MENMON 2nd Edition

Embedded Solutions

MENMON™

User Manual

mikro elektronik gmbh · nürnberg

About this Document

This user manual describes the global (non implementation specific) MENMON 2nd Edition release.

It is completed by implementation (board) specific descriptions included in the hardware user manual of the respective board.

History

Issue	Comments	Date
E1	First edition	2005-06-29
E2	General update, USB functions added	2008-04-10
E3	Added chapter 3.7; commands *AS* and *DI* no longer supported; FAT32 support added; USB commands updated	2008-07-29
E4	General update, EE-<param> – command added	2012-03-21

Conventions

This sign marks important notes or warnings concerning proper functionality of the product described in this document. You should read them in any case.

italics Folder, file and function names are printed in *italics*.

bold **Bold** type is used for emphasis.

`monospace` A `monospaced` font type is used for hexadecimal numbers, listings, C function descriptions or wherever appropriate. Hexadecimal numbers are preceded by "0x".

comment Comments embedded into coding examples are shown in green color.

hyperlink Hyperlinks are printed in blue color.

The globe will show you where hyperlinks lead directly to the Internet, so you can look for the latest information online.

IRQ#
/IRQ Signal names followed by "#" or preceded by a slash ("/") indicate that this signal is either active low or that it becomes active at a falling edge.

in/out Signal directions in signal mnemonics tables generally refer to the corresponding board or component, "in" meaning "to the board or component", "out" meaning "coming from it".

Vertical lines on the outer margin signal technical changes to the previous issue of the document.

Legal Information

Changes

MEN Mikro Elektronik GmbH ("MEN") reserves the right to make changes without further notice to any products herein.

Warranty, Guarantee, Liability

MEN makes no warranty, representation or guarantee of any kind regarding the suitability of its products for any particular purpose, nor does MEN assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages. TO THE EXTENT APPLICABLE, SPECIFICALLY EXCLUDED ARE ANY IMPLIED WARRANTIES ARISING BY OPERATION OF LAW, CUSTOM OR USAGE, INCLUDING WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR USE. In no event shall MEN be liable for more than the contract price for the products in question. If buyer does not notify MEN in writing within the foregoing warranty period, MEN shall have no liability or obligation to buyer hereunder.

The publication is provided on the terms and understanding that:

1. MEN is not responsible for the results of any actions taken on the basis of information in the publication, nor for any error in or omission from the publication; and

2. MEN is not engaged in rendering technical or other advice or services.

MEN expressly disclaims all and any liability and responsibility to any person, whether a reader of the publication or not, in respect of anything, and of the consequences of anything, done or omitted to be done by any such person in reliance, whether wholly or partially, on the whole or any part of the contents of the publication.

Conditions for Use, Field of Application

The correct function of MEN products in mission-critical and life-critical applications is limited to the environmental specification given for each product in the technical user manual. The correct function of MEN products under extended environmental conditions is limited to the individual requirement specification and subsequent validation documents for each product for the applicable use case and has to be agreed upon in writing by MEN and the customer. Should the customer purchase or use MEN products for any unintended or unauthorized application, the customer shall indemnify and hold MEN and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim or personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that MEN was negligent regarding the design or manufacture of the part. In no case is MEN liable for the correct function of the technical installation where MEN products are a part of.

Trademarks

All products or services mentioned in this publication are identified by the trademarks, service marks, or product names as designated by the companies which market those products. The trademarks and registered trademarks are held by the companies producing them. Inquiries concerning such trademarks should be made directly to those companies.

Conformity

MEN products are no ready-made products for end users. They are tested according to the standards given in the Technical Data and thus enable you to achieve certification of the product according to the standards applicable in your field of application.

RoHS

Since July 1, 2006 all MEN standard products comply with RoHS legislation.

Since January 2005 the SMD and manual soldering processes at MEN have already been completely lead-free. Between June 2004 and June 30, 2006 MEN's selected component suppliers have changed delivery to RoHS-compliant parts. During this period any change and status was traceable through the MEN ERP system and the boards gradually became RoHS-compliant.

WEEE Application

The WEEE directive does not apply to fixed industrial plants and tools. The compliance is the responsibility of the company which puts the product on the market, as defined in the directive; components and sub-assemblies are not subject to product compliance.

In other words: Since MEN does not deliver ready-made products to end users, the WEEE directive is not applicable for MEN. Users are nevertheless recommended to properly recycle all electronic boards which have passed their life cycle.

Nevertheless, MEN is registered as a manufacturer in Germany. The registration number can be provided on request.

Germany
MEN Mikro Elektronik GmbH
Neuwieder Straße 3-7
90411 Nuremberg
Phone +49-911-99 33 5-0
Fax +49-911-99 33 5-901
E-mail info@men.de
www.men.de

France
MEN Mikro Elektronik SA
18, rue René Cassin
ZA de la Châtelaine
74240 Gaillard
Phone +33 (0) 450-955-312
Fax +33 (0) 450-955-211
E-mail info@men-france.fr
www.men-france.fr

USA
MEN Micro, Inc.
24 North Main Street
Ambler, PA 19002
Phone (215) 542-9575
Fax (215) 542-9577
E-mail sales@menmicro.com
www.menmicro.com

Contents

1 General

1.1 Introduction

MENMON is the CPU board firmware that is invoked when the system is powered on.

The basic tasks of MENMON are:

- Initialize the CPU and its peripherals.
- Load the FPGA code (if applicable).
- PCI auto configuration.
- Perform self-test.
- Provide debug/diagnostic features on MENMON command line.
- Interaction with the user via touch panel/TFT display (if supported by FPGA).
- Boot operating system.
- Update firmware or operating system.

1.2 Primary and Secondary MENMON

There are two copies of MENMON in boot Flash:

- Primary MENMON
- Secondary MENMON

The main reason why two MENMONs are present is that the secondary MENMON can be updated by the user (by several methods, see Chapter 4.1.6 Program Update Menu on page 32 and Chapter 4.2.3 Program Update Features on page 40). If the update fails, the primary MENMON is still intact and can be used to download the secondary MENMON again.

Primary MENMON is invoked at system reset. After a minimum of initialization it checks if a valid secondary MENMON is present. If the primary MENMON detects the board's abort pin being asserted, it does not check for secondary MENMON.

If a valid secondary MENMON is found, control is immediately passed to it; secondary MENMON will perform all further initialization and boot-up. Secondary MENMON is protected by an XOR checksum algorithm and size validation.

If no valid secondary MENMON is found, primary MENMON either enters the command-line interface or continues execution, depending on MENMON system parameter *stignfault* (see Chapter 9 System Parameters on page 86).

The Flash sectors where primary MENMON resides are hardware sectors. On some boards they are protected and in this case cannot be overwritten by software.

2 Consoles

2.1 Introduction

MENMON consoles generally support two different types of text-based input:

- Screen-oriented menu interface
(see Chapter 4.1 Screen-Oriented Menu User Interface on page 21)
- Command-line interface
(see Chapter 4.2 Command-Line User Interface on page 36)

Full-graphics consoles such as a VGA card or frame-buffer device provide a full graphics screen visible during normal start-up and displaying the boot logo, and a text mode screen as the normal MENMON console.

Other consoles, such as a Telnet connection, only support text.

More than one console can be active at a time. All active consoles receive the same output, and MENMON accepts input from any active console.

One console can be defined as the debug console (receives all debug strings when BIOS debug level set != 0, see Chapter 4.2.7 Set Debug Options on page 45), another console may be used as the serial console for serial download. This must be a UART or Telnet console. One console can be used as a graphics console to display the boot logo or serve as a touch console. This must be a full-graphics console.

It is even possible to run MENMON without any console.

Every MENMON implementation includes a different range of console interface drivers to support different UARTs, graphics devices and input devices, or a touch panel, for instance.

2.2 Selecting Consoles

Each console device is assigned a unique CLUN (Controller Logical Unit Number) during MENMON start-up. You always need to specify CLUN numbers when selecting consoles.

You can select the active consoles by means of system parameters *con0..conN*. *N* is board-specific, but typically 3. *N* must be between 1 and 9.

In MENMON, the value of parameter *conX* can be set through an *EE-* command, e.g.

```
MenMon> EE-CON0 40
```

Changes to the console configuration have no effect until MENMON is restarted.

(For details on system parameters, see Chapter 9 System Parameters on page 86.)

Network consoles (Telnet and HTTP monitor page) are handled differently and do not need to be activated by *conX* parameters. They are active consoles whenever the corresponding server is started.

Table 1. *System parameters for console selection*

Parameter (alias)	Description	Default	User Access
con0..conN	CLUN of console 0..n. CLUN=0x00: disable CLUN=0xFF: Autoselect next available console *con0* is implicitly the debug console	Port-specific	Read/write
gcon	CLUN of graphics device to display boot logo CLUN=0x00: disable CLUN=0xFF: Autoselect first available graphics console	Port-specific, typically 0xFF	Read/write

MENMON handles situations where the user configured

- non-existent CLUNs
- CLUNs not belonging to consoles
- same CLUN configured more than once.

If any of the *clunX* parameters is configured incorrectly (and not 0), MENMON uses the default value for that parameter.

The serial console changes dynamically. It is assigned to the serial console on which the last input was received (UART or Telnet).

This feature is implemented so you can perform a *SERDL* serial download (YModem download) either on a UART port or Telnet port. YModem on Telnet is supported by Windows HyperTerminal or Linux *sz* program.

2.3 Testing the Console Configuration

Before permanently reconfiguring *conX* values, you can test what would happen using *CONS-ACT*.

This activates the specified CLUNs without saving this setting permanently, e.g. this command:

```
MenMon> CONS-ACT 8 9
```

This would activate COM1 and COM2 on EM4 as a console.

The setting made through *CONS-ACT* is active only until the system is restarted.

2.4 Use of Abort Pin for Default Console

If the board's abort pin is activated, MENMON ignores *conX* and baud settings and uses the hardwired defaults.

2.5 Operation without a Console

It is possible to work without any active console, if all *conX* parameters are set to 0. In this case, communication is still possible over network – if network servers haven't been disabled. The TFT/touch console can also be used only to display the boot logo.

2.6 Console-Related Commands

CONS	**Show active consoles**
CONS-ACT <clun1> [<clun2>] ...	**Test console configuration**
CONS-GX <clun> ...	**Test graphics console**

2.7 Example Console Configurations

2.7.1 Example for EM4

On EM4 the following console CLUNs are available:

- `0x08` – COM1
- `0x09` – COM2
- `0x0A` – TFT/Touch
- `0x40` – Telnet
- `0x41` – HTTP monpage
- `0x??` – COM3
- `0x??` – COM4
- …

Scenario 1: Default case (Virgin EEPROM, or abort pin set)

The EM4 MENMON would use these consoles if you had not specified something different (EM4 supports four consoles):

- *con0* (= debug console): `0x08` – COM1
- *con1*: `0x0A` – TFT/Touch
- *con2*: `0x00` – disabled
- *con3*: `0x00` – disabled
- Implicitly active: `0x40` – Telnet
- Implicitly active: `0x41` – HTTP monpage
- *gcon*: `0x0A` – TFT/Touch

Scenario 2: User has disabled all consoles

- *con0*: `0x00` – disabled
- *con1*: `0x00` – disabled
- *con2*: `0x00` – disabled
- *con3*: `0x00` – disabled
- Implicitly active: `0x40` – Telnet
- Implicitly active: `0x41` – HTTP monpage
- *gcon*: `0x0A` – TFT/Touch

Scenario 3: Misconfigured console

- *con0*: `0x01` – **bad**: IDE device! Using default `0x08` – COM1
- *con1*: `0x7F` – **bad**: non-existent device! Using default `0x0A` – TFT/Touch
- *con2*: `0x40` – **bad**: already active. Using default: `0x00` – disabled
- *con3*: `0x00` – disabled

Scenario 4: Automatic console selection

- *con0*: (= serial and debug console) `0xFF` – autoselect: COM1
- *con1*: `0x0A` – TFT/Touch
- *con2*: `0xFF` – autoselect: COM2
- *con3*: `0xFF` – autoselect: COM3
- Implicitly active: `0x40` – Telnet
- Implicitly active: `0x41` – HTTP monpage
- *gcon*: `0xFF` – autoselect: TFT/Touch

2.7.2 Example for A12

Here: A12a with P1 (graphics PC-MIP), PS/2 keyboard attached

- `0x08` – COM1
- `0x09` – COM2
- `0x0A` – PS/2 keyboard (input only device)
- `0x13` (example) – P1 (output only device)

Scenario: Default case (Virgin EEPROM, or abort button pressed)

con0 (=debug console): `0x08` – COM1

con1: `0xFF` – COM2

con2: `0xFF` – PS/2 keyboard

con3: `0xFF` – P1 PC-MIP

gcon: `0xFF` – P1 PC-MIP

2.8 Selecting the Baud Rate

All UART consoles will run with the same baud rate, determined by system parameter *baud*. The baud rate is set to the default value when the abort pin is set.

Table 2. *Baud rate selection – system parameter cbr/baud*

Parameter (alias)	Description	Default	User Access
cbr (*baud*)	Baud rate of all UART consoles (dec)	Board-specific, normally 9600	Read/write

A change of the baud rate will not take effect until the system is restarted.

(For details on system parameters, see Chapter 9 System Parameters on page 86.)

2.9 Selecting the Video Mode

System parameter *vmode* can be used to select the video mode for all active graphic screens present in the system.

Table 3. *Video mode selection – system parameter vmode*

Parameter (alias)	Description	Default	User Access
vmode	Vesa Video Mode for graphics console (hex)	`101` (640x480, 8 bits per pixel)	Read/write

If a graphics device does not support the selected mode, it falls back to the device-specific standard mode.

A change of the video mode will not take effect until the system is restarted.

Please note that not all graphics drivers use this setting!

Table 4. *Supported video modes*

Video Mode Number (hex)	Description
`101`	640 x 480, 8-bit indexed colors
`111`	640 x 480, 16-bit 5-6-5 RGB colors
`112`	640 x 480, 32-bit x-8-8-8 RGB colors
`103`	800 x 600, 8-bit indexed colors
`114`	800 x 600, 16-bit 5-6-5 RGB colors
`115`	800 x 600, 32-bit x-8-8-8 RGB colors
`105`	1024 x 768, 8-bit indexed colors
`117`	1024 x 768, 16-bit 5-6-5 RGB colors
`118`	1024 x 768, 32-bit x-8-8-8 RGB colors
`107`	1280 x 1024, 8-bit indexed colors
`11A`	1280 x 1024, 16-bit 5-6-5 RGB colors

(For details on system parameters, see Chapter 9 System Parameters on page 86.)

2.10 TFT/Touch Panel Screen

If the CPU board supports a touch panel and a TFT display, MENMON can be controlled by interacting with these TFT/touch panel devices.

For menu interaction, the screen is divided into two parts:

- The upper area is a text-mode terminal.
- The lower area contains the "touchable area" as a virtual keyboard. The virtual keyboard consists of cursor keys, alphanumerical keys, and some control keys.

Figure 1. *Touch panel virtual keyboard*

When a button is pressed, it changes its appearance to provide a visible response to the user. The "Sh" and "Lck" keys don't signal when they are pressed but have a small simulated LED that reflects their state.

Upper-case characters can be entered by pressing "Sh" (Shift) first and then the required character button. The "Sh" key resets itself when any key is pressed. The "Lck" key is the same as "Sh", but toggles its state only if "Lck" is pressed.

There is no autorepeat function for keys.

3 MENMON Start-up

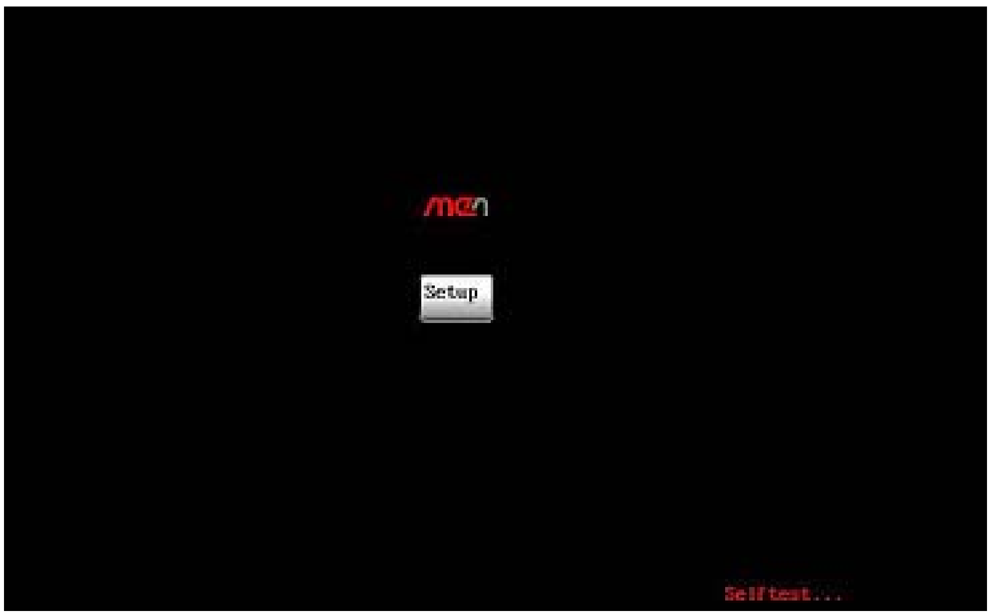
MENMON starts up just like any BIOS, with board-specific self-test routines and a secure mode (Degraded Mode) if needed.

3.1 Power-up Screen

At start-up, the graphics screen normally shows the MEN boot logo (see also Chapter 3.8 Changing the Boot Logo on page 20). If no graphics screen is connected, the boot logo screen will not appear.

If there is an input device, the graphics screen then shows a "Setup" button as long as the internal self-test is running.

Figure 2. *Boot logo screen*



MENMON displays the booting state at the bottom right edge of the screen. The text displayed in this area shows the current main state:

- Selftest
- Auto update check
- Booting

3.2 Entering the Setup Menu/Command Line

During normal boot, you can abort the booting process in different ways during the self-test, depending on your console:

- With a touch panel press the "Setup" button to enter the Setup Menu.
- With a text console press the "s" key to enter the Setup Menu.
- With a text console press "ESC" to enter the command line.

By default, the self-test is not left until 3 seconds have elapsed (measured from the beginning of the self-test), even if the actual test has finished earlier, to give the user a chance to abort booting and enter the Setup Menu. (See Chapter 4.1.2 Main Menu on page 23.)

The wait time can be modified using system parameter *stwait*. (For details on system parameters, see Chapter 9 System Parameters on page 86.)

3.3 Start of Networking

Networking – and therefore Telnet and HTTP server – are started only when MENMON stops the boot process for user interaction, for example when the screen menu or the command-line interface was invoked.

3.4 MENMON Start-up Screen

The MENMON start-up text screen appears only on text consoles at start-up.

You can display this screen also from the command line through command *LOGO*. It provides information on the hardware and board versions, and on the MENMON version.

