



IPnexus®
PTBMON

Boot Monitor

User's Guide

Version 2.0

Performance Technologies
205 Indigo Creek Drive
Rochester, NY 14626
585.256.0200
support@pt.com

www.pt.com

© 2007 Performance Technologies, Inc.
All Rights Reserved.



Document Revision History

Part Number	Date	Explanation of changes
106p0104.10	October 28, 2004	Initial Release
106p0104.20	September 27, 2007	Revised for AMC131 functionality

Copyright Notice

© Copyright 2007 by Performance Technologies, Inc. All Rights Reserved.

IPnexus® and the Performance Technologies logo are trademarks and registered trademarks of Performance Technologies, Inc.

CompactPCI® and PICMG® is a registered trademark of the PCI Industrial Computer Manufacturers Group.

All other product and brand names are trademarks or registered trademarks of their respective owners.

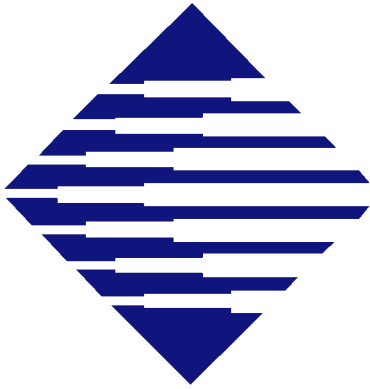
This document is the sole property of Performance Technologies Inc.

Errors and Omissions

Although diligent efforts are made to supply accurate technical information to the user, occasionally errors and omissions occur in manuals of this type. Refer to the Performance Technologies, Inc. Web site to obtain manual revisions or current customer information:

<http://www.pt.com>.

Performance Technologies, Inc., reserves its right to change product specifications without notice.

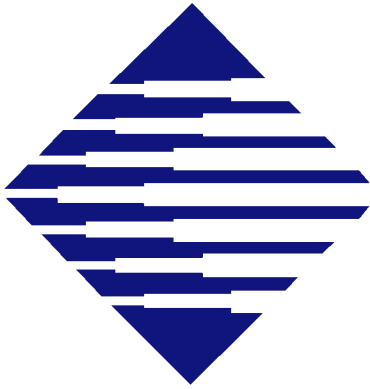


Contents

Chapter 1: PTBMON Commands	7
Introduction	7
Boot and Load Commands	8
boot	8
berase	9
load	9
oload	10
Miscellaneous Commands	11
Call	11
flush	11
flash	12
devls	13
reboot	14
Debugger Commands	15
g	15
l	15
Shell Commands	16
more	16
sh	17
eval	19
hi	19
about	21
h	21
vers	23
stty	23

date	24
Network Commands	26
ifaddr	26
ping	27
Memory Commands	28
m	28
compare	29
fill	30
mt	30
ecc	31
d	32
copy	33
search	34
viewmem	35
PCI Commands	36
pcicfg	36
pciscan	37
Environment Commands	38
env	38
unset	39
set	39
eset	41
PTI Commands	42
bFlash	42
setprom	43
initboot	44
getboot	45
dumpEmac	46
initprom	46
getprom	47
setboot	47
dumpPld	48
dumpMal	49
notice	49
ipmc	50
dumpPhy	50
dumpTsec	51

dumpiic	51
initiic	52
readiic	52
writeiic	53
bUpdate	53
cksum	54
rt	54
sd	55
sdRead	55
sdWrite	56
The Bootdata Structure	57
The Promdata Structure	59
Installing and Booting an Image from FLASH	60



PTBMON Commands

Introduction

The commands for PTBMON are broken down into the following areas:

- [“Boot and Load Commands” on page 8](#)
- [“Miscellaneous Commands” on page 11](#)
- [“Debugger Commands” on page 15](#)
- [“Shell Commands” on page 16](#)
- [“Network Commands” on page 26](#)
- [“Memory Commands” on page 28](#)
- [“PCI Commands” on page 36](#)
- [“Environment Commands” on page 38](#)
- [“PTI Commands” on page 42](#)

The bootdata structure, promdata structure, and instructions for installing and booting an image from FLASH are also included in this chapter:

- [“The Bootdata Structure” on page 57](#)
- [“The Promdata Structure” on page 59](#)
- [“Installing and Booting an Image from FLASH” on page 60](#)

Boot and Load Commands

The Boot and Load class of commands are used to load programs into memory. These commands accept S-Record and ELF file formats. Some of the commands will accept raw binary files. The `boot` and `load` commands use tftp to download the files, while the `oload` command uses the serial console port. The boot command will load and start execution of the program while the load and oload command may be used with the `flash` and `bFlash` commands to save the executable in FLASH.

boot

Description

The boot command loads and starts a program. The boot command is similar to the load command except that it also starts execution of the program that has been loaded. The command can read raw binary files, S-Record files and files in the ELF format as used in:

- Algorithmics SDE-MIPS
- Newer SGI compilers
- Systems compliant with the MIPS/ABI standard
- Older MIPS ECOFF format
- OpenBSD PowerPC 32-Bit ELF and MIPS 32Bit ELF
- Some Linux for PowerPC implementations

The boot command may return a large number of different error messages, relating to network problems or file access permissions on the remote host. For a file to be loaded via TFTP it must be publicly readable, and it may have to be in a directory which is acceptable to the remote server.

Command Format

```
boot [-r][-o offs][-e entry] path
```

<code>-r</code>	load raw file
<code>-o<offs></code>	load offset
<code>-e<entry></code>	entry address
<code>path</code>	path and filename,

For example, to boot a file via tftp:

```
PTBMON boot tftp://192.168.0.100/file.bin
```

Environment

The command uses no environment variables.

See Also

The `load` and `flash` commands.

berase

Description

The berase command allows erasing any sector of the boot FLASH, except for the boot program.

Command Format

```
berase sector
sector          which sector to erase
```

Functional Description

The berase command erases the specified sector from the boot FLASH. The top 16 sectors of the boot FLASH contains the PTBMON image and can not be erased using this command.

Environment

No environment variables effect this command.

See Also

The [initprom](#) and [initboot](#) commands for restoring these sectors.

load

Description

The load command loads a program into memory.

Command Format

```
load [-ir][-o offs]
-i          ignore checksum errors
-o<offs>   load offset
-r          load raw file
path       path and filename
```

Functional Description

The load command is similar to the [boot](#) command except that it does not start execution of the program that has been loaded. Instead it returns to the PTBMON prompt awaiting the next command.

The command can read raw binary files, S-Record files and files in the ELF format as used in:

- Algorithmics SDE-MIPS
- Newer SGI compilers
- Systems compliant with the MIPS/ABI standard
- OpenBSD PowerPC 32-Bit ELF and MIPS 32Bit ELF.
- Some Linux for PowerPC implementations

The command saves the entry point of the program. Therefore, to execute the downloaded program, only the [g](#) command is required.

The command may return a large number of different error messages, relating to network problems or file access permissions on the remote host. For a file to be loaded via TFTP it must be publicly readable, and it may have to be in a directory which is acceptable to the remote server.

Environment

The command uses no environment variables.

See Also

The [boot](#) command and [url/pathname](#) description.

oload

Description

The oload command allows loading memory over the Console Serial Port.

Command Format

```
oload [-i][-o ofs]
```

- i ignore checksum errors
- v verbose messages

Functional Description

The oload command is similar to the [load](#) command except it uses the serial Console port to accept data. The command can read S-Record files.

