RedBoot™ User's Guide

Document Version 1.5, May 2001

Copyright

Warranty

eCos is open source software, covered by the Red Hat eCos Public License, and you are welcome to change it and/or distribute copies of it under certain conditions. This version of eCos is supported for customers of Red Hat. See http://sources.redhat.com/ecos/license-overview.html

For non-customers, eCos software has NO WARRANTY.

Because this software is licensed free of charge, there are no warranties for it, to the extent permitted by applicable law. Except when otherwise stated in writing, the copyright holders and/or other parties provide the software “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the software is with you. Should the software prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event, unless required by applicable law or agreed to in writing, will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

How to Contact Red Hat

Red Hat Corporate Headquarters
2600 Meridian Parkway
Durham NC 27713 USA
Telephone (toll free): +1 888 REDHAT 1 (+1 888 733 4281)
Telephone (main line): +1 919 547 0012
Telephone (FAX line): +1 919 547 0024
Website: http://www.redhat.com/

Contents

RedBoot™ User's Guide

1 Getting Started with RedBoot

RedBoot™ is an acronym for "Red Hat Embedded Debug and Bootstrap", and is the standard embedded system debug/bootstrap environment from Red Hat, replacing the previous generation of debug firmware: CygMon and GDB stubs. It provides a complete bootstrap environment for a range of embedded operating systems, such as embedded Linux and eCos™, and includes facilities such as network downloading and debugging. It also provides a simple flash file system for boot images.

RedBoot provides a wide set of tools for downloading and executing programs on embedded target systems, as well as tools for manipulating the target system's environment. It can be used for both product development (debug support) and for end product deployment (flash and network booting).

Here are some highlights of RedBoot's capabilities:

- Boot scripting support
- Simple command line interface for RedBoot configuration and management, accessible via serial (terminal) or Ethernet (telnet)
- Integrated GDB stubs for connection to a host-based debugger via serial or ethernet. (Ethernet connectivity is limited to local network only)
- Attribute Configuration - user control of aspects such as system time and date (if applicable), default Flash image to boot from, default failsafe image, static IP address, etc.
- Configurable and extensible, specifically adapted to the target environment
- Network bootstrap support including setup and download, via BOOTP, DHCP and TFTP
- X/YModem support for image download via serial
- Power On Self Test

Although RedBoot is derived from Red Hat eCos, it may be used as a generalized system debug and bootstrap control software for any embedded system and any operating system. For example, with appropriate additions, RedBoot could replace the commonly used BIOS of PC (and certain other) architectures. Red Hat is currently installing RedBoot on all embedded platforms as a standard practice, and RedBoot is now generally included as part of all Red Hat Embedded Linux and eCos ports. Users who specifically wish to use RedBoot with the eCos operating system should refer to the *Getting Started with eCos* document, which provides information about the portability and extendability of RedBoot in an eCos environment.

This chapter explains how to install and configure RedBoot for your target.

1.1 Installing RedBoot

To install the RedBoot package, follow the procedures detailed in the accompanying README.

Although there are other possible configurations, RedBoot is usually run from the target platform's flash boot sector or boot ROM, and is designed to run when your system is initially powered on. The method used to install the RedBoot image into non-volatile storage varies from platform to platform. In general, it requires that the image be programmed into flash in situ or programmed into the flash or ROM using a device programmer. In some cases this will be done at manufacturing

time; the platform being delivered with RedBoot already in place. In other cases, you will have to program RedBoot into the appropriate device(s) yourself. Installing to flash in situ may require special cabling or interface devices and software provided by the board manufacturer. The details of this installation process for a given platform will be found in Installation and Testing. Once installed, user-specific configuration options may be applied, using the `fconfig` command, providing that persistent data storage in flash is present in the relevant RedBoot version. See Section 1.3 for details.

1.2 User Interface

RedBoot provides a command line user interface (CLI). At the minimum, this interface is normally available on a serial port on the platform. If more than one serial interface is available, RedBoot is normally configured to try to use any one of the ports for the CLI. Once command input has been received on one port, that port is used exclusively until reset. If the platform has networking capabilities, the RedBoot CLI is also accessible using the `telnet` access protocol. By default, RedBoot runs `telnet` on port TCP/9000, but this is configurable and/or settable by the user.

RedBoot also contains a set of GDB "stubs", consisting of code which supports the GDB remote protocol. GDB stub mode is automatically invoked when the '$' character appears as the first character of a command line. The platform will remain in GDB stub mode until explicitly disconnected (via the GDB protocol). The GDB stub mode is available regardless of the connection method; either serial or network. Note that if a GDB connection is made via the network, then special care must be taken to preserve that connection when running user code. eCos contains special network sharing code to allow for this situation, and can be used as a model if this methodology is required in other OS environments.