Example (EM4):

```
 ___________________ Secondary MENMON for MEN EM04(N) 3.4 ____________________
|                                                                              |
|          (c) 2002 - 2005 MEN Mikro Elektronik GmbH Nuremberg                 |
|         MENMON 2nd Edition, Created  Jun 24 2005     10:31:04                |
|______________________________________________________________________________|
|      CPU Board: EM04N11              |             CPU: MPC8245              |
|  Serial Number: 12555                |  CPU/MEM Clock: 384 / 128 MHz         |
|    HW Revision: 01.09.00             |    DIMM Module:  64 MB 222            |
|______________________________________________________________________________|
|  Carrier Board: EC04-00, Rev 00.00.00, Serial 20                             |
 \____________________________________________________________________________/
```

3.5 Start-up String

When MENMON starts up, it reads the system parameter *mmstartup*. (For details on system parameters, see Chapter 9 System Parameters on page 86.)

- If *mmstartup* is not empty, MENMON executes the commands stored in *mmstartup*, which typically include NBOOT or DBOOT commands.
- If *mmstartup* is empty or all commands in *mmstartup* have been executed, MENMON checks the parameter *bsadr* (bootstrapper address).
- If the bootstrapper address is not zero, MENMON enters the bootstrapper (same as *BO* command, see Chapter 8.1.3 Boot from an Existing Image using BO on page 80).
- If both *mmstartup* and the bootstrap settings are invalid, MENMON enters the command-line interface.

The start-up string is composed of MENMON commands, separated by semicolon, e.g.:

```
DBOOT 1; NBOOT CLUN=3
```

You can set the start-up string from the command line or through the Setup Menu (Expert Setup, see Chapter 4.1.3 Basic/Expert Setup Menus on page 24).

To set the start-up string from the command line:

```
MenMon> ee-mmstartup DBOOT 1; NBOOT CLUN=3
```

To clear the start-up string:

```
MenMon> ee-mmstartup -
```

3.6 Problems during Start-up

Normally, MENMON continues to boot on self-test failure. However, if system parameter *stignfault* is 0, booting stops. (For details on system parameters, see Chapter 9 System Parameters on page 86.)

If the main memory (DRAM) is not working, MENMON enters the so-called "Degraded Mode". In this mode, MENMON will run in Flash and will only use on-chip RAM. (Cf. Chapter 3.7.1 Degraded Start-up on page 20.)

In Degraded Mode it is possible to run a board bring-up command, such as a memory test. However, it is not possible to boot an operating system. Whether network operation is possible depends on the CPU type.

3.7 Special Start-up Options

3.7.1 Degraded Start-up

When MENMON is starting, it checks the serial console for incoming characters.

Very early, it checks for characters "d" or "D". If one of these characters is detected, normal start-up is aborted and MENMON goes into Degraded Mode, i.e. it does not use SDRAM.

If "D" has been pressed, DRAM is not set up at all. If "d" is pressed, DRAM is set up but not used.

3.7.2 FPGA Loading

Normally, when MENMON starts on a board that has an FPGA and finds the FPGA is not loaded, a power on reset is assumed.

Before the FPGA is loaded, however, MENMON checks whether "F" was pressed. If so, it does not load the FPGA.

3.8 Changing the Boot Logo

Before self-test, MENMON attempts to load a boot logo from a port-specific medium, typically onboard CompactFlash or NAND Flash, unless disabled through system parameter *ldlogodis*.

- MENMON looks for a file named *bootlogo.bmp* on the medium. The file must be in the root file system of a DOS FAT 12/16[1] file system on any partition on the medium. The medium may or may not have a partition table.
- *bootlogo.bmp* must be a Windows (not OS/2!) BMP file with the following attributes:
  - Bitmap can be stored uncompressed or compressed with compression type 1 (8-bit Run Length Encoding).
  - The BMP file must include a color table with 256 entries.
  - The BMP's bits per pixel must be 8.
  - The BMP's size must be smaller than or equal to the graphics screen. If the BMP file is smaller, it is centered on the screen.

If no *bootlogo.bmp* file can be found on the medium or boot logo loading is disabled, the default "MEN" boot logo is shown.

A "Setup" button is shown when the selected graphic console is a touch panel. The boot logo's image shall reserve some space for this button.

The "Setup" button is only shown during the self-test and disappears when the self-test is finished. The underlying bitmap is then restored.

The upper left position of the setup button is at (y=center+50; x=center-27), the size of the button is 55 by 35 pixels (width by height).

Additionally, MENMON displays the booting state in a rectangular area at the bottom right edge of the screen starting at (y=height-20; x=width-200). This area should be left black by the boot logo in the boot logo bitmap.

1 Newer MENMONs may also support FAT 32.

4 Using MENMON

4.1 Screen-Oriented Menu User Interface

MENMON provides a user-friendly, PC-like setup menu to configure MENMON parameters, perform firmware or operating system updates, and run diagnostic tests.

The menu-driven user interface is available

- on VT100 terminals
- on VT100 over Telnet
- on MENMON HTTP */monpage*
- on frame-buffer graphics (color or b/w display).

You can enter the main menu by pressing the "Setup" button on the touch panel or "s" on the serial console during the self-test procedure. Alternatively, you can invoke the setup menu from the command line through command *SETUP*.

The following sections give an overview of navigation and functionality of the setup menu dialogs.

4.1.1 General Menu and Dialog Navigation

4.1.1.1 Menus

Examples: Main Menu, Basic Setup

- ESC/Arrow left: leave menu
- ENTER/Arrow right: enter menu item
- Arrow down/ '+': down one item
- Arrow up/ '-': up one item
- On touch panels the user can also directly hit the positions at which the menu items are printed.

4.1.1.2 String Dialogs

Examples: Basic Setup > Hostname of this Machine

- ENTER: take over value, leave dialog
- ESC: leave dialog, discard changes
- Arrow right: cursor right
- Arrow left: cursor left
- Backspace: cursor left, delete previous character

4.1.1.3 Multiple Choice Dialogs

Examples: Basic Setup > Boot Sequence

- ENTER: save changes, leave dialog
- ESC: undo changes, leave dialog
- TAB/Arrow right: change active cell to right
- Arrow left: change active cell to left
- Arrow down/ '-': select previous choice in cell
- Arrow up/ '+': select next choice in cell

4.1.1.4 Real time clock Dialog

Examples: Basic Setup > Real time clock

- ENTER/ESC: leave dialog
- TAB/Arrow right: change active cell to right (e.g. YY to MM)
- Arrow left: change active cell to left (e.g. MM to YY)
- Arrow down/ '-': decrement value of current cell
- Arrow up/ '+': increment value of current cell
- '0' - '9': Directly enter cell's value

4.1.2 Main Menu

The Main Menu branches to the submenus described below. Some of them are password protected by the simple password "42".

Table 5. *Main Menu – submenus and password protection*

Submenu	Password protected
Basic Setup	Yes
Expert Setup	Yes
Hardware Info	No
Diagnostics	Yes
Program Update	Yes
Touch Calibration	No
Touch Verification	No

```
Main Menu

  >Basic Setup

   Expert Setup

   Hardware Info

   Diagnostics

   Program Update

   Touch Calibration

   Touch Verification
```

Once you have entered the password correctly, you do not need to enter the password again until you return to the Main Menu.

Note: The main menu may have additional, implementation-specific entries.

4.1.3 Basic/Expert Setup Menus

In the Basic and Expert Setup menu, you can modify all important persistent parameters of the CPU board. In addition, the real-time clock can be set.

All parameter changes are first temporary. When you leave these menus, you will be asked whether to store or discard the changes. The setup of the real-time clock is an exception: any changes made here immediately take effect.

All of these settings can also be modified through the MENMON command line interface.

(For details on system parameters, see Chapter 9 System Parameters on page 86.)

Table 6. *Basic Setup menu*

Setting	Description
Boot sequence	User-friendly start-up string editing
Real time clock	Set real time clock
IP Address of this Machine	For MENMON Ethernet port
Subnet Mask of this Machine	For MENMON Ethernet port
IP Address of Boot Host	To reach host IP
IP Default Gateway Address	
Hostname of this Machine	

```
Basic Setup

  >Boot Sequence                        Any Disk, (None), (None)

   Real time clock                      05/06/15

   IP Address of this Machine           192.1.1.25

   Subnet Mask of this Machine          (unset)

   IP Address of Boot Host              192.1.1.22

   IP Default Gateway Address           192.1.1.22

   Hostname of this Machine             kp
```

Table 7. *Expert Setup menu*

Setting	Description
Startup string	Raw start-up string editing
Speed setting of network interface 0	Auto/fixed speed selection of Ethernet ports
Name of bootfile	Name of the boot file to use
Linux kernel parameters	

```
Expert Setup

 >Startup string                        DBOOT

  Speed setting of network interface 0  AUTO

  Name of bootfile                      vxworks.st

  Linux kernel parameters               (unset)
```

The parameters are explained in more detail now.

4.1.3.1 Basic Setup: Boot Sequence

With this parameter, the user can select up to three boot methods that will be executed on every system start. MENMON will try each boot method in the specified order until it finds a valid boot file.

Table 8. *Basic Setup: Boot Sequence – possible settings*

Boot Method	Description
(None)	No action
Any Disk	Boot from any disk (*DBOOT* command without arguments)
Ether (BP)	Boot from network (*NBOOT* command without arguments)
Ether (FTP/BP)	Boot from network (*NBOOT BOOT FTP*)
Ether (FTP)	Boot from network (*NBOOT STATIC FTP*)
Ether	Boot from network (*NBOOT TFTP*)
...	Boot from a specific disk

Specific disks are determined automatically. You can select any named MENMON BIOS disk device (DLUN/CLUN combination) that is known at the time the Setup menu was called.

Note: Some newer MENMON portations support read access to a USB memory stick as a disk device, which can then also be booted from.

The selected sequence is stored in system parameter *mmstartup* as a string of MENMON commands. For example, if the user selects: "Int. CF, Ether, (None)", the *mmstartup* string will be set to "DBOOT 0; NBOOT TFTP". (See also Chapter 3.5 Start-up String on page 19.)

For more information about boot methods, see Chapter 8 Operating System and Program Execution on page 76 and Chapter 6.2 Network Boot on page 65.

4.1.3.2 Basic Setup: Real time clock

Here, you can adjust the date/time of the real-time clock of the CPU board. Any changes made here immediately take effect. There is no way to undo the changes.

4.1.3.3 Basic Setup: IP Address of this Machine

Used to modify the IP address used for the Ethernet interface that MENMON will use for networking. This setting is stored in system parameter *netaddr*. The operating system can use this parameter to configure the Ethernet interface accordingly.

The parameter can be left blank (displayed as "(unset)") if the IP address shall be determined automatically (via BOOTP), otherwise it should contain a valid value.

In the dialog, you can enter only numerical characters [0-9] and a dot '.'.

4.1.3.4 Basic Setup: Subnet Mask of this Machine

Used to modify the IP subnet mask used for the Ethernet interface that MENMON will use for networking. This setting is stored in system parameter *netsm*. The operating system can use this parameter to configure the Ethernet interface accordingly.

The parameter can be left blank (displayed as "(unset)"). In this case MENMON and operating system will determine the subnet mask automatically from the IP address class (unless overridden by BOOTP response).

In the dialog, you can enter only hexadecimal characters [0-9A-Fa-f].

4.1.3.5 Basic Setup: IP Address of Boot Host

Only used for the *Ether* boot method. It is the address of a remote machine that contains the boot file for this machine. The remote host must run the TFTP server process.

This setting is stored in system parameter *nethost*.

In the dialog, you can enter only numerical characters [0-9] and a dot '.'.

4.1.3.6 Basic Setup: IP Default Gateway Address

Used to configure a default gateway for the system. If not empty, this parameter must be the address of a remote machine that acts as a gateway. The gateway must be in the same network as this machine.

This setting is stored in system parameter *netgw*. The operating system can use this parameter to configure the network stack accordingly.

The parameter can be left blank (displayed as "(unset)"). In this case no default gateway is assumed (unless overridden by BOOTP response).

In the dialog, you can enter only numerical characters [0-9] and a dot '.'.

4.1.3.7 Basic Setup: Hostname of this Machine

Allows the user to configure the hostname of this machine.

This setting is stored in system parameter *netname*. The operating system can use this parameter. MENMON does not use this setting.

In the dialog, you can enter characters [0-9A-Za-z.-].

4.1.3.8 Expert Setup: Startup string

Allows to directly edit the list of commands to execute on start-up (startup string). The maximum size of the string is 511 bytes. This size may be smaller, depending on the MENMON implementation. (See also Chapter 3.5 Start-up String on page 19.)

4.1.3.9 Expert Setup: Speed setting of network interface X

The Expert Menu contains one item for each Ethernet interface found in the system. A maximum of 3 interfaces can be edited through this menu.

Each setting allows the user to set up a fixed speed/duplex mode for the Ethernet interface in case Ethernet autonegotiation/autosensing causes problems.

Table 9. *Expert Setup: Speed setting of network interfaces – possible settings*

Setting	Description
AUTO	Use autonegotiation/autosensing
10HD	Use fixed parameters: 10 Mbits/s speed, half duplex
10FD	Use fixed parameters: 10 Mbits/s speed, full duplex
100HD	Use fixed parameters: 100 Mbits/s speed, half duplex
100FD	Use fixed parameters: 100 Mbits/s speed, full duplex
1000	Use fixed parameters: 1 Gbit/s speed

This setting is stored in system parameter *nspeedX*, which is stored in the serial EEPROM connected to the Ethernet interface or as a persistent parameter inside the CPU non-volatile storage.

4.1.3.10 Expert Setup: Name of bootfile

Allows to edit the default boot file name (system parameter *bf*). The maximum size of the string is 79 bytes.

4.1.3.11 Expert Setup: Linux kernel parameters

Allows to edit the Linux kernel parameter string (system parameter *kerpar*). The maximum size of the string is 399 bytes. This size may be smaller, depending on the MENMON implementation.

4.1.4 Hardware Info

This screen shows all important information about the hardware as well as the firmware revision, similar to the following screen. The information shown is board-specific, but shall look similar to the following screen:

```
Hardware Information

              CPU board:
        Production data: EM04N02, Rev 01.07.00, Serial 11856
             CPU clocks: 256 / 128 MHz
                MENMON: MENMON for MEN EM04(N) 3.2 (May 11 2005 - 12:38:05)
     FPGA code in flash: EM04nAD66-2A1
     fallback FPGA code: (none)

                SO-DIMM:  64 MB 222
  Internal CompactFlash: 61.3 MB / 125184 total sectors
                         TOSHIBA THNCF064MMG
             Boot Flash: 2.0 MB

          Carrier board:
        Production data: AD66-00, Rev 02.04.00, Serial 1670
  External CompactFlash: 31.1 MB / 63488 total sectors
                         TOSHIBA THNCF032MBA
             Static RAM: 4.0 MB
         Backup Voltage: 3.1 Volts
            Temperature: 45.9 degrees celsius

Hit any key: _
```

Note that the information in this screen is not dynamically updated. For example, if you change the external CompactFlash, you must leave this screen and enter it again in order to view information about the new card.

The capacity of SDRAM, SRAM, Flash and CompactFlash cards is shown in megabytes (1024*1024 bytes), not in million bytes, therefore 8MB CompactFlash cards will be displayed as 7.6 MB, for example.

4.1.5 Diagnostics

This menu allows you to perform different diagnostic functions and to branch to MENMON's command line interface:

```
Diagnostics

  >Run Automatic Tests

   Run Interactive Tests

   Run Endless Tests

   Enter MenMon
```

This chapter only deals with the functionality provided through the Setup menu. For more in-depth information on MENMON's diagnostic functions, please see Chapter 7 Diagnostic Functions on page 73.

4.1.5.1 Diagnostics: Run Automatic Tests

Automatic tests are all tests that require no external test equipment or user interaction.

You can either

- run all tests at once or
- run one specific test.

The list of available tests depends on the implementation. The progress and status of each test is displayed in the same line as the test name.

Example:

```
Automatic Test Menu

  >Run all tests
   Quick SDRAM connection test                         OK
   ETHER0 PCI & internal loopback                      OK
   Quick SRAM connection test                          OK
   EEPROM cell 0 check                                 OK
   RTC presence                                        OK
   Internal CF access / sector 0 read                  FAILED
   Touch controller                                    OK
   Printer communication                               FAILED
```

4.1.5.2 Diagnostics: Run Interactive Tests

Interactive tests work in the same way as automatic tests, but the lists of tests include

- tests that require external test equipment
- tests that take a long time to execute
- tests that require user interaction (e.g. press button)
- tests that destroy memory/media content.

The behavior and appearance of the menu are the same as in the Automatic Tests menu.

Example:

```
Interactive Test Menu

  >Run all tests
   Full SDRAM test
   ETHER0 PCI & external loopback
   Full SRAM test
   RTC running
   COM1 Tx/Rx external loopback
   External CF access / sector 0 read
   COM3 Line transceivers
   COM4 Tx/Rx external loopback
   COM5 Tx/Rx external loopback
   CAN1 front/rear interface
   CAN2 with SA08
   Print test page on printer
   Printer formfeed button
   COM2 Tx/Rx external loopback
```

4.1.5.3 Diagnostics: Run Endless Tests

Endless tests are mainly used for qualification or burn-in tests.

Please note that this test option is especially designed for production and temperature test and normally should not be entered by the user.

The lists include tests similar to interactive tests, e.g. requiring external test equipment or user interaction.

You can select which tests shall be executed in the endless test by selecting a menu item. Each hit on an item toggles the execution state (YES/NO).

Item "Start endless tests" starts to run the endless test. It executes all tests (marked YES) in the listed order (one pass through each test) and wraps to the first test when the end of the list is reached. This is repeated until you abort the test (through ESC or ^C).

Failure during tests will not stop the endless test, but the number of errors will be counted and the last error will be displayed.

Example:

```
Endless Test Menu

  >Start endless tests
   Full SDRAM test                        YES             OK
   ETHER0 PCI & external loopback         NO
   Full SRAM test                         YES             OK
   RTC running                            YES             OK
   Internal CF access / sector 0 read     YES             OK
   COM1 Tx/Rx external loopback           YES             SKIPPED
   Touch controller                       YES             OK
   External CF access / sector 0 read     YES             OK
   COM3 Line transceivers                 YES             ABORTED
   COM4 Tx/Rx external loopback           NO
   COM5 Tx/Rx external loopback           NO
   CAN1 front/rear interface              YES             FAILED
   CAN2 with SA08                         NO
   Printer communication                  YES             OK
   COM2 Tx/Rx external loopback           YES             FAILED

   Last Error :
   Error Msg  :
   Loop : 1                Errors :
   RS232: Rx stuck at TTL 1 in pass 0
```

4.1.6 Program Update Menu

This implementation-specific, optional menu item allows users to update internal memory (e.g. boot Flash, internal CompactFlash) with the content of external, removable media (such as external CompactFlash, USB stick).

The following shows the Program Update Menu as implemented on MEN's EM4 CPU:

```
Program Update Menu

  >Copy external CF -> internal CF (1:1)

   Copy external CF:IMAGE.C00 -> internal CF

   Copy external CF:IMAGE.FP0 -> boot flash FPGA code

   Copy external CF:IMAGE.FP1 -> boot flash fallback FPGA code

   Copy external CF:IMAGE.SMM -> boot flash sec. MENMON

   Copy internal CF -> external CF (1:1)
```

You can find details on each board's implementation in the respective hardware user manual.

