FORCE COMPUTERS
A SOLECTRON COMPANY

SPARC/CPU-50

Reference Guide

P / N 213380 Revision AC
March 2003

Copyright

A SOLECTRON COMPANY

World Wide Web: www.fci.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.
4211 Starboard Drive
Fremont, CA 94538
U.S.A.

Tel.: +1 (510) 445-6000
Fax: +1 (510) 445-5301
Email: support@fci.com

Europe

Force Computers GmbH
Lilienthalstr. 15
D-85579 Neubiberg / München
Germany

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

Asia

Force Computers Japan KK
Shibadaimon MF Building 4F
2-1-16 Shiba Daimon
Minato-ku, Tokyo 105-0012 Japan

Tel.: +81 (03) 3437 6221
Fax: +81 (03) 3437 6223
Email: support-de@fci.com

213380 420 000 AC

Table of Contents

List of Tables and Figures

Page Tab./Fig.

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-50 into their systems.

Overview of the Manual

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

It includes:

- A brief overview of the product, the specifications, the ordering information: see section 1 “Introduction” on page 1
- The installation instructions, a mechanical overview of the product, the safety notes, initialization prerequisites and requirements as well as the default configuration, for example, the default switch setting and the connector pinouts for the SPARC/CPU-50

  The installation instructions are separated into 3 sections: one general for the complete SPARC/CPU-50 (see section 2 “Installation” on page 5), one for the Base-50(G) (see section 3 “Base-50(G) Installation” on page 17), and one for the I/O-50(G)T (see section 4 “I/O-50(G)T Installation” on page 51). This modular concept should help you to easily find the information needed for your SPARC/CPU-50 configuration.

  The installation instructions also appear as the product’s installation guide – a separate manual delivered together with each product shipped.
- A detailed hardware description: see section 5 “Hardware Description” on page 63
- A detailed description of OpenBoot which controls the CPU board operations: see section 6 “Force OpenBoot Enhancements” on page 107

The Sun OpenBoot 3.x manuals are available on the following web site:

`http://docs.sun.com`

Data Sheets

The following data sheets of module components are relevant for the SPARC/CPU-50. They contain information relevant for configuring and integrating the module in systems and can be found on the respective company's webpage.

- APB (Advanced PCI bus Bridge) – SME2411 (www.sun.com)
- VMEbus Interface – Universe IIb (www.tundra.com)
  - Universe User Manual
  - Universe IIb User's Guide
- UltraSPARC-IIi – SME1040/SME1430 (www.sun.com)
- PCIO (PCI I/O Controller) – STP2003QFP (www.sun.com)
- PHYceiver – ICS1890 (www.icsinc.com)
- PCI-Ultra SCSI (Fast-20) I/O Interface – SYM53C875 (www.lsilogic.com)
- Enhanced Serial Communication Controller – SAB 8253 (www.siemens.com)
- Super I/O – PC87332VLJ (www.national.com)
- Audio Controller – CS4231A (www.crystal.com)
- Real-time Clock and NVRAM – RTC/NVRAM M48T58 (www.st.com)
- Flash Memory – Am29F080B and Am29F016B (www.intel.com)
- Temperature Sensor – LM75 (www.national.com)

Table a History of Manual Publication

SAP No.	Ed./Rev.	Date	Description
208917	1	March 1998	First print
208917	2.0	August 1998	Added descriptions for installing Solaris, added description for OpenBoot plcc2tsop command (version 3.10.4 or greater)
208917	3.0	January 1999	Solaris 2.5.1 core system support instructions removed, added OpenBoot 3.10.4 commands to OpenBoot Enhancements chapter, SPARC/MEM-50-x-5 information added, memory module installation notice added, Ethernet interface information corrected, added note for I^2C_SDAO, added OpenBoot 3.10.6 information, SMART Service Information added, Solaris installation updated
208917	4.0	November 1999	Added and revised safety notes section, removed data sheet section, editorial changes
208917	5.0	March 2000	Processor data sheets and frequencies updated
208917	5.1	July 2000	Replacement Universe II with Universe IIb Replaced ‘300 MHz‘ with ‘333 MHz‘ in table 3 “Excerpt from the SPARC/CPU-50 Ordering Information” on page 3
213380	AA	Sept. 2000	Added VMEbus information on section “VMEbus Interface” on page 14
213380	AB	August 2001	Added chapter Sicherheitshinweise
213380	AC	March 2003	Changed address page, added copyright page

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.

Table c **Product Naming Conventions**

Name	Description
SPARC/CPU-50	Refers to all available product configurations
Base-50(G)	Refers to all base board configurations as described below
Base-50G	Refers to base board with 2-slot front panel (incl. UPA64S slot)
Base-50	Refers to base board with 1-slot front panel (no UPA64S slot)
I/O-50(G)T	Refers to all I/O-board configurations as described below
I/O-50GT	Refers to I/O board supporting UPA64S slot
I/O-50T	Refers to I/O board without UPA64S-slot support

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

The following three types of safety notes appear in this manual. 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: serious injuries to people or severe damage to objects.

Caution

Possibly dangerous situation: slight injuries to people or damage to objects possible.

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

Safety Notes

This section provides safety precautions to follow when installing, operating, and maintaining the SPARC/CPU-50. For your protection, follow all warnings and instructions found in the following text.

General

This *Installation Guide* provides the necessary information to install and handle the SPARC/CPU-50. 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-50 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-50. 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 or uninstalling the board in a VMEbus rack:**
  - **Check all installed boards for steps that you have to take before turning off the power**
  - **Take those steps**
  - **At last turn off the power**
  - **see table 5 “Environmental Requirements of the SPARC/CPU-50” on page 9**
  - **see table 4 “SPARC/CPU-50 Maximum Power Consumption” on page 8**
- **Before touching integrated circuits, ensure that you are working in an electrostatic-free 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 2 “Installation” on page 5.**
- **Before installing or uninstalling an additional device or module, read the respective documentation.**
- **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.**

Power Up

If an unformated floppy disk resides in a floppy drive connected to the SPARC/CPU-50 during powering up, the SPARC/CPU-50 does not boot and the OpenBoot does not appear. Therefore: Never boot the SPARC/CPU-50 with an unformated floppy disk residing in a floppy drive connected to the SPARC/CPU-50.

Operation

While operating the board ensure that the environmental and power requirements as given in table 5 “Environmental Requirements of the SPARC/CPU-50” on page 9 and table 4 “SPARC/CPU-50 Maximum Power Consumption” on page 8 are met.

When operating the board in areas of strong electromagnetic radiation ensure that the board is bolted on the VMEbus rack and shielded by closed housing.

EMC

If boards are integrated into open systems, always cover empty slots.

The front panel of the SPARC/CPU-50(G)T provides 1 cutout for a UPA64S card module. Accordingly the I/O-50(G) front panel provides two cutouts for PMC modules. 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-50(G)T with the blind panel or with a UPA64S card and the I/O-50(G) with blind panels or with PMC modules installed.

Expansion

Check the total power consumption of all components installed (see the technical specification of the respective components). For the total power consumption of the SPARC/CPU-50, see table 4 “SPARC/CPU-50 Maximum Power Consumption” on page 8.

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.

Power Module

The VMEbus power module must be installed if no I/O-50(G)T is installed (see section 3.2.2 “Uninstalling the VMEbus Power Module” on page 22). To protect its components, the SPARC/CPU-50 only powers up, if both the 5 V and the 12 V supply voltages are stable and within their tolerance limits. This is in compliance with the VMEbus specification. However, there are not fully VMEbus compliant systems with power supplies which do not turn on their 12 V supply if the 5 V supply is not loaded. To prevent such systems from running into a power-up deadlock, use a VMEbus board in the system design which loads the 5 V.

Memory Module

Do not install SPARC/MEM-50x and SPARC/MEM-50x-5 memory modules on the same board, otherwise system malfunction may occur.

IOBP

The SPARC/IOBP-50/x is especially designed for the Base-50(G) and for the I/O-50(G)T. Do not use any other I/O panels on the Base-50(G). In addition note the following:

- **Either use the front-panel or the I/O panel Ethernet interface, not both. Check the configuration of your I/O panel.**
- **SW5-2 on the Base-50(G) must be configured to disable the corresponding backplane SCSI termination. This is necessary because the I/O panel provides automatic termination.**

System Controller

If more than one system controller is active in the VMEbus system, the board or other VMEbus participants can be damaged. This is of major importance because this board uses ETL-buffers (enhanced tranceiver logic) which are able to source 60 mA and sink 90 mA on the VMEbus side. Therefore, ensure that only one CPU board is configured to be system controller in the VMEbus system.

Flash Programming

Before programming the boot flash EPROM on-board, save the area containing the OpenBoot image for reprogramming purposes. For example, damage to the image in the boot flash EPROM can occur, if power fails during on-board reprogramming.

Protect your Environment

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

Battery Change

The Lithium battery of the RTC/NVRAM provides a data retention of at least 7 years summing up all periods of actual battery use. Therefore Force Computers assumes that there usually is no need to exchange the Lithium battery except for example in the case of long-term spare part handling.

Please observe the following:

- **Exchange the battery before 7 years of actual battery use have elapsed.**
- **Exchanging the battery always results in data loss of the devices which use the battery as power backup. Therefore, back up affected data before exchanging the battery.**
- **Always use the same type of Lithium battery as is already installed.**
- **Use appropriate tools to remove the battery**
- **When installing the new battery ensure that the marked dot on top of the battery covers the dot marked on the chip.**

RJ-45 Connector

If an RJ-45 connector is available on the board, take into account that the RJ-45 connector type is used for telephone connectors and for twisted pair Ethernet (TPE) connectors. Note that mismatching these 2 connectors may destroy your telephone as well as your SPARC/CPU-50. Therefore:

- **Make sure that TPE connectors near your working area are clearly marked as network connectors.**
- **Make sure that TPE bushing of the system is connected only to safety extra low voltage (SELV) circuits.**
- **Verify that the length of the electric cable connected to a TPE bushing does not exceed 1 kilometer outside the building.**
- **If in doubt, ask your system administrator.**

Sicherheitshinweise

Dieser Abschnitt enthält Sicherheitshinweise, welche bei der Installation, dem Betrieb und der Wartung des SPARC/CPU-50 zu beachten sind. Beachten Sie zu Ihrem Schutz alle folgenden Warnhinweise und Anleitungen.

Dieses Installationshandbuch enthält alle notwendigen Informationen zur Installation und zum Betrieb des SPARC/CPU-50. Da es sich um ein komplexes Produkt mit einer aufwendigen Bedienung handelt, kann keine Garantie dafür übernommen werden, dass die enthaltenen Informationen vollständig sind. Für weitere Informationen wenden Sie sich bitte an Ihren Vertreter der Firma Force Computers.

Das SPARC/CPU-50 erfüllt die gültigen industriellen Sicherheitsanforderungen. Dieses Produkt darf ausschließlich für Anwendungen innerhalb der Telekommunikationsindustrie und der industriellen Steuerung verwendet werden.

Lediglich von Force Computers eingewiesene oder im Bereich Elektrotechnik oder Elektronik qualifizierte Personen sind zur Installation, zum Betrieb und zur Wartung dieses Produktes befugt. Die in dieser Dokumentation enthaltenen Informationen sollen lediglich als Hilfestellung für entsprechend qualifiziertes Fachpersonal dienen. Keinesfalls kann es dieses ersetzen.

Installation

Elektrostatische Entladung und unsachgemäße Installation und Ausbau des Boards kann Schaltkreise beschädigen oder ihre Lebensdauer verkürzen. Deswegen sind folgende Punkte vor der Installation zu überprüfen:

- **Beachten Sie das folgende vor Einbau oder Ausbau des Boards in einem VME Rack:**
  - **Überprüfen Sie alle installierten Boards auf Schritte, die Sie vor dem Abschalten unternehmen müssen.**
  - **Unternehmen Sie diese Schritte.**
  - **Schalten Sie dann den Strom ab.**
  - **Überprüfen Sie die “Environmental Requirements der SPARC/CPU-50” auf Seite 5**
  - **Überprüfen Sie Tabelle 4 “SPARC/CPU-50 Maximum Power Consumption” auf Seite 4**
- **Bevor Sie integrierte Schaltkreise berühren, vergewissern Sie sich, dass Sie in einem ESD-geschützten Bereich arbeiten.**
- **Drücken Sie beim Einbau oder Ausbau des Boards nicht auf das Front Panel, sondern benutzen Sie die Griffe.**

- **Lesen Sie vor Einbau oder Ausbau des Boards den Abschnitt “Installation” auf Seite 1.**
- **Lesen Sie vor dem Einbau oder Ausbau von zusätzlichen Geräten oder Modulen das jeweilige Benutzerhandbuch.**
- **Vergewissern Sie sich, dass das Board über die Stecker P1 und P2 an den VMEbus angeschlossen ist und Strom an allen Power Pins anliegt.**

Hochfahren

Wenn während des Hochfahrens eine unformatierte Diskette in einem Diskettenlaufwerk ist, das mit dem SPARC/CPU-50 verbunden ist, bootet das SPARC/CPU-50 nicht, und OpenBoot erscheint nicht. Fahren Sie deshalb niemals das SPARC/CPU-50 hoch, wenn eine unformatierte Diskette in einem mit dem SPARC/CPU-50 verbundenen Diskettenlaufwerk ist.

Betrieb

Während des Betriebs müssen die Umweltschutz- und die Stromsversorgungsbedingungen gewährleistet sein.

Wenn das Board in Gebieten mit starker elektromagnetischer Strahlung betrieben wird, stellen Sie sicher, dass das Board auf dem VME Rack verschraubt ist und mit einem Gehäuse geschützt ist.

EMV

Wenn Boards in ein offenes System eingebaut werden, decken Sie freie Steckplätze ab.

Das Front Panel des SPARC/CPU-50(G)T hat eine Aussparung für ein UPA64S Karten-Modul. Entsprechend bietet das I/O-50(G) Front Panel zwei Aussparungen für zwei PMC Module. Wenn das Board ohne installiertes Modul geliefert wird, ist die Aussparung im Front Panel durch eine Blende abgedeckt, um EMV-Schutz zu gewährleisten.

UM EMV-Schutz zu gewährleisten, betreiben Sie das SPARC/CPU-50(G)T mit der Blende oder der UPA64S-Karte und das I/O-50(G) mit den zwei Blenden oder den installierten PMC-Modulen.

Erweiterung

Beachten Sie den Gesamtstromverbrauch aller installierter Komponenten (siehe technische Daten der entsprechenden Komponente). In Tabelle 4 “SPARC/CPU-50 Maximum Power Consumption” finden Sie den Gesamtstromverbrauch der SPARC/CPU-50.

Vergewissern Sie sich, daß jeder individuelle Ausgangsstrom jedes Stromverbrauchers innerhalb der zulässigen Grenzwerte liegt (siehe technische Daten des entsprechenden Verbrauchers).

Benutzen Sie bei der Erweiterung ausschließlich von Force Computers empfohlene Komponenten und Systemteile. Ansonsten sind Sie für die Auswirkungen auf EMV und die möglicherweise geänderte Funktionalität des Produktes verantwortlich.

Power Modul

Das VMEbus Power Modul muss installiert werden, wenn das I/O-50(G)T nicht installiert wurde (siehe Abschnitt 3.2.2 “Uninstalling the VMEbus Power Module” auf Seite 18). Zum Schutz seiner Komponenten fährt das SPARC/CPU-50 nur hoch, wenn sowohl die 5V und 12V Spannungsversorgung stabil ist und innerhalb der Toleranzen liegt. Dies geschieht in Übereinstimmung mit der VMEbus Specification. Es gibt jedoch auch System, die nicht völlig VMEbus-kompatibel sind. Deren Spannungsversorgung startet die 12V Spannung nicht, wenn die 5V Spannung nicht existiert. Um bei solchen Systemen einen Stillstand zu vermeiden, verwenden Sie in dem System ein VMEbus Board, das die 5V lädt.

IOBP

Das SPARC/IOBP-50/x wurde speziell für das Base-50(G) und das I/O-50(G)T entwickelt. Verwenden Sie keine anderen I/O Panels mit dem Base-50(G). Außerdem beachten Sie bitte das folgende:

- **Verwenden Sie entweder die Ethernet-Schnittstelle am Front Panel oder am I/O Panel und nicht beide. Überprüfen Sie die Konfiguration Ihres I/O Panels.**
- **SW5-2 auf dem Base-50(G) muß so konfiguriert sein, dass die entsprechend SCSI-Backplane Terminierung ausgeschaltet ist. Dies muß aufgrund der automatischen SCSI-Terminierung durch das I/O Panels geschehen.**

Speichermodul

- **Installieren Sie keine SPARC/MEM-50x und SPARC/MEM-50x-5 Speichermodule auf demselben Board, um Systemfehler zu vermeiden.**

System Controller

Bei mehr als einem aktiven System Controller im VMEbus System können das Board oder andere VMEbus-Karten beschädigt werden. Das ist aufgrund der vom Board verwendeten ETL-Buffer (Enhanced Transceiver Logic) sehr wichtig, die 60mA erzeugen und auf der VMEbus-Seite 90mA herabsenken. Vergewissern Sie sich deshalb, dass nur ein CPU Board im VMEbus System als System Controller eingestellt ist.

Flash Programmierung

Bevor Sie das Boot Flash EPROM on-board programmieren, speichern Sie den Bereich, der das OpenBoot Image enthält, für den Fall von Umprogrammierungen. Es kann zum Beispiel das Image im Boot Flash EPROM beschädigt werden, falls es bei der On-Board Umprogrammierung zu einem Stromausfall kommt.

Umweltschutz

Alte Batterien und/oder Boards oder Systeme müssen stets gemäß der in Ihrem Land gültigen Gesetzgebung entsorgt werden.

Batterie

Die Lithium-Batterie der RTC/NVRAM hat einen Datenretention von mindestens sieben Jahren tatsächlicher Betriebsdauer. Darum geht Force Computers davon aus, dass die Batterie normalerweise nicht ausgewechselt weren muss, außer zum Beipiel bei der Langzeitlagerung von Ersatzteilen. Die folgenden Sicherheitshinweise müssen beim Austausch einer Batterie beachtet werden:

- **Wechseln Sie die Batterie aus, bevor die sieben Jahre tatsächlicher Betriebsdauer abgelaufen sind.**
- **Das Austauschen von Batterien führt immer zu einem Datenverlust bei den Bauteilen, die diese Batterie als Strom-Backup verwenden. Sichern Sie daher die betroffenen Daten vor dem Austausch der Batterie.**

- **Es darf nur der Batterietyp verwendet werden, der auch bereits eingesetzt ist.**
- **Verwenden Sie die entsprechenden Werkzeuge zum Batterieentfernen.**
- **Achten Sie beim Einbau der neuen Batterie das der Punkt auf der Batterie den Punkt auf dem Chip bedeckt.**

RJ-45 Stecker

RJ-45 Stecker werden sowohl für Telefonanschlüsse als auch für Twisted-pair-Ethernet (TPE) verwendet. Die Verwechslung solcher Anschlüsse kann sowohl das Telefonsystem als auch das Board zerstören. Daher:

- **TPE-Anschlüsse in der Nähe Ihres Arbeitsplatzes müssen deutlich als Netzwerkanschlüsse gekennzeichnet sein.**
- **An TPE-Buchsen dürfen nur SELV-Kreise angeschlossen werden (Sicherheitskleinspannungsstromkreise).**
- **Die Länge der an einer TPE-Buchse angeschlossenen Leitung darf nicht mehr als 100 Meter betragen.**

1 Introduction

The SPARC/CPU-50 is a VMEbus computer combining workstation performance and functionality with the ruggedness and expandability of the industry standard 6U VMEbus form factor. Through this combination of powerful processing power with a full set of I/O interfaces the SPARC/CPU-50 becomes a high performance solution for embedded applications.

It is based on:

- The UltraSPARC-IIi CPU
- The Universe IIb (32-bit master/slave) VMEbus interface
- The standard PCI bus as local bus of the CPU board.

Memory

- 32 to 1024 Mbyte EDO DRAM
- Up to 1 Mbyte secondary (L2) cache
- Up to 2 Mbyte boot flash
- Up to 4 Mbyte user flash

Interfaces

Interfaces of the Base-50(G):

- VME64 (with 5-row or factory option 3-row P2 connector)
- Ethernet TPE and MII
- Ultra-wide SCSI
- Two serial I/O ports provide full single-board computer functionality
- Floppy disk, parallel and keyboard/mouse ports
- UPA64S interface (only G-variant)

Interfaces of the I/O-50(G)T:

- VME64 (with 5-row or factory option 3-row P2 connector)
- Ethernet TPE and MII
- Ultra-wide SCSI
- Two standard PMC card slots

Every SPARC/CPU-50 includes OpenBoot – an EPROM based monitor/debugger, which provides the functionality of the boot device as well as the setup for the VMEbus interface.

Table 1 Specifications of the SPARC/CPU-50

Processor	UltraSPARC-IIi 333 MHz processor clock
Shared main memory	32 MByte to 1 GByte DRAM with ECC configurable via number and capacity of memory modules
L2 cache	256 KByte or 1 MByte late write SRAM with parity
PMC slots	2 slots for 32 bit at 33 MHz PMC modules
PCI interface	32 bit at 33 MHz
VMEbus interface	Universe IIb (32-bit master/slave)
SCSI	Ultra-wide SCSI I/O on front panel (8-bit) and backplane (wide)
Ethernet	MII and 10BaseT/100BaseTx half and full duplex Ethernet Twisted Pair on front panel and MII on backplane
Parallel interface with DMA	Centronics compatible, uni- or bidirectional I/O on backplane
Floppy disk interface	I/O on backplane
Serial I/O	2 ports with RS-232 configuration (as factory option RS-422) I/O on front panel (both configurations) and backplane (only RS-232 configuration)
Audio interface	I/O on front panel (microphone and headphone) or backplane
Keyboard/mouse port	I/O on front panel and backplane
Boot PROM (PLCC)	1 MByte PROM (OTP)
Boot flash EPROM (TSOP)	2 MByte flash EPROM On-board programmable with hardware write protection
User flash EPROM (TSOP)	Up to 4 MByte (default: 2 MByte) On-board programmable with hardware write protection
Real-time clock with NVRAM and battery	M48T58 battery-backuped RTC; NVRAM reserved for OpenBoot
Additional features	Reset and abort key, status LEDs, hexadecimal display, rotary switch
Firmware	OpenBoot with diagnostics
Power consumption	**see table 4 “SPARC/CPU-50 Maximum Power Consumption” on page 8**

Table 1 Specifications of the SPARC/CPU-50 (cont.)

Environm. conditions	see “Thermal Requirements” on page 8
PCI compliants VME compliants	PCI Specification Rev. 2.1 ANSI/VITA 1-1994

Product nomenclature

The SPARC/CPU-50 is available in several variants, with or without I/O-50(G)T as well as several memory and speed options. Consult your local sales representative to confirm availability of specific combinations. The table below explains the general product nomenclature.

Table 2 Product Nomenclature of the SPARC/CPU-50

SPARC/CPU-50GT/*mmm-sss-c-uu-ggg*	
G	UPA64S interface slot available on Base-50(G)
T	I/O-50(G)T with dual SCSI, dual Ethernet and 2 PMC slots
mmm	DRAM capacity in MByte
sss	CPU speed in MHz
c	L2-cache size in KByte to be multiplied by 256; for example, *c* = 4 means 1024 KByte (= 4 * 256 KByte)
uu	User flash EPROM size in MByte
ggg	UPA64S card type if preinstalled

Ordering information

The following table is an excerpt from the SPARC/CPU-50 data sheet. Please ask your local Force Computers representative for the current SPARC/CPU-50 data sheet.

Table 3 Excerpt from the SPARC/CPU-50 Ordering Information

Product Name	**Description**
SPARC/CPU-50…	
...GT/64-333-4-2	CPU-50 with 64 MByte DRAM, 333 MHz CPU, 1 MByte cache, 2 MByte user flash, UPA64S slot and I/O-50(G)T (dual SCSI, dual Ethernet, 2 PMC slots)
User upgradable memory module SPARC/MEM-… The memory modules have to be installed on top of the lower memory module installed at delivery of the SPARC/CPU-50 which has 32, 64, 128, or 256 MByte capacity.	
...50U/*xxx*	for memory-module-slot 2 or 4, *xxx* Mbyte, *xxx* = 128 or 256

Table 3 Excerpt from the SPARC/CPU-50 Ordering Information (cont.)

Product Name	Description
Accessories SPARC/…	
…IOBP-50/3	3-row I/O panel for the Base-50(G) and I/O-50(G)T with factory option 3-row P2 connector pinout
…IOBP-50/5	5-row I/O panel for the Base-50(G) and I/O-50(G)T with 5-row P2 connector pinout
…CPU-50/AccKit/*x*	IOBP-50/*x* (*x* = 3 or 5) with cables (see “I/O Panel” on page 39 for the Base-50(G) and page 60 for the I/O-50(G)T)
…UPA/C*x*D	Creator 2D (*x* = 2) or 3D (*x* = 3) graphic card
…CPU-50/TM	*Technical Reference Manual* for SPARC/CPU-50

2 Installation

The following section provides you with information about which variants of the SPARC/CPU-50 are available and how to install them .

2.1 Variants

This section describes the SPARC/CPU-50 variants you may purchase from Force Computers. It is intended to get an overview over all possible configurations with named components which will help to find the information necessary for your configuration in this manual.

Installation

1. Read the overview on the CPU board variants in the remaining parts of this section to get familiar with the naming conventions used in this manual.
2. Read section 2.2 “Installation Prerequisites and Requirements” on page 7 and section 2.2 “Installation Prerequisites and Requirements” on page 7.
3. Depending on the variant under consideration, proceed with reading the respective information in section 3 “Base-50(G) Installation” on page 17 and section 4 “I/O-50(G)T Installation” on page 51.

CPU Board Variants

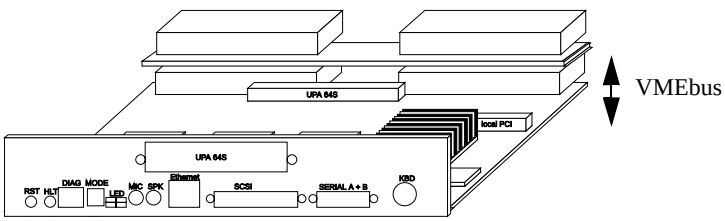
There are 3 variants of the SPARC/CPU-50 all built from the major components shown in figure 4 “Mechanical Construction of Fully-Equipped CPU Board (Schematic)” on page 7:

…CPU-50G

- The SPARC/CPU-50G consists of
  - The 2-slot high base board Base-50G which provides the option to install a UPA64S card
  - The VMEbus power module installed in the 2nd slot of the Base-50G.

Figure 1 **SPARC/CPU-50G (Schematic View)**

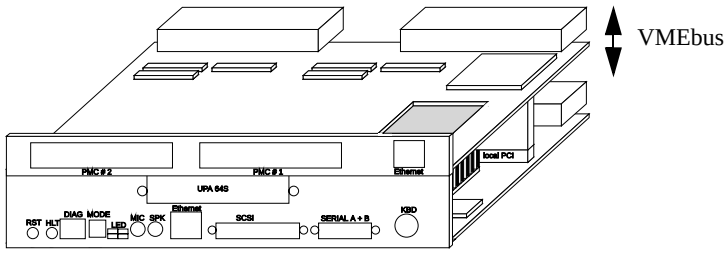
Base-50G with VMEbus power module



…CPU-50GT

- The SPARC/CPU-50GT consists of
  - The 2-slot high base board Base-50G, which provides the option to install a UPA64S card,
  - The 1-slot high I/O board I/O-50GT.

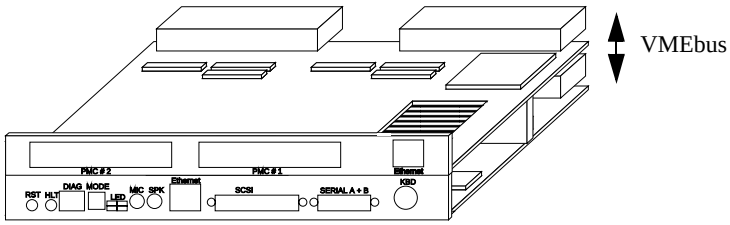
Figure 2 **SPARC/CPU-50GT (Schematic View)**



…CPU-50T

- The SPARC/CPU-50T consists of
  - The 1-slot high base board Base-50
  - The 1-slot high I/O board I/O-50T.

Figure 3 **SPARC/CPU-50T (Schematic View)**



Major Components

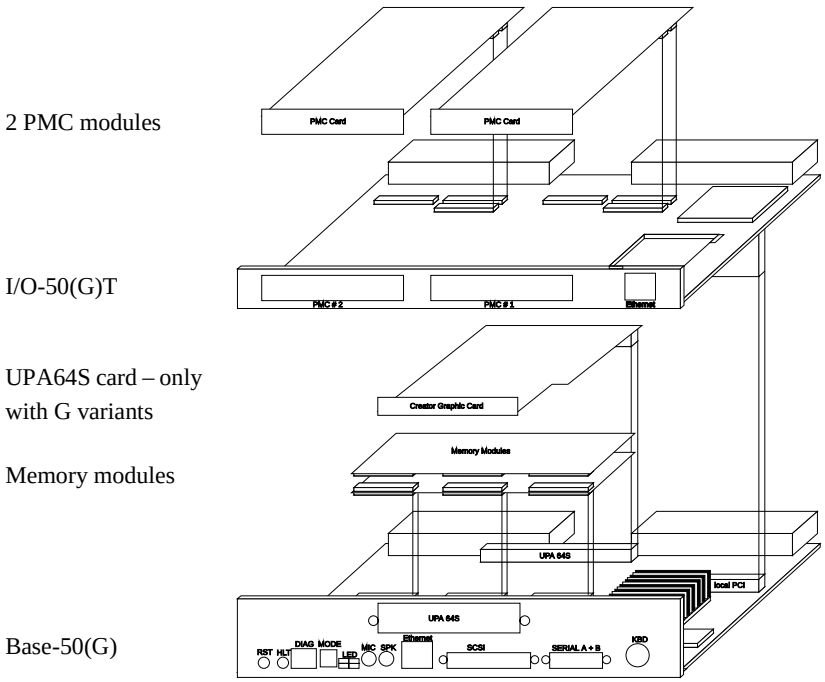
The following figure is intended to provide an overview of all major components of a SPARC/CPU-50.

Note: **Note that in the following figure the terms Base-50(G) and I/O-50(G)T are generalizations of the available base board and I/O board variants.**

In general:

- Base-50 and Base-50G are collectively called Base-50(G)
- I/O-50T and I/O-50GT are collectively called I/O-50(G)T.

Figure 4 Mechanical Construction of Fully-Equipped CPU Board (Schematic)



2.2 Installation Prerequisites and Requirements

Note: **Before powering up check this section for installation prerequisites and requirements, section "Safety Notes" on page xxi for relevant safety notes and check the consistency of the current switch setting (see section 3.4 "Switch Settings" on page 25).**

2.2.1 Requirements

The installation requires only

- Power supply
- Minimum airflow meeting the thermal requirements
- 2 slots of a VMEbus backplane with P1 and P2 connectors

Power Supply

The power supply needs to provide the following voltages:

- +5V, +12V, and –12V

Concerning the SPARC/CPU-50, the power supply must meet the specifications given in the following table.

Table 4 SPARC/CPU-50 Maximum Power Consumption

CPU Board	+5V	+12V	-12V	Major Components of CPU Board Configuration
SPARC/…				
...CPU.50/mmm-333-4-2	7.1A	500mA	200mA	- (1-slot board)
…CPU-50G	7.5A	500mA	200mA	including VMEbus power module and 1 MEM-50L/64, but excluding any I/O board
…CPU-50(G)T	9A	500mA	200mA	including the I/O-50(G)T and 1 MEM-50L/64, but excluding any UPA64S card and excluding any PMC module
…CPU-50GT	12A	500mA	200mA	including I/O-50GT and UPA64S Creator 3D graphics card and 1 MEM-50L/64, but excluding any PMC module

Caution

The VMEbus power module must be installed if no I/O-50(G)T is installed (see section 3.2.2 “Uninstalling the VMEbus Power Module” on page 22).
To protect its components, the SPARC/CPU-50 only powers up, if both the 5V and the 12V supply voltages are stable and within their tolerance limits. This is in compliance with the VMEbus specification. However, there are not fully VMEbus compliant systems with power supplies which do not turn on their 12V supply if the 5V supply is not loaded. To prevent such systems from running into a power-up deadlock, use a VMEbus board in the system design which loads the 5V.

Thermal Requirements

The operating temperature is 0×C to +55×C (humidity 5% to 95% non-condensing at +40×C), when operating the SPARC/CPU-50 in systems providing a minimum forced airflow of 300 LFM (linear feet per minute). The typical operating temperature of the system is 0 °C to +40 °C.

Table 5 Environmental Requirements of the SPARC/CPU-50

	Operating	Non-operating
Temperature	0°C to +55°C	–40°C to +85°C
Forced air flow	300 LFM (linear feet per minute)	–
Temp. change	+/– 0.5°C/min	+/– 1°C/min
Rel. humidity	5% to 95% noncondensing at +40°C	5% to 95% noncondensing at +40°C
Altitude	–300 m to +3,000 m	–300 m to +13,000 m

Backplane Configuration

The CPU board includes an IACK daisy-chain driver. If the CPU board is plugged in slot 1 and configured accordingly by SW800-1 and SW800-2 (see table 12 “Default Switch Settings” on page 25), the board acts as IACK daisy-chain driver. Plugged in any other slot the board closes the IACKIN-IACKOUT path. Therefore:
If not on an active backplane,

- Remove the jumper on the backplane connecting BG3IN* and BG3OUT* for the SPARC/CPU-50 slots which actually are connected to the backplane.
- Assemble the jumpers for BG3IN* and BG3OUT* on
  - Lower and higher slots on the backplane where no board is plugged
  - On the middle slot of a SPARC/CPU-50GT which has no backplane connection due to the installation space for a UPA64S card.

Slot-1 Function

If more than one system controller is active in the VMEbus system, the board or other VMEbus participants can be damaged. This is of major importance because this board uses ETL-buffers (enhanced tranceiver logic) which are able to source 60 mA and sink 90 mA on the VMEbus side.

Note: **Always ensure that only one CPU board is configured to be system controller in the VMEbus system.**

Audio Interfaces

Simultaneous use of the audio interfaces available on the front panel and on the backplane can damage on-board hardware or connected devices. For example: never use the headphone/line audio output at the backplane, if a headphone is plugged into the front-panel jack.

Note: **Always use at most one of the interfaces if an audio interface is available on both the front panel and the backplane.**

Table 6 Audio Interfaces Requirements

Interface	Description
Stereo Micro In (op-amp pre-amp with 18 dB gain)	• Signal level: single-ended condensator microphones with signal level – Up to 12mV with 20 dB gain inside Codec enabled – Up to 120mV with 20 dB gain inside Codec disabled • Availability: on front panel and as factory option on backplane instead of Aux#2 In
Stereo Head-phone/ Line Out	• Signal level: maximum $2V_{RMS}$ line-level signal output (also designed to directly drive headphones) • Availability: on front panel and on backplane
Stereo Line In	• Signal level: typical 47 kΩ audio input impedance; maximum full scale input of $2V_{RM}$ • Availability: on backplane
Stereo Aux#1 In	• Signal level: ~10 kΩ input impedance; maximum full scale input of $2V_{RMS}$ • Availability: on backplane
Stereo Aux#2 In	• Signal level: ~10 kΩ input impedance; maximum full scale input of $2V_{RMS}$ • Availability: on backplane

Table 6 Audio Interfaces Requirements (cont.)

Interface	Description
Mono In	• Signal level: typical 47 kΩ audio input impedance; nominally $1V_{RMS}$ maximum (centered around 2.1V) input signal level • Availability: as factory option on front panel instead of Micro In and as factory option on backplane instead of Aux#1 In
Mono Out	• Signal level depends on the setting of `OLB` which is a bit in the Codecs Alternate Feature Enable I register (I16) – Maximum $1V_{RMS}$ output (centered around 2.1V) if `OLB` = `1` – Maximum $0.707V_{RMS}$ (centered around 2.1V) if `OLB` = `0` Default is `OLB` = `0`. • Availability: as factory option on backplane instead of Headphone/Line Out

2.2.2 Memory Modules

The main memory capacity is adjustable via installation of the appropriate memory modules. The qualified memory modules depend on the SPARC/CPU-50 processor frequency. They are given in the following table.

Table 7 Qualified Memory Modules

Processor frequency	Memory modules
up to 300 MHz	SPARC/MEM-50x
	SPARC/MEM-50x-5
333 MHz and above	SPARC/MEM-50x-5

Caution

Do not install SPARC/MEM-50x and SPARC/MEM-50x-5 memory modules on the same board, otherwise system malfunction may occur.