1.3 Configuring the RedBoot Environment

Once installed, RedBoot will operate fairly generically. However, there are some features that can be configured for a particular installation. These depend primarily on whether flash and/or networking support are available. The remainder of this discussion assumes that support for both of these options is included in RedBoot.

1.3.1 Target Network Configuration

Each node in a networked system needs to have a unique address. Since the network support in RedBoot is based on TCP/IP, this address is an IP (Internet Protocol) address. There are two ways for a system to "know" its IP address. First, it can be stored locally on the platform. This is known as having a static IP address. Second, the system can use the network itself to discover its IP address. This is known as a dynamic IP address. RedBoot supports this dynamic IP address mode by use of the BOOTP (a subset of DHCP) protocol. In this case, RedBoot will ask the network (actually some generic server on the network) for the IP address to use.

NOTE

Currently, RedBoot only supports BOOTP. In future releases, DHCP may also be supported, but such support will be limited to additional data items, not lease-based address allocation.

The choice of IP address type is made via the `fconfig` command. Once a selection is made, it will be stored in flash memory. RedBoot only queries the flash configuration information at reset, so any changes will require restarting the platform.

Here is an example of the RedBoot `fconfig` command, showing network addressing:

```
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.29
Default server IP address: 192.168.1.101
GDB connection port: 9000
Network debug at boot time: false
```

In this case, the board has been configured with a static IP address listed as the Local IP address. The default server IP address specifies which network node to communicate with for TFTP service. This address can be overridden directly in the TFTP commands.

If the selection for `Use BOOTP for network configuration` had been `true`, these IP addresses would be determined at boot time, via the BOOTP protocol. The final number which needs to be configured, regardless of IP address selection mode, is the `GDB connection port`. RedBoot allows for incoming commands on either the available serial ports or via the network. This port number is the TCP port that RedBoot will use to accept incoming connections.

These connections can be used for GDB sessions, but they can also be used for generic RedBoot commands. In particular, it is possible to communicate with RedBoot via the telnet protocol. For example, on Linux®:

```
% telnet redboot_board 9000
Connected to redboot_board
Escape character is '^]'.
RedBoot>
```

1.3.2 Host Network Configuration

RedBoot may require two different classes of service from a network host:

- dynamic IP address allocation, using BOOTP
- TFTP service for file downloading

Depending on the host system, these services may or may not be available or enabled by default. See your system documentation for more details.

In particular, on Red Hat Linux, neither of these services will be configured out of the box. The following will provide a limited explanation of how to set them up. These configuration setups must be done as `root` on the host or server machine.

1.3.2.1 Enable TFTP on Red Hat Linux 6.2

1. Ensure that you have the tftp-server RPM package installed. By default, this installs the TFTP server in a disabled state. These steps will enable it:
2. Make sure that the following line is uncommented in the control file `/etc/inetd.conf`

```
tftp dgram   udp     wait    root    /usr/sbin/tcpd      /usr/sbin/in.tftpd
```

3. If it was necessary to change the line in Step 2, then the inetd server must be restarted, which can be done via the command:

```
# service inet reload
```

1.3.2.2 Enable TFTP on Red Hat Linux 7

1. Ensure that the xinetd RPM is installed.
2. Ensure that the tftp-server RPM is installed.
3. Enable TFTP by means of the following:

```
/sbin/chkconfig tftp on
```

Reload the xinetd configuration using the command:

```
/sbin/service xinetd reload
```

Create the directory /tftpboot using the command

```
mkdir /tftpboot
```

NOTE

Under Red Hat 7 you must address files by absolute pathnames, for example: `/tftp-boot/boot.img` not `/boot.img`, as you may have done with other implementations.