4.1.6.1 Sector-By-Sector Media Copy Program Update

For example: Update internal from external CompactFlash ("Copy external CF -> internal CF (1:1)" on EM4).

This method copies the source medium to the destination medium sector-by-sector overwriting all information – including the partition table – in the destination medium. There are no special formatting requirements of the source media. The source medium can have any number of partitions with any type of file systems on it.

Both media shall have the same size (number of sectors).

If the media sizes do not match exactly, you will be asked whether you want to perform the update nevertheless.

In any case, you need to confirm the start of the update process.

Warning: If the destination medium is smaller than the source medium, you will propably loose information. Not all partitions may be copied completely. You should avoid copying in this case unless you know what you are doing!

4.1.6.2 Media Update from Image File

For example: Update internal CompactFlash from IMAGE.C00 ("Copy external CF:IMAGE.C00 -> internal CF" on EM4).

This method copies an image file (e.g. *IMAGE.C00*) on the source medium to the destination medium sector-by-sector overwriting all information – including the partition table.

The image file must be an uncompressed image file containing a file system image. MENMON will not interpret the data contained in the image file.

The image file must be in the root directory of a DOS FAT 12/16[1] file system on the source medium.

The image file and destination medium shall have the same size (number of sectors). Otherwise the same warning messages are issued as for the "Media Copy" method.

Warning: If the destination medium is smaller than the source medium, you will propably loose information. Not all partitions may be copied completely. You should avoid copying in this case unless you know what you are doing!

4.1.6.3 Flash Update from Image File

For example: Update FPGA code from IMAGE.FP0 ("Copy external CF:IMAGE.FP0 -> boot flash FPGA code" on EM4).

This method can be used to update parts of Flash from an image file located on the source medium.

The destination areas can be the same as those descibed in the Chapter 4.2.3 Program Update Features on page 40, e.g. "IMAGE.FP0" for the first FPGA code.

The image file will be loaded from the source medium, the Flash will be erased, and the image file will be programmed into Flash.

Note: Use this option with care! If the file format or size of the image file is incorrect, the system may no longer work!

4.1.6.4 Auto Update Check Menu

This implementation-specific, optional menu is invoked during the normal MENMON start-up sequence, just before booting state is entered.

It checks external, removable media on whether they are bootable media or media containing files that can be used to update internal devices.

The exact behavior depends on the implementation.

[1] Newer MENMONs may also support FAT 32.

4.1.7 Touch Calibration

You can enter the touch-panel calibration function through the Setup Main Menu.

This function is also entered automatically during the self-test, if you hit the touch screen at any position outside the "Setup" button. You may have missed the "Setup" button because the touch panel was incorrectly calibrated. Therefore, you are asked what to do now:

```
You’ve hit the touch screen outside the "Setup" button.
The touchscreen may need recalibration.


- Press ’s’ to enter Setup Menu

- Press ’ESC’ to continue boot

- Press touchscreen anywhere else to calibrate touchscreen
```

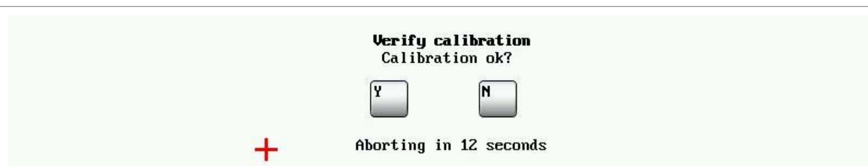
If you did not (or cannot) hit the ESC or "s" button, the touch calibration procedure is entered.

During touch calibration, you need to hit the upper left and then the lower right edge.

Figure 3. *Touch screen calibration*

Then you can hit the touch screen and a red cross should appear exactly at the position where you have hit the screen. You have to explicitly confirm the correct calibration by pressing the "Y" button:

Figure 4. *Touch screen calibration: confirm*



If you did not press the "Y" or "N" button within 15 seconds, the procedure is aborted.

If you hit the "Y" button, the new calibration parameters are permanently saved.

4.1.8 Touch Verification

Touch verification is similar to touch calibration. You can hit the touch screen and a red cross should appear exactly at the position where you have hit the screen.

You can finish verification by hitting the "Exit" button.

4.2 Command-Line User Interface

The command-line interface is available on all consoles. It is a simple, shell-like command interpreter and is invoked in case of an error at system start-up. You can also enter the command-line interface through the Setup Menu, submenu Diagnostics > Enter MenMon.

4.2.1 Command Line Editing

Each MENMON command conforms to the syntax

```
COMMAND [<arg1>] [<arg2>] [<argn…>]
```

The meaning and number of arguments are specific to each command.

The command name itself is not case sensitive. Arguments can be case sensitive, depending on the command.

4.2.1.1 Numerical Arguments

Many MENMON commands require numerical arguments. Numerical arguments may be numbers or simple expressions:

<num>	*num* is interpreted as a hexadecimal value (command dependent)
$<num>	*num* is also interpreted as a hexadecimal value
#<num>	*num* is interpreted as a decimal value
%<num>	*num* is interpreted as a binary value

These arguments can be combined using the arithmetic operators "+" and "-".

Example:[1]

`MenMon> D 10000`	Dumps address `0x10000`

[1] Some of the addresses used in our examples may not be suitable for your board's address mapping. If you want to try out MENMON's functions, please compare the example addresses with your mapping first!

4.2.1.2 Line Editing Features / Command History

MENMON supports a fancy line editor, supporting the following special keys:

<CTRL> <H>	Backspace and delete previous character
<CTRL> <X>	Delete current line
<CTRL> <A>	Reprint last line
<CTRL> <M> (<CR>)	Finish line through carriage return
<CTRL> <P> (Arrow up)	Go to previous line in history
<CTRL> <N> (Arrow down)	Go to next line in history
<CTRL> <B> (Arrow left)	Cursor left
<CTRL> <F> (Arrow right)	Cursor right
<CTRL> <L>	List history

All other control characters smaller than `0x20` are ignored.

On the output side, line editing is implemented only by using the characters space (`0x20`) and backspace (`0x08`), so line editing will work even on dumb terminals.

The maximum size of each line is 400 characters.

The number of commands kept in the history buffer is 10.

Note: Arrow up/down/left/right work only on VT100 compatible terminals or terminal emulations.

4.2.1.3 Help on Commands

Command *HELP* or simply *H* prints help for each command.

H **HELP**	**Print short help on all commands**
H <cmd> **HELP <cmd>**	**Print detailed help on specific command**
cmd	Command name

4.2.2 Memory Commands

C[<opts>] <addr> [<val> ...]	**Change memory**
opts	
B/W/L/LL	Specifies byte/word/long/longlong access
A	Change ASCII byte
X	Byte swap value
N	Don't read memory (write only)
#	Optional increment after access
addr	Memory address to change
val	Optional values

Interactively examine/change memory. If no address is given, the command uses the most recently used address.

If *val* is specified, the value(s) are written to memory without user interaction.

D [<addr>] [<cnt>]	**Dump memory**
addr	Memory address to dump
cnt	Number of bytes to dump

Dump memory in hex/ASCII. If no address is given, the command uses the most recently used address.

If no *cnt* is given, the command dumps 256 bytes.

Command can be repeated by simply pressing <CR> on command line. Dump continues on the next address.

MT[<opts>] <start> <end> [<runs>]	**Memory test**
opts	
B	8-bit access
W	16-bit access
L	32-bit access (default)
F	32-bit access, use special pattern that flips every 64 bits
P	Run next test with linear test pattern (default: random)
V	Don't write to RAM, just verify
start	Start address to test memory
end	End address + 1
num	Number of test cycles (default=endless)
Destructive (!) memory test.	
Test specified RAM starting at <start> to <end>-1 with the specified access modes.	
Examples:	
`MT 100000 200000`	32-bit test, random pattern, `0x100000..0x1FFFFFF`
`MTBL 100000 200000`	8-, then 32-bit test, random pattern, `0x100000..0x1FFFFFF`
`MTBPWL 100000 200000`	8-bit random, 16-bit linear, 32-bit random

FI <from> <to> <val>	**Fill memory (byte)**
from	Start address
to	End address
val	(Byte) value to fill memory

MC <addr1> <addr2> <cnt>	**Compare memory**
addr1	Address of first memory block
addr2	Address of second memory block
cnt	Number of bytes to compare

MO <from> <to> <cnt>	**Move (copy) memory**
from	Source address
to	Destination address
cnt	Number of bytes to copy

MS <from> <to> <val>	**Search pattern in memory**
from	Start address
to	End address
val	(Long) pattern to find in memory

4.2.3 Program Update Features

MENMON and FPGA code can be updated by uploading files over a serial line or a Telnet console.

These methods require a remote computer (e.g. notebook) connected to the target's UART or LAN interface running a terminal emulation (e.g. HyperTerminal or Minicom).

HyperTerminal supports sending of files even through Telnet connection. Minicom does not support this. Linux users have to run the *sz* program.

Note: Use this option with care! If the file format or size of the image file is incorrect, the system may no longer work!

4.2.3.1 Update via Serial Interface using *SERDL*

To initiate an upload, you have to enter

```
MenMon> SERDL [passwd]
```

on the MENMON prompt.

The terminal emulation program must be advised to start upload via "Ymodem" protocol and send the required file.

For some destination areas, a password is required to avoid unintended damage of the area (see table below).

The uploaded file must have a special extension to tell MENMON at which address to program the file. The following table shows the standard locations:

Table 10. *Standard locations for serial downloads*

File Name Extension	Typical File Name	Password for *SERDL*	Location
.SMM	*MENMON_EM04.SMM*	MENMON	Secondary MENMON
.FP0	*EM04N11IC002A0.FP0*	FPGA0	FPGA0 code
.FP1	*EM04N11IC002A0.FP1*	FPGA1	FPGA1 code (backup)
.Fxxx	*MYFILE.F000*	-	Starting at sector *xxx* in boot Flash
.Exx	*MYFILE.E00*	-	Starting at byte *xx* in EEPROM
.Cxx	*DSKIMG.C00*	DISK	Starting at sector *xx* in first disk
.Bxx	*DSKIMG.B00*	DISK	Starting at sector *xx* in second disk

For compatibility, further board-specific extensions can be defined.

- The file is first stored completely into RAM before Flash programming begins.
- After programming, Flash contents are verified against the contents in RAM. If it fails an error message will be displayed.
- During the command *MenMon> SERDL DISK*, the *DSKIMG.C00* will be flashed in RAW mode starting at sector *xx* in first disk. This command overwrites MBR including table of partitions information as well.

4.2.3.2 Update from Network using *NDL*

You can use the network download command *NDL* to download the update files from a TFTP server in network. The file name extensions, locations and passwords are the same as for the *SERDL* command.

See Chapter 6.4 Network Load & Program Command on page 65 for more info.

4.2.3.3 Update from Local Disk using *PGM-xxx*

You can use the *PGM-xxx* command to perform a disk-to-disk, image file-to-disk or image file-to-Flash copy from the command line with more flexibility as in the Main Menu "Program Update Menu" (see Chapter 4.1.6 Program Update Menu on page 32).

PGM-CLONE <sclun> <sdlun> <dclun> <ddlun>	**Media copy 1:1**
sclun	CLUN of source medium
sdlun	DLUN of source medium
dclun	CLUN of destination medium
ddlun	DLUN of destination medium
PGM-FILE <sclun> <sdlun> <sfile>	**Copy from file**
sclun	CLUN of source medium
sdlun	DLUN of source medium
sfile	Source file name

4.2.3.4 Update from RAM using *PFLASH*

The *PFLASH* command programs on-board Flash directly from RAM.

PFLASH <D> <O> <S> [<A>]	**Program Flash** Programs Flash with data from RAM
D	Device code (e.g. "F" for Flash, "C" for first disk, see Chapter 4.2.3.1 Update via Serial Interface using SERDL on page 40) Note: Device codes "FP0" and "SMM" are not used!
O	Start offset within device (in bytes)
S	Number of bytes to program
A	Buffer in RAM that contains data to be programmed If missing, uses default buffer address of *SERDL*/ *NBOOT*/*DBOOT* command.
Example: Program first `0x10000` bytes in boot Flash with content in RAM at `0x100000` `PFLASH F 0 10000 100000`	

4.2.3.5 Erasing Flash Sectors

MENMON provides a command to erase Flash sectors. Normally you should not need this command. If you do, however, you should use it with great care!

ERASE <D> [<O>] [<S>]	**Erase Flash sectors**
D	Device code (e.g. F=Flash) Note: Supported codes depend on implementation.
O	Start offset within device (in bytes)
S	Number of bytes to erase

The primary and secondary MENMON cannot be erased.

4.2.4 Get/Set the RTC Time

The *RTC* command shows the current time of the real-time clock or sets the time with the specified values. MENMON supports only one instance of an RTC.

RTC		**Display current time**
RTC-SET YY MM DD hh mm ss		**Set time**
	YY	Year (decimal, two digits)
	MM	Month
	DD	Day
	hh	Hours
	mm	Minutes
	ss	Seconds

4.2.5 Show Board/CPU Information

The *I* command dumps all important information to the console. In contrast with the screen-menu hardware information screen this information is intended to be used by developers.

I [D]	**List board/CPU information**
D	Show detailed information

Information will typically include:

- CPU type, clocks, revision, mask
- Address maps (i.e. CPU view and PCI view)
- PCI information

Example:

```
MenMon> I D
CPU:
 PVR: 80811014 (PCI Rev. 0x14)
Clocks:
 PCI:     32000000
 MEM:    128000000
 DEC:     32000000
 COR:    256000000
SODIMM:
 Size: 0x04000000
Addr Map:
 EUMB: fc000000
Binaries:
 PMM     : fff00000 (00080000)
 SMM     : fff80000 (00080000)
  Attached binary   valid "MENMON for MEN EM04(N) 3.2 (May 11 2005 - 12:38:05)"
 FPGA0   : ffe00000 (00080000)
  Attached binary   valid "EM04nAD66-2A1"
 FPGA1   : ffe80000 (00080000)
 CODE    : 01f00000 (00080000)
 STACK   : 01f80000 (00010000)
 HEAP0   : 01fa0000 (00060000)
 HEAP1   : 01d00000 (00200000)
 EXCEPT  : 00000000 (00004400)

NUMBER OF MAPPED PCI BUSSES => 0
PCI IO:
    START => fe000100
    END   => fe00ffff
    ALLOC => fe000100
PCI MEMORY:
    START => 90000000
    END   => 91ffffff
    ALLOC => 90002000
PCI INT ROUTING:
    INTA =>   9
    INTB =>  10
    INTC =>  11
    INTD =>  12

PCI BRIDGES:
    PrimBus DevNo SecBus
    ------- ----- ------
```

4.2.6 ESM Carrier Board Commands

MENMON may provide specific commands for ESM carrier boards. Any *ESMCB-xxx* commands implemented are documented in the respective online help. Typical commands include *ESMCB-TEMP*, for instance. This command displays the current carrier board temperature.

4.2.7 Set Debug Options

For debugging purposes, MENMON provides several modes that can be selected through command *BIOS_DBG*:

BIOS_DBG <mask> [net]		**Set MENMON BIOS or network debug level**
	mask	16-bit mask for debug settings: Bit 15: Enable error messages Bit 2: Enable debug level 3 Bit 1: Enable debug level 2 Bit 0: Enable debug level 1
	net	Network debug level

	Examples:
	Enable **all** BIOS debug messages:
	`BIOS_DBG 8007`
	Disable all BIOS debug messages:
	`BIOS_DBG 0`
	Enable most important network debug messages:
	`BIOS_DBG 8001 net`	

BIOS_DBG cons <clun>		**Set debug console**
	clun	CLUN of debug console

5 Device and Driver Management

MENMON BIOS is responsible to keep track of all available drivers and devices. In addition, the entire disk support is done in MENMON BIOS.

This chapter provides a detailed look into how MENMON manages devices and boots operating systems (see Chapter 8 Operating System and Program Execution on page 76).

5.1 BIOS Tables

5.1.1 Controller Logical Unit Numbers

MENMON supports several device tables. At the lowest level there is the controller device, an instantiation of a controller driver. For example, an IDE controller is a controller device. Each controller device is assigned a Controller Logical Unit Number (CLUN), to refer to the controller device. The controller device table is built only at start-up of the CPU.

The following controller classes are maintained by MENMON BIOS:

- SCSI controllers
- IDE controllers
- USB controllers
- FDC (floppy disk) controllers
- Ethernet controllers
- PCMCIA controllers (not used)
- Consoles (text mode)
- Graphics consoles
- Miscellaneous controllers

The implementation may assign symbolic names to each instance of a controller device (e.g. COM1 or TOUCH). If no specific name is assigned, each controller device is assigned an automatic name according to its device class, such as "ETHER0".

The CLUN is an 8-bit number, the exact numbers are board-specific, but shall be as follows:

***Table 11.** Recommended CLUN ranges*

CLUN	Description
`0x00..0x1F`	On-board devices `0x00` shall be used for disk devices only, as this value has special meaning for network and console devices
`0x20..0x3F`	Auto detected devices (typically off-board)
`0x40..0xFE`	For console devices like Telnet, HTTP monitor page
`0xFF`	Reserved

5.1.2 Device Logical Unit Numbers

For disks, there is an additional layer: devices. For example, an IDE or USB hard disk would be called a device by the MENMON BIOS. Each device is assigned a Device Logical Unit Number (DLUN) that is unique for the controller. The MENMON device table is built dynamically on request (entries are added by the *IOI* or *DBOOT* command, for example).

DLUNs are used only for disks, not for other types of devices.

Interpretation of DLUNs is up to the device driver. For example, the IDE driver uses DLUN=0 for the IDE master drive and DLUN=1 for IDE slave.

5.1.3 Display MENMON BIOS Tables

You can use the *IOI* command to display the CLUNs and DLUNs known by MENMON and to scan for devices behind each controller.

