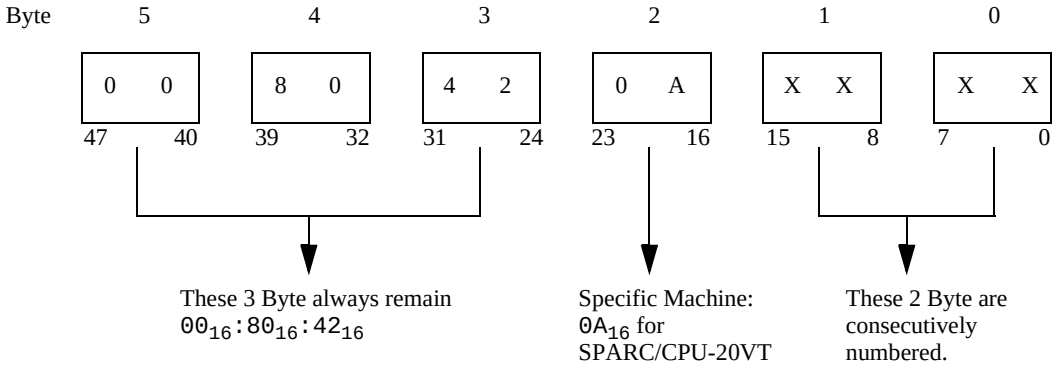
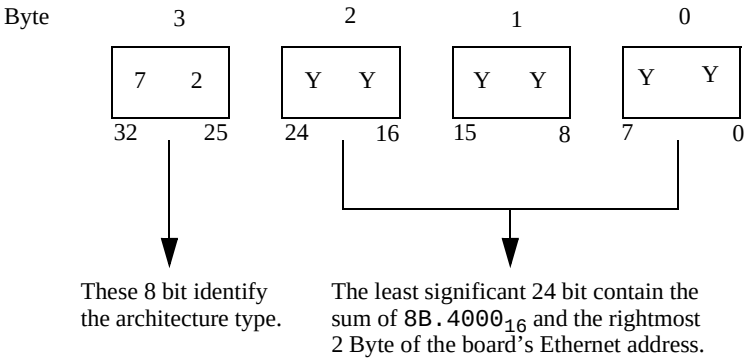


Figure 19 Host ID



3.7 Powering Up

The initial powering up can easily be done by connecting a terminal to the front-panel serial I/O port A. The advantage of using a terminal is that you do not need any frame buffer, monitor, or keyboard for initial powering up.

VME System Controller

If configured appropriately, the SPARC/CPU-20VT recognizes automatically whether it is plugged in slot one of the VMEbus backplane or in any other slot. This auto-configuration feature requires that B_SW7-1 is set appropriately: OFF = enabled (default “OFF”, see page 36). Via the auto-configuration the VME system controller is enabled, when the SPARC/CPU-20VT is plugged in slot 1, otherwise it is disabled.

If the auto-configuration is disabled, the VME system controller has to be controlled manually by setting B_SW7-2 appropriately: OFF = installed in VMEbus slot 1 or ON = not installed in VMEbus slot 1 (default “OFF”, see page 36).

Booting

The SPARC/CPU-20VT boot PROM consists of two flash memory devices (256K*8). They are installed in PLCC sockets on the SBus I/O board. They are write-protected if B_SW8-1 is set appropriately: OFF = write-protected (default “OFF”, see page 37).

Per default the SPARC/CPU-20VT is shipped with its boot PROM containing the OpenBoot firmware (see section 3.8 “OpenBoot Firmware” on page 65).

User Applications

The SPARC/CPU-20VT provides two flash memory devices (1M*8) to store user applications. They are write-protected if B_SW8-2 is set appropriately: OFF = write-protected (default “OFF”, see page 37).

Solaris

Booting the system is described in section 3.8.1 “Boot the System” on page 65 (support of `ttyc` and `ttyd` on page 68).

Note: **Note the information given in section 3.9 “Booting Solaris 2.5” on page 76 and section 3.10 “Patching Solaris 2.5 for 200 MHz MBus Modules” on page 80.**

204223 8 – 0 May 2000

3.8 OpenBoot Firmware

This chapter describes the use of the OpenBoot firmware. The following tasks will be described in detail:

- Boot the system
- Run diagnostics
- Display system information
- Reset the system
- OpenBoot help

For further information on the OpenBoot firmware see the *OPEN BOOT PROM 2.0 MANUAL SET*.

3.8.1 Boot the System

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

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

To boot the system from the default boot device, enter the following command at the Forth monitor prompt `ok`:

```
ok boot
```

If you are at the restricted monitor prompt `>`, enter:

```
> b
```

The boot command has the following format:
`boot [`*`device-specifier`*`] [`*`filename`*`] [-ah]`

Optional Boot Parameters

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

`[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 section.

`[-a]` `-a` prompt interactively for the device and name of the boot file

`[-h]` `-h` halt after loading the program

Devices to Boot from

To explicitly boot from the internal disk using the Forth monitor enter:

```
ok boot disk
```

At the restricted monitor prompt enter:

```
> 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 18 **Device Alias Definitions**

Alias	Description
	Defined for SCSI #1:
`scsi`	SCSI #1
`disk`	Default disk (1st internal) SCSI-target-ID 3
`disk3`	First internal disk SCSI-target-ID 3
`disk2`	Additional internal disk SCSI-ID 2
`disk1`	External disk SCSI-target-ID 1
`disk0`	External disk SCSI-target-ID 0

204223 8 – 0 May 2000

Table 18 Device Alias Definitions (cont.)

Alias	Description
tape (or tape0)	First tape drive SCSI-target-ID 4
tape1	Second tape drive SCSI-target-ID 5
cdrom	CD-ROM partition d, SCSI-target-ID 6
	Defined for Ethernet #1:
net	Ethernet #1
net-tpe	Twisted-pair Ethernet #1
net-aui	AUI Ethernet #1
	Defined for SCSI #2:
disk-2	Default disk SCSI-target-ID 3
disk23	First internal disk SCSI-target-ID 3
disk22	Disk SCSI-target-ID 2
disk21	Disk SCSI-target-ID 1
disk20	Disk SCSI-target-ID 0
tape2 (or tape20)	First tape drive SCSI-target-ID 4
tape21	Second tape drive SCSI-target-ID 5
cdrom2	CD-ROM partition d, SCSI-target-ID 6
	Defined for Ethernet #2:
net2	Ethernet #2
net2-aui	AUI Ethernet #2
floppy	Floppy disk
vme	VME
busnet busnet-tftp busnet-raw	Busnet
flash	Flash memory
ttyc	Serial port C
ttyd	Serial port D

Solaris Support for serial I/O Ports `ttyc` and `ttyd`

Note: **In order to use all four instead of only two serial I/O ports after booting Solaris, the `zs:maxzs` variable has to be changed and the system has to be rebooted using the reconfigure option (`-r`, see section 3.9 "Booting Solaris 2.5" on page 76). To change the `zs:maxzs` variable, insert the following line in /`etc/system`:**

```
set zs:maxzs=6
```

Redirecting in- and output is described in the *OPEN BOOT PROM 2.0 MANUAL SET*. To redirect in- and output to the additional serial I/O ports C and D the device aliases `ttyc` and `ttyd` have to be used.

3.8.2 NVRAM Boot Parameters

The OpenBoot firmware holds its configuration parameters in NVRAM. At the Forth monitor prompt enter `printenv` to see a list of all available configuration parameters.

Note: **Ensure that the `auto-boot?` parameter is always set to `true`.**

To Set Parameters

The OpenBoot command `setenv` may be used to set the parameters:
`setenv [`*`configuration_parameter`*`] [`*`value`*`]`

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

Table 19 Setting Configuration Parameters

Parameter	Default value	Description
`auto-boot?`	`true`	If `true`, automatic booting 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

204223 8 – 0 May 2000

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 from 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.*

3.8.3 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 serial I/O port A. If the system does not work correctly, error messages are displayed which indicate the problem. 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 self-test routine.

The table below lists several diagnostic routines followed by examples for each of these routines:

Table 20 Diagnostic Routines

Command	Description
`probe-scsi`	Identifies devices connected to the SCSI #1 bus
`probe-scsi-all [`*`device-path`*`]`	Performs 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.

Table 20 Diagnostic Routines

`test` *`device-specifier`*	Executes the specified device's self-test method. *`device-specifier`* may be a device path name or a device alias. **Example:** • `Test net` – test network connection • `Test /memory` – test number of MBytes specified in the `self-test-#megs` NVRAM parameter or test all of memory if `diag-switch?` is `true`
`test-all [`*`device-specifier`*`]`	Tests all devices that have a built-in self-test method and that reside below the specified device tree node. If *`device-path`* is omitted, the root node is used.
`watch-clock`	Monitors the clock function.
`watch-net-all`	Monitors network connection via all Ethernet interfaces installed in the system.
`watch-net`	Monitors network connection via Ethernet #1.
`watch-net2`	Monitors network connection via Ethernet #2.

Examples:

SCSI #1 bus To check the SCSI #1 for connected devices enter:

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

All SCSI Buses To test all the SCSI buses installed in the system enter the following (The actual response depends on the devices on the SCSI buses):

```
ok probe-scsi-all
/iommu@0,10000000/sbus@0,10001000/esp@2,100000

Target 6
Unit 0 Disk Removable Read Only Device SONY CD-ROM CDU-8012 3.1a

/iommu@0,10000000/sbus@0,10001000/espdma@4,8400000/esp@4,8800000

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

204223 8 – 0 May 2000

Single Device

To test a single installed device enter:

```
ok test device-specifier
```

This executes the `self-test` device method of the specified device node.

`device-specifier` may be a device path name or a device alias as described in Table 18, “Device Alias Definitions,” on page 66. The response depends on the self-test of the device node.

Group of Devices

To test a group of installed devices enter:

```
ok test-all
```

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

Memory

When you use the memory testing routine, the system tests the number of MBytes of memory specified in the NVRAM configuration parameter `self-test-#megs`. If the NVRAM configuration parameter `diag-switch?` is `true`, the whole 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 above-mentioned example, the first number (`0`) is the base address of the memory bank to be tested, the second number (`27`) is the number of the remaining MBytes. If the CPU board works correctly, the memory is erased and tested and you will receive the **ok** prompt. If the PROM or the on-board memory does not work, you will receive one of several potential error messages indicating the problem.

Clock

To test the clock function enter:

```
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.

Network

To monitor the network connection enter:

```
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 a dot (`.`) each time it receives a valid packet and displaying an `X` each time it receives a packet with an error which can be detected by the network hardware interface.

3.8.4 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 specific information to the individual machine, including the serial number, date of manufacture, and assigned Ethernet address. The following table lists these commands:

Table 21 Commands to Display System Information

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

204223 8 – 0 May 2000

3.8.5 Reset the System

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

```
ok reset
```

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

3.8.6 Using the FGA-5100 Enhanced Feature Set

To provide software compatibility the SPARC/CPU-20VTe is configured to run in the FGA-5000 compatibility mode unless explicitly configured to run in FGA-5100 enhanced mode, for example, by issuing the following OpenBoot commands:

```
ok setenv fga-5000-comp? false
ok setenv vme-dma-prio? true
ok reset
```

The first command sets the FGA-5100 into enhanced mode during subsequent initialisation. The second command controls the DMA priority on the VMEbus and enables greater maximum transfer lengths after subsequent initialisation.

3.8.7 OpenBoot Help

The Forth Monitor contains an online help which can be activated by entering:

```
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]`**.

- The online help shows you the Forth word, the parameter stack before and after execution of the Forth word (before -- after), and a short description.
- The online help of the Forth monitor is located in the boot PROM, that means that there is not an online help for all Forth words.

Example:

How to get help for special Forth words or subcategories:

```
ok help tools
 Category: Tools (memory,numbers,new commands,loops)
 Subcategories are:
Memory access
Arithmetic
Radix (number base conversions)
Numeric output
Defining new commands
Repeated loops
ok
```

204223 8 – 0 May 2000

```
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
```

3.9 Booting Solaris 2.5

Booting the system is described in section 3.8.1 “Boot the System” on page 65 (support of `ttyc` and `ttyd` on page 68). However, there are two situations in which you have to remove the two additional serial devices `ttyc` and `ttyd` in Solaris 2.5 to successfully boot the system.

Caution



This information is relevant for the:

- **Use of Solaris 2.5 together with OpenBoot 2.25.1 on a two-slot configuration of a SPARC/CPU-20VT (for example on a SPARC/CPU-20V)**
- **Booting Solaris 2.5 with two or more CPUs (e.g. 1 dual hyperSPARC module or two single hyperSPARC modules) and an SBus graphic card (framebuffer)**
  - **From CD-ROM**
  - **From disk using the reconfigure option (`-r`) of the `boot` or `reboot` command**
- **Only in these two situations with this special hardware configuration you need to remove the serial devices `ttyc` and `ttyd` to successfully boot Solaris 2.5.**
- **You have to remove `ttyc` and `ttyd` whenever you are in one of the two situations. If necessary for your application and if using a three- or four-slot configuration of a SPARC/CPU-20VT, you always have to add them again as described on the next page.**
- **The following description applies from OpenBoot Version 2.25.2 or greater.**

If you do not remove the serial devices `ttyc` and `ttyd` in these two situations, the result would be a system PANIC.

Removal of the Two Additional Serial Devices

Before booting from CD-ROM with Solaris 2.5 or booting with reconfigure, remove the serial devices `ttyc` and `ttyd`. This is necessary because `ttyc` and `ttyd` are two additional serial devices for a standard Solaris host system which usually are not pre-configured.

- Remove them by entering at the OpenBoot prompt:

```
ok setenv probe-ttys? false
```

This is the recommended way to boot the system for the first time after installing a new software package which demands a follow up boot with reconfigure (e.g. after installing Solaris 2.5 from CD-ROM).

204223 8 – 0 May 2000

After removing the serial devices `ttyc` and `ttyd`, Solaris 2.5 does not see any additional serial devices. This means that it is possible to load and execute the Solaris 2.5 operating system without any complain in all situations. If you want to use available additional serial devices during normal Solaris operation, you must include them again as described in the remaining part of this section.

Including the Serial Devices Again

To include the additional serial devices (not available on a SPARC/CPU-20V), you have to edit `/etc/system` and add the following line if not already done (see section 3.8.1 “Boot the System” on page 65):

```
set zs:maxzs = 6
```

This enables the Solaris serial interface driver named `zs` to handle up to six serial devices (keyboard, mouse, term/a, term/b, term/c, term/d – corresponding to the OpenBoot `zs`… devices).

If you want the additional serial devices to be included even after following boot processes (neither with reconfigure option nor boot from CD-ROM) you have to:

1. Log in as root, halt the system, and set `probe-ttys?` to `true`:

```
# sync; sync; halt
…
ok setenv probe-ttys? true
ok reset
…
ok boot
```

2. After logging in as root again, halt the system and enter the following commands:

```
# sync; sync; halt
…
```

3. The `zs` devices are attached to the device node `obio`. Therefore, you need to select this device node using the `cd` command as follows:

```
ok cd /obio
```

4. Enter `ls` and look for output lines including `zs@0`.

   The `ls` command lists all devices attached to the current device node along with the addresses of the device descriptor.

```
ok ls
...
ffd33eb0 zs@0,180008
...
ffd2eed0 zs@0,0
...
ffd2edf0 zs@0,100000
...
```

5. Among those three lines select the one in which `zs`... is followed by the highest number. The other two lines are related to standard serial devices (keyboard, mouse, `ttya`, and `ttyb` depending on the `zs` device and the number).

   In the example above, the line "`ffd33eb0 zs@0,180008`" is the one you are looking for. From this line note the number following the `zs`... string.

Note: **The address at the start of the line may differ between different OpenBoot versions!**

---

6. Boot the system:

```
ok boot
```

7. After logging in as root, enter the following commands. Be sure to replace `180008` used in the sample screen dump below by the number following the `zs`... string as shown in the sample screen dump of step 4 on page 78.

```
# cd /devices/obio
# mknod zs@0,180008:c c 29 4
# mknod zs@0,180008:d c 29 5
# chmod go+w zs@0,180008
# chgrp sys zs@0,180008
# ports
```

   The first `mknod` command (ending with `c 29 4`) creates the correct device node for `ttyc` (the second for `ttyd`). The minor device node number is four since `ttyc` is the fifth serial device following keyboard, mouse, `ttya`, and `ttyb` (Solaris starts counting serial devices at `0`).

   The `ports` command creates the links in the `/dev` directory hierarchy to the corresponding device nodes in `/devices/obio`.

204223 8 – 0 May 2000

After these commands you have set up correctly all links and you are able to use `/dev/term/c` (`ttyc`) and `/dev/term/d` (`ttyd`). However, you cannot boot the system from CD-ROM nor reboot the system with `reconfigure` unless removing `ttyc` and `ttyd` again.

3.10 Patching Solaris 2.5 for 200 MHz MBus Modules

The information in this section pertains to customers running or planning to run Solaris 2.5 with 200 MHz hyperSPARC processors. The 200 MHz hyperSPARC processors belong to the Colorado 4 (C4) family.

Note: **Only for these customers it is important to follow the instructions in this section carefully.**

Overview

This section covers the following topics:

- Obtaining the required patch for Solaris 2.5
- Patching your operating system prior to installing a 200 MHz hyperSPARC MBus module
- Installing Solaris 2.5 from CD-ROM

3.10.1 Obtaining Required Sun Kernel Jumbo Patch

As part of the procedure you will be required to install the following Sun Kernel Jumbo Patch for Solaris 2.5: Sun Kernel Jumbo Patch `103093-08` or later

Note: **If later revisions are available install those instead.**

Access to the Patch

Access to these patches is available from the following places:

- Sunsolve On-line: Access to Sunsolve is available through the world wide web. To go to this destination use the URL `http://access1.sun.com`
- Sun's ftp site: Sun's ftp site is `sunsite.unc.edu`. To download patches from this site, use `anonymous` as the login name and your e-mail address as your password.
- ROSS' ftp site: Ross' ftp site is `ftp.ross.com`. To download patches from this site, use `anonymous` as the login name and your e-mail address as your password. The Sun patches are located in `/pub/Sun-patches`.

204223 8 – 0 May 2000

3.10.2 Patching the Operating System prior to Installing 200 MHz hyperSPARC Modules

If you are currently running Solaris 2.5 with hyperSPARC processors, you can install the above named required Sun Patch prior to upgrading your system with C4 modules. It is not necessary for you to reinstall Solaris 2.5 to prepare for your upgrade.

To install the Sun Kernel Jumbo Patch follow the instructions in the `README` file. The Sun Kernel Jumbo Patch may also be installed using the `pkgadd` command. Once the installation is complete, follow the instructions in the *MBus Module Installation Guide* to install your hyperSPARC module(s).

3.10.3 Installing Solaris 2.5 from a CD-ROM

If you are currently using C4 modules, you cannot successfully install Solaris 2.5 from CD-ROM. The current CD-ROM versions do not include required patches which fix several bugs discovered in Solaris 2.5. If you do need to boot from CD-ROM and you are using 200 MHz MBus modules, you can use the procedure outlined below.

The procedures covered in this section are the following:

- "Booting Solaris 2.5 CD-ROM" on page 81
- "Patching the Operating System to Support C4 Modules" on page 82
- "Completing Installation of Solaris 2.5" on page 83

Booting Solaris 2.5 CD-ROM

Follow these instructions to initially boot Solaris 2.5:

8. Install the Solaris 2.5 CD-ROM in the drive.
9. At the `ok` prompt type:

   ok **boot cdrom kadb -d**

   This loads the kernel diagnostic program `kadb` with display prompt:

   `kadb:`
10. Enter a carriage return <*cr*> at the first `kadb:` prompt.

    kadb: **<*cr*>**
11. To execute the next command type:

    kadb[0]: **:s**

    The line printed should read `text address not found`.

At this point the kernel is loaded, and we are ready to begin patching.

Patching the Operating System to Support C4 Modules

The following five patches provide support for C4 modules. These patches change a `nop` instruction to an `icache` flush instruction. C4 modules require this extra flush instruction to work properly. This patch sequence can be used for systems running Solaris 2.5.

Note: **Follow the procedure and compare your output with the output in this section. Take extra care using capital `W` where indicated. If `kadb` does not show the original `nop` instruction before an address has been patched, do not continue the procedure.**

Patch #1:

1. Type the following command to verify `nop` command exists.

```
kadb[0]: mmu_flushpagectx+0x1c/i <cr>
mmu_flushpagectx+0x1c: nop
```

2. Replace `nop` instruction.

```
kadb[0]:/W c0a00620 <cr>
mmu_flushpagectx+0x1c: 1000000=c0a00620
```

Patch #2:

1. Type the following command to verify `nop` command exists.

```
kadb[0]: mmu_flushseg+0x1c/i <cr>
mmu_flushseg+0x1c: nop
```

2. Replace `nop` instruction.

```
kadb[0]:/W c0a00620 <cr>
mmu_flushseg+0x1c: 1000000=c0a00620
```

Patch #3:

1. Type the following command to verify `nop` command exists.

```
kadb[0]: mmu_flushrgn+0x1c/i <cr>
mmu_flushrgn+0x1c: nop
```

2. Replace `nop` instruction.

```
kadb[0]:/W c0a00620 <cr>
mmu_flushrgn+0x1c: 1000000=c0a00620
```

Patch #4:

1. Type the following command to verify `nop` command exists.

```
kadb[0]: mmu_flushctx+0x14/i <cr>
mmu_flushctx+0x14: nop
```

2. Replace `nop` instruction.

```
kadb[0]: /W c0a00620 <cr>
mmu_flushctx+0x14: 1000000=c0a00620
```

Patch 5#:

1. Type the following command to verify `nop` command exists.

```
kadb[0]: ross625_xcall_medpri+0x90/i <cr>
ross625_xcall_medpri+0x90: nop
```

2. Replace `nop` instruction.

204223 8 – 0 May 2000

