CES PPCMon 5.0

AWX 3317D Monitor

User's Manual
DOC 3317D/UI
Preliminary Version (0.1)
September 2004

CREATIVE ELECTRONIC SYSTEMS S.A.

CES Warranty Information

The information in this document has been checked carefully and is thought to be entirely reliable. However, no responsibility is assumed in case of inaccuracies. Furthermore, CES reserves the right to change any of the products described herein to improve reliability, function or design. CES neither assumes any liability arising out of the application or use of any product or circuit described herein nor conveys any license under its patent rights or the rights of others.

WARNING - FCC COMPLIANCE

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

CES - Creative Electronic Systems S.A.
38 Avenue Eugène-Lance
P.O. Box 584
1212 Grand-Lancy 1
Switzerland

Tel: +41 (0)22 884 51 00
Fax: +41 (0)22 794 74 30
EMail: ces@ces.ch

The information in this document is subject to change without notice and should not be construed as a commitment by CES. Creative Electronic Systems assumes no responsibility for any error that may appear in this document.

September 30, 2004
A. Martin

Table of Contents

AWX 3317D - CES PPCMon

CHAPTER 1 PPCMON COMMANDS 1

Index
AWX 3317D - CES PPCMon

S

T

U

V

X

Chapter 1
PPCMon Commands

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
bma	**x**	**x**	**x**	**x**		

NAME

bma - executes VME bma related operation

SYNTAX

bma *operation [para0] [para1]....*

PARAMETERS

Operation	the following operations are accepted:
start	*des.s src_star t..src_end*
	reset
	status

DESCRIPTION

The **bma** command allows the user to requested operations performed by the BMA controller on the RIO2 8062. It is able to move blocks of data between VME and PCI. The following operations are accepted by the **bma** command.

The **start** operation allows the user to transfer a block of data with the following parameters:

des	destination address
s	destination space:
	v VME
	p PCI
src_start	start address of the source block
src_end	end address of the source block

The **status** operation displays the current value of the hardware registers controlling the BMA.

The **reset** operation clears all status flags and flushes the BMA controller.

After a reset, it is ready for a new data transfer. Static parameters such as VME address modifier, data size, block size, PCI block size and space, address increment and swapping policy can be set using the **set** command with **vme_bma** a as argument.

EXAMPLE

Transfers a 4-KByte block from system memory address 0x500000 to VME address 0x8100000 in A32 D32 block mode with automatic data swapping. The VME block size is set to 16 long words and the VME address is incremented at every transfer. The PCI block size is controlled by the latency timer (vme_bma_pbsz = 0).
Check the settings for the transfer.

```
PPC_Mon>set vme_bma VME block mover
vme_bma_swap [noswap autoswap wswap bswap]: autoswap ->
vme_bma_am [a24u a24s a32u a32s am10 am11 am12 am14]: a32u ->
vme_bma_dsz [d32 d64]: d32 ->
vme_bma_vbsz [0 1 2 4 8 16 32]: 16 ->
vme_bma_pbsz [0 4 8 16]: 0 ->
vme_bma_inc [y n]: y ->
vme_bma_space [pio pmem pmrm sysmem]: sysmem ->
PPC_Mon>
```

Fill the local buffer (at 0x500000) with 0xdeadface and the VME destination buffer (at 0x8100000) with 0x12345678. Check the content of the VME destination buffer.

```
PPC_Mon>fm.l 500000..501000 deadface
PPC_Mon>fv.l 8100000..8101000 12345678
PPC_Mon>dv.l 8100000
0x08100000 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100010 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100020 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100030 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100040 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100050 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100060 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x08100070 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
PPC_Mon>
```

Reset the BMA and start the data transfer.

```
PPC_Mon>bma reset
PPC_Mon>bma start 8100000.v 500000..501000
PPC_Mon>
```

Display the BMA status and verify the data buffer in the VME slave.

```
PPC_Mon>bma status
VAP register: 0x08101001
PAP register: 0x80501001
CAP register: 0x00000000
CTR register: 0x4042b000
PPC_Mon>dv.l 8100000
0x08100000 : deadface deadface deadface deadface ................
0x08100010 : deadface deadface deadface deadface ................
0x08100020 : deadface deadface deadface deadface ................
0x08100030 : deadface deadface deadface deadface ................
0x08100040 : deadface deadface deadface deadface ................
0x08100050 : deadface deadface deadface deadface ................
0x08100060 : deadface deadface deadface deadface ................
0x08100070 : deadface deadface deadface deadface ................
PPC_Mon>
```

SEE ALSO

set

Command		Onboard	33170A	33170B	VME	CPCI	MFCCMon
boot	**srec, disk**	**x**	**x**	**x**	**x**	**x**	
	net		**x**	**x**	**x**	**x**	
	fprom		**x**	**x**	**x**	**x**	**x**
	sdcard			**x**	**x**	**x**	**x**

NAME

boot - boots an executable file

SYNTAX

boot *[dev [file [addr [mode]]]]*

PARAMETERS

dev	boot device
file	file to boot
addr	memory address where to load the file
filetype	boot mode

DESCRIPTION

The **boot** command passes its parameters to the monitor load function (see **load** command) in order to load at address *addr* a data buffer *file* from a device *dev* and tries to start execution (see **exec** command) according to the *filetype* found in the boot environment parameters.

Missing command parameters are taken from the boot environment. The **show** / **set** commands can be used to check or set the boot parameters. They are stored in NVRAM and don't have to be reloaded after power-on unless you want to modify them. PPCMon can boot from ethernet, SCSI, serial line and FPROM.

The *filetype* parameter is very important for PPCMon to know how to execute the file.

If *filetype* is **binary**, after loading the file at address *addr* in system memory, caches are flushed and execution starts at *addr*. This mode shall be used for compatibility with PPCMon version 2.xx.

If *filetype* is **auto**, after loading the file in memory according to address parameters found in the header. Caches are flushed and execution starts at the code entry point. Currently COFF and ELF executables are handled.

EXAMPLE

Check the boot parameters and load a LynxOS kernel from IDE disk.

```
PPC_Mon>show boot

BOOT parameters

   boot_flags [afmnNS] : a
   boot_device [s<1-8>d<0-7><a-d>, i<1-8>d<0-3><a-f>, le, e<0-2>, atm<1-6>@<vci>
, tty<0-1>, fp] : i2d0b
   boot_filename : /lynx.os
   boot_rootfs [s<1-8>d<0-7><a-d>, i<1-8>d<0-3><a-f>, rd] : id0b
   boot_delay : 3 sec
   boot_address : 1000000
   boot_size : 0
   boot_fast [ y n ] : n
   boot_filetype [ binary auto lynx prep srec ] : lynx
   boot_cpunr : 0
   boot_mode [ ppcmon test auto user ] : ppcmon
   boot_line :
PPC_Mon>
PPC_Mon>boot

Trying device  i2d0b   [file : /lynx.os] ...
loading from disk(2,0,2) at address:1000000
disk capacity is 49577472 blocks
disk block size is 512 bytes
partition 2's size is 4096449 blocks
loading /lynx.os...
file length: 0x00210000
expanding file from address 1000000
LynxOS file [xcoff]
starting code execution at address: 4020

starting lynxOS
.....
```

Load a user application (mycode) in system memory from the network using TFTP protocol and execute it. The application is an xcoff file and *boot_filetype* **auto** is chosen.

```
PPC_Mon>set boot

BOOT parameters

   boot_flags [afmnNS] : a ->
   boot_device [s<1-8>d<0-7><a-d>, i<1-8>d<0-3><a-f>, le, e<0-2>, atm<1-6>@<vci>
, tty<0-1>, fp] : id0 -> le
   boot_filename : /lynx.os -> /tftpboot/mycode
   boot_rootfs [s<1-8>d<0-7><a-d>, i<1-8>d<0-3><a-f>, rd] : id0b ->
   boot_delay : 3 sec ->
   boot_address : 1000000 ->
   boot_size : 0 ->
   boot_fast [ y n ] : n ->
   boot_filetype [ binary auto lynx prep srec ] : lynx -> auto
   boot_cpunr : 0  ->
   boot_mode [ ppcmon test auto user ] : ppcmon ->
   boot_line :   ->

PPC_Mon>boot

Trying device  fp   [file : 100000] ...
loading from FPROM offset 0x100000
at address: 0x1000000 [type=80]
xcoff file
starting code execution at address: 4000
testing usercode
test ended
PPC_Mon>
```

The user standalone application (xcoff file) is loaded in FPROM at offset 0x100000 (see **fprom** command). The **boot** command is then used to load that application in system memory and launch it. After execution it returns to PPCMon.

```
PPC_Mon>fprom load 100000
loading fprom... 100000

Trying device  le   [file : /tftpboot/mycode] ...
ethernet load
local address = 00:80:a2:01:00:ca
using ARPA
.file server address 00:06:5b:b8:bd:eb
loading file '/tftpboot/mycode' from server 10.0.4.49 at address 0x1000000
tftp packet number:       d6
transfer rate: 1223 kbyte/sec [len = 0x1a979]
fprom_off = 100000   mem_addr = 1000000  len = 1a979
FPROM erasing sector 00100000
FPROM programming offset 00120000
PPC_Mon>

PPC_Mon>boot fp 100000

Trying device  fp   [file : 100000] ...
loading from FPROM offset 0x100000
at address: 0x1000000 [type=80]
xcoff file
starting code execution at address: 4000
testing usercode
test ended
PPC_Mon>
```

On a board with SDcard controller, lets boot our standalone application from an sdcard formatted with a FAT filesystem on a PC. The application *usrcode* has been copied in the */application* directory

```
PPC_Mon>boot sdcard /applications/usrcode 100000

Trying device  sdcard   [file : /applications/usrcode] ...
loading from SD Card -> initialize sd card...
file length: 0x0001a9b9
expanding file from address 100000
xcoff file
starting code execution at address: 4000
testing usercode
test ended
PPC_Mon>
```

SEE ALSO

sdcard, exec, fprom, load, sec, show

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
cache	x	x	x	x	x	x

NAME

cache - executes cache operation

SYNTAX

cache.*c operation [para]*

PARAMETERS

c indicates which cache to handle:

i	primary cache instruction
d	primary cache data
l2	secondary cache

operation the following operations are accepted:

enable	enables the cache
disable	disables the cache
flush	

start..end invalidates the cache:

lock	locks the cache
unlock	unlocks the cache
status	shows the cache status

DESCRIPTION

The **cache** command allows the user to **enable**, **disable**, **lock**, **unlock** and **flush** the PowerPC primary and secondary caches. The *c* parameter indicates on which cache the operation must be done. If this parameters is missing all caches are affected.

The **status** operation displays the current state of the cache.

The **cache enable** and **disable** commands are there only for test purpose. They apply on the specified cache as a whole (these operations are done by setting the corresponding bits in the HID0 register). These commands can not be used more than once because it could make the system memory and cache memory incoherent, leading to a crash of PPCMon.

If PPCMon is booted without cache (see **cache** environment parameter), it is allowed to enable them later with the cache enable command. Before executing a **cache disable** command, cache should be flushed to avoid inconsistencies. When caches are enabled, coherency is controlled by the MMU mechanisms.

The **cache flush** command applies only for cache lines specified by the address range parameter. If code is loaded using the **load** or **boot** commands, cache flushing is automatically done by PPCMon. When the flush command applies to the primary instruction cache, specified cache lines are made invalid, forcing a reload of the cache. For the primary data cache, specified addresses are made coherent with the contents of the cache. For L2 cache, a global invalidation is applied.

The **cache lock** and **unlock** commands are there only for test purpose. They apply on the specified cache as a whole (these operations are done by setting the corresponding bits in the HID0 register).

EXAMPLE

On a RIO3 8064 equipped with L2 cache, show cache status, set cache environment parameter to automatically enable cache after reboot, reset the board and show cache status again.

```
PPC_Mon>cache status

instuction cache   : disabled  unlocked
data cache         : disabled  unlocked
l2 cache [00008016]: disabled
```

```
PPC_Mon>set cache

CACHE parameters

   Instruction cache control
   cache_i_ena [ y n ] : n -> y

   Data cache control
   cache_d_ena [ y n ] : n -> y

   L2 cache control
   cache_l2_ena [ y n ] : n -> y
   cache_l2_parity [ y n ] : n -> n
   cache_l2_do [ y n ] : n -> n
```

```
PPC_Mon>reset
MINI_Mon:booting PPC_Mon.......

PPC Boot Rev 5.20 created Mon Sep 20 16:17:06 2004

  Module Type: 8064AE  rev: B  serial: 186
  Host CPU: PPC750   ver: 83.02   speed: 450 Mhz
  PCI Bridge: CES XPC
  Memory Size: 256 + 256 = 512 Mbytes
  L2 CACHE:  1 Mbyte SSRAM   180 MHz
  FPROM: Two banks of AMD29F032 installed [16 Mbytes]

Entering boot diagnostics

 0    Check System Memory (0x00004000 - 0x1ff00000)          0  OK
 1    Check Interrupt Handler                                0  OK
 2    Check PCI Bridge                                       0  OK
 3    Check Cache and MMU                                    0  OK
 4    Check MK48T08 RTC and NVRAM                            0  OK
 5    Check SIC6351 Interrupt Controller                     0  OK
 6    Check Serial Lines                                     0  SKIPPED
 7    Check  Micro Timers                                    0  OK
 8    Check FIFO's (0 - 7)                                   0  OK
 9    Check Digital Thermometers                             0  OK
10    Check PCI devices                                      0  OK
11    Check On Board Ethernet Controller  (PCI slot 0)       0  OK
12    Check PMC #1 Extension  (PCI slot 2)                   0  NO DEVICE
13    Check PMC #2 Extension  (PCI slot 3)                   0  OK
14    Check PMC #3 Extension  (PCI slot 4 - agent 2)         0  NO DEVICE
15    Check PMC #4 Extension  (PCI slot 5 - agent 2)         0  NO DEVICE
16    Check VME Interface                                    0  OK
17    Check XPCI Interface                                   0  OK

************************************************
*   PPC_Mon  RIO8064 monitor - version 5.2       *
*      CES SA Copyright 1995 - 2004              *
************************************************

PPC_Mon>
```

```
PPC_Mon>cache status

instuction cache   : enabled   unlocked
data cache         : enabled   unlocked
l2 cache [bb804016]: enabled

PPC_Mon>
```

On a MFCC 8441, check the cache status and set the corresponding environment parameters to enable them automatically after reboot. Then return to PPCMon.

```
MFCC_Mon>cache status

 instuction cache : disabled unlocked
 data cache : disabled unlocked
```

```
MFCC_Mon>set cache

CACHE parameters

   Instruction cache control
   cache_i_ena [ y n ] : n -> y

   Data cache control
   cache_d_ena [ y n ] : n -> y

   L2 cache control
   cache_l2_ena [ y n ] : n -> y
MFCC_Mon>q
PPC_Mon>
```

Now from PPCMon, using the **mfcc** command reset the MFCC 8441 and connect again to its console port.

```
PPC_Mon>mfcc.1 reset
PPC_Mon>mfcc.1 connect

connecting

  MFCC 8442BD  SN  Proto 01/03  Rev PCB 519.0
  CPU : PPC750   ver: 22.14   speed: 500 Mhz
  Memory size : 64 Mbytes
  FPROM :
  PCI bridge :
  I/O interface :
  Adaptator :

Entering boot diagnostics

 0    Check System Memory (0x00000000 - 0x03effffc)          0  SKIPPED
 1    Check Interrupt Handler                                0  OK
 2    Check PCI Bridge                                       0  OK
 3    Check Cache and MMU                                    0  OK
 4    Check Interrupt Controller                             0  OK
 5    Check TIC Timer and WatchDog                           0  OK
 6    Check FIFO's (0 - 4)                                   0  OK

************************************************
*   MFCC_Mon MFCC844b monitor - version 5.2      *
*             CES SA Copyright 2004              *
************************************************

MFCC_Mon>
```

Verify that caches are enabled.

```
MFCC_Mon>cache status

instuction cache   : enabled   unlocked
data cache         : enabled   unlocked
l2 cache [bb804000]: enabled

MFCC_Mon>
```

SEE ALSO

set, show

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
cm	x	x	x	x	x	x

NAME

cm - compares memory

SYNTAX

cm.*x start..end cmp*

PARAMETERS

x	data size
start	start address of the reference block
end	end address of the reference block
cmp	start address of the block to be compared

DESCRIPTION

The **cm** command reads the block of memory starting at address *start* and ending at address *end* and compares it bit per bit to the block of memory starting at address *cmp*. Data accesses are done according to the data size specified in *x* where *x* shall be **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access. Every time a data in the compared block is found to be different than the corresponding one in the source block, both addresses and data are displayed.

EXAMPLE

Move 64 KBytes of data from FPROM (FPROM offset 0 is mapped at CPU address 0xfff00000) to system memory at address 0x100000 using the **mm** command.

```
MFCC_Mon>mm.l fff00000..fff10000 100000
```

Compare the data copied in system memory with FPROM content.

```
MFCC_Mon>cm.l fff00000..fff10000 100000
```

Modify data at address 0x108000 using the **pm** command.

```
MFCC_Mon>pm.l 108000 deadface
```

Do the comparison again...

```
MFCC_Mon>cm.l fff00000..fff10000 100000
0xfff08000 : ffffffff != 0x00108000 : deadface
```

SEE ALSO

mm, pm

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
config	x	x	x	x	x	x

NAME

config - shows the board hardware configuration

SYNTAX

config

PARAMETERS

none

DESCRIPTION

The **config** command shows the board configuration.

Information is gathered from different sources. *Module type, memory size and L2 cache* comes from the board signature stored in EEPROM. The **setenv sign** or **hwconf** allows the setting of that signature. The *Host CPU* and *PCI bridge* information is directly taken from the CPU and bridge internal registers. The *FPROM* identifier is read from the FPROM ID register and the ethernet address from the AMD ethernet controller.

The VME and PCI information comes from the corresponding hardware registers. *Ta* is the temperature read on a thermometer located under the CPU and the current date read from the real-time clock.

EXAMPLE

Display RIO 8064 (serial number 186) configuration.

```
PPC_Mon>config

  Module Type: 8064AE  rev: B  serial: 186
  Host CPU: PPC750   ver: 83.02   speed: 450 Mhz
  PCI Bridge: CES XPC
  Memory Size: 256 + 256 = 512 Mbytes
  L2 CACHE:  1 Mbyte SSRAM   180 MHz
  FPROM: Two banks of AMD29F032 installed [16 Mbytes]
  Ethernet Address: 00:80:a2:01:00:ca
  PCI Dev#0  VendorID = 0x1022  DeviceID = 0x2000  Rev. = 0x44
  PCI PMC#2  VendorID = 0x1095  DeviceID = 0x0646  Rev. = 0x07

  Ta = 46 C       Mon Sep 20 16:39:38 2004
PPC_Mon>
```

Display RIOC 4070 (serial number 2) configuration.

```
PPC_Mon>config

  Module Type: 4070AA  rev: 0  serial: 2
  Host CPU: PPC750GX   ver: 1.01   speed: 500 Mhz
  PCI Bridge: CES XPC
  Memory Size: 256 Mbytes
  L2 CACHE:  1 Mbyte INTERNAL
  FPROM: One bank of AMD29LV256 installed [32 Mbytes]
  Ethernet Address: 00:80:a2:10:30:02
  PCI Dev#0  VendorID = 0x1022  DeviceID = 0x2000  Rev. = 0x36
  PCI PMC#1  VendorID = 0x1000  DeviceID = 0x000b  Rev. = 0x04
  PCI PMC#5  VendorID = 0x1022  DeviceID = 0x2000  Rev. = 0x36

  Ta = 74 C       Mon Sep 20 16:39:38 2004
PPC_Mon>
```

Display RIOC 4068 (serial number 100) configuration.

```
PPC_Mon>config

  Module Type: 4068BE  rev: AA  serial: 100
  Host CPU: PPC7455   ver: 3.03   speed: 600 Mhz
  PCI Bridge: CES XPC
  Memory Size: 128+0 = 128 Mbytes
  L2 CACHE:  256 kbyte INTERNAL
  L3 CACHE: 2Mb div:6 Sample Point: 3(L3):2(CPU)
  FPROM: Two banks of AMD29F032 installed [16 Mbytes]
  Ethernet Address: 00:80:a2:01:40:74
  PCI Dev#0  VendorID = 0x1022  DeviceID = 0x2000  Rev. = 0x44
  PCI PMC#1  VendorID = 0x10d9  DeviceID = 0x844b  Rev. = 0x80
  PCI PMC#2  VendorID = 0x1000  DeviceID = 0x000b  Rev. = 0x04
  PCI PMC#3  VendorID = 0x1022  DeviceID = 0x2000  Rev. = 0x44

  Ta = 46 C       Tue Sep 21 16:15:33 2004
PPC_Mon>
```

Display MFCC 8443 (serial number 198) configuration.

```
MFCC_Mon>config

  MFCC8443DC  SN  198  Rev H:A/B S:4.7
  CPU : PPC750H   ver: 33.11   speed: 666 Mhz
  Memory size : 64 Mbytes
  FPROM :
  PCI bridge :
  I/O interface :
  Adaptator :

MFCC_Mon>
```

SEE ALSO

date, setenv

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
date	x	x	x	x	x	

NAME

date - displays / sets the current date and time

SYNTAX

date *[yymmddhhmm[.ss]]*

PARAMETERS

yymmddhhmm[.ss] current date

DESCRIPTION

date without argument displays the date found in the board's real-time clock MK48T08 according to the UNIX asctime format.

To set the date in the real-time clock, the parameters should be an ASCII string formatted according to the UNIX date syntax (*yymmddhhmm[.ss]*). If the RTC is stopped, setting the date will restart it.

yy	number of years since 1900 [00-99]
mm	month of the year [01-12]
dd	day of the month [01-31]
hh	hour of the day [00-23]
mm	minute in the hour [00-59]
ss	second in the minute [00-59] (optional)

EXAMPLE

Sets date to September 21, 2004 and time to 14:55:00.

```
PPC_Mon>date
Tue Sep 21 15:31:48 2004
PPC_Mon>date 0409211455
PPC_Mon>date
Tue Sep 21 14:55:06 2004
PPC_Mon>
```

SEE ALSO

rtc

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
diag	x	x	x	x	x	x

NAME

diag - runs onboard diagnostics

SYNTAX

diag *[device]*

PARAMETERS

device name of the device to test

DESCRIPTION

During the boot sequence, the PowerPC processor executes a set of diagnostics in order to check the board's hardware resources. For each diagnostic, the **diag** command allows the execution of a more comprehensive version of the tests by hand. If no argument is given, it displays a list of all available diagnostics.

Each diagnostic can be executed at three different levels (level 0, 1 or 2). Level 0 checks the presence of the device and verifies that its registers can be accessed. Level 1 performs an integrity checking while level 2 tests the device functionality. Level 2 is automatically used when the diagnostics are run by the **diag** command, and a status message is printed at every step in the diagnostic procedure.

The results of the diagnostics are stored in NVRAM and can be displayed by the status command.

EXAMPLE

Show all possible diagnostics for a RIO 8064

```
PPC_Mon>diag

syntax: diag <device>

          where device = sysmem (system memory)
                         bridge (PCI bridge)
                         mmu (PPC603 MMU and CACHES)
                         nvram (mk48t08 NVRAM and Real Time Clock)
                         intctl (sic6351 interrupt controller)
                         serial (pc87312 super IO controller)
                         timers (zcio8536 micro timers)
                         fifos (message passing FIFO's)
                         thermo (ds1620 Digital Thermometer)
                         pci (scan PCI tree for devices)
                         ethernet (am79c970 Ethernet Controller)
                         vme (VME master port (need VMDIS8004))
                         xpci (scan XPCI to initialize slaves)

PPC_Mon>
```

The sysmem diagnostics scans randomly all the system memory doing read and write operations until 100% of the locations have been accessed. By pressing the space bar the diagnostic can be interrupted.

```
PPC_Mon>diag sysmem

entering system memory diagnostics

Checking System Memory (0x00004000 - 0x1fefb000)          100  -
PPC_Mon>
```

Execute diagnostics of local ethernet controller.

```
PPC_Mon>diag ethernet

entering am79c970 LANCE Controller diagnostics

ETHERNET controller found at PCI slot 0 [00:a2:80:11:55:99]

AM79C970 Internal loopback test..0..1..2..OK PASSED
AM79C970 External loopback test..0..1..2..OK PASSED
Speed: 100B/T
PPC_Mon>
```

On the PMC#1 slot a FETH mezzanine (Fast Ethernet Controller) is installed. When running the PMC diagnostic, PPCMon identifies the PCI device and checks if a test is available for that device. If this is the case, the test is run. In this particular case, the ethernet connection was not install, leading to a failure of the external loopback test.

```
PPC_Mon>diag pmc1

entering PMC #1 diagnostics

ETHERNET controller found at PCI slot 1 [00:80:a2:00:18:d9]

AM79C970 Internal loopback test..0..1..2..OK PASSED
AM79C970 External loopback test  FAILED
PPC_Mon>
```

Now execute diagnostic of PMC#2.

```
PPC_Mon>diag pmc2

entering PMC #2 diagnostics

SCSI controller found at PCI slot 2

SCSI DEVICE 0:   DISK   IBM     DDRS-34560D     DC1BBX  UNRECOGNIZED
SCSI DEVICE 1:  NO DEVICE
SCSI DEVICE 2:  NO DEVICE
SCSI DEVICE 3:  NO DEVICE
SCSI DEVICE 4:  NO DEVICE
SCSI DEVICE 5:  NO DEVICE
SCSI DEVICE 6:  NO DEVICE
SCSI DEVICE 7:  NO DEVICE
PPC_Mon>
```

Using the status command, a summary of the diagnostic results is displayed.

```
PPC_Mon>status

   diagnostic | level |  status   | step
   -------------------------------------
   sysmem     |   2   |  SKIPPED  |  1
   bridge     |   2   |    OK     |  2
   mmu        |   2   |    OK     |  4
   mvram      |   2   |    OK     |  2
   intctl     |   2   |    OK     |  10
   serial     |   0   |  SKIPPED  |
   timers     |   2   |    OK     |  8
   fifos      |   2   |    OK     |  9
   thermo     |   0   |    OK     |  4
   ethernet   |   2   |    OK     |  6
   vme        |   2   |    OK     |  5
   xpci       |   2   |    OK     |  9

PPC_Mon>
```

We can verify that the ethernet diagnostics has been executed at level 2 and completed successfully, while the PMC test failed at step 5.