Example with EM4N:

```
MenMon> IOI

====== [ Controller Dev Table ] =========
CLUN Name             Driver      param1     param2     param3     Handle
0x00 IDE0             16Z023_IDE  0x90000200 0x00000000 0x00000001 0x01e10720
0x01 IDE1             16Z023_IDE  0x90000280 0x00000000 0x00000001 0x01e106b0
0x02 ETHER0           EEPRO100    0x92000000 0x0000d000 0x00000000 0x01e0e020
0x08 COM1             DUART8245   0xfc004500 0x07a12000 0x00000001 0x01effc70
0x09 COM2             DUART8245   0xfc004600 0x07a12000 0x00000001 0x01e107d0
0x0a TOUCH            EM04TOUCH   0x90000100 0x00000000 0x00000000 0x01e10810
0x0b COM10            Z025_UART   0x90000500 0x07a12000 0x00000000 0x01e0dfe0
0x0c COM11            Z025_UART   0x90000510 0x07a12000 0x00000000 0x00000000
0x0d COM12            Z025_UART   0x90000520 0x07a12000 0x00000000 0x00000000
0x0e COM13            Z025_UART   0x90000530 0x07a12000 0x00000000 0x01e0ccd0
0x10 CAN_TEST         CANGPIO     0x90000600 0x00000000 0x00000000 0x00000000
0x20 GXCONS1          Z032_DISP   0x80000000 0x00000000 0x00000000 0x01efc960
0x21 MISC0            CHAMELEON   0x0000e800 0x00000000 0x00000000 0x00000000
0x40                  TELSVR      0x00000017 0x00000000 0x00000000 0x01e0dba0
0x41                  HTTPD_MON   0x01e0d9f0 0x00000000 0x00000000 0x01e0d970

====== [ Device Table ] =========
CLUN DLUN Name            Model                             Type Handle

Scanning for devices on IDE bus (CLUN=0x00)...
0x00 0x00 Int. CF         TOSHIBA THNCF064MMG               HD   0x01e0def0

Scanning for devices on IDE bus (CLUN=0x01)...
0x01 0x00 Ext. CF         TOSHIBA THNCF032MBA               HD   0x01e0d7e0
```

IOIN just displays the currently known devices without scanning, for example:

```
MenMon> IOIN

====== [ Controller Dev Table ] =========
CLUN Name             Driver       param1     param2     param3     Handle
0x00 IDE0             16Z023_IDE   0x90000200 0x00000000 0x00000001 0x01e10720
0x01 IDE1             16Z023_IDE   0x90000280 0x00000000 0x00000001 0x01e106b0
0x02 ETHER0           EEPRO100     0x92000000 0x0000d000 0x00000000 0x01e0e020
0x08 COM1             DUART8245    0xfc004500 0x07a12000 0x00000001 0x01effc70
0x09 COM2             DUART8245    0xfc004600 0x07a12000 0x00000001 0x01e107d0
0x0a TOUCH            EM04TOUCH    0x90000100 0x00000000 0x00000000 0x01e10810
0x0b COM10            Z025_UART    0x90000500 0x07a12000 0x00000000 0x01e0dfe0
0x0c COM11            Z025_UART    0x90000510 0x07a12000 0x00000000 0x00000000
0x0d COM12            Z025_UART    0x90000520 0x07a12000 0x00000000 0x00000000
0x0e COM13            Z025_UART    0x90000530 0x07a12000 0x00000000 0x01e0ccd0
0x10 CAN_TEST         CANGPIO      0x90000600 0x00000000 0x00000000 0x00000000
0x20 GXCONS1          Z032_DISP    0x80000000 0x00000000 0x00000000 0x01efc960
0x21 MISC0            CHAMELEON    0x0000e800 0x00000000 0x00000000 0x00000000
0x40                  TELSVR       0x00000017 0x00000000 0x00000000 0x01e0dba0
0x41                  HTTPD_MON    0x01e0d9f0 0x00000000 0x00000000 0x01e0d970

====== [ Device Table ] =========
CLUN DLUN Name            Model                               Type Handle
0x00 0x00 Int. CF         TOSHIBA THNCF064MMG                 HD   0x01e0def0
0x01 0x00 Ext. CF         TOSHIBA THNCF032MBA                 HD   0x01e0d7e0
0x00 0x01                                                     ?    0x00000000
0x01 0x01                                                     ?    0x00000000
```

IOID displays the controller driver (not device) table, for example:

```
MenMon> IOID

====== [ Controller Drv Table ] =========
DRV# Driver                Type
0x00 DUART8245             CONSOLE
0x01 Z025_UART             CONSOLE
0x02 Z032_DISP             GXCONS
0x03 Z044_DISP             GXCONS
0x04 EM04TOUCH             GXCONS
0x05 CHAMELEON             MISC
0x06 16Z023_IDE            IDE
0x07 CANGPIO               CONSOLE
0x08 EEPRO100              ETHER
0x09 TELSVR                CONSOLE
0x0a HTTPD_MON             CONSOLE
```

5.1.4 Autoprobe for PCI Devices

Before self-test MENMON scans the PCI bus for devices and all registered drivers are probed for each device. Each device found creates an entry in the MENMON controller device table. At this time however, the hardware of the device will not be initialized. This will be done on demand, when the device is accessed for the first time.

5.1.5 Autoprobe for Chameleon FPGA Units

MEN's Chameleon FPGA is a special PCI device. If such a device is found on PCI, the Chameleon table is scanned and all registered drivers are probed.

If a driver detects that a Chameleon unit is detected that it supports, it creates one or more entries in the MENMON BIOS controller device table.

5.2 Disk Support

MENMON supports different kinds of disk devices, primarily for booting through *DBOOT*.

5.2.1 Support for Disk Boot

Disk boot supports the following:

- Boot from any disk-like device: SCSI hard and floppy disks, IDE hard disks or CompactFlash, USB sticks.
- PReP and DOS disk partitions as well as unpartitioned media.
- Supported file formats: RAW, ELF, PReP and PPCBOOT images.

To be able to boot from disk media, each medium must be prepared in the following way:

Disk Partitions

- Hard disks may or may not have a partition table.
  MENMON supports a PC-BIOS style partition table in the first sector of the device, with the restriction that only primary partitions are supported.
- The partition type must be either
  - DOS (type `0x01`, `0x04`, `0x06`, `0x0E`) or
  - PReP (type `0x41`).

DOS File System

With DOS-formatted partitions (or unpartitioned media) the file system must be a DOS FAT file system (12-bit or 16-bit FAT entries[1]).

MENMON supports read-only access to the root directory of the file system.

File names are supported in 8.3 format only, i.e. a file name may have 8 characters plus 3 characters extension.

PReP File System

PReP (Type `0x41`) partitions have no file system, the entire partition is viewed as a single file (no file name is required). PReP partitions can contain either a PReP file or a PPCBOOT image.

5.2.2 Device Drivers

ATA Driver

MENMON's ATA driver supports:

- IDE disks up to 128 GB, master and slave
- CompactFlash

It does not support ATAPI CD-ROMs.

USB Driver

MENMON supports read access to storage devices which support the bulk protocol.

Floppy Disk Driver

MENMON supports WD765 standard floppy disks up to 1.44 MB.

1 Newer MENMONs may also support FAT 32.

5.2.3 Listing Disk Partitions and Contents

You can use command *LS* on the command line to display the partitions and files of a specific disk device.

LS <clun> <dlun> [<opts>]	**List files/partitions on device**
clun=n	Controller logical unit number (see *IOI* command for list of CLUNs, Chapter 5.1.3 Display MENMON BIOS Tables on page 47) If not specified, use CLUN 0
dlun=n	Device logical unit number (see *IOI* command for list of DLUNs, Chapter 5.1.3 Display MENMON BIOS Tables on page 47) If not specified, use DLUN 0 on controller
opts	
PART=n	Partition number on device 0 = entire drive 1..4 = partition 1..4 If not specified, list first partition

Example:

```
MenMon> LS 1
=== Partition Table on CLUN=0x01, DLUN=0x00 16Z023_IDE, TOSHIBA
THNCF032MBA
# Type Stat Offset             Size
- ---- ---- ---------- -------------
1 0x04 0x80 0x00000020      31664 kB
2 0x00 0x00 0x00000000          0 kB
3 0x00 0x00 0x00000000          0 kB
4 0x00 0x00 0x00000000          0 kB

=== Files on part 1 of CLUN=0x01, DLUN=0x00 16Z023_IDE, TOSHIBA
THNCF032MBA
Filename           Size
------------ ----------
1000_01.TXT      252900
1000_02.TXT      252515
1000_03.TXT      252516
1000_04.TXT      246344
1000_05.TXT      252658
1000_06.TXT      252677
APP.OUT          618090
HW_SPR~1.TXT     242029
HW_SPR~2.TXT     357229
LOGO1.TXT          3417
PIDT1_~1.TXT     252238
PIDT1_~2.TXT     252815
PIDT1_~3.TXT     240135
SW_1_1~1.TXT     305565
SW_2_1~1.TXT     315671
SW_SPR~1.TXT     304987
VXWORKS.ST      1847415
```

5.2.4 Reading from/Writing to RAW Disks

MENMON provides commands to read from and write to RAW disks:

DSKRD <clun> <dlun> <lsn> <blks> [<buf>]	**Read blocks from RAW disk**
clun=n	Controller logical unit number (see *IOI* command for list of CLUNs, Chapter 5.1.3 Display MENMON BIOS Tables on page 47)
dlun=n	Device logical unit number (see *IOI* command for list of DLUNs, Chapter 5.1.3 Display MENMON BIOS Tables on page 47)
lsn=n	Logical block number of first block to read
blks=n	Number of blocks to read
buf=addr	Destination address. If not specified, use download area

DSKWR <clun> <dlun> <lsn> <blks> [<buf>]	**Write blocks to RAW disk** **Caution: This command may destroy your disk contents!**
clun=n	Controller logical unit number (see *IOI* command for list of CLUNs, Chapter 5.1.3 Display MENMON BIOS Tables on page 47)
dlun=n	Device logical unit number (see *IOI* command for list of DLUNs, Chapter 5.1.3 Display MENMON BIOS Tables on page 47)
lsn=n	Logical block number of first block to write
blks=n	Number of blocks to write
buf=addr	Destination address. If not specified, use download area

5.2.5 Displaying and Modifying USB Settings

Depending on the board hardware, newer MENMON versions also provide USB support. The command-line interface includes the following commands:

USB [<bus>]		**Initialize USB controller and devices on a USB bus**
	bus	USB bus number 0..n (default 0)
If no bus number is given, the default bus/port configuration will be tried.		

USBT [<bus> <p1>..<p5>]		**Shows the USB device tree for the current bus**
	bus	USB bus number 0..n (default 0)
	p1	First USB port number
	..	
	p5	Last USB port number
Shows the USB device tree of all buses available (if no parameters given) or just the selected port path.		

USBDP [<bus p1..p5>] [-d<x>]		**Display/modify USB device path**
	bus	USB bus number 0..n (default 0)
	p1	First USB port number
	..	
	p5	Last USB port number
	-d[<x>]	Default port path configuration x = 0..n
Display or modify the port path to the USB boot device. To modify the device path *bus* and *p1* are mandatory, or *-d* must be passed for the default setting. If no arguments are passed, the command only displays the current setting.		

MENMON can boot from USB storage devices which support the bulk protocol. Currently most USB sticks support this protocol.

The user interface is the same as for local hard disks, e.g. you can list files on the USB device selected through *USBDP* using command *LS*. (See Chapter 5.2.3 Listing Disk Partitions and Contents on page 51).

To boot quickly and to rule out problems with incompatible USB devices, the default configuration scans and uses only specified port trees.

The following gives an example scan with a USB stick.

```
MenMon> usb
USB#0 OHCI at f0001000  trying ->portpath->0

USB#1 UHCI at fe000000  trying ->portpath->0

MenMon> usbt
Bus#1
  +  Hub (12MBit/s, 0mA, devAddr 1)
  |  UHCI Root Hub
  |
  +-0  Mass Storage (12MBit/s, 200mA, devAddr 2)
       USB      Flash Disk       35261740230DA519
```

You can also pass the bus number to the *USB* command. Then it scans not only the configured port path but the entire configuration on the specified bus. (In this example the USB stick is connected to an extra hub.)

```
MenMon> usb 1
USB#1 UHCI at fe000000

MenMon> usbt
Bus#1
  +  Hub (12MBit/s, 0mA, devAddr 1)
  |  UHCI Root Hub
  |
  +-0  Hub (12MBit/s, 100mA, devAddr 2)
    |   USB2.0 Hub
    |
    +-3  Mass Storage (12MBit/s, 200mA, devAddr 3)
         USB      Flash Disk       35261740230DA519
```

Command *USBDP* lets you display and configure the current configuration for USB boot.

```
MenMon> usbdp
boot device path is USB bus->0 portpath->0

MenMon> usbdp -d=1
boot device path is USB bus->1 portpath->0
```

You can use the *DBOOT* disk boot command to boot from the configured USB device:

```
MenMon> dboot 5
Looking for bootfile <vxW5_5_em01.st>

MMBIOS_OpenDevice clun/dlun 5/0
 Trying Device CLUN=0x05 DLUN=0x00 USB_BULK, USB...
  Trying Partition 1 (type=0x01)...

Booting from CLUN=0x05, DLUN=0x00 USB_BULK, USB partition #1
Loading file vxW5_5_em01.st 0x170451 byte
to 0x2000000        1473 kB
done.
Starting ELF-file
```

5.3 DRAM Memory

In general, if a board has an (SO-)DIMM bank, MENMON

- reads the SPD EEPROM of the DIMM
- programs the DRAM controller according to the SPD information and capabilities of the controller.

If no SPD is present, the SPD has a bad checksum or incompatible content, the DIMM bank is disabled or initialized with defaults (if possible).

In general, if a board has an on-board, non-removable DRAM bank, MENMON automatically detects the size of the bank.

5.4 PCI Devices

5.4.1 PCI Auto Configuration

Each MENMON implementation will scan the PCI bus during start-up and set up the PCI configuration header of each device found.

Features

- Supports up to 255 PCI/PCI bridges at any nesting level
- Supports single function and multifunction devices
- Supports VGA devices (and bridge's VGA enable bit)
- Can place prefetchable memory BARs into separate region

Limitations

- Only the first 64 K of I/O space is used.
- Only the first 4 GB of memory space is used.
- Expansion ROMs/cardbus CIS pointer is never used.

Header configuration for standard PCI functions (non bridges)

- Set BAR 0..5 to either I/O, memory space or prefetchable memory space.
- Set COMMAND register according to the enabled BARs. Try to enable bus mastering bit.
- Set PCI INT_LINE register automatically (see below).
- Set CACHE_LINE_SIZE / LATENCY timer to fixed, implementation-specific values.

Header configuration for PCI-to-PCI bridges

- Set BAR 0..1, INT_LINE, CACHE_LINE_SIZE, LATENCY_TIMER as for a standard device.
- Set PRIMARY, SECONDARY and SUBORDINARY bus number according to the hierarchy found.
- Set bridge filters for I/O and memory mapped I/O (according to devices found on secondary side).
- Set bridge filter for PF memory space, if implementation and all bridges to the prefetched device allow this.
- Set SECONDARY_LATENCY timer to fixed, implementation-specific values.
- Set BRIDGE_CONTROL VGA enable if a VGA device found behind bridge and this was the first VGA device found.

5.4.2 PCI Commands

MENMON provides the following PCI commands for the command-line interface.

PCI	**PCI probe**
	Lists PCI devices on bus 0..255. Scans the entire PCI hierarchy for devices. Every device found is displayed.

Example:

```
MenMon> PCI
busNo devNo funcNo DEV ID  VEN ID
===== ===== ====== ======  ======
0x 0  0x 0  0x 0   0x0006  0x1057
0x 0  0x1a  0x 0   0x1209  0x8086
0x 0  0x1d  0x 0   0x5104  0x1172
```

PCID[+] <dev> [<bus>] [<func>]	**Dumps the PCI config header of the specified device**
+	Print device special registers
dev	Device number
bus	Bus number
func	Function number

Example:

```
MenMon> PCID 1D
Single Function Dev
 ADDR.     VALUE        DESCRIPTION
 =====     =====        ===========
 0x00      0x1172       Vendor ID
 0x02      0x5104       Device ID
 0x04      0x0002       PCI command
 0x06      0x0400       PCI status
 0x08      0x01         Revision ID
 0x09      0x00         Standard Programming Interface
 0x0a      0x80         Subclass code
 0x0b      0x06         Class code
 0x0c      0x00         Cache line size
 0x0d      0x00         Latency timer
 0x0e      0x00         Header type
 0x0f      0x00         BIST control
 0x10      0x90000000   Base Address Register 0
 0x14      0x80000008   Base Address Register 1
 0x18      0x80800008   Base Address Register 2
 0x1c      0x00000000   Base Address Register 3
 0x20      0x00000000   Base Address Register 4
 0x24      0x00000000   Base Address Register 5
 0x28      0x00000000   Cardbus CIS Pointer
 0x2c      0xff00       Subsystem Vendor ID
 0x2e      0xff00       Subsystem ID
 0x30      0x00000000   Expansion ROM Base Address
 0x34      0x00         Capability Pointer
...
```

PCIC <dev> <addr> [<bus>] [<func>]		**PCI config register change** Allows to interactively modify any register in the config header of the specified device.
	dev	Device number
	addr	Config register address
	bus	Bus number
	func	Function number

PCIR	**List PCI resources** Dumps the PCI resources (I/O and memory) allocated for each device.

Example:

```
MenMon> PCIR
=================[ I/O Resources ]==================
busNo devNo funcNo DEV ID  VEN ID  ADDR (SIZE)
===== ===== ====== ======  ======  ====================
0x 0  0x1a  0x 0   0x1209  0x8086  0x0100 (0x0040)
=================[ MEM Resources ]==================
busNo devNo funcNo DEV ID  VEN ID  ADDR (SIZE)
===== ===== ====== ======  ======  ====================
0x 0  0x1a  0x 0   0x1209  0x8086  0x92000000 (0x00001000)
                                   0x92020000 (0x00020000)
0x 0  0x1d  0x 0   0x5104  0x1172  0x90000000 (0x00002000)
                                   0x80000000 (0x00400000)
                                   0x80800000 (0x00400000)
```

PCI-VPD[-] <devNo> [<busNo>] [<capId>]		**PCI Vital Product Data dump**
	-	Raw dump
	devNo	Device number
	busNo	Bus number
	capId	Address to VPD CAP ID

5.5 Chameleon FPGA Devices

MENMON can handle MEN Chameleon FPGA devices. Chameleon FPGA devices appear as a PCI single function on the PCI bus. A Chameleon FPGA may implement many different sub-units. To allow software to determine which units are present in the FPGA, each Chameleon FPGA implements a ROM table describing all units with some parameters (e.g. base address).

MENMON uses the Chameleon table to automatically adapt to the implemented FPGA units.

5.5.1 Chameleon Table Support

Command *CHAM* shows the Chameleon unit table read from the chameleon FPGA. If a CLUN number is passed, MENMON dumps this Chameleon table.

CHAM [<clun>]	**Dump FPGA Chameleon table**
clun	CLUN for Chameleon table to be displayed

5.5.2 Support for Loadable Chameleon FPGAs

Some MEN boards, especially ESMs, include an on-board FPGA that needs to be loaded by MENMON on start-up (for example: EM1, EM3, EM4, EM8). MENMON reads the FPGA code from boot Flash and loads it via on-board glue logic into the FPGA.

Most MENMON implementations support two FPGA images stored at different locations inside the boot Flash. One image is the "main" FPGA image, the other the "fallback" image, which is used if loading of the main FPGA image failed. The exact loading rules are:

MENMON will load FPGA code

- at each power-up
- when FPGA is currently loaded with a different variant that the one in Flash
- when FPGA is currently loaded with an older revision than the one in Flash
- when *CHAM-LOAD* is issued in the command-line interface.

If the FPGA has to be reloaded, MENMON will load FPGA0 first. If this fails for any reason, MENMON attempts to load a possible backup FPGAx. Failure reasons include:

- FPGA code in boot Flash has bad magic word or bad checksum.
- FPGA does not assert *CONFIG_DONE* after loading.
- FPGA cannot be accessed over PCI bus.