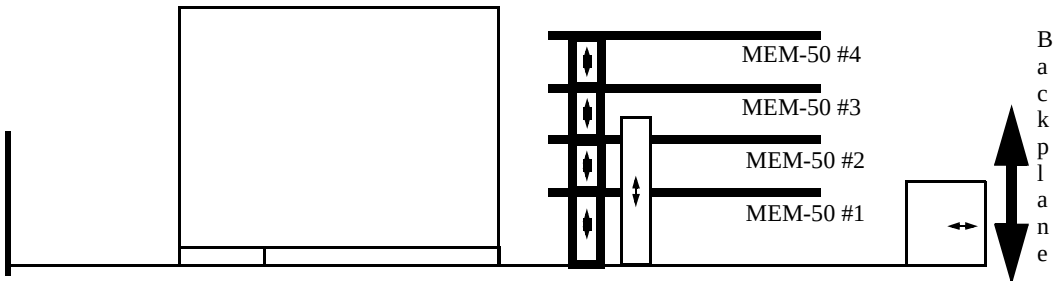
In the following it will be refered to all memory module types as SPARC/MEM-50x.
The Base-50(G) can hold 1 to 4 memory modules providing up to 1 GByte DRAM capacity. 1 memory module can carry 2 memory banks.

Note: **At least 1 lower memory module SPARC/MEM-50L is required.**

See the following figure for the memory module numbering scheme:

- Memory modules #1 and #2 are located facing the first VME slot the SPARC/CPU-50 occupies.
- Memory modules #3 and #4 are located facing the second VME slot the SPARC/CPU-50 occupies.

Figure 5 **MEM-50 – Memory Module Numbering Scheme**



The memory configuration is adjustable to the application's needs via selection of the appropriate memory modules. The memory configuration must fulfill the following requirements:

- The lowest memory module (#1) must be a SPARC/MEM-50L – which is a lower memory module.
- The top memory module (with the greatest number in your configuration given the number scheme in the figure above) can be a SPARC/MEM-50M or SPARC/MEM-50U – which is a middle (M) or upper (U) memory module. The upper module misses the connectors for another memory module to be stacked on top.
- The memory modules between the lowest and the top memory module must be SPARC/ MEM-50M, i.e. middle memory modules.
- If a UPA64S card is installed, at most 2 memory modules can be installed and memory module #2 must be a SPARC/MEM-50U, i.e. an upper memory module.
- Note the limitations given by the SPARC/CPU-50 configuration under consideration (see section 3.2 "Base-50(G) Mechanical Construction" on page 19).

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 8 Qualified Memory Configurations (all data in MByte)

Total capacity	64	128	256	384	512	768	1024
Mem. module #4	–	–	–	–	–	–	256
Mem. module #3	–	–	–	–	–	256	256
Mem. module #2	–	–	–	128	256	256	256
Mem. module #1	64	128	256	256	256	256	256

For installation information see the respective *Installation Guide* delivered together with the memory modules.

2.2.3 Solaris Installation

When installing Solaris, there are some general installation guidelines to be followed before and during Solaris installation and a specific guideline related to Ethernet and SCSI to be followed after Solaris installation (see “” on page 14 and “SCSI” on page 15).

General Installation Guidelines

Note: **Solaris versions and hardware updates prior to 2.5.1 11/97 and 2.6 03/98 are not supported.**

Required Software Packages

The following Solaris software packages must be installed, otherwise Solaris fails to boot. (Applies to Solaris 2.5.1 and 2.6 only.)

Table 9 Required Solaris Packages

Package	Description
`SMEvplr.u`	SME platform links (root)
`SMEvplu.u`	SME platform links (usr)
`SUNWvplr.u`	SMCC sun4u new platform links
`SUNWvplu.u`	SMCC sun4u new usr/platform links

When setting up Solaris interactively, these packages can be installed by selecting the proper software group in the `Software` dialog. Customize the software groups as follows:

Table 10 Customizing Solaris

Software Group	Customization required for		
	Solaris 2.5.1 11/97	Solaris 2.6 03/98 and later	Solaris 2.7 and later
Entire distribution plus OEM support	No customization is required		No customization required
Entire distribution	Select the following cluster: • SMCC platform links		
Developer system support			
End user system support			
Core system support			

VMEbus Interface

In order to support the Universe IIb, the Solaris Driver Package Release 2.8 or higher and the VMEbus Driver FRCvme V2.4.2 or higher have to be used**.**

Ethernet

When installing Solaris for the first time, the installation program prompts for a default network device (`hme0`, `hme1`, ...). The numbering of the network devices depends on the configuration of the board. However, the numbers in the network device name do not change once Solaris has been set up, even if the hardware configuration is changed.
The following list associates the network devices of a given board configuration with the network device names.

Table 11 Network Device Naming when Installing Solaris

Board configuration	Network device name	Network device
Base board only	`hme0`	Eth. #1 on base board
Base board and I/O board	`hme0`	Eth. #2 on I/O board
	`hme1`	Eth. #1 on base board

Table 11 Network Device Naming when Installing Solaris (cont.)

Board configuration	Network device name	Network device
Base board, I/O board and additional `hme` network devices on PMC modules installed on the I/O board	`hme0`	Eth. #2 on I/O board
	`hme1, .., hme<n>`	*n* `hme` devices on PMC modules
	`hme<n+1>`	Eth. #1 on base board

SCSI

The Solaris SCSI driver may revert wide SCSI devices, which are connected to the front-panel SCSI connector, to asynchronous mode. However, it is possible to operate such a configuration in synchronous mode also by inserting the following line into `/kernel/drv/glm.conf`:
`target`*n*`-scsi-options=0x5f8`
where *n* is the SCSI ID of the wide SCSI device under consideration. In case of several wide SCSI devices insert the respective line per device. Terminate the file with a semicolon `;`.

For further information on SCSI configuration, see section 3.5.4 “SCSI #1 Configuration” on page 33.

2.2.4 Terminal Connection

The SPARC/CPU-50 provides 2 serial interfaces (A and B) which are implemented on the Base-50(G). For the initial power up, a terminal can be connected to interface A via the front-panel 26-pin-MicroD-Sub connector SERIAL A+B. Per default, all serial I/O interfaces provide an RS-232 interface. As factory option the 2 interfaces can be configured as RS-422 interfaces.
For information on the serial interface connector pinout, see section 3.5.5 “Serial I/O Connector Pinout” on page 36.

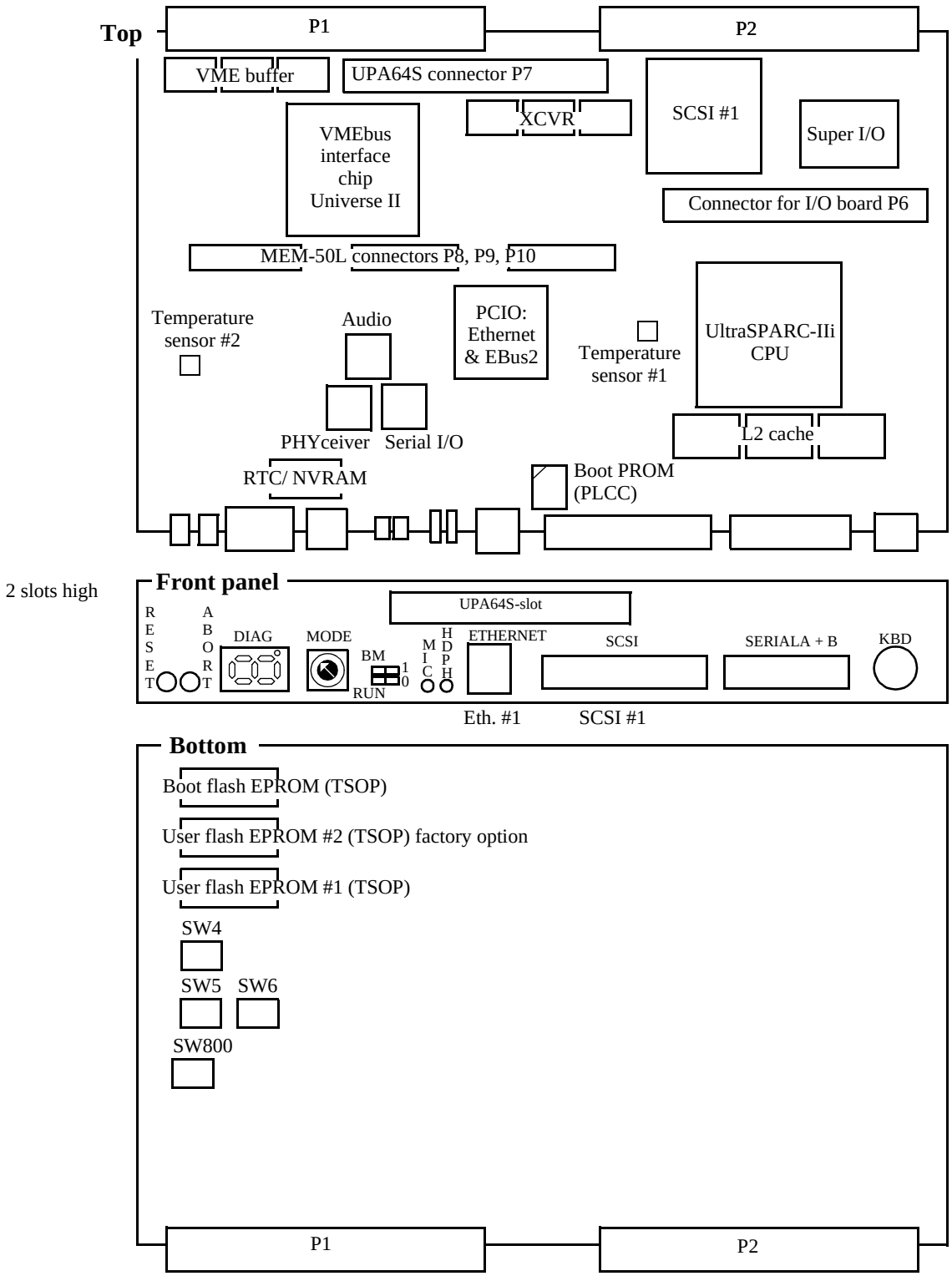
3 Base-50(G) Installation

This section extends **section 2 “Installation” on page 5** by providing the installation information for the Base-50(G) in its various configurations (see section 3.2 “Base-50(G) Mechanical Construction” on page 19).

Note: **Note that there is no need to refer to the installation subsections within section 3.2 “Base-50(G) Mechanical Construction” on page 19 unless you want to change the delivered configuration to upgrade memory, replace a UPA card or the like.**

3.1 Location Overview

Figure 6 Location Diagram of the Base Board (Schematic)



3.2 Base-50(G) Mechanical Construction

The Base-50(G) occupies 2 VMEbus slots and consists of the following major components:

- I/O connector for the VMEbus power module or an I/O-50(G)T
- 3 memory module connectors for up to 4 MEM-50 modules if no UPA64S card is installed or for up to 2 memory modules if a UPA64S card is installed.

The following figures show typical Base-50(G) configurations:

- The Base-50 as used in a 2-slot configuration together with an I/O-50T: see figure 16 “Mechanics of a SPARC/CPU-50T” on page 53.
- The Base-50G in 2-slot configuration with installed VMEbus power module:
  - see figure 7 “Base-50G Mechanical Construction (incl. 4 MEM-50)” on page 19
  - figure 8 “Base-50G Mechanical Construction (incl. 2 MEM-50 and UPA Card)” on page 20.

Figure 7 Base-50G Mechanical Construction (incl. 4 MEM-50)

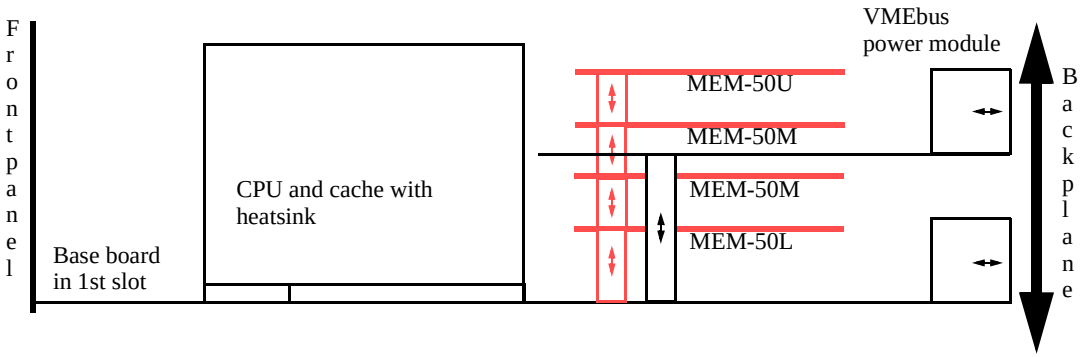
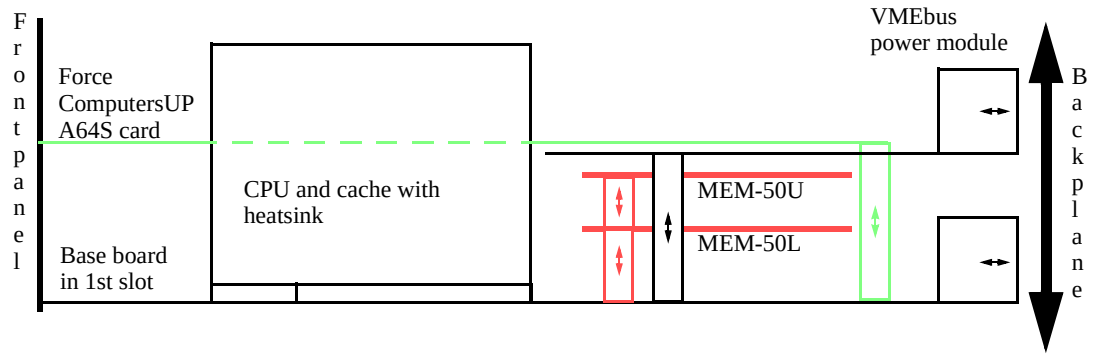
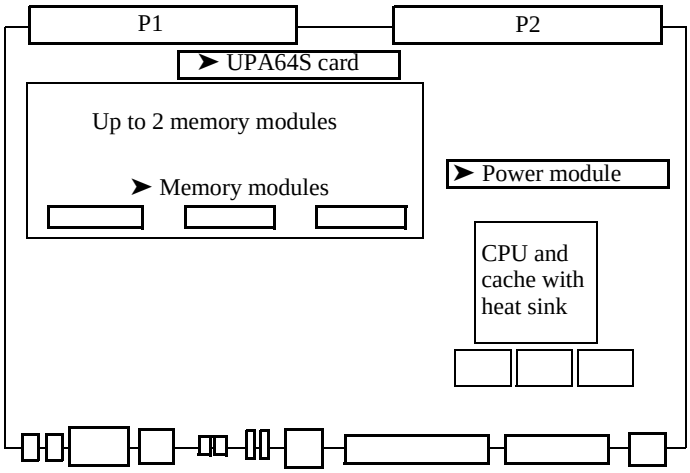


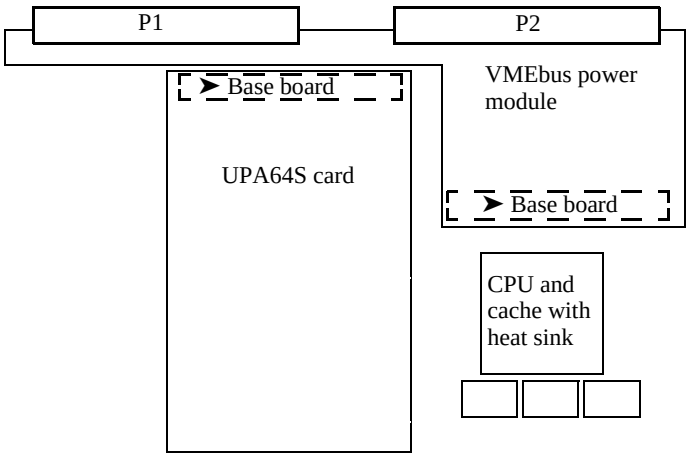
Figure 8 **Base-50G Mechanical Construction (incl. 2 MEM-50 and UPA Card)**



1st of 2 slots
Base board with up to 2 memory modules



2nd of 2 slots
UPA64S card and VMEbus power module

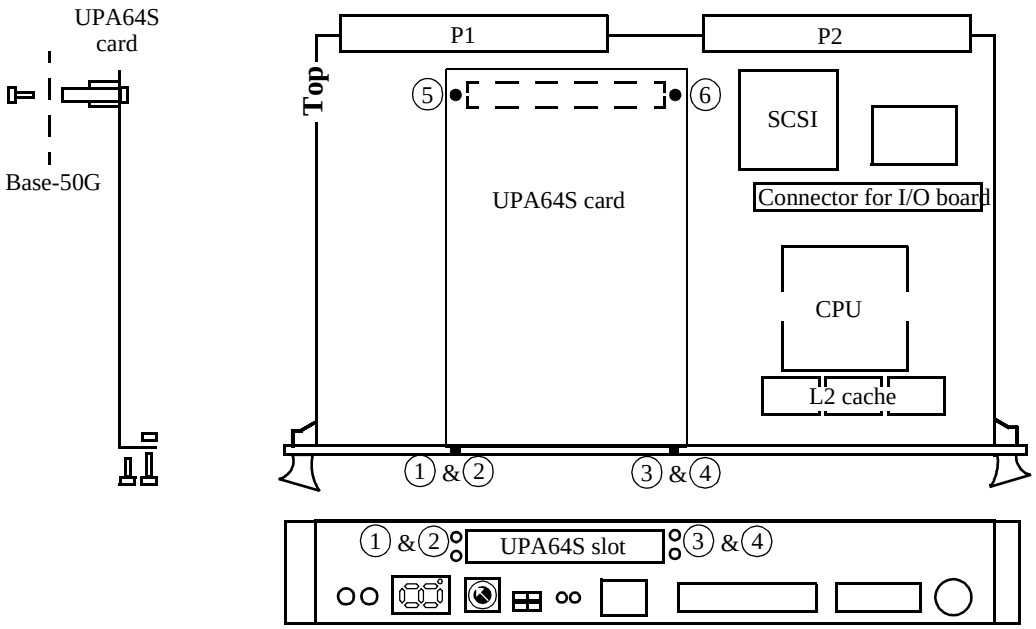


3.2.1 Installing a Force Computers UPA64S Card on a Base-50G

You can only install a Force Computers UPA64S card if you purchased a SPARC/CPU-50 variant with an extra slot for an UPA64S card which is expressed by a G in the product name. The UPA64S card is connected to the Base-50G via the UPA64S connector P7 (see figure 6 “Location Diagram of the Base Board (Schematic)” on page 18).

Note: **Only use UPA64S cards from Force Computers specified to be used with the SPARC/CPU-50.**

Figure 9 Mounting a UPA64S Card



For the locations mentioned in the following description the figure above.

Installation of a UPA64S Card

1. If an I/O-50GT is installed, remove it as described in the I/O-50(G)T installation section.
2. Remove the blind panel fixed in the UPA64S front panel slot of the Base-50G. Store it in a safe place for later use.
3. Plug the UPA64S card to the respective UPA64S connector on the Base-50G.
4. Fix the UPA64S card with 2 screws at location 5 and 6 on the bottom side of the Base-50G.
5. Fix the UPA64S card with the 4 screws and 2 nuts from the blind panel to the front panel of the Base-50G at location 1...4.

Now the UPA64S card is installed.

6. If an I/O-50GT was installed, fix it again as described in the I/O-50(G)T installation section.

Uninstalling a UPA64S Card

1. If an I/O-50GT is installed, remove it as described in the I/O-50(G)T installation section.
2. Remove the 4 screws and 2 nuts on the Base-50G front panel at locations 1...4.
3. Remove the 2 screws at locations 5 and 6 on the bottom side of the Base-520G.
4. Remove the UPA64S card by lifting it.
5. If you do not install the UPA64S card again, fix the blind panel.
6. To install the I/O-50GT again refer to the installation section of the I/O-50(G)T.

3.2.2 Uninstalling the VMEbus Power Module

The SPARC/CPU-50 needs a second VMEbus slot for the heatsink of the UltraSPARC-IIi and for more current. The additional power requirements can be satisfied by

- A separate VMEbus power module which uses the I/O-board connector to provide the base board with +5V-based and +/–12V-based power from a 2nd VMEbus slot
- A separate I/O board I/O-50(G)T satisfying the same power requirements in addition to providing additional interfaces

Caution

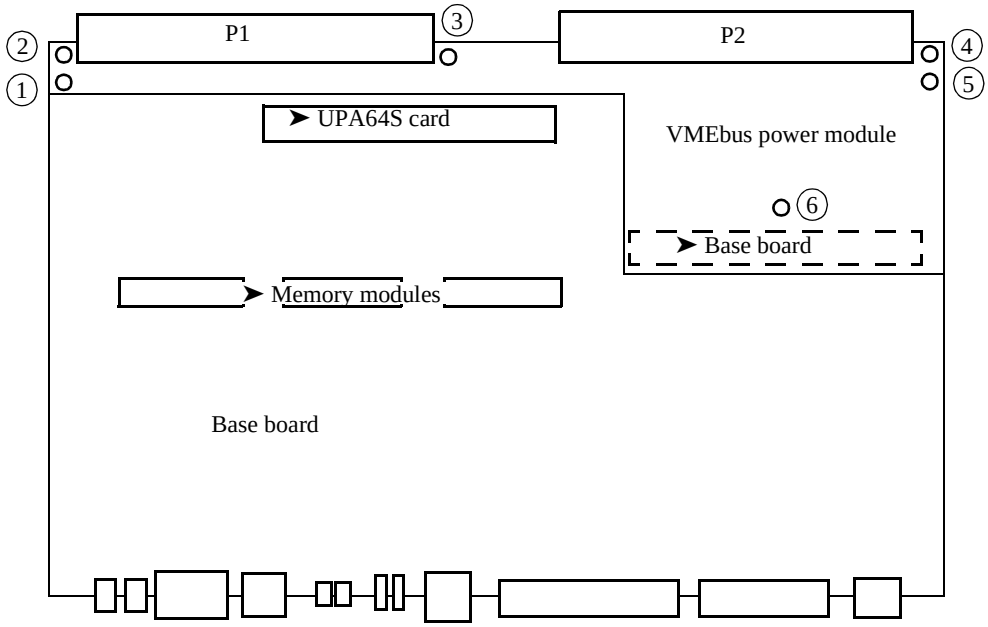
The VMEbus power module must be installed if no I/O-50(G)T is installed.

If you want to extend your Base-50(G) by an I/O-50(G)T you first have to remove the VMEbus power module.

Removing the VMEbus Power Module

To remove the VMEbus power module loosen the 6 screws shown in the figure below and lift the module.

Figure 10 Removing the VMEbus Power Module



3.3 Powering Up

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

VMEbus System Controller

If configured appropriately, the SPARC/CPU-50 recognizes automatically whether it is plugged in slot 1 and 2 of the VMEbus backplane or in any other slot. This auto-configuration feature requires that SW800-1 is set appropriately: OFF (default "OFF", see page 26). Via the auto-configuration the VMEbus system controller is enabled, when the SPARC/CPU-50 is plugged in slot 1, otherwise it is disabled.
If the auto-configuration is disabled, the VMEbus system controller has to be controlled manually by setting SW800-2, see section 3.4 "Switch Settings" on page 25.

Booting

The SPARC/CPU-50 boot PROM consists of a 1 MByte PROM (OTP) PLCC socket device (not writeable). Alternatively a 2 MByte TSOP boot flash EPROM device can be enabled by SW6-2. This boot flash EPROM device is writeable if enabled by SW4-3.

Note: **If an unformated floppy disk resides in a floppy drive connected to the SPARC/CPU-50 during powering up, the SPARC/CPU-50 does not boot and the OpenBoot does not appear. Therefore: Never boot the SPARC/CPU-50 with an unformated floppy disk residing in a floppy drive connected to the SPARC/CPU-50.**

By default the SPARC/CPU-50 is shipped with its boot PROM containing the OpenBoot firmware (see section 3.6 "OpenBoot Firmware" on page 41).

User Application

The SPARC/CPU-50 provides a user flash EPROM to store user applications. For write-protection of the user flash EPROM see SW4-4 in section 3.4 "Switch Settings" on page 25.

3.4 Switch Settings

The following table lists the functions and the default settings of all switches shown in figure 6 “Location Diagram of the Base Board (Schematic)” on page 18.

Note: **Before powering up the board check the current switch settings for consistency. Do not switch during operation.**

Table 12 Default Switch Settings

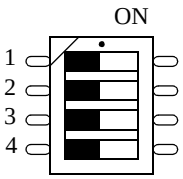
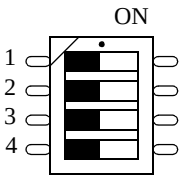
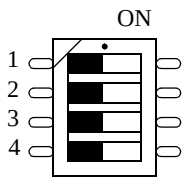
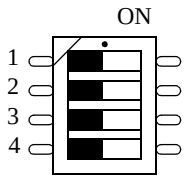
Name and default setting		Function
	SW4-1 OFF	Reset key on front-panel control OFF = RESET key enabled ON = RESET key disabled
	SW4-2 OFF	Abort key control OFF = ABORT key enabled ON = ABORT key disabled
	SW4-3 OFF	Boot flash EPROM write protection – only relevant if SW6-2 = ON OFF = boot flash EPROM write protected ON = boot flash EPROM write enabled
	SW4-4 OFF	User flash EPROM write protection OFF = user flash EPROM write protected ON = user flash EPROM write enabled
	SW5-1 OFF	SCSI Termination for SCSI #1 on front panel OFF = front panel termination automatic ON = front panel termination disabled
	SW5-2 OFF	SCSI Termination for SCSI #1 on P2 OFF = backplane termination disabled ON = backplane termination enabled
	SW5-3 OFF	Reserved, must be OFF
	SW5-4 OFF	Reserved, must be OFF

Table 12 Default Switch Settings (cont.)

Name and default setting		Function
	SW6-1 OFF	Reserved, must be OFF
	SW6-2 OFF	Boot device selection OFF = boot from boot PROM ON = boot from boot flash EPROM
	SW6-3 OFF	VMEbus SYSRESET on power-up OFF = enabled ON = disabled
	SW6-4 OFF	Watchdog enable switch OFF = disabled ON = enabled
	SW800-1 OFF	Automatic VMEbus slot-1 detection OFF = Automatic detection of VMEbus Slot 1 function ON = Automatic detection of VMEbus slot 1 function disabled. Set SW800-2 appropriately.
	SW800-2 OFF	Manual VMEbus slot-1 selection – only relevant if SW800-1 = ON OFF = VMEbus slot 1 function enabled ON = VMEbus slot 1 function disabled
	SW800-3 OFF	External VMEbus SYSRESET OFF = VMEbus SYSRESET generates on-board RESET ON = VMEbus SYSRESET does not generate on-board RESET
	SW800-4 OFF	VMEbus SYSRESET generation OFF = SYSRESET is driven to VMEbus ON = SYSRESET is not driven to VMEbus

3.5 Front Panel, Connectors, and Related Information

Front Panel Features

The features of the front panel are described in the following table. For a location diagram see figure 6 “Location Diagram of the Base Board (Schematic)” on page 18.

Table 13 Front-Panel Features

Device	Description
RESET	Mechanical reset key: When enabled and toggled it instantaneously affects the Base-50(G) by generating a push-button Power On Reset (POR) to the UltraSPARC-IIi. Push-button Power On Reset has the same effect as a Power On Reset from the power supply, with the only difference, that the corresponding status bit (`B_POR`) in the UltraSPARC-IIi `Reset_Control` Register is set and the DRAM refresh is not influenced. For information on disabling the reset key, see “SW4-1” on page 25.
ABORT	Mechanical abort key: When enabled and toggled it instantaneously affects the Base-50(G) by generating a push-button external initiated reset (XIR). Push-button external initiated reset allows a user-reset (abort) of part of the processor without resetting the whole system. UltraSPARC-IIi sets the `B_XIR` bit in the `Reset_Control` Register when a push-button external initiated reset is detected. For information on disabling the abort key, see “SW4-2” on page 25.
DIAG	Software programmable hexadecimal display for diagnostics.
MODE	Hexadecimal rotary switch, decoded with 4 bit. Default setting: F_{16}.
RUN	CPU status LED: green normal operation red the processor is halted or reset is active; it starts blinking to signal a hang-up of the SPARC/CPU-50.

Table 13 Front-Panel Features (cont.)

Device	Description
BM	VMEbus busmaster and SYSFAIL LED: green if the Base-50(G) accesses the VMEbus bus as master red if SYSFAIL is asserted from the Universe II to the VMEbus off otherwise
0, 1	2 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.
MIC	Standard 3.5 mm microphone jack
HDPH	Standard 3.5 mm headphone jack
ETHERNET	Standard Twisted-Pair-Ethernet RJ45 connector for 10BaseT/100BaseTX Ethernet.
SCSI	Standard SCSI 50-pin-fine-pitch connector
SERIAL A+B	26-pin shielded fine-pitch connector for 2 serial interfaces
KBD	Standard 8-pin-mini-DIN connector for keyboard and mouse

On-Board Connectors

In addition to the front-panel connectors, the Base-50(G) provides on-board connectors for memory modules and for the I/O-50(G)T. An overview of the on-board connectors is shown in the following table.

Table 14 On-board Connectors

Connector description and location	Connector type
VMEbus backplane connector P1	VG 96-pin connector male
VMEbus backplane connector P2	VG 160-pin connector male (in case of 3-row factory option VG 96-pin connector male)
I/O-50(G)T connector P6	100-pin MBus connector male

Table 14 On-board Connectors (cont.)

Connector description and location	Connector type
UPA64S interface connector P7	120-pin UPA connector female
Memory module connectors P8, P9, P10	80-pin SMD connector

The next table lists the available signals on the P2 backplane connector. For the P2 connector pinout see “VMEbus P2 Connector Pinout” on page 38.

Table 15 Available signals on P2 Backplane Connector

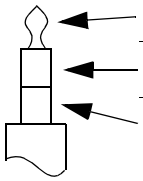
Interface	Backplane connector
Ultra-wide SCSI #1	P2 row A+C
MII Ethernet #1 interface	P2 row D
Floppy interface	P2 row C
Parallel interface	P2 row Z+D
Serial interface A	P2 row A
Serial interface B	P2 row C
Keyboard and mouse	P2 row A
Audio input: Stereo line in, Stereo auxiliary 1 in, Stereo auxiliary 2 in or microphone in (assembly option)	P2 row D
Audio output Stereo headphone out	P2 row D

3.5.1 Audio Interface

The two front panel audio interfaces use standard 3.5-mm-phono jacks supporting

- One single-ended condenser microphone
- One line level signal output, also designed to directly drive low impedance headphones

Table 16 Audio Interface Signals

Connector		Headphone	Microphone
	Tip	Left channel	
	Ring	Right channel	
	Shield	Analog GND	

For further information, see “Audio Interfaces” on page 10.

3.5.2 Ethernet #1 Interfaces, Ethernet Address, and Host ID

The full duplex Ethernet interface is available at the front panel via a 10BaseT/100BaseTx Twisted-Pair-Ethernet connector.

Table 17 Twisted-Pair-Ethernet Connector Pinout

Connector	Pin	Signal
RJ-45 TPE 	1	TX+
	2	TX-
	3	RX+
	4	GND
	5	GND
	6	RX-
	7	GND
	8	GND

The Ethernet #1 interface is also accessible at the 5-row P2 back panel connector via an MII interface. If Ethernet #1 is accessed via I/O panel,

the front-panel connector is disabled automatically. For the connector pinout see figure 14 “Pinout of Row Z and D of a 5-Row P2 Connector” on page 39.

Ethernet Address and Host ID

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

```
ok banner
```

The information below explains how the SPARC/CPU-50 Ethernet address and the host ID are determined.

Figure 11 The 48-bit (6-byte) Ethernet Address

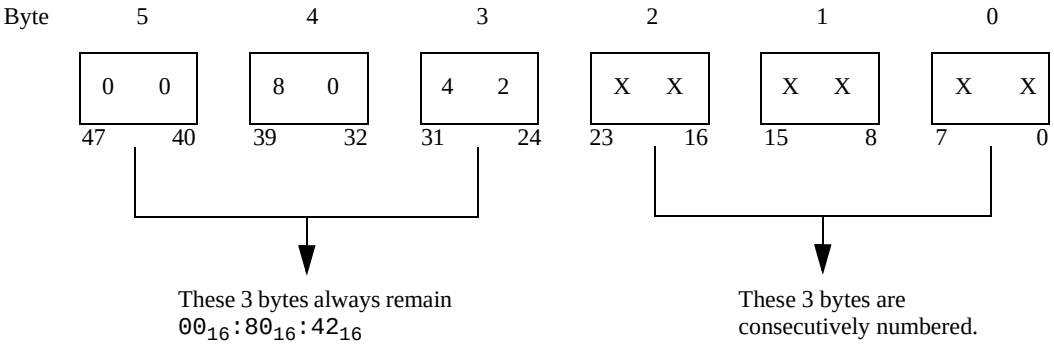
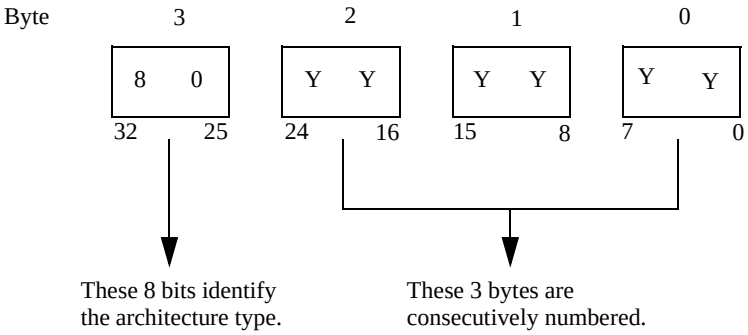


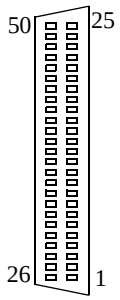
Figure 12 The 32-bit (4-byte) Host ID



3.5.3 SCSI #1 Connector Pinout

The front-panel SCSI #1 interface is single-ended and supports TERMPWR and 8-bit SCSI only. Automatic termination mode means the respective termination is disabled when you connect a standard SCSI cable to the front panel connector. For further information on available SCSI configurations, see section 3.5.4 “SCSI #1 Configuration” on page 33.

Table 18 50-pin SCSI Connector Pinout

Signal	Pin	Connector	Pin	Signal
GND	1		26	D0
GND	2		27	D1
GND	3		28	D2
GND	4		29	D3
GND	5		30	D4
GND	6		31	D5
GND	7		32	D6
GND	8		33	D7
GND	9		34	DP0
GND	10		35	GND
GND	11		36	AUTOTERM
n.c.	12		37	n.c.
n.c.	13		38	TERMPWR
n.c.	14		39	n.c.
GND	15		40	GND
GND	16		41	ATN
GND	17		42	GND
GND	18		43	BSY
GND	19		44	ACK
GND	20		45	RST
GND	21		46	MSG
GND	22		47	SEL
GND	23		48	CD
GND	24		49	REQ
GND	25		50	IO

3.5.4 SCSI #1 Configuration

Note: **Correct SCSI bus selection: The Base-50(G) provides 1 SCSI bus, SCSI #1. A further SCSI controller, SCSI #2 is available on-board the I/O-50(G)T providing access to the SCSI #2 bus which is totally independent from the SCSI #1 bus (see the I/O-50(G)T installation section).**

SCSI #1 Termination

The base board's SCSI #1 bus is accessible via the base board's front-panel SCSI #1 connector providing 8-bit SCSI and via the base board's P2 connector providing wide SCSI. Therefore, the base board holds 2 distinct SCSI bus terminations to enable correct termination of the SCSI #1 bus. Associated to the 2 terminations there are 2 switches – SW5-1 and SW5-2 – which allow easy selection of a valid SCSI #1 bus configuration.

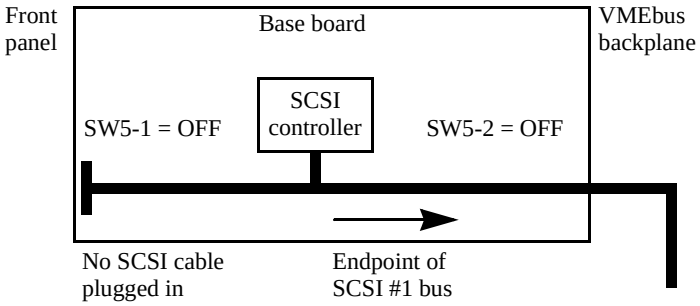
There are 4 valid base board switch settings corresponding to valid SCSI #1 bus configurations. The following factors differentiate the valid SCSI #1 bus configurations:

- 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?
- The SCSI device type being connected to the SCSI #1 bus: Is a Wide-SCSI device connected to the P2 connector?