1.3.2.3 Enable BOOTP/DHCP server on Red Hat Linux

First, ensure that you have the proper package, `dhcp` (not `dhcpd`) installed. The DHCP server provides Dynamic Host Configuration, that is, IP address and other data to hosts on a network. It does this in different ways. Next, there can be a fixed relationship between a certain node and the data, based on that node's unique Ethernet Station Address (ESA, sometimes called a MAC address). The other possibility is simply to assign addresses that are free. The sample DHCP configuration file shown does both. Refer to the DHCP documentation for more details.

```
--------------- /etc/dhcpd.conf -----------------------------
default-lease-time 600;
max-lease-time 7200;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option domain-name-servers 198.41.0.4, 128.9.0.107;
option domain-name "bogus.com";
allow bootp;
shared-network BOGUS {
subnet 192.168.1.0 netmask 255.255.255.0 {
```

```
        option routers 192.168.1.101;
        range 192.168.1.1 192.168.1.254;
}
 }
host mbx {
        hardware ethernet 08:00:3E:28:79:B8;
        fixed-address 192.168.1.20;
        filename "/tftpboot/192.168.1.21/zImage";
        default-lease-time -1;
        server-name "srvr.bugus.com";
        server-identifier 192.168.1.101;
        option host-name "mbx";
}
```

Once the DHCP package has been installed and the configuration file set up, type:

```
# service dhcpd start
```

1.3.2.4 RedBoot network gateway

RedBoot cannot communicate with machines on different subnets because it does not support routing. It always assumes that it can get to an address directly, therefore it always tries to ARP and then send packets directly to that unit. This means that whatever it talks to must be on the same subnet. If you need to talk to a host on a different subnet (even if it's on the same 'wire'), you need to go through an ARP proxy, providing that there is a Linux box connected to the network which is able to route to the TFTP server. For example: `/proc/sys/net/ipv4/conf/<interface>/proxy_arp` where <*interface*>should be replaced with whichever network interface is directly connected to the board.

1.3.3 Verification

Once your network setup has been configured, perform simple verification tests as follows:

- Reboot your system, to enable the setup, and then try to 'ping' the target board from a host.
- Once communication has been established, try using the RedBoot load command to download a file from a host.

2 RedBoot Commands and Examples

2.1 Introduction

RedBoot provides three basic classes of commands:

- Program loading and execution
- flash image and configuration management
- Miscellaneous commands

Given the extensible and configurable nature of eCos and RedBoot, there may be extended or enhanced sets of commands available.

The basic format for commands is:

```
RedBoot> COMMAND [-S] [-s val]operand
```

Commands may require additional information beyond the basic command name. In most cases this additional information is optional, with suitable default values provided if they are not present. The type of information required affects how it is specified:

`[-S]`

An optional switch. If this switch is present, then some particular action will take place. For example in the command

```
RedBoot> fis init -f
```

the -f switch indicates to perform a full file system initialization.

`[-s val]`

An optional switch which requires an associated value. For example the command:

```
RedBoot> load -b 0x00100000 data_file
```

specifies downloading a file (via TFTP) into memory, relocating it to location 0x00100000.

`operand`

This format is used in a case where a command has one operand which must always be present (no -s is required since it is always implied). For example the command

```
RedBoot> go 0x10044
```

specifies executing the code starting at location 0x10044.

The list of available commands, and their syntax, can be obtained by typing `help` at the command line:

```
RedBoot> help
Manage machine caches
      cache [ON | OFF]
Display (hex dump) a range of memory
      dump -b <location> [-l <length>]
Manage flash images
      fis {cmds}
Manage configuration kept in FLASH memory
      fconfig [-l] [-n] [-f] | nickname [value]
```

```
Execute code at a location
      go [-w <timeout>] [entry]
Help about help?
      help [<topic>]
Load a file
      load [-r] [-v] [-d] [-h <host>] [-m {TFTP | xyzMODEM}]
      [-b <base_address>] <file_name>
Network connectivity test
      ping [-v] [-n <count>] [-t <timeout>] [-i <IP_addr]
      -h <IP_addr>
Reset the system
      reset
Display RedBoot version information
      version
```

Commands can be abbreviated to their shortest unique string. Thus in the list above, `d,du,dum` and dump are all valid for the dump command. The `fconfig` command can be abbreviated `fc`, but `f` would be ambiguous with `fis`.

There is one additional, special command. When RedBoot detects $ as the first character in a command, it switches to GDB protocol mode. At this point, the eCos GDB stubs take over, allowing connections from a GDB host. The only way to get back to RedBoot from GDB mode is to restart the platform.

The standard RedBoot command set is structured around the bootstrap environment. These commands are designed to be simple to use and remember, while still providing sufficient power and flexibility to be useful. No attempt has been made to render RedBoot as the end-all product. As such, things such as the debug environment are left to other modules, such as GDB stubs, which are typically included in RedBoot.

The command set may be also be extended on a platform basis.

2.2 RedBoot Editing Commands

RedBoot uses the following line editing commands.

- **Delete** (0x7F) or **Backspace** (0x08) moves the cursor back one character and erases what is there destructively.