Shows all possible diagnostics for a RIO2 8060, RIO2 8061 or RIO2 8062.

```
PPC_Mon>diag

syntax: diag <device>

        where device = sysmem (system memory)
                       bridge (PCI bridge)
                       mmu (PPC603 MMU and CACHES)
                       nvram (mk48t08 NVRAM and Real Time Clock)
                       intctl (sic6351 interrupt controller)
                       serial (pc87312 super IO controller)
                       timers (zcio8536 micro timers)
                       fifos (message passing FIFO's)
                       keyb (upi82c42pc Keyboard and Mouse Controller)
                       thermo (ds1620 Digital Thermometer)
                       pci (scan PCI tree for devices)
                       ethernet (am79c970 Ethernet Controller)
                       pmc1 (PCI Mezzanine-device #1)
                       pmc2 (PCI Mezzanine-device #2)
                       vme (VME master port (need VMDIS8004))

PPC_Mon>
```

Running the **diag vme** command checks that the FPGA's controlling the VME have been loaded and re-initialize the VME interface. It then scans the VME bus to find a VMDIS 8003 or VMDIS 8004 display and verify VME accesses.

```
PPC_Mon>diag vme

entering VME master port diagnostics
Fpga Load : 806213a.bin rev 13a 8-7-97 chsm = 38b9ba4d CHSM OK
Fpga Load : 806214b.bin rev 14b 27-10-97 chsm = cbc0851d CHSM OK
VME interface initialized
Display VMDIS8003 at addr 0x0000 (am = 01)
PPC_Mon>
```

Shows all possible diagnostics for a RIOC 4063 or RIO3 8064.

```
PPC_Mon>diag

syntax: diag <device>

where device = sysmem (system memory)
bridge (PCI bridge)
mmu (PPC603 MMU and CACHES)
nvram (mk48t08 NVRAM and Real Time Clock)
intctl (sic6351 interrupt controller)
serial (pc87312 super IO controller)
timers (zcio8536 micro timers)
fifos (message passing FIFO's)
keyb (upi82c42pc Keyboard and Mouse Controller)
thermo (ds1620 Digital Thermometer)
pci (scan PCI tree for devices)
ethernet (am79c970 Ethernet Controller)
pmc1 (PCI Mezzanine-device #1)
pmc2 (PCI Mezzanine-device #2)
cpci (scan CPCI to initialize slaves)

PPC_Mon>
```

Running the **diag cpci** command re-initializes the CPCI bridge, and if the board is system controller after a delay (**set** in the **cpci_delay** environment parameter, which allows all slave devices to finish booting, CPCI bus is scanned and all slave devices found are initialized.

```
PPC_Mon>diag cpci

entering CPCI master port diagnostics
Fpga Load : 406201b.bin rev 01b 27-10-97 chsm = 0883d779 CHSM OK
board type : 4063
PLX CHIP 9080 REVISION 02
LocSpace Loc_Size _Local_Address_Window _CPCI_Address_Window_
Register 100 _PLX_REGISTER_ACCESS_ 00000000 MEM enabled
Register 100 _PLX_REGISTER_ACCESS_ 00000000 IO enabled
Space0 10000 01400000 IO enabled 00000000 IO enabled
Space1 1000000 00000000 MEM enabled 00000000 MEM enabled
ExpROM 20000 3f000000 MEM disabled 00000000 MEM enabled

scanning PCI bus for devices:
vid | did |slot|func|bus |b_id| addr | size | size_max
------|------|----|----|----|----|----------|----------|----------
 10b5 | 9080 | 00 | 00 | 00 | 00 | 00000000 | 00031000 | 00010000
      |      |    |    |    |    | 00000000 | 03200000 | 01000000
      |      |    |    |    |    | 00000000 | 00000000 | 00000000
 10d9 | 4063 | 01 | 00 | 00 | 00 | 00000000 | 00011000 | 00010000
      |      |    |    |    |    | 00000000 | 01200000 | 01000000
      |      |    |    |    |    | ffffffff | 00000000 | 00000000
 10d9 | 4064 | 05 | 00 | 00 | 00 | 00020000 | 00011000 | 00010000
      |      |    |    |    |    | 02000000 | 01200000 | 01000000
      |      |    |    |    |    | ffffffff | 00000000 | 00000000
PPC_Mon>
```

Shows all possible diagnostics for a MFCC 8441.

```
MFCC_Mon>diag

syntax: diag <device>

where device = sysmem (system memory)
bridge (PCI bridge)
mmu (PPC MMU and CACHES)
intctl (interrupt controller)
timer (TIC timer and WatchDog)
fifos (message passing FIFO's)

MFCC_Mon>
```

Run the FIFO diagnostic.

```
MFCC_Mon>diag fifos

entering message passing FIFO's diagnostics
Checking FIFO -> cleared
non empty

*** External Interrupt ***
got interrupt
data OK
empty
interrupt cleared
MFCC_Mon>
```

And check the status.

```
MFCC_Mon>status

diagnostic | level | status | step
---------------------------------------
    sysmem |     0 | SKIPPED|
      trap |     0 |     OK | 6
    bridge |     0 |     OK | 2
       mmu |     0 |     OK | 1
    intctl |     0 |     OK | 3
    timers |     0 |     OK | 2
     fifos |     2 |     OK | 9

MFCC_Mon>
```

SEE ALSO

set, show, status

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
dis	**x**	**x**	**x**	**x**	**x**	

NAME

dis - disassembles PPC code

SYNTAX

dis *addr*

PARAMETERS

addr address [**0x..** hexa, **od..** dec, **0o..** oct]

DESCRIPTION

The **dis** command disassembles the 32-bit instruction at address *addr*. PPC code is expected. Entering **CR** disassembles the next instruction. To return to PPCMon enter . and **CR**.

EXAMPLE

```
PPC_Mon>dis 200
0x00000200 0x90003000 -> stw    r0,0x3000(r0)
0x00000204 0x90203004 -> stw    r1,0x3004(r0)
0x00000208 0x7c1a02a6 -> mfspr  r0,26
0x0000020c 0x7c1243a6 -> mtspr  274,r0
0x00000210 0x7c1b02a6 -> mfspr  r0,27
0x00000214 0x7c1343a6 -> mtspr  275,r0
0x00000218 0x3c007000 -> addis  r0,r0,0x7000
0x0000021c 0x60004054 -> ori    r0,r0,0x4054
0x00000220 0x7c1a03a6 -> mtspr  26,r0
0x00000224 0x38000030 -> addi   r0,r0,0x0030 .

PPC_Mon>
```

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
dc			x		x	
dm	x	x	x	x	x	x
dp			x	x	x	x
dv			x	x		
dx			x	x	x	

NAME

dc, **dm**, **dp**, **dv, dx** - displays a data buffer from CompactPCI, system memory, local PCI, PCI2, VME

SYNTAX

```
dc.x[s] start[..[end [space] ]
dm.x[s] start[..end ]
dp.x[s] start[..end [space] ]
dv.x[s] start[..end [amode ]
dx.x[s] start[..end [space] ]
```

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
start	start address	[**0x..** hexa , **0d..** dec, **0o..** oct]
end	end address	[**0x..** hexa , **0d..** dec, **0o..** oct]
space	PCI space	[**io**, **mem**]
amode	VME address mode	[from **0x00** to **0xff**]

DESCRIPTION

The **dm** command reads and displays the contents of locations pointed by the CPU addresses starting from *start* address and ending at *end* address. The size of the data is given by *x* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. The number of data displayed is (<*end*> - <*start*>) / <data size>. If **s** is appended to the size parameter *x*, the data is swapped according to its size (32 bit: abcd -> dcba, 16 bit: ab -> ba). This option is useful when accessing PCI or CPCI bus because they follow the little endian byte ordering while the PowerPC follows the big endian byte ordering.

The first address of the data buffer to display is given by the *start* parameter. If no *end* parameter is given, 0x80 Bytes are displayed. Otherwise end - start Bytes are displayed and the default value for the number of Bytes to display is updated with that value.
To display a VME buffer with the dv command, the VME address mode *amode* (from **0x00** to **0xff**) must be defined. On RIO2 806x it defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39). The default value is set to 0x9. On a RIO3 8064 *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0.

To display a PCI or CPCI buffer with the **dp**, **dx** or **dc** command, PCI space must be defined. The following ASCII strings are accepted: **io** for PCI I/O space, **mem** for PCI memory space and **iack** for PCI configuration space. By default it is set to **io**.

Every time the command is executed, its parameters are stored in a static data structure. Missing parameters are taken from that data structure.

EXAMPLE

Display the system memory locations pointed by address 0 to 0x80 (32-bit accesses).

```
PPC_Mon>dm.l 0..80
0x00000000 : 90003000 90203004 7c1a02a6 7c1243a6 ..0...0.......C.
0x00000010 : 7c1b02a6 7c1343a6 3c007000 60004054 ......C...p....T
0x00000020 : 7c1a03a6 38000030 7c1b03a6 38000000 ....8..0....8...
0x00000030 : 9060300c 7c600026 90603094 4c000064 ..0.......0.L..d
0x00000040 : 00000000 00000000 00000000 00000000 ................
0x00000050 : 00000000 00000000 00000000 00000000 ................
0x00000060 : 00000000 00000000 00000000 00000000 ................
0x00000070 : 00000000 00000000 00000000 00000000 ................
PPC_Mon>
```

Display in byte mode the contents of the FPROM at offset 0 (CPU address 0xfef00000).

```
PPC_Mon>dm.b fef00000
0xfef00000 : 43 45 53 20 50 50 43 42 4f 4f 54 20 33 2e 33 31  CES.PPCBOOT.3.31 0xfef00010
: aa aa aa aa 55 55 55 55 00 00 00 00 00 00 00 00  ....UUUU........ 0xfef00020 : 48 00 00
00 48 00 00 00 48 00 00 00 48 00 00 00  H...H...H...H... 0xfef00030 : 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00  ................ 0xfef00040 : 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00  ................ 0xfef00050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00  ................ 0xfef00060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................ 0xfef00070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................ PPC_Mon>
```

On a RIO3 8064 display a data buffer (32-bit accesses) from VME address 0x8100000 to 0x8100100. VME address mode 0xa0 is used to select A32 VME addressing mode.

```
PPC_Mon>dv.l 8100000..8100100 a0
0x08100000 : 66667766 6f7f7767 99010991 90990918  ffwfo.wg........
0x08100010 : 6677f666 6f7ff66f 18000890 90800990  fw.fo..o........
0x08100020 : 77f7ff6e 6feff67f 18000891 90800180  w..no...........
0x08100030 : f7ffff66 6ff7f67f 99000199 90000980  ...fo...........
0x08100040 : e7fff767 6f67ff7f 88000198 90900890
 ...gog..........
0x08100050 : f7fff76f 6f6ff67f 98000899 90800880  ...ooo..........
0x08100060 : e7f7f666 6f7ff66f 19000099 90800800  ...fo..o........
0x08100070 : 667fe766 6f6ef66f 18800199 90900810  f..fon.o........
0x08100080 : e6ffff66 6feff67f 98000899 90900980  ...fo...........
0x08100090 : 67fff66e 6f7ff76f 19000190 90810980  g..no..o........
0x081000a0 : 67fff766 6f6ff7ff 18000998 90000990  g..foo..........
0x081000b0 : 66fff667 6f7ff76f 98000199 90900910  f..go..o........
0x081000c0 : 77fff667 6f6ff67f 19000999 90100980  w..goo..........
0x081000d0 : 67ffff67 6feff67f 18000098 90800810  g..go...........
0x081000e0 : 777ff666 6f77eeff 88800990 90900890  w..fow..........
0x081000f0 : 76ffff6e 6f7ff67f 88000999 90900880  v..no...........
PPC_Mon>
```

Trying to display a data buffer from a non-existent VME device reports a bus error.

```
PPC_Mon>dv.l 0
0x00000000 :
dv : bus error
PPC_Mon>
```

On a RIO2 8062, display the first eight PCI configuration registers of the onboard ethernet device using the bridge special address window 0x808xxxxx. This is done using 16-bit accesses in the PCI I/O space.

```
PPC_Mon>dp.s 800800..800810 io
0x00800800 : 2210 0020 0500 8002 1600 0002 0080 0000  ................
PPC_Mon>
```

The first register (2210) is the AMD vendor ID, the next register (0002) is the device ID and so on. The data swapping is due to the little endian byte ordering of the PCI bus. Using the s extension for the data size displays the registers value correctly.

```
PPCMon>dp.ss 800800..800810 io
0x00800800 : 1022 2000 0005 0280 0016 0200 8000 0000  ................
PPCMon>
```

On a RIOC 4065 board, show CPCI device table using the cpci command.

```
PPC_Mon>cpci show

CPCI interface [enabled]
System Controller
Slave       : A32 Size = 32 Mbytes
A32 Base = 0x40000000
A64 Base = 0x00000000'00000000  [disabled]

+----+--------------------------------------+
|          PCI DEVICES  [tree 8 ]          |
+----+------+------+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|pmc |
+----+------+------+----+----+----+----+----+
|  0 | 10d9 | 4065 | 00 | 00 | 00 |  0 |  x |
|  1 | 10d9 | 4065 | 01 | 00 | 00 |  0 |  x |
|  2 | 10d9 | 4064 | 03 | 00 | 00 |  0 |  x |
+----+------+------+----+----+----+----+----+

PPC_Mon>cpci show dev 3

Single function device found in slot : 0x03   bus : 0x00

| vid  | did  | type |   addr   |   size   | space| bar  |
+------+------+------+----------+----------+------+------+
| 10d9 | 4064 |  D   | 43000000 | 00100000 | MEM  |  0   |
                     | b0000000 | 00001000 | IO   |  1   |
                     | 43100000 | 00100000 | MEM  |  2   |
                     | 42000000 | 01000000 | MEM  |  3   |
                     | 43200000 | 00100000 | PROM |
PPC_Mon>
```

In slot#3 a RIOC 4064 has been found. Display (32-bit access) its base address registers (configuration registers starting at offset 0x10) using the command **pcc**.

```
PPC_Mon>pcc.l 10 ? 3
0x10 [0] [0x03] [0x800] : 43000000 ->
0x14 [0] [0x03] [0x800] : b0000001 ->
0x18 [0] [0x03] [0x800] : 43100000 ->
0x1c [0] [0x03] [0x800] : 42000000 ->
0x20 [0] [0x03] [0x800] : 00000000 ->
0x24 [0] [0x03] [0x800] : 00000000 -> .
PPC_Mon>
```

The register at offset 0x1c controls a window in the MEM space (because bit 0 is 0) pointing to the RIOC 4064 system memory (because the **cpci** environment parameters of that board have been **set** such that the space 1 window points to the system memory). The command **dc** allows the user to display these locations.

```
PPC_Mon>dc.l 42000000
0x42000000 : 90003000 90203004 7c1a02a6 7c1243a6  ..0...0.......C.
0x42000010 : 7c1b02a6 7c1343a6 3c007000 60004054  ......C...p....T
0x42000020 : 7c1a03a6 38000030 7c1b03a6 38000000  ....8..0....8...
0x42000030 : 9060300c 7c600026 90603094 4c000064  ..0.......0.L..d
0x42000040 : 00000000 00000000 00000000 00000000  ................
0x42000050 : 00000000 00000000 00000000 00000000  ................
0x42000060 : 00000000 00000000 00000000 00000000  ................
0x42000070 : 00000000 00000000 00000000 00000000  ................
PPC_Mon>
```

On a MFCC 8441 the hardware signature is stored swapped in the last sector of the data flash (seen by the local CPU at address 0xe807ff00).

```
MFCC_Mon>dm.ls e807ff00
0xe807ff00 : 43455320 4d464343 38343431 41414400 CES.MFCC8441AAD.
0xe807ff10 : 534e2020 50522030 322f3038 00000000 SN..PR.02.08....
0xe807ff20 : 52657620 483a312e 30320000 00000000 Rev.H.1.02......
0xe807ff30 : 50726f64 20303320 39380000 00000000 Prod.03.98......
0xe807ff40 : 54657374 20303420 39380000 00000000 Test.04.98......
0xe807ff50 : 00000000 00000000 00000000 00000000 ................
0xe807ff60 : 00000000 00000000 00000000 00000000 ................
0xe807ff70 : 00000000 00000000 00000000 00000000 ................
MFCC_Mon>
```

SEE ALSO

pm

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
dma	**x**	**x**	**x**	**x**	**x**	**x**

NAME

dma - execute dma related operation

SYNTAX

dma.x *operation [para0] [para1]*

PARAMETERS

operation the following operations are accepted:

open	*slot function bus*		
close			
start	**d** *des ssrc_start..src_end*	**b** *block*	**t** *timeout mode*
restart			
stop			
status	*mode*		

DESCRIPTION

The **dma** command allows the user to request data transfer operations performed by NCR53C7xx or NCR53C8xx SCSI controller installed on local PCI, extended PCI or CPCI. It is able to move blocks of data from PCI to PCI.

The following operations are accepted by the **dma** command:

The **open** operation enables access to the dma controller identified by its PCI slot, function and bus numbers. If these parameters are not given in the open operation, they are taken from the **dma** environment parameters.

The start operation allows the user to transfer a block of data with the following parameters:

des	destination address
src_start	start address of data
src_end	end address of data
block	block size (byte)
timeout	time out (usec)
mode	transfer mode
c	continuous
o	one shot (default)
w	wait

If the **w** mode has been selected, the data transfer is activated by the restart operation.

If the **c** mode has been selected, data transfer is halted by the stop operation.

The **status** operation displays the value of the DMA control registers at the end of DMA transfer. The **u** mode forces the readout of these registers.

The **close** operation disables access to the **dma** controller.

EXAMPLES

Using dma channel 0, transfers a 4-KByte block from system memory address 0x500000 to local PCI address 0xfa020000 (static RAM) by blocks of 16 Bytes.
A time-out of 1 second has been chosen.

Having installed a SCSI 84xx PMC on the local PCI slot 2 of a RIOC 4065, we set up PCI parameters the DMA controller using logical channel 0:

```
PPC_Mon>pci show dev 3

Multi function device found in slot : 0x03   bus : 0x00

| func | vid  | did  | type |   addr   |   size   | space| bar  |
+------+------+------+------+----------+----------+------+------+
|  00  | 1000 | 000b |  D   | f6002000 | 00001000 | IO   |  0   |
                            | c0300000 | 00100000 | MEM  |  1   |
                            | c0400000 | 00100000 | MEM  |  3   |

|  01  | 1000 | 000b |  D   | f6003000 | 00001000 | IO   |  0   |
                            | c0500000 | 00100000 | MEM  |  1   |
                            | c0600000 | 00100000 | MEM  |  3   |

PPC_Mon>set dma_chan0

Channel 0
dma_chan0_slot : 2 -> 3
dma_chan0_func : 0 -> 0
dma_chan0_bus : 100 -> 100
dma_chan0_script : 0 -> 0

PPC_Mon>
```

Fill the local buffer (at 0x500000) with 0xdeadface and the VME destination buffer (at 0x8100000) with 0x12345678. Check the content of the VME destination buffer.

```
PPC_Mon>fm.l 500000..501000 deadface
PPC_Mon>fm.l fa020000..fa021000 12345678
PPC_Mon>dm.l fa020000
0xfa020000 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020010 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020020 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020030 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020040 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020050 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020060 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0xfa020070 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
PPC_Mon>
```

Open the DMA channel and start the data transfer.

```
PPC_Mon>dma.0 open
PPC_Mon>dma.0 status
DMA channel 0 : slot = 3  func = 0  bus = 100    status : open not ready   [ dstat = 84
istat = 01]
PPC_Mon>dma.0 start dfa020000 s500000..501000 b16 t1000000
des = fa020000 src  = 00500000  len = 1000 mode = c0
dma : DMA channel 0 done
PPC_Mon>
```

Display DMA status and show data buffer in the static RAM.

```
PPC_Mon>dma.0 status
DMA channel 0 : slot = 3  func = 0  bus = 100    status : open ready  stopped   [ dstat =
84  istat = 01]
PPC_Mon>dma.0 close
PPC_Mon>dma.0 status
DMA channel 0 : slot = 3  func = 0  bus = 100    status : closed
PPC_Mon>dm.l fa020000
0x08100000 : deadface deadface deadface deadface ................
0x08100010 : deadface deadface deadface deadface ................
0x08100020 : deadface deadface deadface deadface ................
0x08100030 : deadface deadface deadface deadface ................
0x08100040 : deadface deadface deadface deadface ................
0x08100050 : deadface deadface deadface deadface ................
0x08100060 : deadface deadface deadface deadface ................
0x08100070 : deadface deadface deadface deadface ................
PPC_Mon>
```

SEE ALSO

set

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
dr	x	x	x	x	x	x

NAME

dr - displays register

SYNTAX

dr *reg*

PARAMETERS

reg	register set	
	bcr	*bridge control registers (on RIO2 only)*
	mmu	*PPC BAT and SR registers*
	conf	*PPC configuration registers*

DESCRIPTION

The **dr** command displays the current values of PPC and PCI bridge registers.

If the **bcr** register set is chosen, the 256 internal registers of the IBM27-82660 PCI bridge will be displayed. These registers are initialized at boot time by the firmware and should not be changed.

The **mmu** register sets contains the 16 BAT and 12 SR PowerPC registers. The registers are displayed as followed:

Table 1-1 MMU Registers Sets

IBAT0U	IBAT0L	IBAT1U	IBAT1L
IBAT2U	IBAT2L	IBAT3U	IBAT3L
DBAT0U	DBAT0L	DBAT1U	DBAT1L
DBAT2U	DBAT2L	DBAT3U	DBAT3L
SR0	SR1	SR2	SR3
SR4	SR5	SR6	SR7
SR8	SR9	SR10	SR11
SR12	SR13	SR14	SR15

To display the MSR, PVR and HID PowerPC register, the **conf** register set must be used.

EXAMPLE

On a RIO8062 display all IBM27-82660 PCI bridge control registers.

```
PPC_Mon>dr bcr

        0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
 ------------------------------------------------------------------------
 00    14  10  37  00  04  00  00  82  02  00  00  06  00  00  00  00
 10    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 20    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 30    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 40    00  00  80  00  00  00  00  00  00  00  00  00  00  00  00  00
 50    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 60    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 70    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 80    00  20  20  20  00  00  00  00  00  00  00  00  00  00  00  00
 90    1f  1f  1f  1f  00  00  00  00  00  00  00  00  00  00  00  00
 a0    01  36  0a  00  44  44  44  44  00  00  00  00  00  00  00  00
 b0    00  43  00  00  07  00  50  00  0a  00  07  4f  00  00  00  00
 c0    81  00  00  81  10  00  00  06  00  00  00  00  00  00  00  00
 d0    e8  01  f8  01  49  00  00  00  00  00  00  00  00  00  00  00
 e0    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
 f0    00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00

PPC_Mon>
```

Display MMU related registers.

```
PPC_Mon>dr mmu

   0x00001fff   0x00000012   0x3ff0001f   0x1ff00012
   0x10001fff   0x10000012   0x00000000   0x00000000
   0x00001fff   0x00000012   0x3ff0001f   0x1ff00012
   0x10001fff   0x10000012   0xf0001fff   0xf000002a
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000

PPC_Mon>
```

Display CPU configuration registers.

```
PPC_Mon>dr conf
   msr  = 0x00003030   pvr  = 0x00088302   l2cr = 0xbb804016
   hid0 = 0x0000c000   hid1 = 0xb0000000 [450 Mhz]
PPC_Mon>
```

SEE ALSO

pr

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
exec	**x**	**x**	**x**	**x**	**x**	**x**

NAME

exec - starts the execution of xcoff, elf, lynx or prep file

SYNTAX

exec *[addr]*

PARAMETERS

addr base address of the file header [**0x..** hexa, **0d..** dec, **0o..** oct]

DESCRIPTION

exec looks for a known header (xcoff, elf, lynx or prep) at address *addr* and analyzes it to get the code entry parameter. It saves the current toc and executes a jump and link at code entry.