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.

Default Configuration 1 for 8 bit SCSI

- 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:



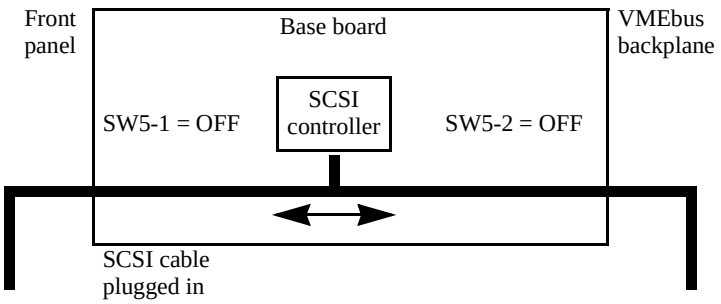
In this configuration (default switch setting):

– SW5-1 must be set to OFF = automatic enabling or disabling of termination by sensing whether a SCSI cable is plugged in

– and SW5-2 must be set to OFF = disabled.

Default Configuration 2 for 8 bit SCSI

- 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:



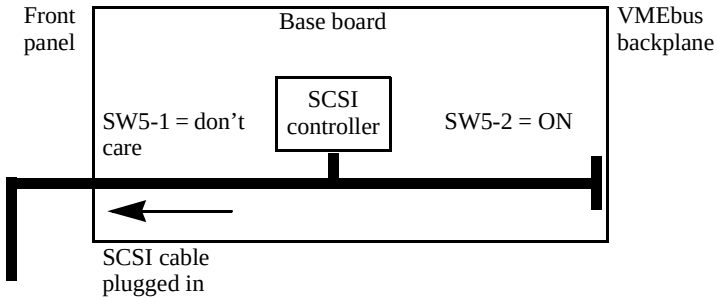
In this configuration (default switch setting):

– SW5-1 must be set to OFF = automatic enabling or disabling of termination by sensing whether a SCSI cable is plugged in

– and SW5-2 must be set to OFF = disabled.

Alternative Configuration for 8 bit SCSI

- Alternative configuration: 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

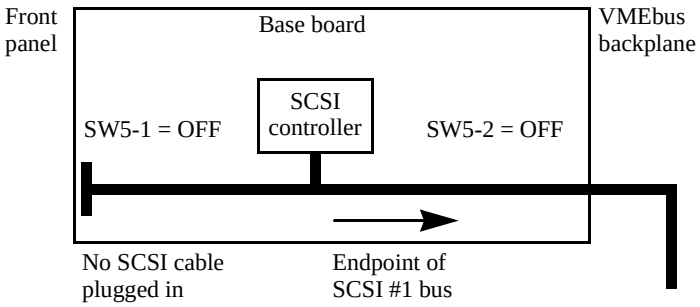
– both settings of SW5-1 are valid

– and SW5-2 must be set to ON = termination enabled.

Default Configuration for Wide-SCSI

Wide SCSI is only available on the P2 VMEbus connector.

- The Wide SCSI configuration 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 (default switch setting):

– SW5-1 must be set to OFF = automatic enabling or disabling of termination by sensing whether a SCSI cable is plugged in

– SW5-2 must be set to OFF = disabled.

3.5.5 Serial I/O Connector Pinout

Both serial I/O interfaces of the Base-50(G) are independent full-duplex channels. For each of them the 4 signals RXD, TXD, RTS, and CTS are also provided via the respective VMEbus P2 connector (for interfaces A and B see figure 13 “3-Row P2 Connector Pinout” on page 38).
SERIAL A+B on the Base-50(G)’s front panel holds the signals for the 2 serial interfaces A and B.

Table 19 26-Pin Serial A+B Connector Pinout RS232

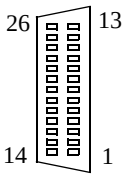
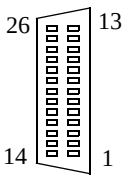
Signal	Pin	Connector	Pin	Signal
n.c.	1		14	TxD_B (Output)
TxD_A (Output)	2		15	RxC_A (Input)
RxD_A (Input)	3		16	RxD_B (Input)
RTS_A (Output)	4		17	RTxC_A (Input)
CTS_A (Input)	5		18	RxC_B (Input)
DSR_A (Input)	6		19	RTS_B (Output)
GND_A (Ground)	7		20	DTR_A (Output)
DCD_A (Input)	8		21	DSR_B (Input)
n.c.	9		22	RTxC_B (Input)
n.c.	10		23	GND_B (Ground)
DTR_B (Output)	11		24	TxC_A (Output)
DCD_B (Input)	12		25	TxC_B (Output)
CTS_B (Input)	13		26	n.c.

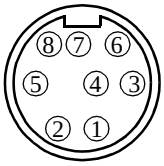
Table 20 26-Pin Serial A+B Connector Pinout RS422 (Factory Option)

Signal	Pin	Connector	Pin	Signal
n.c.	1		14	CTS+_B (Input))
CTS+_A (Input))	2		15	nc
RTS–_A (Output)	3		16	RTS–_B (Output)
RTS+_A (Output)	4		17	nc
CTS–_A (Input)	5		18	nc
nc	6		19	RTS+_B (Output)
RxD–_A (Input)	7		20	RxD+_A (Input)
TxD–_A (Output)	8		21	nc
n.c.	9		22	nc
n.c.	10		23	RxD–_B (Input)
RxD+_B (Input)	11		24	TxD+_A (Output)
TxD–_B (Output)	12		25	TxD+_B (Output)
CTS–_B (Input)	13		26	n.c.

3.5.6 Keyboard/Mouse Connector Pinout

The SUN-type keyboard/mouse interface is available at the front panel via an 8-pin mini-DIN connector.

Table 21 Keyboard/Mouse Connector Pinout

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

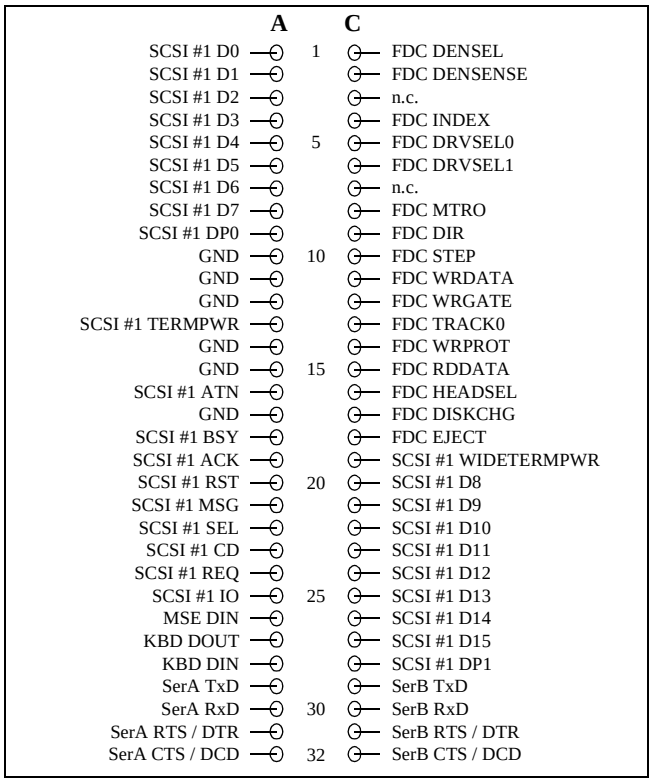
3.5.7 VMEbus P2 Connector Pinout

The pinout shown in the figure below applies to RS-232 configuration of the Base-50(G)'s serial I/O interfaces. Serial I/O interfaces configured for RS-422 (factory option) are only available via the front panel connector.
The names used in the following pinouts are given in brackets:
SCSI (`SCSI`), MII (`MII`), parallel (`LPT`), floppy (`FDC`), serial interface A (`SerA`), serial interface B (`SerB`), audio (`AUD`), keyboard (`KBD`), mouse (`MSE`), fused 5 V power for the I/O panel (`VP5`).

The standard Base-50(G) is delivered with 5-row P2 VMEbus connector. However, a 3-row P2 connector variant is also available as factory option.

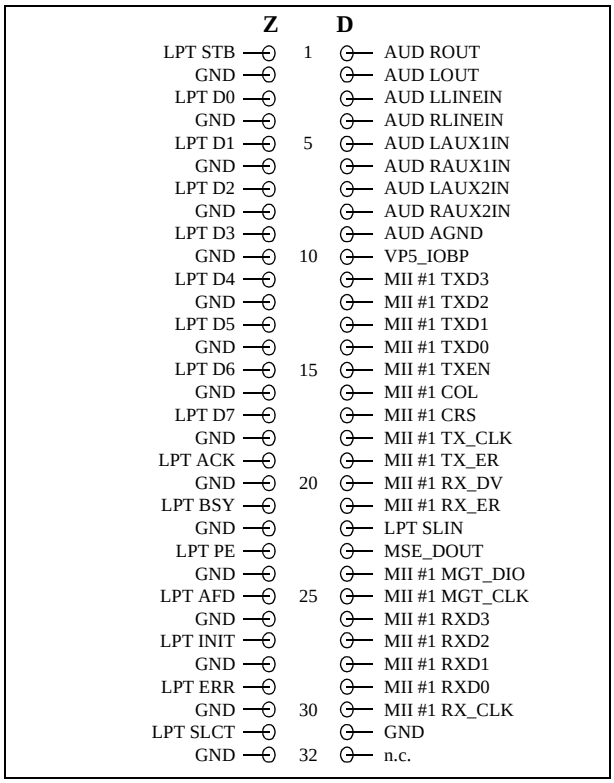
For further information, see "Backplane Configuration" on page 9 and "Slot-1 Function" on page 9.

Figure 13 3-Row P2 Connector Pinout



The signals for rows Z and D (Parallel interface, Ethernet MII interface, Audio) are shown in the following pinout for row D and Z.

Figure 14 Pinout of Row Z and D of a 5-Row P2 Connector



I/O Panel

As a separate price list item an I/O panel is available for the Base-50(G), the SPARC/IOBP-50/*x*. In case of *x*=3 a 3-row backplane connector is supported; in case of *x*=5 a 5-row connector. The corresponding SPARC/CPU-50/AccKit/*x* contains the following cables in addition to the I/O panel itself:

- A serial splitter cable for the front panel and the I/O panel
- A flat ribbon SCSI cable for the I/O panel
- AMicro D-Sub SCSI cable for the front panel
- ATwisted-Pair-Ethernet cable for the front panel

The I/O panel supports the following interfaces:

- SCSI #1
- MII Ethernet #1
- Serial A/B interface
- Audio interface

- Keyboard/mouse
- Parallel interface
- Floppy interface

Danger

The SPARC/IOBP-50/x is especially designed for the Base-50(G). Do not use any other I/O panels on the Base-50(G). In addition note the following:

- **Either use the front-panel or the I/O panel Ethernet interface, not both. Check the configuration of your I/O panel.**
- **SW5-2 on the Base-50(G) must be configured to disable the corresponding backplane SCSI termination. This is necessary because the I/O panel provides automatic termination.**

3.6 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

Note: **The examples in this section can differ from the appearance on your monitor according to your device tree (CPU architecture).**

For more information on the OpenBoot firmware see the *Open Boot 3.x Manual Set*.

The OpenBoot firmware is subject to changes. For newest version and how to upgrade refer to the SMART service accessible via the Force Computers World Wide Web site.

3.6.1 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 Open-Boot 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
```

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

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.

[*-bootoption*] Boot option may be one of the following:

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

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

`[-r]` reconfigure Solaris device drivers after changing hardware configuration.

`[-v]` print verbose information during boot procedure.

Devices to Boot from

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

```
ok boot 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 22 Device Alias Definitions

Alias	Description
	Defined for SCSI
`scsi`	SCSI
`disk`	Default disk SCSI-target-ID 0
`disk6`	disk SCSI-target-ID 6
`disk5`	disk SCSI-target-ID 5
`disk4`	disk SCSI-target-ID 4

Table 22 Device Alias Definitions (cont.)

Alias	Description
disk3	disk SCSI-target-ID 3
disk2	disk SCSI-target-ID 2
disk1	disk SCSI-target-ID 1
disk0	disk SCSI-target-ID 0
tape (or tape0)	1st tape drive SCSI-target-ID 4
tape1	2nd tape drive SCSI-target-ID 5
cdrom	CD-ROM partition f, SCSI-target-ID 6
net	Defined for Ethernet Ethernet
floppy	Floppy disk
vme	VME
audio	Audio
keyboard	Keyboard
mouse	Mouse
ebus	EBus2
pci	primary PCI bus
flash-prog	Flash EPROM programming mode
flash	Flash EPROM
ttya	Serial interface A
ttyb	Serial interface B

3.6.2 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: **Per default the SPARC/CPU-50 boots the OS automatically. If not, ensure that the `auto-boot?` parameter is always set to `true`.**

To Set Parameters

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

The configuration parameters in the following table are involved with the boot process.

Table 23 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

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.

3.6.3 Diagnostics

At Hardware Power On or Button Power On the OpenBoot firmware executes POST. The extent of certain tests executed within the POST depend on the state of the configuration parameter `diag-level.` The operator can choose between minimal or maximal testing by setting this configuration parameter to `min` or `max`. Furthermore an enhanced diagnostic menu is available if setting this parameter to `menu`. 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 interface 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, keyboard and audio. 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 24 Diagnostic Routines

Command	Description
`probe-scsi`	Identifies devices connected to the primary SCSI 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.
`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-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 primary Ethernet.

Examples:

SCSI Bus

To check the SCSI #1 for connected devices enter:

```
ok probe-scsi
Target 3
Unit 0 Disk FUJITSU M2952ESP SUN2.1G2545
ok
```

All SCSI Buses

To check 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
/pci@1f,0/scsi@2

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

/pci@1f/pci@4,1/scsi@2

Target 3
Unit 0 Disk FUJITSU M2952ESP SUN2.1G2545
ok
```

Note: **The command `probe-scsi-all` can last up to 2 minutes without terminal message.**

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 22, “Device Alias Definitions,” on page 42. 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.

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
Internal loopback test -- succeeded.
Transceiver check -- Using Onboard transceiver -- Link Up.
passed
Using Onboard transceiver -- Link Up.
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.

3.6.4 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 25 Commands to Display System Information

Command	Description
`banner`	Displays system banner
`show-pci-devs-all`	Displays list of installed and probed PCI Bus devices
`.enet-addr`	Displays the 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

3.6.5 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 initialization procedures and executes the POST (not available up to now) if having pressed the reset button. Then the system either boots automatically or enters the Forth Monitor, just as it would have done after a power-on cycle.

3.6.6 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:
Numeric output
Radix (number base conversions)
Arithmetik
Memory access
Line editor
System and boot configuration parameters
Select I/O devices
Floppy eject
Power on reset
Diag (diagnostic routines)
Resume execution
File download and boot
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 power
reset-all         reset-machine, (simulates power cycling )
power-off         Power Off
ok
```

```
ok help memory
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
x? ( addr -- ) display the 64-bit number from location addr
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
x@ ( addr -- n ) place on the stack the 64-bit data at 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
x! ( n addr -- ) store the 64-bit value n 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
```

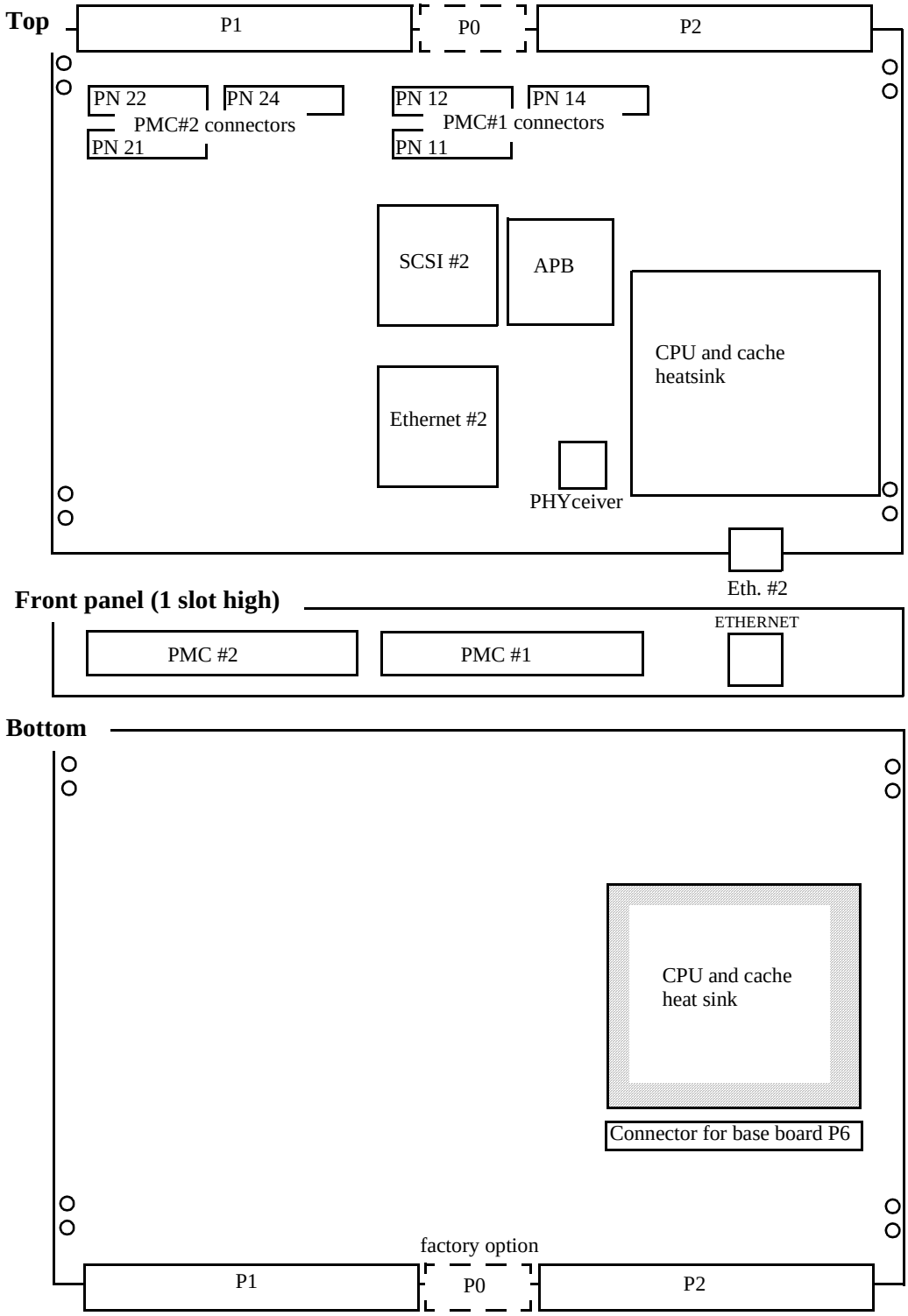
4 I/O-50(G)T Installation

This section extends **section 2 “Installation” on page 5** by providing the installation information for the I/O-50(G)T in its various configurations (see section 4.2 “I/O-50(G)T Mechanical Construction” on page 53).

Note: **Note that there is no need to refer to the installation subsections within section 4.2 “I/O-50(G)T Mechanical Construction” on page 53 unless you want to change the delivered configuration to replace a UPA card or the like.**

4.1 Location Overview

Figure 15 Location Diagram of the I/O-50(G)T (Schematic)



4.2 I/O-50(G)T Mechanical Construction

The I/O-50(G)T is an extension to the Base-50(G). It occupies 1 VMEbus slot and consists of the following major components:

- Two PMC slots
- One SCSI #2 interface
- One Ethernet #2 interface

The following figures show the SPARC/CPU-50 in 2-slot and 3-slot configuration with an I/O-50(G)T.

Figure 16 Mechanics of a SPARC/CPU-50T

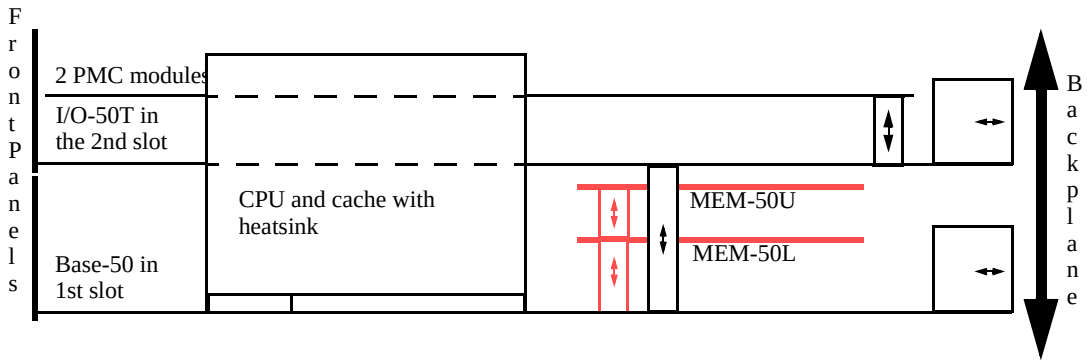


Figure 17 Mechanics of a SPARC/CPU-50GT with UPA64S Card

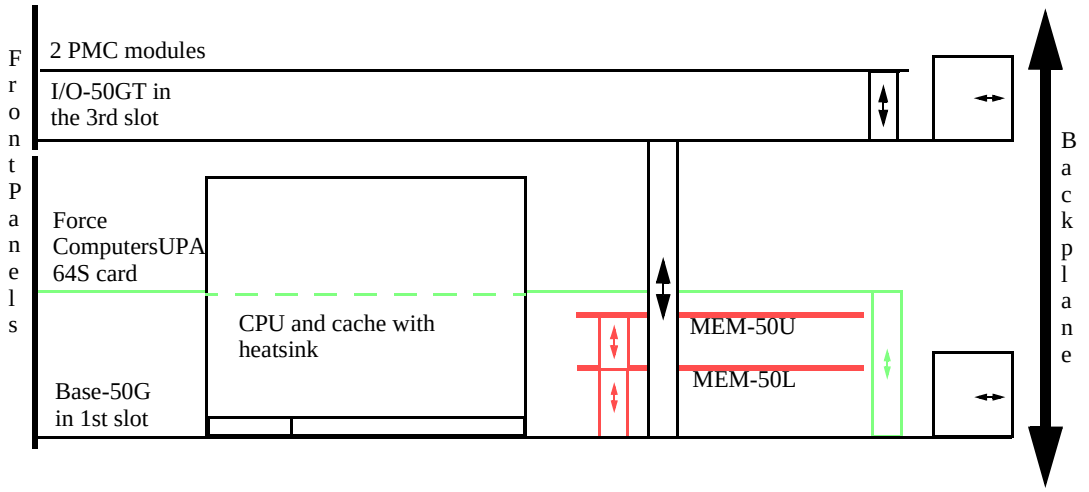
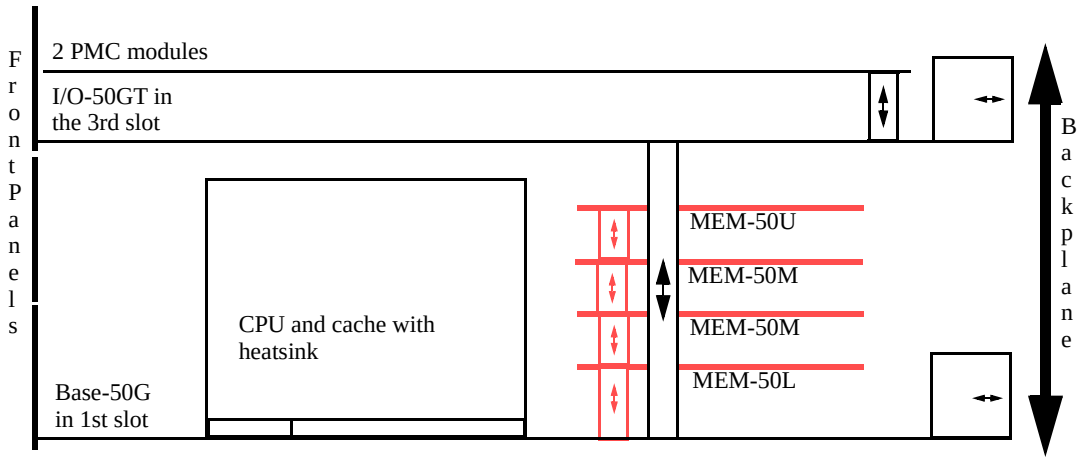


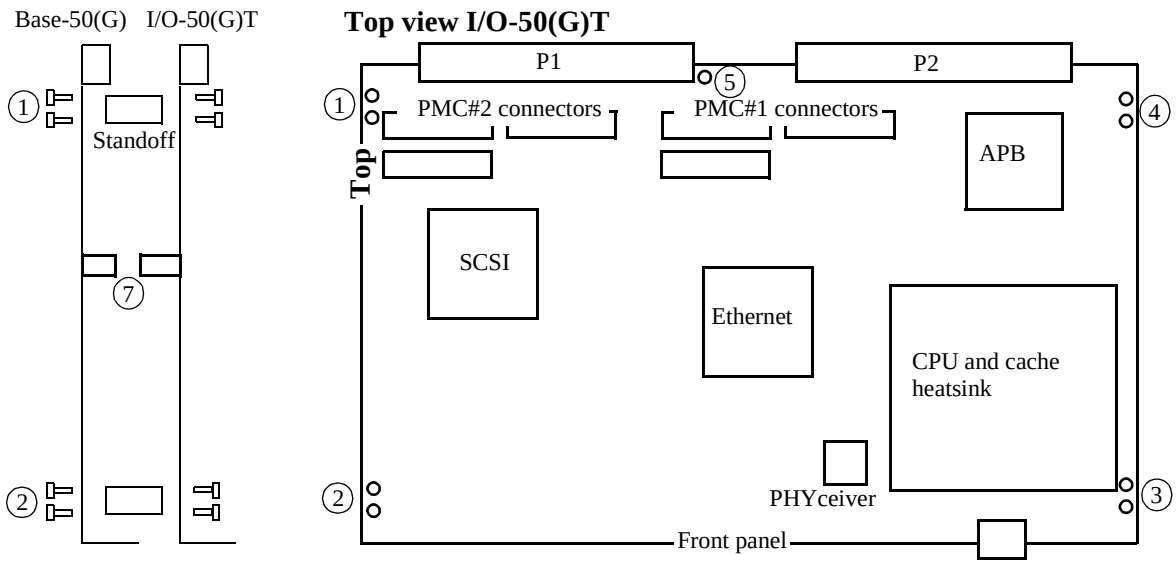
Figure 18 Mechanics of a SPARC/CPU-50GT and four Memory Modules



4.2.1 Installing and Uninstalling the I/O-50(G)T

This section describes how to install or uninstall the I/O-50(G)T. Refer to the figure below to locate the locations mentioned in the description.

Figure 19 Uninstalling the I/O-50(G)T



Uninstalling the I/O-50(G)T

To uninstall the I/O-50(G)T follow the steps below:

1. Remove the 9 screws at location 1…5 on the I/O-50(G)T.
2. Remove the I/O-50(G)T from the Base-50(G) by lifting it.
3. Fix the removed 9 screws on the open ends of the standoffs to have them available when installing the I/O-50(G)T again.

Installing the I/O-50(G)T

To install the I/O-50(G)T follow the steps below:

1. Remove the 9 screws at location 1…5 on the open end of the stand-offs of the Base-50(G).
2. Plug the I/O-50(G)T to the Base-50(G) via the I/O-50(G)T to Base-50(G) connector at position 7.
3. Fix it with the 9 removed screws on the standoffs at location 1…5.

4.3 Front Panel, Connectors, and Related Information

Front Panel Features

The features of the front panel are described in the following table. For a location diagram see figure 15 “Location Diagram of the I/O-50(G)T (Schematic)” on page 52.

Table 26 Front-Panel Features

Device	Description
ETHERNET	Standard Twisted-Pair-Ethernet RJ45 connector for 10BaseT/100BaseTX Ethernet
PMC #1	Hole for the PMC #1 front panel
PMC #2	Hole for the PMC #2 front panel

On-Board Connectors

In addition to the front-panel connectors, the I/O-50(G)T provides on-board connectors for connection to the Base-50(G), to the VMEbus and for 2 PMC modules. An overview is shown in the following table.

Table 27 On-board Connectors

Connector	Description	Connector type
VMEbus P1	–	VG 96-pin male connector
VMEbus P2	SCSI #2 and Ethernet MII #2	VG 160-pin connector male (in case of 3-row factory option VG 96-pin connector male)
VMEbus P0 (factory option)	PMC #1, #2 user I/O	95-pin female connector
P6	I/O-50(G)T extension connector	100-pin MBUS connector female • low: for 2 slot solution SPARC/CPU-50T • high: for 3 slot solution SPARC/CPU-50GT

Table 27 On-board Connectors (cont.)

Connector	Description	Connector type
PN11, PN12, PN14	PMC #1	64-pin SMD connector
PN21, PN22, PN24	PMC #2	64-pin SMD connector

4.3.1 Ethernet #2 Interfaces and Configuration

The full duplex 10BaseT/100BaseTx Ethernet #2 interface is available at the front panel via a Twisted-Pair-Ethernet connector.

Table 28 Twisted-Pair-Ethernet Connector Pinout

Connector	Pin	Signal
RJ-45 TPE 	1	TX+
	2	TX-
	3	RX+
	4	GND
	5	GND
	6	RX-
	7	GND
	8	GND

Ethernet #2 Configuration

The Ethernet #2 interface is also accessible at the 5-row P2 backplane connector via an MII interface. If Ethernet #2 is accessed via the I/O panel, the front panel connector is normally disabled automatically. For other configurations see the respective jumper settings in the *SPARC/IOBP-50 Installation Guide*. For the connector pinout see figure 16 “Pinout of row Z and D of a 5-row P2 connector” on page 52.

Note: **Correct Ethernet selection: The I/O-50(G)T provides 2 Ethernet #2 interfaces: via a TPE #2 interface connected to a front-panel RJ-45 connector or an MII #2 interface available at the VMEbus P2 connector.**

Ethernet address and host ID

For the SPARC/CPU-50 only 1 Ethernet address and host ID exists, see section 3.5.2 “Ethernet #1 Interfaces, Ethernet Address, and Host ID” on

page 30. Use the Ethernet #2 TPE or MII interface of the I/O-50(G)T only in a separate network which is unrelated to the Ethernet #1 network of the Base-50(G).

4.3.2 PMC Slots

The I/O-50(G)T provides 2 PMC slots compliant with IEEE P1386 ("Draft Standard Physical and Environmental Layers for PCI Mezzanine Cards: PMC"). The PCI bus, a high speed local bus compliant with Rev. 2.1., connects different high speed I/O cards with the SPARC/CPU-50. Both PMC slots support 32-bit data bus width with a maximum frequency of 33 MHz.

PMC Voltage Keys

The PCI bus uses a 5V voltage to signal bus levels. The voltage keys prevent 3.3V PMC cards from being plugged into the PMC slots.

Connectors and P0 Factory Option

The 32-bit PCI bus requires 2 PMC connectors. The 3rd PMC connector connects additional user I/O signals of PMC slot 1 and PMC slot 2 to the VMEbus P0 connector (factory option).

– PMC Slot 1

- For the PCI bus: PN11 and PN12
- For 64 user I/O signals: PN14

– PMC Slot 2

- For the PCI bus: PN21 and PN22
- For 32 user I/O signals: PN24

PMC 1/2 Factory Option

As an additional factory option the PMC slot 1 signals can be connected to the PMC slot 2 signals in the following way:

- PMC 1 I/O 1 is connected to PMC 2 I/O 33
- PMC 1 I/O 2 is connected to PMC 2 I/O 34
- …
- PMC 1 I/O 32 is connected to PMC 2 I/O 64

4.3.3 SCSI #2 Configuration

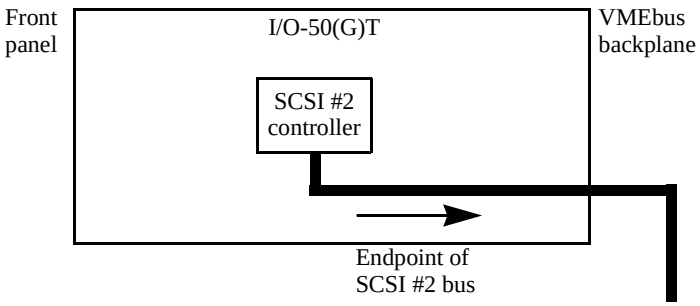
The SCSI #2 bus is only available at the VMEbus P2 connector (see section 4.3.4 “VMEbus P2 Connector Pinout” on page 59).

Note: **Correct SCSI bus selection (see section 3.5.4 “SCSI #1 Configuration” on page 33): The I/O-50(G)T provides a second SCSI bus, SCSI #2. The SCSI #2 bus is always terminated at the SCSI #2 controller.**

Valid Configuration

There is only 1 valid I/O-50(G)T SCSI #2 bus configuration:

- The I/O-50(G)T is located at an endpoint of the SCSI #2 bus, the SCSI #2 bus is extended via the VMEbus P2 connector:

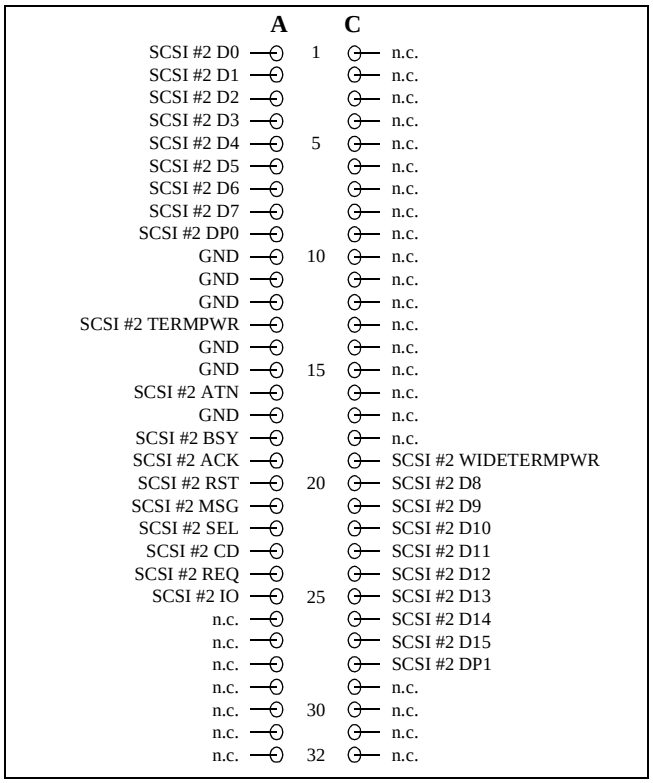


4.3.4 VMEbus P2 Connector Pinout

The standard I/O-50(G)T is delivered with 5-row P2 VMEbus connector. However, a 3-row P2 connector variant is also available as factory option.

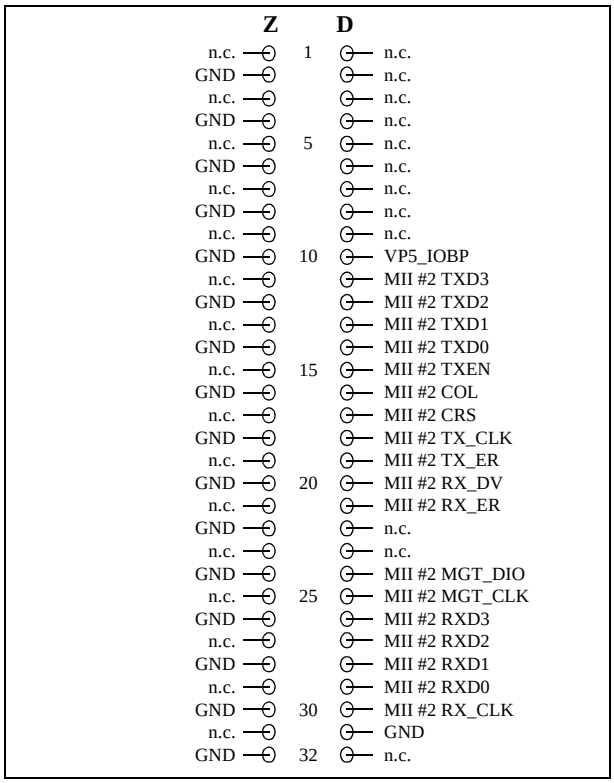
For further information, see “Backplane Configuration” on page 9 and “Slot-1 Function” on page 9.

Figure 20 3-row P2 Connector Pinout



The signals for rows Z and D (Ethernet MII interface) are shown in the following pinout for row Z and D.

Figure 21 **Pinout of Row Z and D of a 5-row P2 Connector**



I/O Panel

As a separate price list item an I/O panel is available for the I/O-50(G)T, the SPARC/IOBP-50/x (x=3 3-row backplane connector; x=5 5-row connector). An extended variant is the SPARC/CPU-50/AccKit/x which contains additionally to the I/O panel the following cables:

- A flat ribbon SCSI cable for the I/O panel
- A Twisted-Pair-Ethernet cable for the front panel.