  The mention of ^ is in the context of the `fconfig` command. This command uses some special line-editing features. When certain characters appear alone on the input line, a behavior is elicited.
- ^ (caret) switch to editing the previous item in the `fconfig` list. If fconfig edits item A, followed by item B, pressing ^ when changing item B, allows you to change item A. This is similar to the up arrow.
- . (period) stop editing any further items. This does not change the current item.
- **Return** (blank line) leaves the value for this item unchanged. Currently it is not possible to step through the value for the start-up script; it must always be retyped.

2.3 Common Commands

The general format of commands is:

```
command <options, parameters>
```

Elements are separated by the space character. Other control characters, such as **Tab** or editing keys (**Insert**) are not currently supported.

Numbers, such as a memory location, may be specified in either decimal or hexadecimal (requires a 0x prefix).

Commands may be abbreviated to any unique string. For example, `lo` is equivalent to `loa` and `load`.

2.3.1 Connectivity

ping - Check network connectivity ping

```
ping [-v] [-n <count>] [-l <length>] [-t <timeouts>] [-r
<rate>][-i <IP_addr>] -h <IP_addr>
```

The ping command checks the connectivity of the local network by sending special (ICMP) packets to a specific host. These packets should be automatically returned by that host. The command will indicate how many of these round-trips were successfully completed.

Arguments

-v	Be verbose, displaying information about each packet sent.
-n <count>	Controls the number of packets to be sent. Default is 10 if -n is not specified.
-t <timeout>	How long to wait for the round-trip to complete, specified in milliseconds. Default is 1000ms (1 second).
-r <rate>	How fast to deliver packets, i.e. time between successive sends. Default is 1000ms (1 second). Specifying "-r 0" will send packets as quickly as possible.
-l <length>	Each packet contains some amount of payload data. This option specifies the length of that data. The default is 64 and the value is restricted to the range 64 .. 1400.
-i <local IP>	This allows the ping command to override its local network address. While this is not recommended procedure, it can help diagnose some situations, for example where BOOTP is not working properly.
-h <host IP>	The address of the other device to contact.

2.3.2 General

Manage machine caches.

cache [ON | OFF]

With no options, this command specifies the state of the system caches.

When an option is given, the caches are turned off or on appropriately.

version

Display RedBoot version information.

This command simply displays version information about RedBoot.

```
RedBoot> version
RedBoot(tm) debug environment - built 09:12:03, Feb 12 2001
Platform: XYZ (PowerPC 860)
Copyright (C) 2000, 2001, Red Hat, Inc.
RAM: 0x00000000-0x00400000
RedBoot>
```

reset

Reset the system.

This command resets the platform. This should be equivalent to a power-on reset.

dump -b <location> [-l <length>]

Display (hex dump) a range of memory.

This command displays the contents of memory in hexadecimal format. It is most useful for examining a segment of RAM or flash. Note that it could be detrimental if used on memory mapped hardware registers.

The memory is displayed at most sixteen bytes per line, first as the raw hex value, followed by an ASCII interpretation of the data.

```
RedBoot> du -b 0x100 -l 0x80
0x00000100: 3C60 0004 6063 2000 7C68 03A6 4E80 0020 |<‘..‘c .|h..N.. |
0x00000110: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000120: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000130: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000140: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000150: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000160: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000170: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
RedBoot> d -b 0xfe00b000 -l 0x80
0xFE00B000: 2025 700A 0000 0000 4174 7465 6D70 7420 | %p.....Attempt |
0xFE00B010: 746F 206C 6F61 6420 532D 7265 636F 7264 |to load S-record|
0xFE00B020: 2064 6174 6120 746F 2061 6464 7265 7373 | data to address|
0xFE00B030: 3A20 2570 205B 6E6F 7420 696E 2052 414D |: %p [not in RAM|
0xFE00B040: 5D0A 0000 2A2A 2A20 5761 726E 696E 6721 |]...*** Warning!|
0xFE00B050: 2043 6865 636B 7375 6D20 6661 696C 7572 | Checksum failur|
0xFE00B060: 6520 2D20 4164 6472 3A20 256C 782C 2025 |e - Addr: %lx, %|
0xFE00B070: 3032 6C58 203C 3E20 2530 326C 580A 0000 |02lX <> %02lX...|
0xFE00B080: 456E 7472 7920 706F 696E 743A 2025 702C |Entry point: %p,|
```

cksum -b <location> -l <length>

Computes the POSIX checksum on a range of memory (either RAM or FLASH). The value printed can be compared with the output from the Linux program ’chksum’.