```
kadb[0]: /W c0a00620 <cr>
ross625_xcall_medpri+0x90: 1000000=c0a00620
```

Completing Installation of Solaris 2.5

After completing patching the kernel, it is time to resume executing the kernel. Follow these steps to complete the CD-ROM installation:

3. To continue type:

   kadb[0]:**:c**

   The system installation continues.

4. Select the `no reboot` choice during the installation process. This option disables automatic rebooting.

5. Halt the system when the installation is complete. To halt the system, use the right most mouse button to select `System Control` and `Halt System`.

6. Boot the disk using `kadb`.

   ok **boot disk kadb -d**

7. Enter a carriage return at the `kadb` prompt to load the kernel:

   kadb[0]: <***cr***>

8. Type `:s` at the `kadb` prompt to start executing the kernel:

   kadb[0]: **:s**

9. Re-patch the kernel as described in “Patching the Operating System to Support C4 Modules” on page 82.

10. Type **`:c`** at the `kadb` prompt to continue running the patched kernel:

    kadb[0]: **:c**

    The system continues to boot, and a login prompt is displayed.

11. Installing the kernel jumbo patches: After applying the `kadb` fixes, the Solaris 2.5 Sun Kernel patch needs to be installed as described in section 3.10.2 “Patching the Operating System prior to Installing 200 MHz hyperSPARC Modules” on page 81.

204223 8 – 0 May 2000

4 Hardware Description

The SPARC/CPU-20VT is a high performance computer providing a full VME64 compliant VMEbus interface including DMA. It is based on:

- The scalable hyperSPARC multiprocessing architecture (see section 4.2 “MBus Modules” on page 92)
- The VMEbus, and the FGA-5x00 (see section 4.5 “VMEbus Interface – FGA-5x00” on page 97)

Features

Besides the VMEbus interface the SPARC/CPU-20VT provides the following components on the three on-board buses – the MBus, the SBus and the EBus:

MBus Participants

- Two MBus slots for the hyperSPARC MBus modules (see section 4.2 “MBus Modules” on page 92)
- The RMC memory controller on the MBus to access the ECC DRAM on memory modules (see section 4.4 “Main Memory and Memory Controller RMC” on page 95)

MBus-to-SBus

- The M2S interfacing the MBus to the SBus (see section 4.3 “MBus-to-SBus Interface – M2S” on page 93, this section also gives a short overview of the MBus Participants and of the SBus Participants)

SBus Participants

- Two SBus expansion slots for industry-standard SBus modules. SBus modules enable the expansion of I/O interfaces and memory and processing performance with a wide range of solutions. The two SBus slots fully support SBus DMA transfers of SBus modules having DMA capabilities.
- Two MACIO devices – one on the base board and one on the SBus I/O board – implementing two distinct Ethernet interfaces, two distinct SCSI interfaces, and one parallel I/O port (see section 4.6 “Ethernet, SCSI, Parallel I/O – 2 MACIO Devices” on page 104)

SBus-to-EBus



EBus Participants

- The SEC interfacing the SBus to the EBus (see section 4.7 “SBus-to-EBus Controller – SEC” on page 106), this section also gives a short overview of the SBus Participants)
- Boot PROM (see section 4.8.2 “Boot PROM” on page 111)
- User flash (see section 4.8.3 “User Flash” on page 112)
- Real-time clock and NVRAM (see section 4.8.5 “RTC / NVRAM – MK48T18” on page 116)
- Floppy interface (see section 4.8.6 “Floppy Disk Interface – FDC” on page 117)
- Four serial I/O ports (see section 4.8.7 “Serial I/O Ports A, B, C, and D – SCC” on page 118)
- Sun-compatible keyboard / mouse port (see section 4.8.9 “Keyboard and Mouse Port – SCC” on page 120)
- Two temperature sensors (see section 4.9.4 “Temperature Sensors” on page 128)
- As factory option one audio port (see section 4.8.8 “Audio Port – Am79C30A (Factory Option)” on page 119)

Interfaces on Front Panel

The front panel of the SPARC/CPU-20VT provides both Ethernet ports, the SCSI #1 port, the parallel I/O port (for all four see section 4.6 “Ethernet, SCSI, Parallel I/O – 2 MACIO Devices” on page 104), all four serial I/O ports (see section 4.8.7 “Serial I/O Ports A, B, C, and D – SCC” on page 118), an audio port (see section 4.8.8 “Audio Port – Am79C30A (Factory Option)” on page 119) and the keyboard / mouse port (see section 4.8.9 “Keyboard and Mouse Port – SCC” on page 120). These ports serve as console port (serial I/O port A), for download and for data communication.

204223 8 – 0 May 2000

Interfaces on VMEbus P2 Connector

The following interfaces are available on the three-row VMEbus P2 connector of the base board (see section 3.4.4 “VME P2 Connector Pinout of the Base Board” on page 47) and of the SBus I/O board (see section 3.4.5 “VME P2 Connector Pinout of the SBus I/O Board” on page 48):

- Ethernet #1 and #2, SCSI #1 and #2 interfaces and the parallel I/O port (see section 4.6 “Ethernet, SCSI, Parallel I/O – 2 MACIO Devices” on page 104)
- Floppy interface (see section 4.8.6 “Floppy Disk Interface – FDC” on page 117)
- Serial I/O ports (see section 4.8.7 “Serial I/O Ports A, B, C, and D – SCC” on page 118)
- Loud speaker audio port (see section 3.4.5 “VME P2 Connector Pinout of the SBus I/O Board” on page 48)
- Keyboard / mouse port (see section 4.8.9 “Keyboard and Mouse Port – SCC” on page 120)

Factory Options

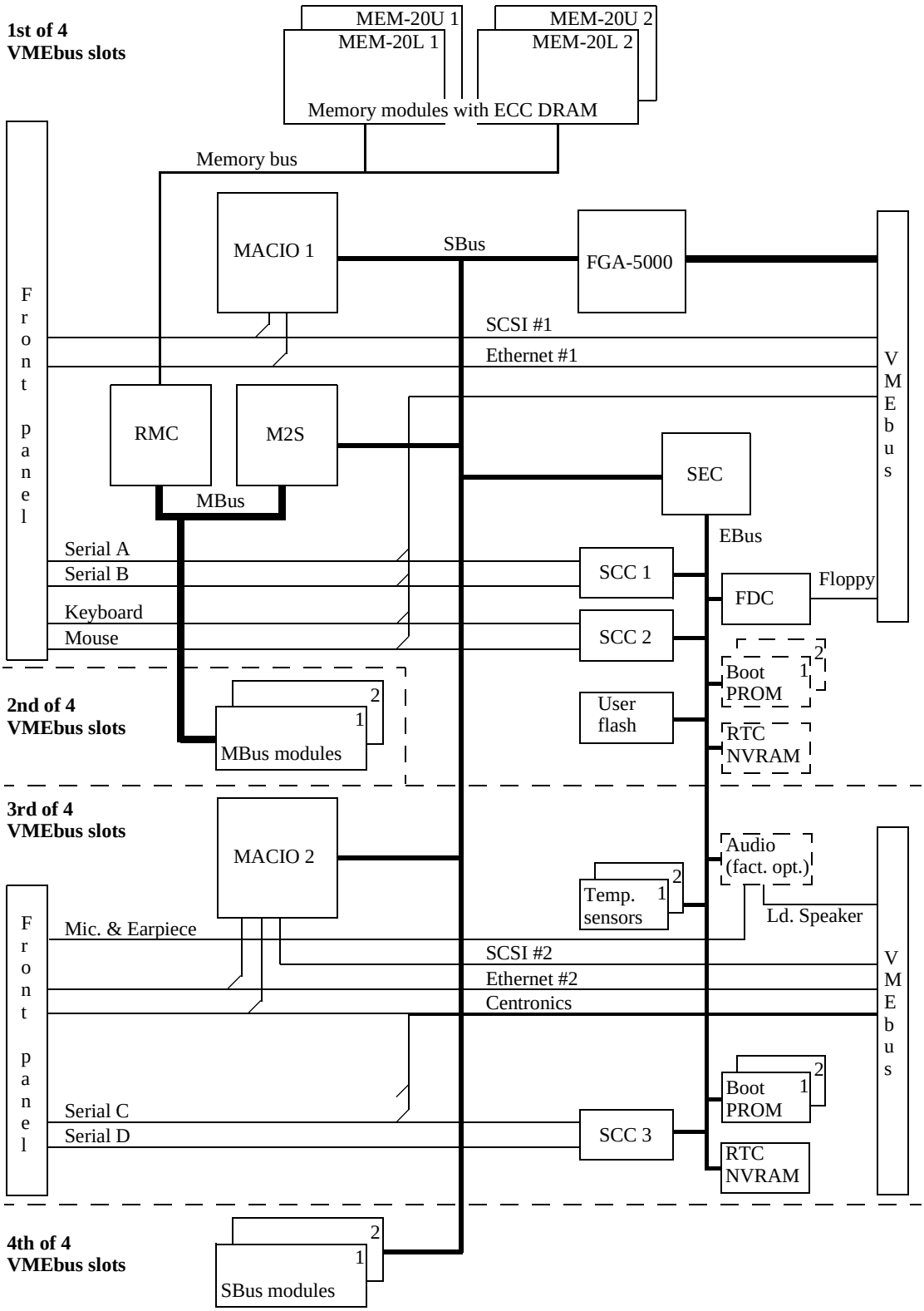
The following factory options are available:

- 0, 1, 2 (=default), or 4 MByte user flash (see section 4.8.3 “User Flash” on page 112)
- Audio port (see section 4.8.8 “Audio Port – Am79C30A (Factory Option)” on page 119)

Besides these factory options, the SPARC/CPU-20VT is configurable to the user’s needs by:

- Number and type of MBus modules (see table 3 “Excerpt from the Data Sheet’s Ordering Information” on page 8)
- Number and capacity of memory modules (see section 4.4 “Main Memory and Memory Controller RMC” on page 95)

Figure 20 Block Diagram of the SPARC/CPU-20VT



204223 8 – 0 May 2000

4.1 Address Map of the SPARC/CPU-20VT

The two tables below list the physical address ranges of the SPARC/CPU-20VT. For further information on the accessible VMEbus memory see table 27 “Relating SBus Slots to SBUS_SSELx_x+1 Contents” on page 100. For the detailed address map of the local I/O devices see table 31 “Physical Address Map of Local I/O Devices” on page 110.

Note: **The FGA-5100^{E} provides extended access to VMEbus memory so that the physical address map differs for an FGA-5100^{E} based CPU boards as compared with an FGA-5000 and FGA-5100^{C} based CPU board. Please note:**

- Only the address range for SBus slot utilization of FGA-5100^{E} based SPARC/CPU-20VT boards is affected. The FGA-5100^{E} register map base address (E.FFFF.FE0016) is different to both the FGA-5000 and the FGA-5100^{C} register map base address (E.9FFF.FE0016).

  See table 22 “FGA-5100E: SPARC/CPU-20VT Physical Address Map” on page 89.

- FGA-5000 and FGA-5100^{C} both have the same physical address map.

  See table 23 “FGA-5000, FGA-5100C: SPARC/CPU-20VT Physical Address Map” on page 91.

Table 22 FGA-5100^{E}: SPARC/CPU-20VT Physical Address Map

Physical Address Range	Size	Device
$0.0000.0000_{16}$ …$0.1FFF.FFFF_{16}$	512 MB	Main memory on mem. modules
$0.2000.0000_{16}$ …$D.FFFF.FFFF_{16}$	55.5 GB	Reserved
$E.0000.0000_{16}$ …$E.0FFF.FFFF_{16}$	256 MB	VMEbus memory
$E.1000.0000_{16}$ …$E.1FFF.FFFF_{16}$	256 MB	SBus module 1 or VMEbus memory[1)]
$E.2000.0000_{16}$ …$E.2FFF.FFFF_{16}$	256 MB	SBus module 2 or VMEbus memory[1)]
$E.3000.0000_{16}$ …$E.DFFF.FDFF_{16}$	2.75 GB	VMEbus memory

Table 22 FGA-5100^E: SPARC/CPU-20VT Physical Address Map (cont.)

Physical Address Range	Size	Device
$E.E000.0000_{16}$ …$E.EFFF.FFFF_{16}$	256 MB	MACIO 2 on SBus I/O board: Ethernet #2, SCSI #2, parallel I/O port
$E.F000.0000_{16}$ …$E.F7FF.FFFF_{16}$	128 MB	MACIO 1 on base board: Ethernet #1, SCSI #1
$E.F800.0000_{16}$ …$E.FFFF.FDFF_{16}$:	128 MB – 512 Byte	VMEbus memory
$E.FFFF.FE00_{16}$ …$E.FFFF.FFFF_{16}$	512 Byte	FGA-5100^E registers
$F.0000.0000_{16}$ …$F.0FFF.FFFF_{16}$	256 MB	RMC: memory controller
$F.1000.0000_{16}$ …$F.DFFF.FFFF_{16}$	3.25 GB	Reserved
$F.E000.0000_{16}$ …$F.EFFF.FFFF_{16}$	256 MB	M2S: MBus-to-SBus interface
$F.F000.0000_{16}$ …$F.F1FF.FFFF_{16}$	32 MB	SEC: SBus-to-EBus interface; boot PROM, user flash, RTC/NVRAM, and local I/O (serial ports, keyboard / mouse port, floppy interface, LCAs, and as fact. option audio port, for detailed information, see table 31 “Physical Address Map of Local I/O Devices” on page 110)
$F.F200.0000_{16}$ …$F.FFFF.FFFF_{16}$	224 MB	Reserved

1. OpenBoot ensures that VMEbus memory is only decoded if no SBus module is installed. Other software must ensure that the FGA-5100^E does not use any address decoded by installed SBus modules.

204223 8 – 0 May 2000

Table 23 FGA-5000, FGA-5100^{C}: SPARC/CPU-20VT Physical Address Map

Physical address range	Size	Device
$0.0000.0000_{16}$ …$0.1FFF.FFFF_{16}$	512 MB	Main memory on mem. modules
$0.2000.0000_{16}$ …$D.FFFF.FFFF_{16}$	55.5 GB	Reserved
$E.0000.0000_{16}$ …$E.0FFF.FFFF_{16}$	256 MB	SBus (unused)
$E.1000.0000_{16}$ …$E.1FFF.FFFF_{16}$	256 MB	SBus module 1
$E.2000.0000_{16}$ …$E.2FFF.FFFF_{16}$	256 MB	SBus module 2
$E.3000.0000_{16}$ …$E.3FFF.FFFF_{16}$	256 MB	SBus (unused)
$E.4000.0000_{16}$ …$E.8FFF.FFFF_{16}$	1.25 GB	VMEbus memory
$E.9000.0000_{16}$ …$E.9FFF.FDFF_{16}$	256 MB – 512 Byte	VMEbus memory
$E.9FFF.FE00_{16}$ …$E.9FFF.FFFF_{16}$	512 Byte	FGA-5000 or FGA-5100^{C} registers
$E.A000.0000_{16}$ …$E.DFFF.FFFF_{16}$	1 GB	SBus (unused)
$E.E000.0000_{16}$ …$E.EFFF.FFFF_{16}$	256 MB	MACIO 2 on SBus I/O board: Ethernet #2, SCSI #2, parallel I/O port
$E.F000.0000_{16}$ …$E.FFFF.FFFF_{16}$	256 MB	MACIO 1 on base board: Ethernet #1, SCSI #1
$F.0000.0000_{16}$ …$F.0FFF.FFFF_{16}$	256 MB	RMC: memory controller
$F.1000.0000_{16}$ …$F.DFFF.FFFF_{16}$	3.25 GB	Reserved
$F.E000.0000_{16}$ …$F.EFFF.FFFF_{16}$	256 MB	M2S: MBus-to-SBus interface
$F.F000.0000_{16}$ …$F.F1FF.FFFF_{16}$	32 MB	SEC: SBus-to-EBus interface; boot PROM, user flash, RTC/NVRAM, and local I/O (serial ports, keyboard / mouse port, floppy interface, LCAs, and as fact. option audio port, for detailed information, see table 31 “Physical Address Map of Local I/O Devices” on page 110)
$F.F200.0000_{16}$ …$F.FFFF.FFFF_{16}$	224 MB	Reserved

4.2 MBus Modules

The SPARC/CPU-20VT MBus modules are processor modules based on the hyperSPARC architecture. The hyperSPARC architecture provides scalable CPU performance by choosing the appropriate frequency, the appropriate number of processors per MBus module (single or dual), and the appropriate number of MBus modules (1 or 2).

Available MBus Modules

For information on the MBus modules being available from Force Computers, see table 3 “Excerpt from the Data Sheet’s Ordering Information” on page 8.

Sample Data Sheets

For a sample data sheet of a single and a dual hyperSPARC MBus module, see “hyperSPARC single- and dual-CPU module – RT6224/6226K.

Main hyperSPARC MBus module features are listed below.

Hypersparc Features

- Superscalar SPARC CPU with dual integer ALU, integrated floating point unit and 8 KByte instruction cache
- Cache controller, memory management, and tag unit
- Cache data units
- Intra-module bus incorporates low voltage logic to reduce power and increase speed
- Dual-level caches
- Demand-paged virtual memory management
- Full multiprocessing implementation: hardware support for symmetric, shared memory multiprocessing and level two MBus support for cache coherency
- SPARC compliant: SPARC instruction set architecture (ISA) version eight compliant, conforms to SPARC reference MMU architecture, and conforms to SPARC level 2 MBus module specification (rev. 1.2)

204223 8 – 0 May 2000

4.3 MBus-to-SBus Interface – M2S

The M2S interfaces the MBus to the SBus, thereby enabling the communication between the following components on the MBus and the SBus:

MBus Participants

- The MBus is the processor bus of the SPARC/CPU-20VT and connects the MBus modules, the RMC memory controller and the M2S itself.

SBus Participants

- The 32-bit wide SBus has the following participants: the M2S itself, the FGA-5000 to interface to the VMEbus, the two standard SBus modules, the SEC to interface to the EBus and the I/O devices available on the EBus and the two MACIO to interface to Ethernet, to SCSI and for parallel I/O.

Physical Address Range

The physical MBus address range for register accesses to the M2S is `F.E000.000016`…`F.EFFF.FFFF16`.

Data Sheet

For a detailed M2S description, see “MBus-to-SBus Interface – M2S. Main M2S tasks are outlined below:

M2S Tasks

- MBus arbitration: The MBus arbiter inside the M2S supports up to six MBus masters including four CPUs and the M2S itself. The arbiter uses a round robin arbitration scheme.
- SBus arbitration: The M2S supports up to eight SBus masters including the M2S itself. On the SPARC/CPU-20VT there are up to six SBus devices having master capabilities: the two SBus modules (depending on the SBus module’s type), the MACIO 1 and 2, the FGA-5000 and the M2S itself. For the arbitration of the SBus a round-robin scheme is used.
- MBus-to-SBus protocol conversion: The M2S bridges between the two protocols on the two buses. It controls:
  - Access of MBus masters to SBus devices
  - Access of SBus masters to the main memory connected to the MBus via the RMC memory controller
- Data buffering: The M2S interfaces the 64-bit wide data path of the MBus and the 32-bit data path of the SBus. To match the different bandwidths of MBus and SBus the M2S contains four sets of data buffers.
- I/O reference MMU: SBus masters send out virtual addresses to access the main memory via the RMC on the MBus whereas the MBus is based on physical addresses. Therefore, the IOMMU of the M2S translates virtual addresses into physical addresses. The IOMMU has a translation look-aside buffer (TLB) with thirtytwo entries to speed up translation.

Configuration Registers

The following table shows the M2S configuration registers relevant for the SPARC/CPU-20VT SBus participants. It also shows the SBus slave base addresses of the SBus participants.

Table 24 Configuration Registers for SBus Participants

Physical reg. addr.	Slave base addr.	SBus participant
F.E000.1014$_{16}$	E.1000.000016	SBus module 1
F.E000.1018$_{16}$	E.2000.000016	SBus module 2
F.E000.1020$_{16}$	E.E000.000016	MACIO 2
F.E000.1024$_{16}$	E.F000.000016	MACIO 1
F.E000.1028$_{16}$	E.9FFF.FE0016	FGA-5000

204223 8 – 0 May 2000

4.4 Main Memory and Memory Controller RMC

The main memory of the SPARC/CPU-20VT is provided by memory modules. It is controlled by the RMC memory controller (see "Memory Controller RMC" on page 97). The memory data path is 128 bit wide with additional 16 bit for the Error Correction Code (ECC).

Main Memory

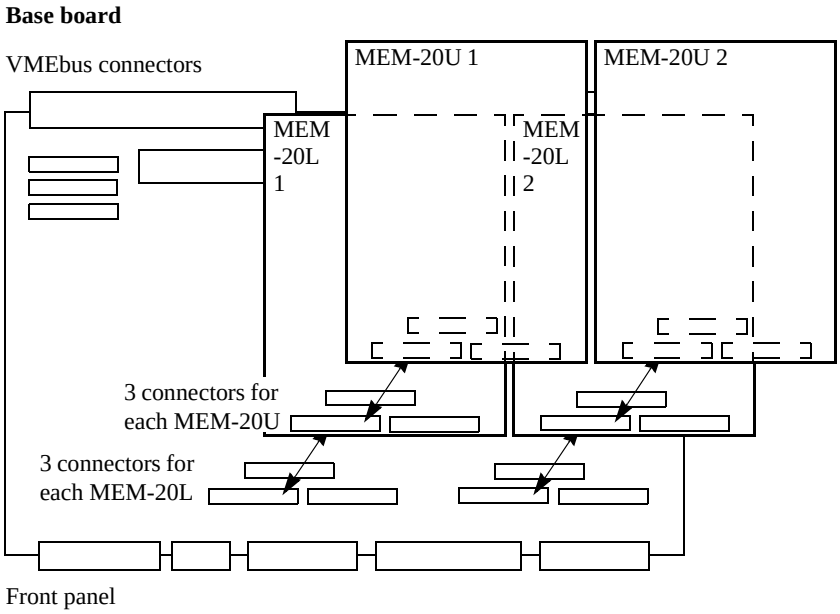
Upper and Lower Memory Modules

There are two types of memory modules:

- Lower memory modules MEM-20L for direct connection to the base board
- Upper memory modules MEM-20U for connection to the MEM-20L

The SPARC/CPU-20VT can hold up to two MEM-20L and two MEM-20U, totalling to up to four memory modules.

Figure 21 Locating MEM-20L 1, MEM-20L 2, MEM-20U 1 and MEM-20U 2



Note: **To avoid malfunction ensure that MEM-20L 1 is installed.**

Memory Capacities

Upper and lower memory modules are available with 64 MByte or 128 MByte capacity implemented in one or two DRAM banks. Thereby, a maximum of 512 MByte main memory is available.

The following table relates the installed memory modules to the physical address range for the main memory. Contiguous physical memory addresses are automatically provided for the two memory modules MEM-20L 1 and -20U 1 as well as for MEM-20L 2 and -20U 2.

Table 25 Physical Memory Adresses for MEM-20L 1 and MEM-20U 1

Physical Address Range	MEM-20L 1 [MByte]	MEM-20U 1 [MByte]	Sum [MByte]
$0.0000.0000_{16}$ …$0.03FF.FFFF_{16}$	64	Not assembled	64
$0.0000.0000_{16}$ …$0.07FF.FFFF_{16}$	64	64	128
$0.0000.0000_{16}$ …$0.07FF.FFFF_{16}$	128	Not assembled	128
$0.0000.0000_{16}$ …$0.0BFF.FFFF_{16}$	128	64	196
$0.0000.0000_{16}$ …$0.0BFF.FFFF_{16}$	64	128	196
$0.0000.0000_{16}$ …$0.0FFF.FFFF_{16}$	128	128	256

Table 26 Physical Memory Adresses for MEM-20L 2 and MEM-20U 2

Physical Address Range	MEM-20L 2 [MByte]	MEM-20U 2 [MByte]	Sum [MByte]
$0.1000.0000_{16}$ …$0.13FF.FFFF_{16}$	64	Not assembled	64
$0.1000.0000_{16}$ …$0.17FF.FFFF_{16}$	64	64	128
$0.1000.0000_{16}$ …$0.17FF.FFFF_{16}$	128	Not assembled	128
$0.1000.0000_{16}$ …$0.1BFF.FFFF_{16}$	128	64	196
$0.1000.0000_{16}$ …$0.1BFF.FFFF_{16}$	64	128	196
$0.1000.0000_{16}$ …$0.1FFF.FFFF_{16}$	128	128	256

204223 8 – 0 May 2000

Memory Controller RMC

The Memory Controller (RMC) on the SPARC/CPU-20VT directly interfaces the 128-bit wide main memory to the 64-bit wide MBus. The RMC always operates as an MBus slave device. It provides full DRAM control including refresh cycles. It uses eight RAS lines to select the eight DRAM banks. Two CAS lines are used to select the upper or the lower 64 data bit. Partial write accesses (less than 8 Byte) result in a read-modify-write cycle.

Physical Address Range

The physical MBus address range for register accesses to the RMC is `F.0000.000016…F.0FFF.FFFF16`.

Data Sheet

For detailed information, see “Memory Controller – RMC. Main features are listed below:

RMC Features

- Supports level two coherent invalidate
- Supports SPARC wrapping order
- Uses a single-bit correction and multi-bit detection ECC scheme
- Queues four MBus 32-Byte write requests (4 x 32 Byte data buffers with a four entry-address queue
- Memory access supports byte, half-word, word, and double-word and up to 128 Byte burst transfers
- 1- to 128-Byte read/write DRAM transactions (fast page mode)
- Staggered CAS_ before RAS_ refresh
- MBus read latency of (MAS to 1st data) A+7 clocks

4.5 VMEbus Interface – FGA-5x00

The SPARC/CPU-20VT utilizes the FGA-5x00 to provide fully SBus and VMEbus compliant interfaces. 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.

FGA-5x00 Features

- VMEbus master and slave interface
- DMA controller
- Interrupts
- VMEbus arbiter and system controller functions
- Force message broadcast FMB, Mailboxes, and semaphores
- Reset functions

- Timers

FGA-5100^{E} Extended Features

The following list gives an overview of the extensions provided by the FGA-5100^{E} as compared with the FGA-5000:

- FGA-5100 identification and two basic FGA-5100 operating modes
- 2eVME support
- Three buffers for each direction and an additional buffer for DMA
- Automatic three-level write posting
- VMEbus master interface
  - Additional VMEbus master range register sets
  - Enhanced SBus slave slot selection via additional FGA-5100 pins: `SSELEX[3:0]` and `SSELEXEN`
  - Programmable address modifier codes
  - VMEbus CR/CSR accesses
- VMEbus slave interface
  - Additional SBus slave range register sets
  - Extended VMEbus slave interface capabilities
- SBus master interface
  - SBus master read pipelining (VMEbus slave read pipelining)
  - SBus 64-bit extended transfers
- Added VMEbus interrupter
- Enhanced and extended DMA capabilities accompanied by a new style of DMA controller operation
  - Controllable minimum and maximum block size the DMA controller generates depending on its mode of operation
  - Additional DMA controller modes
  - Additional 2eVME DMA data capability
- Extended transfer error information via `VWPAR_RNG`, `SWPAR_RNG`, and `SLERR_RNG`
- Added data buffer status register for VMEbus master and slave transfers
- Added IBOX_COUNT register
- Additional strobe glitch filters including control bits in the `VME_HANDSHAKE` register for VBGIN[3:0], VIACK, and VBRI[3:0] signals
- Enhanced Force Message Broadcast (see note below)

204223 8 – 0 May 2000

Note: **Due to the enhanced Force message broadcast implementation of the FGA-5100 it is no longer necessary to use the FMB channel data discard registers and section 4.5.1 "Adapting the FGA-5000" on page 101 may be skipped in case of FGA-5100 based SPARC/CPU-20VT boards (FGA-5100^{E} and FGA-5100^{C}).**

Note: **For compatibility reasons read accesses to both the FMB Channel-0 Data Discard Status Register at `F.F124.000C`$_{16}$ and FMB Channel-1 Data Discard Status Register at `F.F124.000D`$_{16}$ will always return `MSG_VALID` = `1` (= data valid, see page 102).**

For further information including a detailed comparison of the FGA-5100 with the FGA-5000, refer to the *FGA-5x00 Reference Guide.*

FGA-5x00 Register Base Address

After an SBus reset the FGA-5x00 is accessible within SBus slot four at offset `000.0000`$_{16}$ but is reprogrammed to respond to accesses within:

- SBus slot nine at offset `FFF.FE00`$_{16}$ in case of FGA-5000 and FGA-5100^{C}
- SBus slot 1fifteen at offset `FFF.FE00`$_{16}$ in case of FGA-5100^{E}

For further information see table 22 "FGA-5100E: SPARC/CPU-20VT Physical Address Map" on page 89 and table 23 "FGA-5000, FGA-5100C: SPARC/CPU-20VT Physical Address Map" on page 91.

Access VMEbus Memory

The accessible VMEbus memory has been substantially increased for FGA-5100^{E} based CPU boards in case of:

- FGA-5000 and FGA-5100^{C} based CPU boards a total of 1.5 GB – 512 Byte VMEbus memory is accessible
- FGA-5100^{E} based CPU boards a total of 3.6 GB VMEbus memory is accessible

The following table relates the physical address ranges for the VMEbus interface to the values to be stored in the `SSELx` or `SSELx+1` field when using a particular range.

Table 27 Relating SBus Slots to SBUS_SSEL*x*_*x+1* Contents

Sbus-Slot#	Address Range	SSEL*x* or SSEL*x*+1 Value in case of FGA-5000 and FGA-5100^{C}	SSEL*x* or SSEL*x*+1 Value in case of FGA-5100^{E}
0	E.0000.0000$_{16}$ …E.0FFF.FFFF$_{16}$	Not applicable	C$_{16}$
1	E.1000.0000$_{16}$ …E.1FFF.FFFF$_{16}$	Not applicable	D$_{16}$ [1)]
2	E.2000.0000$_{16}$ …E.2FFF.FFFF$_{16}$	Not applicable	E$_{16}$ [1)]
3	E.3000.0000$_{16}$ …E.3FFF.FFFF$_{16}$	Not applicable	F$_{16}$
4	E.4000.0000$_{16}$ …E.4FFF.FFFF$_{16}$	0$_{16}$	0$_{16}$
5	E.5000.0000$_{16}$ …E.5FFF.FFFF$_{16}$	1$_{16}$	1$_{16}$
6	E.6000.0000$_{16}$ …E.6FFF.FFFF$_{16}$	2$_{16}$	2$_{16}$
7	E.7000.0000$_{16}$ …E.7FFF.FFFF$_{16}$	3$_{16}$	3$_{16}$
8	E.8000.0000$_{16}$ …E.8FFF.FFFF$_{16}$	4$_{16}$	4$_{16}$
9	E.9000.0000$_{16}$ …E.9FFF.FDFF$_{16}$	5$_{16}$	5$_{16}$
	E.9FFF.FE00$_{16}$ …E.9FFF.FFFF$_{16}$	FGA register space	5$_{16}$
10	E.A000.0000$_{16}$ …E.AFFF.FFFF$_{16}$	Not applicable	6$_{16}$
11	E.B000.0000$_{16}$ …E.BFFF.FFFF$_{16}$	Not applicable	7$_{16}$
12	E.C000.0000$_{16}$ …E.CFFF.FFFF$_{16}$	Not applicable	8$_{16}$
13	E.D000.0000$_{16}$ …E.DFFF.FFFF$_{16}$	Not applicable	9$_{16}$
14	E.E000.0000$_{16}$ …E.EFFF.FFFF$_{16}$	Not applicable	not applicable

204223 8 – 0 May 2000

Table 27 Relating SBus Slots to SBUS_SSEL*x*_*x+1* Contents (cont.)

Sbus-Slot#	Address Range	SSEL*x* or SSEL*x*+1 Value in case of FGA-5000 and FGA-5100[C]	SSEL*x* or SSEL*x*+1 Value in case of FGA-5100[E]
15	$E.F000.0000_{16}$ …$E.F7FF.FFFF_{16}$	Not applicable	not applicable
	$E.F800.0000_{16}$ …$E.FFFF.FDFF_{16}$	Not applicable	B_{16}
	$E.FFFF.FE00_{16}$ …$E.FFFF.FFFF_{16}$	Not applicable	FGA register space

1. OpenBoot ensures that VMEbus memory is only decoded if no SBus module is installed in the respective SBus slot. Other software must ensure that the FGA-5100[E] does not use any address decoded by installed SBus modules.

4.5.1 Adapting the FGA-5000

The FGA-5000 requires a small amount of glue logic to be used on this SBus expansion device.

Additional Registers for BERR Handling

When the FGA-5000 is not the VMEbus master, the VMEbus input signal BERR is not directly routed to the FGA-5000. This masking is required for normal VMEbus transfers.

However, a VMEbus slave should see the input signal BERR when:

- Slave is one of several selected slaves during an FMB cycle
- One or more of the other slaves acknowledge a transfer with BERR

Because of the BERR masking this is not true for the FGA-5000 and therefore, the FGA-5000 does not recognize invalid messages during FMB transfers.

In order to enable software to discard invalid messages during FMB cycles, additional registers have been implemented in a separate programmable on-board device.

Registers for FGA-5000 Support

The following registers are provided additionally to the FGA-5000 registers:

- FMB channel-0 data discard status register
- FMB channel-1 data discard status register

These additional registers are documented in the remaining section. For information on the FMB implementation in the FGA-5000 and on the FGA-5000 registers refer to the *FGA-5000 Reference Guide*.

FMB Channel-0

FMB channel-0 consists of an eight-stage FIFO and so does the FMB channel-0 data discard status register. Read accesses to this register switch the internal read pointer one step ahead in the FIFO.

Note: **Force Computers recommends coordinating access to this register and the related FGA-5000 registers so that synchronization of both FIFOs is not violated.**

Table 28 FMB Channel-0 Data Discard Status Register

F.F124.000C$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	1	1	1	MSG_ VALID

MSG_VALID (R)

MSG_VALID indicates whether the data in the FMB channel 1 of the FGA-5000 has to be discarded.

= 0 FMB channel's data are invalid.

= 1 FMB channel's data are valid.

FMB channel-1

FMB channel 1 consists of a one-stage FIFO.

Table 29 FMB Channel-1 Data Discard Status Register

F.F124.000D$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	1	1	1	MSG_ VALID

MSG_VALID (R)

MSG_VALID indicates whether the data in the FMB channel 1 of the FGA-5000 has to be discarded.

= 0 FMB channel's data are invalid.

= 1 FMB channel's data are valid.

4.5.2 VMEbus SYSRESET Input and Output

SYSRESET Input

An external SYSRESET generates an on-board reset if B_SW7-3 is set appropriately (default "B_SW7-3", see page 36): OFF = enabled, i.e. VMEbus SYSRESET generates on-board reset.

204223.8 – 0 May 2000

SYSRESET Output

To enable SYSRESET signalling to the VMEbus it is necessary to set B_SW7-4 appropriately (default “B_SW7-4”, see page 36): OFF = enabled, i.e. on-board reset is driven to VMEbus.

In the following situations the SPARC/CPU-20VT generates a SYSRESET signal to the VMEbus:

- On-board local SBus reset
- Power-up reset. Power-up reset occurs after powering on the SPARC/CPU-20VT and reaching a stable board supply voltage

4.6 Ethernet, SCSI, Parallel I/O – 2 MACIO Devices

The SPARC/CPU-20VT provides two MACIO, one on the base board and another on the SBus I/O board. They implement:

SCSI

- Two distinct fast 8-bit SCSI interfaces, one on the base board and one on the SBus I/O board (see section 3.5 “SCSI Configuration” on page 56)

  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 ten Mbytes per second.

  The STP2000 has on-chip 48 mA drivers and therefore it provides the direct drive of the single-ended SCSI bus. The SCSI core is a superset of the industry standard NCR53C90A which has been modified to support fast SCSI.

  SCSI #1 is available both on the front panel and on the VMEbus P2 connector of the base board whereas SCSI #2 is only available on the VMEbus P2 connector of the SBus I/O board. Both are single-ended and support TERMPWR.

Ethernet

- Two distinct Ethernet interfaces, one on the base board and one on the SBus I/O board (see section 3.6 “Ethernet Configuration” on page 60).

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

  The Ethernet interfaces #1 and #2 both are available on the respective front-panel and on the respective VMEbus P2 connector.

Parallel I/O

- One parallel I/O port on the SBus I/O board

  The uni- or bi-directional parallel I/O port is Centronics compliant and can operate in either programmed I/O or DMA mode.

  The parallel I/O port is available via the front panel and the VMEbus P2 connector of the SBus I/O board. The frontpanel connector is a standard Centronics 25-pin DSUB connector.

IOBP-10

All interfaces described above are available on the IOBP-10, an I/O panel for the SPARC/CPU-20VT (see section 3.4.6 “IOBP-10” on page 49). The IOBP-10 is available from Force Computers as a separate price list item.

Physical Address Ranges

For information on the physical address range, see table 22 “FGA-5100E: SPARC/CPU-20VT Physical Address Map” on page 89 and table 23

204223 8 – 0 May 2000

"FGA-5000, FGA-5100C: SPARC/CPU-20VT Physical Address Map" on page 91, respectively.

Data Sheet

For detailed information, see "Ethernet, SCSI, Parallel-I/O – MACIO STP2000 (NCR89C100). Main features are described below.

MACIO Features

The MACIO is an SBus I/O chip which has full SBus master capabilities. It integrates high performance I/O macrocells and logic including an Ethernet controller core, a fast 53C9X SCSI core, a high-speed parallel I/O port, a LS64854-compatible DMA2 controller, and a glueless 20 MHz-SBus interface. It concurrently supports:

- 10 MByte/s SCSI transfers
- 3.3 MByte/s parallel port transfers
- 1.25 MByte/s Ethernet transfers

It provides a 64-Byte FIFO for SCSI and parallel port data and supports SBus burst modes: four-word, eight-word, and "no/burst".

The DMA2 block includes the logic used to interface SCSI, Ethernet, and parallel I/O 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 I/O port. The DMA2 incorporates an improved cache and FIFO draining algorithm which allows better SBus utilization than previous DMA implementations.

4.7 SBus-to-EBus Controller – SEC

The SEC is the interface between the 32-bit wide SBus and the EBus, an 8-bit wide local I/O bus. The SEC is an SBus slave device which provides all the control signals for the 8-bit wide devices on the EBus:

- Two flash memory devices as boot PROM
- Two flash memory devices as user flash
- One RTC/NVRAM – MK48T18
- One high speed floppy disk controller – 82077SL
- Three dual channel serial controllers – SCC 85C30
- Audio controller – Am79C30A
- Two FPGAs (LCAs XC4003 and XC3030) for general board and temperature sensor control

In addition, the SEC provides the counters, timers and interrupt logic which is necessary for the MBus multi-processor system. The SEC also contributes to the system reset logic of the SPARC/CPU-20VT.

SEC Features

- Interface between 32-bit wide SBus and 8-bit wide EBus
- Control of all the 8-bit wide devices
- Interrupt controller
- System reset control
- Programmable 22-bit counters and timers

Data Sheet

The following sections describe the SEC Interrupt Controller, the NMI (Nonmaskable Interrupt), and the SEC Interrupt Mapping. For further information, see “SBus-to-Ebus Interface – SEC STP2014.

4.7.1 SEC Interrupt Controller

The SEC provides the SPARC/CPU-20VT interrupt controller. The interrupt controller fully supports the MBus multi-processor architecture. It provides software interrupts on all interrupt levels. In addition, the SEC monitors the hardware interrupts and translates them into the MBus processor interrupts according to a hardwired encoding scheme. The following devices have interrupt capabilities:

- Both MBus modules
- RMC
- M2S

204223 8 – 0 May 2000

- Both SBus modules
- Both MACIO devices
- FGA-5000
- Floppy disk controller
- All three SCC devices
- Temperature sensors

For each of the maximally four MBus processors the SEC provides a group of four interrupt lines. The four interrupt lines are used for encoding interrupt levels between 0 and 15. Level 15 is the highest interrupt level, whereas level 0 indicates that no interrupt is pending. The interrupts generated by the SEC can be categorized into directed, undirected and broadcast interrupts. For information on the processor interrupt levels generated by the corresponding hardware and software interrupts, see table 30 “SEC Interrupt Mapping” on page 108.

4.7.2 NMI (Nonmaskable Interrupt)

On the SPARC/CPU-20VT the MBus level 15 interrupt is the nonmaskable interrupt (NMI). The NMI can be triggered by the M2S, the RMC, the abort key, the temperature sensors or by the MBus modules via the Aerr IRQ (asynchronous error).

4.7.3 SEC Interrupt Mapping

The mapping of the VME interrupts and the abort switch interrupt is handled inside the FGA-5000. For a detailed description please refer to the *FGA-5000 Reference Guide*.

The mapping of the temperature sensor interrupts is handled inside one of the LCAs of the SPARC/CPU-20VT. For a detailed description, see section 4.9.4 “Temperature Sensors” on page 128.

Note: **The VME interrupts VME_IRQ<7..1>, the abort switch interrupt and the temperature sensor interrupts can be mapped to any of the seven SBus IRQ<7..1> or to the NMI (IRQ level 15).**

The following table shows the processor interrupt levels generated by the corresponding hardware and software interrupts.

Table 30 SEC Interrupt Mapping

IRQ Level	Interrupt Sources
0	No interrupt pending
1	Soft interrupt<1>
2	Soft interrupt<2>, SBus IRQ<1>, FGA-5000 interrupts, temperature sensor interrupt
3	Soft interrupt<3>, SBus IRQ<2>, FGA-5000 interrupts, temperature sensor interrupt, Centronics IRQ
4	Soft interrupt<4>, SCSI IRQ
5	Soft interrupt<5>, SBus IRQ<3>, FGA-5000 interrupts, temperature sensor interrupt
6	Soft interrupt<6>, Ethernet IRQ
7	Soft interrupt<7>, SBus IRQ<4>, FGA-5000 interrupts, temperature sensor interrupt
8	Soft interrupt<8>
9	Soft interrupt<9>, SBus IRQ<5>, FGA-5000 interrupts, temperature sensor interrupt, MBus module IRQ
10	Soft interrupt<10>, system timer IRQ
11	Soft interrupt<11>, SBus IRQ<6>, FGA-5000 interrupts, temperature sensor interrupt, floppy IRQ
12	Soft interrupt<12>, serial IRQ, audio IRQ, key-board/mouse IRQ
13	Soft interrupt<13>, SBus IRQ<7>, FGA-5000 interrupts, temperature sensor interrupt
14	Soft interrupt<14>, processor timer IRQ
15	Soft interrupt<15>, MBus Aerr IRQ, M2S IRQ, RMC IRQ, FGA-5000 interrupts, temperature sensor interrupt

2042238 – 0 May 2000

4.8 Local I/O Devices

The SEC provides all the control signals for the 8-bit wide devices on the EBus (see section 4.7 “SBus-to-EBus Controller – SEC” on page 106):

- Two flash memory devices as boot PROM – in three- or four-slot configurations of the SPARC/CPU-20VT on the SBus I/O board, whereas in two-slot configurations on the base board (see section 4.8.2 “Boot PROM” on page 111)
- Two flash memory devices as user flash – on the base board (see section 4.8.3 “User Flash” on page 112)
- One RTC/NVRAM – MK48T18 – in three- or four-slot configurations of the SPARC/CPU-20VT on the SBus I/O board, in two-slot configurations of the SPARC/CPU-20VT on the base board (see section 4.8.5 “RTC / NVRAM – MK48T18” on page 116)
- One high speed floppy disk controller – 82077SL – on the base board (see section 4.8.6 “Floppy Disk Interface – FDC” on page 117)
- Three dual channel serial controllers – SCC 85C30:
  - Two on the base board for the base board’s serial I/O ports A, B, and keyboard / mouse (see section 4.8.7 “Serial I/O Ports A, B, C, and D – SCC” on page 118 and section 4.8.9 “Keyboard and Mouse Port – SCC” on page 120)
  - One on the SBus I/O board for the SBus I/O board’s serial I/O ports C and D (see section 4.8.7 “Serial I/O Ports A, B, C, and D – SCC” on page 118)