The I/O panel supports the following interfaces:

- SCSI #2,
- MII #2 Ethernet.

Danger

The SPARC/IOBP-50/x is especially designed for the I/O-50(G)T. Do not use any other I/O panels on the I/O-50(G)T. In addition note:

- **Either use the front-panel or the I/O panel Ethernet interface, not both. Check the configuration of your I/O panel.**

4.3.5 VMEbus P0 Connector Pinout (Factory Option)

Note: **Note that in case of the PMC 1/2 Factory Option, which is described on page 57, half of the PMC 1 user I/O signals is connected to certain PMC 2 user I/O signals.**

Figure 22 **Factory Option P0 (5-row Female 95-pin Metric Connector)**

A	B	C		D	E
PMC 2 I/O 01	PMC 2 I/O 02	PMC 2 I/O 03	19	PMC 2 I/O 04	PMC 2 I/O 05
PMC 2 I/O 06	PMC 2 I/O 07	PMC 2 I/O 08	18	PMC 2 I/O 09	PMC 2 I/O 10
PMC 2 I/O 11	PMC 2 I/O 12	PMC 2 I/O 13	17	PMC 2 I/O 14	PMC 2 I/O 15
PMC 2 I/O 16	PMC 2 I/O 17	PMC 2 I/O 18	16	PMC 2 I/O 19	PMC 2 I/O 20
PMC 2 I/O 21	PMC 2 I/O 22	PMC 2 I/O 23	15	PMC 2 I/O 24	PMC 2 I/O 25
PMC 2 I/O 26	PMC 2 I/O 27	PMC 2 I/O 28	14	PMC 2 I/O 29	PMC 2 I/O 30
PMC 1 I/O 01	PMC 1 I/O 02	PMC 1 I/O 03	13	PMC 1 I/O 04	PMC 1 I/O 05
PMC 1 I/O 06	PMC 1 I/O 07	PMC 1 I/O 08	12	PMC 1 I/O 09	PMC 1 I/O 10
PMC 1 I/O 11	PMC 1 I/O 12	PMC 1 I/O 13	11	PMC 1 I/O 14	PMC 1 I/O 15
PMC 1 I/O 16	PMC 1 I/O 17	PMC 1 I/O 18	10	PMC 1 I/O 19	PMC 1 I/O 20
PMC 1 I/O 21	PMC 1 I/O 22	PMC 1 I/O 23	9	PMC 1 I/O 24	PMC 1 I/O 25
PMC 1 I/O 26	PMC 1 I/O 27	PMC 1 I/O 28	8	PMC 1 I/O 29	PMC 1 I/O 30
PMC 1 I/O 31	PMC 1 I/O 32	PMC 1 I/O 33	7	PMC 1 I/O 34	PMC 1 I/O 35
PMC 1 I/O 36	PMC 1 I/O 37	PMC 1 I/O 38	6	PMC 1 I/O 39	PMC 1 I/O 40
PMC 1 I/O 41	PMC 1 I/O 42	PMC 1 I/O 43	5	PMC 1 I/O 44	PMC 1 I/O 45
PMC 1 I/O 46	PMC 1 I/O 47	PMC 1 I/O 48	4	PMC 1 I/O 49	PMC 1 I/O 50
PMC 1 I/O 51	PMC 1 I/O 52	PMC 1 I/O 53	3	PMC 1 I/O 54	PMC 1 I/O 55
PMC 1 I/O 56	PMC 1 I/O 57	PMC 1 I/O 58	2	PMC 1 I/O 59	PMC 1 I/O 60
PMC 1 I/O 61	PMC 1 I/O 62	PMC 1 I/O 63	1	PMC 1 I/O 64	VP5_IOBP

4.4 OpenBoot Firmware Alias Definitions for I/O-50(G)T

The following table lists alias definitions related to the I/O-50(G)T.

Table 29 Device Alias Definitions

Alias	Description
	Defined for SCSI #2:
scsi-2	SCSI #2
disk26	disk SCSI #2-target-ID 6
disk25	disk SCSI #2-target-ID 5
disk24	disk SCSI #2-target-ID 4
disk23	disk SCSI #2-target-ID 3
disk22	disk SCSI #2-target-ID 2
disk21	disk SCSI #2-target-ID 1
disk20	disk SCSI #2-target-ID 0
tape2 (or tape20)	1st tape drive SCSI #2-target-ID 4
tape21	2nd tape drive SCSI #2-target-ID 5
cdrom2	CD-ROM partition f, SCSI #2-target-ID 6
	Defined for Ethernet #2:
net2	Ethernet #2
pcib-io	secondary PCI Bus B

5 Hardware Description

The SPARC/CPU-50 is based on:

- The UltraSPARC-IIi processor supporting 3 high-speed interfaces concurrently operating:
  - The memory interface with ECC
  - The external (L2) cache interface
  - The PCI interface
- The Universe IIb realising the VMEbus interface to the processor's PCI bus providing fast VMEbus access with maximized address space

The base board carries the components of the UltraSPARC-IIi chip set:

- XCVR data multiplexers
- UIC (UPA Interrupt Concentrator)
- PCIO (PCI IO) chip which interfaces to the local I/O bus

Described Features of the Base-50(G)

Besides the Processor – UltraSPARC-IIi – and the VMEbus Interface – Universe IIb – the Base-50(G) provides the following components on the 2 on-board buses – the PCI bus and the EBus2:

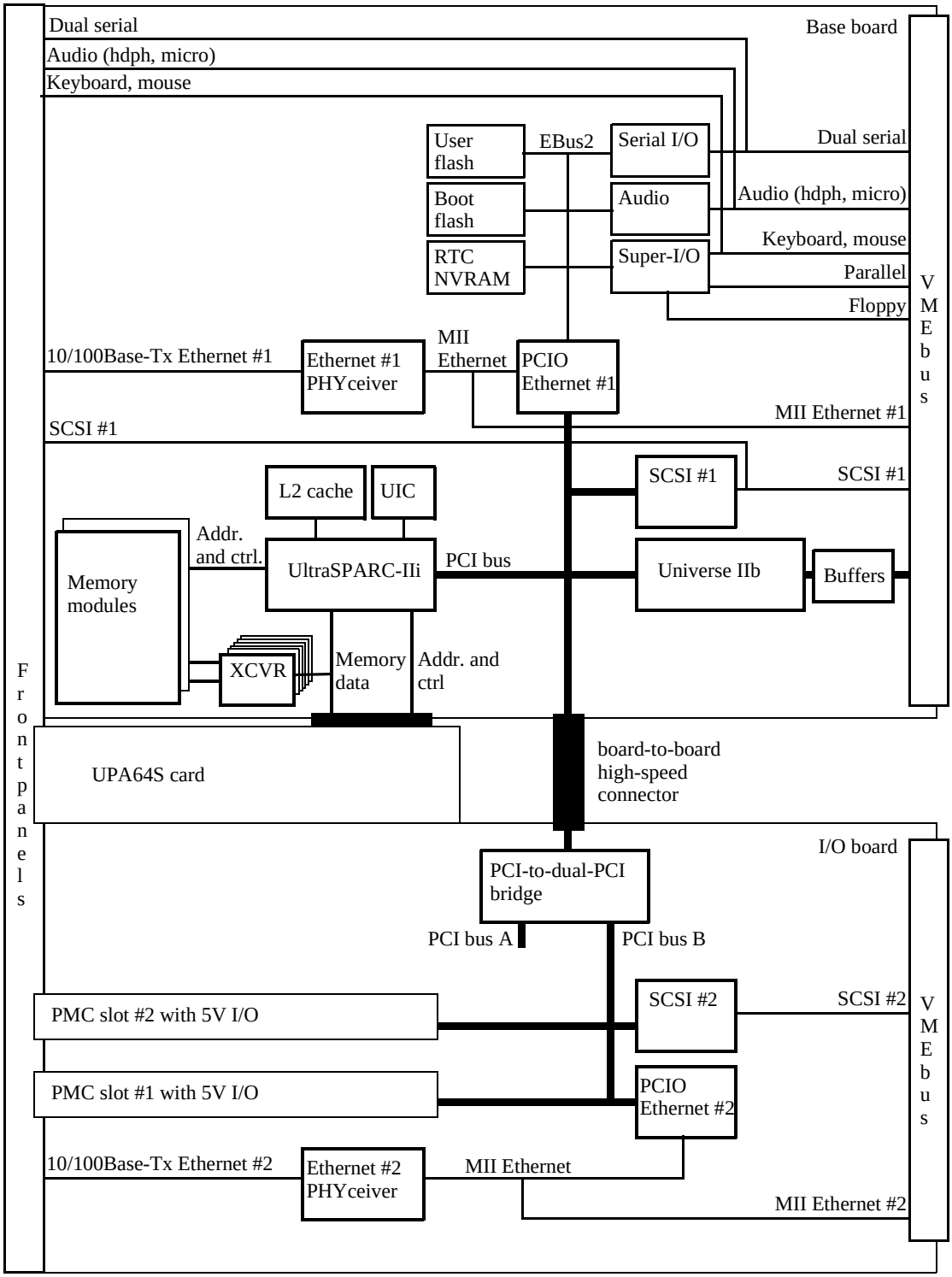
- Ethernet interface (Twisted-Pair and MII) Ethernet #1
- Ultra-wide SCSI interface SCSI #1
- Boot PROM, boot flash EPROM, and user flash EPROM
- Two Serial Interfaces – SAB 82532
- Keyboard/Mouse, FDC and Parallel Interface – Super I/O
- RTC/NVRAM – M48T58
- 16-bit stereo audio interface
- System configuration registers
- 7- segments LED, rotary-switch, system and user LEDs

Described Features of the I/O-50(G)T

The PCI-to-Dual-PCI Bridge – APB on the I/O-50(G)T drives 2 secondary PCI buses A and B. PCI bus A is not used and reserved. PCI bus B connects to

- Two PMC Slots with Busmode Support
- The second SCSI interface SCSI #2
- A second Ethernet interface Ethernet #2

Figure 23 Block Diagram CPU-50GT



The following table gives an overview of the buses, their bus modes, and the connected devices.

Table 30 Buses, Bus Modes, and Connected Devices

Bus, bus mode	Connected devices
EC bus, big endian	• Processor – UltraSPARC-IIi (see page 67) • L2 cache with tag (see page 69)
Memory bus, big endian	• Processor – UltraSPARC-IIi (see page 67) • Memory banks (see page 69)
UPA bus, big endian	• Processor – UltraSPARC-IIi (see page 67) • UPA64S card
PCI bus, little endian	• Processor – UltraSPARC-IIi (see page 67) • PCIO (PCI I/O Controller) – STP2003QFP providing Ethernet #1 (see page 82) and the EBus2 interface (see page 83) • PCI-Ultra SCSI (Fast-20) I/O Interface – SYM53C875 providing SCSI #1 (see page 102) • VMEbus Interface – Universe II (see page 74) • APB on I/O-50(G)T (see page 103)
VMEbus, big endian	• VMEbus Interface – Universe II (see page 74)

Table 30 Buses, Bus Modes, and Connected Devices (cont.)

Bus, bus mode	Connected devices
EBus2, little endian	• Boot PROM, Boot and User Flash EPROM (see page 85) • Enhanced Serial Communication Controller – SAB 82532 with serial interfaces (see page 87) • Super I/O – PC87332VLJ with parallel, floppy, and keyboard/mouse interface (see page 87) • Audio Controller – CS4231A with audio interface (see page 90) • Real-time Clock and NVRAM – RTC/NVRAM M48T58 (see page 89) • System configuration registers (see page 91)
PCI bus B on I/O-50(G)T, little endian	• APB (Advanced PCI bus Bridge) – SME2411 (see page 103) • Ethernet #2 via PCIO (PCI I/O Controller) – STP2003QFP (see page 82) • PCI-Ultra SCSI (Fast-20) I/O Interface – SYM53C875 providing SCSI #2 (see page 102) • 2 PMC modules (see page 103)

5.1 Processor – UltraSPARC-IIi

UltraSPARC-IIi is a highly integrated 64-bit SPARC V9 superscalar processor. The interfaces have been optimized to typical uniprocessor system requirements.

Features

- Binary compatible with all SPARC application codes
- VIS instruction set
- 4-way SuperScalar design with 9 execution units (SPARC V9)
  - four integer execution units
  - three floating-point execution units
  - two graphics execution units
- Directly addresses little- or big-endian data
- 64-bit address pointers
- 16-KByte non-blocking data cache
- 16-KByte instruction cache
- Integrated L2 cache controller
- Integrated control of 400-MByte/s EDO DRAM memory subsystem
- 64-Byte block load and block store instructions
- Supports software data prefetch into L2 cache
- Supports up to 3 outstanding L2 cache misses
- Supports UPA64S interface
- High sustained PIO and DMA PCI I/O bandwidth
- Read prefetch and write gathering and posting
- PCI DMA is cache coherent
- Dedicated TLB provides mapping and protection

5.1.1 Physical Memory Map

Table 31 UltraSPARC-IIi Physical Address Map (41-bit Physical Addresses)

Address range in PA<40:0>	Size	Addressed interface	Access type
$000.0000.0000_{16}$ …$000.3FFF.FFFF_{16}$	1 GByte	Main memory	cacheable
$000.4000.0000_{16}$ …$1FF.FFFF.FFFF_{16}$	reserved	undefined	cacheable
$000.0000.0000_{16}$ …$1FB.FFFF.FFFF_{16}$	reserved	undefined	noncacheable
$1FC.0000.0000_{16}$ …$1FD.FFFF.FFFF_{16}$	8 GByte	UPA64S	noncacheable
$1FE.0000.0000_{16}$ …$1FF.FFFF.FFFF_{16}$	8 GByte	PCI	noncacheable

Table 32 UltraSPARC-IIi Internal CSR space (16 MByte)

Address range in PA<40:0>	Size	Description/owner
$1FE.0000.0000_{16}$ …$1FE.0000.01FF_{16}$	512 Byte	PBM (PCI bus module)
$1FE.0000.0200_{16}$ …$1FE.0000.03FF_{16}$	512 Byte	IOM (IO memory management unit)
$1FE.0000.0400_{16}$ …$1FE.0000.1FFF_{16}$	7 KByte	PIE (PCI interrupt)
$1FE.0000.2000_{16}$ …$1FE.0000.5FFF_{16}$	16 KByte	PBM
$1FE.0000.6000_{16}$ …$1FE.0000.9FFF_{16}$	12 KByte	PIE
$1FE.0000.A000_{16}$ …$1FE.0000.A7FF_{16}$	2 KByte	IOM
$1FE.0000.A800_{16}$ …$1FE.0000.EFFF_{16}$	22 KByte	PIE
$1FE.0000.F000_{16}$ …$1FE.00FF.F018_{16}$	23 MByte	MCU (memory control unit)
$1FE.00FF.F020_{16}$	8 Byte	PIE
$1FE.00FF.F028_{16}$ …$1FE.00FF.FFFF_{16}$	4 KByte	MCU

5.1.2 External Cache Control Unit

The UltraSPARC-IIi obtains an integrated L2 cache controller providing a backside interface with a 72-bit wide data bus and 167 MHz speed. The L2 cache capacity is either 256 KByte or 1 MByte. 3 synchronous late write SRAM devices (2 for data and 1 for the tags) are provided.

5.1.3 Memory Controller Unit, Memory Modules, and Main Memory Configuration

Memory Controller Unit

The memory controller unit of the SPARC/CPU-50 is included in the UltraSPARC-IIi. The memory interface provides full EDO DRAM support including refresh. It uses 8 RAS lines to select 8 DRAM banks and 2 identical CAS signals. The memory interface is 72 bit wide, 64 bits are shared with the UPA64S interface and 8 bits are used for ECC. 6 bidirectional registered multiplexers and demultiplexers (XCVRs) are used to extend the memory interface from 72 bit to 144 bit. The control of the XCVRs is also included in the UltraSPARC-IIi. 60 ns EDO DRAMs with 10-bit column address (CAS) are supported.

The CPU supports the following accesses to main memory:

- Refresh: Refresh is a 4-way staggered CAS before RAS refresh. One refresh at a time refreshes the 2 memory banks of 1 memory module.
- 64-byte read to fill one L2 cache line: containing 1 burst access (EDO fast page mode) with 4 data (CAS) cycles each 128 bit wide (16 byte).
- 64-byte write to flush one L2 cache line: containing 1 burst access (EDO fast page mode) with 4 data (CAS) cycles each 128 bit wide (16 byte). To write data words smaller than 64 byte:
  - Fill one cache line,
  - Modify this cache line
  - Write it back

Memory Modules

The main memory capacity is adjustable via installation of the appropriate memory modules. For naming conventions, see figure 5 “MEM-50 – Memory Module Numbering Scheme” on page 12.

Table 33 Relating Memory Capacity to Device Type and Number of Banks

Memory module	Memory capacity	Device type	No. of banks
MEM-50L	32 MByte	2M*8	1
	64 MByte		2
	128 MByte	8M*8	1
	256 MByte		2
MEM-50M or -50U	128 MByte		1
	256 MByte		2

Up to a total of 8 memory banks are possible. Every bank has a fixed physical starting address and an end address depending on the bank size.

Table 34 Physical Memory Addresses for Memory Modules

Physical address range	Module number	Bank	Size in MByte	Memory module MEM-…
$000.0000.0000_{16}$- $000.01FF.FFFF_{16}$	1	1	32	…50L/32 or …50L/64
$000.2000.0000_{16}$- $000.21FF.FFFF_{16}$		2	32	…50L/64
$000.0000.0000_{16}$- $000.07FF.FFFF_{16}$		1	128	…50L/128 or …50L/256
$000.2000.0000_{16}$- $000.27FF.FFFF_{16}$		2	128	…50L/256
$000.0800.0000_{16}$- $000.0FFF.FFFF_{16}$	2	1	128	…50M or U/128 or /256
$000.2800.0000_{16}$- $000.2FFF.FFFF_{16}$		2	128	…50M or U/256
$000.1000.0000_{16}$- $000.17FF.FFFF_{16}$	3	1	128	…50M or U/128 or /256
$000.3000.0000_{16}$- $000.37FF.FFFF_{16}$		2	128	…50M or U/256
$000.1800.0000_{16}$- $000.1FFF.FFFF_{16}$	4	1	128	…50M or U/128 or /256
$000.3800.0000_{16}$- $000.3FFF.FFFF_{16}$		2	128	…50M or U/256

5.1.4 Interrupt Map

Interrupt Concept

The UltraSPARC-IIi provides a 6-bit wide interrupt vector for 63 interrupt sources. A separate device, the UPA interrupt concentrator (UIC), provides the inputs for all necessary interrupts. The UIC monitors all interrupts by a round-robin-scheme with 33 MHz, converts them to an own vector and transmits this vector to the processor. The processor interrupt PCI unit (PIE) reflects every vector in 1 state bit. From the state bit a new vector is generated and transmitted to the processor's execution unit. If more than 1 interrupt state bit is active, the transmitting sequence of the new interrupt vector is priority controlled.

Enabling Interrupts

Every interrupt routed to the UIC can be separately enabled or disabled in the interrupt source and in the processor.

The following table lists all interrupt sources, their vectors from the UIC to the PIE, their vectors from the PIE to the processor's execution unit and the respective priority.

Table 35 Interrupt Sources of the Base-50(G)

Function	Device	UIC vector	CPU internal vector	CPU internal priority
Ethernet #1	PCIO	21_{16}	21_{16}	3
SCSI #1	SYM53C875	20_{16}	20_{16}	3
Serial	SAB82532	$2d_{16}$	$2b_{16}$	7
Keyboard	Super I/O	$2b_{16}$	29_{16}	4
Mouse		$2c_{16}$	$2a_{16}$	4
Floppy		29_{16}	27_{16}	8
Parallel		22_{16}	22_{16}	2
UPA64S	UPA64S card	23_{16}	software controlled	5
Audio playback	CS4231	$1f_{16}$	24_{16}	7
Audio capture		24_{16}	23_{16}	8
WatchDog	EBus2 CTRL	18_{16}	$0c_{16}$	6
Temperature		02_{16}	03_{16}	2

Table 35 Interrupt Sources of the Base-50(G) (cont.)

Function	Device	UIC vector	CPU internal vector	CPU internal priority
Universe #0	Universe	07_{16}	00_{16}	7
Universe #1		17_{16}	08_{16}	6
Universe #2		38_{16}	09_{16}	5
Universe #3		10_{16}	$0a_{16}$	2
Universe #4		12_{16}	$0b_{16}$	1
Universe #5		39_{16}	$0d_{16}$	4
Universe #6		00_{16}	$0e_{16}$	2
Universe #7		$1a_{16}$	$0f_{16}$	1
ACFAIL	EBus2 CTRL	$1a_{16}$	$0f_{16}$	1
SYSFAIL		$1a_{16}$	$0f_{16}$	1

Table 36 Interrupt Sources of the I/O-50(G)T

Function	Device	UIC vector	CPU internal vector	CPU internal priority
Ethernet #2	PCIO	15_{16}	02_{16}	5
SCSI #2	SYM53C875	05_{16}	01_{16}	5
PMC1 #A	PMC1	$0e_{16}$	14_{16}	6
PMC1 #B		$0c_{16}$	15_{16}	4
PMC1 #C		$0b_{16}$	16_{16}	3
PMC1 #D		09_{16}	17_{16}	1
PMC2 #A	PMC2	16_{16}	18_{16}	6
PMC2 #B		14_{16}	19_{16}	4
PMC2 #C		13_{16}	$1a_{16}$	3
PMC2 #D		11_{16}	$1b_{16}$	1

5.1.5 UltraSPARC-IIi PCI Bus Interface

The CPU uses a 33 MHz PCI bus as its bus for I/O extensions. This bus is 32 bits wide.

Table 37 UltraSPARC-IIi PCI Address Space (8 GByte)

Address range in PA<40:0>	Size	Description	Generated PCI commands
$1FE.0000.0000_{16}$ …$1FE.00FF.FFFF_{16}$	16 MByte	CPU internal CSR space	n.a.
$1FE.0100.0000_{16}$ …$1FE.01FF.FFFF_{16}$	16 MByte	PCI configuration space	Configuration read or write (may be special cycle)
$1FE.0200.0000_{16}$ …$1FE.02FF.FFFF_{16}$	16 MByte	PCI bus I/O space	I/O read or write
$1FE.0300.0000_{16}$ …$1FE.FFFF.FFFF_{16}$	4 GByte minus 48 MByte	reserved	May wrap to configuration or I/O space behavior
$1FF.0000.0000_{16}$ …$1FF.FFFF.FFFF_{16}$	4 GByte	PCI bus memory space	Memory read or write

For a list of devices connected to the UltraSPARC-IIi PCI bus, see table 30 “Buses, Bus Modes, and Connected Devices” on page 65.

5.2 VMEbus Interface – Universe IIb

The VMEbus interface of the SPARC/CPU-50 is realized via Universe IIb, a PCI-to-VME bridge. Universe IIb provides a direct PCI-to-VME interface, acts as both master and slave in the VMEbus system.

Features

- Full VMEbus system controller functionality
- Integral FIFOs for write posting to maximize bandwidth utilization, 64-bits wide, 32 entries deep
- Programmable DMA controller with linked list support
- Complete suite of VMEbus address and data transfer modes:
  - A32/A24/A16 master and slave
  - D64 (MBLT)/D32/D16/D08 master and slave
  - BLT, ADOH, RMW, LOCK
- Flexible register set, programmable from both PCI bus and VMEbus
- PCI burst transfer length of 128 bytes
- Four 32-bit mailbox registers for interrupt generation on either buses
- Location monitor for interrupts and message passing
- Eight semaphores for control over access to system resources
- Seven software interrupts hardwired to matching VMEbus interrupt level
- PCI base address registers: PCI register image granularity is 4KByte. Two PCI image registers to allow Universe IIb register access from memory and I/O spaces
- PCI slave channel (VMEbus master access) and VME slave channel images

Starting with the “VMEbus Interface Overview” on page 75 a description of selected features is given in the following sections:

- “VMEbus Master Interface” on page 76 and “VMEbus Slave Interface” on page 77
- “DMA-Controller” on page 78
- “Exception Signals SYSRESET, SYSFAIL, and ACFAIL” on page 78
- “VMEbus Slot-1 Functions” on page 79, “VMEbus Timer” on page 80, and “VMEbus Arbitration and VMEbus Requester” on page 81

Data Sheets — For further information and a detailed description, see “VMEbus Interface – Universe IIb” documentation on the vendor’s web page (www.tundra.com).

5.2.1 VMEbus Interface Overview

Supported Transfers — The VMEbus interface supports 64-bit (MBLT), 32-bit, 16-bit, and 8-bit data transfers, as well as unaligned data transfers (UAT). The extended, standard, and short I/O address modifier codes are implemented.

RMW Cycles — Read-modify-write cycles on the VMEbus (RMW cycles) are also supported. The address strobe signal is held low during RMW cycles while the data strobe signals are driven low twice, once for the read cycle and once for the write cycle, and high between both of them.

Interrupt Handler — The complete VMEbus interrupt management is done by Universe IIb enabling the use of a high-end multiprocessor board with distributed interrupt handling. Universe IIb acts as D08(O) interrupt handler, i.e. 8-bit interrupt vectors are supported, whereas 16-bit interrupt vectors are not supported.
All 7 VMEbus interrupt request (IRQ) signals are connected to the interrupt handling logic of Universe IIb.

- All 7 VMEbus IRQ signals can be separately enabled or disabled.
- Every VMEbus interrupt request level causes a PCI interrupt via the UIC if enabled by software.
- Seven software VMEbus interrupts are supported.

Universe IIb Interrupt — The Universe IIb controller routes its PCI-interrupt output to the UIC for interrupting the processor (see table 35 “Interrupt Sources of the Base-50(G)” on page 71 and table 36 “Interrupt Sources of the I/O-50(G)T” on page 72).

Slot-1 function — An arbiter with several arbitration modes and release functions is implemented with all slot-1 system controller functions (see section 5.2.6 “VMEbus Slot-1 Functions” on page 79):
VMEbus arbiter, SYSCLK driver, and IACK daisy-chain driver.

5.2.2 VMEbus Master Interface

PCI Images

Universe IIb transfers data to the VMEbus within a maximum of 8 programmed PCI slave images. Each image can be enabled or disabled independently. Posted write accesses are also programmable.

Access Address

The VMEbus access address is programmable via Universe IIb. Both the start and the end address of the accessible VMEbus address range are programmable in 4-KByte or 64-KByte increments.

Data Transfer Size

The VMEbus master interface supports all 32 data lines. It supports 64-bit (MBLT), 32-bit, 16-bit, and 8-bit data transfers as well as unaligned data transfers (UAT) and read-modify-write transfers.

The VMEbus address range is the largest portion of the address map (see table 31 “UltraSPARC-IIi Physical Address Map (41-bit Physical Addresses)” on page 68). Universe IIb maps the PCI bus transaction to the maximum programmed VMEbus data width. According to this data width, data are packed or unpacked.

Table 38 VMEbus Master Transfer Cycles Defined for Data Bus Width D32

Transfer type	D31…24	D23…16	D15…08	D07…00
Byte on odd address Byte on even address			 x	x
Word			x	x
Long-word	x	x	x	x
Unaligned word Unaligned long-word A Unaligned long-word B	 x 	x x x	x x x	 x
Read-modify-write byte on odd address byte on even address word long-word	 x	 x	 x x x	 x x x

Table 39 VMEbus Master Transfer Cycles Defined for Data Bus Width D16

Transfer type	D31…24	D23…16	D15…08	D07…00
Byte on odd address Byte on even address			 x	x
Word			x	x
Read-modify-write byte on odd address byte on even address word			 x x	 x x

Access Modes

For VMEbus master access, A32/A24/A16 accesses, and CR/CSR accesses are allowed. Single-cycle transfers or block transfers are programmable.
Universe IIb allows data and program accesses in the privileged (supervisor) or non-privileged (user) mode. Additionally, Universe IIb supports 2 user defined address modifier codes. Every access mode (address modifier) can be separately enabled or disabled within the PCI slave image.

5.2.3 VMEbus Slave Interface

Slave Images

Within a maximum of 8 programmed VMEbus slave images Universe IIb can be accessed from the VMEbus. Every image can be enabled or disabled independently. Prefetched read accesses and posted write accesses are also programmable.

Access Address

The access address of the shared RAM (main memory) for other VMEbus masters is programmable via Universe IIb. Both the start and the end address of the shared RAM are programmable in 4-KByte or 64-KByte increments.

Data Transfer Size

The VMEbus slave interface for the shared RAM supports all 32 data lines. It supports 64-bit (MBLT), 32-bit, 16-bit, and 8-bit data transfers as well as unaligned data transfers (UAT).

Access Modes

For VMEbus slave access to the shared RAM, A32/A24/A16 and CR/CSR accesses are allowed.
Universe IIb allows accesses in the privileged (supervisor) or non-privileged (user) mode for both data and program accesses. Additionally Universe IIb provides support for 2 user defined address modifier codes. Each access mode (address modifier) can be separately enabled or disabled within the slave images.

5.2.4 DMA-Controller

Universe IIb uses a DMA controller for high-performance data transfer between the PCI bus and the VMEbus. Source, destination, length of transfer, and transfer protocol are programmable in several registers of Universe IIb.

Direct or Linked List DMA

The DMA controller can be set to one of the following modes:

- Direct DMA: In direct DMA mode the DMA registers are programmed directly by software.
- Linked list DMA: In linked list mode Universe IIb loads the registers from the system memory. The block of the system memory containing the values for the DMA registers is called command packet. The command packets may be linked. If all linked command packets are executed, the DMA is complete.

32-entry FIFO

For increased performance the DMA uses a 32-entry deep, 64-bit wide bidirectional FIFO.

5.2.5 Exception Signals SYSRESET, SYSFAIL, and ACFAIL

SYSRESET*, SYSFAIL*, and ACFAIL* signal exceptions or status. They are connected to the CPU board via buffers, switches, and Universe IIb.

SYSRESET* Input

The VMEbus SYSRESET* signal is monitored by the CPU board if SW800-3 is set appropriately: OFF = enabled (= default, see page 26).

SYSRESET* Output

A SYSRESET* is generated by SPARC/CPU-50 if SW800-4 is set appropriately: OFF = enabled (= default, see page 26). The reason for the SYSRESET* generation may be one of the following:

- The front-panel reset key is active
- The `SW_SYSRESET` bit is set in the `MISC_CTL` register of the Universe IIb
- The watchdog timer generates a reset
- Power-up occurs
- A PCI-reset occurred
- The voltage sensor detects a low voltage on-board

For information to identify the on-board reset source which generated the latest hardware reset, see table 56 “Reset Status Register” on page 98.

SYSFAIL* — Universe IIb can be programmed to generate local interrupts when the SYSFAIL* signal is active. In addition, the System Configuration Registers – SCR – allow to program an interrupt on the rising edge of SYSFAIL* (see section 5.3.12 “SCR: SYSFAIL and ACFAIL” on page 99).

ACFAIL* — The ACFAIL* line can be mapped to a PCI interrupt. The CPU board can never drive the ACFAIL* signal. In addition, the System Configuration Registers – SCR – allow to programm an interrupt on the rising edge of ACFAIL* (see section 5.3.12 “SCR: SYSFAIL and ACFAIL” on page 99).

5.2.6 VMEbus Slot-1 Functions

SPARC/CPU-50 may be used as system controller when the board is plugged into slot-1 (see “Slot-1 Function” on page 9 as well as SW800-1 and SW800-2 in section 3.4 “Switch Settings” on page 25). By default, it detects automatically whether it is plugged into slot 1. It can also be programmed to be system controller via the Universe IIb register `MASC_CTL`.

Caution

If more than one system controller is active in the VMEbus system, the board or other VMEbus participants can be damaged. This is of major importance because this board uses ETL-buffers (enhanced tranceiver logic) which are able to source 60 mA and sink 90 mA on the VMEbus side. Therefore:

- **Always ensure that only one CPU board is configured to be system controller in the VMEbus system.**

Slot-1 (System Controller) Functions — When the CPU board is a slot-1 device, the SPARC/CPU-50 sets up the required system controller functions:

- Enables the arbiter (see section 5.2.8 “VMEbus Arbitration and VMEbus Requester” on page 81),
- Drives SYSCLK to the VMEbus (see “SYSCLK Driver” on page 80),
- Drives floating bus grant levels 0, 1, 2, and 3,
- Allows the Universe IIb bus timer to terminate VME cycles (time-out), if it is enabled (see section 5.2.7 “VMEbus Timer” on page 80).

Slot-1 Auto-detection — The board’s slot-1 auto-detection mechanism probes the VMEbus bus-grant-in-level-3 pin (BG3IN*) during power up to see whether it is possible to pull this signal down to a low-signal level.

- When SPARC/CPU-50 is plugged into slot 1, it succeeds in pulling the VME signal to a low-signal level, because BG3IN* is floating on slot 1. Hence, the CPU board detects slot 1.

- When SPARC/CPU-50 is not plugged into slot 1, it receives the BG3IN* from a board plugged into a lower slot. It fails trying to pull the VME signal to a low-signal level. Hence, the CPU board detects not being plugged into slot 1.

Manual slot-1 Configuration

The following situation may cause SPARC/CPU-50 to detect being plugged into slot-1 although actually being plugged into a different slot:

A VMEbus system begins with the highest daisy-chain priority at slot 1, the left most slot. The higher the slot number, the lower the daisy-chain priority. So slot 2 has higher daisy-chain priority than slot 3, and slot 3 has higher daisy-chain priority than slot 4, and so on. When a board which is not compliant with ANSI-VITA 1-1994 is plugged into a slot with higher daisy-chain priority, auto-detection may fail. This occurs if the higher daisy-chain priority board does not drive the bus-grant-out-level-3 (BG3OUT*) on the VMEbus to the high-signal level as defined by ANSI-VITA 1-1994.

In this situation SPARC/CPU-50 probes its BG3IN* at a low-signal level and concludes that slot 1 is detected. However, the conclusion does not fit the actual system setup. To prevent any mismatch you can enable or disable the slot-1 function manually:

1. Disable the auto-detection by setting SW800-1 appropriately: ON = autodetection disabled (see page 26).
2. Enable or disable the slot-1 functions manually by setting SW800-2 appropriately: ON = slot-1 function disabled (see page 26).

Slot-1 Status

The Universe IIb register `MISC_CTL` can be used to read the status of the slot-1 configuration.

SYSCLK Driver

The CPU board contains all necessary circuits to support the SYSCLK signal as part of being system controller. The output signal is a stable 16-MHz signal. The driver circuitry for the SYSCLK signal can source a current of 60 mA at high level and sink a current of 90 mA at low level. SYSCLK is driven until the board is powered down or low voltage is detected at the low-voltage monitor.

5.2.7 VMEbus Timer

Universe IIb has a VMEbus timer to terminate VME transfers generating a bus error when no acknowledge can be detected after a programmable timeout period.
In addition to the VMEbus timer, Universe IIb provides a VMEbus arbiter timer. This timer can only be enabled when the CPU board provides system controller functions. The timeout periods can be configured in the Universe IIb register `MISC_CTL`.

5.2.8 VMEbus Arbitration and VMEbus Requester

Each transfer to or from an off-board address causes a VMEbus access cycle which is carried out in accordance with the arbitration mechanism defined for the VMEbus. The SPARC/CPU-50 includes:

- A VMEbus arbiter so that it may act as slot-1 system controller (see section 5.2.6 “VMEbus Slot-1 Functions” on page 79)
- A VMEbus requester so that it may access external VMEbus resources.

Arbitration Modes

SPARC/CPU-50 includes one of the following arbitration modes programmable via the Universe IIb `MISC_CTL` register:

- A 4-level fixed-priority arbitration mode,
- A single-level arbitration mode,
- A 4-level round-robin arbitration mode (default).

Note: **According to ANSI-VITA 1-1994, the arbiter must be enabled if the CPU board is located in slot 1 of the VMEbus backplane. It must be disabled if the CPU board is located in any other slot.**

Single-level arbitration is a subset of fixed-priority arbitration mode. To achieve single-level arbitration mode, set Universe IIb to fixed-priority mode. The VMEbus mastership is only granted on level 3.

Requester

SPARC/CPU-50 includes a VMEbus requester so that it may access external VMEbus resources.
The request level is programmable via the Universe IIb `MAST_CTL` register.

Note: **The selection of the VMEbus request level does not depend on the VMEbus arbiter located in Universe IIb.**

Release Modes

The SPARC/CPU-50 provides several software-selectable VMEbus release modes to release VMEbus mastership. The bus release operation is independent of the fact whether the on-board VMEbus arbiter is enabled and independent of the VMEbus arbitration level. Easy handling and use of the VMEbus release modes is provided via programming the Universe IIb `MAST_CTL` register.
Before the bus is released, a read-modify-write (RMW) cycle in progress is always completed.