5.5.2.1 Format of Load Images

The FPGA code generated by the Altera FPGA development tools must be stored in *.ttf* format and then has to be converted using the tools *ttf2bin* and *fpga_addheader* on a Windows or Linux development host before it can be put into boot Flash.

fpga_addheader adds a header at the beginning of the image that includes additional information about the FPGA code, such as checkum, length and file name of the original file. *fpga_addheader* can either generate a 48-byte or 256-byte header. MENMON supports both.

MENMON uses the *filename* from the header in order to verify correct loading of the FPGA. *filename* must conform to the following convention:

YYYYZUUICNNNVR (e.g. *EM04N02IC002A0*)

After loading the FPGA, MENMON compares the file name *V* and *R* fields with the global *variant* and *revision* field in the Chameleon table. If they do not match, MENMON assumes that loading was not successful.

5.5.2.2 Force FPGA Loading

You can force an FPGA load from the command line through command *CHAM-LOAD*. The FPGA will be programmed with the contents stored at the specified address. This command can be used to test an FPGA image before it is actually stored in boot Flash.

CHAM-LOAD [<addr>]	**Load FPGA**
addr	Address of FPGA If no address is given, perform default loading sequence (as described above).

Note: After a *CHAM-LOAD* command, MENMON device drivers for the FPGA devices are not restarted (except the driver for the system unit), so IDE/TFT/touch and UARTs in the FPGA may not be functional.

6 Networking Functions

The MENMON networking subsystem features:

- Autoconfiguration through BOOTP/DHCP
- File download via FTP or TFTP
- Telnet server for remote login
- HTTP server for remote administration through web browser

6.1 Network Configuration

6.1.1 Network Persistent Parameters

Networking can be statically configured by a number of persistent parameters listed below. Some of them can be optionally automatically configured through *BOOTP*.

(See also Chapter 9 System Parameters on page 86.)

Table 12. *System parameters for networking*

Parameter	Description
netaddr	IP address and subnet mask of attached network interface, e.g. 192.1.1.28
netsm	Subnet mask of attached network interface, e.g. `FFFFFF00`
netgw	IP address of default gateway
nethost	Host IP address
bootfile	Boot file path name
u	User name on host (e.g. FTP download)
p	Password for user on host (e.g. FTP download)
ecl	CLUN of attached network interface (hex) 0 = None `0xFF` = First available Ethernet
tto	Minimum timeout between network retries (decimal, in seconds) 0 or 255 = default = 0.8s (decimal 0.254)
tries	Number of tries 0 = Infinite number of retries 1 = One try, no retries 2..254 = Number of tries 255 = Default number of tries = 20
tdp	Telnet server TCP port (decimal) 0 = Don't start Telnet server -1 = Use default port 23 Otherwise TCP port for Telnet server
hdp	HTTP server TCP port (decimal) 0 = don't start HTTP server -1 = use default port 80 Otherwise TCP port for HTTP server

6.1.2 Assignment of Network Interface

Parameter *ecl* can be used to select the CLUN of the network interface that should be used by MENMON for networking.

MENMON can use only one interface at a time.

If *ecl* is set to `0xFF`, MENMON selects the first available network interface.

If *ecl* is set to `0x00`, no network interface will be attached and networking is disabled.

Otherwise, *ecl* selects the CLUN of the network interface to be used. If this is not a network device or the CLUN does not exist, no networking is possible.

The active network interface can be changed at runtime by specifying the *CLUN=* parameter with one of the network commands (*NBOOT, NDL*, *BOOTP*). When this parameter is given, the currently active network interface is shut down and the new network interface is attached and is kept attached even after the command has finished. However the *CLUN=* parameter does not change the persistent value of *ecl*.

6.1.3 Automatic Configuration

IP configuration can be automatically retrieved from a network server.

The only available protocol for this is BOOTP. This protocol can be used for BOOTP and DHCP servers (DHCP servers support the BOOTP protocol as a subset).

Automatic configuration can be performed

- while MENMON executes the network servers (BOOTP in background)
- by *NBOOT*, *NDL*, *BOOTP* (BOOTP in foreground).

Before starting network servers, MENMON checks whether a static IP address was configured. If no IP address or "0.0.0.0" was configured, MENMON tries to retrieve parameters through autoconfiguration. In this case BOOTP will operate in the background (see below).

Commands *NBOOT* and *NDL* also perform IP autoconfiguration when the *TFTP* option is **not** given.

6.1.3.1 BOOTP Background Operation

To avoid blocking of MENMON when no BOOTP server exists (and no IP address has been configured), BOOTP can operate in the background.

The first BOOTP request is sent before starting the network servers. BOOTP requests are retried forever in the background as long as no valid BOOTP reply has been received.

You can display the current BOOTP state using the *netstat* command.

6.1.3.2 Parameters used with BOOTP

MENMON will set up the BOOTP request with the following parameters:

- *CIADDR* set to 0.0.0.0
- *FILE* is set to the value of the *bf* parameter ("generic file name")
- BOOTP contains DHCP cookie and DHCP request option

When the BOOTP reply is received, system parameters will be updated through autoconfiguration:

Table 13. *System parameter autoconfiguration through BOOTP*

Parameter	Description	Updated by Autoconfig?
netaddr	IP address	Yes, set to YIADDR of BOOTP reply
netsm	Subnet mask	Yes, always modified after BOOTP reply Set when BOOTP reply contains RFC1048 tag #1. If no such tag, apply automatic subnet mask.
netgw	IP address of default gateway	Maybe, when BOOTP reply contains RFC1048 tag #3
nethost	Host IP address	Yes, set to SIADDR of BOOTP reply
bf	Boot file path name	Maybe, if *FILE* in reply is not an empty string

These parameters remain volatile until one of the parameters is explicitly saved through an *EE-xxx* command.

The network stack will immediately use the new parameters.

6.2 Network Boot

Network boot is done through MENMON's *NBOOT* command. You can find details on booting over network in Chapter 8.1.2 Network Boot using NBOOT on page 78.

6.3 Obtaining the IP Configuration via *BOOTP*

The *BOOTP* command performs the first step of the *NBOOT* command:

BOOTP [<opts>] [TEST]	**Obtain IP configuration via BOOTP**
opts	See NBOOT command
TEST	Don't update network configuration after reply

This first step can be skipped by passing the optional parameter *TFTP* to the *NBOOT* command.

(See also Chapter 8.1.2 Network Boot using NBOOT on page 78.)

6.4 Network Load & Program Command

The *NDL* command performs a similar job as *SERDL*, but loads the image file from a TFTP server in the network.

Its behavior and parameters are basically identical to command *NBOOT*, but it additionally programs the specified file into Flash after download.

The LAN interface to be used can be selected through parameter *CLUN*.

The file name extension and password (if required) are identical to the *SERDL* command.

NDL [<opts>]	**Update Flash from network**
opts	
BOOTP	(default) Obtain IP addresses from BOOTP server.
STATIC	Use static IP addresses.
TFTP	(default) Fetch file via TFTP method
FTP	Fetch file via FTP method
CLUN=n	Controller logical unit number (see IOI command) If not specified, use first Ethernet controller
FILE=file	File name to boot. If not specified, file name must be provided by BOOTP server
LOAD=addr	Temporary buffer address (before file is programmed)
PW=passwd	Boot sector protection password (see SERDL command)

Example:

```
MenMon> NDL FILE=MENMON_EM04.SMM PW=MENMON
```

Retrieves all IP parameters by BOOTP protocol, then loads "MENMON_EM04.SMM" from the host/directory specified by the BOOTP server over TFTP and programs the file into the location in boot Flash used to store the secondary MENMON.

6.5 Network Status Commands

PING <host> [<opts>]	**Network connectivity test** The *PING* command is a diagnostic command to verify the network function. The command first attempts to resolve the destination MAC address through ARP. If successful, it sends ICMP echo requests to the remote host and waits for replies. By default, 4 packets, with a netto payload of 64 bytes, are sent every second, but this can be modified through options.
host	Address of host
opts	
n=<count>	Number of packets to test (hexadecimal)
t=<timeout>	Max time to wait for reply, in ms
l=<length>	Size of payload
r=<rate>	Time between packets, in ms

NETSTAT	**Show current state of networking parameters** Shows the settings currently used by MENMON's Network subsystem.

Example:

```
MenMon> NETSTAT
my IP:                 192.1.1.28
BOOTP status:          finished
subnet mask:           0xffffff00
host IP:               192.1.1.23
gateway IP:            0.0.0.0
bootfile:              /users/kp/bootfile
ethernet clun:         0x02, 00:c0:3a:21:20:5f
ethernet link:         link is up, 100FD, autoneg complete

minimum retransmit:    0 secs
number of tries:       20
```

ARP	**Dump network stack ARP table** (See Chapter 6.6.4 ARP Subsystem on page 68.)

6.6 Built-In Clients

TFTP and FTP clients implement the functions to retrieve files from a network server. No upload functionality is implemented.

6.6.1 BOOTP Client

The BOOTP client is implemented according to

- RFC 951 – bootstrap protocol
- RFC 1048 – BOOTP Vendor Information Extensions
- RFC 2132 – DHCP Options and BOOTP Vendor Extensions

The BOOTP request is sent by Ethernet/IP broadcast (Ethernet and IP address: all ones) to local from UDP port 68 to remote UDP port 67.

BOOTP replies are accepted on any IP address during this phase. The Ethernet address of BOOTP replies must be either a broadcast or directed to the MAC address of the local port.

BOOTP requests are retried when no BOOTP reply is received after the time specified by parameter *tto*. The number of retries is determined by parameter *tries*.

6.6.2 TFTP Client

The TFTP client is implemented according to

- RFC 1350 – The TFTP Protocol (Revision 2)

TFTP client first negotiates the block size to be 1024 with the server. This packet size was chosen to avoid packet fragmentation (which the MENMON TCP/IP stack cannot handle). However, it is still possible that fragmentation occurs, if MTU is smaller than approx. 1070 bytes. This case cannot be handled yet.

Each TFTP block is retried after the time configured by *tto* and the number of retries (for each block) is determined by *tries*.

The maximum number of TFTP blocks that can be transported by the TFTP client is limited only by the available RAM (not limited to `0x10000` blocks).

6.6.3 FTP Client

The FTP client is implemented according to

- RFC 959 – File Transfer Protocol

The FTP client uses user/pass (system parameters *u* and *p*) to log in on the host system. It is assumed that the FTP server always requires a password for the user.

The FTP client then switches to binary mode (TYPE I) and fetches the file from the server via the *RETR* command. Data connection PASV mode is not supported.

6.6.4 ARP Subsystem

The ARP (Address Resolution protocol) subsystem translates IP addresses into Ethernet MAC addresses. It is actually both a client and a server.

Whenever networking is active, incoming ARP requests for the local IP address are answered.

ARP requests are sent whenever MENMON needs to communicate to a host whose IP address is currently unknown. When an ARP reply is received, the IP/MAC address tuple is put into an ARP cache. This cache can hold 16 entries.

ARP requests are retried **40 times**, after waiting for 250ms for each reply (i.e. ARP look-up will **time out after 10 seconds**).

6.7 Built-In Servers

The network server starts as described in Chapter 3.3 Start of Networking on page 18. Before actually starting servers, MENMON checks whether the daemons are enabled (using system parameters *tdp* and *hdp*).

If any of the servers is enabled, the network interface is attached and optionally IP autoconfiguration is performed (see Chapter 6.1.3 Automatic Configuration on page 63).

6.7.1 Telnet Server

The MENMON Telnet server provides an additional MENMON text console. It appears as an additional console in the MENMON BIOS controller device table.

Only one Telnet client at a time can be connected to the server.

If no client is attached to the Telnet server, the server buffers the last 1024 bytes that have been printed to the console in a local buffer. Once a client attaches, it will receive the buffered bytes at once.

When a client is attached to the Telnet server, the local buffer (FIFO) decouples the MENMON console from the Telnet network activity. If the FIFO fills up, the MENMON console will be blocked until there is enough space in the FIFO.

6.7.2 HTTP Server

The MENMON HTTP server provides the general ability to configure/control MENMON from any web browser. It appears as an additional console in the MENMON BIOS controller device table.

General features:

- HTTP 1.1 compliant
- GET method supported only
- Allows multiple clients to be connected at the same time
- Allows multiple open sockets from the same client
- Support "keepalive" connections

6.7.2.1 HTTP Monitor Page

The HTTP monitor page provides an additional MENMON console.

You can connect your browser to this console via URL *http://<ipaddr>/monpage*.

MENMON internally maintains a virtual screen (25*80 chars) that records the recent MENMON output. This virtual screen is delivered to the browser on each access to the */monpage* webpage.

You can send MENMON commands using the HTTP input field and press the submit button.

There is no automatic refresh! You need to press the refresh/reload button of your web browser to refresh the display.

```
HTTP daemon started on port 80
MenMon> D 5
00000005: 5fffbf00 ff6fffff 5bc7eb00 ffffffff _.?..o..[Gk.....
00000015: fdfef700 fbffffdf f3fed502 ef7fffbf }~w.{.._s~U.o..?
00000025: 7fffdf00 ffefffef f7dfff80 fffbffff .._..o.ow_...{..
00000035: f6ffff04 fffffff7 ebbcfd40 ffffffff v......wk<}@....
00000045: 1fffff00 ffffffff 1fffaf08 fffeffff ........../..~..
00000055: fffdf600 fffffeff b3fffb00 fffffbff .}v...~.3.{...{.
00000065: 7eefef00 ffffffff 5fde9f18 ffffffff ~oo....._^......
00000075: f3bafe00 fff7ffff e2fff348 ffffffff s:~..w..b.sH....
00000085: 3fffef00 ffffffff bdffff00 ffffffff ?.o.....=.......
00000095: eafffc00 feffffff f3ffff00 ffff7fff j.|.~...s.......
000000a5: 1fffcf00 ffffffff ddf7ef08 fcffffff ..O.....]wo.|...
000000b5: 6bfffb08 fdfffff7 ebfffe00 ffffffbf k.{.}..wk.~....?
000000c5: 5dff7f00 ffffffff 9def5f00 efffffff ]........o_.o...
000000d5: e3ffff20 ffffffff f5ffff00 dfffffff c.. ....u..._...
000000e5: 8fffff00 ffffffff dcffbf00 feffffff ........\.?.~...
000000f5: fabffd34 fdf7ffef f3feff00 ffffffff z?}4}w.os~......
MenMon>
```

submit

6.7.3 ICMP Server

As long as networking is active, MENMON replies to all incoming ICMP ECHO (ping) requests with an echo reply.

The maximum size of payload that can be echoed is limited by the size of the Ethernet frame (no fragmentation support).

6.8 Network Interface (NETIF) Subsystem

6.8.1 NETIF Devices

Each instance of a network interface (NETIF) driver is called a network device.

- A driver can have multiple instances.
- Each instance is assigned a CLUN in MENMON BIOS.

Even if only one device can be attached to the network stack, further devices can be active at the same time, for instance to perform diagnostics (loopback test).

6.8.2 Speed Setup During Boot

On each MENMON start, every network device is forced to use its configured persistent speed setting. This happens regardless of whether MENMON's network stack will be activated or not.

For each network interface, the following sequence is performed:

- Get persistent speed from system parameter *nspeedX*, which may be *AUTO* or a fixed speed. If no persistent speed is available, use *AUTO*.
- The *nspeedX* parameter obtains its value either from the CPU EEPROM or from dedicated SROM.
- Start up network interface with the required speed.
- Do not wait until link setup is complete.

6.8.3 Diagnostic Command for Ethernet PHY

This diagnostic command directly communicates with Ethernet PHYs' management interface (works for those PHYs connected over MII).

MII <clun> [<reg>] [<val>]	**Ethernet MII register command**
	Dump change MII reg of Ethernet PHY
clun	Dumps all regs of specified CLUN
clun reg	Dumps only register *reg*
clun reg val	Writes value to register *reg*

6.8.4 SROM Programming of On-board Ethernet Devices

If on-board Ethernet devices have dedicated non-volatile storage (e.g. SROM) to store MAC addresses etc., MENMON implementations will automatically program this SROM whenever the CPU board serial number is changed.

This is done implicitly by the *EE-PROD* command. The MAC address can be overwritten independently of the serial number of the board through parameter *nmacx*.

Whenever the MAC address is programmed into SROM, each driver reprograms the entire SROM with default values (applies for on-board devices only).

6.8.5 Attachment of New Network Interface

When a new network interface is attached, the following sequence is performed:

- Get MAC address from *nmacX* (which gets it from the device or CPU EEPROM).
- Verify device EEPROM checksum, if supported by device (print warning on failure).
- Get persistent speed from *SYSPARAM*, which may be *AUTO* or a fixed speed. If no persistent speed is available, use *AUTO*.
- Start up network interface with the required speed.
- Do not wait until link setup is complete (see next chapter).

6.8.6 Link State Monitoring

As long as a network interface is attached to the network stack, its link state is periodically monitored. Every 5 seconds, the network stack tells the driver to check its link state (and reconfigure the network chip if required).

No special indication is given to the user to tell him about the link state. You can display the current link state using the *netstat* command.

6.8.7 Network Interface Shutdown

The active network interface is automatically shut down just before MENMON passes control to the operating system or client program, to make sure that no DMA activity damages the integrity of the started program.

7 Diagnostic Functions

Diagnostic tests can be used to test if the hardware is in good health. MENMON provides a pool of tests, the list of provided tests is implementation specific.

Diagnostic tests for non-optional hardware components are always registered (hardwired in MENMON). MENMON scans for optional components and registers available diagnostic tests for the components found during the scan.

Each test can be in one or more of the following test classes (or none at all). The MENMON implementation decides which test belongs to which class(es):

- POST – Executes during power on, requires no external equipment
- AUTO – Requires no external equipment
- NONAUTO – Requires external equipment or user interaction
- ENDLESS – Executed by endless test

You can run diagnostic tests

- during POST (see below)
- from the MENMON command line (see below)
- from the screen menu (see Chapter 4.1.5 Diagnostics on page 29).

From the command line or screen menu, any test can be aborted from any active console by entering "ESC" or "^C"; the status of the test is then set to ABORTED.

Further tests in a sequence of tests are not executed; control returns to user.

7.1 Diagnostic Tests from Command Line

From the command line, tests can be invoked using the *DIAG* command:

DIAG [<test> AUTO NON-AUTO POST ALL SHOW] [VTF]		**List/run diagnostic tests**
	<no argument>	List all available tests (with their revision)
	test	Run a specific test *test* is one of the names listed by the *DIAG* command, e.g. "COM1"
	AUTO	Run all tests in class "AUTO"
	NONAUTO	Run all tests in class "NONAUTO"
	POST	Run all tests in class "POST"
	ALL	Run all tests in any class, including those requiring external equipment
	SHOW	Show the status of any test executed since start of MENMON. All DIAG calls update the test status string (parameter *mmst*).
	V	Verbose flag: Test will issue additional output to about their current activity
	T	Traceability flag: Issue all output in traceability format (only for MEN internal use)
	F	Forever flag: Repeat given test or test group forever

By default any *DIAG* command issues a progress status (% completed) during test.

Examples

Execute extended SDRAM test forever:

```
MenMon> DIAG SDRAM_X F
```

Run all automatic tests once:

```
MenMon> DIAG AUTO
```

7.2 Power-On Self-Test (POST)

The power-on self-test (POST) is automatically run at start-up time, unless it was disabled by system parameter *stdis*.