- An audio controller – Am79C30A (see section 4.8.8 “Audio Port – Am79C30A (Factory Option)” on page 119),
- Two FPGA devices (LCAs XC4003 and XC3030), the LCA XC4003 on the base board (see section 4.8.10 “LCA for General Board Control” on page 121) and the LCA XC3030 on the SBus I/O board (see section 4.8.11 “LCA for Temperature Sensor Control” on page 122).

4.8.1 Address Map of Local I/O Devices

Table 31 Physical Address Map of Local I/O Devices

Physical Address	Device
$F.F000.0000_{16}$ …$F.F003.FFFF_{16}$	Boot PROM device 1 (256 Kbyte): read operation only
$F.F004.0000_{16}$ …$F.F007.FFFF_{16}$	Boot PROM device 2 (256 Kbyte): read operation only
$F.F100.0000_{16}$	Mousecontrol port
$F.F100.0002_{16}$	Data port
$F.F100.0004_{16}$	Keyboardcontrol port
$F.F100.0006_{16}$	Data port
$F.F110.0000_{16}$	Port B (serial I/O)control port
$F.F110.0002_{16}$	Data port
$F.F110.0004_{16}$	Port A (serial I/O)control port
$F.F110.0006_{16}$	Data port
$F.F110.0008_{16}$	Port D (serial I/O)control port
$F.F110.000A_{16}$	Data port
$F.F110.000C_{16}$	Port C (serial I/O)control port
$F.F110.000E_{16}$	Data port
$F.F120.0000_{16}$ …$F.F121.FFFF_{16}$	RTC / NVRAM
$F.F122.0000_{16}$ …$F.F123.FFFF_{16}$	LCA XC3030 for temperature sensor control and address decoding for audio (for a detailed register overview, see table 47 “Register Map of the LCA XC3030 – Temp. Sensor and Audio” on page 122)
$F.F124.0000_{16}$	LCA XC4003 for general board control (for a detailed register overview, see table 45 “Register Map of the LCA XC4003 – General Board Control” on page 121)
$F.F128.0000_{16}$ …$F.F12F.FFFF_{16}$	512 KByte Boot PROM and 4 MByte (max.) user flash: read and programming operation
$F.F170.0000_{16}$	Floppy disk controller

204223 8 – 0 May 2000

Table 31 Physical Address Map of Local I/O Devices (cont.)

Physical Address	Device
$F.F180.0000_{16}$	SEC auxiliary 1 register for control of termination of current floppy transfer and for floppy density sense control
$F.F1A0.1000_{16}$	Auxiliary 2 register for monitoring of MBus/SBus frequency and controlling software power down

4.8.2 Boot PROM

The boot PROM consists of two 2-Mbit (256K*8) flash memory devices.

Note: **The base board holds sockets for boot PROM devices and for the RTC/NVRAM. These sockets are only provided for downgrading to a 2-slot configuration (see section 3.2.2 “Two-, Three-, and Four-Slot SPARC/CPU-20VT Configurations” on page 18).**

- Ensure that no devices are assembled in these sockets when operating the SPARC/CPU-20VT in a three- or four-slot configuration.

In three- or four-slot configurations of the SPARC/CPU-20VT the boot PROM devices are installed on the SBus I/O board: device 1 in socket J9 and device 2 in J10. For the location and orientation (pin 1) of the boot PROM devices, see figure 6 “SBus I/O Board: Boot PROM and RTC/NVRAM Device Locations” on page 24.

Table 32 Physical Addresses for Boot PROM

Physical Address	Device
$F.F000.0000_{16}$ …$F.F003.FFFF_{16}$	Boot PROM device 1 (256 Kbyte): read operation only
$F.F004.0000_{16}$ …$F.F007.FFFF_{16}$	Boot PROM device 2 (256 Kbyte): read operation only
$F.F128.0000_{16}$ …$F.F12F.FFFF_{16}$	512 KByte Boot PROM and 4 MByte (max.) user flash: read and programming operation

4.8.3 User Flash

The user flash consists of two 8-Mbit (1M*8) flash memory devices. It can be used to store ROMable operating systems as well as application specific code.

Factory Option

The SPARC/CPU-20VT is already prepared for future factory option 16-Mbit (2M*8) and 32-Mbit (4M*8) flash memory devices.

Table 33 Physical Addresses for User Flash

Physical Address	Device
F.F128.0000$_{16}$ …F.F12F.FFFF$_{16}$	512 KByte Boot PROM and 4 MByte (max.) user flash: read and programming operation

4.8.4 Programming the Boot PROM and the User Flash

On the SPARC/CPU-20VT 2.5 MByte flash memory is available (512 KByte boot PROM and 2 MByte user flash). The 2.5-MByte flash memory can be accessed in 512-KByte pages (programming window) for read and write operation. Additionally, the boot PROM is accessible in the boot PROM address range for read only operations.

Write-Protection

Boot PROM and user flash can be write-protected:

- The boot PROM is write-protected if the base board switch B_SW8-1 is set appropriately (default “OFF”, see page 37): OFF = write-protected, ON = writing enabled.
- The user flash is write-protected if the B_SW8-2 is set appropriately (default “OFF”, see page 37): OFF = write-protected, ON = writing enabled.

Note: **If power fails during on-board reprogramming, the contents of the boot PROM may be lost. If so, it is necessary to uninstall the boot PROM devices from their sockets and program them in a standard EPROM programmer. Therefore, before programming the boot PROM devices on-board, save the contents of the boot PROM devices for reprogramming purposes.**

204223 8 – 0 May 2000

Programming Window

The size of the programming window is limited to 512 KByte, that means that only 512 KByte of the 2.5-MByte flash memory area can be accessed contiguously.

The following registers are involved during flash memory programming:

- The boot ROM size control register controls the decoding of the boot PROM devices (see table 34 “Boot ROM Size Control Register (BOOT_ROM_SIZE_CTRL)” on page 113).
- The flash programming control registers one and two are used to map 512 KByte of the 2.5-MByte flash memory area into the programming window:
  - For information on defining the programming window, see table 37 “Flash Memory Selection Control Bits (x = don’t care)” on page 114.
  - For information on the two registers, see page 113.
- The flash programming voltage control register is used to control the +12V programming voltage (see table 38 “Flash Programming Voltage Control Register (FLASH_VPP_CTRL)” on page 115).

Table 34 Boot ROM Size Control Register (BOOT_ROM_SIZE_CTRL)

F.F124.0008$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	1 r	1 r	1 r	256K SEL r/w

256K_SEL (R/W) 256K_SEL selects the size of the flash memories installed in the boot PROM sockets.

= 0 512K*8 (4Mbit) flash memory devices are used as boot PROM.

= 1 256K*8 (2Mbit) flash memory devices are used as boot PROM.

Table 35 Flash Programming Control Register 2 (FLASH_CTRL2)

F.F124.0009$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	1 r	1 r	1 r	B R S r

BOOT_ROM_-SELECT (R/W)

BOOT_ROM_SELECT selects either the boot PROM or the user flash.

= 0 Selects the user flash devices for programming.

= 1 Selects the boot PROM devices for programming.

Table 36 Flash Programming Control Register 1 (FLASH_CTRL1)

F.F124.0002$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	A[21:19] r/w			SELECT_ROM r/w

A[21:19] (R/W)

A[21:19] control the address lines A21 … A19 of the user flash devices if selected by BOOT_ROM_SELECT (R/W) in the Flash Programming Control Register 2 (FLASH_CTRL2). By means of these address lines, a specific 512-KByte page is selected from the user flash area and mapped into the 512-KByte programming window. This enables addressing of up to four MByte per user PROM device.

SELECT_ROM (R/W)

SELECT_ROM selects one of the two devices of the boot EPROM or the user flash (depending on the one being selected by BOOT_ROM_SELECT (R/W) in the Flash Programming Control Register 2 (FLASH_CTRL2).

= 0 Selects device 1 for programming.

= 1 Selects device 2 for programming.

Programming Window selectIon

The table below shows the valid bit settings for the FLASH_CTRL2 and FLASH_CTRL1 registers above. By default two 1-Mbyte devices of user flash memory are installed. That means that only bit A19 is relevant to select one of the two 512-KByte pages of each device. A20 and A21 are don't care bits. A20 and A21 are for use with the user flash factory option (see section 4.8.3 "User Flash" on page 112).

Table 37 Flash Memory Selection Control Bits (x = don't care)

BOOT_ROM_SELECT	**SELECT_ROM**	**A[…]**			**Programs**
		21	**20**	**19**	
Boot PROM:					
1	0	x	x	x	device 1with 256 KByte
1	1	x	x	x	device 2with 256 KByte
User flash:					

204223 8 – 0 May 2000

Table 37 Flash Memory Selection Control Bits (x = don't care) (cont.)

BOOT_ ROM_ SELECT	SELECT _ROM	A[…] 21	20	19	Programs
0	0	x	x	0	device 11st 512 KByte
0	0	x	x	1	2nd 512 KByte
0	1	x	x	0	device 21st 512 KByte
0	1	x	x	1	2nd 512 KByte
0	0	0	0	0	Reset setting

Table 38 Flash Programming Voltage Control Register (FLASH_VPP_CTRL)

$F.F124.000A_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	1 r	1 r	1 r	VPPON r/w

VPPON (R/W) VPPON controls the +12V programming voltage for the flash memories.

= 0 Default setting: Programming voltage turned off.

= 1 Programming voltage turned on.

4.8.5 RTC / NVRAM – MK48T18

The RTC / NVRAM is implemented by an MK48T18 device.

Note: **The base board holds sockets for boot PROM devices and for the RTC/NVRAM. These sockets are only provided for downgrading to a 2-slot configuration (see section 3.2.2 “Two-, Three-, and Four-Slot SPARC/CPU-20VT Configurations” on page 18). Ensure that no devices are assembled in these sockets when operating the SPARC/CPU-20VT in a three- or four-slot configuration.**

In three- or four-slot configurations of the SPARC/CPU-20VT the RTC / NVRAM is installed on the SBus I/O board. For the location and orientation (pin 1) of the device, see figure 6 “SBus I/O Board: Boot PROM and RTC/NVRAM Device Locations” on page 24.

The MK48T18 combines an 8K * 8 full CMOS SRAM, a bytewide accessible real-time clock, a crystal, and a long-life lithium carbon monofluoride battery in a single plastic DIP package.

Data Sheet — For further information, see “Real-time Clock and NVRAM – RTC/NVRAM MK48T18.

Table 39 Physical Addresses for RTC/NVRAM

Physical Address	**Device**
$\texttt{F.F120.0000}_{16}$	RTC / NVRAM

204223 8 – 0 May 2000

4.8.6 Floppy Disk Interface – FDC

The floppy disk interface is implemented by the 82077SL floppy disk controller. The 82077SL is able to transfer data at data rates of 250, 300, or 500 Kbit/s, or at 1 Mbit/s. It integrates drivers, receivers, a data separator, and a 16-Byte bidirectional FIFO. The floppy disk controller supports all standard disk formats (typically 720 KByte and 1.44 MByte) and also the 2.88 MByte and 4 MByte floppy format.

Data Sheet — For further information, see “Floppy Disk Controller – FDC 82077SL.

P2 and IOBP-10 — The floppy disk interface is available on the base board’s VMEbus P2 connector and on the IOBP-10, an I/O panel for the SPARC/CPU-20VT (see section 3.4.6 “IOBP-10” on page 49). The IOBP-10 is available from Force Computers as a separate price list item.

Table 40 Physical Addresses for Floppy Disk Controller

Physical Address	Device
F.F170.0000$_{16}$	Floppy disk controller
F.F180.0000$_{16}$	SEC auxiliary 1 register for control of termination of current floppy transfer and for floppy density sense control

The floppy disk interface on the SPARC/CPU-20VT has two additional control pins: the FLOPPY_TC pin and the FLOPPY_DENSENSE pin:

Table 41 SEC Auxiliary 1 Register

F.F124.000A$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	FLOPPY DEN SENSE r	1 r	1 r/w	FLOPPY TC r/w	1 r/w	1 r/w

FLOPPY-DENSENSE (R) — FLOPPYDENSENSE indicates whether the density of the disk is high or low. It is the status of the signal on the FLOPPY_DENSENSE pin, which is an output pin on the floppy drive.

FLOPPYTC (R/W) — FLOPPYTC controls termination of the current disk transfer. It controls the status of the FLOPPY_TC pin, which is an input pin on the 82077SL.

4.8.7 Serial I/O Ports A, B, C, and D – SCC

On the base board the two independent full-duplex serial I/O ports A and B are available at a 26-pin shielded connector on the front panel. They are implemented by the SCC 1 on the base board. Similarly the SBus I/O board's SCC 3 provides the two serial I/O ports C and D. RS-232 and RS-422 configurations are supported via assembly options. By default, all four serial I/O ports are configured as RS-232.

If configured as RS-232, all four ports are also available on the VMEbus P2 connector, each with the signals RXD, TXD, RTS and CTS: A and B on the base board's P2 connector and C and D on the SBus I/O board's P2 connector. Via the P2 availability, all four serial I/O ports are also available on an IOBP-10, an I/O panel for the SPARC/CPU-20VT (see section 3.4.6 "IOBP-10" on page 49). The IOBP-10 is available from Force Computers as a separate price list item.

Table 42 Physical Addresses for Serial I/O A and B

Physical Address	Device
$F.F110.0000_{16}$	Port B (serial I/O)control port
$F.F110.0002_{16}$	Data port
$F.F110.0004_{16}$	Port A (serial I/O)control port
$F.F110.0006_{16}$	Data port
$F.F110.0008_{16}$	Port D (serial I/O)control port
$F.F110.000A_{16}$	Data port
$F.F110.000C_{16}$	Port C (serial I/O)control port
$F.F110.000E_{16}$	Data port

For information on changing the configuration and on the front-panel connector pinout, see section 3.4.1 "Serial I/O Port RS-232 and RS-422 Connector Pinout" on page 44.

Data Sheet For further information, see "Serial I/O and Keyboard / Mouse – SCC AM 85C30.

204223 8 – 0 May 2000

4.8.8 Audio Port – Am79C30A (Factory Option)

On the SBus I/O board's SCC 3 provides the two serial I/O ports C and D. RS-232 and RS-422 configurations are supported via assembly options. By default, all four serial I/O ports are configured as RS-232. There is an SBus I/O board variant with installed audio factory option supporting

- Two audio inputs for one passive microphone (Microphone+ and Microphone–)
- One dc-free output (> 540 Ω impedance and < 100 pF capacitive load) for one earpiece
- Two differential push-pull outputs (> 40 Ω impedance and < 100 pF capacitive load) for one loud speaker

Table 43 Physical Addresses for Audio Port

Physical Address	Device
F.F123.0000$_{16}$ ...F.F123.FFFF$_{16}$	Audio controller

Although the audio controller is accessible via the EBus, the audio controller's interrupt line is routed directly to the SBus as SB_INT7.

Data Sheet For detailed information, see "Audio Controller – Am79C30A.

For further information (for example, on technical data and on the front-panel connector pinout), see section 3.4.3 "Audio Connector Pinout (Factory Option)" on page 46.

4.8.9 Keyboard and Mouse Port – SCC

The keyboard and mouse port is available on the base board's front panel via an 8-pin mini DIN connector. The pinout of the keyboard and mouse connector on the front panel is described in section 3.4.2 "Keyboard/Mouse Connector Pinout" on page 46.

The following keyboard and mouse port signals are available on the base board's or SBus I/O board's VMEbus P2 connector: KEYBOARD_IN, KEYBOARD_OUT, MOUSE_IN, and MOUSE_OUT.

- The signals KEYBOARD_IN, KEYBOARD_OUT and MOUSE_IN are available on the base board's P2.
- Whereas MOUSE_OUT is available on the SBus I/O board's P2.

Via the P2 availability, the keyboard / mouse signals are also available on an IOBP-10, an I/O panel for the SPARC/CPU-20VT (see section 3.4.6 "IOBP-10" on page 49). The IOBP-10 is available from Force Computers as a separate price list item.

The keyboard and mouse port is controlled via one of two dual-channel serial controllers (SCC) assembled on the base board. Channel A provides the keyboard control and channel B the mouse control.

Data Sheet For further information, see "Serial I/O and Keyboard / Mouse – SCC AM 85C30.

Table 44 Physical Addresses for Keyboard / Mouse

Physical Address	Device
F.F100.0000$_{16}$	Mousecontrol port
F.F100.0002$_{16}$	Data port
F.F100.0004$_{16}$	Keyboardcontrol port
F.F100.0006$_{16}$	Data port

204223 8 – 0 May 2000

4.8.10 LCA for General Board Control

The logic cell array XC4003 (LCA) is an FPGA for general board control. The following control functions are implemented:

- Address decoding for local I/O devices
- Flash memory control
- Front panel diagnostic control
- FGA-5000 BERR and FMB control

It is configured during powering up by a serial PROM which is located in base board socket J7.

Table 45 Register Map of the LCA XC4003 – General Board Control

Physical Address	Reset Value	Register
$F.F124.0000_{16}$	$F0_{16}$	User LED 1 control register (see page 125)
$F.F124.0001_{16}$	$F0_{16}$	User LED 2 control register (see page 125)
$F.F124.0002_{16}$	$F0_{16}$	Flash programming ctrl. reg. 1 (see page 114)
$F.F124.0003_{16}$	FX_{16}	Rotary switch status register (see page 127)
$F.F124.0004_{16}$	FC_{16}	ID PROM control register for Force Computers internal use only
$F.F124.0005_{16}$	XX_{16}	B_SW5-*x* and B_SW4-*x* status register (see page 132)
$F.F124.0006_{16}$	XX_{16}	B_SW7-1/2 and B_SW6-1/2/3 status register (see page 133)
$F.F124.0007_{16}$	FX_{16}	B_SW8-*x*, B_SW7-3, and B_SW6-4 status register (see page 133)
$F.F124.0008_{16}$	FE_{16}	Boot PROM size control register (see page 113)
$F.F124.0009_{16}$	FE_{16}	Flash programming ctrl. reg. 2 (see page 113)
$F.F124.000A_{16}$	FE_{16}	Flash programming voltage ctrl. reg. (see page 115)
$F.F124.000B_{16}$	FF_{16}	Seven segment LED display control register (see page 126)
$F.F124.000C_{16}$	FE_{16}	FMB channel 0 data discard status register (see page 102)
$F.F124.000D_{16}$	FE_{16}	FMB channel 1 data discard status register (see page 102)
$F.F124.000E_{16}$	FF_{16}	reserved
$F.F124.000F_{16}$	FX_{16}	LCA identification register (see below)

Table 46 LCA XC4003 Identification Register

F.F124.000F$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	REV r			

REV (R) REV indicates the revision of the LCA.

= F_{16} This value is not valid.

= E_{16} Revision 1

= D_{16} Revision 2

...

4.8.11 LCA for Temperature Sensor Control

The logic cell array XC3030 (LCA) is an FPGA used to control the temperature sensors located on the SBus I/O board. It is configured during powering up by a serial PROM which is located in SBus I/O board socket J28.

Table 47 Register Map of the LCA XC3030 – Temp. Sensor and Audio

Physical Address	Reset Value	Register
F.F122.0000$_{16}$	FE$_{16}$	Temperature sensor select register (see page 129)
F.F122.0001$_{16}$	F7$_{16}$	Temperature sensor communication register (see page 129)
F.F122.0002$_{16}$	F0$_{16}$	Temperature sensor interrupt status register (see page 130)
F.F122.0003$_{16}$	F0$_{16}$	Temperature sensor status register (see page 130)
F.F122.0004$_{16}$	F0$_{16}$	Temperature sensor interrupt control register (see page 131)
F.F122.0005$_{16}$	F8$_{16}$	Temperature sensor interrupt level register (see page 132)
F.F122.0006$_{16}$ …F.F122.000B$_{16}$	FF$_{16}$	Reserved
F.F122.000C$_{16}$	FE$_{16}$	ID PROM control register for Force Computers internal use only
F.F122.000D$_{16}$	FE$_{16}$	I_SW2 and I_SW1 status register (see page 133)
F.F122.000E$_{16}$	FF$_{16}$	Hardware identification register

204223 8 – 0 May 2000

Table 47 Register Map of the LCA XC3030 – Temp. Sensor and Audio (cont.)

Physical Address	Reset Value	Register
F.F122.000F$_{16}$	FX$_{16}$	LCA identification register (see below)
F.F123.0000$_{16}$ …F.F123.FFFF$_{16}$	See Data Sheet	Audio controller (see page 119)

Table 48 LCA XC3030 Identification Register

F.F122.000F$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	REV r							

REV (R) REV indicates the revision of the LCA.

= FF$_{16}$ This value is not valid.

= FE$_{16}$ Revision 1

= FD$_{16}$ Revision 2

...

4.9 Front-Panel and Switch Related Features and Registers

The SPARC/CPU-20VT provides additional features for diagnostic and control purposes. These features include:

- The reset key and the abort key (see table 7 “Front Panel Features” on page 40)
- The RUN LED: It is either red, green or blinking. This LED is red when any reset signal on the board is active. This LED begins blinking when SB_SEL for the SEC is inactive for more than 0,5 s in order to signal a hangup. In all other cases, this LED is green.
- The BM LED: It reflects all VMEbus master activities of the SPARC/CPU-20VT. When the board accesses the VMEbus, the BM LED lights up green. The BM LED turns red when the SPARC/CPU-20VT is asserting SYSFAIL to the VMEbus.
- Two programmable LEDs: the user LEDs 1 and 2 are user programmable controlled by LCA registers (see section 4.9.1 “User LED 1 and 2 Control Registers” on page 125).
- The 7-segment display (see section 4.9.2 “Seven Segment LED Display” on page 126)
- The hexadecimal rotary switch (see section 4.9.3 “Rotary Switch” on page 127)
- Two temperature LEDs (see table 47 “Register Map of the LCA XC3030 – Temp. Sensor and Audio” on page 122 and section 4.9.4 “Temperature Sensors” on page 128)

204223 8 – 0 May 2000

4.9.1 User LED 1 and 2 Control Registers

Two user programmable user LEDs on the base board's front panel allow for user-defined diagnostic features. They can be accessed via the two user LED control registers.

Table 49 User LED 1 and 2 Control Registers

F.F124.0000$_{16}$ for User LED 1; F.F124.0001$_{16}$ for User LED 2								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	BLINK_FREQ r/w		COLOR r/w	

BLINK_FREQ (R/W) BLINK_FREQ controls the frequency at which the User LED is blinking.

- = 00_2 Blinking is disabled.
- = 01_2 Blinking frequency is 0.5 Hz.
- = 10_2 Blinking frequency is 1 Hz.
- = 11_2 Blinking frequency is 2 Hz.

COLOR (R/W) COLOR controls the status and the color of the User LED.

- = 00_2 The User LED is turned off.
- = 01_2 The User LED is turned on green.
- = 10_2 The User LED is turned on red.
- = 11_2 The User LED is turned on yellow.

4.9.2 Seven Segment LED Display

A user programmable seven-segment LED display on the base board's front panel allows for user-defined diagnostic features. It can be accessed via the seven-segment LED display control register.

Table 50 Seven-Segment LED Display Control Register (SEV_SEG_CTRL)

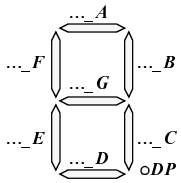
F.F124.000B$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	DP W	SEG_G W	SEG_F W	SEG_E W	SEG_D W	SEG_C W	SEG_B W	SEG_A W

DP and SEG_G … SEG_A(W) The bits control the status of the decimal point (DP) and the segments (SEG_G…SEG_A) in the hexadecimal display (see figure below for naming conventions).

= 0 The respective part of the display is turned off.

= 1 The respective part of the display is turned on.

Figure 22 Naming the parts of the hexadecimal display



204223 8 – 0 May 2000

4.9.3 Rotary Switch

A rotary switch on the base board's front panel allows for user-defined parameterizing of user applications. Its setting can be read via the rotary switch status register.

Table 51 Rotary Switch Status Register (ROTARY_SWITCH_STAT)

F.F124.0003$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	ROT_SWI[3:0] r			

ROT_SWI[3:0] (R) ROT_SWI[3:0] indicates the setting of the rotary switch.

= 0000_2 The setting of the rotary switch is 0_{16}.

= 0001_2 The setting of the rotary switch is 1_{16}.

…

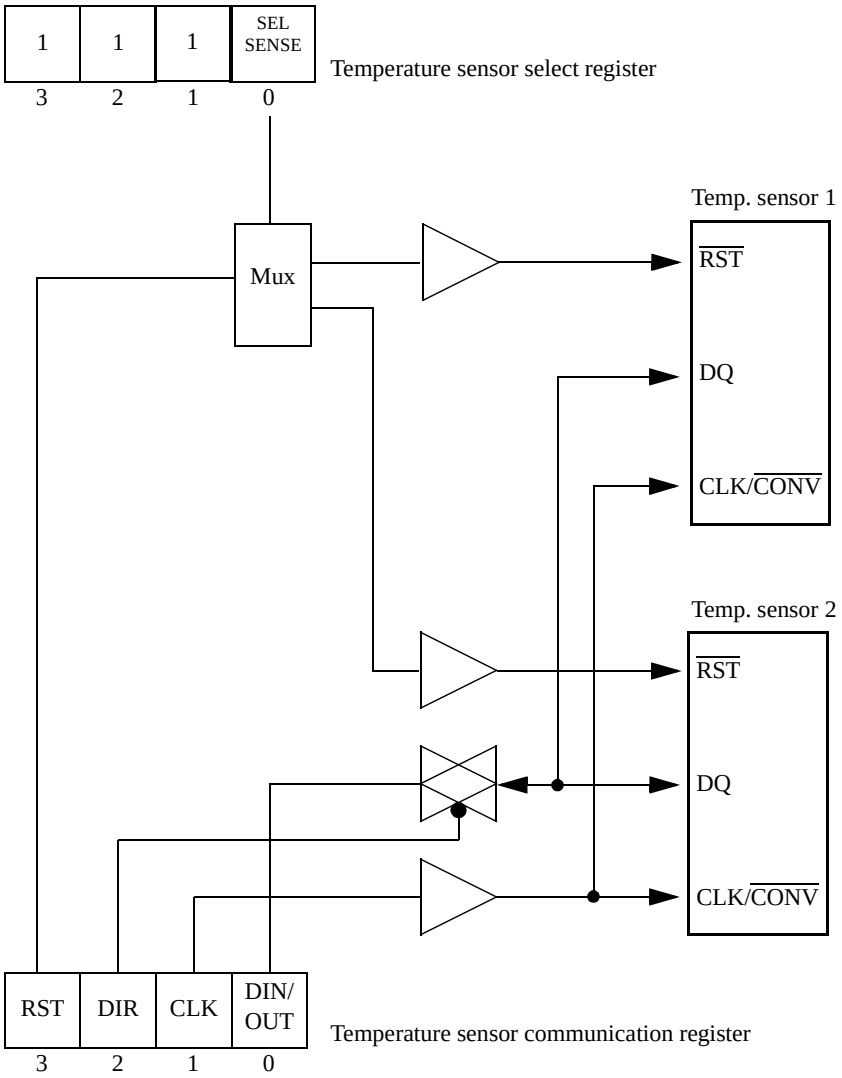
= 1110_2 The setting of the rotary switch is E_{16}.

= 1111_2 The setting of the rotary switch is F_{16}.

4.9.4 Temperature Sensors

Two digital thermometer and thermostat devices are assembled on the bottom side of the SBus I/O board, close to the MBus modules (see figure 9 “Location Diagram of the SBus I/O Board (Schematic)” on page 31). The processor communicates with these devices across a three-wire interface as shown in the figure below. These three lines are controlled by software.

Figure 23 Register Interface of the Temperature Sensors



Data sheet For further information, see “Digital Thermometer and Thermostat – DS1620.

204223 8 – 0 May 2000

Table 52 Temperature Sensor Select Register

F.F122.0000$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	1 r	1 r	1 r	SEL_ SENSE r/w

SEL_SENSE (R/W) SEL_SENSE selects one of the two temperature sensors. The selected temperature sensor is accessible via the temperature sensor communication register.

= 0 Temperature sensor 1 is selected.

= 1 Temperature sensor 2 is selected.

Table 53 Temperature Sensor Communication Register

F.F122.0001$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	RST r/w	DIR r/w	CLK r/w	DIN/ OUT r/w

Note: **In the following bit descriptions of the temperature sensor communication register, the selected temperature sensor is the one, selected via the temperature sensor select register.**

RST (R/W) RST controls whether the selected temperatur sensor is reset.

= 0 The selected temperature sensor is kept in the reset state.

= 1 The selected temperature sensor is ready for operation.

DIR (R/W) DIR controls controls the direction of DIN/OUT.

= 0 DIN/OUT is configured as output and the setting of DIN/OUT is forced on the data line connected to the selected temperature sensor: 0 = low level on data line, 1 = high level on data line.

= 1 DIN/OUT is configured as input and DIN/OUT indicates the status of the data line connected to the selected temperature sensor.

CLK (R/W) — CLK controls the status of the clock line connected to the selected temperature sensor.

= 0 Low level is forced on the clock line.

= 1 High level is forced on the clock line.

DIN/OUT (R/W) — DIN/OUT depends on the setting of DIR (see above).

Table 54 Temperature Sensor Interrupt Status Register

F.F122.0002$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	IP_TH2 r/w	IP_TL2 r/w	IP_TH1 r/w	IP_TL1 r/w

IP_TH2 (R/W) — IP_TH2 indicates whether an interrupt is pending due to temperature sensor 2 measuring a temperature which exceeds the upper temperature limit T_{high}.

= 0 No interrupt is pending for temperature sensor 2.

= 1 An interrupt is pending for temperature sensor 2 and the interrupt is cleared by clearing this bit.

IP_TL2 (R/W) — Same as IP_TH2 except that the temperature has sunk below the lower temperature limit T_{low}.

IP_TH1 (R/W) — Same as IP_TH2 except that temperature sensor 1 is affected.

IP_TL1 (R/W) — Same as IP_TL2 except that temperature sensor 1 is affected.

Table 55 Temperature Sensor Status Register

F.F122.0003$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	STAT_TH2 r/w	STAT_TL2 r/w	STAT_TH1 r/w	STAT_TL1 r/w

STAT_TH2 (R/W) — STAT_TH2 indicates whether the measured temperature $T_{measured}$ by temperature sensor 2 exceeded the upper temperature limit T_{high}.

= 0 $T_{measured} < T_{high}$.

= 1 $T_{measured} > T_{high}$.

204223 8 – 0 May 2000

STAT_TL2 (R/W) STAT_TL2 indicates whether the measured temperature $T_{measured}$ by temperature sensor 2 sank below the lower temperature limit T_{low}.

= 0 $T_{measured} > T_{low}$.

= 1 $T_{measured} < T_{low}$.

STAT_TH1 (R/W) Same as STAT_TH2 except that temperature sensor 1 is affected.

STAT_TL1 (R/W) Same as STAT_TL2 except that temperature sensor 1 is affected.

Table 56 Temperature Sensor Interrupt Control Register

F.F122.0004$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	IE_ TH2 r/w	IE_ TL2 r/w	IE_ TH1 r/w	IE_ TL1 r/w

IE_TH2 (R/W) IE_TH2 controls the interrupt generation in the case that the temperature $T_{measured}$ measured by temperature sensor 2 exceeds the upper temperature limit T_{high}.

= 0 Interrupt generation disabled for $T_{measured}$ exceeding T_{high}.

= 1 Interrupt generation enabled for $T_{measured}$ exceeding T_{high}.

IE_TL2 (R/W) IE_TL2 controls the interrupt generation in the case that the temperature $T_{measured}$ measured by temperature sensor 2 falls below the lower temperature limit T_{low}.

= 0 Interrupt generation disabled for $T_{measured}$ falling below T_{low}.

= 1 Interrupt generation enabled for $T_{measured}$ falling below T_{low}.

IE_TH1 (R/W) Same as STAT_TH2 except that temperature sensor 1 is affected.

IE_TL1 (R/W) Same as STAT_TL2 except that temperature sensor 1 is affected.

Table 57 Temperature Sensor Interrupt Level Register

F.F122.0005$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1 r	1 r	1 r	1 r	1 r	INT_LEV[2:0] r/w		

`INT_LEV[2:0]` (R/W) `INT_LEV[2:0]` controls the level of the SBus interrupt that is generated when the temperature exceeds the upper or falls below the lower temperature limit.

= 111_2 SBus level 7 interrupt

= 110_2 SBus level 6 interrupt

…

= 001_2 SBus level 1 interrupt

= 000_2 Nonmaskable interrupt (NMI)

4.9.5 Switch Status Registers

Note: **The following base board's switch status registers all comply to the same value description:**

- If a bit is `0`, the respective switch is OFF.
- If a bit is `1`, the respective switch is ON.

Table 58 B_SW5-*x* and B_SW4-*x* Status Register

F.F124.0005$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	(serial I/O port B, see page 35) B_SW5…				(serial I/O port A, see page 34) B_SW4…			
Value	…-4 r	…-3 r	…-2 r	…-1 r	…-4 r	…-3 r	…-2 r	…-1 r

204223 8 – 0 May 2000

Table 59 B_SW7-1/2 and B_SW6-1/2/3 Status Register

F.F124.0006$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	(VMEbus slot 1, see page 36) B_SW7…				(SCSI #1 and abort key, see page 36) B_SW6…			
Value	reserved		…-2 r	…-1 r	reserved	…-3 r	…-2 r	…-1 r

Table 60 B_SW8-*x*, B_SW7-3, and B_SW6-4 Status Register

F.F124.0007$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
					(Boot PROM and user flash write-protection, see page 37; SYSRESET input, see page 36; reset key, see page 36)			
Value	1 r	1 r	1 r	1 r	B_SW8-2 r	B_SW8-1 r	B_SW7-3 r	B_SW6-4 r

Note: **The following SBus I/O board's switch status register complies to the following value description:**

- If a bit is 0, the respective switch is ON.
- If a bit is 1, the respective switch is OFF.

Table 61 I_SW2-*x* and I_SW1-*x* Status Register

F.F122.000D$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	… (CSI #2, see page 38; serial I/O port D, see page 39)				I_SW1… (serial I/O port C and SCSI #2, see page 38)			
Value	I_SW1-4 r	I_SW2-2 r	I_SW2-3 r	I_SW2-1 r	reserved	…-2 r	…-3 r	…-1 r

204223 8 – 0 May 2000

5 Force OpenBoot Enhancements

The OpenBoot ported to the SPARC/CPU-20VT is based upon OpenBoot 2.25 obtained from Sun Microsystems. This section describes the enhancements to the standard OpenBoot firmware that have been done for the SPARC/CPU-20VT. For a description of standard OpenBoot firmware features, 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-20VT includes further words for:

- Accessing and controlling the VMEbus interface, which is based on the FGA-5000 or, in case of the SPARC/CPU-20VTe, on the FGA-5100
- Accessing and programming available flash memories
- Controlling the operating mode of the Watchdog Timer
- Making use of the Diagnostic
- The extended feature set of the FGA-5100 as compared with the FGA-5000 (for examples, see pages 155, 159, 166, 184, 186, 186 and compare the address maps given on pages 89 and 91)

The following subsections describe the Force OpenBoot enhancements in detail, and examples are given when it seems necessary to convey the usage of a particular or a group of words.

Notation

In general, each word is described using the notation stated below:

`name` (*stack-comment*) description

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

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

parameters before execution — parameters after execution

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

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

5.1 Controlling the VMEbus Master and Slave Interface

VMEbus Addressing

The VMEbus has a number of distinct address spaces represented by a subset of the 64 possible values encoded by the six 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 six 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 $2D_{16}$ identifying the privileged *short* (A16) address space with 16-bit data transfers.

`vmea16d32` (— *h# 6d*) returns the `AML` constant $6D_{16}$ identifying the privileged *short* (A16) address space with 32-bit data transfers.

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

`vmea24d32` (— *h# 7d*) returns the `AML` constant $7D_{16}$ identifying the privileged *standard* (A24) address space with 32-bit data transfers.

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

`vmea32d32` (— *h# 4d*) returns the `AML` constant $4D_{16}$ 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 00.0000_{16} to $FF.FFFF_{16}$), whereas the *extended* (A32) address space allows to address 4 GByte (32-bit addresses ranging from 0000.0000_{16} to $FFFF.FFFF_{16}$), and the *short* (A16) address space is limited to 64 KByte (16-bit addresses ranging from 0000_{16} to $FFFF_{16}$).

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 4080.0000_{16}:

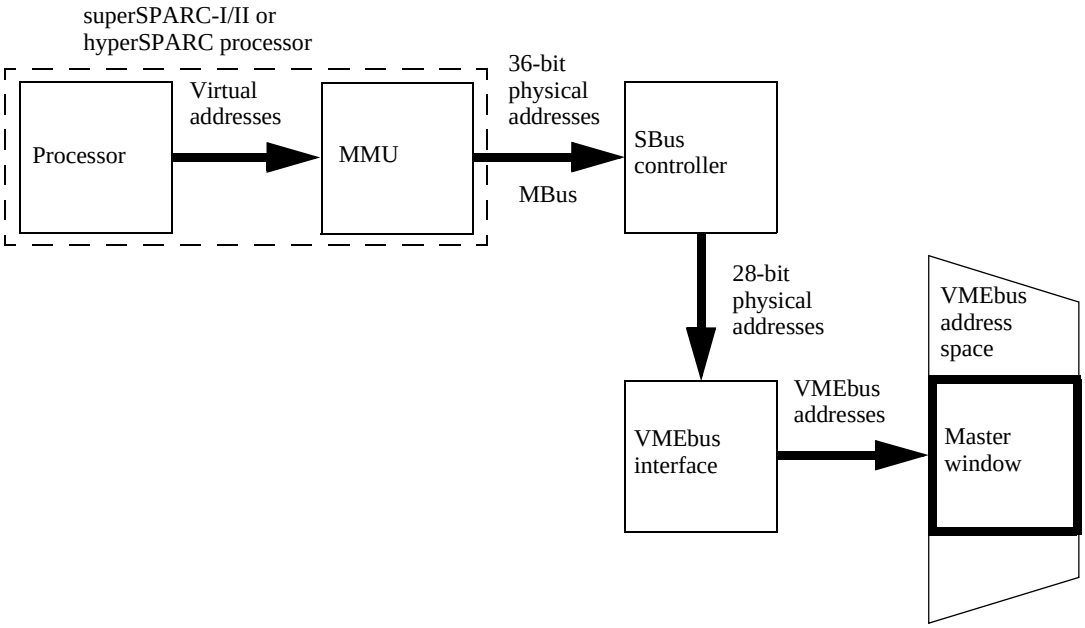
```
ok h# 4080.0000 vmea32d32
```

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

5.1.1 VMEbus Master Interface

As shown in the following figure, the processor emits virtual addresses during a data transfer cycle which are translated to 36-bit physical MBus addresses by the MMU. The VMEbus is connected to the SBus and the SBus controller (MBus-to-SBus bridge) translates 36-bit physical MBus addresses to 28-bit physical SBus addresses and asserts the specific SBus slot select signal. 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): Super- or Hypersparc – MBus – VMEbus



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

- [2]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.
- [2]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.
- [2]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. And the command `vme-free-virtual` removes the VMEbus area which has previously been made available 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.

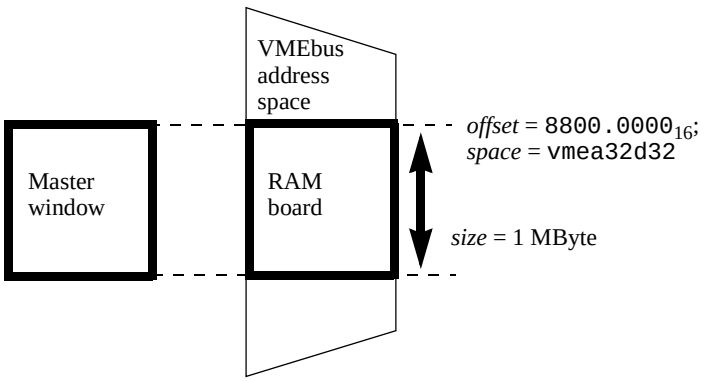
204223 8 – 0 May 2000

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 8800.0000_{16} and ranging to $880F.FFFF_{16}$ (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 8800.0000_{16} 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.

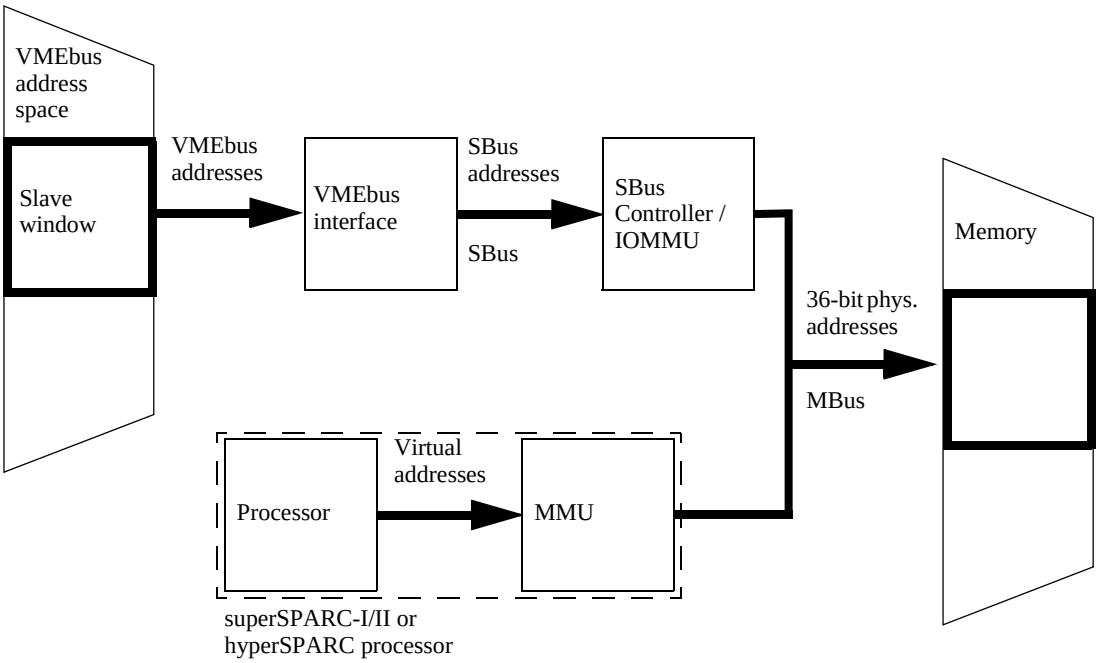
`vme-free-virtual` (*vaddr size* —) removes the VMEbus area associated with the virtual address *vaddr* from the processor's virtual address space. The VMEbus area previously made available to the processor's virtual address space is removed from the virtual address space using the `vme-free-virtual` command as shown below:

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

5.1.2 VMEbus Slave Interface

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 36-bit physical MBus addresses which address a certain area within the on-board memory. (Of course, the IOMMU may be set in such a way that the translated addresses address a participant residing on the SBus.)

Figure 26 **Address Translation (Slave): VMEbus – SBus – MBus – Super-/Hypersparc**



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. *standard* (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, emitted by the VMEbus interface during a slave access, with the physical addresses of the allocated memory. Further-

204223 8 – 0 May 2000

more, 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 one MByte of on-board memory should be made available to the *extended* (A32) address space of the VMEbus beginning at the VMEbus address 4080.0000_{16}, 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 one MByte beginning at physical address 4080.0000_{16} 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-sl
ok
```