EXAMPLE

Using the **load** command, download the application "/tftpboot/mycode" (xcoff file) from Ethernet at memory address 0x1000000 in binary mode and execute it. The **exec** command analyses the file header at 0x1000000 in order to install the code sections in memory, find the entry point of the application (0x404530), saves the monitor context and starts execution. The application entry point is called as a “C” function in order to return to the monitor after execution.

```
PPC_Mon>load le:10.0.4.49 /tftpboot/mycode 1000000 binary

Trying device  le:10.0.4.49   [file : /tftpboot/mycode] ...
ethernet load
local address = 00:80:a2:01:00:ca
using ARPA
.file server address 00:06:5b:b8:bd:eb
loading file '/tftpboot/mycode' from server 10.0.4.49 at address 0x1000000
tftp packet number:       d6
transfer rate: 1210 kbyte/sec [len = 0x1a979]
PPC_Mon>
PPC_Mon>exec 1000000
starting code execution at address: 4000
testing usercode
test ended
PPC_Mon>
```

SEE ALSO

go, **load**

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
fc			x		x	
fm	x	x	x	x	x	x
fp			x	x	x	x
fv			x	x		
fx			x	x	x	

NAME

fc, **fm**, **fp**, **fv**, **fx** - fills memory CPCI, memory PCI, memory VME or extended PCI

SYNTAX

```
fc.x[s] start..end data[cpci_space]
fm.x[s] start..end data
fp.x[s] start..end data[space]
fv.x[s] start..end data[amode]
fx.x[s] start..end [space]
```

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
start	start address	[**0x..** hexa, **0d..** dec, **0o..** oct]
end	end address	[**0x..** hexa, **0d..** dec, **0o..** oct]
data	data to write	[**0x..** hexa, **0d..** dec, **0o..** oct]
space	PCI space	[**io**, **mem**]
amode	VME address mode	[from **0x00** to **0xff**]

DESCRIPTION

The **fm** command fills a block of memory starting at address *start* and ending at address *end* with the data *data*. Data accesses are done according to the data size specified by *x* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. If *s* is appended to the size parameter *x*, the data is swapped according to its size (32-bit: abcd -> dcba, 16-bit: ab -> ba). This option is useful when accessing PCI or CPCI busses because they follow the little endian byte ordering, while the PowerPC follows the big endian byte ordering.

The first address of the data buffer to fill is given by the *start* parameter. If no *end* parameter is given, 0x80 Bytes are copied. Otherwise *end* - *start* bytes are copied and the default value for the number of byte to copy is updated with that value.

To fill a VME buffer with the **fv** command, the VME address mode *amode* (from **0x00** to **0xff**) must be defined. On RIO2 806x it defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39). The default value is set to 0x9. On a RIO3 8064 *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0.

To fill a PCI or CPCI buffer with the **fp**, **fx** or **fc** command, PCI space must be defined. The following ASCII strings are accepted: **io** for PCI I/O space, **mem** for PCI memory space and **iack** for PCI configuration space. By default it is set to **io**.

Every time the command is executed, its parameters are stored in a static data structure. Missing parameters are taken from that data structure.

EXAMPLE

On a RIO3 8064, fill one block of VME memory (0x800000-0x800080) with the data 0xdeadface and address mode 0xa0 (32-bit accesses), then display the contents of that block.

```
PPC_Mon>fv.l 800000..800080 deadface a0
PPC_Mon>dv.l 800000
0x00800000 : deadface deadface deadface deadface ................
0x00800010 : deadface deadface deadface deadface ................
0x00800020 : deadface deadface deadface deadface ................
0x00800030 : deadface deadface deadface deadface ................
0x00800040 : deadface deadface deadface deadface ................
0x00800050 : deadface deadface deadface deadface ................
0x00800060 : deadface deadface deadface deadface ................
0x00800070 : deadface deadface deadface deadface ................
PPC_Mon>
```

SEE ALSO

dm, pm

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
fprom			**x**	**x**	**x**	

NAME

fprom - executes an FPROM operation

SYNTAX

fprom *operation para0 para1....*

PARAMETERS

operation the following operations are accepted:

erase	erases an FPROM address range
load	loads an FPROM address range with data
copy	copies a buffer from the memory to the FPROM
read	copies a buffer from FPROM to the memory

paraX parameters depending on the operation selected

DESCRIPTION

The **erase** operation has the following syntax:

fprom erase *start .. end*

start FPROM address offset of the first location to erase [**0x..** hex, **0d..** dec, **0o.**. oct]
end FPROM address offset of the last location to erase [**0x..** hex, **0d..** dec, **0o..** oct]

The **load** operation has the following syntax:

fprom load *start device file addr*

start FPROM address offset where to load the data file [**0x..** hex, **0d..** dec, **0o..** oct]
device device used to import the data file [**le**, **e0**, **e1**, **e2**, **s**, **s1**, **s2**, **fp**, **tty1**]
file name of the data file
addr address where to load the file temporary for copy [**0x..** hex, **0d..** dec, **0o..** oct]

NOTE Passing an offset lower than 100000 may overwrite PPCMon and make the board unusable.

The **copy** operation has the following syntax:

fprom copy *start addr size*

start FPROM address offset where to copy the data buffer [**0x..** hex, **0d..** dec, **0o..** oct]
addr buffer address [**0x..** hex, **0d..** dec, **0o..** oct]
size buffer size [**0x..** hex, **0d..** dec, **0o..** oct]

The **read** operation has the following syntax:

fprom read *start addr size*

start FPROM address offset from where the data shall be read [**0x..** hex, **0d..** dec, **0o..** oct]
addr buffer address where to store the data[**0x..** hex, **0d..** dec, **0o..** oct]
size number of bytes to read[**0x..** hex, **0d..** dec, **0o..** oct]

EXAMPLE

Load your standalone kernel over ethernet (**le**) and store it in the FPROM at offset 0x100000. A temporary copy of the file is kept in the system RAM at address 0x200000 during load procedure. It is important to choose a system RAM address within the range of available memory not used by other applications. Since PPCMon supports the self-extraction of GZIP files, you can compress the file after the compilation.

The internet address of the boot server is given with the boot device (le:10.0.6.15).

Now we can start loading.

```
PPC_Mon>fprom load 100000 le:10.0.6.15 /tftpboot/vxRio3Rio3.st.gz 200000
loading fprom... 100000

Trying device  le:10.0.6.15   [file : /tftpboot/vxRio3Rio3.st.gz] ...
ethernet load
local address = 00:80:a2:01:40:74
using ARPA
.file server address 00:03:ba:0b:70:85
loading file '/tftpboot/vxRio3Rio3.st.gz' from server 10.0.6.15 at address 0x200000
tftp packet number:      637
transfer rate: 906 kbyte/sec [len = 0xc6afd]
fprom_off = 100000   mem_addr = 200000  len = c6afd
FPROM erasing sector 001c0000
FPROM programming offset 001e0000
PPC_Mon>
```

Read data from FPROM offset 0 (where PPC_Mon is located) and store them in memory at address 0x100000.

```
PPC_Mon>fprom read 0 100000 4000
fprom_off = 0   mem_addr = 100000  len = 4000
PPC_Mon>dm.l 100000
0x00100000 : 43455320 50504342 4f4f5420 352e3230  CES PPCBOOT 5.20
0x00100010 : aaaaaaaa 55555555 52494f20 34303638  ....UUUURIO 4068
0x00100020 : 48000000 48000000 48000000 48000000  H...H...H...H...
0x00100030 : 61626364 65666768 00000000 00000000  abcdefgh........
0x00100040 : f900c000 f9008000 f9000000 f9000600  ................
0x00100050 : fa000000 fa100000 f8000000 f8000000  ................
0x00100060 : f9004000 ffffffff fa107fe0 f9009300  ..@.............
0x00100070 : f9009200 70000000 40000000 7f000000  ....p...@.......
PPC_Mon>
```

SEE ALSO

boot, load

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
go	**x**	**x**	**x**	**x**	**x**	**x**

NAME

go - starts a code execution

SYNTAX

go *address*

PARAMETERS

address	jump address	[**0x..** hexa, **0d..** dec, **0o..** oct]

DESCRIPTION

The firmware executes an absolute branch at *address*. If no address is given, it branches at 0x4000.

EXAMPLE

Load a file according to default parameters set in NVRAM and start the execution at memory address 0x4000.

```
PPC_Mon>load
network boot
loading file '/tftpboot/lynx.rom' from server
144.0.10.1
Ethernet address of server = 08.00.5a.cd.a0.00
loading at address:0
tftp packet number: 29c0
file length = 538200
PPC_Mon>go 4000
starting code execution at address: 4000
ramdisk_start 0x..b00c3000 RFS_SIZE 400000
enter vmeinstall:
sgoff 0 vmeadd 0 size 10000 am 29 rdpref 0 wrpost 0 swap 1
sgoff 1000000 vmeadd 0 size 1000000 am 39 rdpref 0 wrpost 0 swap 1
sgoff 2000000 vmeadd 0 size 1000000 am 39 rdpref 0 wrpost 0 swap 1
sgoff 3000000 vmeadd 0 size d000000 am 9 rdpref 0 wrpost 0 swap 1
VME device has been installed
```

SEE ALSO

boot, exec, load

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
gpio	x	x	x	x	x	

NAME

gpio - executes commands related to the GPIO 8405 PMC mezzanines

SYNTAX

gpio.*x cmd [para0] [para1]...*

PARAMETERS

x	gpio identifier
cmd	sub-command

scan	**sflash**	sign			
sflash	sign	fpga			
sflash	**check**				
sflash	*read*	*off*	*addr*	*len*	
sflash	*load*	*fpga #y*	*dev*	*file*	*addr*

DESCRIPTION

The **gpio** command is used to control a GPIO 8405 PMC mezzanine located directly on the motherboard or on a PMC carrier, from the PPCMon.

The GPIO identifier (PMC number) is given by the command extenuation *x,* where *x* is from **1** to **8**.

The **scan** sub-command doesn't take any identifier or parameter. For each GPIO found, PPCMon shows its PMC slot and PCI addresses..

The **sflash** subcommand by the parameter **sign** shows the GPIO 8405 signature: serial number and hardware revision stored in the last 256 Bytes of the SFLASH.

The SFLASH contains all of the files loaded into FPGA at power-ON. Each file is recognized by a signature containing: file name, creation date, offset in SFLASH, length and a checksum calculated at the loading time. The **sign** operation by the parameter **fpga,** lists the signatures of FPGA files currently loaded

The **sflash** subcommand by the parameter **check,** recalculates dynamically FPGA files parameters and compares the result with the signature. This allows the user to check SFLASH integrity.

The **read** operation transfers data from SFLASH starting at offset **off** to memory at address **addr**. **len** is the number of Bytes transferred.

The **sflash** sub-command allows, using the **load** operation, reloading over ethernet (**le**) **SFLASH** with a new FPGA file. **y** is the FPGA identifier: fpga#1 (PCI FPGA), fpga#2 (FE FPGA). FPGA tools provide different file formats: PPCMon supports “ttf” and “mcs” formats.

FPGAs can be loaded all together at the same time, based on the hardware programmation sequence or individually. In this case, the FPGA identifier is required.

addr is an address in memory used for temporary storage during the loading phase.

For safety a password and a control key is required, in fact passing a wrong file by the command **sflash** load, can make the GPIO unusable. In this case, FGPAs must be reprogrammed by the CISP tool.

EXAMPLE

Display all GPIO 8405 accessible from the RIO2 806x CPU using the scan sub-command.

```
PPC_Mon>gpio scan

 pmc  |  io_addr   |
------|------------|
  1   | 0x30000000 |

PPC_Mon>
```

One GPIO 8405 was detected in PMC slot#1. It is visible in the PCI I/O space at address 0x30000000.

```
PPC_Mon>gpio.1 sflash sign

CES GPIO 8405AA  SN  0120  Rev B  [Prod 05/99,Test 07/99]
PPC_Mon>
```

```
PPC_Mon>gpio.2 sflash sign fpga
FPGA#1 signature
  840501b.ttf
  Start 0x0040000
  Size 0x0000710d
  Pgm 30-05-2001
  CHSM 0x4ea4c9c6



FPGA#2 signature
  840510a.ttf
  Start 0x004710d
  Size 0x00012f57
  Pgm 30-05-2001
  CHSM 0xb4c0a947
```

```
PPC_Mon>gpio.2 sflash check
fpga_off = 40000 fpga_len = 1a06c
reading sflash:0005a06c
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  |  40000 |  710d | 4ea4c9c6 | OK
|  2  |  4710d | 12f57 | b4c0a947 | OK
```

```
PPC_Mon>gpio.1 sflash read 40000 300000 1000
sflash_off = 40000   mem_addr = 300000  len = 1000
```

```
PPC_Mon>gpio.2 sflash load fpga#2 le 840510a.ttf 100000
ethernet load
local address = 00:80:a2:00:8b:5e
using ARPA
file server address 08:00:20:c0:7a:3f
loading file '840510a.ttf' from server 10.0.6.25 at address 0x1000000
tftp packet number:      263
transfer rate: 464 kbyte/sec [len = 0x4c255]
sflash_id = 1020102ger
loading fpga file from mem_addr =  0x1000000  [len =  0x4c255]
converting file...[len = 0x12f57]

enter control key  -> c9284794
file_off = 0 file_len = 12f57 file_cks = b4c0a947
fpga_off = 4710d fpga_len = 12f57
reading sflash:0005a064

WARNING: DO NOT SWITCH OFF THE BOARD WHILE WRITING FPGA !!!

writing 0x12f57 bytes in sflash at offset 0x4710d
continue [n] -> y
writing sflash:0005a064
enter comment -> test
updating fpga signature
```

SEE ALSO

fprom

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
help	x	x	x	x	x	x

NAME

help - lists all commands

SYNTAX

help *[command]*

PARAMETERS

command command name

DESCRIPTION

help without parameters shows a list of all available commands. If a command name is given as parameter, a description of that command is displayed.

EXAMPLES

Show list of commands:

```
PPC_Mon>help

command list:

   bma          boot         cache        cm
   config       cpci         date         diag
   dis          disk         dc           dm
   dma          dp           dr           dv
   dx           exec         fc           fm
   fprom        fv           fp           go
   gpio         help         ide          lance
   lc           lm           load         lp
   lv           map          perf         mfcc
   mm           nvram        pc           pcc
   pci          plc          pm           pmc
   pp           pr           pv           px
   pxc          reset        riomon       rmc
   rmm          rmp          rmv          rtc
   scell        scsi         sdcard       set
   setenv       sflash       show         sc
   sm           sodimm       sv           sp
   status       temp         tstmon       tc
   tm           tv           tp           timer
   tinit        tlist        tstart       tty0
   tty1         user         upara        version
   vme          xpci
enter 'help <command>' for command syntax

PPC_Mon>
```

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
lc	**x**	**x**	**x**		**x**	
lm	**x**	**x**	**x**	**x**	**x**	
lp	**x**	**x**	**x**	**x**	**x**	
lv	**x**	**x**	**x**	**x**		
lx	**x**	**x**	**x**	**x**	**x**	

NAME

lc, lm, lp, lv, **lx** - loops on CPCI, memory, PCI, VME access or PCI2

SYNTAX

```
lc.x addr [data[space]]
lm.x addr [data]
lp.x addr [data[space]]
lv.x addr [data[amode]
lx.x addr [data[space]]
```

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
start	start address	[**0x..** hexa , **0d..** dec, **0o..** oct]
end	end address	[**0x..** hexa , **0d..** dec, **0o..** oct]
space	PCI space	[**io**, **mem**]
amode	VME address mode	[from **0x00** to **0xff**]

DESCRIPTION

The **lm** command must be used to read / write continuously from / to a CPU address. The size of the data is given by *x* where *x* shall be **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. *addr* is a CPU valid address (see CPU address mapping for VME, CPCI or PCI accesses). The address is automatically adjusted to a multiple of the data size chosen to avoid address alignment errors. If a *data* parameter is given, the location pointed by *addr* is continuously overwritten with *data,* otherwise continuous read cycles are performed. Entering any character allows to return to the command interpreter.

lp, **lc**, **lx** and **lv** perform access to PCI, CPCI and VME locations. To access VME or PCI addresses, an additional parameter is needed, *amo*de is the VME address mode (from 0x00 to 0xff). and *space* is the PCI space.

On RIO2 806x *amode* defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39). The default value is set to 0x9. On RIO3 8064 *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0.

The PCI space must be one of the following ASCII strings: **io** for PCI IO space, **mem** for PCI MEM space. If you want to read a VME or PCI addresses, replace data by **?** to indicate to the firmware not to overwrite the address.

At each access, parameters are stored in a data structure, and when they are missing in the command line, the firmware takes them from that data structure.

Command		Onboard	33170A	33170B	VME	CPCI	MFCCM on
load	**srec, disk**	**x**	**x**	**x**	**x**	**x**	
	net		**x**	**x**	**x**	**x**	
	fprom			**x**	**x**	**x**	**x**

NAME

load - loads a file in memory

SYNTAX

load *[dev [file [addr [filetype]]]]*

PARAMETERS

dev	boot device
file	file to load
addr	memory address where to load the file
filetype	type or file to load

DESCRIPTION

The **load** command loads at address *addr* a data buffer *file* from device *dev* according to the *filetype* found in the boot environment parameters.

Missing command parameters are taken from the boot environment. The **show / set** commands can be used to check or set the boot parameters. They are kept in NVRAM and don't have to be reloaded after power-on unless you want to modify them.

PPCMon can load files from ethernet, scsi, serial line and fprom.

To load from ethernet the device should be set to **e***i*, where *i* is the identifier of the PMC holding the ethernet controller. PMC identifiers are from **1** to **6. 0** is reserved for the local ethernet controller and is equivalent to device **le**. In this mode, the host and server internet addresses and the address resolution protocol should be set in the *inet* environment parameters. TFTP is used for loading and file shall be the pathname of the file to be loaded from the server (either full pathname or relative to /tftpboot according the server TFTP daemon configuration).

PPCMon allows booting from a SCSI-controlled disk or a tape. **prep** file type should be selected to boot from these devices in raw mode. In this mode PPCMon expects the first logical blocks to contain a PREP header. The file is then read sequentially according to the length found in the header. For a disk containing a LynxOS file system, PPCMon is able to load from a LynxOS partition a file referenced by its full pathname.

The SCSI load device is **s***ijkl* where *i* is the identifier of the PMC holding the ethernet controller, *j* is **d** for a disk or **t** for a tape, *k* is the SCSI ID of the disk or tape and, *l* is the partition identifier (**a** to **d** for a LynxOS file system). For compatibility with previous versions of PPCMon, **sd***yx* is equivalent to **s2d***yz*.

The serial lines load device is **tty***i* where is the serial port identifier (0 or 1). Only SREC files can be loaded through the serial lines. The command parameter file is not relevant in this case.

To load from FPROM, the device should be set to **fp**. In this case the command parameter file should be set to the FPROM address offset, where the file has been stored (see **fprom** command). When booting from FPROM, PPCMon is able to automatically recognize and expand compressed files produced by the 'gzip' utility.

The *filetype* environment parameter is very important for PPCMon in order to know how to load the file in memory.

If *filetype* is **binary**, *file* is copied Byte per Byte starting at address addr in system memory. This mode should be used for compatibility with PPCMon version 2.xx.

If *filetype* is **auto**, file is copied at address *addr* in system memory. PPCMon tries to recognize the object loaded in memory by looking in the first four Bytes for a magic number. The file is then loaded in memory according to address parameters found in the header.

EXAMPLE

Check the boot parameters and load a LynxOS kernel from disk.

```
PPC_Mon>show boot

BOOT parameters

   boot_flags [afmnNS] : a
   boot_device [s<1-8>d<0-7><a-d>, i<1-8>d<0-3><a-f>, le, e<0-2>, atm<1-6>@<vci>, tty<0-
1>,
 fp] : i2d0b
   boot_filename : /lynx.os
   boot_rootfs [s<1-8>d<0-7><a-d>, i<1-8>d<0-3><a-f>, rd] : id0b
   boot_delay : 3 sec
   boot_address : 1000000
   boot_size : 0
   boot_fast [ y n ] : n
   boot_filetype [ binary auto lynx prep srec ] : lynx
   boot_cpunr : 0
   boot_mode [ ppcmon test auto user ] : ppcmon
   boot_line :
PPC_Mon>

PPC_Mon>load

Trying device  i2d0b   [file : /lynx.os] ...
loading from disk(2,0,2) at address:1000000
disk capacity is 49577472 blocks
disk block size is 512 bytes
partition 2's size is 4096449 blocks
loading /lynx.os...
file length: 0x00210000
expanding file from address 1000000
xcoff file
PPC_Mon>
```

When a file is loaded in “auto” mode, at the end of the load phase, PPC_Mon checks its first bytes in order to recognize the file format. If successful, the file format is displayed and the file is installed in memory accordingly.

```
PPC_Mon>load le:10.0.4.49 /tftpboot/mycode 1000000 auto

Trying device  le:10.0.4.49   [file : /tftpboot/mycode] ...
ethernet load
local address = 00:80:a2:01:00:ca
using ARPA
.file server address 00:06:5b:b8:bd:eb
loading file '/tftpboot/mycode' from server 10.0.4.49 at address 0x1000000
tftp packet number:       d6
transfer rate: 1223 kbyte/sec [len = 0x1a979]
expanding file from address 1000000
xcoff file
PPC_Mon>
```

In binary mode the file is simply copied in memory at the load address.

```
PPC_Mon>load le:10.0.4.49 /tftpboot/mycode 1000000 binary

Trying device  le:10.0.4.49   [file : /tftpboot/mycode] ...
ethernet load
local address = 00:80:a2:01:00:ca
using ARPA
.file server address 00:06:5b:b8:bd:eb
loading file '/tftpboot/mycode' from server 10.0.4.49 at address 0x1000000
tftp packet number:       d6
transfer rate: 1210 kbyte/sec [len = 0x1a979]
PPC_Mon>
```

SEE ALSO

fprom, set, show

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
map	**x**	**x**	**x**	**x**	**x**	

NAME

map - shows the address mapping of the board's resources

SYNTAX

map

PARAMETERS

none

DESCRIPTION

The map command displays the CPU address and address window size of the boards local resources.

EXAMPLES

Show the address mapping of a RIOC 4065 board resources.

```
PC_Mon>map

  SERIAL   (  PC87312) at 0xf900c000
  LOC_REG              at 0xf9008000
  INT_CTL  (  SIC6351) at 0xf9000000
  SRAM     (128 kbyte) at 0xfa000000
  NVRAM    (  8 kbyte) at 0xfa100000
  RTC      (  MK48T59) at 0xfa107fe0
  TIMER    ( ZCIO8536) at 0xf9004000

PPCMon>
```

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
mfcc	**x**	**x**	**x**	**x**	**x**	

NAME

mfcc - executes commands related to the MFCC 8441 PMC mezzanines

SYNTAX

mfcc.*x cmd [para0] [para1]...*

PARAMETERS

x	mfcc identifier						
cmd	sub-command						
	scan						
	reset	*mode*					
	start	*mode*					
	connect						
	fprom		load	*start*	*device*	*file*	*addr*
	sflash	sign					
	sflash	sign	**fpga**				
	check	*read*	*addr*	*len*			
	sflash		*load*	*fpga #y*	*addr*	*file*	
	stop						

DESCRIPTION

The mfcc command should be used to control from PPCMon a MFCC 844x PMC mezzanine located directly on the RIO2 806x or RIO3 8064, RIOC 406x board or on a PEB 64xx extension. The MFCC identifier (PMC number) is given by the command extension x, where x is from 1 to 8.

The scan sub-command doesn't take any identifier, nor parameter. For each MFCC found, PPCMon shows its PMC slot, PCI addresses, boot step and status.

The stop sub-command puts the MFCC 844x in reset state.

The start sub-command releases the MFCC 844x reset signal. If the MFCC 844x is already running, it has no effect.

The reset sub-command executes a stop, followed by a start. It reboots the MFCC 844x. At the boot time PPCMon executes some diagnostic tests by default. The execution of these tests can be skipped by setting the mode to 0xc. This can be useful

to reset the MFCC if in case of problems if the MFCC hangs.

The connect sub-command should be used to interact with the MFCC 844x command interpreter MFCCMon. After that sub-command has been executed, all characters but q are passed by PPCMon to MFCCMon. Entering q allows returning to PPCMon.

The load operation has the following syntax: start is the FPROM offset where to reload the data file. Device is the device used to import the data file [le, e0, e1, e2, s, s1, s2, fp, tty1]. addr is the address where to load the file temporary for copy.

NOTE

Passing an offset lower than 100000 may overwrite the firmware.

The sflash subcommand by the parameter sign, shows the MFCC signature: serial number and hardware revision stored in the last 256 Bytes of the SFLASH.

The SFLASH contains all of the files loaded into FPGA at power-on. Each file is recognized by a signature containing: file name, creation date, offset in SFLASH, length and a checksum calculated at the loading time. The sign operation by the parameter fpga list the signatures of FPGA files currently loaded

The **sflash** subcommand by the parameter **check** recalculates dynamically FPGA file parameters and compares the result with the signature. This allows the user to check SFLASH integrity.

The **read** operation transfers data from SFLASH starting at offset off to memory at address **addr**. **len** is the number of Bytes transferred.

The sflash sub-command allows, using the load operation, to reload over ethernet (le) SFLASH with a new FPGA file. y is the FPGA identifier: fpga#1 (PCI FPGA), fpga#2 (FE FPGA). FPGA tools provide different file formats: PPCMon supports “ttf” and “mcs” formats.

FPGAs can be loaded all together at the same time, basing on the hardware programmation sequence or only one identified by the FPGA identified y.

addr is an address in memory used for temporary storage during the loading phase. For safety a password and a control key is required. In fact passing a wrong file by the command sflash load can make the MFCC unusable. In this case the FPGAs must be reprogrammed by the CISP tool.