2.3.3 Download Process

load

The `load` command is used to download data into the target system. Data can be loaded via a network connection, using either the TFTP protocol, or the console serial connection using the X/Y modem protocol. Files to be downloaded may either be executable images in SREC format or raw data. The format of the command is:

load {*file*} [-v] [-d] [-b *location*] [-r] [-m {[xmodem]|[ymodem]|[tftp]}] [-h *host_IP_address*]

Arguments

file	The name of the file on the TFTP server. Details of how this is specified are host-specific.
-v	Display a small spinner (indicator) while the download is in progress. This is just for feedback, especially during long loads.
-d	Decompress gzipped image during download.
-b	Specify the location in memory to which the file should be loaded. Executable images normally load at the location to which the file was linked. This option allows the file to be loaded to a specific memory location, possibly overriding any assumed location.
-r	Download raw data. Normally, the load command is used to load executable images into memory. This option allows for raw data to be loaded. If this option is given, -b will also be required.
-m	The -m option is used to select the download method. The choices are: *xmodem*, *ymodem* - serial download using standard protocols over the console serial port. When using this method, the *file* parameter is not required. *tftp* - network based download using the TFTP protocol.
-h	Used explicitly to name a host computer to contact for the download data. This works in TFTP mode only.

```
==============================================================
RedBoot> lo redboot.ROM -b 0x8c400000
Address offset = 0x0c400000
Entry point: 0x80000000, address range: 0x80000000-0x8000fe80
==============================================================
```

2.4 Flash Image System (FIS)

If the platform has flash memory, RedBoot can use this for image storage. Executable images, as well as data, can be stored in flash in a simple file store. The `fis` command is used to manipulate and maintain flash images.

The available `fis` commands are:

fis init [-f]

This command is used to initialize the flash Image System (FIS). It should only be executed once, when RedBoot is first installed on the hardware. Subsequent executions will cause loss of data in the flash (previously saved images will no longer be accessible).

If the `-f` option is specified, all blocks of flash memory will be erased as part of this process.

```
==============================================================
RedBoot> fis init -f
About to initialize [format] flash image system - are you sure (y/n)? n
==============================================================
```

fis [-c] [-d] list

This command lists the images currently available in the FIS. Certain images used by RedBoot have fixed names. Other images can be manipulated by the user.

If the *-c* option is specified, the image checksum is displayed instead of the `Mem Addr` field.

If the *-d* option is specified, the image `datalength` is displayed instead of the length [amount of flash used]. The `datalength` is the length of data within the allocated flash image actually being used for data.

```
==============================================================
RedBoot> fis list
Name           flash addr  Mem addr   Length   Entry point
RedBoot        0xA0000000  0xA0000000 0x020000 0x80000000
RedBoot[backup]0xA0020000  0x8C010000 0x010000 0x8C010000
RedBoot config 0xA0FC0000  0xA0FC0000 0x020000 0x00000000
FIS directory  0xA0FE0000  0xA0FE0000 0x020000 0x00000000
RedBoot> fis list -c
Name           flash addr  Checksum   Length   Entry point
RedBoot        0xA0000000  0x34C94A57 0x020000 0x80000000
RedBoot[backup]0xA0020000  0x00000000 0x010000 0x8C010000
RedBoot config 0xA0FC0000  0x00000000 0x020000 0x00000000
RedBoot config 0xA0FE0000  0x00000000 0x020000 0x00000000
==============================================================
```

fis free

This command shows which areas of the flash memory are currently not in use. In use means that the block contains non-erased contents. Since it is possible to force an image to be loaded at a particular flash location, this command can be used to check whether that location is in use by any other image.

NOTE

There is currently no cross-checking between actual flash contents and the image directory, which mans that there could be a segment of flash which is not erased that does not correspond to a named image, or vice-versa.

```
==============================================================
RedBoot> fis free
        0xA0040000 .. 0xA07C0000
        0xA0840000 .. 0xA0FC0000
==============================================================
```

fis create -b <mem_base> -l <length> [-f <flash_addr>] [-e <entry_point>] [-r <ram_addr>] [-s <data_length>] [-n] <name>

This command creates an image in the FIS directory. The data for the image must exist in RAM memory before the copy. Typically, you would use the RedBoot `load` command to load an image into RAM and then the `fis create` command to write it to flash.