5.2 VMEbus Interface

Note: **The result of some OpenBoot commands depends on the VMEbus interface chip being assembled on the CPU board under consideration (FGA-5000 and FGA-5100^{C} versus FGA-5100^{E}).**

The following property of the VMEbus device node and the following commands can be used to:

- Check which VMEbus interface chip is assembled on the CPU board under consideration
- Set the mode of the VMEbus interface chip in case of the FGA-5100 being assembled

Model — The `model` property indicates whether the FGA-5000 or the FGA-5100 is assembled on the CPU board.
When `model` equals `SPARC FGA-5000`, the FGA-5000 is assembled.
When `model` equals `SPARC FGA-5100`, the FGA-5100 is assembled.

`vsi-id2@` (— *id-code* `true` | `false`) returns the contents — the 8 bit data *id-code* — of the FGA-5100 Identification Register (`ID2`). When the data has been fetched successfully, the *id-code* $C6_{16}$ along with `true` is returned. `false` is returned in case of an FGA-5000 based CPU board.

vsi-enhanced? (— `true` | `false`) checks the current FGA--5100 operating mode. `true` indicates that the FGA-5100 operates in the FGA-5100 enhanced mode. `false` indicates that the FGA-5100 operates in the FGA-5000 compatibility mode.

vsi-enhanced! (`true` | `false` —) specifies the FGA-5100 mode to be set. When `true` is passed to the command, the FGA-5100 enhanced mode is turned on. When `false` is passed to the command, the FGA-5000 compatibility mode is turned on. In case of FGA-5000 based CPU boards an interrupt is generated.

204223 8 -- 0 May 2000

Generic Information

The variables described below are used to retrieve generic information about the VMEbus interface. They are declared as `value`.

`vsi-va` (— *vaddr*) returns the virtual base address *vaddr* of the FGA-5x00 registers.

`vsi-base-addr!` (*offset sbus-slot#* —) sets the FGA-5x00 base address according to the given SBus slot number *sbus-slot#* and the offset *offset* within the specified SBus slot.

`vsi-base-addr@` (— *offset sbus-slot#*) returns the base address of the FGA-5x00 represented by the SBus slot number *sbus-slot#* and the offset *offset* within the specific SBus slot.

5.2.1 FGA-5x00 Register Accesses

The commands described below are used to read data from and to store data in FGA-5x00 registers.

`vsi-sbus-base@` (— *long*) returns the contents – a 32 bit data – of the SBus Base Address Register (`SBUS_BASE`).

`vsi-sbus-base!` (*long* —) stores the 32 bit data *long* in the SBus Base Address Register (`SBUS_BASE`).

`vsi-id@` (— *id-code*) returns the contents – the 32 bit data *id-code* – of the Identification Register (`ID`).

`vsi-id2@` (— *id-code* `true` | `false`) returns the contents — the 8 bit data *id-code* — of the FGA-5100 Identification Register (`ID2`). When the data has been fetched successfully, the *id-code* $C6_{16}$ along with `true` is returned. `false` is returned in case of an FGA-5000 based CPU board.

`vsi-comp@` (— *byte*) returns the contents – an 8 bit data – of the Compatibility Register of the FGA-5100 (`COMP`).

`vsi-comp!` (*byte* —) stores the 8 Bit data *byte* in the Compatibility Register of the FGA-5100 (`COMP`). In case of FGA-5000 based CPU boards an interrupt is generated (to check which VMEbus interface chip is assembled, see page 142).

`vsi-vme-range@` (*range#* — *long*) returns the contents – a 32 bit data – of the SBus Address Decoding And Translation Register (`VME_RANGE`*x*) identified by its register number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`15`; FGA-5100^{E}: *range#* = `0`…23).

`vsi-vme-range!` (*long range#* —) stores the 32 bit value *long* in the SBus Address Decoding And Translation Register (`VME_RANGE`*x*) identified by its register number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`15`; FGA-5100^{E}: *range#* = `0`…23).

`vsi-amcode@` (*reg# — byte*) returns the contents – an 8 bit data – of the specified FGA-5100 AM Code Register. The number of the AM Code register *reg#* may be `1` (`AMCODE1`) or `2` (`AMCODE2`). If another value is passed to the definition, then it returns the contents of `AMCODE1`. Only effective for FGA-5100^{E} based CPU boards.

`vsi-amcode!` (*byte reg# —*) stores the 8 bit data *byte* in the specified FGA-5100 AM Code Register. The number of the AM Code register *reg#* may be `1` (`AMCODE1`) or `2` (`AMCODE2`). If another value is passed to the definition, then it returns the contents of `AMCODE1`. Only effective for FGA-5100^{E} based CPU boards.

`vsi-vme-master-cap@` (*range# — byte*) returns the contents – an 8 bit data – of the VMEbus Master Capability Register (`VME_MASTER_CAP`*x*) identified by its register number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`15`; FGA-5100^{E}: *range#* = `0`…23).

`vsi-vme-master-cap!` (*byte range# —*) stores the 8 bit data *byte* in the VMEbus Master Capability Register (`VME_MASTER_CAP`*x*) identified by its register number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`15`; FGA-5100^{E}: *range#* = `0`…23).

`vsi-vme-cap@` (*— byte*) returns the contents – an 8 bit data – of the VMEbus Capability Register (`VME_CAP`).

`vsi-vme-cap!` (*byte —*) stores the 8 bit data *byte* in the VMEbus Master Capability Register (`VME_CAP`).

`vsi-vme-handshake@` (*— byte*) returns the contents – an 8 bit data – of the VMEbus Handshake Register (`VME_HANDSHAKE`).

`vsi-vme-handshake!` (*byte —*) stores the 8 bit data *byte* in the VMEbus Master Handshake Register (`VME_HANDSHAKE`).

204223 8 – 0 May 2000

`vsi-sbus-ssel@` (*range# — byte*) returns the contents – an 8 bit data – of the SBus Slave Slot Select Register `SBUS_SSEL`*x_x+1* identified by its register number *x = range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`, 2, 4, …, 14; FGA-5100^{E}: *range#* = `0`, 2, 4, …, 22).

`vsi-sbus-ssel!` (*byte range# —*) stores the 8 bit data *byte* in the SBus Slave Slot Select Register `SBUS_SSEL`*x_x+1* identified by its register number *x = range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`, 2, 4, …, 14; FGA-5100^{E}: *range#* = `0`, 2, 4, …, 22).

Example:

In the following example the same SBus slot is addressed for master ranges four and five. From the output of the second command it can be derived that it is SBus slot nine.

```
ok 55 4 vsi-sbus-ssel!
ok .vme-master-ranges
no   SBus addr   slot …
…
4     …          9
5     …          9
…
ok
```

`vsi-sbus-cap@` (*— byte*) returns the contents – an 8 bit data – of the SBus Capability Register (`SBUS_CAP`).

`vsi-sbus-cap!` (*byte —*) stores the 8 bit data *byte* in the SBus Capability Register (`SBUS_CAP`).

`vsi-sbus-retry-time-ctrl@` (*— byte*) returns the contents – an 8 bit data – of the SBus Retry Time Control Register (`SBUS_RETRY_TIME_CTRL`).

`vsi-sbus-retry-time-ctrl!` (*byte —*) stores the 8 bit data *byte* in the SBus Retry Time Control Register (`SBUS_RETRY_TIME_CTRL`).

`vsi-sbus-rerun-limit-ctrl@` (*— word*) returns the contents – a 16 bit data – of the FGA-5x00's SBus Rerun Limit Control Register (`SBUS_RERUN_-LIMIT_CTRL`).

`vsi-sbus-rerun-limit-ctrl!` (*word —*) store the 16 bit data *word* in the SBus Rerun Limit Control Register (`SBUS_RERUN_LIMIT_CTRL`).

`vsi-dbuf-stat@` (*— byte*) returns the contents – an 8 bit data – of the FGA-5100 Data Buffer Status Register (`DBUF_STAT`). Only effective for FGA-5100^{E} based CPU boards.

`vsi-swpar@` (*— long*) returns the contents – a 32 bit data – of the SBus Write Posting Er-

ror Address Register (SWPAR).

`vsi-swpar-rng@` (— *byte*) returns the contents – an 8 bit data – of the FGA-5100 SBus Write Posting Error Range Register (SWPAR_RNG). Only effective for FGA-5100^{E} based CPU boards.