Environment

The command uses no environment variables.

See Also

The [load](#) command.

Miscellaneous Commands

These are miscellaneous commands which allow running a function when its memory address is known, flushing either the data or instruction caches, and erasing and programming the application flash, among other functions.

Call

Description

The call command calls a function and prints the return value.

Command Format

```
call addr [val|-s str]..
```

Functional Description

The call command calls the function which address is given by the addr expression argument with the parameters given as arg and -s str. A maximum of five arguments can be used. String and integer values can be mixed in any order. String arguments containing spaces must be quoted.

When the called function returns the value returned is displayed in hex and decimal.

Environment

The command uses no environment variables.

See Also

The go ([g](#)) command.

flush

Description

The flush command flush the processor caches.

Command Format

```
flush [-di]  
-i          flush I-cache  
-d          flush D-cache
```

If the flush command is used without any arguments both the instruction cache and the data cache will be flushed.

Functional Description

The flush command invalidates the instruction cache and will writeback the data cache.

Environment

The command uses no environment variables.

flash

Description

The flash command erases or programs available flash memory.

Command Format

```
flash [-qev] [addr[ size[ data]]]
-q          display info about flash memory
-e          erase flash
-v          verify flash
flashaddr  first address in flash to erase and/or program
size       size to erase and/or program
dataaddr   first address of data to program
```

Functional Description

The flash command programs flash memory on the target board. The available flash memories can be queried by typing the flash command without any parameters or with the -q option. The output from the query is target system dependent. The following is an example.

```
PTBMON flash -q
Available FLASH memory
  Start   Size     Width Sectorsize Type
ff000000 00800000 8*8   00040000  i28F160
fff00000 00080000 1*8   00010000  Am29F040
PTBMON
```

The start address is where in the memory map the flash area starts. The size is the size in the memory map that this flash device occupies. The width indicates how many flash devices and what width each device have. The first value is the number of devices and the second is the width in bits of each device. The total word length can be determined by multiplying the width with the number of devices. The sector size is the minimum size that can be erased and reprogrammed without affecting any other data in the flash. Finally the type indicates what type of devices the flash area is built of.

The `-e` option should be used to erase the entire flash or a part of a flash area. To erase the entire flash area only the start address of the area has to be given. To erase a part of a flash area, the start address and the desired size should be given. The erase operation takes place in multiples of the sector size.

To program a flash area the flash command require three arguments, the `flashaddr`, the `size` and the `dataaddr` arguments. The `flashaddr` argument determines where in the flash area the programming should start, the `size` argument determines the number of bytes that should be programmed and the `dataaddr` argument tells where the data to program into the flash is found. Unless the `-e` option is given, programming is performed in 'patch' mode. Normally the `-e` option should be used to erase the flash before programming.

Environment

The command uses no environment variables.

devls

Description

The `devls` command lists devices

Command Format

```
devls [-an]
```

```
-a          show all device types
```

```
-n          show network devices
```

Functional Description

The `devls` command is used to produce a short list of all devices installed by the boot monitor.

Environment

The command uses no environment variables.

reboot

Description

The reboot command restarts the target.

Command Format

`reboot`

The reboot command has no parameters.

Functional Description

The reboot command is used to restart PTBMON by asserting a hard reset.

When the reboot command is executed, all settings and temporary variables will be lost and all actions like autoboot etc. will be executed as if a panel or power on reset was issued.

Environment

The command uses no environment variables.

Debugger Commands

These are a very basic set of commands which can be used for debugging programs. They allow defining and listings symbols, starting execution of a program and disassembling memory. The `g` command is frequently used after using the `load` command.

g

Description

The `g` command start program execution.

Command Format

```
g [-e adr][-- args]
```

`-e <adr>` start execution at address `adr`. This breakpoint is removed next time execution halts.

`-- <args>` indicates that the argument or arguments `args` are to be passed to the client program. No more options to the `g` command itself can be given after this option

Invoking the `g` command with no options starts execution at the location saved by the `load` command.

Functional Description

The `g` command starts program execution. If the user does not specify a starting address, execution starts at the location saved by the `load` command.

If the user specifies the `--` option, then PTBMON will pass all arguments after `--` to the client program.

Environment

The command uses no environment variables.

I

Description

The `I` command disassembles and display instructions in memory.

Command Format

```
I [adr [cnt]]
```

`adr` address where disassembly should start

`cnt` disassemble `cnt` instructions and then stops

Invoking the I command with no options starts disassembly from the location given by CPC until quit from 'more' is done.

Functional Description

The I command is used to examine memory by disassembling machine code into human readable text. If a starting address is not specified, disassembly starts at the current value of the CPC register, otherwise the disassembly starts at the address given in the addr argument. The output is sent through 'more' and quitting more terminates the command. The cnt argument can be used to disassemble cnt number of instructions and then stop. When the cnt argument is given, the output is not sent through 'more' but all lines are output until all lines have been printed.

Environment

The command uses no environment variables.

See Also

The dump (d) and [more](#) commands.

Shell Commands

The shell commands cover a broad spectrum of functionality. Some commands provide information about the program. Other commands provide information regarding the actions that may be performed at the command-line prompt. The [date](#) command allows setting and displaying the date and the [stty](#) command performs volatile changes to the console's terminal settings.

more

Description

The more command paginates console output.

Command Format

The more command is a built-in function and can not be invoked from the command line.

Functional Description

The more command is not specified by the user on the command line, but is implicitly used by most commands. After displaying the number of lines according to the value of the moresz environment variable, the more command displays the prompt "more-".

The user can enter the following commands at the "more-" prompt:

Space	print one more page
/str	search forward for str

n	repeat last search
<cr>	display one more line
q	quite more and terminate command

Environment

The command uses the `moresz` environment variable to determine how many lines to display at time. If `moresz` is set to zero, the screen scrolls continuously. The `^S` or `^Q` control sequence can be used to pause the output, and the `^C` control sequence may be used to terminate output before the command finishes by itself.

See Also

The [set](#) command and environment for environment variable settings.

sh

Description

The `sh` command is the built-in command shell.

Command Format

The `sh` command is a built-in function and can not be invoked from the command line.

<code>^H</code>	delete char before cursor
<code>^D</code>	delete char at cursor
<code>^K</code>	delete line from cursor
<code>^W</code>	delete line
<code>^F</code>	move cursor right
<code>^B</code>	move cursor left
<code>^A</code>	move cursor far left
<code>^E</code>	move cursor far right
<code>^P</code>	recall previous cmd
<code>^N</code>	recall next cmd
<code>!</code>	repeat last cmd
<code>!str</code>	recall and execute cmd str
<code>!num</code>	recall and execute cmd num
<code>+-*/()</code>	operators
<code>^addr</code>	contents of address
<code>@name</code>	contents of register

&name	value of symbol
Oxnum	hex number
Oonum	octal number
Otnum	decimal number

Functional Description

The sh command is not user invoked command but is implicitly run as the command shell when PTBMON enters interactive mode.

When the command shell is ready to execute a new command it outputs the default command prompt:

```
PTBMON
```

The user can change the string output as the prompt by setting the environment variable prompt to the desired string value.

```
PTBMON set prompt "My very own PTBMON prompt>; "  
My very own PTBMON prompt>
```

The metacharacter '!' when used in the prompt string will be replaced with the command history number assigned to the next command.