EXAMPLES

Display all MFCC 844x devices accessible from the CPU board using the **scan** sub-command.

```
PPC_Mon>mfcc scan

 pmc  | io_addr  | mem_addr | step |  status
------|----------|----------|------|-----------
  1   | f6001000 | d0000000 |  00  |  reset

PPC_Mon>
```

One MFCC 844x was detected in PMC slot#1. It is visible in the PCI I/O space at address 0x30000000 and in the PCI MEM space at address 0x10000000. It is the reset state.

The sub-command **reset** shall reboot the MFCC 844x. The **.1** extension indicates that we access a devices located in PMC slot#1.

```
PPC_Mon>mfcc.1 reset
```

At this point the MFCC 844x is running the MFCCMon command interpreter. We can connect to that interpreter from PPCMon

with the connect sub-command.

```
PPC_Mon>mfcc.1 connect

connecting

  MFCC8443DC  SN  198  Rev H:A/B S:4.7
  CPU : PPC750   ver: 33.11   speed: 666 Mhz
  Memory size : 64 Mbytes
  FPROM :
  PCI bridge :
  I/O interface :
  Adaptator :

Entering boot diagnostics

 0    Check System Memory (0x00000000 - 0x03effffc)          0  SKIPPED
 1    Check Interrupt Handler                                0  OK
 2    Check PCI Bridge                                       0  OK
 3    Check Cache and MMU                                    0  OK
 4    Check Interrupt Controller                             0  OK
 5    Check TIC Timer and WatchDog                           0  OK
 6    Check FIFO's (0 - 4)                                   0  OK

************************************************
*   MFCC_Mon MFCC8443 monitor - version 5.2      *
*             CES SA Copyright 2004              *
************************************************

MFCC_Mon>
```

Now we are connected to the MFCCMon command interpreter. The RIO2/3 or RIOC CPU directly passes every character but **q** to the MFCC 844x.

Entering **help** displays a list of all MFCC_Mon commands. They are very similar to the PPCMon commands.

```
MFCC_Mon>help

command list:

   boot         cache        cm           config
   diag         dm           dr           dp
   ds           exec         fm           fp
   fs           go           help         lm
   load         lp           mm           nvram
   perf         pm           pp           pr
   ps           set          show         sm
   smon         sp           ss           status
   temp         tm           tp           tinit
   tlist        tstart       version
enter 'help <command>' for command syntax

MFCC_Mon>
```

To return to PPCMon just enter **q**.

Show the MFCC sflash content by using the **sflash sign** sun command:

```
PPC_Mon>mfcc.1 sflash sign
Hardware signature
  CES MFCC8443DC
  SN  198
  Rev H:A/B S:4.7
  Prod 37/02
  Test 38/02

PPC_Mon>mfcc.1 sflash sign fpga
FPGA#1 signature
  MFC8443H.mcs
  Start 0x0100000
  Size 0x0002c024
  Pgm 24-09-2002
  CHSM 0x9dc55403

  1c PCI-PPC Bridge

FPGA#2 signature
  afdx_040205.mcs
  Start 0x012c024
  Size 0x00078e64
  Pgm 11-02-2004
  CHSM 0x9fc00b33

  updated by JFG

PPC_Mon
```

The sflash check subcommand performs a scan of the sflash in order to identify FPGA bit streams it contain.

```
PPC_Mon>mfcc.1 sflash check
fpga_off = 100000 fpga_len = a4e90
reading sflash:001a4e90
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  | 100000 | 2c024 | 9dc55403 | OK
|  2  | 12c024 | 78e64 | 9fc00b33 | OK
PPC_Mon>
```

The following example shows how to read “by hand” the board signature (located 256 bytes below the top of SFLASH)

```
PPC_Mon>mfcc.1 sflash read 1fff00 100000 100
sflash_off = 1fff00   mem_addr = 100000  len = 100
reading : 00200000
PPC_Mon>dm.l 100000
0x00100000 : 43455320 4d464343 38343433 44430000  CES MFCC8443DC..
0x00100010 : 534e2020 31393800 00000000 00000000  SN  198.........
0x00100020 : 52657620 483a412f 4220533a 342e3700  Rev H:A/B S:4.7.
0x00100030 : 50726f64 2033372f 30320000 00000000  Prod 37/02......
0x00100040 : 54657374 2033382f 30320000 00000000  Test 38/02......
0x00100050 : 20202020 20202020 20202020 20202000                 .
0x00100060 : 20202020 20202020 20202020 20202000                 .
0x00100070 : 20202020 20202020 20202020 20202000                 .
PPC_Mon>
```

Next, we reload the bit stream file (file name is MFCC8443.mcs) for the 2 FPGAs of an MFCC8443. The file is loaded over ethernet from a TFTP file server.

```
PPC_Mon>mfcc.1 sflash load fpga le MFCC8443.mcs 100000
ethernet load
local address = 00:80:a2:01:00:73
using ARPA
file server address 08:00:20:c0:7a:3f
loading file 'MFCC8443.mcs' from server 10.0.6.25 at address 0x1000000
tftp packet number:      b1a
transfer rate: 1049 kbyte/sec [len = 0x163003]
sflash_id = 1000101ger
loading fpga file from mem_addr =  0x1000000  [len =  0x163003]
converting file...[len = 0x7e358]

enter control key  -> fffe072a

FPGA file MFCC8443.mcs contains 2 sections
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  | 100000 | 2c024 | 27ba998f | OK
|  2  | 12c024 | 52334 | 38ccc44a | OK


WARNING: DO NOT SWITCH OFF THE BOARD WHILE WRITING FPGA !!!

writing 0x7e358 bytes in sflash at offset 0x100000
continue [n] -> y
writing sflash:0017e358

enter comment -> RELEASE_2001-06-14
updating fpga signature
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  | 100000 | 2c024 | 27ba998f |
|  2  | 12c024 | 52334 | 38ccc44a |
```

The PPC_Mon generic fprom command can be used has a subcommand of the mfcc command. In this case operation are performed on the FPROM located on the MFCC. That command shall be used to update MFCC_Mon.

```
PPC_Mon>mfcc.1 fprom load 0 le:10.0.4.43 /tftpboot/8443prom.bin
ethernet load
local address = 00:80:a2:01:02:b5
using ARPA
file server address 00:c0:4f:09:09:53
loading file '/tftpboot/8443prom.bin' from server 10.0.4.43 at address 0x4000
tftp packet number:      202
transfer rate: 939 kbyte/sec [len = 0x40000]
fprom_off = 0   mem_addr = 4000  len = 40000
fprom type = 193 [AMD29F065: 1*8 Mbytes]
FPROM erasing sector offset 00030000
FPROM programming offset 00040000
PPC_Mon>
```

SEE ALSO

fprom, sflash

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
mm	**x**	**x**	**x**	**x**	**x**	**x**

NAME

mm - moves memory blocks

SYNTAX

mm.*x start..end des*

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
start	start address of the source block	[**0x..** hexa, **0d..** dec, **0o..** oct]
end	end address of the source block	[**0x..** hexa, **0d..** dec, **0o..** oct]
des	start address of the destination block	[**0x..** hexa, **0d..** dec, **0o..** oct]

DESCRIPTION

The **mm** command reads the block of memory starting at address *start* and ending at address *end* and copies it bit per bit in the block of memory starting at address *des*. Data accesses are done according to the data size specified in *x,* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access.

EXAMPLES

Fill one block of memory (0x800000 - 0x800080) with data 0x12345678 and another one (0x800000-0x800080) with 0. Then move the first block into the second one and display the contents of the second one.

```
PPC_Mon>fm.w 800000..800080 12345678
PPC_Mon>fm.w 800080..800100 0
PPC_Mon>mm.w 800000..800080 800080
PPC_Mon>mm.w 800000..800080 800080
PPC_Mon>dm.w 800080..800100
0x00800080 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x00800090 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x008000a0 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x008000b0 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x008000c0 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x008000d0 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x008000e0 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
0x008000f0 : 12345678 12345678 12345678 12345678  .4Vx.4Vx.4Vx.4Vx
PPC_Mon>
```

SEE ALSO

fm

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
nvram	**x**	**x**	**x**	**x**	**x**	

NAME

nvram - save / restore /switch the contents of the NVRAM

SYNTAX

nvram *operation*

PARAMETERS

operation	**save**
	restore
	switch

DESCRIPTION

The **nvram** command allows the user to save the overall contents of the NVRAM in FPROM and restore it. When the **save** operation is executed the 8 KBytes of NVRAM data are copied in FPROM at offset 0x80000. The **restore** operation reads the data from FPROM offset 0x80000 and updates the NVRAM. This mechanism is protected by a magic number stored in FPROM to be sure that its content comes from a previous **save** operation. During the boot phase, if a NVRAM corruption is detected, the firmware checks the FPROM at offset 0x80000 for valid NVRAM data. If valid data is found and if the user doesn't interrupt the boot by pressing the space bar within five sec, an automatic update of the NVRAM from the FPROM is performed and a programmed reset is generated. The switch operation saves the current content of the NVRAM in FPROM and restores the content previously saved.

EXAMPLES

Show the **inet** parameters and save NVRAM contents. Modify **inet_bootserver** parameter, check that the modification has been taken, then restore NVRAM from FPROM and check the **inet** parameters again.

```
PPC_Mon>show inet

INTERNET parameters

   inet_host [dotted decimal] : 10.0.108.6
   inet_bplane [dotted decimal] : 0.0.0.0
   inet_bootserver [dotted decimal] : 10.0.4.49
   inet_gateway [dotted decimal] : 0.0.0.0
   inet_nameserver [dotted decimal] : 0.0.0.0
   inet_protocol [ arpa bootp ] : arpa
   inet_mount :
   inet_ethermode [ 10 100 1000 auto ] : auto
PPC_Mon>
```

```
PPC_Mon>nvram save
saving NVRAM content to SFLASH at offset 0xfe000
PPC_Mon>
```

```
PPC_Mon>set inet

INTERNET parameters

   inet_host [dotted decimal] : 10.0.108.6 ->
   inet_bplane [dotted decimal] : 0.0.0.0 ->
   inet_bootserver [dotted decimal] : 10.0.4.49 ->
   inet_gateway [dotted decimal] : 0.0.0.0 ->
   inet_nameserver [dotted decimal] : 0.0.0.0 -> 10.0.4.23
   inet_protocol [ arpa bootp ] : arpa ->
   inet_mount :  ->
   inet_ethermode [ 10 100 1000 auto ] : auto ->

PPC_Mon>show inet

INTERNET parameters

   inet_host [dotted decimal] : 10.0.108.6
   inet_bplane [dotted decimal] : 0.0.0.0
   inet_bootserver [dotted decimal] : 10.0.4.49
   inet_gateway [dotted decimal] : 0.0.0.0
   inet_nameserver [dotted decimal] : 10.0.4.23
   inet_protocol [ arpa bootp ] : arpa
   inet_mount :
   inet_ethermode [ 10 100 1000 auto ] : auto
PPC_Mon>
```

```
PPC_Mon>nvram restore
restoring NVRAM content from SFLASH at offset 0xfe000
PPC_Mon>
PPC_Mon>show inet

INTERNET parameters

   inet_host [dotted decimal] : 10.0.108.6
   inet_bplane [dotted decimal] : 0.0.0.0
   inet_bootserver [dotted decimal] : 10.0.4.49
   inet_gateway [dotted decimal] : 0.0.0.0
   inet_nameserver [dotted decimal] : 0.0.0.0
   inet_protocol [ arpa bootp ] : arpa
   inet_mount :
   inet_ethermode [ 10 100 1000 auto ] : auto
PPC_Mon>
```

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
pc			**x**		**x**	
pm	**x**	**x**	**x**	**x**	**x**	**x**
pp			**x**	**x**	**x**	**x**
pv			**x**	**x**		
px			**x**	**x**	**x**	

NAME

pc, pm, pp, pv, px - patches CPCI, memory, PCI, VME or PCI2

SYNTAX

```
pc.x addr [data [space]]
pm.x addr [data]
pp.x addr [data [space]]
pv.x addr [data [amode]]
px.x addr [data [space]]
```

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
addr	address	[**0x..** hexa, **0d..** dec, **0o..** oct]
data	**?** or data to write	[**0x..** hexa, **0d..** dec, **0o..** oct]
space	PCI space [**io, mem**]	
amode	VME address mode	[from **0x00** to **0xff**]

DESCRIPTION

The **pm** command must be used to read / write a single data from / to a CPU address. The size of the data is given by *x,* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. *addr* is a CPU valid address (see CPU address mapping for VME or PCI accesses). The address is automatically adjusted to a multiple of the data size chosen to avoid address alignment errors. If a *data* parameter is given, the location pointed by **address** is overwritten with *data* and the firmware returns to the command interpreter.

If no *data* parameter is given, the data read at location pointed by *addr* is displayed and the firmware waits for a new data to overwrite the current location or for a control character:

Enter	the contents of next location is displayed
.	the firmware returns to the command interpreter
-	the contents of the previous location is displayed
=	the current location is read again and displayed
data	the current location is overwritten and the contents of the next location is read

pp, **px**, **pc** and **pv** allows direct access to PCI, PCI2, CPCI and VME locations. To patch the contents of VME or PCI addresses, an additional parameter is needed, *amode* is the VME address moder (from **0x00** to **0xff**) and *space* is the PCI space.

On RIO2 806x *amode* defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39). The default value is set to 0x9. On RIO3 8064 *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0.

The PCI space must be one of the following ASCII strings: **io** for PCI IO space, **mem** for PCI MEM space. If you want to read a VME or PCI address, replace data by **?** to indicate to the firmware not to overwrite the address.

At each access, parameters are stored in a data structure, and when they are missing in the command line, the firmware takes them from that data structure.

EXAMPLES

Overwrite the system memory location pointed by address 0x800000 with data 0x12345678 (32-bit access). Then read four consecutive memory locations with 16-bit accesses starting at the location pointed by address 0x800000. Then, exit **pm** command and execute it again with no arguments.

```
PPCMon>pm.w 800000 12345678
PPCMon>pm.h 800000
0x00800000 : 1234 ->
0x00800002 : 5678 ->
0x00800004 : 8108 ->
0x00800006 : 0080 -> .
PPCMon>pm
0x00800006 : 0080 -> .
PPCMon>
```

From a RIO3 8064, overwrite a VME location (address 0x8100000, address mode 0xa0) with 0x123456789 and read it back. Entering CR (carriage return) displays the next location. Overwrite that location with 0xdeadface then entering -, read back the two previous locations. Finally display the VME data buffer starting at 0x8100000.

```
PPCMon>pv.l 8100000 12345678 a0
PPCMon>pv.l 8100000
0x08100000 : 12345678 ->
0x08100004 : 67777f6f -> deadface
0x08100008 : 90090999 -> -
0x08100004 : deadface -> -
0x08100000 : 12345678 -> .
PPCMon>dv
0x08100000 : 12345678 deadface 90090999 18199990  .4Vx............
0x08100010 : 6ef67666 7ff67f6f 90080908 80098090  n.vf...o........
0x08100020 : 6ffff677 7ff6ef6f 90080818 80010890  o..w...o........
0x08100030 : 67fff7f7 7ff6f76f 90010009 80090090  g......o........
0x08100040 : 6ff7ffe7 7fff676f 98010888 80089090  o.....go........
0x08100050 : 6ff7fff7 7ff66f6f 90080098 80088090  o.....oo........
0x08100060 : 67f6f6e7 7ff67f6f 90000119 00088090  g......o........
0x08100070 : 67e77f66 6ff66e6f 90018009 10089090  g..fo.no........
PPCMon>
```

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
pcc			**x**		**x**	
plc	**x**	**x**	**x**	**x**	**x**	
pxc			**x**	**x**	**x**	

NAME

pcc, **plc**, **pxc** - ecutes PCI configuration cycles on CPCI, local PCI or PCI2

SYNTAX

pcc.*x[s] addr [data [device [bus]]]*
plc.*x[***s***] addr [data [device [bus]]]*
pxc.*x[***s***] addr [data [device [bus]]]*

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
addr	register offset	[**0x..** hexa, **0d..** dec, **0o..** oct]
data	**?** or data to write	[**0x..** hexa, **0d..** dec, **0o..** oct]
device	PCI device number	[from **0x00** to **0x1f**]
bus	*VME bus number*	[from **0x00** to **0xff**]

DESCRIPTION

The **pcc** command must be used to read / write a single data from / to a configuration register of a CPCI device. That command can be executed **only** if the board is **system controller**.

The **plc and pxc** command must be used to read / write a single data from / to a configuration register device located on board (local PCI or PCI2).

The size of the data is given by *x* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. If s is appended to the size parameter *x*, the data is swapped according to its size (32-bit: abcd -> dcba, 16-bit: ab -> ba). This option is useful when accessing PCI or CPCI bus because they follow the little endian Byte ordering, while the PowerPC follows the big endian Byte ordering.

addr is a configuration register address offset (four times the register index). The address is automatically adjusted to a multiple of the *data* size chosen to avoid address alignment errors. If a data parameter is given, the location pointed by *addr* is overwritten with *data* and the firmware returns to the command interpreter.

If no *data* parameter is given (a **?** shall be entered in the *data* field if *device* or *bus* parameters must be given), the data read at location pointed by *addr* is displayed and the firmware waits for a new data to overwrite the current location or for a control character:

Enter	the contents of next location is displayed
.	the firmware returns to the command interpreter
-	the contents of the previous location is displayed
=	the current location is read again and displayed
data	the current location is overwritten and the contents of the next location is read

device is the PCI device number of the device (sometimes it is referenced as PCI slot number). Basically the device number is a kind of geographical address for PCI configuration cycles and it is not necessary equivalent to the slot number. For example, on a CPCI bus, the slot number 7 is accessed as device 0xf (see CPCI specification for more information). *bus* is the PCI bus number on which the device is located.

Every time the command is executed, its parameters are stored in a static data structure. Missing parameters are taken from that data structure.

EXAMPLE

Show PCI device table using the pci command.

```
PPC_Mon>pci show

+----+-----------------------------------------+
|              PCI DEVICES  [tree 1 ]           |
+----+------+------+----+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|ivec|pmc |
+----+------+------+----+----+----+----+----+----+
|  0 | 1014 | 0037 | 00 | 00 | 00 |  0 | 00 |  7 |
|  1 | 1022 | 2000 | 01 | 00 | 00 |  0 | 11 |  x |
|  2 | 10d9 | 8441 | 02 | 00 | 00 |  0 | 26 |  2 |
+----+------+------+----+----+----+----+----+----+
```

The onboard ethernet controller has been implemented in PCI slot1. The **plc** command allows to display its configuration registers, such as the vendor ID and the device ID (16-bit registers starting at offset 0).

```
PPC_Mon>plc.s 0 ? 1 0
              0x00 [0] [0x01] [0x00] : 1022 ->
              0x02 [0] [0x01] [0x00] : 2000 -> .
              PPC_Mon>
```

On a RIOC 4065 board, show CPCI device table using the cpci command.

```
PPCMon>cpci show

CPCI interface [enabled]
   System Controller
   Slave       : A32 Size = 32 Mbytes
                 A32 Base = 0x40000000
                 A64 Base = 0x00000000'00000000  [disabled]

+----+---------------------------------------+
|           PCI DEVICES  [tree 8 ]          |
+----+------+------+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|pmc |
+----+------+------+----+----+----+----+----+
|  0 | 10d9 | 4065 | 00 | 00 | 00 |  0 |  x |
|  1 | 10d9 | 4065 | 01 | 00 | 00 |  0 |  x |
|  2 | 10d9 | 4064 | 05 | 00 | 00 |  0 |  x |
+----+------+------+----+----+----+----+----+
```

A RIOC 4064 has been found on the CPCI bus in cpci slot 5:

```
PPC_Mon>cpci show dev 5

Single function device found in slot : 0x05   bus : 0x00

| vid  | did  | type |   addr    |   size    | space| bar  |
+------+------+------+-----------+-----------+------+------+
| 10d9 | 4064 |  D   | 42000000 | 00100000 | MEM  |  0   |
                      | 70000000 | 00001000 | IO   |  1   |
                      | 42100000 | 00100000 | MEM  |  2   |
                      | 42200000 | 00100000 | MEM  |  3   |
                      | 42300000 | 00100000 | PROM |
```

Display (16-bit access) the vendor ID, the device ID, control and status registers of the CPCI device (RIOC 4064) located on CPCI slot 5 (bus 0) and return to PPCMon.

```
PPCMon>pcc.s 0 ? 5 0
0x00 [0] [0x05] [0x00] : 10d9 ->
0x02 [0] [0x05] [0x00] : 4064 ->
0x04 [0] [0x05] [0x00] : 0007 ->
0x06 [0] [0x05] [0x00] : 0280 -> .
PPCMon>
```

Display (32-bit access) its base address registers (configuration registers starting at offset 0x10) using the command **pcc**.

```
PPC_Mon>pcc.l 10 ? 5
0x10 [0] [0x05] [0x800] : 42000000 ->
0x14 [0] [0x05] [0x800] : 70000001 ->
0x18 [0] [0x05] [0x800] : 42100000 ->
0x1c [0] [0x05] [0x800] : 42200000 ->
0x20 [0] [0x05] [0x800] : 00000000 ->
0x24 [0] [0x05] [0x800] : 00000000 ->
0x28 [0] [0x05] [0x800] : 00000000 ->
0x2c [0] [0x05] [0x800] : 908010b5 ->
```

SEE ALSO

cpci, dm, pci

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
pci	x	x	x		x	
xpci	x	x	x	x	x	
cpci	x	x	x	x	x	

NAME

pci - handles PCI devices
xpci - handles extended PCI device table
cpci - handles CPCI device table

SYNTAX

```
pci  operation [para0 para1 para2 para3]
xpci operation [para0 para1 para2 para3]
cpci operation [para0 para1 para2 para3]
```

PARAMETERS

operation	CPCI operation					
	config					
	show	**dev**	*slot*	*bus*		
		pmc	*identifier*			
		idx	*index*			
		excl				
	excl	**dev**	*slot*	*bus*	*bar*	*addr*
		pmc	*identifier*	*bar*	*addr*	
		id	*vid*	*did*	*addr*	
		clear	*index*			

DESCRIPTION

The **pci**, **xpci** and **cpci** command allows to access the tables holding mapping parameters of devices located on local PCI, extended PCI and compact PCI buses. These tables are built during the diagnostic phase of the corresponding bus. They can be re-initialized at any time using the **diag** command.

The **config** operation displays the PCI interface controller identification and the parameters for the slave addresses.

The **show** operation display PCI device information. If no parameter is given, a table of all devices detected on the bus is displayed. Detailed information can be displayed for each device. The device is identified by its index in the table using **idx** as first parameter, followed by its index or by its geographical address on the bus using **dev** as first parameter, followed by its slot and bus number.

For the CPCI bus, if the system controller is a CES board (RIOC 406x), the device information is also available for non-system controllers.

If the address window allocated for mapping PCI devices is not big enough to hold all devices, the excl operation allows the user to exclude any device or function from the mapping process. The device to exclude can be identified either by its PCI geographical address (slot, func and bus) or by it logical identifier (vendor id and device id). The exclude table is stored in NVRAM. During the diagnostic phase, PPCMon checks that table before mapping PCI devices. To clear that table, use the **excl** operation with **clear all** as parameters.