Arguments

name	The name of the file, as shown in the FIS directory.
-b	The location in RAM used to obtain the image. This is a required option.
-l	The length of the image. If the image already exists, then the length is inferred from when the image was previously created. If specified, and the image exists, it must match the original value.
-f	The location in flash for the image, which will be inferred for extant images if not specified. If this is not provided, the first freeVblock which is large enough will be used. See `fis free`.
-e	The execution entry address. This is used if the starting address for an image is not known, or needs to be overridden.
-r	The location in RAM when the image is loaded via `fis load`. This only needs to be specified for images which will eventually loaded via `fis load`. Fixed images, such as RedBoot itself, will not need this.
-s	The length of the actual data to be written to flash. If not present then the image length (-1) value is assumed. If the value given by -s is less than -1, the remainder of the image in flash will be left in an erased state. Note that by using this option it is possible to create a completely empty flash image, for example to reserve space for use by applications other than RedBoot.
-n	If -n is specified, then only the FIS directory is updated, and no data is copied from RAM to flash. This feature can be used to recreate the FIS entry if it has been destroyed.

```
==========================================================================
RedBoot> fis create RedBoot -f 0xa0000000 -b 0x8c400000 -l 0x20000
An image named 'RedBoot' exists - are you sure (y/n)? n
RedBoot> fis create junk -b 0x8c400000 -l 0x20000
... Erase from 0xa0040000-0xa0060000: .
... Program from 0x8c400000-0x8c420000 at 0xa0040000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
==========================================================================
```

If you are loading an existing file, then the fis create command will provide some values automatically, such as the flash address and flash length.

fis load [-b <memory load address>] [-c] [-d] name

This command is used to transfer an image from flash memory to RAM.

Once loaded, it may be executed using the go command. If -b is specified, then the image is copied from flash to the specified address in RAM. If -b is not specified, the image is copied from flash to the load address given when the image was created.

Arguments

name	The name of the file, as shown in the FIS directory
-b	Specify the location in memory to which the file should be loaded. Executable images normally load at the location to which the file was linked. This option allows the file to be loaded to a specific memory location, possibly overriding any assumed location.
-c	Compute and print the checksum of the image data after it has been loaded into memory.
-d	Decompress gzipped image while copying it from flash to RAM.

```
==========================================================================
RedBoot> fis load RedBoot[backup]
RedBoot> go
==========================================================================
```

fis delete name

This command removes an image from the FIS. The flash memory will be erased as part of the execution of this command, as well as removal of the name from the FIS directory.

```
==========================================================================
RedBoot> fis list
Name              flash addr   Mem addr    Length    Entry point
RedBoot          0xA0000000   0xA0000000  0x020000  0x80000000
RedBoot[backup]  0xA0020000   0x8C010000  0x020000  0x8C010000
RedBoot config   0xA0FC0000   0xA0FC0000  0x020000  0x00000000
FIS directory    0xA0FE0000   0xA0FE0000  0x020000  0x00000000
junk             0xA0040000   0x8C400000  0x020000  0x80000000
RedBoot> fis delete junk
Delete image 'junk' - are you sure (y/n)? y
... Erase from 0xa0040000-0xa0060000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
==========================================================================
```

fis lock -f <flash_addr> -l <length>

This command is used to write-protect (lock) a portion of flash memory, to prevent accidental overwriting of images. In order to make make any modifications to the flash, a matching unlock command must be issued. This command is optional and will only be provided on hardware which can support write-protection of the flash space.

NOTE

Depending on the system, attempting to write to write-protected flash may generate errors or warnings, or be benignly quiet.

```
==========================================================================
RedBoot fis lock -f 0xa0040000 -l 0x20000
... Lock from 0xa0040000-0xa0060000: .
==========================================================================
```

fis unlock -f <flash_addr> -l <length>

This command is used to unlock a portion of flash memory forcibly, allowing it to be updated. It must be issued for regions which have been locked before the FIS can reuse those portions of flash.

```
==========================================================================
RedBoot fis unlock -f 0xa0040000 -l 0x20000
... Unlock from 0xa0040000-0xa0060000: .
==========================================================================
```

fis erase -f <flash_addr> -l <length>

This command is used to erase a portion of flash memory forcibly. There is no cross-checking to ensure that the area being erased does not correspond to a loaded image.

```
==========================================================================
RedBoot> fis erase -f 0xa0040000 -l 0x20000
... Erase from 0xa0040000-0xa0060000: .
==========================================================================
```

fis write -b <location> -l <length> -f <flash addr>

Writes data from RAM at <location> to flash.

2.5 Persistent State Flash-based Configuration and Control

RedBoot provides flash management support for storage in the flash memory of multiple executable images and of non-volatile information such as IP addresses and other network information.