PTBMON has a built in command line editor which can be used to edit the command line while typing. It can also be used to edit a previously entered command and re-issue it in its edited version. See the [hi](#) command for information about command history.

When entering commands, environment variables can be called by typing their name preceded by a \$ sign. Variable names may be enclosed in {} to make them stand out. This is useful to get expansion of \${myvar}foo to work properly while \$myvarfoo will not expand. If the variable name contains other characters than A-Z, a-z, and 0-9 it must be enclosed in {}.

To use a \$ sign without expansion \$\$ should be typed.

The maximum length a command line is allowed to expanded to is limited to 200 characters.

Variable substitution can not be nested in the current version of PTBMON.

More than one command can be entered on one command line by separating them with a semicolon character. This is useful when setting a variable to execute a sequence of commands. For example:

```
PTBMON set mycmd "boot boothost:myprog; b testfunc; g start"  
PTBMON $mycmd
```

Note that the quote characters ' and " will cause the embedded string to be treated as a single entity. However one level of quotes will be stripped of each time the string is evaluated.

Typing ^C aborts the current command and returns to the PTBMON shell prompt. Note that this is also the case if PTBMON's I/O routines are used for input and output in a separately executed program.

Environment

The command uses several environment variables which control the behavior of the command shell.

See Also

The [more](#), [hi](#) and [set](#) commands. The environment and expression reference on environment usage and expression construction.

eval

Description

The eval command evaluates an expression and displays the result.

Command Format

```
eval expression
```

Functional Description

The eval command evaluates an expression and displays the result. It can be used to make complex computations without applying the value to a command. The expression can be any valid expression including symbols constants and other objects. The result is displayed in hexadecimal, octal and decimal.

For example:

```
PTBMON eval 2+6*4
0x1a 032 26
```

Environment

The command uses the inbase and inalpha environment variables.

See Also

The expression, reference on how to write expressions.

hi

Description

The hi command list the history buffer.

Command Format

```
hi [cnt]
```

cnt list the last cnt commands in the history buffer

Entering the command with no parameters lists the last up to 200 command lines.

Functional Description

The hi command shows the command history, together with the history number for each command, in reverse order. The last command entered is listed first; the first command entered is listed last. The history and command numbers are reset to zero each time the system is reset.

Entering the hi command with no arguments lists the last 200 commands. This option is useful for determining the history number for a particular command. The user can page through the output of the hi command, one screen at a time.

The optional cnt parameter selects a set number of lines to be output. The history list is intentionally in the reverse order to that used in a C shell, so that the latest entry is displayed first. If a command line is identical to the previous command, it is not added to the command history.

An example of the hi command is shown below.

```
PTBMON hi 3Display the three last commands.

14 hi 3
13 hi
12 l

PTBMON hiDisplay the entire history, using more

13 hito control the screen output.
12 l
11 to
10 t
9 l
8 g start main
7 hi
6 g
5 ls -a @pc
4 d start+200+0t13*4

more-(q)
```

Environment

The command uses the more function and the moresz environment variable.

See Also

The [sh](#) command which maintains the command history.

about

Description

The about command displays credits and information about PTBMON.

Command Format

The about command has no parameters.

Functional Description

The about command display credits and short history of PTBMON.

Environment

No environment variables affect this command.

See Also

The [h](#) command for displaying command information.

h

Description

The h command provide a built in help function.

Command Format

```
h [ * | cmd . . ]
```

* displays detailed help about all commands

cmd displays detailed help about listed commands

If the command is executed without any parameters, then a short lists of all the available commands is printed out.

Functional Description

The h command provides built in help. If used without arguments, all available commands are listed. Used with one or more command names as an option, it displays a detailed help on those commands.

The "*" option produces detailed help on all the commands, using the more command to paginate output to the screen.

Examples illustrating the use of the h command follow.

Note *The actual output is architecture and configuration dependent.*

```

PTBMON h
Shell
    h  on-line help
    sh command shell
    more paginator
    vers print version info
    date get/set date and time
Environment
    set  display/set variable
    unset unset variable(s)
Boot and Load
    load load memory from hostport
Debugger
g start execution (go)
l list (disassemble) memory
Memory
    m  modify memory
    copy copy memory to memory
    search search memory
    d  display memory
    fill fill memory
    mt memory test
Misc
dump dump memory to hostport
    flush flush caches
    reboot reboot system
    flash program flash memory
    call call function
    ping ping remote host
Pci
    pcicfg pci config space
PTBMON
PTBMON h stty
    stty [tty] [-va] [baud] [sane] [term] set terminal options

```

Environment

The command uses no environment variables.

See Also

The [about](#) command.

vers

Description

The vers command displays version information.

Command Format

```
vers
```

Functional Description

The vers command can be used to display information about the PTBMON version and depending on the target platform, information about various platform specific versions.

Environment

The command uses no environment variables.

See Also

The [about](#) command.

stty

Description

The stty command set and display terminal settings.

Command Format

```
stty [device][-a][baud][sane][flags]
```

<i>device</i>	name of the device to change
-a	list all tty parameter settings
< <i>baud</i> >	set port to baud rate
sane	reset settings to default, leaving baudrate as is
< <i>mode</i> >	set mode
-< <i>mode</i> >	clear mode

When invoking the stty command with no parameters, the terminal type and baud rate for the tty0 port is displayed.

Functional Description

The stty command displays and sets the terminal options, such as terminal emulation type, baud rate, and ioctl settings.

```
PTBMON stty
baud=9600
PTBMON stty -a
istrip ixon -ixany -ixoff icanon echo echoe icrnl onlcr
erase=^H stop= start=^Q eol=^J eol2=^C vintr=^C
```

Although the stty command allows all listed flag settings to be changed, the expected effect on the line discipline on the console port may not be achieved. This is due to the built in line edit capability. Setting or clearing these parameters on other tty ports will have the desired effect though.

Environment

The command uses no environment variables.

date

Description

The date command displays or sets the date and time.

Command Format

date [yyyymmddHHMM.SS]

yyyymmddHHMM.SS is the new date and time

Functional Description

Invoking the date command with no arguments displays the current date and time as stored in the board's battery-backed clock/calendar device. If an argument is given, the argument is used to set the current date and time.

The optional argument is a string of digits, with the following meanings:

yyyy	year (2000 through 2199)
mm	month (January = 01)
dd	day of the month
HH	hour (24 hour clock)
MM	minute
.SS	seconds

When setting the date and time, you only need to enter as much as needs changing, starting with the minutes, then hours, then day, etc. Any value which is omitted is unchanged, except for seconds, which will be set to zero if omitted. Note that single digit values has to be typed with a preceding '0'.

An example of the date command is shown below.

```
PTBMON date                               Display current time
Wed Feb 23 13:29:33 2001

PTBMON date 32                             Change minutes
Wed Feb 23 13:32:00 2001

PTBMON date 1405                           Change hours and minutes
Wed Feb 23 14:05:00 2001

PTBMON date 200102241103                   New date and time
Thu Feb 24 11:03:00 1994
```

Environment

The command uses no environment variables.

Network Commands

These three commands are related to setting up the management port of the board. Two commands allow for the volatile setting of the ports MAC address and TCP/IP address. The [ping](#) command allows for quick testing of network connectivity. Refer to the [setprom](#) and [setboot](#) commands for nonvolatile setting of the MAC and TCP/IP addresses.

ifaddr

Description

The ifaddr command configures a network interface.