EXAMPLES

Display the local PCI device table

```
PPC_Mon>pci show

+----+-----------------------------------------+
|              PCI DEVICES  [tree 1 ]           |
+----+------+------+----+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|ivec|pmc |
+----+------+------+----+----+----+----+----+----+
|  0 | 10d9 | 8064 | 00 | 00 | 00 |  0 | 00 |  x |
|  1 | 1022 | 2000 | 01 | 00 | 00 |  0 | 11 |  x |
|  2 | 10d9 | 844b | 02 | 00 | 00 |  0 | 16 |  1 |
|  3 | 1000 | 000b | 03 | 00 | 00 |  0 | 15 |  2 |
|  4 | 1000 | 000b | 03 | 01 | 00 |  0 | 15 |  2 |
|  5 | 1022 | 2000 | 04 | 00 | 00 |  0 | 19 |  3 |
+----+------+------+----+----+----+----+----+----+
PPC_Mon>
```

Display the mapping information of the device located in local PCI slot 3 PMC SCSI 8465).

```
PPC_Mon>pci show dev 3

Multi function device found in slot : 0x03   bus : 0x00

| func | vid  | did  | type |   addr   |   size   | space| bar  |
+------+------+------+------+----------+----------+------+------+
|  00  | 1000 | 000b |  D   | f6002000 | 00000100 | IO   |  0   |
                             | c0600000 | 00000400 | MEM  |  1   |
                             | c0700000 | 00002000 | MEM  |  3   |

|  01  | 1000 | 000b |  D   | f6003000 | 00000100 | IO   |  0   |
                             | c0800000 | 00000400 | MEM  |  1   |
                             | c0900000 | 00002000 | MEM  |  3   |

PPC_Mon>
```

Display the mapping information of the PMC #3 device (FETH 8478).

```
PPC_Mon>pci show pmc 3

Single function device found in slot : 0x04   bus : 0x00

| vid  | did  | type |   addr   |   size   | space| bar  |
+------+------+------+----------+----------+------+------+
| 1022 | 2000 |  D   | f6004000 | 00000020 | IO   |  0   |
                     | c0a00000 | 00000020 | MEM  |  1   |
                     | c0b00000 | 00100000 | PROM |

PPC_Mon>
```

Clear device exclusion table, and exclude PMC #1 bar 1 from mapping.

```
PPC_Mon>pci excl clear all
PPC_Mon>pci excl pmc 1 1
PPC_Mon>diag pci

entering PCI tree diagnostics

PCI interface [enabled]
   System Controller
   Slave       : A32 Size = 192 Mbytes
                 A32 Base = 0x00000000

scanning PCI bus 1 for devices:

+----+-----------------------------------------+
|              PCI DEVICES  [tree 1 ]           |
+----+------+------+----+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|ivec|pmc |
+----+------+------+----+----+----+----+----+----+
|  0 | 10d9 | 8064 | 00 | 00 | 00 |  0 | 00 |  x |
|  1 | 1022 | 2000 | 01 | 00 | 00 |  0 | 11 |  x |
|  2 | 10d9 | 844b | 02 | 00 | 00 |  0 | 16 |  1 |
|  3 | 1000 | 000b | 03 | 00 | 00 |  0 | 15 |  2 |
|  4 | 1000 | 000b | 03 | 01 | 00 |  0 | 15 |  2 |
|  5 | 1022 | 2000 | 04 | 00 | 00 |  0 | 19 |  3 |
+----+------+------+----+----+----+----+----+----+

PPC_Mon>pci show pmc 1

Single function device found in slot : 0x02   bus : 0x00

| vid  | did  | type |   addr   |   size   | space| bar  |
+------+------+------+----------+----------+------+------+
| 10d9 | 844b |  D   | d0000000 | 04000000 | PMEM |  0   |
                     | EXCLUDED | -------- | ---- |  1   |
                     | c0000000 | 00400000 | MEM  |  2   |

PPC_Mon>pci show excl

----+------+------+----+----+----+----+----------+
idx | vid  | did  |slot|func|bus |bar |   addr   |
----+------+------+----+----+----+----+----------+
  0 | any  | any  | 02 |all | 00 | 01 | 40000000 |
----+------+------+----+----+----+----+----------+

PPC_Mon>
```

On a RIOC 4065, display the CPCI interface parameters and show the device table.

```
PPC_Mon>cpci show

CPCI interface [disabled]
System Controller
Slave       : A32 Size = 32 Mbytes
              A32 Base = 0x40000000
              A64 Base = 0x00000000'00000000  [disabled]

+----+---------------------------------------+
|           PCI DEVICES  [tree 8 ]          |
+----+------+------+----+----+----+----+----+
|idx | vid  | did  |slot|func|bus |b_id|pmc |
+----+------+------+----+----+----+----+----+
|  0 | 10d9 | 4065 | 00 | 00 | 00 |  0 |  x |
|  1 | 10d9 | 4065 | 01 | 00 | 00 |  0 |  x |
|  2 | 10d9 | 4064 | 03 | 00 | 00 |  0 |  x |
+----+------+------+----+----+----+----+----+

PPC_Mon>
```

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*

pmc	**x**	**x**	**x**	**x**	**x**	

NAME

pmc - execute PMC operation

SYNTAX

pmc.*x operation*

PARAMETERS

x	PMC identifier
operation	sub-command
	on
	off
	reset

DESCRIPTION

On boards supporting PMC power control, the **pmc** command allows to power ON, power OFF or reset a PMC. These operation can be done automatically during the boot procedure when the local PCI is setup. The **pmc** environment parameters are used to define the power on sequence of each PMC.

EXAMPLES

.

SEE ALSO

diag, set

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
pr	**x**	**x**	**x**	**x**	**x**	

NAME

pr - patches register

SYNTAX

pr *reg idx [data]*

PARAMETERS

reg	register set	
	bcr	bridge control registers
	mmu	*PPC BAT and SR registers*
	hid0,hid1,hid2	*PPC HID registers*
iidx	register index	
data	data to store in register	[**0x..** hexa, **0d..** dec, **0o..** oct]

DESCRIPTION

pr must be used to display or modify the current values of PPC and PCI bridge registers. The *reg* parameters can be one of the following ASCII string:

bcr	IBM27-82660 PCI bridge control registers
mmu	PPC memory management unit registers

The *idx* parameter is the register index. The user must refer to the device user's manual for available registers. For **mmu** registers indices are of the form **ibatu0**, **ibatl0**,...**dbatu3**,**dbatl3**

If the parameter *data* is entered, the register is overwritten with *data*.

If no *data* parameter is given, the contents of the register pointed by *idx* is displayed and the firmware waits for a new data to overwrite the current register or for a control character:

Enter	the contents of next register is displayed
.	the firmware returns to the command interpreter
-	the contents of the previous register is displayed
=	the current register is read again and displayed
data	the current register is overwritten and the contents of the next register is read

EXAMPLES

Display the first four registers of the IBM27-82660 PCI bridge.

```
PPCMon>pr bcr 0
0x00 : 14 ->
0x01 : 10 ->
0x02 : 37 ->
0x03 : 00 ->
0x04 : 04 -> .
PPCMon>
```

Modify PPC750 HID1 to disable store gathering

```
PPC_Mon>pr hid0
hid0 : 8400c080 -> 8400c000
PPC_Mon>
```

Set PPC750 DBAT0 to access the system memory in write through mode

```
PPC_Mon>dr mmu

   0x00001fff   0x00000012   0x3ff0001f   0x07f00012
   0x10001fff   0x10000012   0x00000000   0x00000000
   0x00001fff   0x00000012   0x3ff0001f   0x07f00012
   0x80001fff   0x8000002a   0xf0001fff   0xf000002a
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000

PPC_Mon>pr mmu dbatl0 52
PPC_Mon>dr mmu

   0x00001fff   0x00000012   0x3ff0001f   0x07f00012
   0x10001fff   0x10000012   0x00000000   0x00000000
   0x00001fff   0x00000052   0x3ff0001f   0x07f00012
   0x80001fff   0x8000002a   0xf0001fff   0xf000002a
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000
   0x00800000   0x00800000   0x00800000   0x00800000

PPC_Mon>
```

SEE ALSO

dr

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
reset	x	x	x	x	x	

NAME

reset - reset hardware and reboot the processor

SYNTAX

```
reset
```

PARAMETERS

none

DESCRIPTION

The **reset** command sets the reset bit in the reset control register (LOC_CTL3) in order to activate the onboard central RESET logic.

All hardware logic is reset and the PowerPC starts executing from FPROM at address 0xfff00000 (FPROM offset 0).

During the boot phase, diagnostics of board resources are done and according to the boot environment parameter setting (see **set** and **setenv** commands), either the PPCMon command interpreter is activated or the automatic boot procedure is executed. Entering **SP** (space bar) during diagnostic execution or automatic boot count down allows entering directly the *PPCMon* command interpreter.

EXAMPLE

Reset a RIO 4068.

```
PPC_Mon>reset
MINI_Mon:booting PPC_Mon.......

PPC Boot Rev 5.20 created Tue Sep 21 10:29:46 2004

  Module Type: 8064AE  rev: B  serial: 186
  Host CPU: PPC750   ver: 83.02   speed: 450 Mhz
  PCI Bridge: CES XPC
  Memory Size: 256 + 256 = 512 Mbytes
  L2 CACHE:  1 Mbyte SSRAM   180 MHz
  FPROM: Two banks of AMD29F032 installed [16 Mbytes]

Entering boot diagnostics

 0    Check System Memory (0x00004000 - 0x1ff00000)          1  OK
 1    Check Interrupt Handler                                0  OK
 2    Check PCI Bridge                                       1  OK
 3    Check Cache and MMU                                    1  OK
 4    Check MK48T08 RTC and NVRAM                            1  OK
 5    Check SIC6351 Interrupt Controller                     1  OK
 6    Check Serial Lines                                     1  SKIPPED
 7    Check  Micro Timers                                    1  OK
 8    Check FIFO's (0 - 7)                                   1  OK
 9    Check Digital Thermometers                             1  OK
10    Check PCI devices                                      1  OK
11    Check On Board Ethernet Controller  (PCI slot 0)       1  OK
12    Check PMC #1 Extension  (PCI slot 2)                   1  NO DEVICE
13    Check PMC #2 Extension  (PCI slot 3)                   1  OK
14    Check PMC #3 Extension  (PCI slot 4 - agent 2)         0  NO DEVICE
15    Check PMC #4 Extension  (PCI slot 5 - agent 2)         0  NO DEVICE
16    Check VME Interface                                    1  OK
17    Check XPCI Interface                                   1  OK

***********************************************
*   PPC_Mon  RIO8064 monitor - version 5.2       *
*      CES SA Copyright 1995 - 2004              *
***********************************************

PPC_Mon>
```

Reset a RIOC 4070

```
PPC_Mon>reset
resetting RIO4070..MINI_Mon:booting PPC_Mon.......OK

PPC Boot Rev 5.20 created Tue Sep 21 10:29:46 2004

  Module Type: 4070AA  rev: A  serial: 2
  Host CPU: PPC750GX   ver: 1.01   speed: 500 Mhz
  PCI Bridge: CES XPC
  Memory Size: 256 Mbytes
  L2 CACHE:  1 Mbyte INTERNAL
  FPROM: One bank of AMD29LV256 installed [32 Mbytes]

Entering boot diagnostics

 0    Check System Memory (0x00004000 - 0x0ff00000)          0  OK
 1    Check Interrupt Handler                                0  OK
 2    Check PCI Bridge                                       0  OK
 3    Check Cache and MMU                                    0  OK
 4    Check RAMTRON RTC and FRAM                             0  OK
 5    Check SIC6351 Interrupt Controller                     0  OK
 6    Check Serial Lines                                     0  SKIPPED
 7    Check  Micro Timers                                    0  OK
 8    Check FIFO's (0 - 7)                                   0  OK
 9    Check Digital Thermometers                             0  OK
10    Check PCI devices                                      0  OK
11    Check On Board Ethernet Controller  (PCI slot 0)       0  OK
12    Check PMC #1 Extension  (PCI slot 2)                   0  OK
13    Check PMC #2 Extension  (PCI slot 3)                   0  OK
14    Check PMC #3 Extension  (PCI slot 4 - agent 2)         0  NO DEVICE
15    Check PMC #4 Extension  (PCI slot 5 - agent 2)         0  OK
16    Check PMC #5 (Secondary Ethernet controller)           0  OK
17    Check CPCI Interface                                   0  OK
18    Check XPCI Interface                                   1  OK

************************************************
*   PPC_Mon RIOC4070 monitor - version 5.2       *
*      CES SA Copyright 1995 - 2004              *
************************************************

PPC_Mon>
```

SEE ALSO

set, setenv

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
riomon	**x**	**x**	**x**	**x**	**x**	

NAME

riomon - returns to the RIO monitor (only available on RTPC 8067)

SYNTAX

riomon

PARAMETERS

none

DESCRIPTION

The PPC enters an infinite loop and control of the board is given back to the R3052 processor.

EXAMPLES

From *PPCMon* enter *RIOMon* command interpreter, display the command list by entering **?**, then using **ppc,** return to *PPCMon*.

```
PPCMon>riomon
RIO_Mon>?

Following commands are available :
---------------------------------

the strings wich appear beetween <> have to be entered by the user

bm.<op>    <0xsadd>..<0xdadd> <0xcount>  - Block move memory command
boot                                     - Boot host CPU
cm.<range> <0xadd1>..<0xadd2> <0xcount>  - Compare memory command
dm.<range> <0xadd>                       - Display memory command
fm.<range> <0xsadd>..<0xeadd> <0xdata>   - Fill memory command
go <0xaddress>                           - Go command
help                                     - Show command list
lf.<flash> <0xfoff>..<0xdoff> <0xbcount> - Load flash from sysram
mm.<range> <0xsadd>..<0xeadd> <0xdadd>   - Memory move command
pm.<range> <0xadd> <0xdata>              - Patch memory command (write)
.<range> <0xadd> <cr>                    - Patch memory command (read - write)
ppc                                      - Return to PPC monitor
sm.<range> <0xsadd>..<0xeadd> <0xdata>   - Search memory command
more (y/n) :y
setenv time                              - Set time environment
setenv vdbg                              - Set VME access environment
setenv temp                              - Set temperature limit environment
setenv test                              - Set test environment
setenv sign                              - Set signature environment
setenv pon                               - Set PowerON tests environment
setenv boot                              - Set boot mode for HOST CPU
st <start_test>..<end_test>              - Execution test command
sta                                      - Start table A tests list
stb                                      - Start table B tests list
terr                                     - Display error statistics
tlist                                    - Display the list of available tests
tm.<range> <0xsadd>..<0xeadd> <0xmask>   - Test memory command
tr.<range> <0xadd>                       - Touch read memory command
tw.<range> <0xadd> <0xdata>              - Touch write memory command
twr.<range> <0xadd> <0xdata>             - Touch write and read memory command
vpm.<range> <0xam> <0xadd> <0xdata>      - VME patch memory command (write)
vpm.<range> <0xam> <0xadd> <cr>       - VME patch memory command (read - write)
range = w : word access ; range = h : halfword access ; range = b : byte access
op = d : dual ; op = q : quad ; op = o : octal ; op = h : hex
In all loop modes type <space bar> to stop
RIO_Mon>ppc
PPCMon>
```

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
rmm	**x**	**x**	**x**	**x**	**x**	**x**
rmp			**x**	**x**	**x**	**x**
rmv			**x**	**x**		
rmx			**x**	**x**		

NAME

rmm, **rmp**, **rmv,rmx** - read modify write memory, PCI, VME or PCI2 (not supported on the RTPC 8067)

SYNTAX

rmm.*x addr [data]*
rmp.*x addr [data [space]]*
rmv.*x addr [data [am]]*
rmx.*x addr [data [space]]*

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
addr	address	[**0x..** hexa, **0d..** dec, **0o..** oct]
data	**?** or data to write	[**0x..** hexa, **0d..** dec, **0o..** oct]
space	PCI space	[**io**, **mem**]
amode	VME address moder	[from **0x00** to **0xff**]

DESCRIPTION

The **rmm** command must be used to read a single data from a CPU address *addr* and to replace it in the same bus cycle by *data* (read modify write atomic operation). The size of the data is given by *x,* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for

a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. *addr* is a CPU valid address (see CPU address mapping for VME or PCI accesses). The address is automatically adjusted to a multiple of the data size chosen to avoid address alignment errors.
rmp, **rmx** and **rmv** allows direct access to PCI, PCI2 and VME locations. To patch the contents of VME or PCI addresses, an additional parameter is needed, *am*ode is the VME address mode (from **0x00** to **0xff**) and *space* is the PCI space.
On RIO2 806x *amode* defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39).The default value is set to 0x9. On RIO3 8064, *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0.
The PCI space must be one of the following ASCII strings: **io** for PCI IO space, **mem** for PCI MEM space.
At each access, parameters are stored in a data structure, and when they are missing in the command line, the firmware takes them from that data structure.

EXAMPLES

Display the content of CPU address 0x800000 with the **pm** command. Execute a read modify write operation on the same address with data 0xdeadface and verify the content of the address.

```
PPC_Mon>pm.l 0x800000
0x00800000 : ffffffff ->
0x00800004 : fbfffffdf -> .
PPC_Mon>rmm.l 0x800000 0xdeadface
0x00800000 : ffffffff - deadface
PPC_Mon>
PPC_Mon>pm.l 0x800000
0x00800000 : deadface -> .
```

SEE ALSO

pm

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
rtc	**x**	**x**	**x**	**x**	**x**	

NAME

rtc - execute real-time clock operation

SYNTAX

`rtc` *`operation`*

PARAMETERS

x	mfcc identifier	
operation	sub-command	
	status	
	stop	*mode*

DESCRIPTION

The **rtc** command allows to stop the real time clock to save batteries. It can be restarted using the **date** command. The **status** operation displays the current state of the RTC.

EXAMPLES

Show date and RTC status, then stop the RTC. Show again status and date, then restart the RTC.

```
PPC_Mon>date
Tue Sep 21 15:31:48 2004
PPC_Mon>rtc status
RTC running
PPC_Mon>rtc stop
PPC_Mon>rtc status
RTC stopped
PPC_Mon>date
date out of range

PPC_Mon>date 0409211455
PPC_Mon>rtc status
RTC running
PPC_Mon>date
Tue Sep 21 14:55:06 2004
PPC_Mon>
```

SEE ALSO

date

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
sc			**x**		**x**	
sm	**x**	**x**	**x**	**x**	**x**	**x**
sp			**x**	**x**	**x**	**x**
sv			**x**	**x**		
sx			**x**	**x**		

NAME

sc, **sm**, **sp**, **sv, sx** - searches CPCI memory, PCI, VME or PCI2 for a given pattern

SYNTAX

```
sc.x start . . end data space
sm.x start . . end data
sp.x start . . end data space
sv.x start . . end data amode
sx.x start . . end data space
```

PARAMETERS

x	data size	[**l**, **w**, **s**, **h**, **b**, **c**]
start	start address	[**0x..** hexa, **0d..** dec, **0o..** oct]
end	end address	[**0x..** hexa, **0d..** dec, **0o..** oct]
data	data to write	[**0x..** hexa, **0d..** dec, **0o..** oct]
space	PCI space	[**io**, **mem**]
amode	VME address mode	[from **0x00** to **0xff**]

DESCRIPTION

The **sm** command reads the memory from address *start* to address *end* and displays the current address and data every time the data pointed by the current address is equal to *data*. Data accesses are done according to the data size specified in *x,* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. To display the contents of VME, CPCI, PCI or PCI2 addresses, an additional parameter is needed, *amode* is the VME address mode (from **0x00** to **0xff**), space is the PCI / CPCI space. On RIO2 806x, *amode* defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39).The default value is set to 0x9. On RIO3 8064 *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0. The PCI space must be one of the following ASCII strings: **io** for PCI IO space, **mem** for PCI memory space. Command parameters are stored in a data structure, and when they are missing, the firmware takes them from that data structure.

EXAMPLES

Fill one block of memory (0x80'0000 - 0x80'0080) with data 0x1234'5678, and modify two locations (0x80'0034 and 0x80'006C) with another data (0xdeadface). Then search the block for the modified data.

```
PPCMon>fm.l 800000..800080 12345678
PPCMon>pm.w 800034 deadface
PPCMon>pm.w 80006c deadface
PPCMon>sm.w 800000..800080 deadface
0x00800034 : deadface
0x0080006c : deadface
PPCMon>
```

SEE ALSO

fm, pm

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
sdcard	**x**	**x**	**x**	**x**	**x**	

NAME

sdcard - execute SDCARD related operation

SYNTAX

sdcard *operation*

PARAMETERS

operation	sub-command	
	init	
	read	*mode*
	erase	erases an SDCARD address range
	write	copies a buffer from the memory to the SDCARD
	read	copies a buffer from SDCARD to the memory
	effs	execute file system functions
paraX	parameters depending on the operation selected	

DESCRIPTION

On board equipped with SDcard controller, the **sdcard** command allows to read/write/erase raw data and to manage a FAT file system

The **init** operation performs the SDcard initialization sequence. It doesn't take any parameter.

If a fileFAT file system is present on the SDcard, **sdcard** can be used as device name for the **load** and **boot** commands.

The **erase** operation has the following syntax:

sdcard erase *start .. end*

start	SDCARD offset of the first location to erase [**0x..** hex, **0d..** dec, **0o.**. oct]
end	SDCARD offset of the last location to erase [**0x..** hex, **0d..** dec, **0o..** oct]

The **read** operation has the following syntax:

sdcard read *start addr size*

start	SDCARD offset from where to read data [**0x..** hex, **0d..** dec, **0o..** oct]
addr	address where to copy data [**0x..** hex, **0d..** dec, **0o..** oct]
size	size in byte of the data to transfer [**0x..** hex, **0d..** dec, **0o..** oct]

The **write** operation has the following syntax:

sdcard write *start addr size*

start	SDCARD offset where to write data [**0x..** hex, **0d..** dec, **0o..** oct]
addr	address from where to read data [**0x..** hex, **0d..** dec, **0o..** oct]
size	size in byte of the data to transfer [**0x..** hex, **0d..** dec, **0o..** oct]

The **effs** subcommand allows to perform file system related functions

sdcard effs *function*

funtion	function to be performed	
	init	initialize file system
	df	display file system space usage
	ls	list directory content
	cd	change directory
	pwd	print name of current directory

EXAMPLES

After having inserted an SDcard formatted on a PC, lets read the first block (offset 0).

```
PPC_Mon>sdcard read 0 100000 1000
sdcard_off = 0   mem_addr = 100000  len = 1000
reading : 00001000
PPC_Mon>dm.l 100000
0x00100000 : eb3c904d 53444f53 352e3000 02100100  .<.MSDOS5.0.....
0x00100010 : 02000200 00f8f200 3f00ff00 00000000  ........?.......
0x00100020 : 001f0f00 00002973 5ec4f24e 4f204e41  ......)s^..NO NA
0x00100030 : 4d452020 20204641 54313620 202033c9  ME    FAT16   3.
0x00100040 : 8ed1bcf0 7b8ed9b8 00208ec0 fcbd007c  ....{.... .....|
0x00100050 : 384e247d 248bc199 e83c0172 1c83eb3a  8N$}$....<.r...:
0x00100060 : 66a11c7c 26663b07 268a57fc 750680ca  f..|&f;.&.W.u...
0x00100070 : 02885602 80c31073 eb33c98a 461098f7  ..V....s.3..F...
PPC_Mon>
```

Initialize filesystem, show space usage and list root directory content

```
PPC_Mon>sdcard effs init
sdcard: flash file system commands
initialize sd card...
File System Version: HCC_FAT_LFN ver:2.55
Space: Total = 1e39e000  Free = 1e2ba000  Used = e4000
PPC_Mon>sdcard effs df
sdcard: flash file system commands
Filesystem    512-blocks        Used       Available      Use%    Volume
sdcard           990448         1824        988624       0%     RIO4
PPC_Mon>sdcard effs ls
sdcard: flash file system commands
ppcmon/
kernel/
applications/
fpga/
PPC_Mon>
```

In this example a copy of PPC_Mon has been copied in the directory /ppcom. The filename is 4070prom.bin. The following sequence checks that the fie is present in the file system an load it in memory

```
PPC_Mon>sdcard effs cd ppcmon
sdcard: flash file system commands
PPC_Mon>sdcard effs pwd
sdcard: flash file system commands
/ppcmon
PPC_Mon>sdcard effs ls
sdcard: flash file system commands
./
../
4070prom.bin
PPC_Mon>load sdcard 4070prom.bin 100000 binary

Trying device  sdcard   [file : 4070prom.bin] ...
loading from SD Card -> initialize sd card...
error loading file
PPC_Mon>load sdcard /ppcmon/4070prom.bin 100000 binary

Trying device  sdcard   [file : /ppcmon/4070prom.bin] ...
loading from SD Card -> initialize sd card...
file length: 0x000c0000
PPC_Mon>dm.l 100000
0x00100000 : 43455320 50504342 4f4f5420 352e3230  CES PPCBOOT 5.20
0x00100010 : aaaaaaaa 55555555 52494f20 34303730  ....UUUURIO 4070
0x00100020 : 48000000 48000000 48000000 48000000  H...H...H...H...
0x00100030 : 61626364 65666768 00000000 00000000  abcdefgh........
0x00100040 : f900c000 f9008000 f9000000 f9000600  ................
0x00100050 : fa000000 fa100000 f8000000 f8000000  ................
0x00100060 : f9004000 ffffffff fa108000 f9009300  ..@.............
0x00100070 : f9009200 70000000 40000000 7f000000  ....p...@.......
PPC_Mon>
```

SEE ALSO

load

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
set	x	x	x	x	x	x
show	x	x	x	x	x	x

NAME

set - sets the environment parameters

SYNTAX

set *[para [value]]*
show *[para]*

PARAMETERS

para	environment parameter name
value	value of the parameter

DESCRIPTION

The **set** command sets the environment parameter values. If no *value* is given, it displays the environment parameter's names, allowed values (in brackets), current value and waits for a new value to be overwritten. Entering **CR** leaves the current value unmodified.

The **show** command displays the environment parameter's names, allowed values (in brackets) and current value.

Parameter names are made of sub-names linked by the _ character. The first sub-name indicates the group to which the parameter belongs. A second sub-name (optional) indicates the sub-group. The **set** and **show** commands accept group names. In this case, the **set** command displays sequentially all parameters belonging to the group to be modified. When the **set** command is competed, parameters are updated in the NVRAM. The **show** command displays all parameters belonging to the group or the sub-group.

The **boot** group environment parameters contains information needed by the monitor to boot an operating system (VxWorks, LynxOS, Linux, etc.) or user applications. The **boot_flags**, **boot_rootfs**, **boot_size** and **boot_line** parameters are for use by the operating system. The **boot_device**, **boot_filename**, **boot_address**, **boot_filetype**, **boot_delay** and **boot_mode** parameters are used by the **boot** and **load** commands. If **boot_fast** is set to **y**, the display of diagnostics messages is skipped. If the diagnostic tests fail, the status is displayed (see **status** command) and the command interpreter is launched. If **boot_fast** is set to **n**, a message is printed for each diagnostic test. The diagnostic procedure can be aborted by pressing [**space**]. If no errors are encountered during the diagnostic tests and the **boot_mode** parameter is set to **auto**, the boot command is automatically executed, after waiting for **boot_delay** seconds. This countdown can be interrupted by pressing [**space**] (if **boot_delay** is **0**, the command interpreter is launched instead). The **boot_device** and **boot_filename** parameters support having up to four options separated by **|**. When booting the interpreter will use the first set of parameters, and if that fails it will try the second, third and fourth device.