`vsi-vwpar@` (— *long*) returns the contents – a 32 bit data – of the VMEbus Write Posting Error Address Register (VWPAR).

`vsi-vwpar-rng@` (— *byte*) returns the contents – an 8 bit data – of the FGA-5100 VMEbus Write Posting Error Range Register (VWPAR_RNG). Only effective for FGA-5100^{E} based CPU boards.

`vsi-slerr@` (— *long*) returns the contents – a 32 bit data – of the SBus Late Error Address Register (SLERR).

`vsi-slerr-rng@` (— *byte*) returns the contents – an 8 bit data – of the FGA-5100 SBus Late Error Range Register (SLERR_RNG). Only effective for FGA-5100^{E} based CPU boards.

`vsi-iack-emu@` (*level* — *byte*) returns the contents – an 8 bit data – of the IACK Cycle Emulation Register (VIACK_EMU*x*) associated with the given *level (level* = 1 … 7) of the VMEbus interrupt request.
Only the least significant 3 bits of *level* are considered and when *level* is 0 then the command treats it as if the value 1 has been passed to the command.

`vsi-vme-base@` (— *byte*) returns the contents – an 8 bit data – of the VMEbus Base Address Register (VME_BASE).

`vsi-vme-base!` (*byte* —) stores the 8 bit data *byte* in the VMEbus Base Address Register (VME_BASE).

`vsi-sbus-range@` (*range#* — *long*) returns the contents – a 32 bit data – of the VMEbus Address Decoding and Translation Register (SBUS_RANGE*x*) identified by its range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = 0…2; FGA-5100^{E}: *range#* = 0…4).

`vsi-sbus-range!` (*long range#* —) stores the 32 bit data *long* in the VMEbus Address Decoding and Translation Register (SBUS_RANGE*x*) identified by its range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = 0…2; FGA-5100^{E}: *range#* = 0…4).

`vsi-vme-ext@` (*range#* — *byte*) returns the contents – an 8 bit data – of the VMEbus Address Extension Register (VME_EXT*x*) identified by its range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = 0…2; FGA-5100^{E}: *range#* = 0…4).

204223 8 – 0 May 2000

`vsi-vme-ext!` (*byte range#* —) stores the 8 bit data *byte* in the VMEbus Address Extension Register (`VME_EXT`*x*) identified by its range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`vsi-reset-stat@` (— *byte*) returns the contents – an 8 bit data – of the Reset Source Register (`RESET_STAT`).

`vsi-intr-stat@` (— *long*) returns the contents – a 32 bit data – of the Interrupt Status Register (`INTR_STAT`).

`vsi-intr-stat!` (*long* —) stores the 32 bit data *long* in the Interrupt Status Register (`INTR_STAT`).

`.vsi-intr-stat` (—) displays the actual contents of the Interrupt Status Register (`INTR_STAT`). The contents of the register is displayed as shown below:

```
ok .vsi-intr-stat
VME-IRQ1: 0 VME-IRQ2: 0 VME-IRQ3: 0 VME-IRQ4: 0 VME-IRQ5: 0
VME-IRQ6: 0 VME-IRQ7: 0 VME-IACK: 0 FMB1    : 0 FMB0    : 0
IBOX    : 0 LERR    : 0 WDOG    : 0 DMATERM : 0 VWPERR  : 0
SWPERR  : 0 MAILBOX : 0 ARBTOUT : 0 ABORT   : 0 SYSFAIL+: 0
SYSFAIL-: 0 ACFAIL  : 0
ok
```

When an interrupt is pending the command displays the one (1); otherwise it displays the zero (0) to indicate that the interrupt is not pending.

Note: **The state of the entry `SYSFAIL-` reports the occurrence of a negative edge of the VMEbus `SYSFAIL*` signal which indicates that the `SYSFAIL*` signal has been asserted. The state of the entry `SYSFAIL+` reports the occurrence of a positive edge of the VMEbus `SYSFAIL*` signal which indicates that the `SYSFAIL*` signal has been negated.**

`vsi-irq-map@` (*irq# — byte*) returns the contents – an 8 bit data – of the VMEbus Interrupt Level Select and Enable Register specified by *irq#.*

Example:

```
ok slerr vsi-irq-map@
ok .
ff
ok
```

Table 62 *irq#* Values for VME Interrupt Level Select and Enable Registers

irq#	Interrupt Level Select and Enable Register
abort-key	VMEbus Abort key ~ (ABORT_IRQ_MAP)
acfail	VMEbus ACFAIL ~ (ACFAIL_IRQ_MAP)
arbiter	VMEbus Arbiter Timeout ~ (ARB_IRQ_MAP)
dmac	VMEbus DMA ~ (DMA_IRQ_MAP)
fmb0 fmb1	VMEbus Message Broadcast ~ (FMB_IRQ_MAP*x*)
ibox	VMEbus Interrupt Box ~ (IBOX_IRQ_MAP)
mbox0 mbox1 … mbox15	VMEbus Mailbox ~ (MBOX_IRQ_MAP*x*)
viack	VMEbus IACK ~ (VIACK_IRQ_MAP)
virq1 virq2 … virq7	VMEbus Interrupt ~ (VIRQ_MAP*x*)
wdt	VMEbus Watchdog Timer ~ (WDT_IRQ_MAP)
wpe	VMEbus Write Posting Error ~ (WPE_IRQ_MAP)
slerr	Late Error Interrupt ~ (SLEER_IRQ_MAP)
sysfail+	VMEbus SYSFAIL Negate ~ (SYSFAIL_IRQ_MAP1)
sysfail-	VMEbus SYSFAIL Assert ~ (SYSFAIL_IRQ_MAP0)

204223 8 – 0 May 2000

`vsi-irq-map!` (*byte irq#* —) stores the 8 bit data *byte* in the VMEbus Interrupt Level Select and Enable Register specified by *irq#*. For valid *irq#* values, see table 62 “irq# Values for VME Interrupt Level Select and Enable Registers” on page 148.

Example:

```
ok ff slerr vsi-irq-map!
ok
```

`vsi-arb-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Arbiter Control Register (`ARB_CTRL`).

`vsi-arb-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Arbiter Control Register (`ARB_CTRL`).

`vsi-req-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Requester Control Register (`REQ_CTRL`).

`vsi-req-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Requester Control Register (`REQ_CTRL`).

`vsi-bus-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Bus Capture Control Register (`BUS_CTRL`).

`vsi-bus-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Bus Capture Control Register (`BUS_CTRL`).

`vsi-irq-vector@` (*irq-level#* — *byte*) returns the contents – an 8 bit data – of the VMEbus Interrupter Vector Register (`VIRQ_VECTOR`*x*) associated with the given interrupt request level number *irq-level#* (*irq-level#* = `0` … `6`). Each value specifies one of the seven VMEbus Interrupter Vector Registers. The returned contents are valid only in case of FGA-5100^{C} or FGA-5100^{E} based CPU boards.

`vsi-irq-vector!` (*byte irq-level#* — `true` | `false`) stores the 8 bit data *byte* in the VMEbus Interrupter Vector Register (`VIRQ_VECTOR`*x*) associated with the given interrupt request level number *irq-level#* (*irq-level#* = `0` … `6`). Each value specifies one of the seven VMEbus Interrupter Vector Registers.

`true` is returned when the given byte has been stored in the register successfully. When the VMEbus interrupt request is still pending, `false` is returned. This indicates that the data cannot be stored in the register unless the VMEbus interrupt request is negated. Only effective for FGA-5100^{C} and FGA-5100^{E} based CPU boards.

`vsi-irq-stat@` (— *byte*) returns the contents – an 8 bit data – of the VMEbus Interrupt Status Register (`INTR_STAT`). The returned contents are valid only in case of FGA-5100^{C} or FGA-5100^{E} based CPU boards.

`vsi-mbox@` (*mailbox# — byte*) returns the contents – an 8 bit data – of the Mailbox Register (`MBOX`*x*) identified by its mailbox number *mailbox#* (*mailbox#* = `0` … 15).

`vsi-mbox!` (*byte mailbox# —*) stores the 8 bit data *byte* in the Mailbox Register (`MBOX`*x*) identified by its mailbox number *mailbox#* (*mailbox#* = `0` … 15).

`vsi-mbox-stat@` (*— word*) returns the contents – a 16 bit data – of the Mailbox Status Register (`MBOX_STAT`).

`vsi-mbox-stat!` (*word —*) stores the 16 bit data long in the Mailbox Status Register (`MBOX_STAT`).

`vsi-sem@` (*semaphore# — byte*) returns the contents – an 8 bit data – of the Semaphore Register (`SEM`*x*) identified by its semaphore number *semaphore# (semaphore#* = `0` … `47`).

`vsi-sem!` (*byte semaphore# —*) stores the 8 bit data *byte* in the Semaphore Register (`SEM`*x*) identified by its semaphore number *semaphore# (semaphore#* = `0` … `47`).

`vsi-fmb-ctrl@` (*— byte*) returns the contents – an 8 bit data – of the Message Broadcast Control Register (`FMB_CTRL`).

`vsi-fmb-ctrl!` (*byte —*) stores the 8 bit data *byte* in the Message Broadcast Control Register (`FMB_CTRL`).

`vsi-fmb-addr@` (*— byte*) returns the contents – an 8 bit data – of the Message Broadcast Address Register (`FMB_ADDR`).

`vsi-fmb-addr!` (*byte —*) stores the 8 bit data *byte* in the Message Broadcast Address Register (`FMB_ADDR`).

`vsi-fmb-stat@` (*channel# — byte*) returns the contents – an 8 bit data – of the Message Broadcast Status Register (`FMB_STAT`*x*) identified by its channel number *channel#* (*channel#* = `0`, `1`). Each value specifies one of the two Message Broadcast Status Registers.

`vsi-fmb-stat!` (*byte channel# —*) stores the 8 bit data *byte* in the Message Broadcast Status Register (`FMB_STAT`*x*) identified by its channel number *channel#* (*channel#* = `0`, `1`). Each value specifies one of the two Message Broadcast Status Registers.

`vsi-fmb-msg@` (*channel# — long* `true` | `false`) returns `true` and the contents – a 32 bit data – of the Channel Message Register (`FMB_MSG`*x*) identified by its channel number *channel#* (*channel#* = `0`, `1`). It returns `false`, if an error has occurred.

204223 8 – 0 May 2000

`vsi-gcsr@` (— *byte*) returns the contents – an 8 bit data – of the Global Control and Status Register (`GCSR`).

`vsi-gcsr!` (*byte* —) stores the 8 bit data *byte* in the Global Control and Status Register (`GCSR`).

`vsi-mcsr0@` (— *byte*) returns the contents – an 8 bit data – of the Miscellaneous Control and Status Register 0 (`MCSR0`).

`vsi-mcsr0!` (*byte* —) stores the 8 bit data *byte* in the Miscellaneous Control and Status Register 0 (`MCSR0`).

`vsi-wdt-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Watchdog Timer Control Register (`MCSR1`).

`vsi-wdt-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Watchdog Timer Control Register (`MCSR1`).

`vsi-wdt-restart@` (— *byte*) returns the contents – an 8 bit data – of the Watchdog Restart Register (`WDT_RESTART`).

`vsi-wdt-restart!` (*byte* —) stores the 8 bit data *byte* in the Watchdog Restart Register (`WDT_RESTART`).

`vsi-dma-ctrl@` (— *word*) returns the contents – a 16 bit data – of the DMA Control Register (`DMA_CTRL`).

`vsi-dma-ctrl!` (*word* —) stores the 16 bit data *word* in the DMA Control Register (`DMA_CTRL`).

`vsi-dma-mode@` (— *byte*) returns the contents – an 8 bit data – of the DMA Mode Register (`DMA_MODE`).

`vsi-dma-mode!` (*byte* —) stores the 8 bit data *byte* in the DMA Mode Register (`DMA_MODE`).

`vsi-dma-stat@` (— *byte*) returns the contents – an 8 bit data – of the DMA Status Register (`DMA_STAT`).

`vsi-dma-stat!` (*byte* —) stores the 8 bit data *byte* in the DMA Status Register (`DMA_STAT`).

`vsi-dma-src@` (— *long*) returns the contents – a 32 bit data – of the DMA Source Address Register (`DMA_SRC`).

`vsi-dma-src!` (*long* —) stores the 32 bit data *long* in the DMA Source Address Register (`DMA_SRC`).

`vsi-dma-dest@` (— *long*) returns the contents – a 32 bit data – of the DMA Destination Address Register (`DMA_DST`).

`vsi-dma-dest!` (*long* —) stores the 32 bit data *long* in the DMA Destination Address Register (`DMA_DST`).

`vsi-dma-cap@` (— *long*) returns the contents – a 32 bit data – of the DMA Capability and Transfer Count Register (`DMA_CAPTL`).

`vsi-dma-cap!` (*long* —) stores the 32 bit data *long* in the DMA Capability and Transfer Count Register (`DMA_CAPTL`).

`vsi-ibox-ctrl@` (— *word*) returns the contents – a 16 bit data – of the Interrupt Box Control Register (`IBOX_CTRL`).

`vsi-ibox-ctrl!` (*word* —) stores the 16 bit data *word* in the Interrupt Box Control Register (`IBOX_CTRL`).

`vsi-ibox-addr@` (— *word*) returns the contents – a 16 bit data – of the Interrupt Box Address Register (`IBOX_ADDR`).

`vsi-ibox-addr!` (*word* —) stores the 16 bit data *word* in the Interrupt Box Address Register (`IBOX_ADDR`).

`vsi-ibox-count@` (— *byte*) returns the contents – an 8 bit data – of the FGA-5100 IBOX Count Register (`IBOX_COUNT`). The returned contents are valid only in case of FGA-5100^{E} based CPU boards.

5.2.2 VMEbus Interrupt Mapper

`vsi-irq!` (*flag irq#* —) enables or disables interrupt generation due to the event specified by *irq#*. Enabling or disabling depends on the setting of *flag*.

- *flag* = `true =` enables interrupt generation.
- *flag* = `false =` disables interrupt generation.

Example:

```
ok true viack vsi-irq!
ok
```

Table 63 *irq#* Values for Interrupt Generation

irq#		Event
`abort-key`	27_{10}	… when the abort key is being pressed
`acfail`	26_{10}	… upon assertion of the VMEbus ACFAIL* signal

204223 8 – 0 May 2000

Table 63 *irq#* Values for Interrupt Generation (cont.)

irq#		**Event**
`arbiter`	30_{10}	… by the arbiter when the arbitration timeout expired
`dmac`	28_{10}	… when the DMA process terminated (successfully or due to an error)
`fmb0` `fmb1`	34_{10} 35_{10}	… when either an FMB message has been accepted or rejected
`ibox`	33_{10}	… when the interrupt box is accessed

`mbox0`
`mbox1`
…
`mbox15`
	23_{10}
	22_{10}
	…
	8_{10}
		… when the mailbox specified by *irq#* = `mbox0`, …, `mbox15` is taken

`viack`	0_{10}	… upon receipt of a VMEbus interrupt acknowledge

`virq1`
`virq2`
…
`virq7`
	7_{10}
	6_{10}
	…
	1_{10}
		… upon receipt of a VMEbus interrupt at *irq#* = `virq1`, …, `virq7`

`wdt`	31_{10}	… when half of the watchdog time has expired
`wpe`	29_{10}	… when a write posting error occurs on the SBus or on the VMEbus
`slerr`	32_{10}	… when a late error occurs on the SBus
`sysfail+`	25_{10}	… upon the negation of the VMEbus SYSFAIL* signal
`sysfail-`	24_{10}	… upon the assertion of the VMEbus SYSFAIL* signal

`vsi-irq-mapping!` (*mapping irq# —*) selects the interrupt *mapping* to be generated due to the event specified by *irq#*.

- The parameter *mapping* specifies the interrupt to be asserted by the FGA-5x00 when a certain VMEbus interrupt request is asserted. For valid *mapping* values, see table 64 “Interrupt Mapping” on page 156.
- For valid *irq#* values, see table 63 “irq# Values for Interrupt Generation” on page 152.

Example:

```
ok vsi-sbus-irq-3 virq3 vsi-irq-mapping!
ok
```

`vsi-irq-mapping@` (*irq# — mapping*) returns the interrupt *mapping* asserted by the FGA-5x00 when the event specified by *irq#* has occurred.

- For valid *irq#* values, see table 63 “irq# Values for Interrupt Generation” on page 152.
- The returned *mapping* indicates the interrupt asserted by the FGA-5x00 (see table 64 “Interrupt Mapping” on page 156).

Example:

```
ok virq3 vsi-irq-mapping@
ok .
0
ok
```

204223 8 – 0 May 2000

5.2.3 VMEbus Interrupt Handler and Interrupter

`vme-intr-pending?` (*level* — `true` | `false`) checks whether an interrupt is pending on a given interrupt request level (*level* = `1` … `7`) and returns a flag. When an interrupt is pending the flag is `true`; otherwise it is `false`. Only the least significant 3 bits of *level* are considered and when *level* is `0` then the command treats it as if the value `1` has been passed to the command.

The command verifies the state of the interrupt pending bit in the 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 (*level* = `1` … `7`) and returns the obtained 8-bit *vector*.

Typically, the *vector* returned is within the range `0` … `255`, but when no interrupt is pending, and therefore no interrupt has to be acknowledged, the value `-1` is returned.

Only the least significant 3 bits of *level* are considered and when *level* is `0` then the command treats it as if the value `1` has been passed to the command.

`vme-intr-ack?` (*level* — `true` | `false`) checks whether the VMEbus interrupt specified by *level* has been acknowledged (*level* = `1` … `7`). When an interrupt has been acknowledged the command returns `true`; otherwise `false`. `vme-intr-ack?` is only effective for FGA-5100 based CPU boards (FGA-5100^{C} & FGA-5100^{E}).

Only the least significant 3 bits of *level* are considered and when *level* is `0` then the command treats it as if the value `1` 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 the VMEbus interrupt request has been acknowledged and the command returns `true`. Otherwise – the status bit is cleared – the value `false` is returned.

`vme-intr!` (*vector level* —) generates a VMEbus interrupt at a given interrupt request level (*level* = `1` … `7`) accompanied by a specified 8-bit vector *vector*. `vme-intr!` is only effective for FGA-5100 based CPU boards (FGA-5100^{C} & FGA-5100^{E}).

Only the least significant 3 bits of *level* are considered and when *level* is `0` then the command treats it as if the value `1` has been passed to the com-

mand. The command accesses the VMEbus Interrupter Vector Register with the `cpoke` command, because the FGA-5100 will refuse a write access to the register with a bus error when the VMEbus interrupt – specified by *level* – is still pending, which means that the previous interrupt request has not been acknowledged.

`install-vme-intr-handler` (*mapping level* —) installs the interrupt service routine dealing with the given VMEbus interrupt level (*level* = `1` … `7`). The parameter *mapping* defines the interrupt asserted by the FGA-5x00 when the certain VMEbus interrupt request level is asserted. The value of *mapping* may be one of the values in the range `0` … `7`. Each value specifies one of the eight FGA-5x00 interrupt request lines. All allowed mappings are listed in the table below. The address of the interrupt service routine currently in effect is preserved.

Only the least significant 3 bits of *level* are considered and when *level* is `0` then the command treats it as if the value `1` has been passed to the command.

Note: **The words listed in the second column of the table below may be used to specify a valid interrupt *mapping*.**

Table 64 Interrupt Mapping

mapping	Constant	Interrupt Generated by FGA-5x00
`0`	`vsi-nmi`	INT (connected with *non-maskable* interrupt)
`1`	`vsi-sbus-irq-1`	SINT1 (connected with SBus IRQ1)
`2`	`vsi-sbus-irq-2`	SINT2 (connected with SBus IRQ2)
`3`	`vsi-sbus-irq-3`	SINT3 (connected with SBus IRQ3)
`4`	`vsi-sbus-irq-4`	SINT4 (connected with SBus IRQ4)
`5`	`vsi-sbus-irq-5`	SINT5 (connected with SBus IRQ5)
`6`	`vsi-sbus-irq-6`	SINT6 (connected with SBus IRQ6)
`7`	`vsi-sbus-irq-7`	SINT7 (connected with SBus IRQ7)

`uninstall-vme-intr-handler` (*level* —) removes the interrupt service routine dealing with the given VMEbus interrupt *level* (*level* = `1` … `7`) and installs the old interrupt service routine.

Only the least significant 3 bits of *level* are considered and when *level* is `0` then the command treats it as if the value `1` has been passed to the command.

204223 8 – 0 May 2000

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

OpenBoot maintains seven variables which are modified by the VMEbus interrupt handlers. They are called `vme-intr{1|2|3|4|5|6|7}-vector`. In general, the interrupt handlers store the vector obtained during an interrupt acknowledge cycle in the appropriate variable.

5.2.4 VMEbus Arbiter

The commands listed below are available to control the arbiter:

`vme-arb-mode@` (— *mode*) returns the *mode* the arbiter is currently operating in (*mode* = `0` … `3`). Each value specifies a particular mode, the values `0` and `3` being mapped to the same operating mode. The values correspond to the three constants which are available to specify one of the three bus arbitration modes:

- `pri` – prioritized – equivalent to *mode* = 0_{10} or 3_{10},
- `rrs` – round robin select – equivalent to *mode* = 1_{10},
- `prr` – prioritized round robin – equivalent to *mode* = 2_{10}.

`vme-arb-mode!` (*mode* —) selects the arbiter mode specified by *mode* (*mode* = `0` … `3`). Each value specifies a particular mode: `0` and `3` indicate that the arbiter operates in the *priority* mode; `1` specifies the *round-robin* mode; and `2` specifies the *prioritized-round-robin* mode.

`set-arb-mode` (*addr length* —) sets the arbiter mode according to the contents of the string – the string representation of the arbiter mode. The string is specified by its address *addr* and its length *length*. The string may contain the following:

String Representation	Mode
" pri"	Prioritized
" rrs"	Round Robin Select
" prr"	Prioritized Round Robin Select

`.vsi-arb-ctrl` (—) displays the current contents of the VMEbus Arbiter Control Register.

5.2.5 VMEbus Requester

The commands listed below are available to control the requester and to obtain some information about the requesters's operational state:

`set-rel-mode` (*addr length* —) sets the release mode according to the contents of the string representation of the release mode. The string is specified by its address *addr* and its length *length*. The string may contain the following:

String Representation	Mode
" ror"	Release On Request
" roc"	Release On Bus Clear
" rat"	Release After Timeout
" rwd"	Release When Done

Any combination of the three release modes ROR, ROC, and RAT may be specified. In this case the release modes in the string must be separated by a comma, as described below:

```
ok "roc,rat" set-rel-mode
ok
```

`.vsi-req-ctrl` (—) displays the current contents of the VMEbus Requester Control Register.

`vme-bus-capture!` (`true` | `false` —) enables or disables the *bus-capture-and-hold* capability of the FGA-5x00. 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 the 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 indicated 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
```

204223 8 – 0 May 2000

5.2.6 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`) 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-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`.

5.2.7 VMEbus Master Interface

***Note: range#* values depending on VMEbus interface chip:**

The FGA-5000 provides 16 sets of registers to control any VMEbus master operation whereas the FGA-5100 provides 24. Each set may be used to address a certain address range within the VMEbus address space. A register set is identified by the unique range number *range#.*

- For FGA-5000 or FGA-5100^{C} based CPU boards *range#* may be one of the values `0`…`15`.
- For FGA-5100^{E} based CPU boards *range#* may be one of the values `0`…`23`.

Naming Convention

When the VMEbus is being accessed the part of the FGA-5x00 connected with the SBus is considered as the SBus slave device, whereas the part of the FGA-5x00 that is connected with the VMEbus is operating as VMEbus master. This fact is reflected in the names of the commands available to control the VMEbus master interface.

This section is subdivided into

- "General Initialisation and Control" (see below) and related examples (see page 163)
- "Control of Operating Modes of the SBus Interface" (see page 166)

General Initialisation and Control

The commands listed and described in the following are available to initialise and control the VMEbus master interface:

`#vme-ranges` (— *#vme-ranges*) returns the number *#vme-ranges* of available register sets which are used to control accesses to the VMEbus.

`vme-master-ena` (*range#* —) enables the address decoding associated with the range number *range#* to access the VMEbus (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

`vme-master-dis` (*range#* —) disables the address decoding associated with the range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

`vme-master-wp-ena` (*range#* —) enables write posting within the VMEbus address range associated with the range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

`vme-master-wp-dis` (*range#* —) disables write posting within the VMEbus address range associated with the range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

`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 the privileged mode. Otherwise – the value `false` is passed to the command – the VMEbus is accessed in the non-privileged mode. The mode selected with this command applies to all ranges used to access the VMEbus.

`vme-non-priv!` ([`true` | `false`] *range#* —) enables or disables the non-privileged mode within the VMEbus address range associated with the range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the available register sets used for accessing the VMEbus. When `true` is passed to the definition, the non-privileged mode is enabled and all subsequent accesses to the VMEbus through the particular VMEbus master range register set are carried out in the non-privileged mode. When `false` is passed to the the definition, then the non-privileged mode is disabled, and all subsequent accesses to the VMEbus through the particular VMEbus master range register set are carried out in the supervisory mode.

204223 8 – 0 May 2000

`vme-non-priv?` (*range# — flag*) indicates whether the non-privileged mode is enabled for the VMEbus master range specified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). When `true` is returned, the non-privileged mode is enabled. When `false` is returned, the supervisory mode is enabled.

`sbus-slave-range@` (*range# — offset sbus-slot# size*) returns the SBus slave parameters associated with the range identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

The parameters returned by the command specify the SBus address range to be accessed to reach the VMEbus. The address range is represented by the triple *offset*, *sbus-slot#*, and *size*.

`sbus-slave-range!` (*offset sbus-slot# size range# —*) sets the SBus slave parameters associated with the range identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

The parameters passed to the command specify the SBus address range to be accessed to reach the VMEbus. The address range is represented by the triple *offset*, *sbus-slot#*, and *size*.

`vme-master-range@` (*range# — addr data-capability address-capability size*) returns the VMEbus master capabilities associated with the range number identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

The VMEbus address range being accessed is represented by the *addr-size* pair, where *addr* specifies the physical VMEbus address and *size* identifies the address range covered. The value of *data-capability* and *address-capability* may be one of the values listed in the table below.

`vme-master-range!` (*addr data-capability address-capability size range# —*) sets the VMEbus master capabilities associated with the range number identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0` … `15`; FGA-5100^{E}: *range#* = `0` … `23`). Each value specifies one of the register sets controlling any VMEbus master operation.

The VMEbus address range being accessed is represented by the *addr-size* pair, where *addr* specifies the physical VMEbus address and *size* identifies the address range covered. *data-capability* and *address-capability* may be one of the values listed in the following table.

Value	Data-Capability	Address-Capability
000_2	Cap-d8	Cap-a16
001_2	Cap-d16	Cap-a24
010_2	Cap-d32	Cap-a32
011_2	Cap-blt	Reserved
100_2	Cap-mblt	Cap-a64
101_2	In case of FGA-5100^E: `cap-2eVME` otheRwise reserved	Reserved
110_2	Reserved	Reserved
111_2	Reserved	In case of FGA-5100^E: `cap-cr/csr` Otherwise reserved

`.vme-master-ranges` (—) displays the current settings of all register sets of the VMEbus master interface.

`.vme-cap` (—) displays the contents of the VMEbus Capability register.

`vme-master-map` (*range# — vaddr*) makes the physical address range, as defined by the contents of the range register set specified by the range number *range#*, available to the processor's virtual address space and returns the virtual address *vaddr*.

Since the command obtains all information from the specific range register set, the particular range register *must* be initialised before.

`vme-master-unmap` (*vaddr range# —*) removes the physical address range, as defined by the contents of the range register set specified by the range number *range#* and the virtual address *vaddr*, from the processor's virtual address space.

204223 8 – 0 May 2000

Examples:

The following examples describe how to initialise the VMEbus interface for subsequent VMEbus master accesses. All examples are continued at the end of this section (see page 165).

Example 1

The example below shows how to access a 2 MByte area within the *extended* address space (A32) of the VMEbus beginning at address 4080.0000_{16}. The register set associated with the range number zero (*range#* is `0`) is used to access the VMEbus area mentioned above.

The first command initialises the VMEbus master interface. It sets the data- and address capabilities, as well as the VMEbus address and the size of the area being accessed. The *data capability* is defined using the predefined constant `cap-d32` which enables the VMEbus master interface to access Byte (8-bit data), half-words (16-bit data), and words (32-bit data) within the VMEbus area. The *address capability* is defined using the predefined constant `cap-a32` that enables the VMEbus interface to access the *extended* address space (A32) of the VMEbus.

The SBus slave interface is initialised by the second command which specifies that the VMEbus is accessed when the SBus slot 4 is being accessed at offsets $A0.0000_{16}$ to $BF.FFFF_{16}$ which corresponds to the VMEbus addresses in the range 4080.0000_{16} to $409F.FFFF_{16}$ of the *extended* address space (A32).

```
ok h# 4080.0000 cap-d32 cap-a32 1Meg 2 * 0 vme-master-range!
ok 1Meg d# 10 * 4 1Meg 2 * 0 sbus-slave-range!
ok 0 vme-master-ena
ok 0 vme-master-map value vmebus
ok
```

Finally, the third command enables any access to the VMEbus. The fourth command maps the physical address area to be accessed in order to address the VMEbus to the virtual address space of the processor and stores the virtual address in the variable vmebus. This variable may be used to access the VMEbus area using the commands to read and write data provided by OpenBoot.