Command Format

```
ifaddr ifname ipaddr[:ifparameters]  
ifaddr ifname bootp
```

ifname name of the interface

ifparameters [<ipaddr>][:<netmask>][:<broadcast>][:<gateway>]]]

Functional Description

The ifaddr command is used to configure IP-addresses on interfaces accessible within PTBMON. The interface can be configured using parameters such as ipaddress, netmask, gateway etc. directly or via the bootp protocol.

The minimum required parameters is the interface name and an IP address such as:

```
PTBMON ifaddr tsec1:192.168.16.10
```

By adding ':' separated additional parameters netmask, broadcast and default gateway can be specified. The format of the config string is:

```
interface:ipaddress:netmask:broadcast:gateway
```

The following is an example:

```
PTBMON ifaddr tsec1:192.168.16.10:255.255.255.0:::192.168.16.1
```

Note that when a parameter is omitted, like the broadcast address in the example, the field should be left empty by just typing the ':' separator. Unless a specific need for a non standard netmask or broadcast address is required these fields are typically left empty. Netmask and broadcast is calculated from the IP address.

Environment

The command uses no environment variables.

ping

Description

The ping command test network connectivity.

Command Format

```
ping [-fv][-c n][-s sz][-i sec][-l n] host
```

-f	flood ping. Send packages as fast as possible
-v	verbose
-c n	send n packages and stop
-s sz	size of packet expressed in bytes
-i sec	wait sec seconds before sending next packet
-l cnt	presend cnt packages before falling back to interval sending
host	target to where packets are sent

Functional Description

The ping command sends ICMP_ECHO_REQUEST packages to the given remote host and monitors any ICMP_ECHO_RESPONSE packages received back. This command is useful to test connectivity with a remote host.

The -f flag can be used to flood ping the remote host. Packages are then sent as fast as possible. Note that using this option may lead to network congestion and should be used carefully on networks shared with others.

When using ping for fault isolation, start by pinging 127.0.0.1 (a universal self-address, by internet convention.) This verifies that at least the onboard setup is workable. Then, hosts and gateways further and further away should be pinged. Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round trip time of these packets is used in calculating the minimum/average/maximum round-trip time numbers. When the program is terminated by a ^C a brief summary is displayed.

Ping will report duplicate and damaged packets. Duplicate packets should never happen: they have to be gateway problems. Tell your network manager.

Damaged packets (data doesn't look like it should) are serious cause for alarm and often indicate broken hardware; somewhere in the ping packets path (in the network or in the hosts).

Environment

The command uses no environment variables.

Memory Commands

This set of commands allow manipulation of memory. Individual memory locations or blocks of memory may be viewed, changed, copied or compared. A few commands allow for simple memory testing while another allows for searching for a pattern. Lastly, the level of SDRAM Error Checking and Correction may be modified using these commands. Refer to the [flash](#) command for details of erasing and programming the application FLASH area and the [bFlash](#) command for programming of the boot FLASH area.

m

Description

The m command modifies memory contents.

Command Format

```
m [-bhwnx] adr [val|-s str]..
```

- b access bytes
- h access half-words
- w access words
- x swap bytes
- n non-interactive (no write)

Interactive Options

- <hexval> set memory, forward one
- CR forward one, no change
- = re-read
- ^|- back one
- . quit

Invoking the m command with only the addr argument enters the interactive command mode.

Functional Description

The m command is used to modify memory. Modification can be done in two ways, directly or interactive. In direct mode the data is given on the command line and the command terminates when all the data items given has been stored in memory. When no data is given on the command line the interactive mode is entered and the user can interactively display and modify memory locations.

If no size option is given to the command the environment variable datasize will be used to determine the size of data to work with.

When using a string as the data argument, -b will be the default datasize regardless of what the datasize environment variable is set to. Data is then taken from the given string argument and stored in consecutive locations. When -s option is used only the direct mode of the command is usable.

Environment

The command uses the datasize environment variable.

See Also

The dump ([d](#)) command.

compare

Description

The compare command compares memory to memory.

Command Format

```
compare from to size
```

from start address of first block of memory

to start address of second block of memory

size number of bytes to compare

Functional Description

The compare command performs a byte by byte comparison of the two blocks of memory.

Environment

No environment variables affect this command.

fill

Description

The fill command fill a memory area with a pattern.

Command Format

```
fill from to {val|-s str}..
```

-s next arg is string

fill pattern can be combined from several parameters like '0x13 -s Hi 0x13'

Functional Description

The fill command is used to fill an area of memory with the given data pattern. The pattern is expressed as bytes and can be of any size not exceeding the maximum size of 80 bytes.

Examples of valid pattern arguments are:

0x13 One character pattern.

0x13 0x10 0x0 A three character pattern.

0x13 -s "Hello world" 0x0 A pattern with a zero terminated string.

The fill operation reuses the pattern until the last location has been filled.

Environment

The command uses no environment variables.

See Also

The [search](#) and [copy](#) commands.

mt

Description

The mt command performs a simple memory test.

Command Format

```
mt [-cv] [start [size]]
```

-c continue loop until stopped

-v verbose. Show progress

start start address of area to test

size size of area to test expressed in bytes

Functional Description

The `mt` command is used to do a simple memory test. The test of the memory is simply done by writing each bit in every word and by writing each word locations address with its address and then check that the address can be read back from all locations. To have a slight chance to detect address errors all addresses are first written and then read back. It is not intended to do any rigorous memory testing but instead be a test that can be used to check that installed memory modules comes up as PTBMON probed them and detect possible problems that has to be further investigated.

Environment

The command uses no environment variables.

ecc

Description

The `ecc` command displays and sets the state of the error checking and correction (ECC) of SDRAM.

Command Format

```
ecc [-dgec]
```

- d disable ECC
- g generates ECC only, with no checking or correction
- e performs full ECC checking and correction
- c clears the ECC error registers

Functional Description

The arguments of the `ecc` command allow changing the setting of the ECC. Using `-d` will set disable ECC totally. A `-g` will set ECC to generate the checkbits, but neither checking nor correction occurs. Using `-e` performs full ECC generation, checking and correction. After the argument is processed, the command will display the new ECC setting with a report of any ECC errors which have occurred. Executing `ecc` without arguments displays the current ECC setting with a report of any ECC errors which have occurred.

Environment

The command uses no environment variables.

d

Description

The d command displays memory contents.

Command Format

```
d [-bhwsrx] adr [cnt]
```

-b	display as bytes
-h	display as half-words
-w	display as words
-s	display a null terminated string
-x	swap bytes
-r<reg>	display as register

Functional Description

The d command is used to examine memory. The command prints the memory contents starting at the address given by the `adr` argument. The output is formatted according to the options given to the command and to the setting of certain environment variables. The `-b`, `-h`, `-w` and on 64 bit capable targets `-d` switch, determines whether the data should be displayed as bytes, halfword, word or double word entities. The `-s` switch can be used to display the memory data as a NULL terminated string. The string is limited to maximum 70 characters to avoid overruns.

If no size switch is given, the d command determines the size to display from the environment variable `datasize`.

On targets that support field printing of register values the `-r reg` switch may be used to print the memory value as if it was a register contents value of the type of the register.

The output is sent through 'more' and quitting more terminates the command. The `cnt` argument can be used to display `cnt` number of lines and then stop. When the `cnt` argument is given the output is not sent through 'more' but all lines are output until all lines have been printed.