– ROR
- In the Release on request (ROR) mode bus mastership is released when another VMEbus board requests bus mastership while the CPU board is the current bus master. Ownership of the bus may be assumed by another channel (e.g Universe DMA channel) without rearbitration on the bus, if there are no other requests pending.

– RWD
- In the Release when done (RWD) mode the VME master interface releases the VMEbus ownership, when the channel accessing the VMEbus has transferred its data.

5.3 Ethernet and Access to and for EBus2 Devices – PCIO

PCIO Features

PCIO is a high integration, high performance single chip IO subsystem using a single PCI bus load. As a dual PCI device, PCIO delivers Ethernet and EBus2 functionality to the SPARC/CPU-50 (see section 5.3.1 “Ethernet Interface – PCIO” on page 82 and section 5.3.2 “EBus2 Interface – PCIO” on page 83).

- PCI Local Bus specification 2.1 compliant master/slave interface
- 10/100BaseT Ethernet using a derivative of Media Access Control (MAC), with fully buffered transmit and receive channels; media independent interface (MII)
- Expansion bus 2 interface (EBus2), supporting up to 8 external devices and 4 buffered slave DMA channels.

5.3.1 Ethernet Interface – PCIO

The PCIO on the Base-50(G) delivers the Ethernet #1 interface and the PCIO on the I/O-50(G)T delivers the Ethernet #2 interface. As described above, the PCIO provides Ethernet via a media independent interface (MII). An additional on-board PHYceiver transforms the MII into a 10/100-BaseT interface.

The Ethernet interface consists of 2 major function blocks:

- High performance two-channel DVMA host interface between the MAC and the PCI bus with interrupt generation capability
- Media Access Control (MAC) function for a 10/100 Mbit/s CSMA/CD protocol based network compatible with IEEE 802.3/Ethernet

DVMA

The PCIO DVMA controller enables the Ethernet interface to transfer data to and from the main memory. PCIO supports full duplex operation and provides 2 KByte local on-chip buffers (FIFOs) in each direction.

On-Board PHYceiver

The Twisted Pair Ethernet interface is realized via a PHYceiver device, the PHYceiver – ICS1890. It is directly connected to the MII interface of the PCIO. The PHYceiver is a fully integrated physical layer device supporting 10 and 100 Mbit/s CSMA/CD Ethernet applications. The PHYceiver is compliant with ISO/IEC 8802-3 Ethernet standard for 10- and 100-Mbit/s operation. A station management interface (MII management interface) is provided to enable command and status information exchange between PCIO and PHYceiver. The PHYceiver supports shielded twisted pair (STP) and unshielded twisted pair (UTP) category 5 cables up to 105 m. Operation in half duplex or full duplex mode at either 10 or 100 Mbit/s is possible with control by auto-negotiation or manual selection. By employing auto-negotiation the technology capabilities of the remote link partner may be determined and operation automatically adjusted to the highest performance operating mode common to both. The on-board PHYceiver address is hardwired to 01_{16} as defined by the MII management interface IEEE specification.

Ethernet Interrupt

The Ethernet controller uses the Ethernet interrupt on the UIC for interrupting the UltraSPARC-IIi (see table 35 “Interrupt Sources of the Base-50(G)” on page 71 and table 36 “Interrupt Sources of the I/O-50(G)T” on page 72).

Data Sheets

For a detailed description of the register map of the PCIO Ethernet interface refer to the data sheet “PCIO (PCI I/O Controller) – STP2003QFP” on the vendor’s web page (www.sun.com).

5.3.2 EBus2 Interface – PCIO

The PCIO also provides the interface to the EBus2. EBus2 is a generic slave 8-bit wide DMA bus (pseudo ISA bus) to which off-the-shelf peripherals can be connected.

Base Addresses of PCIO Chip Select Signals

The base addresses of all 8 PCIO chip select signals in the 4 GByte PCI address space is determined by the following 2 registers of PCIO’s configuration address space:

Table 40 PCIO EBus2 Base Address Registers

PCIO configuration space address	Size	Description	Reset Value
010_{16}	32	EBus2 base address register 0: base address for EB_CS#0 (16 MBytes)	$F000.0000_{16}$
014_{16}	32	EBus2 base address register 1: base address for EBus_CS#1…EBus_CS#7 (each 1 MByte)	$F100.0000_{16}$

PCIO Chip Select Signals

The PCIO PCI-to-EBus2 controller delivers 8 decoded chip select signals:

- `EBus_CS#0` (16-MByte space)
- `EBus_CS#1`…`EBus_CS#7` (each 1 MByte space)

It thereby supports up to 8 single- or multi-function Intel-style 8-bit devices with a minimum of glue logic. The resulting memory map in the PCI address space (with the base address registers in the reset value configuration) is described in the table below.
After power up, the Base-50(G) PCIO is in boot mode and uses `EBus_CS#0` for the initial code fetch (OpenBoot). Therefore, the boot PROM is hardwired to `EBus_CS#0`.

Table 41 EBus2 Memory Map in the PCI bus 4 GByte Address Space

PCI addr. range	Description	EBus_ CS#
$F000.0000_{16}$ …$F01F.FFFF_{16}$	Boot PROM or boot flash EPROM, see “Boot PROM, Boot and User Flash EPROM” on page 85	0
$F020.0000_{16}$ …$F0FF.FFFF_{16}$	User flash, see “Boot PROM, Boot and User Flash EPROM” on page 85	0
$F100.0000_{16}$ …$F10F.FFFF_{16}$	“RTC/NVRAM – M48T58” on page 89	1
$F110.0000_{16}$ …$F11F.FFFF_{16}$	reserved	2
$F120.0000_{16}$ …$F12F.FFFF_{16}$	“Audio Interface – CS4231A” on page 90	3
$F130.0000_{16}$ …$F13F.FFFF_{16}$	“Keyboard/Mouse, FDC and Parallel Interface – Super I/O” on page 87	4
$F140.0000_{16}$ …$F14F.FFFF_{16}$	“Serial Interfaces – SAB 82532” on page 87	5
$F150.0000_{16}$ …$F15F.FFFF_{16}$	reserved	6
$F160.0000_{16}$ …$F16F.FFFF_{16}$	“System Configuration Registers – SCR” on page 91	7
$F170.0000_{16}$ …$F17F.FFFF_{16}$	EBus2 controller configuration registers	n.a.

PCIO EBus2 DMA Channels

Additionally, 4 DMA channels for floppy, parallel interface, audio in, and audio out are provided by the EBus2 interface. Each of the 4 DMA channels supports 128-byte deep FIFOs for data stream buffering.

Table 42 PCIO EBus2 DMA Channels

PCIO EBus2 DMA channel no.	Associated device
0	Parallel interface (see page 87)
1	Audio playback (out) (see page 90)
2	Audio capture (in) (see page 90)
3	Floppy disk controller (see page 87)

5.3.3 Boot PROM, Boot and User Flash EPROM

The PCIO's 16-MByte chip select signal `EBus_CS#0` is decoded to 3 chip select signals for

- One boot PROM device or 1 boot flash EPROM device (2 MByte address space)
- Up to two user flash EPROM devices (remaining 14 MBytes address space)

For the base address of `EBus_CS#0` see section 5.3.2 "EBus2 Interface – PCIO" on page 83.

Flash Decoding

The Boot and User Flash Size Control Register indicates the `EBus_CS#0` decoding according to the assembled flash devices (see section 5.3.10 "SCR: Boot and User Flash" on page 95). The decoding is shown in the table below.

Table 43 Boot and User Flash Address Space Configuration

PCI addr. range	Configuration	Device type
$F000.0000_{16}$ …$F00F.FFFF_{16}$	Default config. with SW6-2 = OFF	1 MByte boot PROM one 27C080, 1Mbx8 read-only device
$F020.0000_{16}$ …$F03F.FFFF_{16}$		2 MByte user flash EPROM one 29F016, 2Mbx8, 5V write-protectable via SW4-4
$F040.0000_{16}$ …$F0FF.FFFF_{16}$		12 MByte unused

Table 43 Boot and User Flash Address Space Configuration (cont.)

PCI addr. range	Configuration	Device type
$F000.0000_{16}$ …$F01F.FFFF_{16}$	Alternative config. with SW6-2 = ON	2 MByte boot flash EPROM one 29F016, 2Mbx8, 5V write-protectable via SW4-3
$F020.0000_{16}$ …$F03F.FFFF_{16}$		2 MByte user flash EPROM one 29F016, 2Mbx8, 5V write-protectable via SW4-4
$F040.0000_{16}$ …$F0FF.FFFF_{16}$		12 MByte unused
$F000.0000_{16}$ …$F00F.FFFF_{16}$	Default config. with SW6-2 = OFF in case of 4-MByte user flash fact. opt.	1 MByte boot PROM one 27C080, 1Mbx8 read-only device
$F020.0000_{16}$ …$F05F.FFFF_{16}$		4 MByte user flash EPROM two 29F016, 2Mbx8, 5V write-protectable via SW4-4
$F060.0000_{16}$ …$F0FF.FFFF_{16}$		10 MByte unused
$F000.0000_{16}$ …$F01F.FFFF_{16}$	Alternative config. with SW6-2 = ON in case of 4-MByte user flash fact. opt.	2 MByte boot flash EPROM one 29F016, 2Mbx8, 5V write-protectable via SW4-3
$F020.0000_{16}$ …$F05F.FFFF_{16}$		4 MByte user flash EPROM two 29F016, 2Mbx8, 5V write-protectable via SW4-4
$F060.0000_{16}$ …$F0FF.FFFF_{16}$		10 MByte unused

Programming the Boot or User Flash EPROM

Boot flash EPROM and user flash EPROM are on-board programmable if the switch-selectable hardware write protection is disabled (see section 3.4 “Switch Settings” on page 25).

Caution

Before programming the boot flash EPROM on-board, save the area containing the OpenBoot image for reprogramming purposes. For example, damage to the image in the boot flash EPROM can occur, if power fails during on-board reprogramming.

- To reprogram the boot flash EPROM, see also section 6.6.1 “Copying an Image from the Boot PROM to the Boot Flash EPROM” on page 181.

5.3.4 Serial Interfaces – SAB 82532

The Base-50(G) provides 2 independent full-duplex serial I/O interfaces (A and B). They are implemented via the Enhanced Serial Communication Controller – SAB 82532 at PCI bus base address `F140.000016` on the EBus2.

Device Features

- Two independent full-duplex serial channels
- Two independent baud rate generators
- Hardware handshake support
- Protocol support (HDLC/SDLC)
- Interrupt controlled

Data Sheets

For a detailed description of the registers and functionality refer to the data sheet “Enhanced Serial Communication Controller – SAB 82532” on the vendor’s web page (www.siemens.com).

5.3.5 Keyboard/Mouse, FDC and Parallel Interface – Super I/O

To implement a major part of the UltraSPARC-IIi architecture’s I/O-subsystem a standard PC Super I/O device is utilized, the Super I/O – PC87332VLJ at PCI bus base address `F130.000016` on the EBus2.

Device Features

The Super I/O is a single chip solution for most commonly used I/O peripherals in ISA based computers. It incorporates:

- A floppy disk controller (FDC, see “Floppy Interface” on page 88)
- Two UARTs which are fully NS16450 and NS16550 compatible for the SUN style keyboard/mouse interface,
- An IEEE1284 compatible parallel interface with EPP (Enhanced Parallel Port) and ECP (Enhanced Capabilities Port) compatibility (see “Parallel Interface” on page 88).

Standard PC-AT address decoding for all the on-chip peripherals and a set of configuration registers are also implemented together with advanced power management features.

Data Sheets

For a detailed description of the registers and functionality refer to the data sheet “Super I/O – PC87332VLJ” on the vendor’s web page (www.national.com).

Floppy Interface

The floppy disk controller uses a high performance digital data separator, eliminating the need for any external filter components. One of the 4 DMA channels of the PCIO EBus2 is used for the Super-I/O floppy interface (see table 42 “PCIO EBus2 DMA Channels” on page 85).

- Software compatible with the PC8477
- Superset of DP8473, the 765A and the N82077
- 16-byte FIFO (disabled by default)
- Burst and non-burst modes
- Perpendicular recording drive support
- High-performance internal digital data separator (no external filter components required)
- Low-power CMOS with enhanced power-down mode
- Automatic media-sense support
- Support of all popular 5.25” and 3.5” floppy drives, including the 2.88 MByte 3.5” floppy drive
- Support of fast 2 Mbps and standard 1 Mbps/500 Kbps/250 Kbps tape drives

Parallel Interface

The parallel interface is Centronics compatible. One of the 4 DMA channels of the PCIO EBus2 is used for the Super-I/O parallel interface (see table 42 “PCIO EBus2 DMA Channels” on page 85).

- Uni- or bidirectional parallel interface
- Centronics compliant and operation in either programmed I/O or DMA mode (software or hardware control).
- EPP (Enhanced Parallel Port) and ECP (Enhanced Capabilities Port) compatibility
- Includes protection circuit to prevent damage to the parallel interface when a connected printer is powered up or operated at a higher voltage

Control of Super I/O Power-down Mode

For information on controlling the Super I/O power-down mode see “SUPIO_PWDN (r/w)” on page 99.

5.3.6 RTC/NVRAM – M48T58

The Base-50(G) provides a Real-time Clock and NVRAM – RTC/NVRAM M48T58 at PCI bus base address `F100.000016` on the EBus2.

Table 44 Address Map of the RTC/NVRAM

Address offset range	Access
0000_{16} ...$1FF7_{16}$	NVRAM with 8 KByte minus 8 bytes capacity
$1FF8_{16}$ ...$1FFF_{16}$	RTC registers with clock information in 24-hour BCD format: year, month, date, day, hour, minute, second

Device Features

- 8 KByte ultra low power CMOS SRAM
- Byte-wide accessible real-time clock and power-fail control circuit for automatic power-fail chip deselect and write protection
- Long-life lithium carbon monofluoride battery
- Year-2000 compliant RTC with own crystal

Data Sheets

For further information on the RTC/NVRAM registers and its operation, see “Real-time Clock and NVRAM – RTC/NVRAM M48T58” on the vendor’s web page (www.st.com).

5.3.7 Audio Interface – CS4231A

The Base-50(G) provides an Audio Controller – CS4231A at PCI bus base address `F120.000016` on the EBus2. 2 DMA channels of the PCIO EBus2 are used for the audio interface (see table 42 “PCIO EBus2 DMA Channels” on page 85): one for capture (Micro In, Line/Aux In) and one for playback (Line/Headphone Out).

Device Features

- 16-bit stereo audio converters and complete on-chip filtering for record and playback of 16-bit audio data
- Windows sound system compatible
- ADPCM compression and decompression
- Extensive software support
- MPC level 2 compatible mixer
- Dual DMA registers support full duplex operation for capture and playback
- On-chip FIFOs for higher performance
- Included analog mixing and programmable gain and attenuation

Data Sheets

For further information refer to the data sheet “Audio Controller – CS4231A” on the vendor’s web page (www.crystal.com).

5.3.8 System Configuration Registers – SCR

The Base-50(G) implements a set of system configuration registers via a field programmable gate array XC4003E (FPGA Xilinx LCA) at PCI bus base address F160.000016 on the EBus2. The table below gives an overview of all SPARC/CPU-50 system configuration registers:

Table 45 System Configuration Register set (SCR), all 8-bit Wide

PCI bus addr.	Reset value	Description
F160.000016 (LED 1) and F160.000116 (LED 2)	$F0_{16}$	User LED x Control Registers, x = 1, 2 (see page 92)
F160.000216	$F3_{16}$	Boot and User Flash Size Control Register (see page 95)
F160.000316	$F0_{16}$	I2C Bus Control and Status Register (see page 101)
F160.000416	$F0_{16}$	Miscellaneous Control Register (see page 98)
F160.000516	$F0_{16}$	Miscellaneous Control and Status Register (see page 96)
F160.000616	$F0_{16}$	Watchdog and Temperature Control and Status Register (see page 97)
F160.000716	$F0_{16}$	Watchdog Timer Trigger Register (see page 97)
$F160.0008_{16}$	$F0_{16}$	reserved
$F160.0009_{16}$		
F160.000A16	$F0_{16}$	SYSFAIL and ACFAIL Interrupt Control Register (see page 99)
$F160.000B_{16}$	FF_{16}	reserved
$F160.000C_{16}$		
$F160.000D_{16}$		
F160.000E16	FX_{16}	Reset Status Register (see page 98)
F160.000F16	FX_{16}	System Configuration Identification Register (see page 92)
F160.001016	00_{16}	7-Segment LED Display Control Register (see page 93)
F160.001116	XX_{16}	Rotary Switch Status Register (see page 93)
F160.001216	XX_{16}	SW4 and SW5 Status Register (see page 93)
F160.001316	XF_{16}	SW800 Status Register (see page 94)

Table 46 System Configuration Identification Register

F160.000F$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	ID			

ID (ro) ID indicates the hardware ID of the device containing the system configuration registers.

5.3.9 SCR: Front Panel and Switches

The following registers control front-panel or switch related features:

- “User LED x Control Registers, x = 1, 2” on page 92
- “Rotary Switch Status Register” on page 93
- “7-Segment LED Display Control Register” on page 93
- “SW4 and SW5 Status Register” on page 93
- “SW800 Status Register” on page 94

Table 47 User LED *x* Control Registers, *x* = 1, 2

F160.0000$_{16}$ (LED 1) and F160.0001$_{16}$ (LED 2)								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	BLINK_FREQ		COLOR	

BLINK_FREQ (r/w) BLINK_FREQ specifies the blink frequency:

= 00_2 no blinking
= 01_2 blinking at appr. 0.5 Hz (slow)
= 10_2 blinking at appr. 1 Hz (moderate)
= 11_2 blinking at appr. 2 Hz (fast)

COLOR (r/w) COLOR specifies the status and color of the LED:

= 00_2 off
= 01_2 green
= 10_2 red
= 11_2 yellow

Table 48 7-Segment LED Display Control Register

F160.0010$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	SEG DP	SEG G	SEG F	SEG E	SEG D	SEG C	SEG B	SEG A

DP and SEG_G … SEG_A (w)

DP and SEG_G … SEG_A 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 24 Naming the parts of the 7-segment LED display

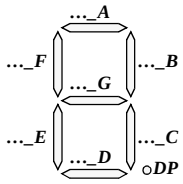


Table 49 Rotary Switch Status Register

F160.0011$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	ROTARY_SW			

ROTARY_SW (ro)

ROTARY_SW indicates the current state of the rotary switch:

= 0000_2 Rotary switch set to 0_{16} (= 0000_2)

= 0001_2 Rotary switch set to 1_{16} (= 0001_2)

= 0010_2 Rotary switch set to 2_{16} (= 0010_2)

… …

= 1110_2 Rotary switch set to E_{16} (= 1110_2)

= 1111_2 Rotary switch set to F_{16} (= 1111_2)

Table 50 SW4 and SW5 Status Register

F160.0012$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	SW5-4	SW6-2	SW5-3	SW5-2	SW5-1	SW4-4	SW4-3

Table 51 SW800 Status Register

F160.0013$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	SW800-1	SW800-2	SW800-3	SW800-4	1	1	1	1

SWx-y (ro), except SW5-2

SWx-y indicates the setting of the respective switch (see table 12 “Default Switch Settings” on page 25):

- SW5-4 – Reserved, must be OFF (see page 25)
- SW6-2 – Boot device selection (OFF = boot from boot PROM, see page 26)
- SW5-3 – Reserved, must be OFF (see page 25)
- SW5-1 – SCSI Termination for SCSI #1 on front panel (OFF = front panel termination automatic, see page 25)
- SW4-4 – User flash EPROM write protection (OFF = user flash EPROM write protected, see page 25)
- SW4-3 – Boot flash EPROM write protection (OFF = boot flash EPROM write protected, see page 25)
- SW800-1 – Automatic VMEbus slot-1 detection (OFF = Automatic detection of VMEbus Slot 1 function, see page 26)
- SW800-2 – Manual VMEbus slot-1 selection (OFF = VMEbus slot 1 function enabled, see page 26)
- SW800-3 – External VMEbus SYSRESET (OFF = VMEbus SYSRESET generates on-board RESET, see page 26)
- SW800-4 – VMEbus SYSRESET generation (OFF = SYSRESET is driven to VMEbus, see page 26)

= 0 Switch is ON.

= 1 Switch is OFF.

SW5-2 (ro)

SW5-2 indicates the setting of the SW5-2: SCSI Termination for SCSI #1 on P2 (OFF = backplane termination disabled, see page 25).

= 0 Switch is OFF.

= 1 Switch is ON.

5.3.10 SCR: Boot and User Flash

The Boot and User Flash Size Control Register within the system configuration register set indicates the decoding of EBus_CS#0 according to the assembled flash device type (see section 5.3.3 “Boot PROM, Boot and User Flash EPROM” on page 85).

Note: **OpenBoot initializes the Boot and User Flash Size Control Register during power up with the correct value. BOOTROM_SIZE only concerns the boot flash EPROM, whereas the boot PROM is always decoded to 1 MByte.**
Therefore:Never reprogram this register.

Table 52 Boot and User Flash Size Control Register

F160.0002$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	USERROM_SIZE		BOOTROM_SIZE	

*x*ROM_SIZE (ro) — USERROM_SIZE and BOOTROM_SIZE indicate the decoding of the user and boot flash EPROM, respectively.

= 00_2, 01_2 — reserved

= 10_2 — Offset range 00.0000_{16} … $0F.FFFF_{16}$

= 11_2 (default) — Offset range 00.0000_{16} … $1F.FFFF_{16}$

Boot device selection — For information on selecting the boot device, see “SW_PLCC_TSOP (r/w)” on page 96.

5.3.11 SCR: Watchdog, Temperature Sensors, and Reset

Watchdog — An on-board watchdog can be enabled by SW6-4 (ON = enabled, see page 26). To start the watchdog timer after enabling it via SW6-4, it is necessary to trigger WDI in the Watchdog Timer Trigger Register once. The watchdog monitors the processor activity. When the watchdog timer interval expires, i.e. the watchdog timer is no longer triggered periodically, the watchdog timer activates its WDO output and a watchdog timer interrupt can be generated if IE_WDT in the Watchdog and Temperature Control and Status Register is set appropriately. The generation of an interrupt is indicated by the Reset Status Register.

Temperature Sensors

The 2 temperature sensors connected to the I^2C Bus can be programmed to generate a temperature control interrupt (see also section 5.3.13 “SCR: I2C-Bus” on page 100).
The temperature sensors may be programmed in such a way that either the `O.S.` output signal is operating in the comparator mode or the interrupt mode. The state of the `O.S.` output signal is indicated by the IS_TEMPn (ro) bits in the Miscellaneous Control and Status Register.

- In the comparator mode `O.S.`
  - is cleared (`0`) when the current temperature exceeds an upper temperature limit T_{OS}
  - is set (`1`) only when the current temperature falls below a lower limit T_{HYST}
- In the interrupt mode `O.S.`
  - is cleared (`0`) whenever the current temperature exceeds an upper temperature limit (T_{OS}) or falls below a lower limit (T_{HYST})
  - is set (`1`) only upon reading one of the temperature sensor’s internal registers via the I^2C-Bus

An interrupt is generated if `O.S.` is cleared. For further information see “Temperature Sensor – LM75” in the *SPARC/CPU-50 Data Sheets*. The generation of an interrupt is controlled and indicated by the Watchdog and Temperature Control and Status Register. OpenBoot initialises the SPARC/CPU-50 temperature sensors for interrupt mode operation.

Table 53 Miscellaneous Control and Status Register

F160.0005$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	IS_TEMP2	IS_TEMP1	0	SW_PLCC_TSOP

`IS_TEMP`*n* (ro) `IS_TEMP1` and `IS_TEMP2` indicate the state of the `O.S.` output signal of the respective temperature sensor (#1 or #2).

= `0` (default) `O.S.` output signal is low (`0`).

= `1` `O.S.` output signal is high (`1`).

`SW_PLCC_-TSOP` (r/w) `SW_PLCC_TSOP` controls the selection of the boot device: the boot PROM or the boot flash EPROM. After reset this bit is cleared (`0`).

= `0` (default) The boot device specified by SW6-2 is selected.

= `1` The boot flash EPROM is selected.

Table 54 Watchdog Timer Trigger Register

F160.0007$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	WDI	1	1	1

WDI (r/w) WDI is used to trigger the watchdog timer by changing the value of WDI. Triggering the watchdog timer clears the watchdog timer. Default is 0.

Table 55 Watchdog and Temperature Control and Status Register

F160.0006$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	IP_ TEMP	IE_ TEMP	IS_ WDT	IE_ WDT

IP_TEMP (ro) IP_TEMP indicates whether one of the 2 temperature sensors signals an alarm condition. Only if enabled via appropriate setting of IE_TEMP, an interrupt is generated.

= 0 No temperature control interrupt is pending.

= 1 A temperature control interrupt is pending.

IE_TEMP (r/w) IE_TEMP specifies whether the generation of temperature control interrupts is enabled or disabled. A pending temperature control interrupt is indicated by IP_TEMP.

= 0 Interrupt generation disabled (default after reset).

= 1 Interrupt generation enabled.

IE_WDT (r/w) IE_WDT specifies whether the generation of watchdog timer interrupts is enabled or disabled. A pending watchdog timer interrupt is indicated by WDT_RESET in the Reset Status Register.

= 0 Interrupt generation disabled (default after reset).

= 1 Interrupt generation enabled.

IS_WDT (ro) IS_WDT indicates status of the WDO output signal and thereby indicates the status of the watchdog timer if started (see “Watchdog” on page 95).

= 0 Watchdog timeout.

= 1 Watchdog timer has not expired.

Table 56 Reset Status Register

$F160.000E_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	1	WDT_ RESET	BUS_ RESET	KEY_ RESET

The Reset Status Register allows to identify the on-board reset source which generated the latest hardware reset:

- Watchdog – WDT_RESET,
- VMEbus SYSRESET – BUS_RESET,
- Front-panel reset key – KEY_RESET,
- Power-on reset: If all status bits in the Reset Status Register are cleared (0) after a reset, the reset has been generated due to a power-on reset. A power-on reset occurs when the power supply unit is turned on, or the power supply sensor detects that one of the required power supply voltages falls below a tolerable limit.

Once one of the bits has been set to 1, it is cleared (0) by setting RESET_STAT_CLR in the Miscellaneous Control Register (see page 98).

KEY_RESET (ro) — KEY_RESET indicates that a reset has been generated via the front-panel reset key if KEY_RESET = 1.

BUS_RESET (ro) — BUS_RESET indicates that a reset has been generated because the VME-bus SYSRESET* signal has been asserted if BUS_RESET = 1.

WDT_RESET (ro) — WDT_RESET indicates that a reset has been generated because of a watchdog timeout if WDT_RESET = 1.

Table 57 Miscellaneous Control Register

$F160.0004_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	RESET _STAT _CLR	reser ved	SUPIO _PWDN	reser ved

RESET_STAT_CLR (r/w) — RESET_STAT_CLR specifies to clear all reset status bits in the Reset Status Register when set to 1.

SUPIO_PWDN (r/w) SUPIO_PWDN controls the Super I/O power-down mode.

= 0 Turns off the power-down mode.

= 1 Turns on the power-down mode.

5.3.12 SCR: SYSFAIL and ACFAIL

The SYSFAIL and ACFAIL Interrupt Control Register allows to generate a CPU interrupt in case of a low-to-high transition of the corresponding VMEbus signal. Thereby it implements a Universe IIb upgrade.

Note: **SYSFAIL and ACFAIL are rising edge sensitive.**

Note: **The ACFAIL interrupt utilizes the same interrupt request pin as the SYSFAIL interrupt. So the interrupt handler must determine with help of the interrupt pending bits IP_ACFAIL and IP_SYSF_ENUM if a SYSFAIL or an ACFAIL low-to-high transition caused the interrupt.**

Table 58 SYSFAIL and ACFAIL Interrupt Control Register

F160.000A$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	IP_ ACFAIL	IE_ ACFAIL	IP_ SYSF_ ENUM	IE_ SYSF_ ENUM

IP_ACFAIL (r/w) IP_ACFAIL indicates whether an ACFAIL interrupt is pending due to a low-to-high transition of the signal ACFAIL. Only if enabled via appropriate setting of IE_ACFAIL, an interrupt is generated. To clear such a pending interrupt, write a 1 to IP_ACFAIL.

= 0 No interrupt is pending.

= 1 An interrupt is pending. Writing a 1 to IP_ACFAIL clears such a pending interrupt.

IE_ACFAIL (r/w) IE_ACFAIL specifies whether the generation of an interrupt due to a low-to-high transition of ACFAIL is enabled or disabled. Such a pending interrupt is indicated by IP_ACFAIL.

= 0 Interrupt generation disabled (default after reset).

= 1 Interrupt generation enabled.

`IP_SYSF_ENUM` (r/w)

`IP_SYSF_ENUM` indicates whether a SYSFAIL interrupt is pending due to a low-to-high transition of the signal SYSFAIL. Only if enabled via appropriate setting of `IE_SYSF_ENUM`, an interrupt is generated. To clear such a pending interrupt, write a `1` to `IP_SYSF_ENUM`.

`= 0` No interrupt is pending.

`= 1` An interrupt is pending. Writing a `1` to `IP_SYSF_ENUM` clears such a pending interrupt.

`IE_SYSF_ENUM` (r/w)

`IE_SYSF_ENUM` specifies whether the generation of an interrupt due to a low-to-high transition of SYSFAIL is enabled or disabled. Such a pending interrupt is indicated by `IP_SYSF_ENUM`.

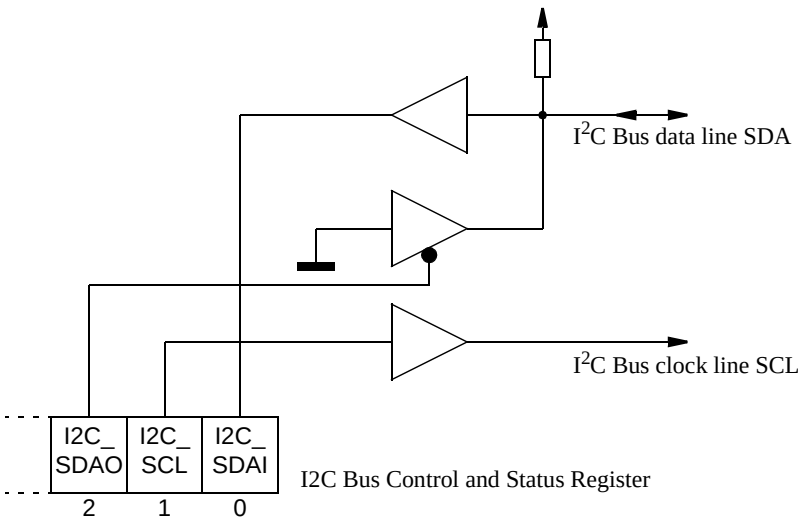
`= 0` Interrupt generation disabled (default after reset).

`= 1` Interrupt generation enabled.

5.3.13 SCR: I²C-Bus

The SPARC/CPU-50 is equipped with an I²C bus consisting of a data line and a clock line which are software-controlled. For further information on the I²C Bus, refer to the I²C Bus specification. The figure below shows the I²C Bus interface implementation in detail.

Figure 25 I²C Bus Interface



Accessible Devices

The devices listed in the table below can be accessed via the I²C Bus. They are I²C Bus slaves and are identified by unique addresses also listed in the table below:

Table 59 I²C Bus Slave Addresses

I²C Bus Slave Address	Description
1001.111_2	Temperature sensor #1 LM75
1001.110_2	Temperature sensor #2 LM75
1010.000_2	ID PROM of the Base-50(G) – XICOR X24C04 Serial E²PROM
1010.010_2	ID PROM of I/O-50(G)T – XICOR X24C04 Serial E²PROM

The ID-PROMs are used to store board specific parameters. The temperature sensors can be programmed for temperature monitoring and protection against overheating. To locate the temperature sensors see figure 6 “Location Diagram of the Base Board (Schematic)” on page 18.

The devices can be accessed via the I²C Bus and can be read and written via the I2C Bus Control and Status Register.

Table 60 I²C Bus Control and Status Register

F160.0003$_{16}$								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	1	1	1	1	reserved	I2C_SDAO	I2C_SCL	I2C_SDAI

See I2C_SDAO for information how to write data to I²C Bus slaves. See I2C_SDAI for information how to read data from I²C Bus slaves.

I2C_SDAO (r/w)

I2C_SDAO is used to force low (0) or high (1) level on the I²C Bus data line.

Note: **Whenever data is read from another I²C Bus participant, I2C_SDAO must be set to 1, otherwise data is corrupted.**

= 0 Low level (0) is forced on the data line.

= 1 High level (1) is forced on the data line.

Note: **Since the I2C_SDAO is realized as an open collector output onto the I^2C Bus data line, it is necessary to be aware of the typical RC-time constant. This time constant is 5 usec. It is important that the read-out of the data line via I2C_SDAI, which immediately follows the setting of the SDA data line to the high level 1, strictly respects the 5 usec time constant before executing the read-out. Otherwise the data will be read incorrectly on I2C_SDAI.**

`I2C_SCL` (r/w) `I2C_SCL` is used to control the state of the clock line of the I^2C bus.

`= 0` Low level (`0`) is forced on the data line.

`= 1` High level (`1`) is forced on the data line.

`I2C_SDAI` (ro) `I2C_SDAI` is used to read data from the I^2C Bus data line. However, to do so `I2C_SDAO` must be set to `1`, otherwise data is corrupted.

- If `I2C_SDAO` is set (`1`), the state of `I2C_SDAI` indicates the state of the data line of the I^2C bus.
- If `I2C_SDAO` is cleared (`0`), low level (`0`) is forced on the data line of the I^2C bus and `I2C_SDAI` is cleared, too.

5.4 SCSI Interface – SYM53C875

Ultra/wide SCSI #1 and #2 (single-ended) are implemented on the Base-50(G) and on the I/O-50(G)T (SCSI), respectively. Ultra SCSI (Fast-20) is an extension of the SCSI-3 standard that expands the bandwidth of the SCSI bus and allows faster synchronous SCSI transfer rates (approximately double the synchronous transfer rates of Fast SCSI-2).
Each – SCSI #1 or #2 – is implemented by a PCI-Ultra SCSI (Fast-20) I/O Interface – SYM53C875, both independent from each other. 8-bit Ultra SCSI is provided via the front panel connector only on the Base-50(G). Ultra/wide (16-bit) SCSI is provided only via the backplane connector of Base-50(G) and I/O-50(G)T.

Device Features

- Single-chip high-performance PCI-Wide Ultra SCSI I/O Processor
- Enhanced PCI performance and flexibility
- SCSI SCRIPTS high-level programming interface and SCRIPTS instruction prefetch, allowing tailored SCSI sequences to be executed locally (SCRIPTS – Symbios Logic-developed SCSI programming language)
- 536-byte DMA FIFO buffer allowing bursts of up to 128 transfers

- Support for PCI extended access cycles: memory read multiple, memory read line, as well as memory write and invalidate
- Supports wide high-speed SCSI bus transfers in single-ended mode up to 40 MB/s synchronous Ultra SCSI (Fast-20) transfers and 14 MB/s asynchronous transfers
- Provides full SCSI-2 capabilities
- SCAM (SCSI configured automatically) Level 1 functionality
- 32 additional scratchpad registers for user-defined functions
- Direct PCI-to-SCSI connection
- Features Symbios Logic Tolerant SCSI driver and receiver technology for reliable operation in all cabling environments
- Solaris driver support for hard disk, tape, CD-ROM, and removable media peripherals

SCSI Interrupt

The SCSI controller uses SCSI interrupt on the UIC for interrupting the UltraSPARC-IIi (see table 35 “Interrupt Sources of the Base-50(G)” on page 71 and table 36 “Interrupt Sources of the I/O-50(G)T” on page 72).

Data sheets

For further information refer to the data sheet “PCI-Ultra SCSI (Fast-20) I/O Interface – SYM53C875” on the vendor’s web page (www.lsilogic.com).

5.5 PCI-to-Dual-PCI Bridge – APB

The APB (Advance PCI Bridge) is a PCI-to-dual-PCI bus bridge on the I/O-50(G)T. It drives 2 secondary PCI buses A and B. PCI bus A is not used and reserved. PCI bus B connects to

- 2 PMC Slots with Busmode Support (see page 103),
- the SCSI #2 via a second SCSI Interface – SYM53C875 (see page 102),
- and Ethernet #2 via a second Ethernet Interface – PCIO (see page 82).

5.6 PMC Slots with Busmode Support

The PMC busmode signals are supported for both PMC slots via 5 general purpose I/O pins of the SCSI #2 controller on-board the I/O-50(G)T (see section 5.4 “SCSI Interface – SYM53C875” on page 102). The bus-

mode signals allow detection of installed PMC cards and proper initialization of the PMC card according to the protocol supported by the installed PMC module.