RedBoot on platforms that support flash based configuration information will report the following message the first time that RedBoot is booted on the target:

```
flash configuration checksum error or invalid key
```

This error can be ignored if no flash based configuration is desired, or can be silenced by running the `fconfig` command as described below. At this point you may also wish to run the `fis init` command. See other fis commands in Section 2.4.

Certain control and configuration information used by RedBoot can be stored in flash.

The details of what information is maintained in flash differ, based on the platform and the configuration. However, the basic operation used to maintain this information is the same. Using the `fconfig -l` command, the information may be displayed and/or changed.

If the optional flag `-l` is specified, the configuration data is simply listed. Otherwise, each configuration parameter will be displayed and you are given a chance to change it. The entire value must be typed - typing just carriage return will leave a value unchanged. Boolean values may be entered using the first letter (`t` for true, `f` for false). At any time the editing process may be stopped simply by entering a period (.) on the line. Entering the caret (^) moves the editing back to the previous item. See “RedBoot Editing Commands”, Section 2.2.

If any changes are made in the configuration, then the updated data will be written back to flash after getting acknowledgement from the user.

If the optional flag `-n` is specified (with or without `-l`) then “nicknames” of the entries are used. These are shorter and less descriptive than “full” names. The full name may also be displayed by adding the `-f` flag.

The reason for telling you nicknames is that a quick way to set a single entry is provided, using the format

```
RedBoot> fconfig nickname value
```

If no value is supplied, the command will list and prompt for only that entry. If a value is supplied, then the entry will be set to that value. You will be prompted whether to write the new information into flash if any change was made. For example

```
==========================================================================
RedBoot> fconfig -l -n
boot_script: false
bootp: false
bootp_my_ip: 10.16.19.176
bootp_server_ip: 10.16.19.66
gdb_port: 9000
net_debug: false
RedBoot> fconfig bootp_my_ip 10.16.19.177
bootp_my_ip: 10.16.19.176 Setting to 10.16.19.177
Update RedBoot non-volatile configuration - are you sure (y/n)? y
... Unlock from 0x507c0000-0x507e0000: .
... Erase from 0x507c0000-0x507e0000: .
... Program from 0x0000a8d0-0x0000acd0 at 0x507c0000: .
... Lock from 0x507c0000-0x507e0000: .
RedBoot>
==========================================================================
```

One item which is always present in the configuration data is the ability to execute a script at boot time. A sequence of RedBoot commands can be entered which will be executed when the system starts up. Optionally, a time-out period can be provided which allows the user to abort the startup script and proceed with normal command processing from the console.

```
==========================================================================
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.29
Default server IP address: 192.168.1.101
GDB connection port: 9000
Network debug at boot time: false
==========================================================================
```

The following example sets a boot script and then shows it running.

```
==========================================================================
RedBoot> fconfig
Run script at boot: false t
      Boot script:
Enter script, terminate with empty line
>> fi li
    Boot script timeout: 0 10
Use BOOTP for network configuration: false .
Update RedBoot non-volatile configuration - are you sure (y/n)? y
... Erase from 0xa0fc0000-0xa0fe0000: .
... Program from 0x8c021f60-0x8c022360 at 0xa0fc0000: .
RedBoot>
RedBoot(tm) debug environment - built 08:22:24, Aug 23 2000
```

```
Copyright (C) 2000, Red Hat, Inc.


RAM: 0x8c000000-0x8c800000
flash: 0xa0000000 - 0xa1000000, 128 blocks of 0x00020000 bytes ea.
Socket Communications, Inc: Low Power Ethernet CF Revision C \
5V/3.3V 08/27/98 IP: 192.168.1.29, Default server: 192.168.1.101 \
== Executing boot script in 10 seconds - enter ^C to abort
RedBoot> fi li
Name              flash addr   Mem addr    Length    Entry point
RedBoot          0xA0000000   0xA0000000  0x020000  0x80000000
RedBoot[backup]  0xA0020000   0x8C010000  0x020000  0x8C010000
RedBoot config   0xA0FC0000   0xA0FC0000  0x020000  0x00000000
FIS directory    0xA0FE0000   0xA0FE0000  0x020000  0x00000000
RedBoot>
==============================================================
```

NOTE

The bold characters above indicate where something was entered on the console. As you can see, the `fi li` command at the end came from the script, not the console. Once the script is executed, command processing reverts to the console.

2.6 Executing Programs from RedBoot

Once an image has been loaded into memory, either via the `load` command or the `fis load` command, execution may be transfered to that image.