Individual tests can be disabled via *stdis_XXX*, where *xxx* is the name of the test, e.g. *stdis_sdram*. However, it depends on the implementation whether a test can be disabled or not.

While POST is running, it reports the result of each test on all active consoles.

If a self-test error is detected, booting stops or continues, depending on the value of parameter *stignfault*. Even if a test fails, all other tests are performed.

The failed tests can be determined by the operating system by checking parameter *mmst*.

You can manually repeat the POST by calling *DIAG POST*.

(For details on system parameters, see Chapter 9 System Parameters on page 86.)

7.3 Test Status

For each inidividual test, MENMON maintains the result, which can be one of the following:

Table 14. *MENMON diagnostics – test status*

State	Description
OK	Test executed with success
FAILED	Test failed
NOW OK	Failed previously, but was ok after repeating test
ABORTED	Test aborted by user (ESC, ^C)
SKIPPED	Test could not be executed (e.g. conflict with other configuration, no resources…)
START	Test not executed or crashed

7.4 Diagnostic Test Status for Operating System

The operating system can read the status of each self-test by reading system parameter *mmst*. This is a string composed of all tests executed since the last power-on followed by the status of each test ('-'=FAILED, '+'=SUCCESS). Only tests belonging to the POST group will be included in the string.

For example, if all POST tests except the FRAM test succeeded, *mmst* will have the following value:

```
ALL-ETHER0+IDE+SDRAM+FRAM-EEPROM+RTC+
```

The special test name "ALL" indicates if any of the tests have failed.

Skipped or aborted tests or tests that have not been executed do not appear in the string.

8 Operating System and Program Execution

In general, MENMON can boot from disk, via network or from Flash.

8.1 Boot Methods

8.1.1 Disk Boot using *DBOOT*

MENMON uses *DBOOT* to boot from disks. Chapter 5.2.1 Support for Disk Boot on page 50 describes the requirements for bootable disks.

DBOOT [<CLUN>] [<DLUN>] [<opts>]	**Boot from disk**
CLUN=n	Controller logical unit number (see *IOI* command) If not specified, try all controllers in system.
DLUN=n	Device logical unit number (see *IOI* command) If not specified, try all devices.
opts	
PART=n	Partition number on device 0=entire drive 1..4=partition 1..4 If not specified try all partitions
FILE=file	File name to boot. Must be in root directory of DOS partition If not specified, file name from EEPROM (*EE-BOOTFILE*) is used File name is not required to boot from type `0x41` (PReP) partition
LOAD=addr	Load address for load image ELF files will be loaded at this address and relocated then other file formats will be executed at this load address. If not specified, use default
START=off	Offset of execution entry point relative to load address. Ignored for ELF files. If not specified, defaults to 0.
HALT=n	1=Stop after loading the file from disk 2=Stop immediately before starting image
KERPAR='p1=x p2=y'	Parameters to add to kernel command line (only used when booting PPCBOOT image)

By using CLUN and DLUN arguments and option *PART*, *DBOOT* can look only on the specified drive or partition:

```
DBOOT 2 1 PART=2
```

Selects CLUN=2, DLUN=1 and the 3rd partition on the disk.

8.1.1.1 Default *DBOOT* Algorithm

The *DBOOT* command tries to find a bootable partition or file on any disk. If no parameters are specified, *DBOOT* will search for devices behind each known CLUN.

On each disk found, it will check if there is a partition table on it, and checks with each partition if it is bootable or not.

Any PReP partition found is assumed to be bootable. For DOS partitions, *DBOOT* searches if the DOS file system contains the specified file.

The boot file must be in the root directory of the medium. The file name to be searched for can be configured by persistent parameter *bf*. Only the file-name part of that name is used (e. g. if you configure "/ata0/vxworks", then *DBOOT* looks for "vxworks"). If nothing is configured, the name "BOOTFILE" is used.

The file name can also be passed to the command line to *DBOOT* (e. g. *DBOOT file=myboot*).

The file is loaded into memory, the image contents are analyzed/moved and MENMON executes the image. See also Chapter 8 Operating System and Program Execution on page 76.

8.1.2 Network Boot using *NBOOT*

Booting from network is done through command *NBOOT*.

NBOOT [<opts>]	**Boot from network**
opts	
BOOTP	(default) Obtain IP addresses from BOOTP server
STATIC	Use static IP addresses
TFTP	(default) Fetch file via TFTP method
FTP	Fetch file via FTP method
CLUN=n	Controller logical unit number (see *IOI* command) If not specified, use first Ethernet controller.
FILE=file	File name to boot If not specified, file name must be provided by BOOTP server
LOAD=addr	Load address for load image ELF files will be loaded at this address and relocated then other file formats will be executed at this load address. If not specified, use default.
START=off	Offset of execution entry point relative to load address. Ignored for ELF files. If not specified, defaults to 0.
HALT=n	1=Stop after loading the file from network 2=Stop immediately before starting image
KERPAR='p1=x p2=y'	Parameters to add to kernel command line (only used when booting PPCBOOT image)
TRY=num	Number of retries for BOOTP/TFTP/ARP requests
TTO=num	Minimum number of seconds to wait for TFTP response

By default, network boot consists of two steps:

- Obtain IP address, host IP and boot file name through *BOOTP* (see Chapter 6.3 Obtaining the IP Configuration via BOOTP on page 65). A generic image name can be passed to the *NBOOT* command using the *FILE* option, see Chapter 6.1 Network Configuration on page 61.
- Load boot file through TFTP.

Once the file has been loaded, the loaded image is analyzed and started as described below.

The IP configuration method and transfer protocol can be specified through the *BOOTP* / *STATIC* and *TFTP* / *FTP* options. For historic reasons, the following combinations of config protocol and load protocol are allowed:

Table 15. *NBOOT – allowed combinations of config protocol and load protocol*

Parameter on Command Line	IP Config Method	Transfer Protocol
(none)	BOOTP	TFTP
TFTP	STATIC	TFTP
FTP	STATIC	FTP
BOOTP TFTP	BOOTP	TFTP
STATIC TFTP	STATIC	TFTP
BOOTP FTP	BOOTP	FTP
STATIC FTP	STATIC	FTP

Further parameters can be used to override persistent parameter settings:

Table 16. *NBOOT – parameters used to override persistent parameter settings*

Parameter	Override Option
bf	*FILE*
ecl	*CLUN*
tto	*TTO*
tries	*TRY* *TRIES*
kerpar	*KERPAR*

Example 1:

```
MenMon> NBOOT FILE=EM04A00unix KERPAR='ip=auto rw'
```

- Uses the Ethernet interface configured through *ecl* (no CLUN specified).
- Sends BOOTP request broadcast. (No *TFTP* parameter specified.)
- BOOTP request contains "EM04A00unix" as generic image name.
- Loads file name specified in BOOTP response over TFTP from host (own IP and host IP addresses also contained in BOOTP response).
- Passes the parameters "ip=auto rw" to the Linux kernel.

Example 2:

```
MenMon> NBOOT CLUN=3
```

- Uses the second Ethernet interface (CLUN=3).
- Since no file option was specified, send firmware boot file parameter as generic image name.
- Since no *KERPAR* option was specified, use firmware *kerpar* parameter as kernel parameters.

8.1.3 Boot from an Existing Image using *BO*

The *BO* command is used to start an image that is already in memory (e.g. RAM or Flash).

BO [<addr>] [<opts>]	**Call operating system bootstrapper**
addr	Start address of bootstrapper
opts	
KERPAR='*p1=x p2=y*'	Parameters to add to kernel command line (only used when booting PPCBOOT image)

The command performs the image format detection at the specified address (see below) and executes the image. It performs the same steps as executed by the *NBOOT*/*DBOOT* commands after the file has been loaded to memory.

If no address is specified, the command uses parameter *bs* for the address, unless this is set to `0x00000000`.

8.2 Special Boot Options

8.2.1 Boot Command *HALT* Option

NBOOT and *DBOOT* both have an option to stop the boot algorithm at certain points:

- *HALT=1* Stop after loading image to memory
- *HALT=2* Stop after executing first instruction of image, after possible relocation/uncompression. On PowerPC architectures, this will work only when booting ELF, RAW or PReP images and ELF image sections have not overwritten MENMON.

8.2.2 Boot File Load Address / *LOAD* Option

Regardless of the file format, the entire boot file will be loaded first to MENMON's download area, which is typically `0x01000000` or `0x02000000` on PowerPC platforms.

Since MENMON code/data sections are normally located at `0x01D00000` to `0x01FEFFFF`, the image size is limited to 13 MB when the load address is at `0x01000000`, therefore the new MENMON implementation will use `0x02000000` as default, if backward compatibility is not an issue, and if at least 64 MB RAM is available.

Some implementations choose the load address dynamically, depending on the size of the loaded file: If the file is larger than 13 MB, the load address is `0x02000000`, or `0x10000000` if it is smaller. This works for the *DBOOT* command only, as the file size is not known for *NBOOT* before the file load has completed.

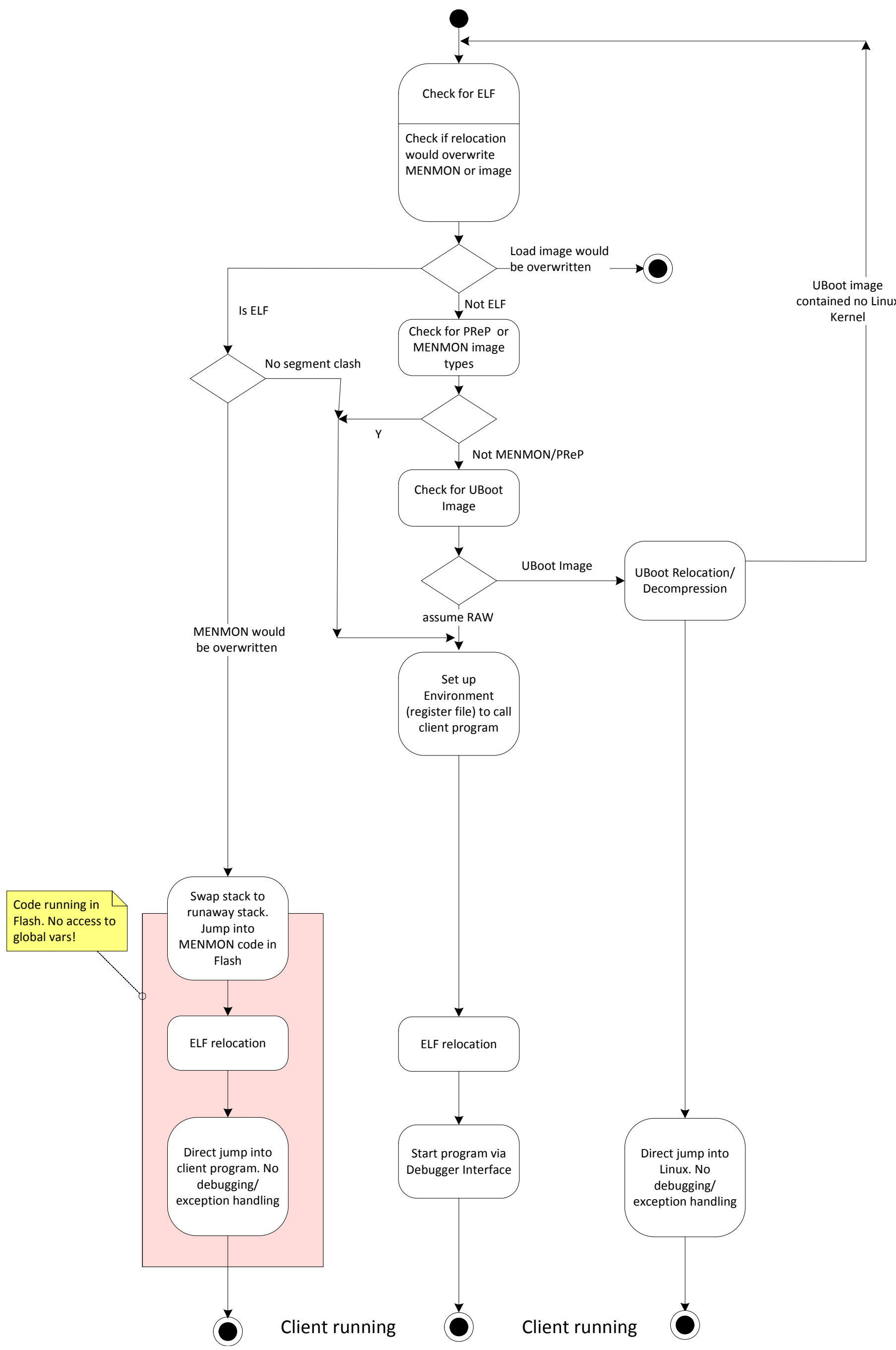
The *LOAD* parameter can be used to override this default address. When the *LOAD* parameter is used, MENMON will not check if the loaded image could overwrite MENMON data!

8.3 Image Formats

MENMON analyzes the type of the file just loaded. It can be

- RAW (no special format); e.g. uncompressed OS-9 boot
- ELF
- PReP
- PPCBOOT; e.g. compressed Linux kernel
- MENMON file

Figure 5. *Operating system execution program flow*



8.3.1 Entry Point for RAW Images / *START* Option

For RAW images, the entry point, relative to the load address, can be specified through the *START* parameter to the *DBOOT*/*NBOOT* command. (The default start offset is 0, i.e. the program execution begins at the load address.)

This parameter is ignored for other image types (see below).

See Chapter 10 MENMON on PowerPC Platforms on page 98 for calling conventions.

8.3.2 Entry Point for PReP Images

PReP images begin with a header, which contains the entry point of the program. PReP images are executed in place at the load address.

See Chapter 10 MENMON on PowerPC Platforms on page 98 for calling conventions.

8.3.3 Relocation of ELF Images

ELF files will not be executed at the load address. Instead MENMON analyzes the ELF program header and sections, and the program sections will be relocated (copied) as specified in the ELF file.

The relocation address may be any address in RAM except the runaway stack and the load image itself. Relocation code is performed from Flash if any ELF section overlaps any MENMON section in RAM.

If the relocation address would overwrite the image loading location, MENMON refuses to relocate that image and issues an error message to the console, except when the load addresses and relocation addresses are identical.

Only the virtual (*p_vaddr*) address entries in the ELF program headers are used (and treated as physical addresses), physical (*p_paddr*) addresses of the header are ignored.

See Chapter 10 MENMON on PowerPC Platforms on page 98 for calling conventions.

8.3.4 PPCBOOT/UBOOT Images

MENMON additionally supports PPCBOOT compatible boot images which are often used to boot Linux. This format is identical to UBOOT. PPCBOOT/UBOOT images can be generated using the *mkimage* tool.

PPCBOOT images have a CRC and can be compressed.

Although PPCBOOT images are typically used to boot Linux, they can be used for other operating systems as well (e.g. to compress OS-9 boot).

MENMON first analyzes and decompresses (or moves) the loaded image as specified in the PPCBOOT header. If it detects that it's not a Linux kernel, it analyzes again whether the decompressed image is ELF, RAW or PReP format (see above).

If the file to be booted is a Linux kernel, then MENMON additionally provides support for an initialized RAMdisk (*initrd*). To boot the Linux kernel with an initialized RAMdisk, you must create a multifile PPCBOOT image, where the first part is the Linux kernel and the second part is the *initrd* image. MENMON will move the *initrd* image to a 4K aligned boundary and pass the address to the Linux kernel.

See Chapter 10 MENMON on PowerPC Platforms on page 98 for calling conventions.

8.3.5 PPCBOOT Boot Info Records

The boot info records have been introduced in recent 2.4.x Linux kernels and provide an alternative way to pass parameters from the boot loader to the Linux kernel.

MENMON passes the following boot info tags (on PowerPC pointed to by R3):

- ***BI_FIRST*** – Start of boot info records.
- ***BI_CMD_LINE*** – Contains the kernel command line. This tag is missing when no command line is supplied.
- ***BI_INITRD*** – Contains info about the initial RAMdisk. This tag is missing when no *initrd* has been included.
- ***BI_MENMON_PARAMETERS*** (tag value `0x1100`) – Contains a copy of the MENMON parameter string. This tag has been added by MEN.
- ***BI_LAST*** – End of records.

8.3.6 PPCBOOT Command Line Passing / *KERPAR* Option

Persistent parameter *KERPAR* stores a string with kernel parameters to be passed to the Linux kernel.

NBOOT and *DBOOT* have an option to override this parameter from the command line.

Kernel parameters can be passed only to PPCBOOT image formats, they have no effect on other image types.

If the kernel parameters contain blanks, the string must be surrounded by apostrophes:

```
NBOOT KERPAR='ip=auto rw'
```

If neither persistent parameter *KERPAR* nor the *KERPAR*= switch is present, no command line will be passed to the Linux kernel, and the Linux kernel will use the command line specified during kernel compilation.

8.3.7 MENMON Images

For MENMON images, the entry point is stored in the MENMON header.

8.3.8 Used Image Formats for Operating Systems

Table 17. *Used image formats for operating systems*

Operating System	Used Image Format
Linux	PPCBOOT
OS-9	RAW (fixed load address)
QNX	RAW or ELF
VxWorks	ELF

9 System Parameters

MENMON uses a number of parameters to control boot-up behavior and to set up its devices and boot methods. All parameters are stored in non-volatile storage (NVS), e.g. I2C EEPROM, boot Flash.

In general, when new parameters are added, designers will keep the layout of NVS intact; i.e. old content must still be recognized.

The new MENMON 2nd Edition release is also backward-compatible with older MENMON versions.

This is also important when the primary and secondary MENMON have different versions.

9.1 Storage of Parameters

Most (or even all) parameters are usually stored in the I²C EEPROM on the CPU board.

Some data sections requiring huge amount of space can be stored in a part of the boot Flash.

Typically, the following parameters are stored in boot Flash:

- Linux kernel parameters (space for 399 chars)
- MENMON start-up string (space for 511 chars)

Some implemenations (such as EM4) allow the user to choose where to store those strings. The *useflpar* parameter determines whether to store them in CPU EEPROM (*useflpar*=0) or in Flash (*useflpar*=1).

In the latter case, if the application needs to change those parameters, the operating system needs write access to the boot Flash (which is not always the case).

If network interfaces provide their own SROM, the system parameters *nspeedX* and *nmacX* access the corresponding values inside the SROM.

9.2 Checksum Protection of Parameter Sections

Parameter data is divided into groups, e.g.

- Production data
- MENMON parameters

Each section is protected by a 4-bit "magic" ID and a 4-bit XOR checksum over the rest of the section, as shown in the following example:

```
/** Standard production data section (16 bytes) */
#define EEID_PD      0xd     /**< struct ID for EEPROD */
typedef struct {
    u_int8      pd_id;          /**< struct ID (0xD) and parity */
    u_int8      pd_revision[3]; /**< board HW revision xx.yy.zz */
    u_int32     pd_serial;      /**< board serial number */
    u_int8      pd_model;       /**< board model */
    char        pd_hwName[6];   /**< name of HW e.g. "EM04", null term. */
    u_int8      pd_resvd[1];
} EEPROD;

static u_int8 CalcParity( u_int8 *ptr, u_int32 len)
{
      u_int8 parity = 0xF;

      while( len-- ){
         parity ^= (*ptr >> 4);
         parity ^= (*ptr & 0xf);
         ptr++;
      }

      return parity;
}

/* T update pd_id: */
EEPROD p;

p.pd_id = EEID_PD << 4;
p.pd_id |= CalcParity( (u_int8 *)p+1, sizeof(EEPROD)-1 );
```

If either the magic ID or checksum is incorrect, MENMON uses defaults for all parameters of that section.