```
ok vmebus 0 vme-master-unmap
ok
```

When the translation (SBus to VMEbus) defined by the contents of the register set associated with the range number zero is no longer used, then the memory mapped to the processor's virtual address space to access the VMEbus must be released *before* the contents of this register set are modified. This has to be done with the command `vme-master-unmap` as stated above.

Example 2

In the next example the VMEbus interface is initialised to allow accesses to the *standard* address space (A24) of the VMEbus beginning at address 98.0000_{16} . The size of this area is 512 KByte and the register set associated with the range number one (*range#* is `1`) is used to access this VMEbus area.

The first command initialises the VMEbus master interface. It sets the data- and address capabilities, as well as the VMEbus address and the size of the area being accessed. The *data capability* is defined using the predefined constant `cap-d16` which enables the VMEbus master interface to access Byte (8-bit data), and half-words (16-bit data) within the VMEbus area. The *address capability* is defined using the predefined constant `cap-a24` that enables the VMEbus interface to access the *standard* address space (A24) of the VMEbus.

The SBus slave interface is initialised by the second command which specifies that the VMEbus is accessed when the SBus slot 5 is being accessed at offsets 120.0000_{16} to $127.FFFF_{16}$ which corresponds to the VMEbus addresses in the range 98.0000_{16} to $9F.FFFF_{16}$ of the *standard* address space (A24).

```
ok h# 98.0000 cap-d16 cap-a24 1Meg 2 / 1 vme-master-range!
ok 1Meg d# 18 * 5 1Meg 2 / 1 sbus-slave-range!
ok 1 vme-master-ena
ok 1 vme-master-map value vmebus
ok
```

Finally, the third command enables any access to the VMEbus. The fourth command maps the physical address area to be accessed in order to address the VMEbus to the virtual address space of the processor and stores the virtual address in the variable vmebus. This variable may be used to access the VMEbus area using the commands to read and write data provided by OpenBoot.

```
ok 1 vme-master-map value vmebus
ok
```

When the translation (SBus to VMEbus) defined by the contents of the register set associated with the range number zero is no longer used, then the memory mapped to the processor's virtual address space to access the VMEbus must be released *before* the contents of this register set are modified. This has to be done with the command `vme-master-unmap` as stated above.

204223 8 – 0 May 2000

Example 3

The last example describes how to initialise the VMEbus interface to allow accesses to the *short* address space (A16) of the VMEbus beginning at address 0000_{16}. The size of this area is 64 KByte and therefore covers the entire *short* address space. The register set associated with the range number two (*range#* is `2`) is used to access this VMEbus area.
Again, the first command initialises the VMEbus master interface. It sets the data- and address capabilities, as well as the VMEbus address and the size of the area being accessed. The *data capability* is defined using the predefined constant `cap-d8` which limits the VMEbus master interface to access only Byte (8-bit data) within the VMEbus area. The *address capability* is defined using the predefined constant `cap-a16` that enables the VMEbus interface to access the *standard* address space (A16) of the VMEbus.

The SBus slave interface is initialised by the second command which specifies that the VMEbus is accessed when the SBus slot 6 is being accessed at offsets 400.0000_{16} to $400.FFFF_{16}$ which corresponds to the VMEbus addresses in the range 0000_{16} to $FFFF_{16}$ of the *short* address space (A16).

```
ok h# 0000 cap-d8 cap-a16 h# 1.0000 2 vme-master-range!
ok 1Meg d# 64 * 6 h# 1.0000 2 sbus-slave-range!
ok 2 vme-master-ena
ok 2 vme-master-map value vmebus
ok
```

Finally, the third command enables any access to the VMEbus. The fourth command maps the physical address area to be accessed in order to address the VMEbus to the virtual address space of the processor and stores the virtual address in the variable vmebus. This variable may be used to access the VMEbus area using the commands to read and write data provided by OpenBoot.

```
ok 2 vme-master-map value vmebus
ok
```

When the translation (SBus to VMEbus) defined by the contents of the register set associated with the range number zero is no longer used, then the memory mapped to the processor's virtual address space to access the VMEbus must be released *before* the contents of this register set are modified. This has to be done with the command `vme-master-unmap` as stated above.

Example 1, 2, and 3 Continued

Assumed the first three register sets have been used to access the VMEbus address spaces as described in the examples above, then the following command may be used to display the settings of the registers sets:

```
ok .vme-master-ranges
```

Control of Operating Modes of the SBus Interface

The following commands are available to control the various operating modes of the FGA-5x00 SBus interface.

`sbus-rerun!` (`true` | `false` —) enables or disables the FGA-5x00's capability to generate reruns on the SBus. When the value `true` is passed to the command the FGA-5x00 will initiate SBus rerun if necessary. Otherwise – the value `false` is passed to the command – the FGA-5x00's capability to initiate SBus reruns is disabled.

`sbus-rerun-limit@` (— *#rerun-limit*) returns the number of reruns *#rerun-limit* before the FGA-5x00 terminates an SBus cycle with an error (*#rerun-limit* = `0` … `255`).

`sbus-rerun-limit!` (*#rerun-limit* —) sets the number of reruns *#rerun-limit* before the FGA-5x00 terminates an SBus cycle with an error (*#rerun-limit* = `0` … `255`). The command treats *#rerun-limit* as a modulo 256 number. When the command is called it verifies whether the given number of reruns falls below the limit specified by `min-rerun-limit`. If this value falls below the given limit, then the command uses the value of `min-rerun-limit` instead. This ensures that the SBus interface is operating properly.

`.sbus-cap` (—) displays the current contents of the SBus Master Capability Register as shown below:

```
ok .sbus-cap
Split: 1   Split Flow: 1   Arbiter: 1   Burst: 1
Master Read Stop Point: 32 bytes Max. Burst Length: 32 bytes
ok
```

`.sbus-retry-time-ctrl` (—) displays the current contents of the SBus Retry Time Control Register as stated below:

```
ok .sbus-retry-time-ctrl
Retry time: 10
ok
```

`.sbus-rerun-limit-ctrl` (—) displays the current contents of the SBus Rerun Limit Control Register as depicted below:

```
ok .sbus-rerun-limit-ctrl
Enable Reruns: 0   Rerun limit: 255
ok
```

5.2.8 VMEbus Slave Interface

Note: ***range#*** **values depending on VMEbus interface chip:**

204223 8 – 0 May 2000

The FGA-5000 and the FGA-5100^{C} both provide three sets of registers to control any VMEbus slave access whereas the FGA-5100^{E} provides five. Each set may be used to make a certain slave address range – standard (A24) or extended (A32) slave address range – available to the VMEbus' address space. A register set is identified by the unique range number *range#.*

- For FGA-5000 or FGA-5100^{C} based CPU boards *range#* may be one of the values `0…2`.
- For FGA-5100^{E} based CPU boards *range#* may be one of the values `0…4`.

Only the A24 and A32 slave mode allows a VMEbus master to access the memory of the SPARC/CPU-20VT. In the A16 slave mode all VMEbus master accesses are limited to the FGA-5x00 registers which are accessible from the VMEbus.

When the VMEbus interface is being accessed from the VMEbus, then the part of the FGA-5x00 connected with the VMEbus is considered as VMEbus slave device. Whereas the part of the FGA-5x00 that is connected with the SBus is operating as the SBus master. This fact is reflected in the names of the commands available to control the VMEbus master interface.

The commands listed and described in the following are available to initialise and control the A16 VMEbus slave interface (for an example see page 169):

`vme-a16-slave-ena` (—) enables the capability to access the FGA-5x00 registers from the VMEbus in the *short* address space (A16).

`vme-a16-slave-dis` (—) disables the capability to access the FGA-5x00 registers from the VMEbus in the *short* address space (A16).

`vme-a16-slave-addr@` (— *addr*) returns the 16 bit address *addr* at which the FGA-5x00 registers are accessible within the *short* address space (A16).

`vme-a16-slave-addr!` (*addr* —) defines the 16-bit address *addr* at which the FGA-5x00 registers are accessible within the *short* address space (A16). The least significant nine bits of the address *addr* are ignored by the command – the command treats them as if they are cleared –, because the FGA-5x00 is only accessible from the VMEbus beginning at 512-Byte boundaries.

The commands listed and described in the following are available to initialise and control the A24 and A32 VMEbus slave interface:

`vme-slave-ena` (*range#* —) enables the address decoding associated with the range number *range#* to allow accesses from the VMEbus (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`vme-slave-dis` (*range#* —) disables the address decoding associated with the range number *range#* to forbid accesses from the VMEbus (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`vme-slave-wp-ena` (*range#* —) enables *write posting* within the VMEbus slave address range associated with the range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`vme-slave-wp-dis` (*range#* —) disables *write posting* within the VMEbus slave address range associated with the range number *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`sbus-a24-master-range@` (*range#* — *vaddr size*) returns the SBus master parameters associated with the A24 slave interface identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

The parameters *vaddr* and *size* identify the virtual address range within the SBus, into which all A24 slave accesses are translated.

`sbus-a24-master-range!` (*vaddr size range#* —) defines the SBus master parameters associated with the A24 slave interface identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

The parameters *vaddr* and *size* identify the virtual address range within the SBus, into which all A24 slave accesses are translated.

`vme-a24-slave-range@` (*range#* — *paddr size*) returns the VMEbus base address *paddr* and the size *size* of the A24 slave window associated with the range identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`vme-a24-slave-range!` (*paddr size range#* —) sets the VMEbus base address *paddr* and the size *size* of the A24 slave window associated with the range identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`sbus-a32-master-range@` (*range#* — *vaddr size*) returns the SBus master parameters associated with the A32 slave interface identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

The parameters *vaddr* and *size* identify the virtual address range within the SBus, into which all A32 slave accesses are translated.

`sbus-a32-master-range!` (*vaddr size range#* —) defines the SBus master parameters

204223 8 – 0 May 2000

associated with the A32 slave interface identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

The parameters *vaddr* and *size* identify the virtual address range within the SBus, into which all A32 slave accesses are translated.

`vme-a32-slave-range@` (*range# — paddr size*) returns the VMEbus base address *paddr* and the size *size* of the A32 slave window associated with the range identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`vme-a32-slave-range!` (*paddr size range# —*) sets the VMEbus base address *paddr* and the size *size* of the A32 slave window associated with the range identified by *range#* (FGA-5000 & FGA-5100^{C}: *range#* = `0`…`2`; FGA-5100^{E}: *range#* = `0`…4).

`.vme-slave-ranges` (—) displays the current settings of *all* register sets controlling any VMEbus slave access.

Examples:

Example 1

The following example lists all steps to be taken, to initialise the VMEbus interface for A32 accesses from the VMEbus beginning at address `2340.0000`$_{16}$ and ranging to `235F.FFFF`$_{16}$. The register set associated with the *range number* zero (`0`) is used to control this particular VMEbus slave interface.

```
ok h# 2340.0000 1Meg 2 * 0 vme-a32-slave-range!
ok h# ffe0.0000 1Meg 2 * 0 sbus-a32-master-range!
ok h# 10.0000 obmem h# ffe0.0000 1Meg 2 * iomap-pages
ok 0 vme-slave-ena
ok
```

As shown above, the first command defines the VMEbus slave interface's base address and size of the slave window. The second command defines that any A32 access is translated to an access of the SBus beginning at SBuss address `FFE0.0000`$_{16}$. And the third command creates all necessary entries within the IOMMU to translate the SBus access to an access of the on-board memory beginning at physical address `10.0000`$_{16}$ Finally, the VMEbus slave interface is enabled using the fourth command.

Example 2

The next example lists all steps to be taken, to initialise the VMEbus interface for A24 accesses from the VMEbus beginning at address $C0.0000_{16}$ and ranging to $CF.FFFF_{16}$. The register set associated with the *range number* one (1) is used to control this particular VMEbus slave interface.

```
ok h# c0.0000 1Meg 1 vme-a24-slave-range!
ok h# fff0.0000 1Meg 1 sbus-a24-master-range!
ok h# 20.0000 obmem h# fff0.0000 1Meg iomap-pages
ok 1 vme-slave-ena
ok
```

As shown above, the first command defines the VMEbus slave interface's base address and size of the slave window. The second command defines that any A24 access is translated to an access of the SBus beginning at SBuss address $FFF0.0000_{16}$. And the third command creates all necessary entries within the IOMMU to translate the SBus access to an access of the on-board memory beginning at physical address 20.0000_{16}. Finally, the VMEbus slave interface is enabled using the fourth command.

5.2.9 VMEbus Device Node

The OpenBoot device 3contains the device node for the VMEbus interface and is called "VME". It is a *child device* of the device node "/iommu/sbus" (The full pathname of the VMEbus interface device node is displayed by the command show-devs). The device alias vme is available as a shorthand 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 further methods of the device node
ok selftest .
0
ok device-end
ok
```

The example listed above, selects the VMEbus device and makes it the current node. The word words displays the names of the *methods* of the VMEbus device. And the third command calls the method selftest and the value return by this method is displayed. The last command unselects the current device node leaving no node selected.

204223 8 – 0 May 2000

The following methods are defined in the vocabulary of the VMEbus device:

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

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

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

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

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

`map-in` (*low high size — vaddr*) creates a mapping associating the range of physical address beginning at *low*, extending for *size* Byte, 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* Byte 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-20VT does not support caching. Thus the *cachable?* flag is ignored.

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

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

5.2.10 VMEbus NVRAM Configuration Parameters

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

This section is subdivided into

- General Parameters (see below)
- Parameters Associated to A16 Slave Accesses (see page 177)
- Parameters Associated to A24 Slave Accesses (see page 179)
- Parameters Associated to A32 Slave Accesses (see page 180)
- Parameters Associated to A16 Master Accesses (see page 181)
- Parameters Associated to A24 Master Accesses (see page 182)
- Parameters Associated to A32 Master Accesses (see page 183)
- Parameters Associated to FGA-5100 Mode Selection (see page 184)
- Parameters Associated to DMA (see page 184)

General Parameters

`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, but the operating system which is loaded has to clear it. (default: `true`)

The state of this NVRAM configuration parameter is considered independent from the state of the `vme-init?` configuration parameter.

`vme-bus-timer?` controls whether the VMEbus transaction timer in the FGA-5x00 is used to watch each VMEbus access. When the flag is `true` the transaction timer is enabled. If the flag is `false` the transaction timer is disabled. (default: `true`)

The state of this NVRAM configuration parameter is considered independent from the state of the `vme-init?` configuration parameter.

`vme-bus-timeout` contains the timeout value of the FGA-5x00 VMEbus transaction timer and is a value in the range one to three. Each value selects a particular timeout period. Independent of the state of the configuration parameter `vme-bus-timer?` the timeout value is stored in the appropriate regis-

204223 8 – 0 May 2000

ter. When the value of this configuration parameter is not in the range one through three, then the value three is used instead. (default: 3_{10})

The state of this NVRAM configuration parameter is considered independent from the state of the `vme-init?` configuration parameter.

`vme-slot#` specifies the *logical* VMEbus slot number assigned to the SPARC/CPU-20VT board. The values may be in the range one through 255, but preferably should be set in such a way that it corresponds with the number of an available VMEbus slot.

The state of this configuration parameter does *not* control whether the VMEbus interface is operating as system controller when the configuration parameter's value is one. (default: 1_{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: `false`)

`vme-req-level` specifies the level on which the FGA-5x00 requests the VMEbus. The value of this configuration parameter may be in the range zero through three. Each value corresponds directly with one of the four available bus request levels. (default: 3_{10}, `BR3*` request level)

`#sbus-burst-len` selects the maximum length of SBus burst transactions generated by the FGA-5x00. The value of this configuration parameter may be `8`, `16`, `32`, or `64` – corresponding with the SBus bursts of 8 Byte, 16 Byte, 32 Byte, and 64 Byte. When the value differs from the values listed above the SBus burst length is set to 32 Byte. (default: 32 Byte)

`#sbus-read-stop` selects the SBus "Read Stop Boundary". The value of this configuration parameter may be `8`, `16`, `32`, or `64` – corresponding with the SBus read stop points of 8 Byte, 16 Byte, 32 Byte, and 64 Byte. When the value differs from the values listed above the SBus read stop point is set to 64 Byte. (default: 64 Byte)

`sbus-burst?` controls whether the FGA-5x00 generates SBus burst transactions. When the configuration flag is `true`, then the FGA-5x00 capability to generate SBus bursts is enabled. In the case that the configuration flag is `false` the FGA-5x00 will carry out only single SBus transactions. (default: `true`)

`sbus-hidden-arb?` controls whether the FGA-5x00 operates in the "Hidden SBus Arbitration" mode. When the configuration flag is `true`, then the FGA-5x00 operates in the hidden arbitration mode. In the case that the configuration flag is `false` the FGA-5x00 does not operate in the hidden arbitration mode. (default: `true`)

`sbus-split-flow?` controls whether the FGA-5x00 operates in the "Flow Through" mode. When the configuration flag is `true`, then the FGA-5x00 operates in the flow through mode. In the case that the configuration flag is `false` the FGA-5x00 does not operate in the flow through mode. (default: `true`)

`#sbus-retry` selects the maximum number of SBus clocks before a retry occurs. The value of this configuration parameter may be in the zero through 255. In the case that the value is below a certain limit – typically three SBus clocks – the SBus retry counter is prevented from being set below this limit. (default: 20 SBus clocks)

`vme-sgl-filter?` controls whether the strobe glitch filter for the VMEbus handshake signals are enabled. When the configuration flag is `true`, then the strobe glitch filters for these signals are enabled. In the case that the configuration flag is `false` the strobe glitch filters are disabled. (default: `false`) Only effective for FGA-5100^{E} based CPU boards.

`vme-as-slow?` controls whether the VMEbus AS handshake signal operates in the slow mode. When the configuration flag is `true`, then the AS handshaking operates in the slow mode. In the case that the configuration flag is `false` AS handshaking operates in the fast mode. (default: `false`)

`vme-ds-slow?` controls whether the VMEbus DS handshake signal operates in the slow mode. When the configuration flag is `true`, then the DS handshaking operates in the slow mode. In the case that the configuration flag is `false` DS handshaking operates in the fast mode. (default: `false`)

`vme-arb-mode` selects the arbitration mode of the VMEbus arbiter. This configuration string shall identify one of the following arbitration modes: "`pri`" (priority), "`rrs`" (round robin select), "`sgl`" (single level – not supported), or "`prr`" (priority round robin select). (default: `pri`)

`vme-rel-mode` selects the release mode of the VMEbus requestor. This configuration string shall identify one of the following release modes: "`ror`" (release on request), "`roc`" (release on bus clear), "`rat`" (release after timeout), or "`rwd`" (release when done). (default: `ror`)

When the VMEbus interface requestor is capable of supporting more than one type of release mode, then the value may be as many of the release modes mentioned before as are applicable, but separated by a comma. (For example "`ror,rat`")

204223 8 – 0 May 2000

`vme-early-rel?` controls whether the FGA-5x00 releases the VMEbus at the beginning or end of the current transaction. When the configuration flag is `true`, then the FGA-5x00 releases the bus at the beginning of the current transaction. In the case that the configuration flag is `false` the VMEbus is released at the end of the transaction. (default: `false`)

`vme-bbsy-filter?` controls whether the second stage `BBSY*` glitch filter within the FGA-5x00 is enabled or disabled. When the configuration flag is `true`, the filter is enabled. When it is `false` the filter is disabled. (default: `false`)

`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`)

Note: **The state of the NVRAM configuration parameters listed in the following is only considered by OpenBoot when the configuration parameter `vme-init?` is `true`!**

`vme-intr1` controls whether the VMEbus interrupt request level 1 has to be enabled. When the value is `255` then the VMEbus interrupt request level 1 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 1 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 1 occurs. (default: 255_{10})

`vme-intr2` controls whether the VMEbus interrupt request level 2 has to be enabled. When the value is `255` then the VMEbus interrupt request level 2 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 2 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 2 occurs. (default: 255_{10})

`vme-intr3` controls whether the VMEbus interrupt request level 3 has to be enabled. When the value is `255` then the VMEbus interrupt request level 3 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 3 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 3 occurs. (default: 255_{10})

`vme-intr4` controls whether the VMEbus interrupt request level 4 has to be enabled.

When the value is `255` then the VMEbus interrupt request level 4 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 4 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 4 occurs. (default: 255_{10})

`vme-intr5` controls whether the VMEbus interrupt request level 5 has to be enabled. When the value is `255` then the VMEbus interrupt request level 5 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 5 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 5 occurs. (default: 255_{10})

`vme-intr6` controls whether the VMEbus interrupt request level 6 has to be enabled. When the value is `255` then the VMEbus interrupt request level 6 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 6 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 6 occurs. (default: 255_{10})

`vme-intr7` controls whether the VMEbus interrupt request level 7 has to be enabled. When the value is `255` then the VMEbus interrupt request level 7 is not enabled. In the case that the value is within the range one to seven, the corresponding interrupt handler is activated and the VMEbus interrupt request level 7 is enabled. The values one to seven specify the FGA-5x00 interrupt request line to be asserted when a VMEbus interrupt request level 7 occurs. (default: 255_{10})

`vme-sysfail-assert?` controls whether a non-maskable interrupt is generated upon the assertion of the VMEbus signal `SYSFAIL*`. When the flag is `true` an interrupt handler, dealing with this interrupt, is installed and the ability to generate a non-maskable interrupt upon the assertion of the `SYSFAIL*` signal is enabled. In the case that the flag is `false` the ability to generate a non-maskable interrupt upon the assertion of the `SYSFAIL*` signal is disabled. (default: `false`)

`vme-sysfail-negate?`controls whether a non-maskable interrupt is generated upon the negation of the VMEbus signal `SYSFAIL*`. When the flag is `true` an interrupt handler, dealing with this interrupt, is installed and the ability to generate a non-maskable interrupt upon the negation of the `SYSFAIL*` signal is enabled. In the case that the flag is `false` the ability to generate a non-maskable interrupt upon the negation of the `SYSFAIL*` signal is disabled. (default: `false`)

`vme-acfail-assert?` controls whether a non-maskable interrupt is generated upon the

204223 8 – 0 May 2000

assertion of the VMEbus signal `ACFAIL*`. When the flag is `true` an interrupt handler, dealing with this interrupt, is installed and the ability to generate a non-maskable interrupt upon the assertion of the `ACFAIL*` signal is enabled. In the case that the flag is `false` the ability to generate a non-maskable interrupt upon the assertion of the `ACFAIL*` signal is disabled. (default: `false`)

`vme-ibox-addr` defines the address at which the interrupt box (IBOX) of the FGA-5x00 is accessible within the *short* address space (A16). Only the least significant 16 bits of this 32-bit configuration parameter are considered, and the state of the remaining bits are ignored. Independent of the configuration parameter `vme-ibox-ena?` OpenBoot will set the address of the IBOX. (default: 0_{16})

`vme-ibox-ena?` indicates whether the interrupt box (IBOX), accessible in the *short* (A16) address range of the VMEbus, should be enabled. When this NVRAM configuration parameter is `true` then the IBOX is enabled. In the case that the NVRAM configuration parameter is `false` the IBOX is not enabled. The default value of this NVRAM configuration parameter is `false`.

`fmb-init?` controls whether the FMB system is initialised by OpenBoot. When this flag is `true` the FMB system is initialised according to the state of the NVRAM parameter listed below. In the case that the flag is `false` the FMB system is not initialised. The FMB system is initialised only during the initialisation of the VMEbus interface, which means that the `vme-init?` configuration parameter must be `true`, in order to set up the FMB system. (default: `true`)

`fmb-slot#` specifies the *logical* slot number assigned to the FMB channels of the SPARC/CPU-20VT board. The values may be in the range one through `21`, and preferably should be set in such a way that it corresponds with the number of an available VMEbus slot. (default: 1_{10})

`fmb-addr` specifies the address – the most significant eight bits of a 32-bit address – where the FMB system resides in the *extended* address space (A32) of the VMEbus. (default: FA_{16})

Parameters Associated to A16 Slave Accesses

The NVRAM configuration parameters listed below are associated with the slave interface accessible in the *short* (A16) address range.

`vme-a16-slave-addr` specifies the base address of the slave interface accessible in the *short* (A16) address range of the VMEbus. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a16-slave-size` specifies the size of the memory which is made available to the *short* (A16) address range of the VMEbus. When the value of this configuration parameter is zero OpenBoot will not initialise the slave interface, even if the `vme-a16-slave-ena?` configuration parameter is `true`! The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a16-slave-ena?` indicates whether the slave interface, accessible in the *short* (A16) address range of the VMEbus, should be enabled. When this NVRAM configuration parameter is `true` then the VMEbus slave interface is enabled. In the case that the NVRAM configuration parameter is `false` the VMEbus slave interface is not enabled, and any attempt to access the VMEbus slave interface from the VMEbus will lead to an error termination on the VMEbus. The default value of this NVRAM configuration parameter is `false`.

If the NVRAM configuration parameter `vme-init?` is `true`, OpenBoot will initialise the slave interface according to the configuration parameters described above.

When the `vme-a16-slave-ena?` configuration parameter is `true`, then OpenBoot will initialise the VMEbus slave interface according to the NVRAM configuration parameters `vme-a16-slave-addr` and `vme-a16-slave-size`. It will provide the required amount of physical on-board memory and build up the necessary MMU and IOMMU settings to make the memory available to the VMEbus. The *virtual* base address of the physical on-board memory provided for VMEbus slave accesses is stored in the variable `vme-a16-slave-mem`.

Thus, applications executed within the OpenBoot environment may benefit from this mechanism, because OpenBoot will initialise the slave interface completely according to the NVRAM configuration parameters associated with the slave interface.

In addition, this mechanism allows to report the parameters of the slave interface to an operating system loaded, which in turn provides its own memory and the corresponding MMU and IOMMU settings. In this case the VMEbus device driver is responsible for the access to the slave interface from the VMEbus. In general, the configuration parameter `vme-a16-slave-ena?` must be set to `false` to prevent OpenBoot from initialising and enabling the slave interface when an operating system will be loaded. Supposed that the slave interface is initialised and enabled by OpenBoot prior to loading the operating system, any access from the VMEbus to the slave interface while loading the operating system may alter memory and cause severe damage.

204223 8 – 0 May 2000

Note: **The SPARC/CPU-20VT does *not* provide the ability to access its on-board memory from the VMEbus within the *short* (A16) address range. Therefore, the NVRAM configuration parameters associated with the A16 slave interface, control the access to the registers of the FGA-5x00, which are accessible within the *short* address range. The configuration parameter `vme-a16-slave-size` is not of any importance and will be ignored.**

Parameters Associated to A24 Slave Accesses

The NVRAM configuration parameters listed below are associated with the slave interface accessible in the *standard* (A24) address range.

`vme-a24-slave-addr` specifies the base address of the slave interface accessible in the *standard* (A24) address range of the VMEbus. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a24-slave-size` specifies the size of the memory which is made available to the *standard* (A24) address range of the VMEbus. When the value of this configuration parameter is zero OpenBoot will not initialise the slave interface, even if the `vme-a24-slave-ena?` configuration parameter is `true`! The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a24-slave-ena?` indicates whether the slave interface, accessible in the *standard* (A24) address range of the VMEbus, should be enabled. When this NVRAM configuration parameter is `true` then the VMEbus slave interface is enabled. In the case that the NVRAM configuration parameter is `false` the VMEbus slave interface is not enabled, and any attempt to access the VMEbus slave interface from the VMEbus will lead to an error termination on the VMEbus. The default value of this NVRAM configuration parameter is `false`.

If the NVRAM configuration parameter `vme-init?` is `true`, OpenBoot will initialise the slave interface according to the configuration parameters described above.

When the `vme-a24-slave-ena?` configuration parameter is `true`, then OpenBoot will initialise the VMEbus slave interface according to the NVRAM configuration parameters `vme-a24-slave-addr` and `vme-a24-slave-size`. It will provide the required amount of physical on-board memory and build up the necessary MMU and IOMMU settings to make the memory available to the VMEbus. The *virtual* base address of the physical on-board memory provided for VMEbus slave accesses is stored in the variable `vme-a24-slave-mem`.

Thus, applications executed within the OpenBoot environment may benefit from this mechanism, because OpenBoot will initialise the slave interface completely according to the NVRAM configuration parameters associated with the slave interface.

In addition, this mechanism allows to report the parameters of the slave interface to an operating system loaded, which in turn provides its own memory and the corresponding IOMMU settings. In this case the VMEbus device driver is responsible for the access to the slave interface from the VMEbus. In general, the configuration parameter `vme-a24-slave-ena?` must be set to `false` to prevent OpenBoot from initialising and enabling the slave interface when an operating system will be loaded. Supposed that the slave interface is initialised and enabled by OpenBoot prior to loading the operating system, any access from the VMEbus to the slave interface while loading the operating system may alter memory and cause severe damage.

Parameters Associated to A32 Slave Accesses

The NVRAM configuration parameters listed below are associated with the slave interface accessible in the *extended* (A32) address range.

`vme-a32-slave-addr` specifies the base address of the slave interface accessible in the *extended* (A32) address range of the VMEbus. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a32-slave-size` specifies the size of the memory which is made available to the *extended* (A32) address range of the VMEbus. When the value of this configuration parameter is zero OpenBoot will not initialise the slave interface, even if the `vme-a24-slave-ena?` configuration parameter is `true`! The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a32-slave-ena?` indicates whether the slave interface, accessible in the *extended* (A32) address range of the VMEbus, should be enabled. When this NVRAM configuration parameter is `true` then the VMEbus slave interface is enabled. In the case that the NVRAM configuration parameter is `false` the VMEbus slave interface is not enabled, and any attempt to access the VMEbus slave interface from the VMEbus will lead to an error termination on the VMEbus. The default value of this NVRAM configuration parameter is `false`.