The output is divided in three fields. First the address is printed and then the memory contents in hexadecimal. The last field shows the memory contents as printable characters. If the memory contents can not be represented by a printable character it is output as a dot (.).

The following example displays memory starting at 0xa0010000.

```
PTBMON d -w a0010000
a0010000 bfc0 2b00 bfc0 2b00 bfc0 2b00 bfc0 2b60 ..+...+...+...+`
a0010010 bfc0 2b20 bfc0 2ba8 bfc0 2b78 bfc0 2b60 ..+...+...+x...+`
a0010020 bfc0 2b48 bfc0 2ba8 bfc0 2ba8 bfc0 2ba8 ..+H..+...+...+.
a0010030 bfc0 2b78 bfc0 2b60 bfc0 2b48 bfc0 2e78 ..+x...+`..+H...x
a0010040 bfc0 2f08 bfc0 2ec4 bfc0 2e80 bfc0 2f90 ../......./.
a0010050 bfc0 2f90 bfc0 2f90 bfc0 2e78 bfc0 2e78 ../.../...x...x
a0010060 bfc0 2e78 0000 0000 0000 0000 0000 0000 ...x.....
```

Environment

The command uses the `datasize` environment variable.

See Also

The [more](#) and [modify \(m\)](#) commands.

copy

Description

The `copy` command copies one memory area to another address

Command Format

copy from to size

from	address where to copy from
to	address where to copy to
size	number of bytes to copy

Functional Description

The `copy` command is used to copy an area of memory to another memory address. If the source and destination areas overlap `copy` will copy in the direction that will result in a non destructive copy. When the copy operation is finished, any caches will be synchronized so copied code will be cached in correctly by the processor caches.

Environment

The command uses no environment variables.

See Also

The [fill](#) command.

search

Description

The search command search the memory for patterns.

Command Format

search from to pattern

from address where search should start

to address where search should end

pattern pattern specification. See below on how to specify patterns.

Functional Description

The search command is used to search an area of memory for the given data pattern. The pattern is expressed as bytes and can be of any size not exceeding the maximum size of 80 bytes.

Examples of valid pattern arguments are:

0x13 One character pattern.

0x13 0x10 0x0 A three character pattern.

0x13 -s "Hello world" 0x0 A zero terminated string.

The search operation is terminated when the to address is reached. Each time the pattern is found, the memory locations containing the pattern is displayed.

The following example searches the first megabyte of memory for the NULL terminated string PTBMON and display all occurrences.

```
PTBMON search 0 100000 -s PTBMON 0x0
0000fcfc 504d4f4e 00000000 00000001 000871c8 PTBMON.....q
0006739a 504d4f4e 00000000 00000a50 4d4f4e2f PTBMON.....PTBMON
0007f140 504d4f4e 00307830 00000000 00000000 PTBMON.0x0.....
00087ee3 504d4f4e 0000087e e8000000 03682073 PTBMON...~.....h
PTBMON
```

Environment

The command uses no environment variables.

See Also

The [copy](#) and [fill](#) commands.

viewmem

Description

The viewmem command performs a memory test of a location in memory.

Note *This command is not supported on all products.*

Command Format

```
viewmem addr val -[bhw]
addr          Memory location to test
val           Value to write to memory before test starts
-b            access bytes
-h            access half-words
-w            access words
```

Functional Description

The viewmem command repeatedly tests the same memory address for errors in reading from the memory location. After writing val to the address, it writes the bit-wise inverse of val to a nearby memory location. It then alternates reading from the two locations, each time checking that what is read from the address under test is equal to val. If not, the value read and the expected value are printed out with a time stamp.

Environment

No environment values affect this command.

See Also

See the [mt](#) command for an additional memory test.

PCI Commands

There are two commands provided for rudimentary support of the local PCI bus. The commands allow scanning the PCI bus looking for active devices, and reading and modifying the configuration registers of any devices found. Most PCI devices are left in reset by the boot program. Refer to the Hardware Manual for details of how to remove them from reset by writing to a PLD.

pcicfg

Description

The pcicfg command access PCI configuration space.

Note *This command is not supported on all products.*

Command Format

```
pcicfg bus device [:subfunc] register
bus          PCI bus to access
device       PCI device to access
subfunc      PCI subfunction to access
register     PCI register to access
```

Functional Description

The pcicfg command is used to access the PCI configuration space in the given bus. The command takes four arguments, the bus number, the device number, the subfunction number and the first register to access. The register number must be 32 bit aligned and all accesses are made as 32 bit read and writes. When executing the command an interactive mode is entered. The register number along with the current contents of the configuration register is displayed. The user can then enter one of the following commands:

expression	value to be stored
=	reread the current location
<cr>	forward to next address
-	back to previous address
^	back to previous address
.	exit interactive mode

If an expression is given, the register displayed will be written with the new value and the next register will be displayed.

Values are always displayed and entered in little endian or the native PCI endian. This means that the bytes do not have to be swapped if the target is a big endian target.

Environment

The command uses no environment variables.

See Also

The [pciscan](#) command.

pciscan

Description

The pciscan command scan for devices on PCI bus.

Note *This command is not supported on all products.*

Command Format

```
pciscan [-b <bus>][ -d <dev>]  
-b <bus>      bus no  
-d <dev>      dev no
```

Functional Description

The pciscan command scans and displays information about devices found on the PCI bus.

Environment

The command uses no environment variables.

See Also

The [pcicfg](#) command.

Environment Commands

These commands allow the creation, viewing, editing and deletion of environment variables. None of the changes are saved in nonvolatile memory and they apply only during the current session. Nevertheless, it is convenient to modify the settings of some of the variables such as the `moresz` variable. It controls the number of lines displayed by `more`. It is convenient to view other variables such as the TCP/IP addresses to verify that the values are being correctly interpreted by the program.

env

Description

The `env` command displays the setting of environment variables.

Command Format

```
env [name]
```

name name of environment variable

Functional Description

The `env` command is used to examine the environment variables stored in memory along with their values. Issuing `env` with an argument will display the setting of just that one variable. Issuing `env` without an argument will display all environment variables.

The output is sent through 'more' and quitting more terminates the command.

Environment

No environment variables affect this command.

See Also

The [set](#), [unset](#) and [eset](#) commands for setting and editing variables.

unset

Description

The unset command removes or resets a variable.

Command Format

```
unset name
```

name name of variable to remove or reset

Functional Description

The unset command unsets an environment variable. If the variable is one of the system variables and has a default value, the variable will be set to the default value.

Environment

The command changes the environment variables.

See Also

The [set](#) and [eset](#) commands.

set

Description

The set command set and display environment variables.

Command Format

```
set [name [value]]
```

name environment variable name

value string value to set variable

Functional Description

The set command is used to set or display environment variable values.

Some variables can only be set to predefined values. When such a variable is displayed PTBMON also displays the variables list of allowed values enclosed in square brackets. No list is shown if the variable can have any value. In general, when the value is a numeric value, or when the value has an unlimited range of possible values, no list is shown.

The set command does not evaluate the specified value but check constrained variables against a list of allowed values. Value checking on other variables are only performed when a command uses a variable.

To set a variable to a multiple-word value, enclose the value in single or double quotation marks.

The maximum length for a variable name and its value must be less than 255.