Group	*Sub-Group*	*Parameter*	*Description*	*Default*
Boot		flags	Flags used by OS for booting	S
		device	Device where to get the boot code	
		filename	File containing the boot code	
		rootfs	Device used by the OS for its file system	
		delay	Delay in seconds before booting	0
		address	Address used by the monitor to copy files during loading	0
		size	Memory size allocated to the OS	0
		fast	If set use fast boot mode	n
		filetype	Type of the file to boot	binary
		cpunr	CPU number	0
		mode	Action to take after diagnostic execution	ppcmon
		line	128 character string used by OS	

The **inet** group environment parameters contain information related to internet access over ethernet. The **inet_host, inet_bootserver** and **inet_protocol** parameters are used by the **boot** and **load** commands if the ethernet controller has been chosen as **boot_device**. Other parameters from that group are for use by the operating system. Setting these parameters takes effect immediately.

Group	*Sub-Group*	*Parameter*	*Description*	*Default*
inet		host	Internet address of host (the board running PPCMon)	0.0.0.0
		bplane	Internet address of host on backplane network	0.0.0.0
		bootserver	Internet address of server used to boot from	0.0.0.0
		gateway	Internet address of gateway	0.0.0.0
		nameserver	Internet of name server	0.0.0.0
		protocol	Address resolution protocol	arpa
		mount	Name of the remote file system to be mounted by OS	
		ethermode	Ethernet speed (10Mbit, 100Mbit, 1Gbit,autonegociation)	auto

The **vme** group environment parameter contains information used by the firmware to initialize the VME interface when the corresponding diagnostic is executed. For RIO2 (8060, 8061, 8062 and 8067), it is divided in four sub-groups. The **vme_arb** sub-group is related to the VME arbitration function of the board (active only if slot 1). The **vme_req** sub-group allows to define how the board will request access to the VME bus (bus master function). The **vme_slv** sub-group defines the address windows to be used to access the board internal resources from VME (bus slave function). The **vme_bma** sub-group contains parameters used by the **bma** command. Setting these parameters takes effect immediately.

Group	Sub-Group	Parameter	Description	Default
vme	arb	mode	VME arbitration mode	prio
		bto	VME arbitration time out	16
	req	mode	VME request mode	rwd
		hidden	Enable hidden mode	n
		level	VME bus request level	3
		lto	Local time out	32
		retry	Enable bus retry	n
	slv	a24base	VME base address for a24 slave access	0.d
		a32base	VME base address for a32 slave access	0.d
		a32size	Address window size for a32	16
		latency	Local PCI latency for slave access	127
	bma	swap	Data swapping mode for bma master access	noswap
		am	Address modifier for bma master access	a24s
		dsz	Data size for bma master access	d32
		vbsz	VME block size for bma master access	0
		pbsz	PCI block size for bma master access	0
		inc	VME address auto increment for bma master access	y
		space	PCI space for bma master access	pio

For RIO3 8064, it is divided in six sub-groups. The **vme_arb** sub-group is related to the VME arbitration function of the board (active only if slot 1). The **vme_req** sub-group allows to define how the board will request access to the VME bus (bus master function). The **vme_slv** sub-group defines the address windows to be used to access the board internal resources from VME (bus slave function). The **vme_64x** sub-group is used to set up the VME64X and VME64X/LI specific features. The **vme_intr** sub-group is related to **vme** command. The **vme_bma** sub-group contains parameters used by the **bma** command. Setting these parameters takes effect immediately.

Group	Sub-Group	Parameter	Description	Default
vme	arb	mode	VME arbitration mode	prio
		bto	VME arbitration time out	16
	req	mode	VME request mode	rwd
		level	VME bus request level	3
	slv	size	Address window size for a32	32
		a32	VME base address for a32 slave access	0
		a64	VME base address for a64 slave access	0
		a32bc	VME base address for a32 broadcast slave access	0
		a64bc	VME base address for a64 broadcast slave access	0
		enable	Enable slave access	n
		prefetch	Block size of read prefetch	16
		retry	Enable VME retry	y
	64x	2efast	Enable 2eVME	n
		broadcast	Set slave broadcast mode	no_brc
		rcfilter	RC filter on DS and DTACK	n
		uncpld	Set read uncompelled	n
		masli	VME master interface enable	y
	intr	vector	Interrupt vector	0
		level	Interrupt level	0
	bma	swap	Data swapping mode for bma master access	noswap
		am	Address modifier for bma master access	a24s
		dsz	Data size for bma master access	d32
		vbsz	VME block size for bma master access	0
		pbsz	PCI block size for bma master access	0
		inc	VME address auto increment for bma master access	y
		space	PCI space for bma master access	pio

The **cpci** group environment parameters contains information used by the firmware to initialize the RIOC 4063 / 4064 CPCI interface (PLX PCI 9080) when the corresponding diagnostic is executed. Three CPCI address windows can be used to access the RIOC 406x internal resources. For each window, 4 parameters must be set before using the CPCI slave interface. The access mode from the CPCI bus is defined by the CPCI address and address space. The CPCI address is set dynamically by the system controller when it configures all devices on the CPCI bus. The address *space* is defined by the slave itself (the RIOC 406x) in the **space** environment parameter. The **addr** and **mode** parameters define the window target address on the boards local PCI bus. If **mem** mode is selected, the window points to the system memory at address offset addr. For the xprom window, local addressing mode is always **mem**. The **size** parameters set the size in Bytes of the corresponding window. Setting these parameters takes effect immediately.

Group	*Sub-Group*	*Parameter*	*Description*	*Default*
cpci	slv0	addr	Local PCI base address pointed by CPCI slave window 0	0
		size	CPCI slave window 0 size	1000
		space	CPCI space to be used to access slave window 0	
		mode	Access mode on local CPCI slave window 0	
	slv1	addr	Local PCI base address pointed by CPCI slave window 1	0
		size	CPCI slave window 01 size	1000
		space	CPCI space to be used to access slave window 1	
		mode	Access mode on local PCI for window 1 slave access	
	prom	addr	Local PCI base address pointed by CPCI xprom window	0
		size	CPCI xprom window 0 size	1000
		space	CPCI space to be used to access xprom window	mem
		latency	Local PCI latency for slave access	127
		delay	Delay before intialization of slave devices	0

The **diag** group environment parameters allow the user to set the level at which each diagnostic is executed at boot time (see **diag** command). If a parameter is set to **y**, the corresponding diagnostic will be executed at level 1, otherwise (**n**) it will be executed at level 0. Setting these parameters takes effect after reboot.

Group	*Sub-Group*	*Parameter*	*Description*	*Default*
diag		sysmem	System Memory	n
		mmu	Cache and MMU	n
		intctl	SIC6351 Interrupt Controller	n
		serial	PC87312 Super IO	n
		timers	ZCIO8536 Micro Timers	n
		fifos	FIFO's (0-7)	n
		thermo	Digital Thermometers	n
		pci	PCI devices	n
		ethernet	AM79C970 Ethernet Controller	n
		scsi	NCR53C825 SCSI Controller (RTPC8067)	n
		pmc	PMC Extension (RTPC8067)	n
		pmc1	PMC #1 Extension (RIO806x)	n
		pmc2	PMC #2 Extension (RIO806x)	n
		vme	VME Interface (RTPC8067 and RIO806x)	n
		cpci	CPCI Interface (RIOC406x)	n
		xpci	Extended PCI (RIO4065 &RIO8064)	n

The **bridge** group environment parameters allows to set-up the CPU-to-PCI and CPU to memory bridges.

The **ecc** and **pci** sub-groups are related to the IBM27-82660 bridge present on RTPC 8067, RIO 8060, RIO 8061, RIO2 8062, RIOC 4063 and RIOC 4064. The **bridge_ecc_count** determines the number of ECC error corrections allowed before an error status is generated by the bridge. If **bridge_pci_serr** is set to **y**, PCI_SERR will be asserted by the bridge if a system error is detected and the bridge is PCI target. If **bridge_pci_parity** is set to y, detection of parity error will be reported to the CPU using MCP (bus error), if bridge is master. When **bridge_pci_watchdog** is set to **y**, if a PCI master cycle has not terminated 2000 clock cycles after assertion of DEV_SEL, the cycle is master aborted. The **bridge_pci_disc** counter determines the maximum number of PCI clock (from 1 to 255) that a PCI master can burst access system more before a target disconnect is initiated. Setting it to 0, disables that mechanism.

The **lpci**, **xpci** and **cpci** sub-groups are related to the CES XPC_PCI bridges controlling the local PCI, extended PCI and CompactPCI buses on RIOC 4065 and RIO3 8064. The **ena** parameters allows the user to enable access to the corresponding bus (local PCI is always enabled).

Setting these parameters takes effect immediately.

Group	*Sub-Group*	*Parameter*	*Description*	*Default*
bridge	ecc	count	Generate error after count ECC errors	
	pci	serr	Enable PCI_SERR assertion	
		parity	Enable PCIPARITY recognition	
		watchdog	Enable watchdog timer	
		disc	Number of PCI clock cycles before disconnects the PCI bus master	
	lpci	s64	Enable 64 bit backplane	y
		prefetch	Size of read prefetch	8
		flush	Flush write cycles before read	n
		xpcarb	XPC arbitration mode	0
		pciarb	PCI arbitration mode	0
		mretry	Enable master retry	n
		timer	Read slave discard timer	2.00E+016
		parity	Report PCI parity to XPC	0
		rcmpld	Enable read compelled	n
		2pmc	Enable dual PMC handling	y
	xpci	ena	Enable interface	n
		s64	Enable 64 bit backplane	y
		size	Size of slave window (Mbytes)	32
		prefetch	Size of read prefetch	8
		flush	Flush write cycles before read	n
		mretry	Enable master retry	n
		sretry	Enable retry on slave configuration cycles	n
		timer	Read slave discard timer	2.00E+016
		parity	Report PCI parity to XPC	0
		rcmpld	Enable read compelled	n

Group	*Sub-Group*	*Parameter*	*Description*	*Default*
bridge	cpci	ena	Enable interface	n
		s64	Enable 64 bit backplane	y
		a64	Size of slave window	n
		size	Size of slave window	32
		prefetch	Size of read prefetch	8
		flush	Flush write cycles before read	n
		pciarb	PCI arbitration mode	0
		mretry	Enable master retry	n
		timer	Read slave discard timer	2.00E+016
		parity	Report PCI parity to XPC	n
		rcmpld	Enable read compelled	n
		enum	Mode for ENUM interrupt	switch

The **pmc** group environment parameters is used to control PMCs dynamic power on (on boards supporting it). If **power** is set to **off**, the PMC is not powered ON during the boot procedure. To power it ON in the pci diagnostic a time-out value (**10ms**, **100ms** or **1s**) shall be selected. This will define how long PPC_Mon will wait after the power on until it tries to initialize the PMC configuration registers.

Group	Sub-Group	Parameter	Description	Default
pmc	1	power	power mode	off
	2	power	power mode	off

The **intr** group environment parameters determines how the firmware interrupt handler will report when an exception occurs. If **intr_mask_err** is set to **y**, exception reporting is disabled. In case of MCP exception (bus error), the content of all processor registers will be displayed if **intr_debug** is set to **y**. Setting these parameters takes effect immediately

Group	Sub-Group	Parameter	Description	Default
intr	mask	err	Mask messages from interrupt handler	
		debug	Display processor registers if bus error	

The **cache** group environment parameters allows to enable / disable primary instruction and data caches as well as level 2 cache if present during the boot phase. The **cache** command is used to control the cache status. If the boot is interrupted during the diagnostic phase, caches will stay disabled. If **cache_l2_parity** is set to y, a parity error detected while reading from L2 cache will be reported to the CPU by MCP. **cache_l2_do** and **cache_l2_wt** are only valid for the PPC750 L2 cache controller and allow to set the corresponding bit in the L2 cache control register (see MPC750 User's Manual). For processors like PPC745x with L3 cache controller, the **cache_l3_xxx** parameters shall be used to set it up.

Setting these parameters takes effect after reboot.

Group	Sub-Group	Parameter	Description	Default
cache	i	enable	Enable instruction cache	y
	d	enable	Enable data cache	y
	l2	enable	Enable l2 cache	y
		parity	Enable parity checking on l2 cache	n
		do	Data only (for PPC750)	n
		wt	Write through (for PPC750)	n
	l3	ena	Enable L3 cache	y
		parity	Enable parity checking on l3 cache	n
		do	Data Only	n
		io	instruction Only	n
		param	l3 speed parameter	12

The **dma** group environment parameters allows to set up the mapping of up to four PCI DMA controllers used by the **dma** command to transfer block of data over local PCI, extended PCI or compact PCI buses.

Group	Sub-Group	Parameter	Description	Default
dma	chan0	slot	PCI slot of DMA controller	2
		func	PCI function of DMA controller	0
		bus	PCI bus of DMA controller	0
		script	Address of DMA controller scripts	0
	chan1	slot	PCI slot of DMA controller	3
		func	PCI function of DMA controller	0
		bus	PCI bus of DMA controller	0
		script	Address of DMA controller scripts	0
	chan2	slot	PCI slot of DMA controller	2
		func	PCI function of DMA controller	1
		bus	PCI bus of DMA controller	0
		script	Address of DMA controller scripts	0
	chan3	slot	PCI slot of DMA controller	3
		func	PCI function of DMA controller	1
		bus	PCI bus of DMA controller	0
		script	Address of DMA controller scripts	0

The **cpu** group environment parameters allows to handle parity errors. That information is used after reset at the very beginning of the CPU boot phase.

Group	Sub-Group	Parameter	Description	Default
cpu	parity	ena	Enable XPC parity generation	n
		gen	Enable CPU parity generation	n
		check	Enable CPU parity checking	n

The **temp** group environment parameters allows control of the board temperature. Up to three thermometers are located on the board. The first one monitors the CPU temperature. It is read every minute and if temperature goes over the upper high limit, it either issues a warning or stops the CPU, according to the control mode selected. On RIO3 8064 or RIOC 4065 boards, two additional thermometers are installed, one under the system RAM piggy pack and the other under the PMC 2 PCI slot.

Group	Sub-Group	Parameter	Description	Default
temp	cpu	low	Lower limit	55
		high	Upper limit	65
		ctl	Control mode	warn
	sdram	low	Lower limit	55
		high	Upper limit	65
		ctl	Control mode	warn
	pmc2	low	Lower limit	55
		high	Upper limit	65
		ctl	Control mode	warn

The test group environment parameters allows to set up the CES onboard test environment.

Group	Sub-Group	Parameter	Description	Default
test		debug	Print error messages	y
		loop	Number of time to execute test	1
		error	Action to take on error	stop
		abort	Action to take on abort	
		ini	Execute start-up test at initialization of test environment	n
		start	Index of start-up test to execute at initialization time	0

EXAMPLES

Set the value of the parameters belonging to the **boot** group. **boot_device** and **boot_filetype**, other parameters are not changed.

```
PPC_Mon>set boot

BOOT parameters

  boot_flags [afmnNS] : S ->
  boot_device [s<1-2>d<0-7><a-d>, le, e<0-2>, atm<1-6>@<vci>, tty<0-1>,
  fp] : le -> e2
  boot_filename : vxrio3wg0.st ->
  boot_rootfs [s<1-2>d<0-7><a-d>, rd] : rd ->
  boot_delay : 1 sec ->
  boot_address : 1000000 ->
  boot_size : 0 ->
  boot_fast [ y n ] : n ->
  boot_filetype [ binary auto lynx prep srec ] : auto -> lynx
  boot_cpunr : 31 ->
  boot_mode [ ppcmon test auto user ] : ppcmon ->

  PPC_Mon>
```

Displays the current value of the **inet_bootserver** parameter, change it to 144.0.10.7 and display all parameters from the **inet** group.

```
PPCMon>show inet_bootserver
   inet_bootserver [dotted decimal] : 144.0.10.3
PPCMon>set inet_bootserver 144.0.10.7
PPCMon>show inet

INTERNET parameters

   inet_host [dotted decimal] : 144.0.10.5
   inet_bootserver [dotted decimal] : 144.0.10.7
   inet_gateway [dotted decimal] : 144.0.4.200
   inet_nameserver [dotted decimal] : 144.0.6.15
   inet_protocol [ arpa bootp ] : arpa
   inet_mount : perly.ces.ch:/

PPCMon>
```

On a RIOC 4064, display the current value of the CPCI environment parameters. The CPCI slave interface allows the user to set up two windows to access the boards internal resources, plus another one to access an expansion PROM foreseen to contain the board initialization procedure. The size of each of these windows and the local PCI addresses pointed by the corresponding CPCI base address registers, are set by the CPCI environment parameters. In this example, the slave window 0 points to the PCI address (**cpci_slv0_addr**) 0x1400000 (board local resources) in the IO space (**cpci_slv0_mode**), has a size of 64 KBytes (**cpci_slv0_size**) and can be accessed from the CPCI IO space (**cpci_slv0_space**). Actually the setting of that window is chosen to allow the access of the RIOC 406x FIFO starting at offset 0xe000 (see examples for the **pc** command). The slave window 1 points to the system memory (**cpci_slv1_addr** and **cpci_slv0_mode**), has a size of 16 MBytes (**cpci_slv1_size**) and is accessed from CPCI MEM space (**cpci_slv1_space**). Since the RIOC 406x has no expansion prom, the corresponding window points to the RIOC 406x SRAM (**cpci_prom_addr**), has a size of 128 KBytes (**cpci_prom_size**) and can be accessed from the CPCI MEM space (**cpci_prom_space**). The CPCI base addresses of these windows are set dynamically by the CPCI system controller during its boot phase, if the CPCI diagnostic was set, and every time that diagnostic is executed by hand.

```
PPCMon>show cpci

CPCI parameters

   CPCI slave window 0
   cpci_slv0_addr : 1400000
   cpci_slv0_size : 10000
   cpci_slv0_space [ io mem ] : io
   cpci_slv0_mode [ io mem ] : io
   cpci_slv1_addr : 0
   cpci_slv1_size : 1000000
   cpci_slv1_space [ io mem ] : mem
   cpci_slv1_mode [ io mem ] : mem
   cpci_prom_addr : 3f000000
   cpci_prom_size : 20000
   cpci_prom_space [ io mem ] : mem
   cpci_latency : 127 clock cycles
   cpci_delay : 0 sec
PPCMon>
```

SEE ALSO

bma, **boot**, **cache**, **diag**, **load**

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
setenv	**X**	**X**	**X**	**X**	**X**	

NAME

setenv - sets the board configuration parameters

SYNTAX

setenv *para*

PARAMETERS

para the following parameters are accepted

boot	boot mode
hwconf	RAM parameters
sign	board signature
temp	temperature limits

DESCRIPTION

The **setenv boot** command allows the use to define the boot sequence mode. The 8-bit mode field is divided in 2 as follows:

bit <02:00>RTPC RIO related options (Not significant on the RIO2 806x)
= [0] -> Stays at RIOMON
= [1] -> Jumps to RIO SMON (Factory Test Utility)

bit <03> RTPC RIO related options (Not significant on the RIO2 806x)
= [0] -> boot PPCMon from FPROM offset 0x2000
= [1] -> boot alternate firmware

bit <07:04> PowerPC related options
= [0] -> Does not boot PowerPC
= [1] -> Boots PowerPC & stays at PPCMon
= [2] -> Boots PowerPC & jumps to PPCTstMon (Factory Test Utility)
= [3] -> Boots PowerPC & jumps to PPC SMON (Factory Test Utility)
= [4] -> Boots PowerPC & jumps to PPC STA (Factory Test Utility)
= [5] -> Boots PowerPC & continues PowerPC boot defined by set boot
= [7] -> Boots PowerPC, executes user code & continues the boot

The **setenv sign** command allows the user to load the factory board signature. This command is protected with a password and is reserved for factory updates and field services.

The **setenv hwconf** command allows the user to update the hardware configuration board signature. This command should be used carefully to define the following parameters:

System Memory Configuration
Magic Code
Ethernet Physical Address

The system memory related options are encoded as follows:

SYS DRAM BKx Size: This 1 Byte field specifies the DRAM size used for Bank 0.
[00] No memory installed
[01] 1M x 64. (with 1M x 4 DRAM devices)
[02] 2M x 64. (with 2M x 8 DRAM devices)
[04] 4M x 64. (with 4M x 4 DRAM devices)
[11] EDO 1M x 64. (with 1M x 4 DRAM devices)
[12] EDO 2M x 64. (with 2M x 8 DRAM devices)
[14] EDO 4M x 64. (with 4M x 4 DRAM devices)

SYS DRAM BKx Type: This 1 Byte field specifies the DRAM access time
[00] 60 ns No ECC
[01] 70 ns No ECC
[02] 80 ns No ECC
[80] 60 ns ECC
[81] 70 ns ECC
[82] 80 ns ECC

The **setenv temp** command allows the user to load the thermostat limits in the DS1620. These limits will control the front panel over temperature LED. The limits are defined in centigrade degrees.

If the control flag is set to "1", the processor will be stopped if the temperature goes over the high limit.

If the flag is set to "0", a warning is displayed.

EXAMPLES

Set boot mode to automatically launch code defined in boot environment parameters after diagnostic execution.

```
PPCMon>setenv boot
Host boot mode [00] : 50
PPCMon>
```

Show configuration of system memory and ethernet physical address.

```
PPCMon>setenv hwconf
SYS DRAM BK0 Size         [04] :
SYS DRAM BK0 Type         [80] :
SYS DRAM BK1 Size         [00] :
SYS DRAM BK1 Type         [00] :
SYS DRAM BK2 Size         [00] :
SYS DRAM BK2 Type         [00] :
SYS DRAM BK3 Size         [00] :
SYS DRAM BK3 Type         [00] :
Magic Number            [6e3c] :
Ethernet Address [00.80.a2.00.c0.90] :
update EEPROM signature
checksum = 0x38dc

!! To use new configuration you must reset the board !!
   If the board doesn't reboot set boot switch to 'manual'
   to use default signature
PPCMon>
```

Show temperature and alarm limits, then set limit high to 70 degrees.

```
PPCMon>temp
CPU temperature = 54 degrees [60-65]
PPCMon>setenv temp
Temp Limit High [65] : 70
Temp Limit Low [60] :
Temp Limit Control [1] :
PPCMon>
```

SEE ALSO

config, **reset**, **temp**

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
sflash	**X**	**X**		**X**	**X**	

NAME

sflash - executes SFLASH operations

SYNTAX

sflash *cmd [para0] [para1]...*

PARAMETERS

para	the following parameters are accepted		
	sign		
	fpga		
	check		
	read	*addrlen*	
	load	*fpga #y dev*	*fileaddr*

DESCRIPTION

The SFLASH contains all of the files loaded into FPGA at Power-ON. Each file is recognized by a signature containing: file name, creation date, offset in SFLASH, length and a checksum calculated at the loading time. The **sign** operation by the parameter **fpga** lists the signatures of FPGA files currently loaded.

The **sflash** subcommand by the parameter **check** recalculates dynamically FPGA files parameters and compares the result with the signature. This allows the user to check SFLASH integrity.

The **read** operation transfers data from SFLASH, starting at offset off to memory at address **addr**. **len** is the number of Bytes transferred.

The **sflash** sub-command allows, using the **load** operation, reloading over Ethernet (**le**) **SFLASH** with a new FPGA file. **y** is the FPGA identifier, it is from 1 to 6.

FPGA tools provide different file formats: PPCMon support “ttf” and “mcs” formats. FPGAs can be loaded all together at the same time, based on the hardware programmation sequence or only one identified by the FPGA identifier **y**.

addr is an address in memory used for temporary storage during the loading phase. For safety a password and a control key is required. In fact, passing a wrong file by the command **sflash load** can make the board unusable. In this case the FPGAs must be reprogrammed by the CISP tool.

EXAMPLES

On a RIOC 4065 list the files loaded in FPGAs.

```
PC_Mon>sflash sign fpga
FPGA#1 signature
  R465_128.mcs
  Start 0x0100000
  Size 0x00011114
  Pgm 29-08-2001
  CHSM 0xcb163152

  RELEASE_2001-05-22

FPGA#2 signature
  R465_128.mcs
  Start 0x0111114
  Size 0x0001111c
  Pgm 29-08-2001
  CHSM 0x58b02dc2

  RELEASE_2001-05-22

FPGA#3 signature
  R465_128.mcs
  Start 0x0122230
  Size 0x0001fbec
  Pgm 29-08-2001
  CHSM 0x2af26251

  RELEASE_2001-05-22

FPGA#4 signature
  R465_128.mcs
  Start 0x0141e1c
  Size 0x0001fbec
  Pgm 29-08-2001
  CHSM 0x213393e4

  RELEASE_2001-05-22

FPGA#5 signature
  R465_128.mcs
  Start 0x0161a08
  Size 0x00028c54
  Pgm 29-08-2001
  CHSM 0x1234df00

  RELEASE_2001-05-22

FPGA#6 signature
  R465_128.mcs
  Start 0x018a65c
  Size 0x00028c50
  Pgm 29-08-2001
  CHSM 0xe1381486

  RELEASE_2001-05-22
```

```
PC_Mon>sflash check
fpga_off = 100000 fpga_len = b32b4
reading sflash:001b32b4
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  | 100000 | 11114 | cb163152 | OK
|  2  | 111114 | 1111c | 58b02dc2 | OK
|  3  | 122230 | 1fbec | 2af2251 | OK
|  4  | 141e1c | 1fbec | 213393e4 | OK
|  5  | 161a08 | 28c54 | 1234df00 | OK
|  6  | 18a65c | 28c50 | e1381486 | OK
```

```
PPC_Mon>sflash read 40000 300000 1000
sflash_off = 40000   mem_addr = 300000  len = 1000
reading : 00041000
```

```
 PPC_Mon>sflash load fpga le R465_128.mcs 200000
ethernet load
local address = 00:a2:80:01:40:87
using ARPA
...file server address 08:00:20:c3:0b:d1
loading file 'R465_128.mcs' from server 10.0.6.15 at address 0x200000
tftp packet number:      fc1
transfer rate: 989 kbyte/sec [len = 0x1f7f60]
sflash_id = 1000000ger
loading fpga file from mem_addr =  0x200000  [len =  0x1f7f60]
converting file...[len = 0xb32ac]

enter control key  -> 297d176d

FPGA file R465_128.mcs contains 6 sections
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  | 100000 | 11114 | cb163152 | OK
|  2  | 111114 | 1111c | 58b02dc2 | OK
|  3  | 122230 | 1fbec | 2af26251 | OK
|  4  | 141e1c | 1fbec | 213393e4 | OK
|  5  | 161a08 | 28c54 | 1234df00 | OK
|  6  | 18a65c | 28c50 | a7629586 | OK


WARNING: DO NOT SWITCH OFF THE BOARD WHILE WRITING FPGA !!!

writing 0xb32ac bytes in sflash at offset 0x100000
continue [n] -> y
writing sflash:001b32ac

enter comment -> Release-04-09-2001
updating fpga signature
| idx | start  |  len  |   cks    |
|-----+--------+-------+----------|
|  1  | 100000 | 11114 | cb163152 |
|  2  | 111114 | 1111c | 58b02dc2 |
|  3  | 122230 | 1fbec | 2af26251 |
|  4  | 141e1c | 1fbec | 213393e4 |
|  5  | 161a08 | 28c54 | 1234df00 |
|  6  | 18a65c | 28c50 | a7629586 |
PPC_Mon>
```

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
status	**x**	**x**	**x**	**x**	**x**	

NAME

status - displays diagnostics results

SYNTAX

`status`

PARAMETERS

none

DESCRIPTION

The **status** command displays the result of the diagnostics executed at boot time or from the **diag** command.
The NVRAM contains an 8-bit status word for each diagnostic. It is set to 0 before entering the diagnostic. Bits 0 to 3 are used to store the current step, bits 4 to 5, the level and bits 6 to 7, the final status. When entering the diagnostic procedure, the level bits are set. The step bits are updated after every step in the diagnostic procedure. The final **status** is set when exiting the procedure.