If the NVRAM configuration parameter `vme-init?` is `true`, OpenBoot will initialise the slave interface according to the configuration parameters described above.

204223 8 – 0 May 2000

When the `vme-a32-slave-ena?` configuration parameter is `true`, then OpenBoot will initialise the VMEbus slave interface according to the NVRAM configuration parameters `vme-a16-slave-addr` and `vme-a32-slave-size`. It will provide the required amount of physical on-board memory and build up the necessary MMU and IOMMU settings to make the memory available to the VMEbus. The *virtual* base address of the physical on-board memory provided for VMEbus slave accesses is stored in the variable `vme-a32-slave-mem`.

Thus, applications executed within the OpenBoot environment may benefit from this mechanism, because OpenBoot will initialise the slave interface completely according to the NVRAM configuration parameters associated with the slave interface.

In addition, this mechanism allows to report the parameters of the slave interface to an operating system loaded, which in turn provides its own memory and the corresponding IOMMU settings. In this case the VMEbus device driver is responsible for the access to the slave interface from the VMEbus. In general, the configuration parameter `vme-a32-slave-ena?` must be set to `false` to prevent OpenBoot from initialising and enabling the slave interface when an operating system will be loaded. Supposed that the slave interface is initialised and enabled by OpenBoot prior to loading the operating system, any access from the VMEbus to the slave interface while loading the operating system may alter memory and cause severe damage.

Parameters Associated to A16 Master Accesses

The NVRAM configuration parameters listed below are associated with the master interface to access the *short* (A16) address range.

`vme-a16-master-addr` specifies the base address of the *short* (A16) address range to be accessed on the VMEbus. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a16-master-size` specifies the size of the area in the *short* (A16) address range of the VMEbus which will be accessed. When the value of this configuration parameter is zero OpenBoot will not initialise the master interface, even if the `vme-a16-master-ena?` configuration parameter is `true`! If the specified size exceeds the size of the *short* (A16) address range, then it limits the specified size to 64 KByte. Due to the FGA-5x00 capabilities OpenBoot will always adjust the specified size to 64 KByte. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a16-master-ena?` indicates whether the master interface, to access the *short* (A16) address range of the VMEbus, should be enabled. When this NVRAM

configuration parameter is `true` then the VMEbus master interface is enabled. In the case that the NVRAM configuration parameter is `false` the VMEbus master interface is not enabled. The default value of this NVRAM configuration parameter is `false`.

In the case that the NVRAM configuration parameter `vme-init?` is `true` OpenBoot will initialise the master interface according to the configuration parameters described above. When the `vme-a16-master-ena?` configuration parameter is `true`, then OpenBoot will initialise the necessary registers in the master interface and provides the virtual memory to access the VMEbus. The *virtual* base address necessary to access the VMEbus is stored in the variable `vme-a16-master-mem`. Thus, applications executed within the OpenBoot environment may benefit from this mechanism, because OpenBoot will initialise the master interface completely according to the NVRAM configuration parameters associated with the master interface.

In addition, this mechanism allows to report the parameters of the master interface to an operating system loaded, which in turn provides its own virtual memory to access the VMEbus. In this case the VMEbus device driver is responsible for providing the necessary virtual address range to access the VMEbus. In general, the configuration parameter `vme-a16-master-ena?` must be set to `false` to prevent OpenBoot from initialising and enabling the master interface when an operating system will be loaded.

Parameters Associated to A24 Master Accesses

The NVRAM configuration parameters listed below are associated with the master interface to access the *standard* (A24) address range.

`vme-a24-master-addr` specifies the base address of the *standard* (A24) address range to be accessed on the VMEbus. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a24-master-size` specifies the size of the area in the *standard* (A24) address range of the VMEbus which will be accessed. When the value of this configuration parameter is zero OpenBoot will not initialise the master interface, even if the `vme-a24-master-ena?` configuration parameter is `true`! If the specified size exceeds the size of the *standard* (A24) address range, then it limits the specified size to 16 MByte. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a24-master-ena?` indicates whether the master interface, to access the *standard* (A24) address range of the VMEbus, should be enabled. When this NVRAM configuration parameter is `true` then the VMEbus master interface is enabled. In the case that the NVRAM configuration parameter

204223 8 – 0 May 2000

is `false` the VMEbus master interface is not enabled. The default value of this NVRAM configuration parameter is `false`.

In the case that the NVRAM configuration parameter `vme-init?` is `true` OpenBoot will initialise the master interface according to the configuration parameters described above. When the `vme-a24-master-ena?` configuration parameter is `true`, then OpenBoot will initialise the necessary registers in the master interface and provide the virtual memory to access the VMEbus. The *virtual* base address necessary to access the VMEbus is stored in the variable `vme-a24-master-mem`. Thus, applications executed within the OpenBoot environment may benefit from this mechanism, because OpenBoot will initialise the master interface completely according to the NVRAM configuration parameters associated with the master interface.

In addition, this mechanism allows to report the parameters of the master interface to an operating system loaded, which in turn provides its own virtual memory to access the VMEbus. In this case the VMEbus device driver is responsible for providing the necessary virtual address range to access the VMEbus. In general, the configuration parameter `vme-a24-master-ena?` must be set to `false` to prevent OpenBoot from initialising and enabling the master interface when an operating system will be loaded.

Parameters Associated to A32 Master Accesses

The NVRAM configuration parameters listed below are associated with the master interface to access the *extended* (A32) address range.

`vme-a32-master-addr` specifies the base address of the *extended* (A32) address range to be accessed on the VMEbus. The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a32-master-size` specifies the size of the area in the *standard* (A24) address range of the VMEbus which will be accessed. When the value of this configuration parameter is zero OpenBoot will not initialise the master interface, even if the `vme-a32-master-ena?` configuration parameter is `true`! The default value of this 32-bit NVRAM configuration parameter is zero (`0`).

`vme-a32-master-ena?` indicates whether the master interface, to access the *extended* (A32) address range of the VMEbus, should be enabled. When this NVRAM configuration parameter is `true` then the VMEbus master interface is enabled. In the case that the NVRAM configuration parameter is `false` the VMEbus master interface is not enabled. The default value of this NVRAM configuration parameter is `false`.

In the case that the NVRAM configuration parameter `vme-init?` is `true` OpenBoot will initialise the master interface according to the configuration parameters described above. When the `vme-a24-master-ena?` configuration parameter is `true`, then OpenBoot will initialise the necessary registers in the master interface and provide the virtual memory to access the VMEbus. The *virtual* base address necessary to access the VMEbus is stored in the variable `vme-a24-master-mem`.
Thus, applications executed within the OpenBoot environment may benefit from this mechanism, because OpenBoot will initialise the master interface completely according to the NVRAM configuration parameters associated with the master interface.

In addition, this mechanism allows to report the parameters of the master interface to an operating system loaded, which in turn provides its own virtual memory to access the VMEbus. In this case the VMEbus device driver is responsible for providing the necessary virtual address range to access the VMEbus. In general, the configuration parameter `vme-a32-master-ena?` must be set to `false` to prevent OpenBoot from initialising and enabling the master interface when an operating system will be loaded.

Parameters Associated to FGA-5100 Mode Selection

`fga-5000-comp?` controls whether OpenBoot sets the FGA-5100 into the FGA-5000 compatibility mode during initialisation. When `false`, OpenBoot sets the FGA-5100 into FGA-5100 enhanced mode during initialization, otherwise into FGA-5000 compatibility mode.

`fga-5000-comp?` is only effective for FGA-5100 based CPU boards (FGA-5100^{C} and FGA-5100^{E}). The default value of this NVRAM configuration parameter is `false`.

Parameters Associated to DMA

`vme-dma-prio?` controls the DMA priority on the VMEbus. If `true`, the DMA controller generates 2eVME and MBLT cycles of 2 KByte maximum length, and BLT transfers of 256 Byte maximum length. If `false`, the DMA controller generates 2eVME and MBLT cycles of 256 Byte maximum length, and BLT transfers of 64 Byte maximum length.

Note: **For maximum performance, `vme-dma-prio?` should be set to `true` for DMA. This is especially important for DMA read transfers to slave interfaces which have extensive prefetching support. However, for systems where latency is an important factor,**

204223 8 – 0 May 2000

`vme-dma-prio?` should be set to `false`, as the long block transfers prevent other masters from accessing the VMEbus.

`vme-dma-prio?` is only effective for FGA-5100^E based CPU boards. The default value of this NVRAM configuration parameter is `false`.

5.2.11 DMA Controller Support

The commands listed below are available to control the DMA controller of the FGA-5x00, as well as to get information about the actual state of the DMA controller. See last section for parameters associated to DMA.

`dma-ip?` (— `true` | `false`) checks whether an interrupt is pending because a DMA process has been terminated. The value `true` is returned when an interrupt is pending due to the termination of a DMA process. Otherwise the value `false` is returned to indicate that no interrupt is pending.

`dma-ena` (—) enables the DMA controller and starts a DMA process.

`dma-dis` (—) disables the DMA controller and stops the DMA process currently running.

`dma-halt` (—) halts the DMA process currently running.

`dma-resume` (—) resumes the DMA process that has been halted before.

`dma-src-cap@` (— *data-capability address-capability*) returns the *data-capability* and *address-capability* currently defined for the source of the DMA process.

`dma-src-cap!` (*data-capability address-capability* —) sets the *data-capability* and *address-capability* for the source of the DMA process.
The constants listed below are available to specify the *data-capability* and the *address-capability*:

Value	Data-Capability	Address-Capability
000_2	`cap-d8`	`cap-a16`
001_2	`cap-d16`	`cap-a24`
010_2	`cap-d32`	`cap-a32`
011_2	`cap-blt`	Reserved
100_2	`cap-mblt`	Reserved

Value	Data-Capability	Address-Capability
101_2	In case of FGA-5100^E: `cap-2eVME` otherwise reserved	Reserved
110_2	Reserved	Reserved
111_2	Reserved	In case of FGA-5100^E: `cap-cr/csr` Otherwise reserved

`dma-dest-cap@` (— *data-capability address-capability*) returns the *data-capability* and *address-capability* currently defined for the destination of the DMA process.

`dma-dest-cap!` (*data-capability address-capability* —) sets the *data-capability* and *address-capability* for the destination of the DMA process. The constants listed below are available to specify the *data-capability* and the *address-capability*:

Value	Data-Capability	Address-Capability
000_2	`cap-d8`	`cap-a16`
001_2	`cap-d16`	`cap-a24`
010_2	`cap-d32`	`cap-a32`
011_2	`cap-blt`	Reserved
100_2	`cap-mblt`	Reserved
101_2	In case of FGA-5100^E: `cap-2eVME` Otherwise reserved	Reserved
110_2	Reserved	Reserved
111_2	Reserved	in Case of FGA-5100^E: `cap-cr/csr` Otherwise reserved

`dma-count@` (— *transfer-count*) returns the current state of the transfer count. The value *transfer-count* indicates the number of Byte to be transferred by the DMA controller.

204223 8 – 0 May 2000

Since the DMA controller only transfers a multiple of 32-bit data (*long word*, which is a word in the SPARC terminology), the command returns the appropriate number of *words* to be transferred.

`dma-count!` (*transfer-count* —) sets the number of Byte – *transfer-count* – to be transferred by the DMA controller.

Since the DMA controller only transfers a multiple of 32-bit data (*long-word*, which is a *word* in the SPARC terminology), the command calculates the appropriate number of *words* to be transferred. The *transfer-count* is considered to be a modulo 4 MByte less four Byte number.

`dma-running?` (— `true` | `false`) checks whether the DMA controller is in the *running* state. The value `true` is returned when the DMA controller is currently running. Otherwise the value `false` is returned to indicate that the DMA controller is disabled.

`dma-waiting?` (— `true` | `false`) checks whether the DMA controller is in the *waiting* state. The value `true` is returned when the DMA controller is currently waiting, which means that it has been halted. Otherwise the value `false` is returned to indicate that the DMA controller is not waiting.

`dma-normal-terminated?` (— `true` | `false`) checks whether the DMA process has been terminated successfully. It returns the value `true` when the DMA process has been terminated successfully. Otherwise the value `false` is returned to indicate that the DMA process has been terminated due to a fail state, or because the DMA process is still in progress.

`dma-error-terminated?` (— `true` | `false`) checks whether the DMA process has been terminated unsuccessfully. It returns the value `true` when the DMA process has been terminated due to a fail state. Otherwise the value `false` is returned to indicate that the DMA process has been terminated due to normal termination, or because the DMA process is still in progress.

`.dma-stat` (—) displays the current state of the DMA Status Register.

```
ok .dma-stat
ERR:3 NT:0 HALT:0 RUN:0
ok
```

The fields `NT`, `HALT`, and `RUN` reflect the current state of the DMA controller. When the `NT` field is set to one (`1`) then the DMA controller terminated successfully (*normal termination*). In the case that the `HALT` field is set to one (`1`) then the DMA controller is halted – in general, this field is set along with the `RUN` field. The DMA controller is running when the `RUN` field is set to one (`1`). When the one of the fields described previous are cleared (`0`) the DMA controller is *not* in the particular state.

Typically, the ERR field indicates the course of the DMA controller operation and may indicate the fail states listed in the table below:

Error Code	Description
0	Error occurred on source bus
1	Error occurred on destination bus
2	No error termination
3	No error termination

The following two commands to initiate a DMA transfer do not set the data and address capabilities of the source area and destination area. The capabilities must be set with the `dma-src-cap!` and `dma-dest-cap!` commands appropriately before the DMA transfer is started.

`dma-mem>vme` (*src-addr dest-addr count* — `true` | `false`) initiates a DMA transfer from the SBus to the VMEbus and awaits the termination of the DMA process. The number of Byte given by *count* are transferred from *src-addr* – an address area on the SBus (*virtual* address) – to *dest-addr* – an address area on the VMEbus (*physical* address). The command returns the value `false` when all data have been transferred successfully. Otherwise the value `true` is returned to indicate that an error occurred during the DMA process.

Since the DMA controller only transfers a multiple of 32-bit data (*longword*, which is a *word* in the SPARC terminology), the command calculates the appropriate number of *words* to be transferred. Furthermore, the *count* is considered to be a modulo 4 MByte less four Byte number.

`dma-vme>mem` (*src-addr dest-addr count* — `true` | `false`) initiates a DMA transfer from the VMEbus to the SBus and *awaits* the termination of the DMA process. The number of Byte given by *count* are transferred from *src-addr* – an address area on the VMEbus (*physical* address) – to *dest-addr* – an address area on the SBus (*virtual* address). The command returnes the value `false` when all data have been transferred successfully. Otherwise the value `true` is returned to indicate that an error occurred during the DMA process.

Since the DMA controller only transfers a multiple of 32-bit data (*longword*, which is a *word* in the SPARC terminology), the command calculates the appropriate number of *words* to be transferred. Furthermore, the *count* is considered to be a modulo 4 MByte less four Byte number.

204223 8 – 0 May 2000

5.2.12 Mailboxes and Semaphores

The commands described in this section control the mailboxes, the semaphors, and the interrupt box (IBOX, for an example see page 190).

Commands Associated to Mailboxes and Semaphores

`vme-mbox-take` (*mailbox#* — `true` | `false`) takes the mailbox semaphore specified by *mailbox#* and returns the value `true` when the mailbox semaphore has been taken successfully. The value `false` is returned when the mailbox semaphore has been taken already.

The value of *mailbox#* may be one of the values in the range zero through `15`. Each value specifies one of the 16 Mailbox Registers.

`vme-mbox-give` (*mailbox#* —) gives – releases – the mailbox semaphore specified by *mailbox#*.

The value of *mailbox#* may be one of the values in the range zero through `15`. Each value specifies one of the 16 Mailbox Registers.

`vme-sem-take` (*semaphore#* — `true` | `false`) takes a semaphore specified by *semaphore#* and returns the value `true` when the semaphore has been taken successfully. The value `false` is returned when the semaphore has been taken already.

The value of *semaphore#* may be one of the values in the range zero through `47`. Each value specifies one of the 48 Semaphore Registers.

`vme-sem-give` (*semaphore#* —) gives – releases – the semaphore specified by *semaphore#*.

The value of *semaphore#* may be one of the values in the range zero through `47`. Each value specifies one of the 48 semaphore registers.

Commands Associated to the Interrupt Box

The Interrupt Box is *only* accessible from the VMEbus within the *short* address space (A16). Any Byte access – reading or writing – may lead the FGA-5x00 to generate an interrupt. The address of the interrupt box within the *short* address space may be any Byte location in the range 0000_{16} through $FFFF_{16}$. The commands listed below are available to control and initialise the Interrupt Box.

`vme-ibox-ena` (—) enables the interrupt box.

`vme-ibox-dis` (—) disables the interrupt box.

`vme-ibox-addr@` (— *addr*) returns the physical address *addr* of the interrupt box.

`vme-ibox-addr!` (*addr* —) sets the physical address *addr* of the interrupt box.

Example:

As shown in the example below the first command sets the address of the interrupt box. The interrupt box is accessible at the address 4002_{16} within the VMEbus *short* address space. An SBus IRQ 5 is generated by the FGA-5x00 whenever the interrupt box is accessed from the VMEbus. The fourth command enables the interrupt box.

```
ok h# 4002 vme-ibox-addr!
ok 5 ibox vsi-irq-mapping!
ok true ibox vsi-irq
ok vme-ibox-ena
ok
```

5.2.13 Force Message Broadcast

The commands listed below are availabel to control the Force Message Broadcast (FMB) system and to obtain status information about the state of the FMB system.

`fmb-super-only` (`true` | `false` —) allows or prevents the FMB message registers from being accessed in the *non-privileged* mode. When the value `true` is passed to the command the FMB message register is accessible in the *privileged* mode, as well as in the *non-priviliged* mode. Otherwise – the value `false` is passed to the command – the FMB message registers are accessible in the *privileged* mode only.

`fmb-ena` (*channel#* —) enables the FMB channel specified by *channel#*. The value of *channel#* may be zero or one. Each value specifies one of the two FMB channels.

`fmb-dis` (*channel#* —) disables the FMB channel specified by *channel#*. The value of *channel#* may be zero or one. Each value specifies one of the two FMB channels.

`fmb!` ([`true` | `false`] *channel#* —) enables or disables the FMB channel specified by *channel#*. When the value `true` is passed to the command the FMB channel is enabled. Otherwise – the value `false` is passed to the command – the FMB channel is disabled.

`fmb-slot@` (— *slot#*) returns the slot number *slot#* assigned to the FMB channels.

`fmb-slot!` (*slot#* —) assigns the slot number *slot#* to the FMB channels. The value of *slot#* may be one of the values in the range zero to `21`. Each value speci-

204223 8 – 0 May 2000

fies a specific slot.

`fmb-addr@` (— *fmb-space*) returns the most significant eight bits — the *fmb-space* — of the 32-bit VMEbus address the FMB will respond to when an FMB transaction on the VMEbus is detected.

`fmb-addr!` (*fmb-space* —) sets the most significant eight bits — the *fmb-space* — of the 32-bit VMEbus address the FMB will respond to when an FMB transaction on the VMEbus is detected.

`fmb-ip?` (*channel#* — `true` | `false`) checks whether an interrupt is pending because an FMB message has been accepted or rejected by the channel specified by *channel#*. The value of *channel#* may be zero or one. Each value specifies one of the two FMB channels.

The value `true` is returned when an interrupt is pending. Otherwise the value `false` is returned to indicate that no interrupt is pending.

`fmb-msg@` (*channel#* — *message* `true` | `false`) fetches a message – a 32-bit data – from the FMB channel specified by *channel#*. The *message* and the value `true` are returned when an FMB is available. Otherwise the value `false` is returned to indicate that no FMB message is available.

`fmb-msg!` (*message slot-list channel#* — `true` | `false`) sends the *message* – a 32-bit data – to all FMB channels identified by the *slot-list* and *channel#*. The value `true` is returned when the message has been sent out successfully. Otherwise the value `false` is returned to indicate that one or more FMB channels have rejected the message.

The value of *channel#* may be one of the values in the range zero through one. Each value specifies one of the two FMB channels.

The value of *slot-list* identifies the hosts participating in the FMB transaction. Each bit of the slot list is associated with a host identified by a *unique* FMB slot number. The first bit – bit `0` – relates to the host with the FMB slot number one (`1`); the second bit – bit `1` – relates to the host with the FMB slot number two (`2`); and so forth.

Since the FMB system allows only up to 21 hosts, the command considers only the least significant bits of the parameter *slot-list* (bit `0` through `20`).

`fmb-init` (*slot# fmb-space* —) performs all rudimentary steps to initialise the FGA-5x00 in such a way that the subsequent FMB cycles are carried out using the `fmb-msg!` command.

The slot number *slot#* specifies the slot number the FMB channels are associated with. The value of *slot#* may be one of the values in the range zero to 21. Each value specifies a specific slot.

The last available register set in the FGA-5x00 is initialised to carry out an FMB cycle on the VMEbus within the appropriate VMEbus address area that has been specified by *fmb-space*. The parameter *fmb-space* defines the most significant eight bits (one of 256 16-MByte pages) of the VMEbus address where the FMB area is located. The capabilities of this VMEbus master range are A32/D32 and write posting is disabled. The variable `fmb-va` contains the virtual address to be accessed to execute a FMB cycle on the VMEbus.

The example below assigns the slot number 15_{10} to the FMB channels available (all other hosts must have a different FMB slot number). The FMB address space is set to FA_{16} which means that the FMB system is accessed when the address $FAXX.XXXX_{16}$ appears on the VMEbus address lines (the least significant 24 bits are used to select a specific FMB channel and specific hosts). And the second command enables the second FMB channel.

```
ok d# 15 h# fa fmb-init
ok true 1 fmb!
ok h# 1234AA55 h# 0010.800f 1 fmb-msg!
ok 1 fmb-msg@
ok .s 2drop
1234AA55 ffffffff
ok 1 fmb-msg@
ok .s drop
0
ok
```

Finally the message $1234.AA55_{16}$ is sent to the second FMB channel available on the hosts with the FMB slot number one, two, three, four, 15, and 20. Since the message is sent to the host with the FMB slot number 15 – the host that sent the message –, the message is read from the second FMB channel on the host, as shown by the fourth command. When the FMB channel is read again, and supposed the host did not receive another FMB message, the command `fmb-msg@` will return the value `false` to indicate that no more messages are available.

5.3 Standard Initialisation of the VMEbus Interface

Besides the initialisation performed according to the state of the NVRAM configuration parameters, the VMEbus interface – mainly the FGA-5x00 – is initialised as described below.

204223 8 – 0 May 2000

FGA-5x00 Registers

The FGA-5x00 registers are accessible within SBus slot nine in case of a FGA-5000 or FGA-5100^{C} based CPU board and within SBus slot 15 in case of a FGA-5100^{E} based CPU board.

They begin at offset $\texttt{0FFF.FE00}_{16}$ and occupy the slot's last 512 Byte in ($\texttt{0FFF.FE00}_{16}$ ... $\texttt{0FFF.FFFF}_{16}$). The area in the range $\texttt{0E00.0000}_{16}$ … $\texttt{0FFF.FDFF}_{16}$ within the SBus slot is available for any application. Preferably, this area may be used to access the standard (A24, max 16 MB) and short (A16, max 64 KB) address space of the VMEbus.

FGA-5100^{E}

In addition to the initialisation performed for FGA-5x00 based CPU boards, the following initialisation is done in case of FGA-5100^{E} based CPU boards:

- VME slave read pipelining is enabled for all VMEbus slave range registers.
- BR*, IACKIN*, and BG* filters are enabled if the CPU board operates as system controller.

Using the FGA-5100 Enhanced Feature Set

To provide software compatibility the SPARC/CPU-20VTe is configured to run in the FGA-5000 compatibility mode unless explicitly configured to run in FGA-5100 enhanced mode, for example, by issuing the following OpenBoot commands:

```
ok setenv fga-5000-comp? false
ok setenv vme-dma-prio? true
ok reset
```

The first command sets the FGA-5100 into enhanced mode during subsequent initialisation (see "Parameters Associated to FGA-5100 Mode Selection" on page 184). The second command controls the DMA priority on the VMEbus and enables greater maximum transfer lengths after subsequent initialisation (see "Parameters Associated to DMA" on page 184).

VMEbus Transaction Timer

The FGA-5x00 contains a VMEbus transaction timer which is disabled after a RESET. This timer is enabled during the initialisation phase of OpenBoot and the transaction timeout period is set to the longest possible value (512 µs).

SBus Rerun Limit

The SBus Rerun Limit counter, within the FGA-5x00, is *disabled* to avoid any unproper behaviour of the system.

Interrupts

The FGA-5x00 is initialised in such a way that in the case of pressing the ABORT switch, a nonmaskable interrupt (level 15 interrupt) is generated.

5.4 System Configuration

This section consists of the following parts:

- "System Configuration Register Accesses" on page 194
- "Watchdog Timer" on page 196
- Including "Watchdog Timer NVRAM Configuration Parameters" on page 197
- "Abort Switch" on page 198, including "Abort Switch NVRAM Configuration Parameter" on page 198
- "LEDs, Seven Segment Display and Rotary Switch" on page 198
- "Reset" on page 200
- "ID PROM" on page 201

5.4.1 System Configuration Register Accesses

The following commands are available to read data from and store data in the System Configuration Registers.

`led1-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the First User LED Control Register.

`led1-ctrl!` (*byte* —) stores the 8 bit data *byte* in the First User LED Control Register.

`led2-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Second User LED Control Register.

`led2-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Second User LED Control Register.

`flash-ctrl1@` (— *byte*) returns the contents – an 8 bit data – of the Flash Memory Control Register 1.

`flash-ctrl1!` (*byte* —) stores the 8 bit data *byte* in the Flash Memory Control Register 1.

`idprom-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the IDPROM Control Register.

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

`rotary-switch-stat@` (— *byte*) returns the contents – an 8 bit data – of the Rotary Switch Status Register.

`boot-rom-size-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Boot ROM

204223 8 – 0 May 2000

Size Control Register.

`boot-rom-size-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Boot ROM Size Control Register.

`flash-ctrl2@` (— *byte*) returns the contents – an 8 bit data – of the Flash Memory Control Register 2.

`flash-ctrl2!` (*byte* —) stores the 8 bit data *byte* in the Flash Memory Control Register 2.

`flash-vpp-ctrl@` (— *byte*) returns the contents – an 8 bit data – of the Flash Memory Programming Voltage Control Register.

`flash-vpp-ctrl!` (*byte* —) stores the 8 bit data *byte* in the Flash Memory Programming Voltage Control Register.

`led-display@` (— *byte*) returns the contents – an 8 bit data – of the LED Display Control/Status Register. Since the LED Display Control Register is only writable, the command returns the contents of the LED Display Control Shadow Register.

`led-display!` (*byte* —) stores the 8 bit data *byte* in the LED Display Control/Status Register. Since the LED Display Control Register is only writable, the command stores the given data in the LED Display Control Shadow Register, too.

`fmb-0-data-discard@` (— *byte*) returns the contents – an 8 bit data – of the FMB Channel 0 Data Discard Status Register.

`fmb-1-data-discard@` (— *byte*) returns the contents – an 8 bit data – of the FMB Channel 1 Data Discard Status Register.

`lca-id@` (— *byte*) returns the contents – an 8 bit data – of the LCA ID Register.

5.4.2 Watchdog Timer

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

`wd-dis` (—) stops and disables the watchdog timer.

`wd-timeout@` (— *timeout*) returns the watchdog timer's refernece value in use. The value of *timeout* may be one of the values in the range zero through seven. Each value identifies a particular timeout period as shown in the table below.

`wd-timeout!` (*timeout* —) sets the watchdog timer's reference value for timeout according to the given *timeout.* The value of *timeout* may be one of the values in the range zero through seven. Only the least significant three bit of the value *timeout* are considered. The values select a particular timeout period. The table below lists all possible values:

Table 65 Watchdog Timer Timeout Values

Timeout	$t_{wd\text{-}timeout\text{-}min}$
0	408 ms
1	1.68 s
2	6.7 s
3	26.8 s
4	1 min 48 s
5	7 min 9 s
6	28 min 38 s
7	1 h 54 min

`wd-restart` (—) resets the watchdog timer and starts a new time count. In particular the command invokes one of the commands **`vsi-wdt-restart@`** or **`vsi-wdt-restart!`** to restart the watchdog timer.