Examples illustrating the use of the set command follow.

```
PTBMON set                                Display all current values.
brkcmd = "l @cpc l"
datasz = -b                               [-b -h -w]
inalpha = hex                             [hex symbol]
inbase = 16                               [auto 8 10 16]
moresz = 10
regstyle = sw                             [hw sw]
rptcmd = trace                            [off on trace]
trabort = ^K
uleof = %
ulcr = off                                 [off on]
validpc = "_ftext etext"
heaptop = 80020000
dlecho = off                              [off on lfeed]
dlproto = EtxAck                          [none XonXoff EtxAck]
hostport = ttyl
prompt = "PTBMON "
etheraddr = 12:34:56:78:90:ab
ipaddr = 71.0.0.211
vxWorks =
diag = 0                                  [N[:dev]]
PTBMON set moresz                          Display current moresz.
moresz = 10
PTBMON set moresz 20                       Set moresz to 20 decimal.
```

Environment

The command changes environment variables.

See Also

The [unset](#) and [eset](#) commands.

eset

Description

The `eset` command edits and sets an environment variable.

Command Format

```
eset name
```

`name` environment variable name is edited

Functional Description

The `eset` command is used to edit a previously set environment variable value.

The value of the given variable is displayed on the command line and the Shell line edit keys can be used to edit the value.

The `eset` command does not evaluate the specified value but check constrained variables against a list of allowed values. Value checking on other variables are only performed when a command uses a variable.

The maximum total length for a variable name and its value must be less than 255 characters.

Environment

The command changes the environment variables.

See Also

The [unset](#) and [set](#) commands.

PTI Commands

This class of commands may be broken down into two basic categories. The first displays PLD registers and networking related registers on the PowerPC processor. This category includes the [dumpPld](#), [dumpPhy](#), [dumpMal](#), [dumpEmac](#), or [dumpTsec](#) commands. Refer to the board's *Hardware Manual* for more information on the PLD registers and to the *PPC440GX Embedded Processor User's Manual* or the *MPC8641D Integrated Host Processor Family Reference Manual* for information related to the EMAC and MAL registers.

The other category facilitates the initialization, modification and viewing of the bootdata and promdata contained in the FLASH. The [initboot](#) and [initprom](#) commands will restore all variables of their respective areas to their factory defaults. The [setboot](#) and [setprom](#) commands allow changing the variables while the [getboot](#) and [getprom](#) commands allow viewing of the data.

The last two commands of this class are the [bFlash](#) and [berase](#) commands. The bFlash allows the programming of a new version of boot code into the boot FLASH. The command copies the contents of the first 0x180000 bytes of SDRAM. DO NOT execute this command without first having loaded a valid boot code executable into this region of SDRAM! The berase command allows erasing select sectors of the boot FLASH.

bFlash

Description

The bFlash command programs the boot flash.

Command Format

`bFlash`

The bFlash command has no parameters.

Functional Description

The bFlash command reprograms the boot flash with the contents of RAM from the memory range of 0x00000000 to 0x0017FFFF.

Environment

No environment variables affect this command.

See Also

The [load](#) command for loading RAM with an executable image and the [bUpdate](#) command.

setprom

Description

The setprom command allows modifying the promdata variables.

Command Format

```
setprom [-t tester] [-s serialnumber] [-m n xxx xxx ...]
```

- t saves the tester (up to 11 characters) in promdata FLASH
- s saves the serial number (11 characters maximum) in promdata FLASH
- m allows setting the nth mac address

Functional Description

The setprom command is used to set the fields of the promdata, saving the changes in FLASH. Any changes will become effective the next time the board is booted.

The promdata includes the factory tester's name, the serial number of the board and up to 64 MAC addresses. The `-t` and `-s` arguments allow setting of the first two while the `-m` is used to set the MAC addresses.

Multiple MAC addresses may be set using one command. The following command will set the 3rd through 5th MAC addresses:

```
setprom -m 2 00:C0:8C:D8:4C:38 00:C0:8C:D8:4C:B8 00:C0:8C:D8:4C:78
```

Note that each MAC address is separated from the others by a space. Also, the 64 MAC addresses are numbered from 0 to 63.

Either up or lower case letters may be used in the MAC address, although the command must be in lower case.

The MAC address used by the board's management port depends upon which eTsec is used as the management port. For example, eTsec1 uses the 0th MAC address while eTsec2 uses the second. Assignment of the other MAC addresses is done by the application software.

Environment

No environment variables affect this command.

See Also

See the [getprom](#) command to display the promdata.

See the [initprom](#) command to reset the promdata to factory default settings.

initboot

Description

The initboot command initializes the bootdata information to its factory default values.

Command Format

```
initboot
```

The initboot command has no parameters.

Functional Description

Running the initboot command will initialize all the items of the bootdata information to their factory default values and stores the changes in FLASH. The new values will become operational the next time the board is rebooted.

Environment

The initboot command uses no environment variables.

See Also

See the [getboot](#) to display the bootdata variables.

See the [setboot](#) command to modify them.

getboot

Description

The getboot command displays the contents of the bootdata information stored in FLASH.

Command Format

```
getboot
```

The getboot command has no parameters.

Functional Description

Running the getboot command produces a three-column listing of the contents of the bootdata information. As shown below, the first column is an enumerated value used by the [setboot](#) command to change the value of an item. The second column is the item's name while the third column is its value.

```
PTBMON getboot
1   board_id:                0x00000007
2   disable_poc:             1
3   boot_on_poc_failure:     1
4   which_application         0
5   abort_boot_timer         3
6   reboot_on_watchdog       0
7   manufacturing_test_mode_flag 0
8   firmware_part_number     810P0733.10
9   console_baud_rate        9600
... ..
```

Environment

The getboot command uses no environment variables.

See Also

See the [initboot](#) command to initialize all items to their factory default value.

See the [setboot](#) command to change the value of an individual item.

dumpEmac

Description

The dumpEmac command display PowerPC 440GX EMAC registers.

Note This command is not supported on all products.

Command Format

```
dumpEmac
```

The dumpEmac command has no parameters.

Functional Description

The dumpEmac command displays the contents of all registers of the PowerPC 440GX processor's Ethernet Media Access Controllers.

Environment

The command uses no environment variables.

initprom

Description

The initprom command initializes the promdata to its default settings.

Command Format

```
initprom
```

The initprom command has no parameters.

Functional Description

Running the initprom command will initialize all the items of the promdata to their factory default settings and store the changes in FLASH. The changes will take effect the next time the board is booted. Using this command will erase all MAC addresses assigned to the board.

Environment

The initprom command uses no environment variables.

See Also

See the [getprom](#) command to display the promdata variables.

See the [setboot](#) command to modify them.

getprom

Description

The getprom command displays the contents of the promdata information stored in FLASH.

Command Format

```
getprom
```

The getprom command has no parameters.

Functional Description

Running the getprom command produces a listing of the contents of the promdata information.

Environment

The getprom command uses no environment variables.

See Also

See the [initprom](#) command to initialize all items of the promdata to their factory default settings.

See the [setprom](#) command to set or change the value of the individual items.

setboot

Description

The setboot allows modifying the bootdata variables.

Command Format

```
setboot item value
```

item enumerated value for the item being changed as displayed by the [getboot](#) command

value the new value for the item

Functional Description

The setboot command is used to change the value of an individual bootdata item. After validation, the new value of the item is stored in FLASH. The change will become effective the next time the board is rebooted.