Table 61 I/O Pins for PMC Busmode Function

Busmode signal	Pin name at SCSI#2 controller
BUSMODE[4]	GPIO[0] / FETCH
BUSMODE[3]	GPIO[1] / MASTER
BUSMODE[2]	GPIO[2] / MAS2
PMC#1 BUSMODE[1]	GPIO[3]
PMC#2 BUSMODE[1]	GPIO[4]

BUSMODE[4…2] are driven by the host and specify the busmode command transferred to the PMC modules as described in table 62 “BUSMODE [4..2] (r/w) Commands” on page 104. The answer of PMC module #*x* is transferred via PMC#*x* BUSMODE[1] as described in table 63 “PMC#x BUSMODE[1] (ro) Response Encoding” on page 105.

Table 62 BUSMODE [4..2] (r/w) Commands

Busmode [4..2]	Command
000_2	The modules at PMC slot 1 and 2 shall return "Card Present", if they are plugged into the slot and no bus protocol is used. This is the default setting.
001_2	The modules at PMC slot 1 and 2 shall return "Card Present" if they are PCI capable and PCI protocol is used (default).
010_2	The modules at PMC slot 1 and 2 shall return "Card Present" if they are SBus capable and SBus protocol is used.
111_2	No host present
011_2, 100_2, 101_2, 110_2 are reserved	

Table 63 PMC#*x* BUSMODE[1] (ro) Response Encoding

PMC#*x* BUSMODE[1]	Description
0	“Card Present”: PMC module present which has the requested capability and uses the requested protocol
1	no PMC module present or PMC module does not have the requested capability or PMC module does not use the requested protocol

6 Force OpenBoot Enhancements

The OpenBoot ported to the SPARC/CPU-50 is based upon OpenBoot V3.10 obtained from Sun Microsystems. This section describes the enhancements to the standard OpenBoot firmware that have been done for the SPARC/CPU-50. Examples are given when it seems necessary to convey the usage of a particular word or a group of words.

Note: **OpenBoot is subject to changes. Some features are only available with specific versions of the software. Features not available on all OpenBoot versions are marked with the version given, e.g. ... (OpenBoot 3.10.4 and above) ... . For information on the latest OpenBoot version and how to upgrade refer to the SMART service accessible via the Force Computers World Wide Web site.**

Base Information

For a description of standard OpenBoot 3.x firmware features, refer to the *OpenBoot 3.x Manual Set* via `http://docs.sun.com`.

Besides the commands already provided by the standard OpenBoot firmware, the OpenBoot firmware available on the SPARC/CPU-50 includes further words for:

- Accessing and controlling the VMEbus interface which is implemented by the Universe IIb (see section 6.1.1 “Controlling the VMEbus Master and Slave Interface” on page 109 which also includes information on the VMEbus Interface Initialization)
- Accessing and configuring system specific components (see section 6.2 “System Configuration” on page 152)
- Controlling BusNet (see section 6.3 “BusNet Support (OpenBoot 3.10.6 and above)” on page 160)
- NVRAM configuration variable setting (see section 6.4 “NVRAM Configuration Variables” on page 171)
- Accessing and programming available flash memories (see section 6.5 “Flash Memory Support” on page 175).

For information on additional hardware dependencies, see section 6.6 “Hardware Dependencies” on page 181.

Notation

In general, each word is described using the following notation:

name (*stack-comment*) *description*

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

The *stack-comment* notation which is enclosed in parentheses describes the stack parameters passed to and returned from a word. It shows the effect of the word on the evaluation stack. The parameters passed and returned to the word are separated by the "—" within the *stack-comment*:

parameters before execution — parameters after execution

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

6.1 VMEbus Interface

VMEbus Interface Initialization

The following is done as standard initialization of the VMEbus interface during power up:

- Disable SYSFAIL* at register VCSR_CLR,SYSFAIL*
- Disable all PCI bus Slave/Master Window control/BS/BD registers
- Clear bit MISC_CTL,BI to exit the Universe IIb from BI-mode

Section Overview

This section describes the OpenBoot enhancements related to the VMEbus interface:

- Starting with section 6.1.1 "Controlling the VMEbus Master and Slave Interface" on page 109 giving general aspects which are extended to by
  - section 6.1.2 "Controlling the VMEbus Master Interface" on page 111,
  - and section 6.1.3 "Controlling the VMEbus Slave Interface" on page 113,
- Continuing with a detailed discussion of OpenBoot words in
  - section 6.1.4 "Generic Information" on page 116
  - section 6.1.5 "Universe IIb Register Accesses" on page 116
  - section 6.1.6 "Interrupter (OpenBoot 3.10.6 and above)" on page 123
  - section 6.1.7 "Interrupt Mapping (OpenBoot 3.10.6 and above)" on page 126
  - section 6.1.8 "Interrupt Handler (OpenBoot 3.10.6 and above)" on page 128
  - section 6.1.9 "VMEbus Arbiter" on page 132

Further Information

For further information refer to the VMEbus Interface – Universe IIb data sheet on the vendor’s web page (www.tundra.com).

6.1.1 Controlling the VMEbus Master and Slave Interface

The OpenBoot enhancements controlling the VMEbus interface of the SPARC/CPU-50 basically provide support for VMEbus single cycles.

VMEbus Addressing

The VMEbus has a number of distinct address spaces represented by a subset of the 64 possible values encoded by the 6 address modifier bits. The size of the address space depends on the particular address space. For example, the standard (A24) address space is limited to 16 MByte, whereas the extended (A32) address space allows to address 4 GByte. An additional bit – which corresponds with the VMEbus LWORD* signal – is used to select 16-bit data or 32-bit data.
A physical VMEbus address is numerically represented by the pair *phys.hi* (also called *space*) and *phys.lo* (also called *offset*). *phys.hi* consists of:

- Six address modifier bits `AM0`...`AM5` corresponding with bit `0`...`5`
- The data width bit `LWORD*` in bit `6`: `0` = 16-bit, `1` = 32-bit.

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.

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.lo*) and the second part represents the *space* (*phys.hi*).

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 (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 $FFFF_{16}$)

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

Example

```
ok vmea24d32 burst .
7f
ok
```

`burst-mblt` (*phys.high-single - phys.high-burst*) available with OpenBoot 3.10.6 and above, converts the numeric representation of any VMEbus AML constant in single-transaction form to its burst-transaction (MBLT) form.

Example

```
ok vmea32d32 burst -mblt .
7c
ok
```

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

Example

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

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

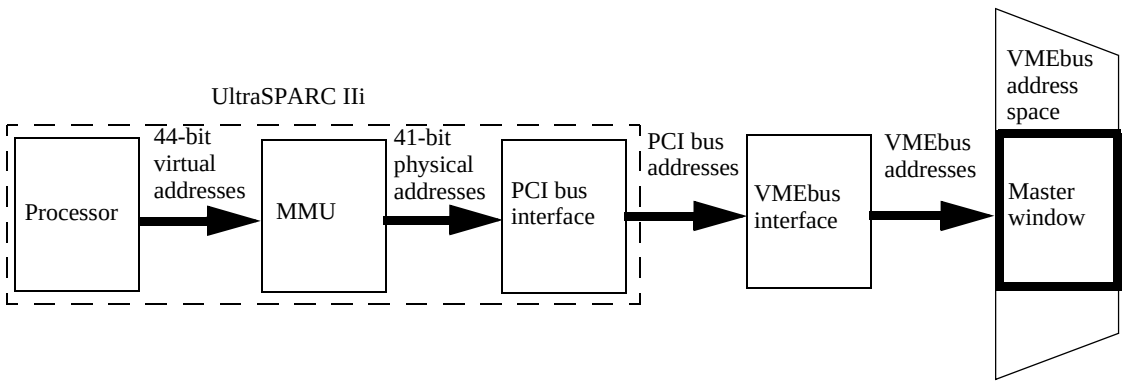
Example

```
ok vmea32d16 vme-program .
6e
ok
```

6.1.2 Controlling the VMEbus Master Interface

As shown in the following figure the processor emits virtual addresses during a data transfer cycle which are translated to physical addresses by the MMU. The VMEbus is connected to the PCI bus interface. The VMEbus interface itself responds to unique PCI bus addresses and executes the appropriate VMEbus transfer. Depending on the PCI bus addresses and the state of specific registers within the VMEbus interface, the VMEbus interface addresses a specific VMEbus address space.

Figure 26 Address Translation (Master): UltraSPARC – PCI bus– VMEbus



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

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

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

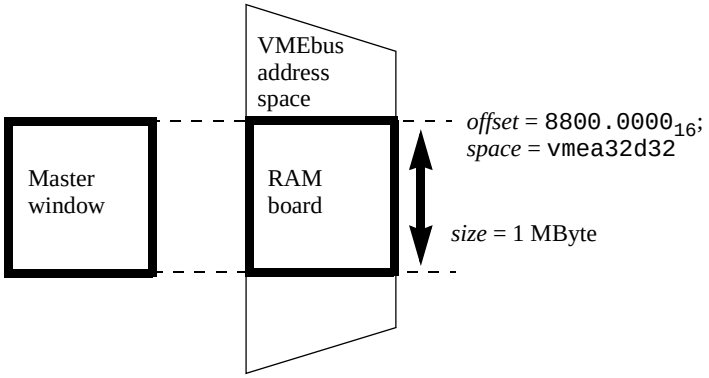
`vme-memmap` (*offset space size — vaddr*) initializes the VMEbus master interface according to the parameters *offset* and *space* and returns the virtual address *vaddr* to be used to access the specified VMEbus area. The parameters describe the VMEbus address area in detail:

- *offset* specifies the physical VMEbus address of the area to be accessed and must be 64 kByte aligned.
- *space* specifies the address space where the VMEbus area is located in.
- *size* specifies the size of the VMEbus area and must be 64 kByte aligned.

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 27 **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 used later to store the virtual address of the VMEbus area. The second command listed above makes 1 MByte within the extended (A32) VMEbus address space available to the processor's virtual address space beginning at physical address 8800.0000_{16}. 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.

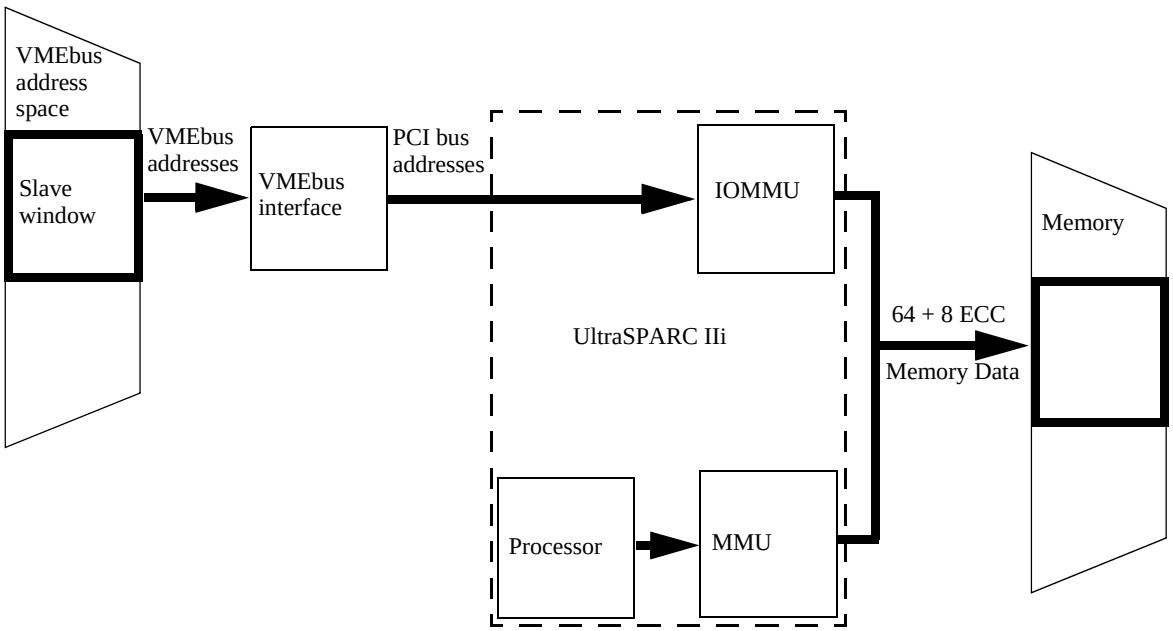
The VMEbus area previously made available to the processor's virtual address space is removed from the virtual address space using the `vme-free-virtual` command as shown below:

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

6.1.3 Controlling the VMEbus Slave Interface

As shown in the figure below the VMEbus interface responds to unique VMEbus addresses and translates these addresses to virtual PCI bus addresses. The IOMMU translates these virtual PCI bus addresses to 64-bit physical memory addresses which address a certain area within the onboard memory.

Figure 28 Address Translation (Slave): VMEbus – PCI bus – UltraSPARC-IIi



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 PCI bus addresses, emitted by the VMEbus interface during a slave access, with the physical addresses of the allocated memory. Furthermore, the contents of the MMU table are modified to associate the virtual addresses, which are emitted by the processor during accesses to the on-board memory, with the physical addresses of the allocated memory.
- The VMEbus interface has to be enabled, in order to allow accesses from the VMEbus to the on-board memory.

OpenBoot provides commands to make the on-board memory available to one of the VMEbus address spaces, and to remove the on-board mem-

ory from these VMEbus address spaces. `set-vme-slave` performs all steps to make a specified amount of memory available at a specific VMEbus address space. `reset-vme-slave` removes the on-board memory from the VMEbus address space.

`set-vme-slave` (*phys.lo phys.hi 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 VMEbus address range is represented by *phys.lo phys.hi* and *size.* `set-vme-slave` returns a virtual address to access the VMEbus area on local memory. The local PCI bus address and slave memory is requested by OpenBoot.

Example

Assumed that 1 MByte of on-board memory should be made available to the extended (A32) address space of the VMEbus beginning at the VMEbus address 4080.0000_{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 1 MByte available within the extended (A32) VMEbus address space beginning at physical address 4080.0000_{16}. The virtual address returned by the command is stored in the variable `my-mem` which has been defined by the command `value`. The variable `my-mem` may be used later to access the on-board memory.

`reset-vme-slave` (*vaddr size* —) , available with Openboot 3.10.4 or higher, resets the VMEbus slave interface associated with the virtual address *vaddr* and destroys the mapping which was necessary to make the memory available to VMEbus. Previous OpenBoot versions use instead of *size* the window *n* as argument for determining the corresponding VMEbus mapping.

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

6.1.4 Generic Information

The variable described below is used to retrieve generic information about the VMEbus interface. It is declared as `value`.

`universe-va` (— *vaddr*) returns the virtual base address *vaddr* (32-bit) of the Universe IIb control and status registers (UCSR) at processor address range (PCI memory space). Universe IIb registers occupy 4 kBytes of internal processor memory space.

6.1.5 Universe IIb Register Accesses

The commands described below are used to read data from and store data in a register of the Universe IIb. If reading and storing data is possible for the register under consideration, the 2 words associated to the 2 actions differ by their last character:

- *command*`@` is used for reading a register. The register's value is returned on top the stack. This is indicated by the stack-comment notation (— *long*).
- *command*`!` is used for storing data in a register. The register's value has to be provided on top the stack. This is indicated by the stack-comment notation (*long* —).

Every register is 32-bit wide, but not every bit is used.

Naming conventions

The following naming conventions are used throughout this section:

- Commands for register access with offsets `0`…$\texttt{FF}_{16}$ have a prefix `uni-`… to be uniquely identified among other PCI related words. The remaining part of the words are closely related to the register abbreviations used with the Universe IIb.
- *long* refers to a 32-bit sized value as following:
  - OpenBoot versions below 3.10.6 in big endian notation
  - OpenBoot version 3.10.6 and above in little endian notation.

Note: **All words described below are only available if the VMEbus device has been selected (FCode package). To select the VMEbus device node, see section 6.1.16 "VMEbus Device Node Specific Words" on page 150.**

`uni-pci-id@` (— *long*) returns the value of the PCI Configuration Space ID Register.

`uni-pci-csr@` (— *long*) returns the 32-bit value of the PCI Configuration Space Control and Status Register. Use `uni-pci-csr!` (*long* —) to store a 32-bit value in this register.

`uni-pci-class@` (— *long*) returns the 32-bit value of the PCI Configuration Class Register.

`lsi0-ctl@` (— *long*) returns the 32-bit value of the PCI Slave Image Control Register 0. Use `lsi0-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi0-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 0 Base Address Register. Use `lsi0-bs!` (*long* —) to store a 32-bit value in this register.

`lsi0-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 0 Bound Address Register. Use `lsi0-bd!` (*long* —) to store a 32-bit value in this register.

`lsi0-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 0 Translation Offset. Use `lsi0-to!` (*long* —) to store a 32-bit value for this offset.

`lsi1-ctl@` (— *long*) returns the 32-bit value of the PCI Slave Image Control Register 1. Use `lsi1-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi1-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 1 Base Address Register. Use `lsi1-bs!` (*long* —) to store a 32-bit value in this register.

`lsi1-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 1 Bound Address Register. Use `lsi1-bd!` (*long* —) to store a 32-bit value in this register.

`lsi1-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 1 Translation Offset. Use `lsi1-to!` (*long* —) to store a 32-bit value for this offset.

`lsi2-ctl@` (— *long*) returns the 32-bit value of the PCI Slave Image Control Register 2. Use `lsi2-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi2-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 2 Base Address Register. Use `lsi2-bs!` (*long* —) to store a 32-bit value in this register.

`lsi2-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 2 Bound Address Register. Use `lsi2-bd!` (*long* —) to store a 32-bit value in this register.

`lsi2-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 2 Translation Offset. Use `lsi2-to!` (*long* —) to store a 32-bit value for this offset.

`lsi3-ctl@` (— *long*) returns the 32-bit value of the PCI Slave Image Control Register 3. Use `lsi3-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi3-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 3 Base Address Register. Use `lsi3-bs!` (*long* —) to store a 32-bit value in this register.

`lsi3-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 3 Bound Address Register. Use `lsi3-bd!` (*long* —) to store a 32-bit value in this register.

`lsi3-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 3 Translation Offset. Use `lsi3-to!` (*long* —) to store a 32-bit value for this offset.

`lsi4-ctl@` (— *lon*g) returns the 32-bit value of the PCI Slave Image Control Register 4. Use `lsi4-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi4-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 4 Base Address Register. Use `lsi4-bs!` (*long* —) to store a 32-bit value in this register.

`lsi4-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 4 Bound Address Register. Use `lsi4-bd!` (*long* —) to store a 32-bit value in this register.

`lsi4-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 4 Translation Offset. Use `lsi4-to!` (*long* —) to store a 32-bit value for this offset.

`lsi5-ctl@` (— *lon*g) returns the 32-bit value of the PCI Slave Image Control Register 5. Use `lsi5-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi5-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 5 Base Address Register. Use `lsi5-bs!` (*long* —) to store a 32-bit value in this register.

`lsi5-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 5 Bound Address Register. Use `lsi5-bd!` (*long* —) to store a 32-bit value in this register.

`lsi5-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 5 Translation Offset. Use `lsi5-to!` (*long* —) to store a 32-bit value for this offset.

`lsi6-ctl@` (— *lon*g) returns the 32-bit value of the PCI Slave Image Control Register 6. Use `lsi6-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi6-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 6 Base Address Register. Use `lsi6-bs!` (*long* —) to store a 32-bit value in this register.

`lsi6-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 6 Bound Address Register. Use `lsi6-bd!` (*long* —) to store a 32-bit value in this register.

`lsi6-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 4 Translation Offset. Use `lsi6-to!` (*long* —) to store a 32-bit value for this offset.

`lsi7-ctl@` (— *lon*g) returns the 32-bit value of the PCI Slave Image Control Register 7. Use `lsi7-ctl!` (*long* —) to store a 32-bit value in this register.

`lsi7-bs@` (— *long*) returns the 32-bit value of the PCI Slave Image 7 Base Address Register. Use `lsi7-bs!` (*long* —) to store a 32-bit value in this register.

`lsi7-bd@` (— *long*) returns the 32-bit value of the PCI Slave Image 7 Bound Address Register. Use `lsi7-bd!` (*long* —) to store a 32-bit value in this register.

`lsi7-to@` (— *long*) returns the 32-bit value of the PCI Slave Image 7 Translation Offset. Use `lsi7-to!` (*long* —) to store a 32-bit value for this offset.

`mast-ctl@` (— *long*) returns the 32-bit value of the Master Control Register. Use `mast-ctl!` (*long* —) to store a 32-bit value in this register.

`misc-ctl@` (— *long*) returns the 32-bit value of the Miscellaneous Control Register. Use `misc-ctl!` (*long* —) to store a 32-bit value in this register.

`misc-stat@` (— *long*) returns the 32-bit value of the Miscellaneous Status Register.

`vsi0-ctl@` (— *long*) returns the 32-bit value of the VMEbus slave Image 0 Control Register. Use `vsi0-ctl!` (*long* —) to store a 32-bit value in this register.

`vsi0-bs@` (— *long*) returns the 32-bit value of the VMEbus slave Image 0 Base Address Register. Use `vsi0-bs!` (*long* —) to store a 32-bit value in this register.

`vsi0-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 1 Bound Address Register. Use `vsi0-bd!` (*long* —) to store a 32-bit value in this register.

`vsi0-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 1 Translation Offset. Use `vsi0-to!` (*long* —) to store a 32-bit value for this offset.

`vsi1-ctl@` (— *long*) returns the 32-bit value of the VMEbus slave Image 1 Control Register. Use `vsi1-ctl!` (*long* —) to store a 32-bit value in this register.

`vsi1-bs@` (— *long*) returns the 32-bit value of the VMEbus slave Image 1 Base Address Register. Use `vsi1-bs!` (*long* —) to store a 32-bit value in this register.

`vsi1-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 1 Bound Address Register. Use `vsi1-bd!` (*long* —) to store a 32-bit value in this register.

`vsi1-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 1 Translation Offset. Use `vsi1-to!` (*long* —) to store a 32-bit value for this offset.

`vsi2-ctl@` (— *long*) returns the 32-bit value of the VMEbus slave Image 2 Control Register. Use `vsi2-ctl!` (*long* —) to store a 32-bit value in this register.

`vsi2-bs@` (— *long*) returns the 32-bit value of the VMEbus slave Image2 Base Address Register. Use `vsi2-bs!` (*long* —) to store a 32-bit value in this register.

`vsi2-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 2 Bound Address Register. Use `vsi2-bd!` (*long* —) to store a 32-bit value in this register.

`vsi2-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 2 Translation Offset. Use `vsi2-to!` (*long* —) to store a 32-bit value for this offset.

`vsi3-ctl@` (— *long*) returns the 32-bit value of the VMEbus slave Image 3 Control Register. Use `vsi3-ctl!` (*long* —) to store a 32-bit value in this register.

`vsi3-bs@` (— *long*) returns the 32-bit value of the VMEbus slave Image 3 Base Address Register. Use `vsi3-bs!` (*long* —) to store a 32-bit value in this register.

`vsi3-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 3 Bound Address l Register. Use `vsi3-bd!` (*long* —) to store a 32-bit value in this register.

`vsi3-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 3 Translation Offset. Use `vsi3-to!` (*long* —) to store a 32-bit value for this offset.

`vsi4-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 4 Bound Address l Register. Use `vsi4-bd!` (*long* —) to store a 32-bit value in this register.

`vsi4-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 4 Translation Offset. Use `vsi4-to!` (*long* —) to store a 32-bit value for this offset.

`vsi5-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 5 Bound Address l Register. Use `vsi5-bd!` (*long* —) to store a 32-bit value in this register.

`vsi5-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 5 Translation Offset. Use `vsi5-to!` (*long* —) to store a 32-bit value for this offset.

`vsi6-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 6 Bound Address Register. Use `vsi6-bd!` (*long* —) to store a 32-bit value in this register.

`vsi6-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 6 Translation Offset. Use `vsi6-to!` (*long* —) to store a 32-bit value for this offset.

`vsi7-bd@` (— *long*) returns the 32-bit value of the VMEbus slave Image 7 Bound Address Register. Use `vsi7-bd!` (*long* —) to store a 32-bit value in this register.

`vsi7-to@` (— *long*) returns the 32-bit value of the VMEbus slave Image 7 Translation Offset. Use `vsi7-to!` (*long* —) to store a 32-bit value for this offset.

`vcsr-clr@` (— *long*) returns the 32-bit value of the VMEbus CSR bit clear Register. Use `vcsr-clr!` (*long* —) to store a 32-bit value in this register.

`vcsr-set@` (— *long*) returns the 32-bit value of the VMEbus CSR bit set Register. Use `vcsr-set!` (*long* —) to store a 32-bit value in this register.

`vcsr-bs@` (— *long*) returns the 32-bit value of the VMEbus CSR Base Address Register. Use `vcsr-bs!` (*long* —) to store a 32-bit value in this register.

`lm-ctl@` (— *long*) returns the 32-bit value of the Location Monitor Control Register. Use `lm-ctl!` (*long* —) to store a 32-bit value in this register.

`lm-bs@` (— *long*) returns the 32-bit value of the Location Monitor Base Address Register. Use `lm-bs!` (*long* —) to store a 32-bit value in this register.

Additional Words in OpenBoot 3.10.6 and above

The following words are available with OpenBoot 3.10.6 and above:

`uni-pci-misc0@` (— *long*) returns the 32-bit value of the PCI Configuration Miscellaneous 0 Register. Use `uni-pci-misc0!` (*long* —) to store a 32-bit value in this register.

`uni-pci-misc1@` (— *long*) returns the 32-bit value of the PCI Configuration Miscellaneous 1 Register. Use `uni-pci-misc1!` (*long* —) to store a 32-bit value in this register.

`vint-en@` (— *long*) returns the 32-bit value of the VMEbus Interrupt Enable Register. Use `vint-en!` (*long* —) to store a 32-bit value in this register.

`vint-stat@` (— *long*) returns the 32-bit value of the VMEbus Interrupt Status Register. Use `vint-stat!` (*long* —) to store a 32-bit value in this register.

`vint-map0@` (— *long*) returns the 32-bit value of the VMEbus Interrupt Map 0 Register. Use `vint-map0!` (*long* —) to store a 32-bit value in this register.

`vint-map1@` (— *long*) returns the 32-bit value of the VMEbus Interrupt Map 1Register. Use `vint-map1!` (*long* —) to store a 32-bit value in this register.

`vint-map2@` (— *long*) returns the 32-bit value of the VMEbus Interrupt Map 2 Register. Use `vint-map2!` (*long* —) to store a 32-bit value in this register.

`lint-en@` (— *long*) returns the 32-bit value of the PCI Interrupt Enabe Register.
Use `lint-en!` (*long* —) to store a 32-bit value in this register.

`lint-stat@` (— *long*) returns the 32-bit value of the PCI Interrupt Status Register.
Use `lint-stat!` (*long* —) to store a 32-bit value in this register.

`lint-map0@` (— *long*) returns the 32-bit value of the PCI Interrupt Map 0 Register.
Use `lint-map0!` (*long* —) to store a 32-bit value in this register.

`lint-map1@` (— *long*) returns the 32-bit value of thePCI Interrupt Map 1 Register.
Use `lint-map1!` (*long* —) to store a 32-bit value in this register.

`lint-map2@` (— *long*) returns the 32-bit value of the PCI Interrupt Map 2 Register.
Use `lint-map2!` (*long* —) to store a 32-bit value in this register.

`statid@` (— *long*) returns the 32-bit value of the Interrupt Status ID Out Register.
Use `statid!` (*long* —) to store a 32-bit value in this register.

`v1-statid@` (— *long*) returns the 32-bit value of the Virq1 Status ID Register.
Use `v1-statid!` (*long* —) to store a 32-bit value in this register.

`v2-statid@` (— *long*) returns the 32-bit value of the Virq2 Status ID Register.
Use `v2-statid!` (*long* —) to store a 32-bit value in this register.

`v3-statid@` (— *long*) returns the 32-bit value of theVirq3 Status ID Register.
Use `v3-statid!` (*long* —) to store a 32-bit value in this register.

`v4-statid@` (— *long*) returns the 32-bit value of the Virq4 Status ID Register.
Use `v4-statid!` (*long* —) to store a 32-bit value in this register.

`v5-statid@` (— *long*) returns the 32-bit value of the Virq5 Status ID Register.
Use `v5-statid!` (*long* —) to store a 32-bit value in this register.

`v6-statid@` (— *long*) returns the 32-bit value of the Virq6 Status ID Register.
Use `v6-statid!` (*long* —) to store a 32-bit value in this register.

`v7-statid@` (— *long*) returns the 32-bit value of the Virq7 Status ID Register.
Use `v7-statid!` (*long* —) to store a 32-bit value in this register.

`mbox0@` (— *long*) returns the 32-bit value of the Mailbox 0 Register.
Use `mbox0!` (*long* —) to store a 32-bit value in this register.

`mbox1@` (— *long*) returns the 32-bit value of the Mailbox 1 Register.
Use `mbox1!` (*long* —) to store a 32-bit value in this register.

`mbox2@` (— *long*) returns the 32-bit value of the Mailbox 2 Register.
Use `mbox2!` (*long* —) to store a 32-bit value in this register.

`mbox3@` (— *long*) returns the 32-bit value of the Mailbox 3 Register.
Use `mbox3!` (*long* —) to store a 32-bit value in this register.

`sema0@` (— *long*) returns the 32-bit value of the Semaphore 0 Register.
Use `sema0!` (*long* —) to store a 32-bit value in this register.

`sema1@` (— *long*) returns the 32-bit value of the Semaphore 1 Register.
Use `sema1!` (*long* —) to store a 32-bit value in this register.

`lmisc@` (— *long*) returns the 32-bit value of the PCI Miscellaneous Register.
Use `lmisc!` (*long* —) to store a 32-bit value in this register.

6.1.6 Interrupter (OpenBoot 3.10.6 and above)

For a detailed description of the VMEbus interrupter refer to the Universe II User Manual.

Declaration: The term *"SYSFAIL-"* - *assertion* - describes the falling edge condition of the SYSFAIL* signal and the term *"SYSFAIL+"* - *negation* - describes the rising edge condition of the SYSFAIL* signal.

Note: **The words described below control the Interrupter functions but are only available if the VMEbus device has been selected (FCode package).**

To select the VMEbus device node, see section 6.1.16 "VMEbus Device Node Specific Words" on page 150.

`lint-irq!` (*flag irq#* —) enables or disables local 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 sysfail- lint-irq!
ok
```

The possible *`irq#`* sources and their respective meaning is described in the table below:

Table 64 Local Generated Interrupt Sources

irq#	Event
`vown` (0_{16})	VMEbus ownership interrupt
`virq1` (1_{16}) `virq2` (2_{16}) ... `virq7` ($7)_{16}$	receipt of a VMEbus interrupt
`local-dma` (8_{16})	DMA controller terminated write error
`local-lerr` (9_{16})	local PCI bus error
`local-verr` (a_{16})	VMEbus error
`viack` (c_{16})	VMEbus software interrupt acknowledge
`local-sw-int` (d_{16})	PCI software interrupt
`sysfail-` (e_{16})	assertion of VMEbus SYSFAIL* signal
`acfail-` (f_{16})	assertion of VMEbus ACFAIL* signal
`mbox0` (10_{16}) `mbox1` (11_{16}) ... `mbox3` (13_{16})	mailbox specified by irq# taken
`lm0` (14_{16}) `lm1` (15_{16}) ... `lm3` (17_{16})	location monitor specified by irq# selected
`sysfail+` (18_{16})	negation of VMEbus SYSFAIL* signal
`acfail+` (19_{16})	negation of VMEbus ACFAIL* signal

`vint-irq!` (*flag irq#* —) enables or disables VMEbus 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 lint6 vint-irq!
ok
```

The possible *irq#* sources and their respective meaning is described in the table below:

Table 65 VMEbus Generated Interrupt Sources

irq#	Event
`lint0` (0_{16}) `lint1` (1_{16}) ... `lint7` (7_{16})	receipt of a PCI bus interrupt at *irq#*
`vme-dma` (8_{16})	receipt of a VMEbus DMA error
`vme-lerr` (9_{16})	receipt of a local bus error
`vme-verr` (a_{16})	receipt of a VMEbus error
`vme-sw-int` (c_{16})	receipt of a software IACK
`vme-sw1` (18_{16}) `vme-sw2` (19_{16}) ... `vme-sw7` ($1e_{16}$)	software interrupt at a specific VMEbus level (1-7) is executed

`vme-intr-pending? (` *level*— `true | false )` evaluates the VMEbus interrupt pending register of the universe. With respect to the VMEbus interrupt *level* specified it returns a flag signalling the behavior of the interrupt. In case of `true` is returned the interrupt at *level* is pending.

`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` 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*. This command has no effect, if a interrupt is still pending at the level specified - which means that the previous interrupt request has not been acknowledged.

`.vint-irq-stat ( — )` displays the contents of the VMEbus interrupt status registers for interrupts generated on the VMEbus.

Example

```
ok .vint-irq-stat
LINT0 :0  LINT1 : 0 LINT2 : 0 LINT3 : 0 LINT4 : 0
LINT5 :0  LINT6 : 0 LINT7 : 0 VME-DMA : 0 VME-LERR: 0
VME-VERR:0 VME-SINT: 0 MBOX0 : 0 MBOX1 : 0
MBOX2 :0  MBOX3 : 0 VME-SW1 : 0 VME-SW2 : 0 VME-SW3 : 0
VME-SW4 :0 VME-SW5 : 0 VME-SW6 : 0 VME-SW7 : 0
ok
```

`.lint-irq-stat ( — )` displays the contents of the local interrupt status registers for interrupts generated on the PCI bus. For completeness there are LCA sources displayed too.

Example

```
ok .lint-irq-stat
VOWN : 0 VIRQ1 : 0 VIRQ2 : 0 VIRQ3 : 0 VIRQ4 : 0
VIRQ5 : 0 VIRQ6 : 0 VIRQ7 : 0 LCL-DMA : 0 LCL-LERR: 0
LCL-VERR: 0 VIACK : 0 LCL-SINT: 0 SYSFAIL-: 0
ACFAIL- : 0 MBOX0 : 0 MBOX1 : 0 MBOX2 : 0 MBOX3 : 0
LM0 :0 LM1 : 0 LM2 : 0 LM3 : 0
SYSFAIL+: 0 ACFAIL+ : 0 WATCHDOG: 0 TEMP : 0
ok
```

6.1.7 Interrupt Mapping (OpenBoot 3.10.6 and above)

For a detailed description of the VMEbus interrupt mapping refer to the Universe II User Manual.

Declaration

The term *"SYSFAIL-" - assertion* - describes the falling edge condition of the SYSFAIL* signal and the term *"SYSFAIL+" - negation* - describes the rising edge condition of the SYSFAIL* signal.

Note: **The words described below control the Interrupt mapping functions but are only available if the VMEbus device has been selected (FCode package):**
To select the VMEbus device node, see section 6.1.16 "VMEbus Device Node Specific Words" on page 150.

`vint-irq-mapping!` (*mapping irq#* —) maps an event *irq#* generated from the Universe IIb to a specific VMEbus interrrupt level *mapping*. For valid *irq#* values and their respective parameter mappings, see below.

Table 66 VMEbus Generated Interrupt Mapping

mapping (effect)	*irq#* (cause)
virq1 (1_{16}) virq2 (2_{16}) ... virq7 (7_{16})	lint0 (0_{16}) lint1 (1_{16}) ..... lint7 (7_{16}) vme-dma (8_{16}) vme-lerr (9_{16}) vme-verr (a_{16}) vme-sw-int (c_{16}) vme-sw1 (18_{16}) vme-sw2 (19_{16}) ... vme-sw7 ($1e_{16}$

Example

```
ok virq3 lint4 vint-irq-mapping!
ok
```

`vint-irq-mapping@ ( irq# — mapping )` returns the VMEbus interrupt mapping - to which the given event *irq#* is mapped to.
For valid *irq#* values , see table 66 “VMEbus Generated Interrupt Mapping” on page 127.

Example

```
ok lint4 vint-irq-mapping@
ok .
3
ok
```

`lint-irq-mapping! ( mapping irq# — )` selects the local interrupt mapping to be generated due to the event specified by *irq#*.
For valid *irq#* values and their respective parameter mappings, see below.

Table 67 Local Generated Interrupt Mapping

mapping (effect)	irq# (cause)
lint0 (0_{16}) lint1 (1_{16}) ..... lint7 (7_{16})	virq1 (1_{16}) virq2 (2_{16}) ... virq7 (7_{16}) local-dma (8_{16}) local-lerr (9_{16}) local-verr (a_{16}) viack (c_{16}) local-sw-int (d_{16}) sysfail- (e_{16}) acfail- (f_{16}) mbox0 (10_{16}) mbox1 (11_{16}) ... mbox3 (13_{16}) lm0 (14_{16}) lm1 (15_{16}) ... lm3 (17_{16}

Example

```
ok lint7 sysfail- lint-irq-mapping!
ok
```

`lint-irq-mapping@ (` *irq# -- mapping* `)` returns the local interrupt mapping asserted by the Universe IIb when the event specified by *irq#* has occurred.
For valid *irq#* values , see table 67 “Local Generated Interrupt Mapping” on page 128.