The step bits are updated just before executing the test. If the test completes with an unrecoverable error, the global status bits are updated and the diagnostic procedure exits. If test execution crashes the system, the global status bit will stay at 0 and the step bits indicates which test caused the crash. At any level, the first test being executed is an access to the hardware resource (presence test), so a failure at step 1 indicates that the resource is not accessible.

The **sys_mem** diagnostic randomly tests the memory with pseudo random data until all the locations have been touched. If a mismatch is found, between written and read data, the test stops.

Step	*Level*	*Test*
1	0,1,2	Before memory test
2	1,2	After memory test

The bridge diagnostic tries to identify the PCI bridge and initializes it. Error handling is tested.

Step	*Level*	*Test*
1	0,1,2	Before PCI bridge identification (for IBM82660 only)
2	0,1,2	Before PCI bridge initialization (for IBM82660 only)
3	0,1,2	Before enabling error handling
4	0,1,2	Before checking bus error handling
5	0,1,2	After catching bus error

The **mmu** diagnostic prepares the PPC MMU register in order to enable address translation (this is needed before enabling the data cache in order to insure cache coherency on PCI).

Step	*Level*	*Test*
1	0,1,2	Before mmu register initialization (for IBM82650 only)
2	0,1,2	Before checking register initialization (for IBM82650 only)
3	0,1,2	Before enabling address translation (for IBM82650 only)
4	0,1,2	After address translation enabled (for IBM82650 only)

nvram

Reads NVRAM parameters and performs a checksum calculation in order to validate the data.

Step	*Level*	*Test*
1	0,1,2	Before reading NVRAM
2	0,1,2	After reading NVRAM

intctl

This diagnostic initializes the SIC 6351 interrupt controller and uses the empty / full flags of a FIFO to activate the interrupt / acknowledge mechanism and to identify the interrupt source.

Step	*Level*	*Test*
1	0,1,2	Before SIC 6351 register initialization
2	1,2	Before checking vector from IACK cycle for FIFO not empty
3	1,2	Before checking interrupt source
4	1,2	Before checking automask level
5	1,2	Before checking vector from IACK cycle for FIFO empty
6	1,2	Before checking interrupt source
7	1,2	Before waiting first interrupt
8	1,2	Before waiting next interrupt
9	1,2	After resetting SIC6351

timers

On RIO2 8062 and RIOC 4064 this diagnostic initializes the ZCIO 8536. It checks access to the I/O port and runs the internal timers.

Step	*Level*	*Test*
1	0,1,2	Before ZCIO initialization
7	1,2	After ZCIO registers access

On RIO3 8064 and RIOC 4065 this diagnostic initializes six microtimers. Checks also if timeout activate interrupt flag.

Step	*Level*	*Test*
1	0,1,2	Before microtimers initialization
2	1,2	Before timer 1 test
3	1,2	Before timer 2 test
4	1,2	Before timer 3 test
5	1,2	Before timer 4 test
6	1,2	Before timer 5 test
7	1,2	Before timer 6 test
8	1,2	End of tests

fifos

The eight FIFOs are initialized and each of them is filled until overflow and emptied until underflow.

Step	*Level*	*Test*
1	0,1,2	Before FIFO intialization
2	1,2	Before fifo #0 test
3	1,2	Before fifo #1 test
4	1,2	Before fifo #2 test
5	1,2	Before fifo #3 test
6	1,2	Before fifo #4 test
7	1,2	Before fifo #5 test
8	1,2	Before fifo #6 test
9	1,2	Before fifo #7 test

thermo

On RIO2 8062 and RIOC 4064 this diagnostic reads the thermometer current temperature, checks over.

Step	*Level*	*Test*
1	0,1,2	Before configuring the thermometer
2	1,2	Before starting temperature conversion
3	1,2	Before setting temperature limits
4	1,2	Before checking over-temperature alarm
5	1,2	Before hysteresis tests
6	1,2	Before checking under-temperature alarm
7	1,2	Before setting alarm limits from NVRAM
8	1,2	After temperature tests

On RIO3 8064 and RIOC 4065, this diagnostic sets limits for the three onboard thermometer (PowerPC, PowerPC-to-XPC Bridge, PMC2).

Step	*Level*	*Test*
1	0,1,2	Before configuring the thermometer
2	1,2	Before starting temperature conversion
3	1,2	Before setting temperature limits
4	1,2	After temperature checking

ethernet
The **ethernet** diagnostics initialize the **ethernet** controllers, performs internal and external loopback, then checks the 10/100Base-T SCSI autonegotiation mechanism.

Step	*Level*	*Test*
1	0,1,2	Before scan of ethernet devices
2	0,1,2	Before ethernet controller initialization
3	1,2	Before internal loopback
4	1,2	Before external loopback
5	2	Before autonegotiation mechanism
6	2	Before resetting to normal mode

pmc
Checks the presence of a PMC and identifies it.

Step	*Level*	*Test*
1	0,1,2	Before PMC identification
2	0,1,2	After scan of PCI devices

If this diagnostics identifies a **SCSI 8465 PM** performs **dma** internal loopback, then executes **scsi** scripts and checks the scsi bus control and data lines. If level 2 is selected, a scan of all possible **scsi** devices is performed.

Step	*Level*	*Test*
1	0,1,2	Before SCSI controller initialization
2	0,1,2	Before SCSI internal loopback
3	1,2	Before SCSI script test
4	1,2	Before control and data line test
5	2	Before scan of SCSI devices
6	2	Before scan of SCSI devices

vme
On RIO2 8062 the **vme** diagnostic checks that the two FPGA's controlling the VME interface have been loaded. If not, it reads the FPGA code from FPROM and loads them. Information about the content of the FPGA is displayed and the access to the FPGA's internal registers is checked. The VME control registers are initialized according to NVRAM parameters.
At level 2, it tries to identify a VMDIS 8003 or VMDIS 8004 board in order to run VME tests. On RIO3 8064, it initialize the VME interface based on environment parameters setting.

Step	*Level*	*Test*
1	0,1,2	Before VME register initialization
2	2	Before running VME tests
3	2	After running VME tests

cpci
If is system controller, it scans and initializes CPCI devices on the bus. If is slave, and the system controller is a CES board, it reads addresses from system controller devices table.

Step	*Level*	*Test*
1	0,1,2	Before cpci bridge identification
2	2	Before cpci bridge initialization
3	2	Before cpci slave mapping initialization
4	2	Before cpci master mapping initialization
5	2	Before scanning cpci bus (only if system controller)
6	2	Before filling cpci devices address table (only if system controller)
7	2	Before stocking cpci devices table in SRAM (only if system controller)
8	2	Before reading cpci devices table
9	2	After printing cpci devices table

xpci

This diagnostic scans and initializes PCI devices on the xpci bus. It prints the address devices table.

Step	*Level*	*Test*
1	0,1,2	Before xpci bridge identification
2	2	Before xpci bridge initialization
3	2	Before xpci slave mapping initialization
4	2	Before xpci master mapping initialization
5	2	Before scanning xpci bus
6	2	Before filling xpci devices address table
7	2	Before stocking xpci devices table in SRAM
8	2	Before reading xpci devices table
9	2	After printing xpci devices table

EXAMPLES

On a RIOC 4065 using the **set** command enables all diagnostics:

```
PPCMon>set diag

DIAGNOSTIC flags

   diag_sysmem [y/n]: n -> y
   diag_bridge [y/n]: n -> y
   diag_mmu [y/n]: n -> y
   diag_intctl [y/n]: n -> y
   diag_timers [y/n]: n -> y
   diag_fifos [y/n]: n -> y
   diag_thermo [y/n]: n -> y
   diag_bridge [y/n]: n -> y
   diag_ethernet [y/n]: n -> y
   diag_pmc [y/n]: n -> y
   diag_cpci [y/n]: n -> y
   diag_xpci [y/n]: n -> y
   PPCMon>
```

Now **reset** the board:

```
PPCMon>reset
PPC Boot Rev 4.60 created Tue Sep 18 09:41:03 2001

  Module Type: 4065AA  rev: 5  serial: 119
  Host CPU: PPC750   ver: 83.02   speed: 500 Mhz
  PCI Bridge: CES XPC
  Memory Size: 128 + 128 = 256 Mbytes
  L2 CACHE:  1 Mbyte SSRAM   200 MHz
  FPROM: Two banks of AMD29F032 installed [16 Mbytes]
  Ethernet Address: 00:a2:80:01:40:87

Entering boot diagnostics

 0    Check System Memory (0x00004000 - 0x0ff00000)          1  OK
 1    Check Interrupt Handler                                0  OK
 2    Check PCI Bridge                                       1  OK
 3    Check Cache and MMU                                    1  OK
 4    Check MK48T08 RTC and NVRAM                            1  OK
 5    Check SIC6351 Interrupt Controller                     1  OK
 6    Check Serial Lines                                     1  SKIPPED
 7    Check  Micro Timers                                    1  OK
 8    Check FIFO's (0 - 7)                                   1  OK
 9    Check Digital Thermometers                             1  OK
10    Check PCI devices                                      1  OK
11    Check On Board Ethernet Controller  (PCI slot 0)       1  OK
12    Check PMC #1 Extension  (PCI slot 2)                   1  NO DEVICE
13    Check PMC #2 Extension  (PCI slot 1)                   1  NO DEVICE
14    Check CPCI Interface                                   1  OK
15    Check XPCI Interface


PPCMon>
```

The **status** command displays the diagnostic results:

```
PPC_Mon>status

   diagnostic | level |  status   | step
   -------------------------------------
   sysmem     |   0   |  SKIPPED  |  1
   bridge     |   0   |    OK     |  2
   mmu        |   0   |    OK     |  4
   mvram      |   0   |    OK     |  2
   intctl     |   0   |    OK     |  1
   serial     |   0   |  SKIPPED  |
   timers     |   0   |    OK     |  1
   fifos      |   0   |    OK     |  9
   thermo     |   0   |    OK     |  4
   ethernet   |   0   |    OK     |  3
   cpci       |   0   |    OK     |  6
   xpci       |   0   |    OK     |  6

PPCMon>
```

The PMC diagnostic shows a failure. Using the **diag** command, we execute the test procedure at level 2 on a FETH 8478:

```
PPCMon>diag pmc

entering PMC diagnostics

ETHERNET controller found at PCI slot 1 [00:80:a2:00:18:8c]

AM79C970 Internal loopback test..0..OK  PASSED
AM79C970 External loopback test..0
receive time out

diag_am79c970: error number -1
PPCMon>
```

It tells us that the PMC is an ethernet controller and the external loopback test failed because the connection to the network was missing.

SEE ALSO

diag, **set**

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
tc			**x**		**x**	
tm	**x**	**x**	**x**	**x**	**x**	**x**
tp			**x**	**x**	**x**	**x**
tv			**x**	**x**		
tx			**x**	**x**		

NAME

tc, **tm**, **tp**, **tv**,**tx** - tests the CPCI, memory, PCI,VME or PCI2

SYNTAX

tc.*x start..end[space]*
tm.*x start..end*
tp.*x start..end[space]*
tv.*x start..end[amode]*
tx.*x start..end[space]*

PARAMETERS

x	data size[**l**, **w**, **s**, **h**, **b**, **c**]
start	start address[**0x..** hexa, **0d..** dec, **0o..** oct]
end	end address[**0x..** hexa, **0d..** dec, **0o..** oct]
space	PCI space[**io**, **mem**]
amode	VME address mode[from **0x00** to **0xff**]

DESCRIPTION

tm executes a set of read / write accesses on a block of memory starting at address *start* and ending at address *end*. The block is first filled with 0s and checked. Then every bit is set to 1, checked and reset to 0. Then the block is filled with 1s and checked. Then every bit is set to 0, checked and reset to 1. Finally the block is filled with a random data pattern and checked. Data accesses are done according to the data size specified in *x,* where *x* is **w** or **l** for a 32-bit access, **h** or **s** for a 16-bit access, **b** for a 8-bit access and **c** for an ASCII character access. The number of data displayed is (| *end - start* |) *x*. Any error occurring during the test procedure is signaled by an error message. The test goes on until a character is entered. Every time the test is replayed, the loop count is displayed.
To test VME, CPCI, PCI or PCI2 addresses, an additional parameter is needed, **amode** is the VME address mode (from **0x00** to **0xff**) and space is the PCI / CPCI space.

On RIO2 806x, *amode* defines address modifier (A32 addressing mode: *amode* = 0x9, A24 addressing mode: *amode* = 0x39). The default value is set to 0x9. On a RIO3 8064, *amode* encodes the A_Type & D_Type & P_Type & 0 field used to build the VME addressing mode (A32 addressing mode: *amode* = 0xa0, A24 addressing mode: *amode* = 0x80). The default value is set to 0xa0.

The PCI space must be one of the following ASCII strings: **io** for PCI IO space, **mem** for PCI memory space. Command parameters are stored in a data structure, and when they are missing, the firmware takes them from that data structure.

EXAMPLES

Fills a memory block from address 0x800000 to 0x800080, then tests it.

```
PPCMon>fm.w 800000..800080
PPCMon>tm.w 800000..800080 13131313
60 testing: 0x00800000..0x00800080
```

To stop the memory test enter SP (space bar). Display the memory contents after the test.

```
PPCMon>dm.w 800000..800080
0x00800000 : 13131313 36eca2c6 5ac63279 7e9fc22c  ....6...Z.2y....
0x00800010 : a27951df c652e192 ea2c7145 0e0600f8  .yQ..R....qE....
0x00800020 : 31df90ab 55b9205e 7992b011 9d6c3fc4  1...U...y....l..
0x00800030 : c145cf77 e51f5f2a 08f8eedd 2cd27e90  .E.w.....~.....
0x00800040 : 50ac0e43 74859df6 985f2da9 bc38bd5c  P..Ct.....8..
0x00800050 : e0124d0f 03ebdcc2 27c56c75 4b9efc28  ..M.....luK...
0x00800060 : 6f788bdb 93521b8e b72bab41 db053af4  ox...R.....A....
0x00800070 : fedecaa7 22b85a5a 4691ea0d 6a6b79c0  ......ZZF...jky.
PPCMon>
```

SEE ALSO

fm

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
temp	**X**	**X**	**X**	**X**	**X**	

NAME

temp - displays the CPU temperature

SYNTAX

temp

PARAMETERS

none

DESCRIPTION

On RIO2 8062 and RIOC 4064 the **temp** command displays the temperature read from a digital thermometer located under the PowerPC CPU. On RIOC 4065 it displays temperature from three onboard sensors: one is located inside the PowerPC and controls the temperature of the silicon; the other two are located in SDRAM and in the middle of the PMC2 area. On RIO3 8064 the temperature sensors are located in PowerPC CPU and in the two PMC areas. The temp command shows also the value of the low and high limits defining the threshold at which an alarm signal is set by each thermometer. The **setenv** command allows the user to set these limits. A temperature control parameter allows the user to automatically put the PowerPC in sleep mode if an over-temperature is detected.

EXAMPLES

On a RIOC 4065, show current temperature and alarm threshold.

```
PPC_Mon>temp
temperature CPU   = 38 degrees [70-75]
temperature SDRAM = 39 degrees [70-75]
temperature PMC2  = 46 degrees [70-75]
PPC_Mon>
```

SEE ALSO

setenv

Command	*Onboard*	*33170A*	*33170B*	*VME*	*CPCI*	*MFCCMon*
upara	**x**	**x**	**x**	**x**	**x**	

NAME

upara - creates a user defined environment parameters.

SYNTAX

upara.*x op [name] [value]*

PARAMETERS

x	parameter index	
op	operation	
	create	*name [value]*
	save	
	restore	

DESCRIPTION

The **upara** command allows the user to create up to 15 user-defined environment parameters. They are stored in NVRAM and are manipulated by the **set** and **show** commands just like any other environment parameter.

By default user parameter are labeled '1','2'..'15' and belong to the parameter group 'para'. Their value is a 48 Bytes string initialized by PPCMon to the NULL string.

The operation **create** allows the user to rename any parameter as well as their group name. After having built new environment parameters, they can be stored in NVRAM using the **save** operation. Each time PPCMon boots, its rebuilds the environment parameters from NVRAM. At any time, the **restore** operation restores parameter names and values from NVRAM.

EXAMPLE

Display all user parameters using the show command

```
PPCMon>show para

User's parameters

   para_1 :
   para_2 :
   para_3 :
   para_4 :
   para_5 :
   para_6 :
   para_7 :
   para_8 :
   para_9 :
   para_10 :
   para_11 :
   para_12 :
   para_13 :
   para_14 :
   para_15 :

PPCMon>
```

The operation **create** allows the user to give a new definition of the group name.

```
PPCMon>upara create mygroup
PPCMon>
```

At this point the user parameter groups is 'mygroup' and not anymore **para**. This can be checked with the **show** command.

```
PPCMon>show para
PPCMon>show mygroup

User's parameters

   mygroup_1 :
   mygroup_2 :
   mygroup_3 :
   mygroup_4 :
   mygroup_5 :
   mygroup_6 :
   mygroup_7 :
   mygroup_8 :
   mygroup_9 :
   mygroup_10 :
   mygroup_11 :
   mygroup_12 :
   mygroup_13 :
   mygroup_14 :
   mygroup_15 :
PPCMon>
```

To create individual parameters, just use the upara.x command, where x is the index of the parameter you want to rename. The initial value of that parameter can be given.

```
PPCMon>upara.6 create mypara 12345678
PPCMon>show mygroup

User's parameters

   mygroup_1 :
   mygroup_2 :
   mygroup_3 :
   mygroup_4 :
   mygroup_5 :
   mygroup_mypara : 12345678
   mygroup_7 :
   mygroup_8 :
   mygroup_9 :
   mygroup_10 :
   mygroup_11 :
   mygroup_12 :
   mygroup_13 :
   mygroup_14 :
   mygroup_15 :
PPCMon>
```

The parameter **mypara** belonging to **mygroup** can then be set to any value (stored as a 48-Byte string) using the **set** command.

```
MFCC_Mon>set mygroup_mypara
   mygroup_mypara : 12345678 -> hello
PPCMon>show mygroup_mypara
   mygroup_mypara : hello
PPCMon>
```

Now using the save operation, we can store the modified parameters in NVRAM and reset the board.

```
PPCMon>upara save
PPCMon>reset


PPC Boot Rev 5.20 created Mon Sep 20 16:17:06 2004
....

************************************************
*   PPC_Mon RIOC4068 monitor - version 5.2      *
*      CES SA Copyright 1995 - 2004             *
************************************************

PPCMon>show mygroup

User's parameters

   mygroup_1 :
   mygroup_2 :
   mygroup_3 :
   mygroup_4 :
   mygroup_5 :
   mygroup_mypara : hello
   mygroup_7 :
   mygroup_8 :
   mygroup_9 :
   mygroup_10 :
   mygroup_11 :
   mygroup_12 :
   mygroup_13 :
   mygroup_14 :
   mygroup_15 :
PPCMon>
```

SEE ALSO

set, **show**

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
user	x	x	x	x	x	

NAME

user - executes a user code

SYNTAX

user

PARAMETERS

none

DESCRIPTION

The **user** command allows manually to execute a user standalone application stored at the FPROM offset 0x90000 on RIO2 8062 and RIOC 4964, and at the offset 0xa0000 on RIO3 8064 and RIOC 4065. That region of FPROM is reserved to hold some code to be automatically executed by the PPCMon automatic boot procedure just before launching an operating system (see **setenv boot**). Usually, that mechanism is used to setup some specific configuration needed by the OS before booting.

EXAMPLES

On a RIO2 8062, load a user code in FPROM at offset 0x90000.

```
PPC_Mon>fprom load 90000 le /usr/ces/PPCMon/l2tset/mycode 200000
network boot
loading file '/usr/ces/PPCMon/l2tset/mycode' from server 144.0.10.3
ethernet local address = 00:80:a2:00:a0:74
ethernet address of server = 00:80:a2:00:c0:90
loading at address:200000

tftp packet number:        1
file length = 0x3ecf [16079]
transfer rate: 38 kbyte/sec
fprom_off = 90000   mem_addr = 200000  len = 3ecf
erasing sector 0x88000
loading at address : 0x80293c00
PPCMon>user
```

Then execute the user code.

```
PPCMon>user
entering user code ...

This is a test program for the Rio2 8062

returning to PPC_Mon>....
PPCMon>
```

Now we set the boot mode to 70, in order to force the user code execution before launching an OS. After reset, PPCMon will automatically call the user code before going on according to the boot environment parameters setting.

```
PPCMon>setenv boot
Host boot mode [50] : 70
PPCMon>reset
...
enter any character to stop auto boot...
entering user code ...

This is a test program for the 8062
returning to PPC_Mon>....
loading from disk(2,0,2) at address:0
scsi device 0 found
starting code execution at address: 4000
...
```

SEE ALSO

setenv

Command	Onboard	33170A	33170B	VME	CPCI	MFCCMon
version	x	x	x	x	x	x

NAME

version - shows the version of PPCMon

SYNTAX

`version`

PARAMETERS

none

DESCRIPTION

The **version** command displays the name of the firmware, the revision number of the version installed by CES and the week of installation. This information is related to the CES traceability system and are stored in the board signature data structure, kept in EEPROM. Since the **fprom** command allows the user to update of the firmware, the revision number and the date of compilation of the currently running version of the firmware is also displayed.

EXAMPLES

Display firmware version information.

```
PPC_Mon>version

   PPC_mon : CES installed version 4.8   ( 2000  week  0)
             current version       5.20  ( Tue Sep 21 14:26:32 2004)

PPC_Mon>
```

SEE ALSO

fprom

Chapter 2
PPC_FlashLoad Commands

WARNING This utility is only available on the RIO2 806x, RIOC 4063 and RIOC 4064 platforms.

The PPC_FlashLoad utility provides the user with functions to erase the Flash EPROMs and to program them from either a piggy-back EPROM or from an S-record file transferred over the tty1 serial line. Upon entry, `PPC_FlashLoad` attempts to determine automatically which types of Flash EPROM are installed and how many banks there are. A message is printed to inform the user.

Certain sectors of the Flash EPROMs are protected by a password. If the user gives an incorrect password, no action is taken. If the correct password is given, the command is executed, and the password will not be requested again.

The address range given by the user for the `erase` and `prom` commands is treated differently according to whether the address is below 8 MBytes or not. If it is above 8 MBytes, it is assumed to be the real address of the Flash EPROM (in PCI I/O space, seen from the CPU). If it is below 8 MBytes, it is assumed to be a "logical" address, which is then mapped onto a physical address.

This mapping depends on the Flash EPROMs size, and is given here as the address within the EPROM and the corresponding EPROM bank number.

User Supplied Address	*16-Mbit Devices*		*8-Mbit Devices*	
	Bank	*Mapping*	*Bank*	*Mapping*
0 M -> 1 M	1	3 M -> 4 M	1	1 M - 2 M
1 M -> 2 M	1	2 M -> 3 M	1	0 M - 1 M
2 M -> 3 M	1	0 M -> 1 M	2	1 M - 2 M
3 M -> 4 M	1	1 M -> 2 M	2	0 M - 1 M
4 M -> 5 M	2	0 M -> 1 M	x	illegal
5 M -> 6 M	2	1 M -> 2 M	x	illegal
6 M -> 7 M	2	2 M -> 3 M	x	illegal
7 M -> 8 M	2	3 M -> 4 M	x	illegal

NOTE The second bank may be not implemented, in which case addresses in these ranges are also illegal, and are reported as such to the user.