The format of the value argument varies depending upon which item is being modified. Sometimes, the format will be a decimal number, a hexadecimal number, an IP address or a string. Which format to use is indicated by the `getboot` command. If the value is displayed as a hexadecimal number, then the value argument is entered as a hexadecimal number. The same for the decimal, IP address or string formats.

For example, the following line would set the board ID to 0x0A:

```
setboot 1 A
```

Hexadecimal numbers need not be preceded by a “0x” and may be either upper or lower case.

Environment

No environment variables affect this command.

See Also

See the [getboot](#) command to display the bootdata values.

See the [initboot](#) command to restore the factory default settings.

dumpPld

Description

The `dumpPld` command displays the contents of the PLD registers.

Command Format

```
dumpPld
```

The `dumpPld` command has no parameters.

Functional Description

The `dumpPld` command is used to display the contents of the PLDs.

Environment

No environment variables affect this command.

dumpMal

Description

The dumpMal command display PowerPC 440GX MAL register contents.

Note This command is not supported on all products.

Command Format

```
dumpMal
```

The dumpMal command has no parameters.

Functional Description

The dumpMal command displays the contents of all registers of the PowerPC 440GX processor's Memory Access Layer hardware core.

Environment

The command uses no environment variables.

notice

The notice command has no parameters.

Functional Description

The notice command displays the list of copyright holders to this software. The list is not short due to the software being developed under the BSD copyright.

Environment

The notice command uses no environment variables.

ipmc

Description

The ipmc command allows viewing the current revision of the Intelligent Peripheral Management Controller (IPMC) software and downloading a new program to the controller.

Note *This command is only supported on product configured with IPMC/PM controllers.*

Command Format

```
ipmc [[-p] [-b] path]
```

-p program IPMC FLASH, excepting the boot loader

-b program full IPMC FLASH

path path and filename

Issuing the ipmc command without any parameters will display the current version of the IPMC software.

Environment

No environment variables effect this command.

dumpPhy

Description

The dumpPhy command displays the phy's registers.

Command Format

```
dumpPhy
```

The dumpPhy command has no parameters.

Functional Description

The dumpPhy command displays the contents of the phy attached to the processor.

Environment

The command uses no environment variables.

dumpTsec

Description

The dumpTsec command displays the MPC8641D enhanced three speed ethernet controller registers.

Note *This command is not supported on all products.*

Command Format

```
dumpTsec
```

The dumpTsec command has no parameters.

Functional Description

The dumpTsec command displays the contents of all registers of the MPC8641D processor's Ethernet Media Access Controllers.

Environment

The command uses no environment variables.

dumpiic

Description

The dumpiic command display IIC registers.

Command Format

```
dumpiic bus
```

bus which iic bus (1 or 2).

Functional Description

The dumpiic command displays the contents of all registers of the processor's IIC Controller registers.

Environment

The command uses no environment variables.

initiic

Description

The initiic command initializes the selected iic bus.

Command Format

```
initiic bus freq
```

bus which iic bus (1 or 2).

freq the bus frequency (100 or 400)

Functional Description

The initiic command initializes the selected IIC Controller.

Environment

The command uses no environment variables.

readiic

Description

The readiic command reads from the selected IIC bus device.

Command Format

```
readiic bus addr reg cnt
```

bus which iic bus (1 or 2).

addr which IIC addr (bbbbbbbx)

reg register to begin reading data from (use 4 char for 2 byte addr)

cnt number of bytes to read (1-1024)

Functional Description

The readiic command reads and displays the contents of the selected registers of the target IIC device.

Environment

The command uses no environment variables.

writeiic

Description

The writeiic command writes to the selected IIC bus device.

Command Format

```
writeiic bus addr reg cnt d1 [d2[ d3[ d4]]]
```

bus which iic bus (1 or 2).

addr which IIC addr (bbbbbbbx)

reg register to begin writing data to (use 4 char for 2 byte addr)

cnt number of bytes to write (<=16)

dx data byte[s]

Functional Description

The writeiic command writes data to the selected registers of the target IIC device.

Environment

The command uses no environment variables.

bUpdate

Description

The bUpdate command updates the Boot PROM from the selected path

Command Format

```
bUpdate path
```

path path and filename

Functional Description

The bUpdate command downloads and programs a new image into the Boot PROM. It is used to update PTBMON.

Environment

The command uses no environment variables.

cksum

Description

The cksum command calculates the checksum of the Boot PROM

Command Format

```
cksum [-l laddr] [-u uaddr]
-l laddr      set lower address
-u uaddr      set upper address
```

Functional Description

The cksum is used to calculate the checksum of the Boot PROM. The lower and upper addresses default to 0xFFE80000 and 0xFFFFFFFF, respectively, if they are not included on the command line.

Environment

The command uses no environment variables.

rt

Description

The rt command runs diagnostic test(s).

Note *This command is not supported on all products.*

Command Format

```
rt * | test range [-t] [-c]
*          run all tests
test range run selected test(s)
-c         continuous test
-t         display test list
```

Functional Description

The rt command runs one or multiple diagnostic tests (if implemented).

Environment

The command uses no environment variables.

sd

Description

The sd command displays information about the miniSD device attached (where applicable)

Command Format

```
sd [-v]
```

-v verbose

An example of the display from this command is:

Memory size: 0x0f400000 (255852544) bytes

Logical blocks: 0x0007a000 (499712) blocks

Block size: 0x00000200 (512) bytes

Functional Description

The sd command displays information such as the size of the device, the number of logical blocks and the block size of the attached miniSD device. This command is only valid for implementations that contain miniSD support.

Environment

The command uses no environment variables.

sdRead

Description

The sdRead command reads from miniSD device into RAM.

Command Format

```
sdRead blk ramAddr cnt
```

blk source block address

ramAddr destination RAM address

cnt number of 512-byte blocks to xfer

Functional Description

The sdRead command reads “cnt” blocks starting from “blk” into RAM addressed by “ramAddr.” This command is only valid for implementations that contain miniSD support.

Environment

The command uses no environment variables.

sdWrite

Description

The sdWrite command writes to miniSD device from RAM.

Command Format

```
sdWrite ramAddr blk cnt
blk          destination block address
ramAddr     source RAM address
cnt         number of 512-byte blocks to xfer
```

Functional Description

The sdWrite command writes "cnt" blocks of data to RAM addressed by "ramAddr" starting from block "blk." This command is only valid for implementations that contain miniSD support.

Environment

The command uses no environment variables.

The Bootdata Structure

The bootdata structure is located in Boot PROM. The sector location may vary depending on the hardware implementation. The bootdata structure contains variables used by PTBMON. The contents of this structure can be changed using the [setboot](#) command and displayed using the [getboot](#) command.

bootdata structure

```
typedef struct _BOOTDATA {
    unsigned long board_id;
    char disable_poc;
    char boot_on_poc_failure;
    unsigned char which_application;
    unsigned char abort_boot_timer;
    char reboot_on_watchdog;
    char manufacturing_test_mode_flag;
    char firmware_part_number[24];
    unsigned char console_baud_rate;
    char reserved[17];
    unsigned char self_test_results[8];
    unsigned long ip_addr;
    unsigned long netmask;
    unsigned long broadcast;
    unsigned long ip_gateway;
    char boot_string_array[4][256];
    unsigned short pci_cfg_did;
    unsigned short pci_cfg_vid;
    unsigned short pci_cfg_sdid;
    unsigned short pci_cfg_svid;
    unsigned long plx_csr;
    unsigned long plx_class;
    unsigned long plx_lat_cache;
    unsigned long plx_bar0;
    unsigned long plx_bar1;
    unsigned long plx_bar2;
    unsigned long plx_dwnstrm_bar_cntrl;
    unsigned char boot_addr_test;
    char boot_on_sys_failure;
    char software_break_detect;
    char mgmtport_focus;
    unsigned char auto_recovery_enable;
    unsigned short boot_watchdog_timeout;
    unsigned long long core1_vector;
    unsigned char pad[892];
    int checksum;
} BOOTDATA;
```

Below are the default contents (as displayed by the [getboot](#) command) and sector location of the bootdata structure for current hardware designs.