9.3 Production Data

Every MENMON implementation supports production data sections according to *MEN/eeprod.h*. The CPU EEPROM or any carrier board EEPROM has to begin with one of the sections described below, in order to allow MENMON to detect the production data and type of board automatically.

Standard production data section (16 bytes)

```
#define EEID_PD      0xd        /**< struct ID for EEPROD */

typedef struct {
    u_int8      pd_id;          /**< struct ID (0xD) and parity */
    u_int8      pd_revision[3]; /**< board HW revision xx.yy.zz */
    u_int32     pd_serial;      /**< board serial number */
    u_int8      pd_model;       /**< board model */
    char        pd_hwName[6];   /**< name of HW e.g. "EM04", null term. */
    u_int8      pd_resvd[1];
} EEPROD;
```

Extended EEPROD structure containing production/repair date (24 bytes)

```
#define EEID_PD2     0xe            /**< struct ID for EEPROD2*/
#define EEPROD2_DATE_YEAR_BIAS 1990 /**< year bias for repdat/prodat  */

typedef struct {
    u_int8      pd_id;          /**< struct ID (0xE) and parity */
    u_int8      pd_revision[3]; /**< board HW revision xx.yy.zz */
    u_int32     pd_serial;      /**< board serial number */
    u_int8      pd_model;       /**< board model */
    char        pd_hwName[6];   /**< name of HW e.g. "EM04", null term. */
    u_int8      pd_resvd[1];
    u_int16     pd_prodat;              /**< production date */
    u_int16     pd_repdat;              /**< last repair date */
    u_int8      pd_resvd2[4];
} EEPROD2;
```

Definition of EEPROD2 production/repair date fields

- Stored in big endian format
- Bits 15..9 (7 bits) contain the year since 1990 in binary format. This allows a range from 1990..2117.
- Bits 8..5 (4 bits) contain the month in binary format (1..12).
- Bits 4..0 (5 bits) contain the day of month in binary format (1..31).
- If the field is 0xFFFF, it means that the field has not been programmed.

9.4 ESM Carrier Board Parameters

ESM carrier board EEPROMs have an I2C EEPROM to allow automatic detection of the boards. ESM carrier board EEPROMs always begin with EEPROD format (not EEPROD2).

MENMON for ESM CPUs autodetect any carrier board following that standard but will also work when no carrier board EEPROM is available (the carrier board specific parameters will not exist in this case).

If one of ESM carriers AD65, AD66, EC1(N) or EC4 is detected, MENMON also supports the *c-tcal* parameter that stores calibration data for a touch screen.

9.5 MENMON Parameter String / VxWorks Bootline

MENMON maintains two strings to pass data to the operating system or client program: the MENMON parameter string and the VxWorks bootline.

On PowerPC, these strings have fixed physical addresses.

9.5.1 MENMON Parameter String Format

The MENMON parameter string stores multiple parameters with their value in a single ASCII string.

The following specification does not apply to the VxWorks bootline, which has a similar, but not identical format.

Example of a tuple string:

```
par1=value1 par2='value with blanks' par4=yy
```

- Parameter names may consist of ASCII characters [`0x21..0x7E`], except blanks and '=' characters.
- Parameter names are case-sensitive.
- A parameter name does not appear more than once in the tuple string.
- The equal sign '=' immediately follows the parameter name.
- The parameter value starts after the '=' and ends with the next blank or EOS.
- If the value contains blanks, the value must be surrounded by apostrophes.
- Apostrophes can be nested, but only if each opening apostrophe is preceeded by a '=' character, e.g:

  ```
  par2='value with blanks subpar='nested value''
  ```

- The parameter value may contain ASCII characters [`0x20..0x7E`].
- Tuples are separated by any number of blanks.

9.5.2 MENMON Parameter String Content

On PowerPC platforms, the MENMON parameter string is located at address `0x3000`. The string has a length of 1024 chars (including \0).

It contains:

- All non-persistent parameters detected by MENMON
- Production data (of CPU and possibly carrier board), for backward compatibility
- Parameters for on-board Ethernet devices with or without their own SROM.

9.5.3 VxWorks Bootline

MENMON passes a string to the client program that conforms to the standard VxWorks bootline. The parameters in the bootline can be used both by MENMON and by the operating system and client programs.

On PowerPC platforms, the VxWorks bootline string is located at address `0x4200`. The string has a length of 512 characters (including \0).

MENMON command *EE-VXBLINE* allows to change the bootline interactively (same behavior as VxWorks routine *bootChange()*). Alternatively, the parameters can be changed by the *EE-PARAM* command.

The bootline is a null-terminated ASCII string:

```
bootdev(unitnum,procnum)hostname:filename e=# b=# h=# g=# u=userid
pw=passwd f=#
tn=targetname s=startupscript o=other
```

For example:

```
enp(0,0)host:/usr/wpwr/target/config/mz7122/vxWorks e=90.0.0.2
b=91.0.0.2 h=100.0.0.4 g=90.0.0.3 u=bob pw=realtime f=2 tn=target
s=host:/usr/bob/startup o=any_string
```

9.6 Standard System Parameters

Parameters marked by "Yes" in field "Parameter String" are part of the MENMON parameter string.

Table 18. *Standard system parameters – autodetected parameters*

Parameter (alias)	Description	Standard Default	Parameter String	User Access
clun	MENMON controller unit number that MENMON used as the boot device (hexadecimal)		Yes	Read-only
cons	Selected console. Board specific, for backward compatibility. Setup according to *con0*		Yes	Read-only
cpu	CPU type as ASCII string (e.g. "MPC8245")		Yes	Read-only
cpuclkhz	CPU core clock frequency (decimal, Hz)		Yes	Read-only
dlun	MENMON device unit number that MENMON used as the boot device (hexadecimal)		Yes	Read-only
flash[0..n]	Flash size (decimal, kilobytes)		Yes	Read-only
mem0	RAM size (decimal, kilobytes)		Yes	Read-only
mem1..n	Board-specific size of additional memory (SRAM, FRAM)		Yes	Read-only
memclkhz	Memory clock frequency (decimal, Hz)		Yes	Read-only
mm	Info whether primary or secondary MENMON has been used for booting, either "smm" or "pmm"		Yes	Read-only
mmst	Status of diagnostic tests, as a string, see Chapter 7.3 Test Status on page 75 and Chapter 7.4 Diagnostic Test Status for Operating System on page 75		Yes	Read-only
pciclkhz	PCI bus clock frequency (decimal, Hz)		Yes	Read-only
rststat	Reset status code as a string, see Chapter 9.6.1 Parameter rststat (Reset Cause) on page 95		Yes	Read-only

Table 19. Standard system parameters – production data

Parameter (alias)[1]	Description	Standard Default	Parameter String	User Access
brd	Board name	-	Yes	Read-only
brdmod	Board model "mm"	-	Yes	Read-only
brdrev	Board revision "xx.yy.zz"	-	Yes	Read-only
prodat	Board production date MM/DD/YYYY	-	Yes	Read-only
repdat	Board last repair date MM/DD/YYYY	-	Yes	Read-only
sernbr	Board serial number	-	Yes	Read-only

[1] Parameters for production data of carrier boards will use prefixed parameter names, e.g. *c-brd*.

Table 20. Standard system parameters – MENMON persistent parameters

Parameter (alias)	Description	Standard Default	Parameter String	User Access
bsadr (bs)	Bootstrapper address. Used when BO command was called without arguments. (hexadecimal, 32 bits)	0	No	Read/write
cbr (baud)	Baudrate of all UART consoles (dec) (see Chapter 2.8 Selecting the Baud Rate on page 15)	9600	Yes	Read/write
con0..conN	CLUN of console 0..n. (hex) (see Chapter 2.2 Selecting Consoles on page 11)	Depends on implementation	No	Read/write
ecl	CLUN of attached network interface (hex) (see Chapter 6.1.1 Network Persistent Parameters on page 61)	`0xFF`	No	Read/write
gcon	CLUN of graphics screen (hex) (see Chapter 2.2 Selecting Consoles on page 11)	`0xFF` = auto	No	Read/write
hdp	HTTP server TCP port (decimal)	-1	No	Read/write
kerpar	Linux Kernel Parameters (399 chars max)	Empty string	No	Read/write
ldlogodis	Disable load of boot logo (bool)	0	No	Read/write
mmstartup (startup)	Start-up string (511 chars max)	Empty string	No	Read/write
nmacX	MAC address of Ethernet interface x (0..n). Format e.g. "00112233445566". Stored either in CPU board EEPROM or dedicated ROM of Ethernet device. If no valid Ethernet address, *nmacX* is an empty string.	`0xFFFFFFFFFFFF`	Yes	Read/write
nobanner	Disable ASCII banner on start-up	0	No	Read/write

Parameter (alias)	Description	Standard Default	Parameter String	User Access
nspeedX	Speed setting for Ethernet interface x (0..n). Stored either in CPU board EEPROM or dedicated ROM of Ethernet device. Possible values: *AUTO*, *10HD*, *10FD*, *100HD*, *100FD*, *1000*	AUTO	Yes	Read/write
stdis	Disable POST (bool)	0	No	Read/write
stdis_XXX	Disable POST test with name XXX (bool) Implementations only allow to disable the most important POSTs.	0	No	Read/write
stignfault	Ignore POST failure, continue boot (bool)	1	No	Read/write
stwait	Time in 1/10 seconds to stay at least in SELFTEST state (decimal) 0 = Continue as soon as POST has finished	30	No	Read/write
tdp	Telnet server TCP port (decimal)	-1	No	Read/write
tries	Number of network tries	20	No	Read/write
tto	Minimum timeout between network retries (decimal, in seconds)	0	No	Read/write
u00..u15	User parameters (hex, 16 bits)	`0x0000`	No	Read/write
updcdis	Disable auto update check (bool)	0	No	Read/write
useflpar	Store "kerpar" and "startup" parameters in boot Flash rather than in EEPROM. Parameter is used on boards where backward compatibility must be preserved (bool)	0	No	Read/write
vmode	Vesa Video Mode for grafics console (hex) (see Chapter 2.9 Selecting the Video Mode on page 15)	`0x0101`	No	Read/write
wdt	Time after which watchdog timer shall reset the system after MENMON has passed control to operating system (decimal, in 1/10 s) If 0, MENMON disables the watchdog timer before starting the operating system.	0 (disabled)	No	Read/write

Table 21. Standard system parameters – VxWorks bootline parameters

Parameter (alias)	Description	Standard Default	Parameter String	User Access
bf (bootfile)	Boot file name (127 chars max)	Empty string	No	Read/write
bootdev	VxWorks boot device name	Empty string	No	Read/write
e (netip)	IP address, subnet mask, e.g. 192.1.1.28:ffffff00	Empty string	No	Read/write
g (netgw)	IP address of default gateway	Empty string	No	Read/write
h (nethost)	Host IP address (used when booting over *NBOOT TFTP*)	Empty string	No	Read/write
hostname	VxWorks name of boot host	Empty string	No	Read/write
netaddr	Access the IP address part of *netip* parameter		No	Read/write
netsm	Access the subnet mask part of *netip* parameter		No	Read/write
procnum	VxWorks processor number (decimal)	0	No	Read/write
s	VxWorks start-up script	Empty string	No	Read/write
tn (netname)	Host name of this machine	Empty string	No	Read/write
unitnum	VxWorks boot device unit number (decimal)	0	No	Read/write

Table 22. Standard system parameters – carrier-board specific parameters

Parameter (alias)	Description	Standard Default	Parameter String	User Access
c-tcal	Touch calibration parameters (exists for all boards that have a touch controller)	0,0,0,0	Yes	Read/write

9.6.1 Parameter *rststat* (Reset Cause)

When MENMON starts up, it determines the reset cause and sets firmware parameter *rststat* accordingly.

The following values are specified. However, not every implementation may support all values:

Table 23. *rststat values*

rststat **Value**	**Description**
pwon	Power On
pdrop	Power Drop Error. A monitor circuit has detected that power supply was out of range and has reset the board. No full power loss occurred.
swrst:nn *swrst*	Board was reset by software (by means of the board's reset controller). *nn* is the hexadecimal value of an additional register that can be set through software. If *:nn* is omitted, the hardware does not support any additional register. The following values are defined for *nn*: 00 = No special reason 80 = OS panic (general) A0 = OS panic due to ECC error
wdog	Board was reset by watchdog timer unit
rbut	Board was reset by an external reset pin (e.g. reset button)
temp	Board was reset due to activation of temperature monitor
srst	Board was reset due to activation of SRESET line (PowerPC only)
hrst	Board was reset due to activation of HRESET line (PowerPC only)

9.7 Console Interface *EE* Commands

The following commands to modify persistent parameters exist on all MENMON implementations. For a list of parameter names, see Table 18, Standard system parameters – autodetected parameters, on page 91.

Basic EE Commands

EE	Show most important parameters
EE-ALL	Show all parameters
EE-<param>	Show a parameter, with help, current value and default value
EE-<param> –	Clear the parameter value
EE-<param> value	Modify a parameter The interpretation of *value* is parameter-specific.
EE-HELP <param>	Show parameter help If the parameter is missing, show help on all parameters.

Production Parameter Modification/Dump Commands

EE-PROD passwd [param value ...]	Modify a production parameter for CPU board
EE-X-PROD passwd [param value ...]	Modify a production parameter for carrier board
X	*X* is an identifier for the carrier board. For ESM carrier boards, *X* is always "c".
passwd	Special word to authorize the user.
param	*param* is one of the following:

Parameters to *EE-PROD*	Corresponding System Parameter[1]	Description
name	*brd*	Board name
mod	*brdmod*	Board model "mm"
rev	*brdrev*	Board revision "xx.yy.zz"
ser	*sernbr*	Board serial number
pd	*prodat*	Board production date MM/DD/YYYY (only with EEPROD2)
rd	*repdat*	Board last repair date MM/DD/YYYY (only with EEPROD2)

[1] Carrier board parameters are prefixed with "X-".

It is possible to modify only one or all production parameters at the same time.

Restore Default Parameters

EE-DEF	Restores the default value of all parameters, except product data such as serial number, model name etc. *EE-DEF* affects parameters in all non volatile storage sections, including the sections in carrier board EEPROMs (if any).

Restore Virgin State of Persistent Storage

EE-ERASEME [<nvs-name>]		Erases/fills the specified non-volatile storage (NVS) section with virgin values. E.g. for sections in EEPROM, erases the EEPROM.
	nvsname	NVS section to erase. If no *nvsname* is passed, this command erases all known sections.
EE-NVS		Get a list of available "nvs" sections
EER-ERASEME [<addr>]		Optional implementation-specifc command that can directly access NV storage
	addr	Hardware specific address (e.g. SMB address).

Dump Raw Content of Persistent Storage

EE-DUMP <nvsname> [<numbytes>]		Dumps the specified non-volatile storage
	nvsname	NVS section to dump
	numbytes	Number of bytes to dump
EER-DUMP [<addr>]		Optional implementation-specifc command that can directly access NV storage
	addr	Hardware specific address (e.g. SMB address)

10 MENMON on PowerPC Platforms

This chapter describes special features available only on PowerPC platforms.

10.1 Cache Control

By default, instruction cache is on and data cache is off while MENMON is executed.

However, MENMON 2nd Edition is also designed to run with enabled data cache. Some internal operations temporarily enable data caching:

- ECC memory fill
- Diagnostic Test "Extended RAM Test"
- BitBlit operations in some frame-buffer graphic drivers
- Degraded Mode with data cache locking

10.1.1 Cache Control Commands

Command-line interface commands to manipulate cache state:

DCACHE 0FF|ON	Enable or disable data cache
ICACHE 0FF|ON	Enable or disable instruction cache

10.1.2 Common CPU State for Operating System/Program Calling

Regardless of the type of image, the CPU will be in the following state:

On 603e CPUs:

- Interrupts are disabled (MSR.EE is cleared).
- CPU is in Big Endian Mode.
- MMU is enabled. BATs are set up.
- Instruction cache is enabled.
- Data cache disabled (or enabled if DCACHE ON has been called).

On 85xx CPUs:

- Interrupts are disabled (MSR.EE is cleared).
- CPU is in Big Endian Mode.
- TLB1 entries and LAWBARs initialized.
- Instruction cache is enabled.
- Data cache disabled (or enabled if DCACHE ON has been called).
- L2 cache is enabled.

10.1.2.1 Operating System/Program Calling for ELF, RAW, PReP

User level registers are set as follows (on all PPC architectures):

- R1 is set to the top of runaway stack – 512 bytes.
- R3 is set to 0.
- R4 is set to the image loading address. (Not the relocation address!)
- R5..R7 are cleared.
- R8..R31 are undefined.

10.1.2.2 Operating System/Program Calling for PPCBOOT

When MENMON calls the Linux kernel inside a PPCBOOT image, the registers of the CPU are in the following state:

- R1: Normal MENMON stack
- R3: Points to an array of boot info records
- R4: Start of initial RAMdisk (0 if none)
- R5: End of initial RAMdisk+1
- R6: Start of kernel command line
- R7: End of kernel command line+1
- R8..R31 are undefined

Note that the usage of R3 does not conform to the originial *uboot* boot monitor. *uboot* passes the address of a board info structure in R3. MENMON uses boot info records to be more flexible.

10.2 Special Processor Support

10.2.1 82XX Processors

10.2.1.1 Degraded Mode

In Degraded Mode, 82XX MENMONs use the 16-KB L1 data cache as data memory. The entire L1 data cache is allocated and locked for this purpose.

MENMON instructions are executed from boot Flash.

This is the initial start-up mode and the "Degraded Mode", if MENMON detects that DRAM is not working.

Table 24. *Address map for 82XX processors in Degraded Mode*

Address	Size	Description
`0xD000 0000..0xD000 0BFF`	3 KB	Initial Stack (L1 cache)
`0xD000 0C00..0xD000 17FF`	3 KB	Initial Heap (L1 cache)
`0xD000 1800..0xD000 3FFF`	10 KB	Initial Data (L1 cache)
`0xFFFx 0000..0xFFxF FFFF`	512 KB	Text + Reloc (x=8 for primary, 0 for secondary)

During start, MENMON copies the initialized data from Flash to RAM and fills the *.bss* section with zeroes.

Restrictions in Degraded Mode:

- In secondary MENMON, no exception handling is possible as exceptions could not be directly directed to secondary MENMON. (MSR.IP allows only to switch exception vector prefix between `0x0` and `0xFFF0 0000`)
- In Degraded Mode, only the basic command-line commands will work, since very little memory is available.
- DMA to system memory is not possible (DMA to cache not possible); therefore no network operation is possible.
- Boot Flash programming in Degraded Mode is not possible (as MENMON executes instructions from Flash).