Users must never try to use addresses above 8 MBytes, because it could cause unexpected Flash EPROMs erase / overwrite.

ERASE

Name

```
erase
```

Syntax

```
erase <start>..<end>
```

Parameters

```
<start>  start address   [0x.. hexa, 0d.. dec, 0o.. oct]
<end>    end address     [0x.. hexa, 0d.. dec, 0o.. oct]
```

Description

Erases the contents of the Flash EPROM sectors from <start> to <end>. These may be logical addresses, as described above.

Example

```
PPC_FlashLoad>erase 5e0000..5ffffe
Flash EPROM erase
Range 0x005e0000 - 0x005fffff
Sector 0x005e0000 : Mapped 0x801e0000 : Base 0x801e0000 : Size 0x20000

PPC_FlashLoad>
```

SREC

Name

```
srec
```

Syntax

```
srec
```

Description

Loads an S-record file into the Flash EPROM via the serial line tty1. The S-records are checked for legality as they are loaded, and loading stops as soon as any error occurs. After successfully loading the file into memory, the S-records are decoded and loaded into the Flash EPROMs. The S-records must contain the correct EPROM target addresses, since this is not checked by the software. The sectors to be loaded should have been erased first using the erase command.

Example

```
PPC_FlashLoad>srec
Flash EPROM SREC Download
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
..................
..................................................................
Total Size  =  0x..1ebe8
0x80 bytes from 0x0021fa4f to 0x007dc80 (0x0 left)
Number of bytes loaded = 0x1ebe8
PPC_FlashLoad>
```

PROM

Name

```
prom
```

Syntax

```
prom <start>..<end> <dest>
```

Parameters

```
<start>    start address        [0x.. hexa, 0d.. dec, 0o.. oct]
```

```
<end>      end address          [0x.. hexa, 0d.. dec, 0o.. oct]
<dest>     dest address         [0x.. hexa, 0d.. dec, 0o.. oct]
```

Description

Copies the contents of the CES piggy-back PROM from address <start> to address <end> into the Flash EPROM beginning at address <dest>. <Start> and <end> must be correct for the EPROM as seen by the CPU, but <dest> may be a logical address as described above. The sectors to be loaded should have been erased first using the erase command.

This special command is used at factory to load, for the first time, the Flash EPROM from a special PMC holding the master PROM. Users must never try to use this command.

Example

```
PPC_FlashLoad>prom fef60000..fef7fffe 60000
Flash EPROM Prom load
From 0xfef60000 to 0x00060000, 0x1fffe bytes
EPROM address  :  0x8037fc00
PPC_FlashLoad>
```

QUIT

Name

`quit`

Syntax

`quit`

Description

Exits from `PPC_FlashLoad` and starts the primary bootstrap.

Example

```
PPC_FlashLoad>quit

PPC Boot Rev 2.0 created Thu Oct  5 19:20:29 1995

1      PPC Board Signature update
2      Fpga Load : vme.s rev 2.0 23.03.95 chsm = c011de92 CHSM OK  loaded
       Fpga Load : bma.s rev 3.1 28.09.95 chsm = 2eb5388c CHSM OK  loaded
...
```

Chapter 3
RIOMon Commands

WARNING This utility is only available on the RTPC 8067 platforms.

3.1 Introduction

The primary purpose of the RIO CPU on the RTPC 8067 is to configure the board, to run power ON tests (PON) and to release the HOST CPU reset. For this, the CPU executes a primary bootstrap process divided in several phases and then starts the monitor code itself. The `RIO_Mon` monitor provides the user with a set of tools, which can be used to download executable code, to examine memory locations, to access PCI or VME resources or to start CES or custom tests.

The RIO CPU code is located into a Flash EPROM, which is divided into sectors. Some sectors are free and therefore are intended to receive user's code or tests. The exact layout of the EPROM structure is described in the next section.

3.2 Flash EPROM Layout

The RTPC 8067 is equipped with three Flash EPROMs, erasable by sectors. The code for the RIO CPU is located in the Flash EPROM 1 and the code for the HOST CPU is located in Flash EPROMs 2 and 3.

After RESET, the RIO starts to execute the primary bootstrap code if the jumper J3 BOOT_JUMP is not set, otherwise the `RIO_FlashLoad` utility is invoked. The purpose of `RIO_FlashLoad` is to give the user a way to erase sectors and download new or updated code into the Flash EPROMs.

SECTOR 4 User (128 KBytes)
SECTOR 3 User (128 KBytes)
SECTOR 2 `RIO_Mon` Primary BOOT (128 KBytes)
SECTOR 1 FPGA (96 KBytes)
SECTOR 0 `RIO_FlashLoad` (32 KBytes)

The following will examine now the contents and the functionality of each sector:

sector 0

This sector should never be erased or updated because this is where the `RIO_FlashLoad` utility is located. As described before, this portion of code is only used at reset and in case of Flash EPROM update. Normally, control is given to the sector 2 where the primary bootstrap sequence can be executed. Therefore, a small program verifies the validity of sector 2 and in case of failure forces a jump into `RIO_FlashLoad`..

sector 1

This sector contains the configuration file for the onboard programmable logic (FPGA's) and power-ON tests used during the primary bootstrap. The FPGA's files are also protected with a checksum.

sector 2

Here is located the entire primary bootstrap sequence and the RIO monitor. The primary boot is split into several phases. The `RIO_Mon` is started if it completes successfully.

sector 3

This sector is used to store CES test programs started with special `RIO_Mon` commands or the user's firmware.

sector 4

This sector is used to store CES test programs started with special `RIO_Mon` commands or the user's firmware.

3.3 The Boot Block

3.3.1 Introduction

The boot block is a 32-KByte area located at the base of the Flash EPROM #1. This sector includes the reset code (reset address for the R3051) and the `RIO_FlashLoad` utility.

3.3.2 The Reset Code

The reset code starts by dumping all registers in a SRAM area, visible from the VME. This is a useful diagnostic in case of bad crashes. Then a basic test of SRAM locations used by the C code is performed. Afterwards, the coherence of the Flash EPROM sector 2 is examined and in case of success, the primary bootstrap procedure is started. Otherwise, the `RIO_FlashLoad` is invoked. Note that you can force the `RIO_FlashLoad` execution by plugging the jumper J3 BOOT_JUMP.

3.3.3 The RIO_FlashLoad Utility

The `RIO_FlashLoad` is a program, which gives a way to erase or update sectors of the Flash EPROM. The update is done from the front panel serial line and the expected format is S-record (commands described hereafter). If you directly hit <RETURN>, a help menu is printed.

Commands are:

Erase Flash EPROM

```
e<flash_number> <sector_base_address>
```

Download from serial line into Flash:

```
l<swap> <flash_number> <srec> <channel>
```

Download from piggy-back EPROM

```
l<swap> <flash_number> <prom>
```

Quit `RIO_FlashLoad` (Continue RIO bootstrap):

```
q
```

with:

Swap	*Swap Type*
0	no swap
1	byte swap
2	word swap
3	word and byte swap

`flash_number`	1, 2 or 3
`sector`	0x0, 0x4000, 0x6000, 0x8000, 0x20000, 0x40000, 0x60000
`channel`	1 or 2

EXAMPLE

```
RIO_FlashLoad>e1 0x40000
flash EPROM erase sector 40000

RIO_FlashLoad>l0 1 srec 1
.................................................
number of bytes loaded = 10a10

RIO_FlashLoad>q

(start the RIO primary bootstrap - see next section)
```

3.4 The RIO Monitor

3.4.1 Introduction

The purpose of this section is to examine the contents of the second sector of the Flash EPROM. In fact it is possible to divide this area into two parts. The first one containing the RIO primary bootstrap and the second one the monitor itself.

3.4.2 The RIO Primary Bootstrap

The RIO primary bootstrap is the code, which is used to perform the board's initialization and basic diagnostics. This RIO bootstrap is divided in several phases as described hereafter:

```
0       RIO Flash EPROM Code OK
1       RIO Board Signature update
2       Fpga Load : vme.s  rev 2.0  23.03.95  chsm = c011de92 CHSM OK  loaded
        Fpga Load : bma.s  rev 3.1  28.09.95  chsm = 2eb5388c CHSM OK  loaded
3       RIO Basic Devices Init 01
4       RIO PON Test_A  Disabled
5       Hardware Configuration
                Mod. Type = 8067EA  SN = 0015  Hard Rev. = A1
                RIO  = R3051     16 Mhz
                HOST = PPC603    66 Mhz
                SYSRAM = 32 Mbytes
                PCI Dev#0   VendorID = 0x1022  DeviceID = 0x2000  Rev. = 0x02
                PCI Dev#1   VendorID = 0x1000  DeviceID = 0x0003  Rev. = 0x02
                PCI PMC     VendorID = 0x1014  DeviceID = 0x0022  Rev. = 0x00
                Ethernet Address = 00.80.a2.00.21.26
                VME Hex-Rotary = 5    Slot1 DISABLED
6      VME Interface Init
7      RIO End Primary Boot

RTPC 8067 RIO monitor version 2.1
22/05/96   08:30:53  Ta = 34 C
RIO_Mon>
```

Several phases can be controlled with environment variables. It is the case for phases 1, 4, 6 and 7. Please refer to the section for a complete description of the set environment (`setenv`) command.

If specified, the RIO will automatically start-up the PowerPC bootstrap at the end of the RIO bootstrap (`refer` to `setenv boot` command). This automatic procedure can be aborted by typing any character at the console.

3.4.3 The RIOMon

The RIO_Mon offers the user a way to load and test new code along with a direct control on environment variables. Once the prompt RIO_Mon> appears on the screen, the user has the choice between a set of commands which can be used with several options. An exhaustive list with examples is provided.

GENERAL RULES

In all cases <op> must be w (32 bits), h (16 bits), b (8 bits), d (2 words), q (4 words), o (8 words), h (16 words).

All data reference are defined in hex with 0x delimiter. In all modes with infinite loop type space bar to stop.

GENERAL COMMANDS

boot

This command allows to manually start the PowerPC bootstrap. PowerPC bootstrap options are defined with the setenv boot command.

```
RIO_Mon>boot
```

go

Starts a user's program go <0x address>

```
RIO_Mon>go 0xa0010000
```

lf

Loads Flash. This command allows to program the Flash EPROMs by copying an array from the PowerPC System DRAM into a selected Flash EPROM. This command is useful after a network download called at PPC_Mon. The Flash EPROM erase must be done at RIO_FlashLoad level. The command parameters are defined as follows:

<flash_number>	= Flash EPROM number -> 1, 2, 3
<0xfoff>	= Flash address offset
<0xdoff>	= System DRAM address offset
<0xbcount>	= Byte count

```
lf.< flash number > <0x..foff> <0x..doff> <0x..bcount>
```

```
RIO_Mon>lf.3 0x20000 0x1000000 0x3000
RIO_Mon>
```

help

Prints a summary of the commands implemented.

```
RIO_Mon>help
...
...
...
```

ppc

This command allows to return to PPC_Mon> directly, after the PowerPC has booted once (without rebooting the PowerPC).

```
RIO_Mon>ppc
PPC_MON>riomon
RIO_Mon>
```

q

This command allows to jump directly to the RIO_FlashLoad utility.

```
RIO_Mon>q
```

setenv boot

This command allows to set-up the bootstrap policy for the two on-board CPUs. The following options are selectable:

- Host boot word 0 defines a delay in second before starting the Host CPU.
- Host boot word 1 is not used.

Host boot mode defines the bootstrap continue policy at the end of RIO bootstrap. The 8-bit mode field is divided in 2 as follows:

Bits	*Name*	*Description*
[00:04]	RIO related options	0 = does not boot the PowerPC 1 = boots PowerPC & stays at PPC_Mon> 2 = boots PowerPC & jumps to PPC_TstMon> 3 = boots PowerPC & jumps to PPC SMon 4 = boots PowerPC & executes PowerPC STA 5 = boots PowerPC & boots PowerPC OS>
[03:00]	PowerPC related options	0 = stays at RIO_Mon 1 = jumps to RIO SMon (CES Test utility)

```
RIO_Mon>setenv boot
Host boot word 0 = [00000000] ->
Host boot word 1 = [00000000] ->
Host boot mode = [00] ->
RIO_Mon>
```

MEMORY RELATED COMMANDS

bm

Memory block move command

```
bm.<op> <0xsadd>..<0xdadd> <0xbcount>
```

```
RIO_Mon>bm.d 0xa0010000..0xa0011000 0x1000
RIO_Mon>
```

cm

Memory array comparison command

```
cm.<range> <0xadd1>..<0xadd2> <0xbcount>
```

```
RIO_Mon>cm.w 0xa0010000..0xa0010020 0x1f
a001000c = 0   a001002c = 12
a0010014 = 0   a0010034 = 23
find 2 differences
RIO_Mon>
```

dm

Memory contents display command

```
dm.<range> <0x..add>
```

```
RIO_Mon>dm.w 0xa0010000
0xa0010000 : 12345678 00000000 00000000 00000000
0xa0010010 : 34567811 00000000 11111111 22222222
0xa0010020 : 00000800 00000000 00000000 00000000
0xa0010030 : 00000800 00000000 00000000 00000000
0xa0010040 : 00000008 00000000 00000000 00000000
RIO_Mon>
```

fm

Memory fill command

```
fm.<range> <0xsadd>..<0xeadd> <0xdata>
```

```
RIO_Mon>fm.w 0xa0010000..0xa0011000 0x11112222
RIO_Mon>
```

MM

Memory contents move command

```
mm.<range> <0xsadd>..<0xeadd> <0xdadd>
```

```
RIO_Mon>mm.w 0xa0010000..0xa0011000 0xa0014000
RIO_Mon>
```

PM

Memory contents patch command

```
pm.<range> <0xadd> <0xdata>
```

```
RIO_Mon>pm.w 0xa0010000 0xdeadface
RIO_Mon>pm.w 0xa0010000
0xa0010000 : 0xdeadface ->
0xa0010004 : 0x00000000 ->
0xa0010008 : 0x00000000 -> -
0xa0010004 : 0x00000000 -> .
RIO_Mon>
```

sm

Memory search command

```
sm.<range> <0xsadd>..<0xeadd> <0xdata>
```

```
RIO_Mon>sm.w 0xa0010000..0xa0011000 0x11112222
RIO_Mon>
```

tm

Memory test command

This command allows to run infinite memory test loops. The memory is first filled with all 0, then all 1, then 0 in 1, then 1 in 0, then random data. The memory test is halted if an error is detected or if the <SPACE> character is typed:

`tm.<range> <0x..sadd>..<0x..eadd> <0x..mask>-` Memory test command

```
RIO_Mon>tm.w 0xa0010000..0xa0011000 0xffffffff
.........................
```

tr

Touch READ command

tw

Touch WRITE command

twr

Touch WRITE-READ command

These commands allow to continuously issue a specified cycle, with all errors by-passed. The cycling is stopped if the <SPACE> character is typed. This is useful for hardware debugging.

```
tr.<range> <0xadd>
tw.<range> <0xadd> <0xdata>
twr.<range> <0xadd> <0xdata>
```

```
RIO_Mon>tw.w 0xa0010000 0x11112222
RIO_Mon>tr.w 0xa0010000
RIO_Mon>twr.w 0xa0010000 0xdeadface
RIO_Mon>
```

vpm

VME patch memory command:

This command allows to directly access the VME without initializing the VME interface. The command `setenv vdbg` allows to select arbitration modes and other VME interface options:

```
vpm.<range> <0xam> <0xadd> <0xdata>
vpm.<range> <0xam> <0xadd> <cr>
```

```
RIO_Mon>vpm.w 0x9 0x200000 0x111122222
RIO_Mon>vpm.w 0x9 0x200000
0x9 0x200000 : 0x11112222 ->
0x9 0x200004 : 0x00000000 -> .
RIO_Mon>
```

setenv vdbg

This command allows to set-up different VME interface options, which are used by the `vpm` command. The following options are selectable:

`arb_level` defines the VME BREQ level used.
- [0] VME BREQ_0
- [1] VME BREQ_1
- [2] VME BREQ_2
- [3] VME BREQ_3

`rel_mode` defines the VME arbitration release policy.
- [0] Release on request (ROR)
- [1] Release when done (RWD)

`fair` defines the VME arbitration request policy.
- [0] Standard mode
- [1] FAIR mode.

`hidden` defines the VME BBSY* deassertion policy.
- [0] Standard BBSY release
- [1] Early BBSY release.

`lto` defines the VME local time-out set-up.
- [0] LTO = 7.6 µs
- [1] LTO = 15.6 µs
- [2] LTO = 30.7 µs
- [3] LTO = 61 µs

`bto` defines the VME bus time-out set-up.
- [0] BTO = 16 µs
- [1] BTO = 32 µs
- [2] BTO = 64 µs
- [3] BTO = disabled

`wpost` defines the PCI-to-VME write policy.
- [0] Standard compelled WRITE
- [1] WRITE posting enabled

`prefetch` defines the PCI-to-VME read policy.
- [0] Standard compelled READ
- [1] CES reserved

`swap` defines the PCI-to-VME swapping policy.

[0] No swapping
[1] AUTO swapping
[2] Word swapping
[3] Word & Byte swapping

`vblt` is a CES reserved option.

```
RIO_Mon>setenv vdbg
arb_level[3]:
rel_mode[1]:
fair[0]:
hidden[0]:
lto[0]:
bto[0]:
wpost[0]:
prefetch[0]:
swap[0]:
vblt[0]:
RIO_Mon>vpm.w 0x9 0x200000 0x111122222
RIO_Mon>
```

ENVIRONMENT COMMAND

setenv time

This command allows to set-up the onboard RTC time and date:

```
RIO_Mon>setenv time
day of month   [30] : 30
month          [11] : 11
year           [94] : 94
hours          [08] : 08
minutes        [51] : 53
seconds        [13] : 00
date : 30/11/94  time : 08:53:00
RIO_Mon>
```

setenv temp

This command allows the user to set-up onboard thermometer watchdog limits. The temperature limit high, is the one where the front panel led OVTEMP will light on.

```
RIO_Mon>setenv temp
Temp Limit High [255]: 55
Temp Limit Low  [255]: 50
RIO_Mon>
```

Chapter 4
Firmware Update

The RTPC, RIO2, RIOC Flash EPROMs are implemented with 64 KBytes sector. Two devices are used in parallel giving a granularity of 128 KBytes per sector. The factory-supplied firmware occupies 512 KBytes.

The PPCMon firmware can be upgraded as follows:

1. Get from CES the latest version of the PPCMon firmware.

The actual version of PPCMon for different CES modules and for the corresponding files are:

Board	*File*
RTPC 8067	8067prom.b370
RIO2 8060	8060prom.b370
RIO2 8061	8061prom.b370
RIO2 8062	8062prom.b370
RIO3 8064	8064prom.b490
RIO3 8064LC	8065prom.b490
RIO3 8066	8066prom.b490
RIOC 4063	4064prom.b370
RIOC 4064	4064prom.b370
RIOC 4065	4065prom.b490
RIOC 4068	4068prom.b490
RIOC 4070	4070prom.b520

2. Update your version of PPCMon.

The `PPC_Mon` can be reloaded directly through a TFTP server at `PPC_Mon` level.

- Copy the `PPC_Mon` binary file on a tftp server
- Connect the network to your unit
- Power-up the module to get the `PPC_Mon>` prompt
- Define the INET parameters with the `set inet` command
- Load the Flash EPROM with the `fprom` command.

```
PPC_Mon>set inet

INTERNET parameters

   inet_host [dotted decimal] : 144.0.10.2
   inet_server [dotted decimal] : 144.0.4.49
   inet_broadcast [dotted decimal] : 255.255.255.255
   inet_mask [dotted decimal] : 0.0.0.0
   inet_protocol [ arpa bootp ] : arpa
PPC_Mon>fprom load 0 le /tftpboot/4070prom.bin
loading fprom... 0
You are about to overwrite the FPROM boot sector !!!
continue? [n] ->y

Trying device  le   [file : /tftpboot/4070prom.bin] ...
ethernet load
local address = 00:80:a2:01:40:74
using ARPA
.file server address 00:06:5b:b8:bd:eb
loading file '/tftpboot/4068prom.bin' from server 10.0.4.49 at address
0x200000
tftp packet number:      602
transfer rate: 1728 kbyte/sec [len = 0xc0000]
fprom_off = 0   mem_addr = 200000  len = c0000
Password ->goldfinger
FPROM erasing sector 000a0000
FPROM programming offset 000c0000
PPC_Mon>
```

Then type reset, and verify with "PPC_Mon> version", that the new version has been installed.

```
PPC_Mon>version

   PPC_mon : CES installed version 5.2   ( 2004  week 30)
             current version       5.20  ( Thu Sep 23 09:21:52 2004)

PPC_Mon>
```

Chapter 5
NVRAM Structure

The NVRAM consists of 8 KBytes of memory, with each Byte aligned on a long-word boundary.

The firmware uses values stored in the NVRAM to control the boot of the processor, and to control the tests used in production of the board. Many of these values are of little interest to the user who wishes to boot an operating system, and thus are not recorded here. For most users, only the hardware configuration information is important. This is all contained in the signature structure, which is currently stored at an offset of 0x4000 from the beginning of the NVRAM. The Bytes from 0 - 0x3FFF are not used by the firmware, and the user may define the contents as he wishes.

A backup copy of the NVRAM can be stored in FPROM using the “nvram save” command.

Elements in the signature structure are defined as long-words, but only the lower 8 bits are valid. Users should never modify these values, as this may prevent the board from booting!

The example given, shows how to retrieve the name of the module-type and it's serial number.

```
void PrintModuleInfo( void )
#define NVRAM_BASE0xfc200000     /* PPC-mapped */
#define NVRAM_COM_OFFSET0x4000
{
  struct Signature *Sign =
 (struct Signature *)(NVRAM_BASE + NVRAM_COM_OFFSET);
  long i, serial;
  char module[7];

  for ( i=0; i<7; i++ )
  { module[i] = *(char *)&Sign->ModuleType[i]; }

  serial = ( ( *(char *)&Sign->Serial[0] ) << 8 ) +
       *(char *)&Sign->Serial[1];

  printf(" This board is a %s, serial number 0x%x\n",
module,serial);
}
```

The interesting elements of the structure are listed. Unless otherwise stated, all parameters are hexadecimal numbers. The actual permitted values and their meanings are not given, as they may change.

The offsets defined below, indicate the start of various sub-structures in the NVRAM. Two of these structures are given in more details later on.

```
#define NVRAM_COM_OFFSET        0x4000
#define NVRAM_OS_OFFSET         0x7000
#define NVRAM_PCI_OFFSET        0x4800
#define NVRAM_XPCI_OFFSET       0x4a00
#define NVRAM_CPCI_OFFSET       0x4c00
```

The signature structure contains vital configuration parameters needed by PPCMon to initialize the card. This structure is prefected by a checksum. If PPCMon detects a checksum error, it will try to restore the signature using a backup copy in FPROM (see `nvram save`).

```
struct Signature
{
  long ModuleType[7];
  long Serial[2];
  long PCBrevMboard;
  long PCBrevCPUpiggy;
  long HardRev[3];
  long Class[2];
  long RioCPUType;
  long RioCPUSpeed;
  long HostCPUType;
  long HostCPUSpeed;
  long HostBridgeType;
  long L2CacheSize;
  long L2CacheType;
  long L2CacheSpeed;
  long SYSRamBK0Size;
  long SYSRamBK0Type;
  long SYSRamBK1Size;
  long SYSRamBK1Type;
  long SYSRamBK2Size;
  long SYSRamBK2Type;
  long SYSRamBK3Size;
  long SYSRamBK3Type;
  long DRAMType;
  long Fabrication[2];
  long Test[2];
  long FactoryCode;
  long TestCode;
  long EnvCode;
  long Magic[2];
  long DRAMSize;
  long SYSRamBK4Size;
  long SYSRamBK4Type;
  long SYSRamBK5Size;
  long SYSRamBK5Type;
  long SYSRamBK6Size;
  long SYSRamBK6Type;
  long SYSRamBK7Size;
  long SYSRamBK7Type;
  long EthAddr[2][6];
  long reserved[9]; /* Room to expand... */
  long checksum;
};
```

At NVRAM_OS_OFFSET, parameters used to boot the operating system (`boot_device`, `filename`, etc.) are stored. Some of these parameters are not used by PPCMon itself, but are passed onto the OS kernel (e.g `inet_mount`).

```
struct os_nvram
{
   long magic;
   short size; short checksum;
   long host_inet_mask;
   long host_inet_bplane_mask;
   short boot_status; char boot_ctl; char root_ctl;
   long reboot_flags;
   long boot_device;
   long root_device;
   long boot_protocol; /* Also used for ethernet speed */
   short boot_delay; char boot_fast; unsigned char boot_filetype;
   long boot_eth_p;
   long boot_addr;
   long host_inet;
   long server_inet;
   long broadcast_inet;
   long mask_inet;
   char boot_file[64];
   long boot_size;
   char inet_mount[48];
   struct bootp bootp_reply;
   long intr_mask;
   long intr_debug;
   unsigned char eth_addr[6];
   char b_rsv[1];
   unsigned char boot_cpunr;
   long host_inet_bplane;
   long cache_ctl;
   long bma_mode;
   struct dma_chan
   {
      unsigned char slot;
      unsigned char func;
      unsigned char tree;
      unsigned char bus;
      char *script;
   } dma[4];
   char boot_line[128];
   char boot_device_2[64];
   char boot_file_2[128];
   char padding[100]; /* to adjust struct size to 0x3c0 */
};
```