AMC131 (Address 0xffe20000, sector 1009)

```
1          board_id: 0x00000010
2          disable_poc: 1
3          boot_on_poc_failure: 1
4          which_application: 0
5          abort_boot_timer: 3
6          reboot_on_watchdog: 0
7 manufacturing_test_mode_flag: 0
8          firmware_part_number: 810Q0911.11
9          console_baud_rate: 9600
10         self_test_results: 0x0000000000000000
11         ip_addr: 0.0.0.0
12         netmask: 0.0.0.0
13         broadcast: 0.0.0.0
14         ip_gateway: 0.0.0.0
15         boot_string_array1:
16         boot_string_array2:
17         boot_string_array3:
18         boot_string_array4:
19         pci_cfg_did: 0x0131
20         pci_cfg_vid: 0x1214
21         pci_cfg_sdid: 0x0131
22         pci_cfg_svid: 0x1214
23         plx_csr: 0x0006
24         plx_class: 0x078000
25         plx_lat_cache: 0x00000000
26         plx_bar0: 0x00000000
27         plx_bar1: 0x00000000
28         plx_bar2: 0x00000000
29         plx_dwnstrm_bar_cntrl: 0xff000000
30         boot_addr_test: 0
31         boot_on_sys_failure: 1
32         software_break_detect: 1
33         mgmtport_focus: 0
34         auto_recovery_enable: 0
35         boot_watchdog_timeout: 120
36         core1_vector: 0x00000100
```

The Promdata Structure

The promdata structure is located in Boot PROM. The sector location may vary depending on the hardware implementation. The promdata structure contains variables used by PTBMON (such as MAC addresses, serial number and date of manufacture) and are initialized during the manufacturing process. THE DATA IN THE PROMDATA STRUCTURE SHOULD NOT BE MODIFIED. The contents of this structure can be displayed using the [getprom](#) command.

promdata structure

```
typedef struct _PROMDATA {
    char mac[64][6];
    char serial_number[12];
    struct tm test_date;
    char tester_login[12];
    char extra[1020];
    int checksum;
} PROMDATA;
```

Below are typical contents (as displayed by the [getprom](#)) and sector location of the promdata structure for current hardware designs.

AMC131 (Address 0xffe40000, sector 1010)

Serial Number: 501894

Tester: pvg

Test date: Wed Jul 18 10:23:00 2007

MAC addresses:

0 - 00:c0:8c:94:28:fe	1 - 00:c0:8c:94:28:01	2 - 00:c0:8c:94:28:81
3 - 00:c0:8c:94:28:41	4 - ff:ff:ff:ff:ff:ff	5 - ff:ff:ff:ff:ff:ff
6 - ff:ff:ff:ff:ff:ff	7 - ff:ff:ff:ff:ff:ff	8 - ff:ff:ff:ff:ff:ff
9 - ff:ff:ff:ff:ff:ff	10 - ff:ff:ff:ff:ff:ff	11 - ff:ff:ff:ff:ff:ff
12 - ff:ff:ff:ff:ff:ff	13 - ff:ff:ff:ff:ff:ff	14 - ff:ff:ff:ff:ff:ff
15 - ff:ff:ff:ff:ff:ff	16 - ff:ff:ff:ff:ff:ff	17 - ff:ff:ff:ff:ff:ff
18 - ff:ff:ff:ff:ff:ff	19 - ff:ff:ff:ff:ff:ff	20 - ff:ff:ff:ff:ff:ff
21 - ff:ff:ff:ff:ff:ff	22 - ff:ff:ff:ff:ff:ff	23 - ff:ff:ff:ff:ff:ff
24 - ff:ff:ff:ff:ff:ff	25 - ff:ff:ff:ff:ff:ff	26 - ff:ff:ff:ff:ff:ff
27 - ff:ff:ff:ff:ff:ff	28 - ff:ff:ff:ff:ff:ff	29 - ff:ff:ff:ff:ff:ff
30 - ff:ff:ff:ff:ff:ff	31 - ff:ff:ff:ff:ff:ff	32 - ff:ff:ff:ff:ff:ff
33 - ff:ff:ff:ff:ff:ff	34 - ff:ff:ff:ff:ff:ff	35 - ff:ff:ff:ff:ff:ff
36 - ff:ff:ff:ff:ff:ff	37 - ff:ff:ff:ff:ff:ff	38 - ff:ff:ff:ff:ff:ff
39 - ff:ff:ff:ff:ff:ff	40 - ff:ff:ff:ff:ff:ff	41 - ff:ff:ff:ff:ff:ff
42 - ff:ff:ff:ff:ff:ff	43 - ff:ff:ff:ff:ff:ff	44 - ff:ff:ff:ff:ff:ff
45 - ff:ff:ff:ff:ff:ff	46 - ff:ff:ff:ff:ff:ff	47 - ff:ff:ff:ff:ff:ff
48 - ff:ff:ff:ff:ff:ff	49 - ff:ff:ff:ff:ff:ff	50 - ff:ff:ff:ff:ff:ff
51 - ff:ff:ff:ff:ff:ff	52 - ff:ff:ff:ff:ff:ff	53 - ff:ff:ff:ff:ff:ff
54 - ff:ff:ff:ff:ff:ff	55 - ff:ff:ff:ff:ff:ff	56 - ff:ff:ff:ff:ff:ff
57 - ff:ff:ff:ff:ff:ff	58 - ff:ff:ff:ff:ff:ff	59 - ff:ff:ff:ff:ff:ff
60 - ff:ff:ff:ff:ff:ff	61 - ff:ff:ff:ff:ff:ff	62 - ff:ff:ff:ff:ff:ff
63 - ff:ff:ff:ff:ff:ff		

Installing and Booting an Image from FLASH

Below is an example procedure to install an image into the FLASH device and configure PTBMON to execute it.

To install and boot an image from FLASH:

1 Attach network cable to MGMT port.

2 Erase area of application FLASH:

```
PTBMON> flash -e ec000000 800000
```

3 If needed, configure network interface:

```
PTBMON> ifaddr tsecl <a>:<b>:<c>:<d>
```

a. ipaddr

b. netmask

c. broadcast

d. gateway

4 Load the image into RAM, avoiding the memory range of 0x00200000 thru 0x00280000:

```
PTBMON> load -r -o 400000 tftp://192.168.0.100/rel/810/810p0698.41/  
NexusWare-440.img
```

5 Program the application FLASH with the image:

```
PTBMON> flash ec000000 800000 400000
```

6 Enter command into *boot_string_array1*:

```
PTBMON> setboot 15 "g -e ec000000"
```

7 Set *which_application* to boot from **boot_string_array1**

```
PTBMON> setboot 4 1
```