Example

```
ok sysfail- lint-irq-mapping@
ok .
7
ok
```

6.1.8 Interrupt Handler (OpenBoot 3.10.6 and above)

For a detailed description of the VMEbus interrupt handler refer to the Universe II User Manual.

Declaration

The term “*SYSFAIL-*” - *assertion* - describes the falling edge condition of the SYSFAIL* signal and the term “*SYSFAIL+*” - *negation* - describes the rising edge condition of the SYSFAIL* signal.

Note: **The words described below control the Interrupt Handler functions but are only available if the VMEbus device has been selected (FCode package):**
To select the VMEbus device node, see section 6.1.16 “VMEbus Device Node Specific Words” on page 150.
There are special interrupt service routines available for servicing the VMEbus interrupts at level 1..7, SYSFAIL*, and ACFAIL* signals. During the initialization process of the Universe IIb they are loaded automatically at power-up; for a detailed description see below.

VMEbus interrupt handler for each VMEbus level 1...7

For each VMEbus interrupt level there is a specific interrupt service routine loaded. Its interrupt mapping for each VMEbus interrupt level (level = 1 ... 7) is predefined accordingly:

Table 68 Interrupt Mapping for VMEbus Interrupts VIRQ1 ... VIRQ7

name of ISR	interrupter	mapping	INO	PIL asserted
vme-l1-handler	VIRQ1	LINT6	e_{16}	2_{16}
vme-l2-handler	VIRQ2	LINT5	d_{16}	3_{16}
vme-l3-handler	VIRQ3	LINT4	b_{16}	5_{16}
vme-l4-handler	VIRQ4	LINT3	a_{16}	7_{16}
vme-l5-handler	VIRQ5	LINT2	9_{16}	9_{16}
vme-l6-handler	VIRQ6	LINT1	8_{16}	b_{16}
vme-l7-handler	VIRQ7	LINT0	0_{16}	d_{16}

Each handler fulfills almost equal tasks regarding clearance of the interrupt pending registers inside the Universe IIb and servicing the interrupt acknowledge cycle if the related interrupt previously is enabled inside the local/VME interrupt enable register.

The VMEbus vectors obtained during servicing an VMEbus interrupt at level 1..7 can be monitored by executing the command `.vme-vectors`.
The user has to decrease the processor interrupt level and enable the desired interrupt source to allow VMEbus interrupt handling.

Example

```
ok true virq3 lint-irq!
ok 0 pil!
<-------- VMEbus interrupt VIRQ3 occurred
ok .vme-vectors
1: --   2: --   3: 22 4: --   5: --   6: --   7: --
ok
```

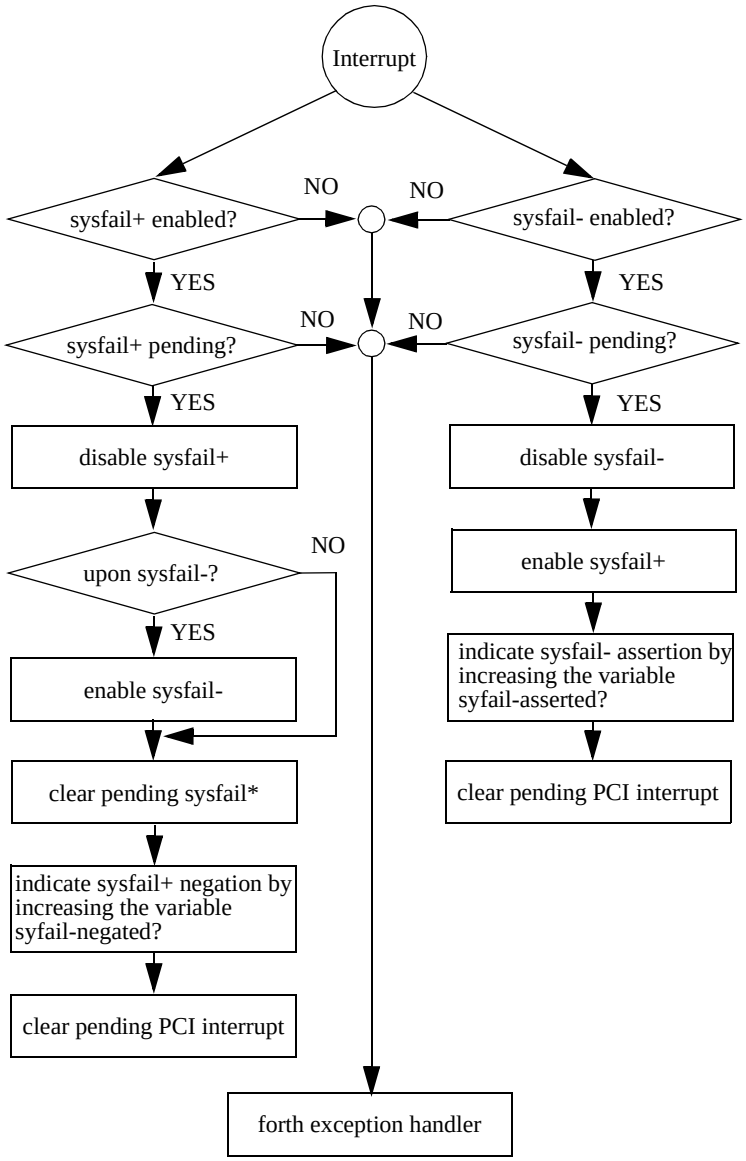
The example above states the reception and interrupt handling of an VMEbus interrupt occurred at level 3 with the vector h# 22.

SYSFAIL*/ACFAIL* (negation/assertion) Interrupt handler

For SYSFAIL*/ACFAIL* an interrupt service routine and the appropriate interrupt mapping is implemented. In case of an interrupt, the SYSFAIL*/ACFAIL* interrupt service routine first determines if the interrupt source is enabled inside the local/VME interrupt enable register of the Universe IIb.
Then it decides if it was a negation or assertion of the interrupt pending. If it was a assertion the handler enables the negation event.
Finally it disables the interrupt event and clears the interrupt pending.

For more information see the detailed sysfail* design flow below.
ACFAIL* handling is performed in the same way.

Figure 29 SYSFAIL*/ACFAIL* Handling



The following variables are available for monitoring the AC-FAIL*/SYSFAIL* signals:

- `sysfail-asserted?`
- `sysfail-negated?`
- `acfail-asserted?`
- `acfail-negated?`

The interrupt mapping for SYSFAIL*/ACFAIL* is predefined as shown below.

Table 69 Interrupt Mapping for ACFAIL*/SYSFAIL*

name	interrupter	mapping	INO	PIL asserted
acsys-handler	ACFAIL* SYSFAIL*	LINT7	f_{16}	6_{16}

For a example, on VMEbus SYSFAIL- handling, see below:

Example

```
ok sysfail-asserted? off
ok sysfail-asserted? @ .
0
ok true sysfail- lint-irq!
ok 0 pil!
<-------- SYSFAIL- occurred
ok sysfail-asserted? @ .
1
ok
```

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

6.1.9 VMEbus Arbiter

Note: **The words described below control the arbiter but are only available if the VMEbus device has been selected (FCode package). To select the VMEbus device node, see section 6.1.16 “VMEbus Device Node Specific Words” on page 150.**

`vme-slot1-ena` (—) enables the board to act as the system controller. To enable the system controller function, `vme-slot1-ena` calls `vme-slot1!` and passes the value `true` to it.

`vme-slot1-dis` (—) disables the board to act as the system controller. To disable the system controller function, `vme-slot1-dis` calls `vme-slot1!` and passes the value `false` to it.

`vme-slot1!` (`true | false` —) enables or disables the board to operate as the system controller. If `true` is passed to the command, the board acts as the system controller. If `false` is passed to the command, the system controller function is disabled (Universe IIb register MISC-CTL, bit SYSCON).

The following words are available with OpenBoot 3.10.6 and above.

`vme-slot1?` (— `true|false`) returns a flag indicating the system controller function. `true` is returned if the Universe IIb is system-controller, `false` if not.

`.vme-arb-ctrl` (—) displays the current contents of the VMEbus Arbiter Control Register.

`vme-arb-mode@` (— *mode*) returns the current arbiter operating *mode*. In case of roundrobin (`RRS`), `zero` is returned, otherwise if priorized (PRI) `one` is returned.

`vme-arb-mode!` (*mode* —) selects the arbiter mode specified by *mode*. There are constants available to specify the arbitration mode. The constant `RRS` (0) selects round robin mode and the constant `PRI` (1) selects the priorized mode.

`set-arb-mode` (*addr length* —) sets the arbiter mode according to the contents of 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
" rrs"	Round Robin Select
" pri"	Priorized

6.1.10 VMEbus Requester (OpenBoot 3.10.6 and above)

For a detailed description of the VMEbus requester refer to the Universe II User Manual.

Note: **The words described below control the requester but are only available if the VMEbus device has been selected (FCode package): To select the VMEbus device node, see section 6.1.16 "VMEbus Device Node Specific Words" on page 150.**

`vme-bus-request-level@` (— *level*) returns the VMEbus request *level* in use when the VMEbus interface tries to gain the ownership of the VMEbus. The value of *level* may be one of the values in the range one through three. Each value specifies one of the four VMEbus request levels.

`vme-bus-request-level!` (*level* —) selects the bus-request *level* to be used when the VMEbus is being accessed. The value of *level* may be one of the values in the range one through three. Each value specifies one of the four VMEbus request levels

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

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

`vme-bus-release-mode@` (— *mode*) returns the VMEbus release *mode* in use when the VMEbus interface has gained the ownership of the VMEbus.

`vme-bus-release-mode!` (*mode* —) selects the release *mode* to be used when the VMEbus interface has gained the ownership of the VMEbus. Two constants are available to specify the release mode. The constant `RWD` (0) selects the release mode release when done and the constant `ROR` (1) selects the release mode release on request.

`set-rel-mode` (*addr length* —) sets the release mode according to the contents of the string—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
“ rwd”	Release When Done

`.vme-req-ctrl` (—) displays the current contents of the VMEbus Requester Control Register.

6.1.11 VMEbus Status Signals (OpenBoot 3.10.6 and above)

Note: **The commands listed below are available to access and control the VMEbus status signals. Be aware the words described below are only available if the VMEbus device is selected. To select the VMEbus device node, see section 6.1.16 “VMEbus Device Node Specific Words” on page 150.**

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

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

6.1.12 Mailboxes and Location Monitor (OpenBoot 3.10.6 and above)

Note: **The commands listed below are available to access and control the mailbox and location monitor register. The words described below are only available if the VMEbus device is selected.**

To select the VMEbus device node, see section 6.1.16 “VMEbus Device Node Specific Words” on page 150.

Mailboxes

The Universe IIb provides four 32 bit mailbox registers `mbox0 .. mbox3`.
Write accesses to these registers can cause an interrupt on the local bus, if the mapping is defined correspondingly and the `mbox0 .. mbox3` is enabled.
In the example below the mailbox 0 is used to generate an interrupt at the local interrupt line (LINT) 3, which is routed to a processor interrupt at level 7.

For more information on the interrupt routing, see table 66 “VMEbus Generated Interrupt Mapping” on page 127.

Example

```
ok interrupt-occurred? off
ok interrupt-occurred? @
0
ok cd vme
ok 7 catch-interrupt
ok lint3 mbox0 lint-irq-mapping!
ok true mbox0 lint-irq!
ok 6 pil!
ok 12345678 mbox0!
ok interrupt-occurred? @
-1
ok
```

To explain this example in detail:

- **`interrupt-occurred? off`**

  resets the variable `interrupt-occurred?`

- **`interrupt-occurred? @`**

  fetches the value of `interrupt-occurred?`, initialized in the line above - (0)

- **`cd vme`**

  makes the Universe IIb device called `vme` the active node, to be able to make use of the VMEbus commands.

- **`7 catch-interrupt`**

  catches the processor interrupt at level 7, due to the fact that interrupts generated at the local interrupt line 3 are routed to processor interrupts at level 7 (see table 68 “Interrupt Mapping for VMEbus Interrupts VIRQ1 ... VIRQ7” on page 129).

- **`lint3 mbox0 lint-irq-mapping!`**

  maps the mailbox 0 to the local interrupt line 3. For more information to the command `lint-irq-mapping` section 6.1.7 “Interrupt Mapping (OpenBoot 3.10.6 and above)” on page 126.

- **`true mbox0 lint-irq!`**

  enables the mailbox 0 interrupt. The command `lint-irq!` is explained within the section 6.1.6 “Interrupter (OpenBoot 3.10.6 and above)” on page 123.

- **`6 pil!`**

  decreases the processor interrupt level to level 6 to enable processor interrupts up to level 7.

- **`12345678 mbox0!`**

  writes the 32 bit value 12345678_{16} to the mailbox 0, thus a interrupt is generated by the Universe IIb.

- **`interrupt-occurred? @`**

  fetches the value of `interrupt-occurred?`. `true (-1)` is returned, indicating an interrupt at processor level 7.

Location Monitor

The location monitor allows interrupt generation on events occurred across the VMEbus backplane. The Universe IIb responds to a read or write cycle within a specific VMEbus range by generating one of four interrupts mapped to the local bus.

`location-monitor-set` (*phys_lo aml* —) sets the Universe IIb location monitor with the VMEbus address specified by *phys_lo* and the address modifier given by *aml*.

`location-monitor-clear ( — )` clears the location monitor by disabling the image bit of the LM_CTL register

The example below generates an location monitor interrupt at processor interrupt level 7. For more information on the interrupt routing, see table 66 “VMEbus Generated Interrupt Mapping” on page 127.

Example

```
ok interrupt-occurred? off
ok interrupt-occurred? @
0
ok cd vme
ok 7 catch-interrupt
ok lint3 lm0 lint-irq-mapping!
ok true lm0 lint-irq!
ok h# a0000.0000 vmea32d32 location-monitor-set
ok h# a0000.0000 vmea32d32 1meg vme-memmap value my-vme
ok 6 pil!
ok my-vme 10 dump
ok interrupt-occurred? @
-1
ok
```

To explain this example in detail:

- **`interrupt-occurred? off`**

  resets the variable `interrupt-occurred?`

- **`interrupt-occurred? @`**

  fetches the value of `interrupt-occurred?`, initialized in the line above - (`0`)

- **`cd vme`**

  makes the Universe IIb device called `vme` the active node, to be able make use of the `vme` commands.

- **`7 catch-interrupt`**

  catches the processor interrupt at level 7, due to the fact that interrupts generated at the local interrupt line 3 are routed to processor interrupts at level 7.

- **`lint3 lm0 lint-irq-mapping!`**

  maps the location monitor 0 to the local interrupt line 3. For more information to the command `lint-irq-mapping,` see “Interrupt Mapping (OpenBoot 3.10.6 and above)” on page 126.

- **`true lm0 lint-irq!`**

  enables the location moniotr 0 interrupt. The command `lint-irq!` is explained within section 6.1.6 “Interrupter (OpenBoot 3.10.6 and above)” on page 123.

- **`h# a0000.0000 vmea32d32 location-monitor-set`**

  sets the location monitor for VMEbus A32 address spaces at $A0000000_{16}$.

- **`h# a0000.0000 vmea32d32 1meg vme-memmap value my-vme`**

  sets up a VMEbus master image for VMEbus A32 address spaces at $A0000000_{16}$. For more information on mapping a VMEbus master image refer to section 6.1.13 “VMEbus Master Interface” on page 139.

- **`6 pil!`**

  decreases the processor interrupt level to level 6 to enable processor interrupts up to level 7.

- **`my-vme 10 dump`**

  tries to read from VMEbus at address $A0000000_{16}$. Thus a processor interrupt is generated at level 7.

- **`interrupt-occurred? @`**

  fetches the value of `interrupt-occurred?` Now `true (-1)` is returned, indicating an interrupt at processor level 7.

6.1.13 VMEbus Master Interface

The Universe IIb provides 8 register sets to control any VMEbus master operation (PCI-slave). Each set may be used to address a certain address range within the VMEbus address space. The registers within a set are called LSI*n*-CTL, LSI*n*-BS, LSI*n*-BD, and LSI*n*-TO with *n* = 0...7 (LSI = Local Slave Image).

When the VMEbus is accessed from the CPU board, the part of the Universe IIb connected to the PCI bus is considered as the PCI bus slave device, whereas the part of the Universe IIb which is connected to the VMEbus is operating as VMEbus master.

Note: **The words described below initialize and control the VMEbus master interface but most of them are only available if the VMEbus device has been selected (FCode package). The only exceptions from this rule are the following words: `vme-memmap`, `vme-free-virtual` and `.vme-master-ranges`.**

To select the VMEbus device node, see section 6.1.16 “VMEbus Device Node Specific Words” on page 150**.**

`#vme-ranges` (— *#vme-ranges*) returns the number *#vme-ranges* of available register sets which are used to control accesses to the VMEbus (for the Universe IIb 8 sets at most).

`vme-master-ena` (*range#* —) enables the address decoding associated with the range number *range#* to access the VMEbus, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation.

`vme-master-dis` (*range#* —) disables the address decoding associated with the range number *range#* to access the VMEbus, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation.

`vme-master-wp-ena` (*range#* —) enables *write posting* within the VMEbus address range associated with the range number *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation.

`vme-master-wp-dis` (*range#* —) disables *write posting* within the VMEbus address range associated with the range number *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation.

`vme-supervisor!` (`true | false` —) selects the mode in which the VMEbus is being accessed. When `true` is passed to the command, the VMEbus is accessed in privileged mode. Otherwise, i.e. `false` is passed to the command, the VMEbus is accessed in non-privileged mode. The mode selected with this command applies to all ranges used to access the VMEbus.

`vme-master-cap@` (*range# — data-capability address-capability*) returns the address and data capabilities associated with the range number *range#* which are used when the VMEbus is accessed, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation. For *data-capability* and *address-capability* values see `vme-master-cap!`.

`vme-master-cap!` (*data-capability address-capability range#* —) defines the address and data capabilities associated with the range number *range#* which are used when the VMEbus is accessed, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation. *data-capability* and *address-capability* may be one of the values listed below.

Table 70 Data and Address Capabilities

Value and description			
data-capability		***address-capability***	
000_2	cap-d8	000_2	cap-a16
001_2	cap-d16	001_2	cap-a24
010_2	cap-d32	010_2	cap-a32
011_2	cap-d64	011_2	reserved
100_2	b-cap-d8	100_2	reserved
101_2	b-cap-d16	101_2	CSR
110_2	cap-blt	110_2	User 1
111_2	cap-mblt	111_2	User 2

`/pci-range` (*range# — size*) , available with OpenBoot 3.10.4 or higher, returns the *size* of the range associated with the range number *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation.

`pci-range` (*range# — size*) , available until OpenBoot 3.10.1, see "`/pci-range` (*range# — size*)" above.

`pci-slave-range@` (*range# — phys.lo phys.mid phys.hi size*) returns the PCI bus slave parameters associated with the range identified by *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master op-

eration. The parameters returned by the command specify the PCI bus address range to be accessed to reach the VMEbus. The address range is represented by *phys.lo phys.mid phys.hi* and *size*.

`pci-slave-range!` (*phys.lo phys.mid phys.hi size range#* —) sets the PCI bus slave parameters associated with the range identified by *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus master operation. The parameters passed to the command specify the PCI bus address range to be accessed to reach the VMEbus. The address range is represented by *phys.lo phys.mid phys.hi* 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#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the 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. For *data-capability* and *address-capability* values see `vme-master-range!`.

`vme-master-range!` (*addr data-capability address-capability size range# —*) sets the VMEbus master capabilities associated with the range number identified by *range#*, *range#* = `0`...`7`. Each value specifies one of the eight 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 below:

Value and description of			
data-capability		***address-capability***	
000_2	`cap-d8`	000_2	`cap-a16`
001_2	`cap-d16`	001_2	`cap-a24`
010_2	`cap-d32`	010_2	`cap-a32`
011_2	`cap-d64`	011_2	`reserved 1`
100_2	`b-cap-d8`	100_2	`reserved 2`
101_2	`b-cap-d16`	101_2	`CSR`
110_2	`cap-blt`	110_2	`User 1`
111_2	`cap-mblt`	111_2	`User 2`

`.vme-master-ranges` (—) displays the current settings of all register sets of the VMEbus master interface.

`vme-memmap` (*phys.lo phys.hi size — vaddr*) initializes a VMEbus master window. The PCI bus area from 1 GByte to 2 GByte (address range 4000.0000_{16} …$7FFF.FFFF_{16}$, PCI bus memory space) is used on the PCI bus slave window side. This area is administrated dynamically. `vme-memmap` scans for an unused PCI bus range and allocates it for a PCI bus slave / VMEbus master window. The VMEbus address range is represented by *phys.lo, phys.hi* and *size*. *vme-memmap* returns the virtual address *vaddr* to access the VMEbus.

`vme-free-virtual` (*vaddr size — *) removes a VMEbus master window. The PCI bus area from 1 GByte to 2 GByte (address range 4000.0000_{16} …$7FFF.FFFF_{16}$, PCI bus memory space) is used on PCI bus slave window side. This area is administrated dynamically. `vme-free-virtual` scans for a used PCI bus range and deletes its setup for this specific VMEbus master window. The window is represented and identified by its virtual address *vaddr* and size *size*.

Example

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}.

```
ok -1 value vmebus
ok h# 4080.0000 vmea32d32 1meg 2 * vme-memmap to vmebus
ok
```

The first command creates a variable in the VMEbus dictionary which holds the virtual address to access the VMEbus. This variable may be used to access the VMEbus area using the OpenBoot commands to read and write data.

The second command initializes 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 `vmea32d32` which enables the VMEbus master interface to access 8-bit data, 16-bit data, and 32-bit data within the VMEbus area. The address capability is defined using the predefined constant `vmea32d32` that enables the VMEbus interface to access the extended address space (A32) of the VMEbus.

When the PCI-to-VMEbus translation defined by the contents of the register sets is no longer used, 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.

```
ok vmebus 1meg 2* vme-free-virtual
ok
```

6.1.14 VMEbus Slave Interface

The Universe IIb provides 8 register sets to control any VMEbus slave operation (PCI-master). Each set may be used to address a certain address range within the VMEbus address space. The registers within a set are called VSI*n*-CTL,VSI*n*-BS, VSI*n*-BD and VSI*n*-TO with *n* = 0...7 (VSI = VMEbus Slave Image).

When the VMEbus interface of the CPU board is accessed from the VMEbus, the part of the Universe IIb connected to the VMEbus is considered as VMEbus slave device. The part of the Universe IIb which is connected to the PCI bus is operating as the PCI bus master. The names of the commands available to control the VMEbus master interface reflect this.

Note: **The words described below initialize and control the VMEbus slave interface but most of them are only available if the VMEbus device has been selected (FCode package). The only exceptions from this rule are the following words: `set-vme-slave` and `reset-vme-slave`.**
To select the VMEbus device node, see section 6.1.16 "VMEbus Device Node Specific Words" on page 150.

`vme-slave-ena` (*range#* —) enables the address decoding associated with the range number *range#* to allow accesses from the VMEbus, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`vme-slave-dis` (*range#* —) disables the address decoding associated with the range number *range#* to allow accesses from the VMEbus, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`vme-slave-wp-ena` (*range#* —) enables write posting within the VMEbus slave address range associated with the range number *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`vme-slave-wp-dis` (*range#* —) disables write posting within the VMEbus slave address range associated with the range number *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`/vme-range` (*range# — size*) available with OpenBoot 3.10.4 or higher, returns the *size* of the extended (A32) slave interface associated with the range number *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`/vme-a32-range` (*range# — size*) available until OpenBoot 3.10.1, see "`/vme-range` (*range# — size*)" above.

`vme-a24-slave-range@` (*range# — paddr size*) available with OpenBoot 3.10.4 or higher, returns the VMEbus base address *paddr* and the *size* of the A24 slave window associated with the range identified by *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`vme-a32-slave-range@` (*range# — paddr size*) available with OpenBoot 3.10.4 or higher, returns the VMEbus base address *paddr* and the *size* of the A32 slave window associated with the range identified by *range#*, *range#* = `0`...`7`. *range#* specifies the register set controlling the VMEbus slave access.

`vme-a24-slave-range!` (*paddr size range# —*) available with OpenBoot 3.10.4 or higher, sets the VMEbus base address *paddr* and the size *size* of the A24 slave window associated with the range identified by *range#*. The value of *range#* may be one of the values in the range zero through seven. Each value specifies one of the eight register sets controlling any VMEbus slave access.

`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#*. The value of *range#* may be one of the values in the range zero through seven. Each value specifies one of the eight register sets controlling any VMEbus slave access.

`pci-a24-master-range@` (*range# — vaddr size*) available with OpenBoot 3.10.4 or higher, returns the PCI bus master parameters *vaddr* and *size* of the A24 slave window associated with the range identified by *range#, range#=0...7*. *range#* specifies the register set controlling the VMEbus slave access.

`pci-a24-master-range!` (*vaddr size range# —*) available with OpenBoot 3.10.4 or higher, resets the PCI bus master parameters *vaddr* and *size* of the A24 slave window associated with the range identified by *range#*. The value of *range#* may be one of the values in the range *0...7*. Each value specifies one of the 8 register sets controlling any VMEbus slave access.

`pci-a32-master-range@` (*range# — vaddr size*) available with OpenBoot 3.10.4 or higher, returns the PCI bus master parameters *vaddr* and *size* of the A32 slave window associated with the range identified by *range#, range#=0...7*. *range#* specifies the register set controlling the VMEbus slave access.

`pci-a32-master-range!` (*vaddr size range# —*) available with OpenBoot 3.10.4 or higher, resets the PCI bus master parameters *vaddr* and *size* of the A32 slave window associated with the range identified by *range#*. The value of *range#* may be one of the values in the range *0...7*. Each value specifies one of the 8 register sets controlling any VMEbus slave access.

`set-vme-slave` (*phys.lo phys.hi size — vaddr*) enables a VMEbus slave interface. The VMEbus address range is represented by *phys.lo phys.hi* and *size.*
`set-vme-slave` returns the virtual address *vaddr* to access local memory.

`reset-vme-slave` (*vaddr size* —) disables a VMEbus slave interface. The VMEbus address range *size* is represented by its virtual address *vaddr* and size *size.* The allocated slave memory is deallocated.

Example

The following example lists all steps to be taken, to initialize the VMEbus interface for A32 accesses from the VMEbus beginning at address 2340.0000_{16} and ranging to $235F.FFFF_{16}$.

```
ok -1 value mymem
ok h# 2340.0000 vmea32d32 1meg 2 *0 set-vme-slave to mymem
ok
```

The first command creates a variable in the VMEbus dictionary which holds the virtual address to access the onboard memory. This variable may be used to access the mapped memory area using the OpenBoot commands to read or write data.
The second command initializes the VMEbus slave interface. It sets the data and address capabilities, the VMEbus address and the size of the area to be accessed.
The data and address capability is set by the predefined constant vmea32d32 that enables the VMEbus slave interface with 32 bit data in the extended address space of the VMEbus. For releasing the VMEbus slave interface and destroying all the existing mappings, execute the following:

```
ok mymem 1meg 2 * reset-vme-slave
ok
```

6.1.15 DMA Controller Support (OpenBoot 3.10.6 and above)

Note: **The commands listed below are available to access and control the VMEbus status signals. The words described below are only available if the VMEbus device is selected.**
To select the VMEbus device node, section 6.1.16 “VMEbus Device Node Specific Words” on page 150.

The commands listed below are available to control the DMA controller of the Universe, as well as to get information about the actual state of the DMA controller.

`dma-ena` (—) enables the DMA controller and starts a DMA transaction.

`dma-dis` (—) disables the DMA controller and stops the DMA transaction currently running.

`dma-halt` (—) halts the DMA transaction currently running.

`dma-vme-cap@` (— *data-capability address-capability*) returns the *data-capability* and *address-capability* currently defined for the DMA transaction..

`dma-vme-cap!` (*data-capability address-capability* —) sets the *data-capability* and *address-capability* 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-d64`	reserved
100_2	reserved	reserved
101_2	reserved	reserved
110_2	`cap-blt`	reserved
111_2	`cap-mblt`	reserved

`dma-count@` (— *transfer-count*) returns the current state of the transfer count. The value *transfer-count* indicates the number of bytes to be transferred by the DMA controller.

`dma-count!` (*transfer-count* —) sets the number of bytes — *transfer-count* — to be transfered by the DMA controller.

`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 transaction has been terminated successfully. It returns the value `true` when the DMA transaction has been terminated successfully. Otherwise the value `false` is returned to indicate that the DMA transaction has been terminated due to a fail state, or because the DMA transaction is still in progress.

`dma-error-terminated?` (— `true | false`) checks whether the DMA transaction has been terminated unsuccessfully. It returns the value `true` when the DMA transaction has been terminated due to a fail state. Otherwise the value `false` is returned to indicate that the DMA transaction has been terminated due to normal termination, or because the DMA transaction is still in progress.

`.dma-stat` (—) displays the current state of the DMA Status Register.

Example

```
ok .dma-stat
P_ERR:0 VERR:0 LERR:0 NT:0 HALT:0 RUN:0
ok
```

The field `P_ERR` indicates a protocol error. It is asserted if the PCI master interface is disabled or the lower three bits of the PCI address and the VME address differ (`0` = No Error).

The fields `VERR` and `LERR` are set if a DMA VMEbus or PCI bus Error occurs (`0` = No Error).

The fields `NT, HALT,` and `RUN` reflect the current state of the DMA controller. When the `NT` field is set to one (`1`) the DMA controller terminated successfully (normal termination). In case that the `HALT` field is set to one (`1`) 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`). If one of the fields is cleared (`0`) the DMA controller is not in the particular state.

`dma-mem>vme` (*src-addr dest-addr count* — `true | false`) initiates a DMA transfer from the PCI bus to the VMEbus and waits for termination of the DMA transaction.

The amount of bytes given by *count* are transfered from *src-addr* — an address area on the PCI bus (PCI device address) — to *dest-addr* — an address area on the VMEbus (physical address). The command returns `false` when all data have been transfered successfully. Otherwise `true` is returned to indicate that an error occurred during the DMA transaction.

Example

The example below transfers 1 MByte of data from onboard memory to VMEbus memory:

```
ok select vme
ok 1meg dma-alloc-mem value my-mem
ok my-mem 1meg false dma-map-in to pci-devaddr
ok my-mem 1meg 12345678 lfill
ok cap-mblt cap-a32 dma-vme-cap!
ok pci-devaddr a000.0000 1meg dma-mem>vme .
0
ok
```

The steps of the example above are described in detail in the following:

- **`select vme`**

  creates an instance chain for the device-node `vme.`

- **`1meg dma-alloc-mem value my-mem`**

  allocates `1meg` of memory within the DMA address space and stores the virtual address to the value `my-mem`

- **`my-mem 1meg false dma-map-in to pci-devaddr`**

  converts the virtual address range specified by `my-mem` and the size `1meg` into an address suitable for DMA on the PCI bus and stores the address to the value `pci-devaddr`.

  Instead of executing the two lines above, the command syntax **`b000.0000 vmea32d32 1meg set-vme-slave value my-mem`** can be used for allocating and mapping DMA memory. Thus the value `pci-devaddr` is initiated too.

- **`my-mem 1meg 12345678 lfill`**

  fills 1 MByte of memory at address `my-mem` with the pattern `12345678`.

- **`cap-mblt cap-a32 dma-vme-cap!`**

  sets the data and address capabilities for the DMA transfer initiated in the line followed.

- **`pci-devaddr a000.0000 1meg dma-mem>vme`**
  transfers 1 MByte of data from onboard memory at address `pci-devaddr` to a VMEbus region at address `a000.0000`. `false` is returned, if the transfer is terminated successfully, else `true` is returned.

`dma-vme>mem` (*src-addr dest-addr count* — `true | false`) initiates a DMA transfer from the VMEbus to the PCI bus and waits for the termination of the DMA transaction.

The amount of bytes given by *count* are transfered from *src-addr* — an address area on the VMEbus (*physical* address) — to *dest-addr* — an address area on the PCI bus (*PCI device* address). The command returnes the value `false` when all data have been transfered successfully. Otherwise the value `true` is returned to indicate that an error occurred during the DMA transaction.

Example

The example below transfers 1 MByte of data from VMEbus memory to onboard memory:

```
ok select vme
ok 1meg dma-alloc-mem value my-mem
ok my-mem 1meg false dma-map-in to pci-devaddr
ok a000.0000 vmea32d32 1meg vme-memmap value vme
ok my-vme 1meg 12345678 lfill
ok cap-mblt cap-a32 dma-vme-cap!
ok a000.0000 pci-devaddr 1meg dma-vme>mem .
0
ok
```

To explain the example above in detail:

- `select vme`

  creates an instance chain for the device-node `vme.`

- **`1meg dma-alloc-mem value my-mem`**

  allocates 1 MByte of memory within the DMA address space and stores the virtual address to the value `my-mem`

- **`my-mem 1meg false dma-map-in to pci-devaddr`**

  converts the virtual address range specified by `my-mem` and the size 1 MByte into an address suitable for DMA on the PCI bus and stores the address to the value `pci-devaddr`.

  Instead of executing the two lines above, the command syntax: **`b000.0000 vmea32d32 1meg set-vme-slave value my-mem`** can be used for allocating and mapping DMA memory. Thus the value `pci-devaddr` is initiated too.

- **`a000.0000 vmea32d32 1meg vme-memmap value my-mem`**

  initializes a VMEbus Master window at the address range represented by the triple phys-lo: a000.0000, phys-hi: vmea32d32 and the size: 1meg. The virtual address returned is stored to the value my-vme. For more details see “VMEbus Master Interface” on page 139.

- **`my-vme 1meg 12345678 lfill`**

  fills 1 MByte of VMEbus memory at address `my-vme` with the pattern `12345678`.

- **`cap-mblt cap-a32 dma-vme-cap!`**

  sets the data and address capabilities explained in table 70 “Data and Address Capabilities” on page 140 for the DMA transfer initiated in the line followed.

- **`a000.0000 pci-devaddr 1meg dma-vme>mem`**
  transfers 1 MByte of data from VMEbus memory at address `a000.0000` to onboard memory at address `pci-devaddr`. `False` is returned, if the transfer is terminated successfully, else `true` is returned.

6.1.16 VMEbus Device Node Specific Words

The OpenBoot device tree contains the device node for the VMEbus interface and is called `vme`. It is a child device of the device node `/pci@1f,0`. The full pathname of the VMEbus interface device node is displayed by the command `show-devs`. The device alias `vme` is available as an abbreviation of the VMEbus interface device-path.

Besides the words listed below, the vocabulary of the VMEbus device includes the standard commands recommended for a hierarchical device. The words of this vocabulary are only available if the VMEbus device has been selected as shown below:

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

`dev vme` selects the VMEbus device and makes it the current node. The word `words` displays the names of the methods of the VMEbus device. Thereafter, the method `selftest` is called and the value returned by this method is displayed. `device-end` unselects the current device node, leaving no node selected.

The following methods are defined in the vocabulary of the VMEbus device:

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

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

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

`selftest` (— *error-number*) performs a test of the VMEbus interface, and returns an *error-number* to report the course of the test. In the case that the device has been tested successfully, `0` is returned. Otherwise, it returns a specific error number to indicate a certain fail state.

`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.

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

`dma-map-in` (*vaddr size cachable?* — *devaddr*) converts a virtual address range, which is specified by *vaddr* and *size*, into an address *devaddr* suitable for direct memory access on the bus. The virtual memory must be already allocated by `dma-alloc`. Since the SPARC/CPU-50 does not support caching, the *cachable?* flag is ignored.

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

6.2 System Configuration

This section consists of the following parts:

- Section 6.2.1 ‘Standard Initialization of the VMEbus Interface’
- section 6.2.2 “System Configuration Register Accesses” on page 152,
- section 6.2.3 “LEDs, Seven Segment Display and Rotary Switch” on page 156,
- section 6.2.4 “ID PROM” on page 158,
- section 6.2.5 “Viewing the Switch Status and Controlling the Temperature Sensors” on page 159,
- section 6.4 “NVRAM Configuration Variables” on page 171.

6.2.1 Standard Initialization of the VMEbus Interface

Universe IIb Registers

The standard initialization during power up contains the following:

- Disable SYSFAIL* at register VCSR_CLR,SYSFAIL*
- Disable all PCI bus Slave/Master Window control/BS/BD registers
- Clear bit MISC_CTL,BI to exit Universe IIb from BI-mode
- Universe IIb default requester setting (modifiable by configuration variables):

  - Fair request mode
  - Request at level 3
  - Release when done (RWD)

- Universe IIb default arbiter setting (modifiable by configuration variables) providing round robin arbitration without arbitration timeout (ROR)

6.2.2 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.