10.2.1.2 System Parameters

Currently no 82XX specific system parameters are defined.

10.2.1.3 Exception Handling

As soon as MENMON enters its full mode, all exceptions are trapped and most of them are reported to the user.

In Degraded Mode exception handling is not possible (any exception will cause undefined results).

Table 25. *82XX exception handling*

Exception	Exception Name	Handling Method
0x200	Machine Check	Reported to console
0x300	Data Storage	Reported to console
0x400	Instruction Storage	Reported to console
0x500	Interrupt	Fetch vector from interrupt controller Report interrupt to console
0x600	Alignment	Reported to console
0x700	Program	If client program is running, check for break-point. Otherwise report as exception
0x800	FPU unavail	Reported to console
0x900	Decrementer	Reported to console
0xC00	System Call	If client program is running, try to handle system call. Otherwise report as exception
0xD00	Debug	If client program is running, handle as single step exception. Otherwise report as exception
0x1000	ITL Miss	Reported to console
0x1100	DTL Miss	Reloads TLB when faulted address ok, otherwise reports exception to console
0x1200	DTS Miss	Reloads TLB when faulted address ok, otherwise reports exception to console
0x1300	Instr. addr. Break	Reported to console
0x1400	SMI	Reported to console

10.2.1.4 Machine Check Handling

On machine checks, additional information is printed to the console, for example MPC8245 error detection registers.

10.2.2 85XX Processors

10.2.2.1 Degraded Mode

In Degraded Mode, 85XX MENMONs use on-chip L2 SRAM as data memory. L2 SRAM is configured as 256KB SRAM for this purpose.

MENMON instructions are executed from boot Flash.

In Degraded Mode, most of MENMON commands will work, including networking over on-chip interfaces.

Boot Flash programming in Degraded Mode is not possible (as MENMON executes instructions from Flash).

10.2.2.2 System Parameters

Table 26. *85XX specific system parameters (autodetected parameters)*

Parameter (alias)	Description	Standard Default	Parameter String	User Access
ccbclkhz	CCB clock frequency (decimal, Hz)		Yes	Read-only
brgclkhz	CPM baud rate generator clock frequency (decimal, Hz), missing on CPUs without CPM		Yes	Read-only
Immr	Physical address of CCSR register block		Yes	Read-only

10.2.3 5200 Processors

10.2.3.1 Degraded Mode

Degraded Mode for 5200 processors is the same as for 82XX processors, see Chapter 10.2.1.1 Degraded Mode on page 100.

10.2.3.2 System Parameters

Table 27. *52XX specific system parameters (autodetected parameters)*

Parameter (alias)	Description	Standard Default	Parameter String	User Access
inclkhz	CPU input/oscillator clock frequency in Hz		Yes	Read-only
xlbclkhz	XLB (platform) clock frequency in Hz		Yes	Read-only
ipbclkhz	IPB clock domain frequency in Hz		Yes	Read-only

10.3 Debugger

10.3.1 Debugger Features

MENMON includes a simple assembly level debugger, featuring

- Built in assembler/disassembler
- Manipulation of all user-level registers
- Instruction breakpoints
- Single stepping
- Exception handling (either caused by client program or MENMON)

Restrictions of MENMON debugger:

- No support for supervisor registers
- On-chip hardware breakpoints not supported
- Explanation of register bits removed (was present in MENMON < 2nd Edition)
- Client program must not overwrite MENMON code/data memory areas
- Client program must not modify exception prefix (MSR.IP on 603E, IPVR on E500)
- Client program must not alter *sprg0..3*

10.3.2 Register File

The register file saves the state of the client program.

It consists of registers

- GPR0..31 (named R0..31)
- IP (instruction pointer)
- MSR (machine state register)
- CR, CTR, LR, XER
- FPR0..31 (named F0..31), FPSR (on CPUs with FPU)

Register File Commands

.	Dump current IP, disassemble instruction at IP, dump CR, MSR, all GPRs
.all	Dump all known registers
.reg	Dump specific register, e.g. *.msr* or *.r16*
.reg value	Modify specific register, e.g. *.msr 3090* or *.r16 abcdef01*

10.3.3 GO Command

GO [<addr>]	Jump to user program
addr	Start address of user program. If missing use value of IP register

Command *GO* executes a client program. It is available in two forms:

- Execute program at specified address:

```
GO 100000
```

- Continue program (at address stored in register file "IP" register)

```
GO
```

In any case, before MENMON jumps to the program,

- it activates possible breakpoints
- it restores the client program's state from the values stored in the register file.

The client program continues execution until

- a break point is encountered
- a client program issues a system call to enter MENMON (see Chapter 10.4.8 System Call RETURN on page 112).

There is no way to abort a running program from the console.

The client program is responsible to serve the watchdog timer (if activated).

10.3.4 Single Step Command

S [<addr>]	Single step user program
addr	If present, begin stepping at *addr*

This command is similar to the *GO* command but executes only exactly one instruction, using the trace facility of CPU:

- Single step program at specified address

```
S 100000
```

- Single step at current IP (at address stored in register file "IP" register)

```
S
```

After each step, MENMON displays a register dump (same format as with "." command) and enters its command line interface.

By contrast to the *GO* command, single stepping does not activate break points.

10.3.5 Break Points

MENMON supports up to four break points in a user program.

B **BD**		Display all break points
B<no> <addr>		Set a break point
	no	Break point number [0..3]
	addr	Absolute address of the instruction where MENMON shall set a break point
	Example: Set break point 1 at address `0x1000` `B1 1000`	
BC [<no>]		Clear a break point If no number is given, clears all break points
	no	Break point number [0..3]

Break points are implemented using illegal instruction opcodes (software break points).

Before MENMON jumps into the client program, the instruction at the break point address is replaced with illegal instruction opcode.

When the CPU tries to execute the illegal opcode, it generates a program exception. The MENMON program exception handler then checks if the faulty instruction address belongs to a break point. If so, it restores the original instruction, displays a register dump and enters the command-line interface.

10.3.6 Line-by-Line Assembler

MENMON versions dating from earlier than 2008 may still include the *AS* and *DI* assembler commands. However, newer versions of MENMON no longer support these commands for licensing reasons.

AS <addr> [<cnt>]		Assemble memory
	addr	Memory target address
	cnt	Number of bytes to assemble
DI [<addr>] [<cnt>]		Disassemble memory Without arguments, *DI* disassembles the previously specified address
	addr	Memory address to disassemble
	len	Number of bytes to disassemble

10.4 PPCBug System Calls

This allows system calls from user programs. MENMON implements a small subset of the system calls implemented in Motorola's PPCBug. The implemented system calls are binary-compatible with PPCBug.

The system calls can be used to access selected functional routines contained within the debugger, including input and output routines. The System Call handler may also be used to transfer control to the debugger at the end of a user program.

10.4.1 Invoking System Calls

The System Call handler is accessible through the *SC* (system call) instruction, with exception vector `0x00C00` (System Call Exception). To invoke a system call from a user program, insert the following code into the source program.

```
ADDI R10,R0,$XXXX
SC
```

- The code corresponding to the particular system routine is specified in register R10.
- Parameters are passed and returned in registers R3 to R*n*, where *n* is less than10.
- $*XXXX* is the 16-bit code for the system call routine, and *SC* is the system call instruction (system call to the debugger). Register R10 is set to `0x0000xxxx`.

10.4.2 System Call *BRD_ID*

Name	*BRD_ID* – Return pointer to board ID packet
Code	`$0070`
Description	This routine returns a pointer in R03 to the board identification packet. The packet is built at initialization time. The format of the board identification packet is shown below. MENMON only implements some fields of the original PPCBug system call.

***Table 28.** MENMON system calls – BRD_ID fields*

	31 … 24	23 … 16	15 … 8	7 … 0
0x00	Eye Catcher			
0x04	Reserved			
0x08	Packet Size		Reserved	
0x0C	Reserved			
0x10	Reserved			
0x14	CLUN		DLUN	
0x18	Reserved			
0x1C	Reserved			

Eye Catcher	Word containing ASCII string "BDID"
Packet Size	Half-word containing the size of the packet
CLUN	CLUN of device used for booting
DLUN	DLUN of device used for booting

Entry Conditions	-	
Exit Conditions different from Entry	R03: Address (word)	Starting address of ID packet

10.4.3 System Call *OUT_CHR*

Name	*OUT_CHR* – Output character routine
Code	`$0020`
Description	This routine outputs a character to the default output port.
Entry Conditions	R03: Bits 7..0 Character (byte)
Exit Conditions different from Entry	Character is sent to the default I/O port.

10.4.4 System Call *IN_CHR*

Name	*IN_CHR* – Input character routine
Code	`$0000`
Description	*IN_CHR* reads a character from the default input port. The character is returned in the LSB of R03.
Entry Conditions	-
Exit Conditions different from Entry	R03: Bits 7..0 contain the character returned R03: Bits 31..8 are zero

10.4.5 System Call *IN_STAT*

Name	*IN_STAT* – Input serial port status routine
Code	`$0001`
Description	*IN_STAT* is used to see if there are characters in the default input port buffer. R03 is set to indicate the result of the operation.
Entry Conditions	No arguments required
Exit Conditions different from Entry	R03: Bit 3 (ne) = 1; Bit 2 (eq) = 0 if the receiver buffer is not empty. R03: Bit 3 (ne) = 0; Bit 2 (eq) = 1 if the receiver buffer is empty.

10.4.6 System Call *RTC_RD*

Name	*RTC_RD* – Read the RTC registers
Code	`$0053`
Description	RTC_RD is used to read the Real-Time Clock registers. The data returned is in packed BCD. Fills a buffer of 8 bytes. The order of the data in the buffer is: ***Table 29.*** *MENMON system calls - RTC_RD buffer data* YY | MM | DD | dd | H | M | S | 0 Begin buffer … Buffer + eight bytes YY: Year (2 nibbles packed BCD) MM: Month (2 nibbles packed BCD) (1..12) DD: Day of month (2 nibbles packed BCD) (1..31) dd: Always 0 H: Hour (2 nibbles packed BCD) (0..23) M: Minutes (2 nibbles packed BCD) (0..59) S: Seconds (2 nibbles packed BCD) (0..59)
Entry Conditions	R03: Buffer address where RTC data is to be returned
Exit Conditions different from Entry	Buffer now contains date and time in packed BCD format.

10.4.7 System Call *DSK_RD*

Name	*DSK_RD* – Disk read routine
Code	`$0010`

Description

This routine is used to read blocks of data from the specified disk device. Information about the data transfer is passed in a command packet which has been built somewhere in memory. (The user program must first manually prepare the packet.) The address of the packet is passed as an argument to the routine. The command packet is eight half-words in length and is arranged as follows:

Table 30. *MENMON system calls – DSK_RD fields*

<table>
<tr><th></th><th>15</th><th>8</th><th>7</th><th>0</th></tr>
<tr><td>0x00</td><td colspan="2">CLUN</td><td colspan="2">DLUN</td></tr>
<tr><td>0x02</td><td colspan="4">Status Half-Word</td></tr>
<tr><td>0x04</td><td colspan="2" rowspan="2">Memory Address</td><td colspan="2">Most Significant Half-Word</td></tr>
<tr><td>0x06</td><td colspan="2">Least Significant Half-Word</td></tr>
<tr><td>0x08</td><td colspan="2" rowspan="2">Block Number (Disk)</td><td colspan="2">Most Significant Half-Word</td></tr>
<tr><td>0x0A</td><td colspan="2">Least Significant Half-Word</td></tr>
<tr><td>0x0C</td><td colspan="4">Number of Blocks</td></tr>
<tr><td>0x0E</td><td colspan="2">Flag Byte</td><td colspan="2">Address Modifier</td></tr>
</table>

CLUN	Logical Unit Number (LUN) of controller to use
DLUN	Logical Unit Number (LUN) of device to use
Status	This status half-word reflects the result of the operation. It is zero if the command completed without errors, otherwise it contains MENMON's internal status code.
Memory Address	Address of buffer in memory. Data is written starting at this address.
Block Number	For disk devices, this is the block number where the transfer starts. Data is read starting at this block.
Number of Blocks	The number of blocks to read from the disk. For streaming tape devices, the actual number of blocks transferred is returned in this field.
Flag Byte	Not implemented by MENMON
Address Modifier	Not used

Entry Conditions	R03: 32-bit address of command packet
Exit Conditions different from Entry	Status half-word of command packet is updated. Data is written into memory. R03: Bit 3 (ne) = 1; Bit 2 (eq) = 0 if errors. R03: Bit 3 (ne) = 0; Bit 2 (eq) = 1 if no errors.

10.4.8 System Call *RETURN*

Name	*RETURN* – Return to monitor
Code	`$0063`
Description	This routine invokes the MENMON command line interface. The user program is stopped.
Entry Conditions	-

11 MENMON Command Reference

The following table gives all MENMON commands that can be entered on the MENMON prompt:

Table 31. *MENMON – command reference*

Command	Description	Details in...
.[<reg>] [<val>]	Display/modify registers in debugger model	Chapter 10.3.2 Register File on page 104
ARP	Dump network stack ARP table	Chapter 6.5 Network Status Commands on page 66
AS <addr> [<cnt>]	Assemble memory (no longer supported by newer MENMON versions)	Chapter 10.3.6 Line-by-Line Assembler on page 106
B[DC#] [<addr>]	Set/display/clear breakpoints	Chapter 10.3.5 Break Points on page 106
BIOS_DBG <mask> [net] | cons <clun>	Set MENMON BIOS or network debug level, set debug console	Chapter 4.2.7 Set Debug Options on page 45
BO [<addr>] [<opts>]	Call OS bootstrapper	Chapter 8.1.3 Boot from an Existing Image using BO on page 80
BOOTP [<opts>]	Obtain IP config via BOOTP	Chapter 6.3 Obtaining the IP Configuration via BOOTP on page 65
C[BWLLNAX#] <addr> [<val> ...]	Change memory	Chapter 4.2.2 Memory Commands on page 38
CHAM-LOAD [<addr>]	Load FPGA	Chapter 5.5.2.2 Force FPGA Loading on page 60
CHAM [<clun>]	Dump FPGA Chameleon table	Chapter 5.5.1 Chameleon Table Support on page 59
CONS	Show active consoles	Chapter 2.6 Console-Related Commands on page 12
CONS-ACT <clun1> [<clun2>] ...	Test console configuration	Chapter 2.6 Console-Related Commands on page 12
D [<addr>] [<cnt>]	Dump memory	Chapter 4.2.2 Memory Commands on page 38
DBOOT [<clun>] [<dlun>] [<opts>]	Boot from disk	Chapter 8.1.1 Disk Boot using DBOOT on page 76
DCACHE 0FF|ON	Enable/disable data cache	Chapter 10.1.1 Cache Control Commands on page 98
DI [<addr>] [<cnt>]	Disassemble memory (no longer supported by newer MENMON versions)	Chapter 10.3.6 Line-by-Line Assembler on page 106

Command	Description	Details in...
DIAG [<which>] [VTF]	Run diagnostic tests	Chapter 7.1 Diagnostic Tests from Command Line on page 74
DSKWR <args>	Write blocks to RAW disk	Chapter 5.2.4 Reading from/ Writing to RAW Disks on page 52
DSKRD <args>	Read blocks from RAW disk	Chapter 5.2.4 Reading from/ Writing to RAW Disks on page 52
EER[-xxx] [<arg>]	Raw serial EEPROM commands	Chapter 9.7 Console Interface EE Commands on page 96
EE[-xxx] [<arg>]	Persistent system parameter commands	Chapter 9.7 Console Interface EE Commands on page 96
ERASE <D> [<O>] [<S>]	Erase Flash sectors	Chapter 4.2.3.5 Erasing Flash Sectors on page 42
ESMCB-xxx	ESM carrier commands	Chapter 4.2.6 ESM Carrier Board Commands on page 45
FI <from> <to> <val>	Fill memory (byte)	Chapter 4.2.2 Memory Commands on page 38
GO [<addr>]	Jump to user program	Chapter 10.3.3 GO Command on page 105
H HELP	Print help	Chapter 4.2.1.3 Help on Commands on page 37
I [<D>]	List board information	Chapter 4.2.5 Show Board/ CPU Information on page 44
ICACHE 0FF|ON	Enable/disable instruction cache	Chapter 10.1.1 Cache Control Commands on page 98
IOI	Scan for BIOS devices	Chapter 5.1.3 Display MENMON BIOS Tables on page 47
LOGO	Display MENMON start-up screen	Chapter 3.4 MENMON Start-up Screen on page 18
LS <clun> <dlun> [<opts>]	List files/partitions on device	Chapter 5.2.3 Listing Disk Partitions and Contents on page 51
MC <addr1> <addr2> <cnt>	Compare memory	Chapter 4.2.2 Memory Commands on page 38
MII <clun> [<reg>] [<val>]	Ethernet MII register command	Chapter 6.8.3 Diagnostic Command for Ethernet PHY on page 71
MO <from> <to> <cnt>	Move (copy) memory	Chapter 4.2.2 Memory Commands on page 38
MS <from> <to> <val>	Search pattern in memory	Chapter 4.2.2 Memory Commands on page 38

Command	Description	Details in...
MT [<opts>] <start> <end> [<runs>]	Memory test	Chapter 4.2.2 Memory Commands on page 38
NBOOT [<opts>]	Boot from Network	Chapter 8.1.2 Network Boot using NBOOT on page 78
NDL [<opts>]	Update Flash from network	Chapter 6.4 Network Load & Program Command on page 65
NETSTAT	Show current state of networking parameters	Chapter 6.5 Network Status Commands on page 66
PCI-VPD[-] <devNo> [<busNo>] [<capId>]	PCI Vital Product Data dump	Chapter 5.4.2 PCI Commands on page 57
PCIC <dev> <addr> [<bus>] [<func>]	PCI config register change	Chapter 5.4.2 PCI Commands on page 57
PCID[+] <dev> [<bus>] [<func>]	PCI config register dump	Chapter 5.4.2 PCI Commands on page 57
PCI	PCI probe	Chapter 5.4.2 PCI Commands on page 57
PCIR	List PCI resources	Chapter 5.4.2 PCI Commands on page 57
PFLASH <D> <O> <S> [<A>]	Program Flash	Chapter 4.2.3.4 Update from RAM using PFLASH on page 41
PGM-XXX <args>	Media copy tool	Chapter 4.2.3.3 Update from Local Disk using PGM-xxx on page 41
PING <host> [<opts>]	Network connectivity test	Chapter 6.5 Network Status Commands on page 66
RTC[-xxx] [<arg>]	Real time clock commands	Chapter 4.2.4 Get/Set the RTC Time on page 43
S [<addr>]	Single step user program	Chapter 10.3.4 Single Step Command on page 105
SERDL [<passwd>]	Update Flash using YModem protocol	Chapter 4.2.3.1 Update via Serial Interface using SERDL on page 40
SETUP	Open Setup Menu	Chapter 4.1 Screen-Oriented Menu User Interface on page 21
USB [<bus>]	Init USB controller and devices on a USB bus	Chapter 5.2.5 Displaying and Modifying USB Settings on page 53
USBT	Shows the USB device tree for the current bus	
USBDP [<bus p1..p5>] [-d<x>]	Display/modify USB device path	