The Watchdog Timer is started by the commands listed below:

```
ok 3 wd-timeout!
ok vsi-nmi wdt vsi-irq-mapping!
ok true wdt vsi-irq!
ok wd-ena
ok
```

In this example the *watchdog timer timeout* is set to 26.8 seconds, and a *non-maskable* interrupt is generated whenever half of the watchdog time has expired. The OpenBoot already contains an interrupt handler dealing with the interrupt generated by the watchdog timer, and this interrupt

204223 8 – 0 May 2000

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
```

`wd-reset?` (— `true` | `false`) determines whether a reset has been generated because the watchdog timer has expired. If a reset has been generated because the watchdog timer reached the timeout value, then the value `true` is returned; otherwise the value `false` is returned.

Watchdog Timer NVRAM Configuration Parameters

The NVRAM configuration parameters listed below are available to control the initialisation and operation of the Watchdog Timer. The current state of these configuration parameters are displayed using the `printenv` command, and are modified using either the `setenv`, or the `setdefault` command provided by OpenBoot.

`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`)

`wd-timeout` contains the timeout value of the watchdog timer and is a value in the range `0` to `7`. Each value selects a particular timeout period. Independent of the state of the configuration parameter `wd-ena?` the timeout value is stored in the appropriate watchdog timer register. (default: 7_{10})

5.4.3 Abort Switch

`abort-switch?` (— `true` | `false`) determines the current state of the abort switch. The value `true` is returned when the abort switch is pressed. And the value `false` is returned when the abort switch is released.

Abort Switch NVRAM Configuration Parameter

The NVRAM configuration parameter listed below is available to control the initialisation and operation of the abort switch. The current state of the configuration parameter is displayed using the `printenv` command, and is modified using either the `setenv`, or the `set-default` command provided by OpenBoot.

`abort-ena?` controls whether the abort switch has been enabled. When this flag is `true` the abort switch is enabled and has the same effect as pressing the `STOP-A` key on an available keyboard. If the flag is `false` then the abort switch is disabled. (default: `false`)

5.4.4 LEDs, Seven Segment Display and Rotary Switch

The commands described below are available to control the seven segment LED display, the user LEDs, as well as to get 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 freq led#* —) controls the user LED identified by *led#*. The value of *led#* may be either zero or one. The value zero specifies the *first* user LED, and the value one specifies the *second* user LED. The command only considers the state of the bit `0` of the value *led#*.

The parameters *colour* and *freq* define the colour of the LED and the frequency at which the LED is blinking. 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 constants `no-blinking`, `slow`, `moderate`, and `fast` are available to specify a frequency. The constant `no-blinking` causes the LED to be turned on permanently.

204223 8 – 0 May 2000

The following example shows how to let the second user LED blinking at round about 1 Hz (moderate) in red

```
ok red moderate 1 led!
```

`led-on` (*led#* —) turns the user LED identified by *led#* on. The value of *led#* may be either zero or one. The value zero specifies the *first* user LED, and the value one specifies the *second* user LED. The command only considers the state of the bit `0` of the value *led#*.

`led-off` (*led#* —) turns the user LED identified by *led#* off. The value of *led#* may be either zero or one. The value zero specifies the *first* user LED, and the value one specifies the *second* user LED. The command only considers the state of the bit `0` of the value *led#*.

`led?` (*led#* — `true` | `false`) determines the state of the LED identified by *led#*, and returns either `true` or `false` to indicate if the LED is turned on or off. The value of *led#* may be either zero or one. The value zero specifies the *first* user LED, and the value one specifies the *second* user LED. The command only considers the state of the bit `0` of the value *led#*. When the LED is turned on, then the value `true` is returned; otherwise the value `false` is returned.

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

The value of *led#* may be either zero or one. The value zero specifies the *first* user LED, and the value one specifies the *second* user LED. The command only considers the state of the bit `0` of the value *led#*. Regardless of the color having been set, the LED shines yellow after using this command.

`rotary-switch@` (— *byte*) returns the current state of the rotary switch. The value of *byte* may be one of the values in the range 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.

5.4.5 Reset

The commands listed below are available to initiate various resets, and to obtain information about a previous reset.

`vme-sysreset` (—) asserts the VMEbus SYSRESET* signal and thus causes a system reset.

`reset-call` (—) forces a *local* reset. This command provides the same function as the OpenBoot command `reset`.

`vme-sysreset-in!` (`true` | `false`) allows or prevents the board from being reset by the assertion of the VMEbus SYSRESET* signal. When the value `true` is passed to the command the board will be reset whenever the VMEbus SYSRESET* signal is asserted. Otherwise – the value `false` is passed to the command – the board will not be reset by the assertion of the SYSRESET* signal.

`sbus-reset?` (— `true` | `false`) determines whether the last reset occurred was due to an SBus reset. The value `true` is returned when the last reset was because of an SBus reset. Otherwise it returns the value `false` to indicate that the last reset was not because of an SBus reset.

`wdt-reset?` (— `true` | `false`) determines whether a reset has been generated because the watchdog timer has expired. If a reset has been generated because the watchdog timer reached the timeout value, then the value `true` is returned; otherwise the value `false` is returned.

`vme-sysreset?` (— `true` | `false`) determines whether the last reset occurred was due to the assertion of the VMEbus SYSRESET* signal. The value `true` is returned when the last reset was a VMEbus SYSRESET* reset. Otherwise it returns the value `false` to indicate that the last reset was not a VMEbus SYSRESET* reset.

`vme-sysreset-call?` (— `true` | `false`) determines whether the last reset occurred was due to a VMEbus SYSRESET* call. The value `true` is returned when the last reset was because of a VMEbus SYSRESET* call. Otherwise it returns the value `false` to indicate that the last reset was not a VMEbus SYSRESET* call.

A VMEbus SYSRESET* call is done by clearing the SYSRESET bit in the FGA-5x00 Miscellaneous Control and Status Register.

`reset-call?` (— `true` | `false`) determines whether the last reset occurred was due to a local reset call. The value `true` is returned when the last reset was because of a local reset call. Otherwise it returns the value `false` to indicate that the last reset was not a local reset call.

204223 8 – 0 May 2000

A local reset call is done by clearing the `RESET` bit in the FGA-5x00 Miscellaneous Control and Status Register.

`vme-reset-call?` (— `true` | `false`) determines whether the last reset occurred was due to a reset call initiated by an access via the VMEbus. The value `true` is returned when the last reset was because of a reset call. Otherwise it returns the value `false` to indicate that the last reset was not because of a reset call initiated by an access via the VMEbus.

A reset call is done by clearing the `LOCRESET` bit in the FGA-5x00 Global Control and Status Register.

5.4.6 ID PROM

On the SPARC/CPU-20VT an ID PROM (a serial E^2PROM X24C04) is connected via an I^2C bus. The OpenBoot provides the following commands to access the ID PROM:

`mem>idprom` (*src-addr dest-addr size* —) copies a number of bytes from the on-board memory to the ID PROM. The number of bytes to be copied are specified by *size*. The data are stored beginning at the virtual address *src-addr* and are copied to the ID PROM beginning at address *dest-addr*. The destination address may range from `0` to `511` and specifies the location within the ID PROM. The value of *size* may be one of the values in the range `1` through `512`.

`idprom>mem` (*src-addr dest-addr size* —) copies a number of bytes from the ID PROM to the on-board memory. The number of bytes to be copied are specified by *size*. The data are stored beginning at the virtual address *dest-addr* and are copied from the ID PROM beginning at address *src-addr*. The source address may range from `0` to `511` and specifies the location within the ID PROM. The value of *size* may be one of the values in the range `1` through `512`.

5.5 Flash Memory Support

5.5.1 Flash Memory Programming

The commands listed below are available to access and program the flash memories available on the SPARC/CPU-20VT.

`flash-messages` (— *vaddr*) returns the virtual address of the *variable* `flash-messages`. The state of this variable controls whether the words to erase and program the flash memories will display messages while erasing or programming the flash memories. Messages will not be displayed after *turning off* this variable by `flash-messages off`, and are displayed after *turning on* this variable by `flash-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 accessing using the `select-flash` word.

`boot-flash-va` (— *vaddr*) returns the virtual base address *vaddr* of the boot flash memory.

`user-flash-va` (— *vaddr*) returns the virtual base address *vaddr* of the user flash memory. When the user flash memory is not accessible directly, but only through the flash memory programming window, then the address returned is zero. On the SPARC/CPU-20VT 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 Byte, specified by *count*, stored at *source-addr*.

204223 8 – 0 May 2000

`flash>move` (*source-addr dest-addr count* —) copies a number of Byte, specified by *count*, from the selected flash memory beginning at *source-addr* to *dest-addr*. The flash memory is accessed through the flash memory programming window for reading data from the memory. Thus, the flash memory has to be prepared for accessing using the command `select-flash`.

`fill-flash` (*dest-addr count pattern* —) fills the selected flash memory beginning at *dest-addr* with a particular *pattern*. The number of Byte 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 0xffe98000 size 512 Kbyte
512 Kbyte BOOT flash memory is available at 0xffe58000.
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 0xffe98000 size 512 Kbyte
512 Kbyte BOOT flash memory is available at 0xffe58000.
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 10.0000_{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 Byte beginning from address `flash-va` + 6.8000_{16} to main memory beginning at address 10.0000_{16}.

5.5.2 Flash Memory Device

The device tree of OpenBoot for the SPARC/CPU-20VT 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@0,280000`" 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 flash memory 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.

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

204223 8 – 0 May 2000

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.

`bootflash-#megs` specifies the amount of available user flash memory in MByte. (default: 0 MByte)

`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`. And 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* Byte 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.

Since 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 returned to indicate the fail state.

`read-blocks` (*addr block# #blocks — #read*) reads the number of blocks identified by *#blocks* of length *block-size* Byte, each from the device beginning at the device bvlock *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 Byte *bytes* of a block which is always the size of the flash memory programming window.

`max-transfer` (— *bytes*) returns the size in Byte *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 Byte *length* read from the flash memory.

This method considers the state of the NVRAM configuration parameter `bootflash-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.

5.5.3 Loading and Executing Programs from USER Flash Memory

Besides the ability to load and execute an executable image from disk, or via a network, or other components, the OpenBoot for the SPARC/CPU-20VT 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 an *FCode* program.

204223 8 – 0 May 2000

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-20VT is equipped with one user flash memory device which size is 1 MByte, then commands listed in the following 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
```

5.5.4 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-number* —) 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`).

5.6 On-board Interrupts

Besides the interrupt handlers already available in the standard OpenBoot, the OpenBoot of the SPARC/CPU-20VT provides further handlers that deal with the interrupts generated by following:

- One of the VMEbus interrupt levels one to seven
- Assertion and negation of the `SYSFAIL*` signal
- Assertion of the `ACFAIL*` signal
- Pressing the abort switch
- Watchdog timer, when half the time has expired

5.6.1 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

204223 8 – 0 May 2000

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 `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.

5.6.2 SYSFAIL Interrupt

OpenBoot for the SPARC/CPU-20VT already includes an interrupt handler to serve the non- maskable interrupt generated upon the assertion and negation of the `SYSFAIL*` signal. This handler need not to be installed because it is already installed by OpenBoot.

By default, the interrupts that will be emitted by a status change of the SYSFAIL* signal are disabled and have to be enabled by

```
ok true sysfail- vsi-irq!
ok true sysfail+ vsi-irq!
ok
```

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

When a non-maskable interrupt occurred due to the assertion of the SYSFAIL* signal, the appropriate interrupt handler increments the variable sysfail-asserted? by one to report the occurrence of such an interrupt. The variable sysfail-negated? is incremented by the interrupt handler when the SYSFAIL* signal has been negated and caused a non- maskable interrupt. The state of both variables are obtained by

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

and

```
ok sysfail-negated? ?
1
ok
```

And these variables are cleared – set to zero – by

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

5.6.3 ACFAIL Interrupt

OpenBoot for the SPARC/CPU-20VT already includes an interrupt handler to serve the non- maskable interrupt generated upon the assertion of the ACFAIL* signal. This handler need not to be installed because it is already installed by OpenBoot.

By default, the interrupt that will be emitted by asserting the ACFAIL* signal is disabled and has to be enabled by

```
ok true acfail vsi-irq!
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, 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
```

2042238 – 0 May 2000

```
ok
```

5.6.4 ABORT Interrupt

OpenBoot for the SPARC/CPU-20VT already includes an interrupt handler to serve the non- maskable interrupt generated by pressing the front panel abort switch. This handler need not 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 true abort-key vsi-irq!
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, 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.

5.6.5 Watchdog Timer Interrupt

OpenBoot for the SPARC/CPU-20VT already includes an interrupt handler to serve the non- maskable interrupt generated by the watchdog timer when half of the time has expired. This handler need not to be installed because it is already installed by OpenBoot.

By default, the interrupt that will be emitted by the watchdog timer is disabled – the watchdog timer is disabled – and has to be enabled by

```
ok true wdt vsi-irq!
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 Open-

Boot 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
```

5.7 Viewing the Switch Status and Controlling the Temperature Sensors

The following command displays the current state of all switches. The commands described in the remaining section control the temperature sensors or display their state. All temperature values are defined in °C.

Note: **After using one of the temperature sensor commands below either reset the SPARC/CPU-20VT or use the `watch-temperature` command for proper functioning of the temperature sensor LEDs:**

`.switch-stat` (—) displays the current state of all switches on the SPARC/CPU-20VT (the command is typically declared as headerless).

`get-t` (*sensor# — 1/10-degree degree*) reads the result of the last temperature conversion for the specified temperature sensor *sensor#*. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor. The temperature is given in one-tenth degree *1/10-degree* and degree *degree*.

`get-t-low` (*sensor# — 1/10-degree degree*) returns the current setting of the low temperature register of the temperature sensor specified by *sensor#*. The value of *sensor#* may either be `1`– specifying the first temperature sensor – or `2`– specifying the second temperature sensor. The temperature is given in one-tenth degree *1/10-degree* and degree *degree*.

`get-t-high` (*sensor# — 1/10-degree degree*) returns the current setting of the high temperature register of the temperature sensor specified by *sensor#*. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor. The temperature is given in one-tenth degree *1/10-degree* and degree *degree*.

204223 8 – 0 May 2000

`get-conf` (*sensor# — byte*) reads the status register of the temperature sensor specified by *sensor#*. The contents of the status register data is left on the evaluation stack. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor. The 8-bit *byte* read from the status register is described in the DS1620 data sheet (see "Digital Thermometer and Thermostat – DS1620" in the *SPARC/CPU-20VT Data Sheets*).

`set-t-low` (*1/10-degree degree sensor# —*) sets the low temperature register of the temperature sensor specified by *sensor#*. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor. The temperature has to be specified in one-tenth degree *1/10-degree* and degree *degree*.

`set-t-high` (*1/10-degree degree sensor# —*) sets the high temperature register of the temperature sensor specified by *sensor#*. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor. The temperature has to be specified in one-tenth degree *1/10-degree* and degree *degree*.

`set-conf` (*byte sensor# —*) stores the value *byte* in the configuration register of the temperature sensor specified by *sensor#*. The contents of the status register data is left on the evaluation stack. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor. The 8-bit *byte* read from the status register is described in the DS1620 data sheet (see "Digital Thermometer and Thermostat – DS1620" in the *SPARC/CPU-20VT Data Sheets*).

`tc-start` (*sensor# —*) initiates a temperature conversion process which is carried out by the temperature sensor specified by *sensor#*. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor.

`tc-stop` (*sensor# —*) terminates a temperature conversion process which is carried out by the temperature sensor specified by *sensor#*. The value of *sensor#* may either be `1` – specifying the first temperature sensor – or `2` – specifying the second temperature sensor.

`watch-temperature` (—) puts both temperature sensors into the stand-alone mode and initiates continous temperature conversion.

5.8 BusNet Support

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:

Limitations

- OpenBoot support for the BusNet protocol only allows a participant to operate as a *slave*.
- Network management services are currently *not* supported. Every received packet containing such a request is refused by the BusNet driver.
- 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.

5.8.1 Loading Programs

The OpenBoot provides several methods for loading and executing a program on the machine. They 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 commands 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. OpenBoot provides the following device alias definitions associated with this device:

Alias	Description
`busnet`	TFTP is used to load program
`busnet-tftp`	TFTP is used to load program
`busnet-raw`	Pure binary data is loaded (*raw* device)

204223 8 – 0 May 2000

5.8.2 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 node `vme`.

5.8.2.1 Device Properties

A device's properties identify the characteristics of the package and its associated physical device. The BusNet device is characterised 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. Since 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 operation of the operating system's own BusNet device driver. The register property is declared as follows:

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

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 Byte, the property's value is set to `2048`. In the case that the value of the configuration parameter is not a multiple of 64 Byte, 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 Byte). 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.

2042238 – 0 May 2000

5.8.2.2 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 Byte of the image loaded.

`read` (*addr length — actual*) receives a network packet and stores at most the first *length* Byte in memory beginning at address *addr*. It returns the *actual* number of Byte 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-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 Byte 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.

5.8.2.3 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

2042238 -- 0 May 2000

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 KByte. The default value of this configuration parameter is 2 KByte. If set to another value it must be a multiple of 64 Byte.

The BusNetBusNet protocol does not permit participants to use different packet buffers 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.

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 a 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* 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-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` (8303.0001_{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?` is 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` (8303.0002_{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?` is 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?` or `bn-rarp?` is set to `true`.

204223 8 – 0 May 2000

`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:
```

This configuration parameter *must* be set when one of the configuration parameters `bn-arp?` or `bn-rarp?` are set to `true`.

5.8.3 Device Operation

In general, OpenBoot provides the `boot` command to load a program through a communication channel into memory. (For detailed information about the `boot` command and the associated NVRAM configuration parameters refer to the OpenBoot Command Reference.) 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

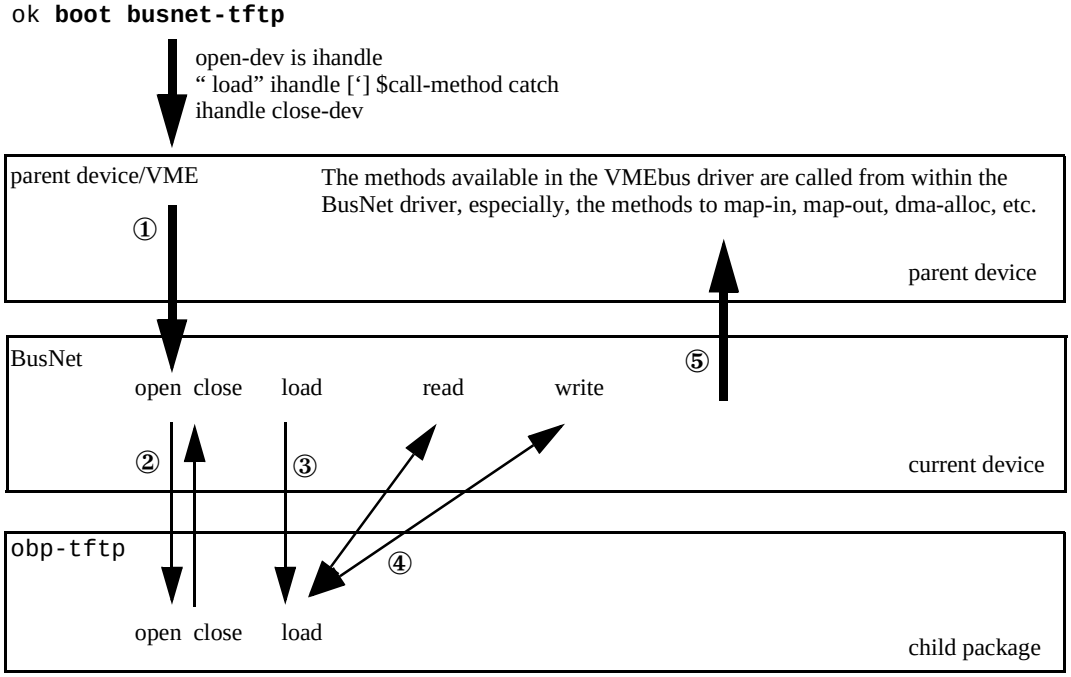
```
ok boot busnet
```

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.

Figure 27 Calling the OpenBoot `boot` Command Using `Busnet-tftp`



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 BusNet-BusNet 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, too – 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.

204223 8 – 0 May 2000

5.8.4 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 reception of data:

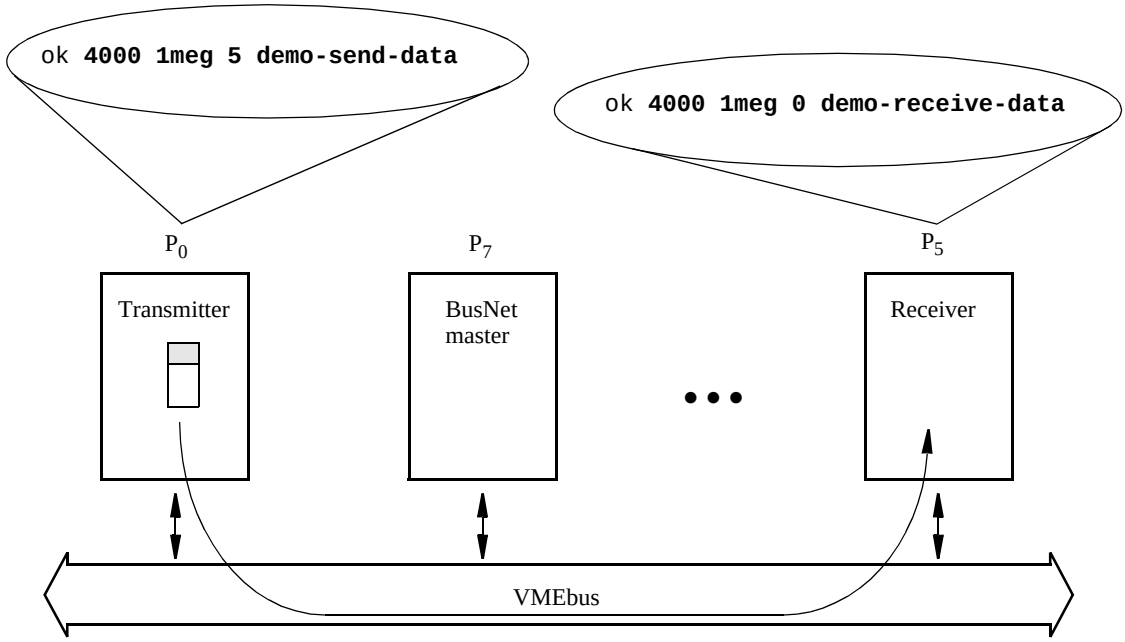
`demo-send-data` (*src-addr size dest-p#* —) sends the amount of data specified by *size* and stored beginning at the address *src-addr* to the participant identified by its logical BusNet address *dest-p#*.

`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 *src-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 providing BusNet master functionality. This is necessary because OpenBoot does not provide BusNet master functionality.**

As shown in the following figure three participants take part in communicating across the network using the BusNet protocol. The logical addresses of the participants are zero, seven and five. Participants P_0 and P_5 are executing OpenBoot, and Participant P_7 runs an operating system which is capable of providing BusNet master functionality – for example Solaris/SunOS, or VxWorks.

Figure 28 Transfering Data Using the BusNet Protocol



When a certain amount of data located in the on-board memory of the participant zero (P_0) – the transmitter – should be transferred 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.**

204223 8 – 0 May 2000

5.8.5 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. Since 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
```

To start the program, the folowing command has to be used:

```
ok init-program go
```

5.8.6 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
```

5.8.7 Booting from a VxWorks BusNet Server

Since 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 a 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` (8303.0001_{16}) is assigned to the NVRAM configuration parameter.

204223 8 – 0 May 2000

`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` (8303.0002_{16}) is assigned to the NVRAM configuration parameter.

`bn-master-en-addr` specifies 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 a 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`).

5.8.8 Setting NVRAM Configuration Parameters

The FGA-5x00 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-a16-slave-addr` must be set to $\texttt{YY00}_{16}$ where `YY` is one of the values $\texttt{00}_{16}$, $\texttt{02}_{16}$, $\texttt{04}_{16}$, ..., $\texttt{FC}_{16}$, or $\texttt{FE}_{16}$. This means that the base address of the FGA-5x00 registers *must* be aligned to a 512 Byte boundary.

`vme-a16-slave-ena?` must be set to `true`.

Table 66 NVRAM Configuration Parameters

Parameter	Default Value	Description
`bn-master-offset`	$\texttt{00000000}_{16}$	
`bn-master-space`	$\texttt{3D}_{16}$	Privileged standard (A24) address range
`bn-master-access`	$\texttt{32}_{16}$	Read/write/D32
`bn-p-offset`	$\texttt{00000000}_{16}$	
`bn-p-space`	$\texttt{3D}_{16}$	Privileged standard (A24) address range
`bn-p-access`	$\texttt{32}_{16}$	Read/write/D32
`bn-p-mbox?`	`true`	Mailbox available (FGA-5x00 mailbox #0)
`bn-p-mbox-offset`	$\texttt{0120}_{16}$	Offset of mailbox #0
`bn-p-mbox-space`	$\texttt{2D}_{16}$	Privileged short (A16) address range
`bn-p-mbox-access`	$\texttt{10}_{16}$	Read/D8
`bn-p-mbox-intr`	`5`	SBus interrupt level 5 is asserted upon a mailbox

204223 8 – 0 May 2000

5.9 Additional Serial Ports

Besides the standard four serial ports – used to connect a keyboard, a mouse, and providing two additional serial ports TTYA and TTYB – the SPARC/CPU-20VT is equipped with two additional serial ports called TTYC and TTYD. The device node of these two serial ports is attached to the device node obio within the device tree.
The OpenBoot for the SPARC/CPU-20VT provides full support for these two additional serial ports.

5.9.1 NVRAM Configuration Parameters

The NVRAM configuration parameters listed and described below are extensions to the standard OpenBoot and are related to TTYC and TTYD.

ttyc-mode (— *addr len*) returns the address *addr* and length *len* of a string that includes the communication characteristics of the serial port TTYC in ASCII representation. The format of the communication characteristics are described in more detail in the *OpenBoot Command Reference Manual*. The default value of this NVRAM configuration parameter is 9600,8,n,1,-.

ttyd-mode (— *addr len*) returns the address *addr* and length *len* of a string that includes the communication characteristics of the serial port TTYD in ASCII representation. The format of the communication characteristics are described in more detail in the *OpenBoot Command Reference Manual*. The default value of this NVRAM configuration parameter is 9600,8,n,1,-.

ttyc-ignore-cd (— true | false) returns either the value true or false to indicate whether the operating system's serial port device driver should ignore carrier-detect on the serial port C. If the value true is returned, then the operating system's serial port device driver ignores the carrier-detect. Otherwise, the device driver has to deal with the assertion and negation of the carrier-detect. The default value of this NVRAM configuration parameter is true.

ttyd-ignore-cd (— true | false) returns either the value true or false to indicate whether the operating system's serial port device driver should ignore carrier-detect on the serial port D. If the value true is returned, then the operating system's serial port device driver ignores the carrier-detect. Otherwise, the device driver has to deal with the assertion and negation of the carrier-detect. The default value of this NVRAM configuration parameter is true.

ttyc-rts-dtr-off (— true | false) returns either the value true or false to in-

dicate whether the operating system's serial port device driver should assert the DTR and RTS signals on the serial port C. If the value true is returned the device driver does not assert the DTR and RTS signals on the serial port. Otherwise, the device driver has to assert and deassert the DTR and RTS signal properly during operation. The default value of this NVRAM configuration parameter is false.

ttyd-rts-dtr-off (— true | false) returns either the value true or false to indicate whether the operating system's serial port device driver should assert the DTR and RTS signals on the serial port D. If the value true is returned the device driver does not assert the DTR and RTS signals on the serial port. Otherwise, the device driver has to assert and deassert the DTR and RTS signal properly during operation. The default value of this NVRAM configuration parameter is false.

204223 8 – 0 May 2000

5.9.2 Selecting Input and Output Device Options

The `input-device` and `output-device` NVRAM configuration parameters control the system's selection of input and output devices after power-on reset. The default `input-device` value is `keyboard` and the default `output-device` value is `screen`. Input and output can be set to the values listed in the table below.

Device-Specifier	**Device Identified by *Device-Specifier***
`keyboard`	Default system keyboard
`screen`	Default graphics display
`ttya`	Serial port A
`ttyb`	Serial port B
`ttyc`	Serial port C
`ttyd`	Serial port D

When the system is reset, the named device becomes the default input or output device. To set the serial port `C` (`TTYC`) as the power-on default input device, the NVRAM configuration parameter `input-device` has to be set as follows:

```
ok setenv input-device ttyc
```

To specify the serial port `C` (`TTYC`) as the default output device, the NVRAM configuration parameter `output-device` has to be modified as described below:

```
ok setenv output-device ttyc
```

The input and/or output device may be changed temporarily by using the `input` and `output` commands described in the *OpenBoot Command Technical Reference Manual* (refer to the section dealing with redirecting input and output).

204223 8 – 0 May 2000

Please Note…

The Sun OpenBoot section is an integral part of the *SPARC/CPU-20VT Reference Guide* (P/N 204223). It is a reprint of the OPEN BOOT PROM 2.0 MANUAL SET. Therefore, it is packaged separately.

The Sun OpenBoot section will always be shipped together with the *Reference Guide*.

Please:

☞ **Insert the Sun OpenBoot section (P/N 203073) now into the *SPARC/CPU-20VT Reference Guide* (P/N 204223).**

☞ **Remove this sheet.**

204223 8 – 0 May 2000

6 Sun OpenBoot (= OPEN BOOT PROM 2.0 MANUAL SET)

Product Error Report

PRODUCT:	SERIAL NO.:
DATE OF PURCHASE:	ORIGINATOR:
COMPANY:	POINT OF CONTACT:
TEL.:	EXT.:
ADDRESS: 	
PRESENT DATE:	
AFFECTED PRODUCT: ❑ HARDWARE ❑ SOFTWARE ❑ SYSTEMS	AFFECTED DOCUMENTATION: ❑ HARDWARE ❑ SOFTWARE ❑ SYSTEMS
ERROR DESCRIPTION: 	

THIS AREA TO BE COMPLETED BY FORCE COMPUTERS:

DATE:

PR#:

RESPONSIBLE DEPT.: ❑ MARKETING ❑ PRODUCTION

ENGINEERING ➠ ❑ BOARD ❑ SYSTEMS

☞ Send this report to the nearest Force Computers headquarter listed on the back of the title page.

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