`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 – a 2-bit data – of the Boot ROM Size Control Register.

`boot-rom-size-ctrl!` (*byte* —) stores the 2-bit data in the Boot ROM Size Control Register.

`user-rom-size-ctrl@` (— *byte*) returns the contents – a 2-bit data – of the User ROM Size Control Register.

`user-rom-size-ctrl!` (*byte* —) stores the 2-bit data in the User ROM Size Control 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.

`lca-id@` (— *byte*) returns the contents – an 8-bit data – of the LCA ID Register.

`supio_pwdn@` (— `true` | `false`) returns the state of the SuperI/O Power Down Mode Register. The SuperI/O is put into Power Down Mode if `true` is returned.

`supio_pwdn!` (`true` | `false` —) controls the Power Down Mode Register of the SuperI/O. The SuperI/O can be put into Power Down Mode if `true` is used with this command.

`eject_fd@` (— `true` | `false`) returns the state of the automatic Floppy Disk Eject Register. The floppy disk is ejected if `true` is returned.

`eject_fd!` (`true` | `false` —) controls the automatic Floppy Disk Eject Register. The floppy can be ejected if `true` is used with this command. Immediately after setting the Floppy Disk Eject Register to eject the floppy via `eject_fd! true`, it should be cleared again via `eject_fd! false`.

`reset_stat_clr!` (`true` | `false` —) controls the Reset Status Control Register. The register is used to clear the status bits in the Reset Status Register after a reset has occurred. All status bits are cleared if `true` is used with this command. Once the Reset Status Control Register is set (`true`), it is cleared automatically.

`boot_wp@` (— `true` | `false`) returns the state of the Boot Flash Write Protection Switch (SW4-3). The boot flash is write protected if `true` is returned.

`user_wp@` (— `true` | `false`) returns the state of the User Flash Write Protection Switch (SW4-4). The user flash is write protected if `true` is returned.

`scsi_front@` (— `true` | `false`) returns the state of the SCSI Front Panel Termination Switch (SW5-1). The SCSI front panel termination is automatic if `true` is returned. In this case the termination is only enabled if no SCSI cable is connected to the front panel. Otherwise, the SCSI front panel termination is disabled.

`scsi_bp@` (— `true` | `false`) returns the state of the SCSI Backplane Termination Switch (SW5-2). The SCSI backplane termination is disabled if `true` is returned.

`vsys_resout@` (— `true` | `false`) returns the state of the VMEbus SYSRESET output enable Switch (SW800-4). The VME SYSRESET output is enabled if `true` is returned.

`vsys_resin@` (— `true` | `false`) returns the state of the VMEbus SYSRESET input enable Switch (SW800-3). The VME SYSRESET input is enabled if `true` is returned.

`man_slot1@` (— `true` | `false`) returns the state of the VMEbus Manual Slot 1 Enable Switch (SW800-2). The manual slot 1 enable switch is in “off” position if `true` is returned.

`auto_slot1@` (— `true` | `false`) returns the state of the VMEbus Automatic Slot 1 Enable Switch (SW800-1). The automatic slot 1 enable switch is in “off” position if `true` is returned.

`ie_sysfail_enum@` (— `true` | `false`) returns the state of the SYSFAIL- Interrupt Enable Register. The SYSFAIL- interrupt is enabled if `true` is returned. After Reset SYSFAIL- is disabled.

`ie_sysfail_enum!` (`true` | `false` —) controls the SYSFAIL- Interrupt Enable Register. The SYSFAIL- interrupt is enabled if `true` is used with this command. After Reset SYSFAIL- is disabled.

`ip_sysfail_enum@` (— `true` | `false`) returns the state of the SYSFAIL- Interrupt Pending Register due to a low to high transition . The SYSFAIL- interrupt is pending if `true` is returned. In that case an interrupt is generated to the processor provided that the SYSFAIL- interrupt is enabled by the SYSFAIL- Interrupt Enable Register. This signal is rising edge sensitive.

`ip_sysfail_enum!` (`true` | `false` —) controls the SYSFAIL- Interrupt Pending Register. The SYSFAIL- interrupt is pending if `true` is used with this command. In that case an interrupt is generated to the processor provided that the SYSFAIL- interrupt is enabled by the SYSFAIL- Interrupt Enable Register. This signal is rising edge sensitive.

`ie_acfail@` (— `true` | `false`) returns the state of the ACFAIL- Interrupt Enable Register. The ACFAIL- interrupt is enabled if `true` is returned. After Reset the ACFAIL- is disabled.

`ie_acfail!` (`true` | `false` —) controls the ACFAIL- Interrupt Enable Register. The ACFAIL- interrupt is enabled if `true` is used with this command. After Reset the ACFAIL- is disabled.

`ip_acfail@` (— `true` | `false`) returns the state of the ACFAIL- Interrupt Pending Register due to a low to high transition. The ACFAIL- interrupt is pending if `true` is returned. In that case an interrupt is generated to the processor provided that the ACFAIL- interrupt is enabled by the ACFAIL- Interrupt Enable Register. This signal is rising edge sensitive.

`ip_acfail!` (`true` | `false` —) controls the ACFAIL- Interrupt Pending Register. The ACFAIL- interrupt is pending if `true` is used with this command. In that case an interrupt is generated to the processor provided that the ACFAIL- interrupt is enabled by the ACFAIL- Interrupt Enable Register. This signal is rising edge sensitive.

`ie_wdt@` (— `true` | `false`) returns the state of the Watchdog Timer Interrupt Register. The watchdog timer interrupt is enabled if `true` is returned. After Reset the Watchdog Timer Interrupt is disabled.

`ie_wdt!` (`true` | `false` —) controls the Watchdog Timer Interrupt Register. The watchdog timer interrupt is enabled if `true` is used with this command. After Reset the Watchdog Timer Interrupt is disabled .

`is_wdt@` (— `true` | `false`) returns the state of the output signal WDO of the MAX815 Watchdog Timer device. The watchdog timer intervall is not expired if `true` is returned.

`wdi!` (`true` | `false` —) controls the Watchdog Timer Trigger Input Register. The watchdog timer is cleared by changing the value of the WDI input. If the watchdog timer intervall expires, the watchdog timer device MAX815 activates its WDO output and a Watchdog timer interrupt may be generated. To start the watchdog timer it is necessary to trigger WDI once.

`ie_temp@` (— `true` | `false`) returns the state of the Temperature Control Interrupt Register. The temperature control interrupt is enabled if `true` is returned. After Reset the temperature control interrupt is disabled.

`ie_temp!` (`true` | `false` —) controls the Temperature Control Interrupt Register. The temperature control interrupt is enabled if `true` is used with this command. After Reset the temperature control interrupt is disabled.

`ip_temp@` (— `true` | `false`) returns the state of both output signals of the temperature sensors. A temperature interrupt is pending and `true` is returned if one of them has put its output signal active. An interrupt is generated to the processor provided that the Temperature Enable Interrupt Register is enabled.

`is_temp1@` (— `true` | `false`) returns the state of the output signal of the first digital temperature sensor and thermal watchdog. The state of the temperature's output signal is high (`1`) if `true` is returned.

`is_temp2@` (— `true` | `false`) returns the state of the output signal of the second digital temperature sensor and thermal watchdog. The state of the temperature's output signal is high (`1`) if `true` is returned.

6.2.3 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#*. *led#* = `0` specifies the first user LED, `1` specifies the second user LED. The command only considers the state of bit `0` of the value *led#*. *colour* and *freq* define the colour of the LED and the frequency at which the LED is blinking. The following constants are defined for *colour*: `black` (= LED is turned off), `green`, `red`, and `yellow`. The following constants are defined for *freq*: `no-blinking` (= LED is turned on permanently), `slow`, `moderate`, and `fast`.

Example

The following command makes the second user LED blink with a moderate frequency in red:

```
ok red moderate 1 led!
```

`led-on` (*led#* —) turns on the user LED identified by *led#*. *led#* = `0` specifies the first user LED, `1` specifies the second user LED. The command only considers the state of bit `0` of the value *led#*.

`led-off` (*led#* —) turns off the user LED identified by *led#*. *led#* = `0` specifies the first user LED, `1` specifies the second user LED. The command only considers the state of 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. *led#* = `0` specifies the first user LED, `1` specifies the second user LED. The command only considers the state of bit `0` of the value *led#*.
If the LED is turned on, `true` is returned; otherwise `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. *led#* = `0` specifies the first user LED, `1` specifies the second user LED. The command only considers the state of bit `0` of the value *led#*. Regardless of the colour 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 `0`…`15`. `0` corresponds to position `0` of the rotary switch, `1` corresponds to position `1`, and so forth.

6.2.4 ID PROM

Depending on the SPARC/CPU-50 variant under consideration there are several ID PROMs, all connected via an I^2C bus. At least 1 ID PROM is present, the one assembled on the base board. Each ID PROM if assembled is accessible as I^2C bus slave via the I^2C bus commands listed below:

`select-idprom` (*slv# — flag*) selects an ID PROM by adjusting the I^2C bus slave address. If the ID PROM to be selected is not accessible or not available, an error message is displayed. The resulting *flag* is `true` only if the ID PROM is present and the selection was successful. The number *slv#* of the I^2C bus device can be one of the values listed in the table below:

slv#	**I^2C bus slave address**	**Description**
x in the I^2C bus slave address depends on the action to be done: • x = `1` for read access and x = `0` for write access.		
`1`	`1010.000`x_2	ID PROM on base board
`2`	`1010.010`x_2	ID PROM on I/O-board

`i2c!` (*addr data i²c-slave-addr —*) transmits a byte *data* to the ID PROM which is identified by its *i²c-slave-addr*. For *i²c-slave-addr* values see the `select-idprom` description above. *addr* specifies the offset within the ID PROM's address range, at which *data* is stored in the ID PROM.

`i2c@` (*addr i²c-slave-addr — data*) reads a byte *data* from the ID PROM which is identified by its *i²c-slave-addr*. For *i²c-slave-addr* values see the `select-idprom` description above. *addr* specifies the offset within the ID PROM's address range, at which *data* is stored in the ID PROM.

`mem>idprom` (*src-addr dest-addr size —*) copies a number of bytes from the on-board memory to the ID PROM selected via `select-idprom`. The number of bytes to be copied is specified by *size*, *size* = `1` …`512`. The source data start at the virtual address *src-addr* in the on-board memory. The start address for the copied data in the ID PROM is specified by *dest-addr*, *dest-addr* = `0` …`511`.

`mem>idprom` (*src-addr dest-addr size —*) copies a number of bytes from the on-board memory to the ID PROM selected via `select-idprom`. The number of bytes to be copied is specified by *size*, *size* = `1` …`512`. The source data start at the virtual address *src-addr* in the on-board memory. The start address for the copied data in the ID PROM is specified by *dest-addr*, *dest-addr* = `0` …`511`.

`idprom>mem` (*src-addr dest-addr size* —) copies a number of bytes from the ID PROM which has been selected via `select-idprom` to the on-board memory. The number of bytes to be copied is specified by *size*, *size* = `1` …`512`. The source data start at the address *src-addr* in the ID PROM, *src-addr* = `0`…`511`. The start address for the copied data in on-board memory is specified by *dest-addr*.

6.2.5 Viewing the Switch Status and Controlling the Temperature Sensors

---

Note: **All temperature values are measured in °C and specified as *degree* degrees plus *1/10-degree* 1/10-degrees.**

`.cpu-switch-stat` (—) displays the current state of all switches on the SPARC/CPU-50.

`get-t` (*sensor# — 1/10-degree degree*) reads the result of the last temperature conversion for the specified temperature sensor *sensor#*, *sensor#* = `1`, `2`.

`get-t-low` (*sensor# — 1/10-degree degree*) returns the current setting of the THYST temperature register of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`.

`get-t-high` (*sensor# — 1/10-degree degree*) returns the current setting of the TOS temperature register of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`.

`get-conf` (*sensor# — byte*) reads the status register of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`. The contents of the status register data is left on the evaluation stack.

`set-t-high` (*1/10-degree degree sensor# —*) sets the TOS temperature register of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`.

`set-t-low` (*1/10-degree degree sensor# —*) sets the THYST temperature register of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`.

`set-conf` (*byte sensor# —*) stores the value *byte* in the configuration register of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`. The contents of the status register data is left on the evaluation stack.

`watch-temperature` (—) puts both temperature sensors into interrupt mode and initiates continous temperature conversion.

`monitor-t` (*sensor# —*) monitors the actual temperature of the temperature sensor specified by *sensor#*, *sensor#* = `1`, `2`.

6.3 BusNet Support (OpenBoot 3.10.6 and above)

This chapter describes in detail the OpenBoot BusNet support features.

6.3.1 Limitations

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

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

6.3.2 Loading Programs

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 2 commands provided by OpenBoot:

ok boot *device-specifier argument*
or
ok 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)

6.3.3 BusNet Device Properties

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`.

Caution

A device's properties identify the characteristics of the package and its associated physical device. The BusNet device is characterized by the properties described below - these properties are static:

`name` 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` 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 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 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 VMEbus address of the participant designated as master. The property's size is 32 bits (integer).

`bn-master-space` specifies the address 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 VMEbus address of the participant's own BusNet region. The property's size is 32 bits (integer).

`bn-p-space` specifies the address space in which the participant's own BusNet region is acces-sible. 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 (0 ... 31). The property's size is 32 bits (integer).

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

`bn-p-mbox-space` specifies the address 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?` 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.

6.3.4 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. The BusNet device provides this method just 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 bytes received (not the number copied), or it returns `zero` if no packet is currently available.
The BusNet device driver copies only the data contained in the BusNet packet into memory and discards all information related to the BusNet-BusNet protocol.

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

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

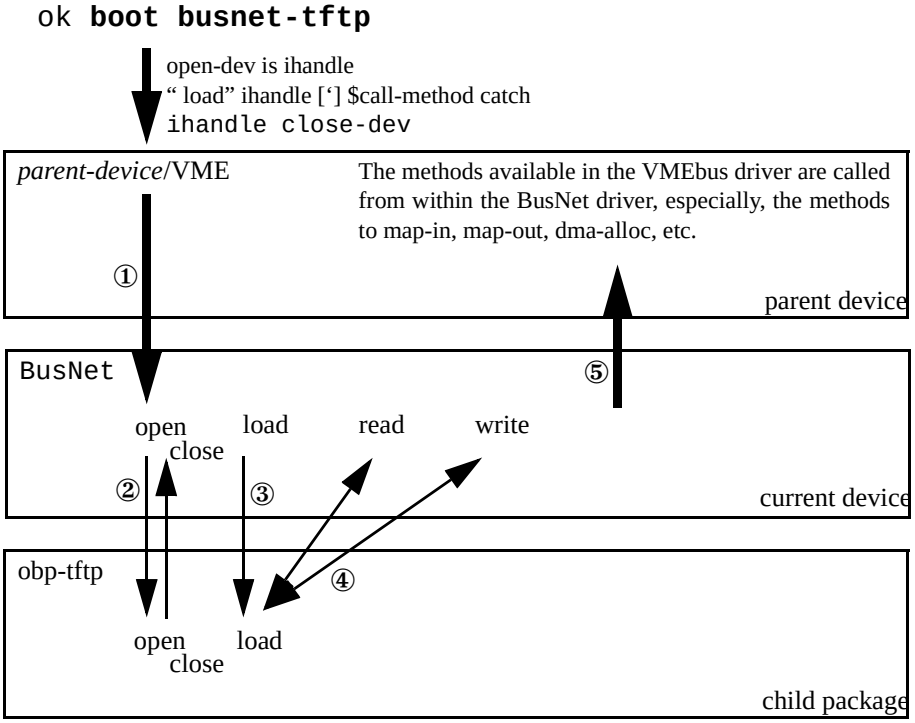
6.3.5 Device Operation

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 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 30 BusNet Booting



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 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.

6.3.6 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 is sent across the network from one BusNet participant to another participant. The following two definitions are available inside the BusNet node to initiate the transmission and receipt of data:

`demo-send-data` (*src-addr size dest-p#* —) sends the amount of data specified by *size* and stored beginning at the address *src-addr* to the participant identified by its logical BusNet address *dest-p#*.

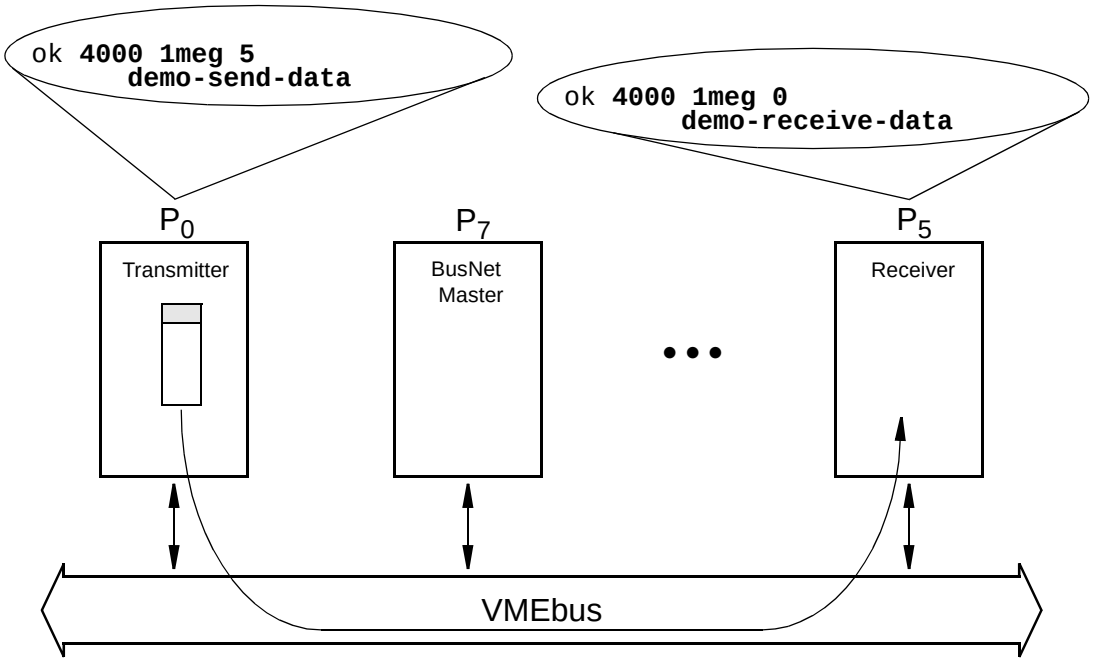
`demo-receive-data` (*dest-addr size src-p#* —) receives as much data as specified by *size* from the participant identified by its logical BusNet address *dest-p#* and stores it beginning at the address *dest-addr*.

Caution

When these commands are used to exchange data between two participants running OpenBoot, then a third participant must be available which provides BusNet master functionality. This is necessary because OpenBoot does not provide BusNet master functionality!

As shown in the figure below, three participants take part in communicating across the network using the BusNet protocol. The logical address of the participants are zero, seven and five. The participants P_0 and P_5 are executing OpenBoot, and the participant P_7 runs an operating system which is capable of providing BusNet master functionality — for example Solaris/SunOS, or VxWorks.

Figure 31 BusNet Demo Use



When a certain amount of data located in the onboard memory of the participant zero (P_0)—the transmitter—should be transfered to the participant five (P_5)—the receiver—then the following command must be used on the transmitter:

```
ok cd busnet
ok 4000 1meg 5 demo-send-data
```

This command initiates a transmission of 1Mbyte of data located at address 4000_{16} in the transmitter's onboard memory to the receiver. To enable the receiver to receive the data the following command must be used:

```
ok cd busnet
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 onboard memory.

Caution

To ensure proper operation of the data exchange, the size applied to the commands on the receiver and transmitter must be the same.

6.3.7 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 onboard 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 PCI bus cards. The second argument passed to this command— value one (1) in the example—specifies the

separation between FCode byte in general. Because the `bn-dload` command loads the FCode into onboard memory, the spacing is one (1).

Binary Executables

Executable binary programs to be loaded with `bn-dload` must be in the `a.out` format. To execute the binary program, the `go` command has to be used as follows:

```
ok go
```

When the program should be started again, the commands listed below have to be used:

```
ok init-program go
```

6.3.8 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
```

6.3.9 Booting from a VxWorks BusNet Server

Because VxWorks currently is not capable of resolving RARP requests, the NVRAM configuration parameters listed below must be set prior to loading an executable image.

`bn-rarp?` specifies whether the BusNet driver should scrutinize all outgoing packets and verifies whether an ethernet frame carries an RARP request.
The flag must be set to `true`, to enable the BusNet driver to check whether an ethernet frame contains a RARP request, and if so, it resolves the request and passes the response to the receiving part of the BusNet driver automatically. The ethernet frame is not sent across the network.
The BusNet driver uses the contents of the NVRAM configuration parameters `bn-master-ip-addr`, `bn-p-ip-addr`, `bn-master-en-addr`, and `bn-p-en-addr` to build up the appropriate response.

`bn-master-ip-addr` specifies the Internet Protocol (IP) Address of the master. The default value of this 32 bit configuration parameter is zero (`0`). The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-master-ip-addr 0x83030001
```

In the example, the internet address `131.3.0.1` (83030001_{16}) is assigned to the NVRAM configuration parameter.

`bn-p-ip-addr` specifies the Internet Protocol (IP) Address of the participant. The default value of this 32 bit configuration parameter is zero (0). The `setenv` command is used to set this configuration parameter as shown below:

```
ok setenv bn-p-ip-addr 0x83030002
```

In the example, the internet address `131.3.0.2` (83030002_{16}) is assigned to the NVRAM configuration parameter.

`bn-master-en-addr` specifies 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 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.SUN4U`).

6.3.10 Setting NVRAM Configuration Parameters

The SPARC CPCI-520/CPU-50 is equipped with the Universe IIb VMEbus Interface Chip which provides a mailbox register located in the *short* address space (A16) of the VMEbus. To enable the mailbox the following NVRAM configuration parameters must be set in addition to the NVRAM configuration parameters listed in the table below:

Table 71 NVRAM Configuration Settings

NVRAM Configuration Parameter	Default Value	Description

`bn-master-offset`
`bn-master-space`
`bn-master-access`
	00000000_{16}
	$3D_{16}$
	32_{16}
		privileged *standard* (A24) address range read/write/D32

Table 71 NVRAM Configuration Settings (cont.)

NVRAM Configuration Parameter	Default Value	Description

`bn-p-offset`
`bn-p-space`
`bn-p-access`
	00000000_{16}
	$3D_{16}$
	32_{16}
		privileged *standard* (A24) address range
		read/write/D32

`bn-p-mbox?`
`bn-p-mbox-offset`
`bn-p-mbox-space`
`bn-p-mbox-access`
	`true`
	0348_{16}
	$2D_{16}$
	10_{16}
		mailbox available (Universe IIb mailbox #0)
		offset of mailbox #0
		privileged *short* (A16) address range
		read/D8

6.4 NVRAM Configuration Variables

Besides the configuration variables already defined within the OpenBoot 3.x Command Reference, there are a few new ones, explained below:

Caution

The current state of these configuration parameters is displayed using the `printenv` command, and is modified using either the `setenv`, or the `set-default` command provided by OpenBoot.

6.4.1 PCI Bus Specific Configuration Variables

`pci-probe-list` (OpenBoot 3.10.6 and above) this variable is used to define the probe-list containing the number of devices to be probed at the primary PCI bus.

`pcib-io-probe-list` specifies the probe-list containing the device numbers to be probed on the I/O-board.

6.4.2 VMEbus specific configuration variables (OpenBoot 3.10.6 and above)

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

`vme-req-level` specifies the level on which the Universe IIb 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: BR3)

`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) or "rwd" (release when done). (default: `ror`)

`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), . (default: `rrs`)

6.4.3 BusNet specific configuration variables (OpenBoot 3.10.6 and above)

The OpenBoot provides the NVRAM configuration parameters as defined by the BusNet Protocol Specification 1.4.2.

`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 spac-es 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).

`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).

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

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

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

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

`bn-p-mbox?` 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 configura-tion 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`. 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`.

`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:10:1
```

This configuration parameter must be set when one of the configuration parameters `bn-arp?` or `bn-rarp?` are set to `true`.

6.5 Flash Memory Support

The boot and user flash available on the SPARC/CPU-50 are both programmable via the appropriate hardware setup and the appropriate OpenBoot commands (see section 6.5.1 “Flash Memory Programming” on page 175 which includes an example for programming the user flash on page 177).
This allows to program the user flash with an executable image and use the Force OpenBoot enhancements to load and execute such an image from user flash (see section 6.5.3 “Loading and Executing Programs from User Flash Memory” on page 180). For general information on NVRAM configuration parameters and methods available for flash memory, see section 6.5.2 “Flash Memory Device Node” on page 177.

Note: **Note the hardware dependencies for the flash memory driver documented in section 6.6 “Hardware Dependencies” on page 181.**

6.5.1 Flash Memory Programming

The commands listed below are available to access and program the onboard flash memory.

`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 display messages while erasing or programming the flash memories. Messages are not displayed after entering `flash-messages off`<eol>. They are displayed after entering `flash-messages on`<eol>. The virtual address is returned after entering `flash-messages` without further text.

`flash-va` (— *vaddr*) returns the virtual base address *vaddr* of the flash memory programming window. The address is only valid, if the flash memory has previously been selected using `select-flash`.

`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-50 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` | `BOOT` —) selects the flash memory to be programmed and prepares the selected flash memory for programming. No further words may follow in the same command line (see “Example: Programming the User Flash” on page 177). `USER` selects the user flash, `BOOT` the boot flash.
In detail, the number and size of the available flash devices 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. The address may be obtained by `flash-va`.

`move>flash` (*source-addr dest-addr count* —) programs the selected flash memory beginning at *dest-addr* with *count* number of Bytes which are fetched from *source-addr*. Use `select-flash` to select the flash.

`flash>move` (*source-addr dest-addr count* —) copies *count* number of Bytes from the selected flash memory beginning at *source-addr* to *dest-addr*. Use `select-flash` to select the 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*. Use `select-flash` to select the flash.

`erase-flash` (*device-number* —) erases the contents of a device of the selected flash memory. The device is identified by *device-number*. Device numbering starts at `0`. Use `select-flash` to select the flash.

`c!-flash` (*byte addr* —) stores *byte* at the location *addr* within the selected flash memory. Use `select-flash` to select the flash.

`w!-flash` (*half-word addr* —) stores the *half-word* (16 bits) at the location *addr* within the selected flash memory. Use `select-flash` to select the flash.

`l!-flash` (*word addr* —) stores the *word* (32 bits) at the location *addr* within the selected flash memory. Use `select-flash` to select the flash.

Example: Programming the User Flash

The user flash memory is prepared for programming by:

```
ok select-flash USER
USER flash memory is selected for programming
2048 Kbyte BOOT flash memory is available at 0xff550000
2048 Kbyte USER flash memory is available at 0xff350000
ok
```

`select-flash` informs the user that the user flash memory has been made accessible. It displays the available boot flash memory and user flash memory.

After the user flash has been selected, all following commands operate on the user flash. For example, to read data 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
```

Thereby, the contents of the entire user flash are copied to main memory beginning at address 10.0000_{16}.

A specific area within the selected flash memory is read as follows:

```
ok flash-va h# 6.8000 + h# 10.0000 h# 5.8c00 flash>move
ok
```

The source data start at address `flash-va` + 6.8000_{16}. They are copied to main memory starting at address 10.0000_{16}. And the amount of data copied is 363520 Bytes.

6.5.2 Flash Memory Device Node

The device tree of OpenBoot for the SPARC/CPU-50 contains a device node associated with the user and boot flash. The device alias `flash` is available as an abbreviation of the flash memory device path.
Thereby, it is possible to load an executable image stored in the available user flash into memory and start such an executable (see section 6.5.3 “Loading and Executing Programs from User Flash Memory” on page 180).

Vocabulary

The vocabulary of the flash memory device node includes the standard commands recommended for a *byte* device. This vocabulary is only available when the flash memory device node has been selected using one of the following 2 methods:

- `cd flash` selects the flash memory device and makes it the current node:

```
ok cd flash
```

- `select-dev` can also be used to select the flash memory device node. However, before using this command, the NVRAM configuration parameters `bootflash-#megs` and `bootflash-#devices` have to be set properly (see "NVRAM Configuration Parameters" on page 178).

After selecting the flash memory device node, the word `words` displays the names of the methods of the flash memory device.

```
ok words
close         open         selftest     reset     load
write-blocks  read-blocks  seek         write     read
max-transfer  block-size
```

To unselect the current device node, i.e. leaving no node selected, use `device-end`.

```
ok device-end
ok
```

NVRAM Configuration Parameters

The NVRAM configuration parameters listed below are available to control loading of an image from the user flash memory, i.e. boot from the user flash.

Note: **Note that the parameters indicate the purpose of booting from the user flash and therefore are called `bootflash`...**

The current state of the configuration parameters

- is displayed using `printenv`,
- and is modified using either `setenv`, or `set-default`.

`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.

Methods

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

`open` (— `true` | `false`) prepares the package for subsequent use. `true` is returned if the device has been opened successfully. Otherwise, `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 `0` as *error-number*.

`read` (*addr length* — *actual*) reads at most *length* Bytes from the flash memory and copies it to memory beginning at address *addr*. If *actual* is `0` or negative, the read failed. The value of *length* can be chosen independently of the device's block size. For information on the start address within the flash memory see the description of the `seek` command.

`write` (*addr length* — *actual*) discards the information passed to the command. It always returns `0` to indicate that the device does not support this function. However, this command is available to be standard compliant.

`seek` (*offset file#* — *error?*) seeks to Byte *offset* within the file identified by *file#*. An internal position counter is maintained and updated whenever a method is called to read data from or to store data in the flash memories. The position counter can be adjusted using the `seek` command.

- If *offset* and *file#* are both `0`, the internal position counter is reset to offset `0`.
- Otherwise, the value of *file#* is ignored and the value of *offset* is assigned to the internal position counter. A subsequent access to the flash memory then starts at the adjusted offset.

After a successful seek, *error?* is `0`, otherwise `-1` is returned to indicate the fail state.

`read-blocks` (*addr block# #blocks* — *#read*) reads *#blocks* number of blocks where each block is of length *block-size* Byte. The blocks are read from the device beginning at the device block *block#*. The read data are copied to memory at address *addr*. `read-blocks` 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. However, this command is available to be standard compliant.

`block-size` (— *bytes*) returns the current setting of the block size *bytes* in Byte. This always is the size of the flash memory programming window.

`max-transfer` (— *bytes*) returns the size *bytes* in Byte of the largest single transfer the device can perform. This always is a multiple of the value returned by `block-size`.

`load` (*addr — length*) reads a stand-alone program from the flash memory beginning at offset 0_{16} and stores it beginning at address *addr*. It returns the number of Bytes *length* read from the flash memory.

6.5.3 Loading and Executing Programs from User Flash Memory

Besides the ability to load and execute an executable image from disk, from a network component, or from other components, the SPARC/CPU-50 OpenBoot also provides a convenient way to load and execute an executable image from available user flash. The executable image to be loaded has to be:

- A binary image in `a.out` format
- A FORTH program,
- An FCode program.

Manual Loading

To load and execute an image from the flash memory use the device alias `flash` together with the `boot` command:

```
ok boot flash
```

Automatic Loading

The NVRAM configuration parameters `auto-boot?` or `boot-device` can be modified to determine whether or not the system loads an executable image automatically after a power-up cycle or system reset.

Example

Assume that the CPU board provides 1 user flash memory device which is 1 MByte in size. The following commands 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
```

6.6 Hardware Dependencies

Note: **To make use of the features described in this section it is necessary to take the following steps.**

- Copy the OpenBoot image from the boot PROM which is a read-only device to the boot flash EPROM (see section 6.6.1 “Copying an Image from the Boot PROM to the Boot Flash EPROM” on page 181)
- Use the boot flash EPROM for booting (SW6-2)

6.6.1 Copying an Image from the Boot PROM to the Boot Flash EPROM

The following section describes how to copy the OpenBoot image from the boot PROM into the boot flash EPROM. The copy is done via some memory space and makes use of the ability to switch between the boot PROM and boot flash EPROM. There are 2 versions of this procedure. The first version uses the OpenBoot command `plcc2tsop` which is available in OpenBoot versions 3.10.4 or greater whereas the second version applies in any case.

`plcc2tsop (—)` copies the contents of the boot PROM into the boot flash EPROM. A sample dialog is given below. To be successful, the boot flash write protection has to be disabled by setting SW4-3 to `ON`.

```
ok
ok plcc2tsop
    COPY OBP from PLCC to TSOP boot flash
    ------------------------------------

    Please enter 'y' to execute boot flash copy
    or any other key to abort
Step1: Mapping memory space at addr 0x1.0000
       Done
Step2: Erasing TSOP Boot flash ...
BOOT flash memory is selected for programming
2048 Kbyte BOOT flash memory is available at 0xff550000
No USER flash memory is available
       Done
Step3: Updating TSOP Boot flash ...
       Done
    COPY OBP into TSOP successful
    ----------------------------

    Now you have to switch SW 6-2 to the ' ON' position
    and thereafter reset the syste
ok
```

Note: **In case of an error during one of the 2 following procedures, select booting from the boot PROM, reset the system, and try again.**

1. Version: OpenBoot 3.10.4 or Higher

To copy the OpenBoot image from the boot PROM into the boot flash EPROM, do the following in case of OpenBoot version 3.10.4 or higher:

1. Make sure to have the CPU board installed with SW4-3 set to `ON` and and SW6-2 set to `OFF`.
2. Copy the OpenBoot image by entering:

   ```
   ok plcc2tsop
   ```

The OpenBoot image is copied from the boot PROM into the boot flash EPROM. To operate the CPU board, install the CPU board with the boot flash EPROM used for subsequential booting by setting SW6-2 to `ON`.

2. Version: any OpenBoot Version

In case of an OpenBoot version previous to 3.10.4, copy the OpenBoot image from the boot PROM into the boot flash EPROM as follows:

1. Make sure to have the CPU board installed with SW4-3 set to `ON` and and SW6-2 set to `OFF`.

2. Map the boot PROM and a specific memory space, and copy the OpenBoot image from the boot PROM into memory by entering:

```
ok 1.0000 0 1meg memmap value my-mem
ok 1ff.f000.0000 0 1meg memmap value my-plcc-flash
ok my-plcc-flash my-mem 1meg move
```

3. Select the boot flash EPROM for booting by entering:

```
ok 1ff.f160.0005 1d spacec@
ok 1 or 1ff.f160.0005 1d spacec!
```

4. Select the boot flash EPROM for programming, erase the boot flash EPROM, and copy the OpenBoot image into the boot flash EPROM by entering:

```
ok select-flash BOOT
BOOT flash memory is selected for programming
2048 Kbyte BOOT flash memory is available at 0xff550000
…
ok 0 erase-flash
Erasing selected flash memory ... passed!
ok my-mem boot-flash-va 1meg move>flash
```

The OpenBoot image is copied from the boot PROM into the boot flash EPROM. To operate the CPU board, install the CPU board with the boot flash EPROM used for subsequential booting by setting SW6-2 to ON.

6.6.2 Drop-in Drivers

OpenBoot supports drop-in drivers, i.e. drivers that may be added to OpenBoot during start-up time. The drop-in drivers are placed inside a specific drop-in driver area within the OpenBoot image so that they cannot be erased during a power-up once they are added.
The drop-in drivers are special FCode drivers having a unique drop-in driver header. At certain points during start-up, OpenBoot scans the drop-in driver area for specific drivers to be loaded at specific points. Each drop-in driver may be dedicated to a specific device and is loaded to the corresponding device node if the probing algorithm has identified a device whose device ID and vendor ID is equal to a specific drop-in driver.
OpenBoot contains commands to display all available drop-in drivers, to add them and to remove them. Thus the drop-in drivers can be loaded at any time to the OpenBoot image from net or other external media.
For example, drop-in drivers can automatically be loaded from a floppy disk when OpenBoot detects that a driver for a present PCI device is not present. In this case, the drop-in driver is loaded from floppy disk and is stored in the drop-in driver area. Thereby, after the next power-up sequence the driver is available in ROM.

6.6.3 Flash Memory Driver

The functionality of the flash memory driver depends on the type of boot flash used:

- If the CPU board boots from the boot flash EPROM (SW6-2 set to `ON`), the full functionality of the flash memory driver can be used as documented in section 6.5 “Flash Memory Support” on page 175.
- If the CPU board boots from the boot PROM (SW6-2 set to `OFF`), the driver cannot write to the boot flash because the boot PROM is a read-only device. However, a write access is done to identify the flash device when selecting the boot flash. For example, selecting the boot flash via `select-flash BOOT` results in the following error message:

  ```
  Flash memory either is not available or
  protected against writing!
  ```

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 address page.