NOTE

The image is assumed to be a stand-alone entity, as RedBoot gives the entire platform over to it. Typical examples would be an eCos application or a Linux kernel.

go - Execute a program

The format of the `go` command is:

```
RedBoot> go [-w time] [location]
```

Execution will begin at `location` if specified. Otherwise, the entry point of the last image loaded will be used.

The `-w` option gives the user `time` seconds before execution begins. The execution may be aborted by typing **Ctrl+C** on the console. This mode would typically be used in startup scripts.

exec - Execute a Linux kernel image

NOTE

This command is not available for all platforms. Its availability is indicated in specific platform information in Chapter 5.

Arguments

```
[-w timeout]
[-b <load addr> [-l <length]]
[-r <ramdisk addr>
[-s <ramdisk length>]]
[-c "kernel command line"] [<entry_point>]
```

This command is used to execute a non-eCos application, typically a Linux kernel. Additional information may be passed to the kernel at startup time. This command is quite special (and unique from the ’go’ command) in that the program being executed may expect certain environmental setups, for example that the MMU is turned off, etc.

The Linux kernel expects to have been loaded to a particular memory location (0xC0008000 in the case of the SA1110). Since this memory is used by RedBoot internally, it is not possible to load the kernel to that location directly. Thus the requirement for the "-b" option which tells the command where the kernel has been loaded. When the exec command runs, the image will be relocated to the appropriate location before being started. The "-r" and "-s" options are used to pass information to the kernel about where a statically loaded ramdisk (initrd) is located.

The "-c" option can be used to pass textual "command line" information to the kernel. If the command line data contains any puncuation (spaces, etc), then it must be quoted using the double-quote character ’"’. If the quote character is required, it should be written as ’\"’.

3 Rebuilding RedBoot

3.1 Introduction

In normal circumstances it is only necessary to rebuild RedBoot if it has been modified, for example if you have extended the command set or applied patches. See the *Getting Started with eCos* document, which provides information about the portability and extendability of RedBoot in an eCos environment.

Most platform HALs provide configuration export files. Before proceding with the following procedures, check “Configuration export files”, Section 3.1.1 first, which may simplify the process for your platform.

RedBoot is configured and built using configuration technology based on Configuration Description Language (CDL). The detailed instructions for building the command-line tool `ecosconfig` on Linux can be found in host/README. For example:

```
mkdir $TEMP/redboot-build
cd $TEMP/redboot-build
$ECOSDIR/host/configure --prefix=$TEMP/redboot-build --with-tcl=/usr
make
```

The simplest version of RedBoot can be built by setting the environment variable ECOS_REPOSITORY to point at the eCos/RedBoot source tree, and then typing:

```
ecosconfig new TARGET redboot
ecosconfig tree
make
```

where TARGET is the eCos name for the desired platform, for example assabet. You will need to have set the environment variable ECOS_REPOSITORY to point at the eCos/RedBoot source tree. Values of TARGET for each board are given in the specific installation details for each board in Chapter 5, *Installation and Testing*.

The above command sequence would build a very simple version of RedBoot, and would not include, for example, networking, FLASH or Compact Flash Ethernet support on targets that supported those. Such features could be included with the following commands:

```
ecosconfig new TARGET redboot
ecosconfig add flash
ecosconfig add pcmcia net_drivers cf_eth_drivers
ecosconfig tree
make
```

In practice, most platform HALs include configuration export files, described in Section 3.1.1, to ensure that the correct configuration of RedBoot has been chosen to avoid needing to worry about which extra packages to add.

The above commands would build a version of RedBoot suitable for testing. In particular, the result will run from RAM. Since RedBoot normally needs to be installed in ROM/flash, type the following:

```
cat >RedBoot_ROM.ecm <<EOF
cdl_component CYG_HAL_STARTUP {
       user_value ROM
```

```
};
EOF
ecosconfig import RedBoot_ROM.ecm
ecosconfig tree
make
```

This set of commands will adjust the configuration to be ROM oriented.

Each of these command sequences creates multiple versions of RedBoot in different file formats. The choice of which file to use will depend upon the actual target hardware and the tools available for programming ROM/flash. The files produced (typically) are:

`install/bin/redboot.elf` This is the complete version of RedBoot, represented in ELF format. It is most useful for testing with tools such as embedded ICE, or other debug tools.

`install/bin/redboot.srec` This version has been converted to Motorola S-record format.

`install/bin/redboot.bin` This version has been flattened; that is, all formatting information removed and just the raw image which needs to be placed in ROM/flash remains.

The details of putting the RedBoot code into ROM/flash are target specific. Once complete, the system should come up with the RedBoot prompt. For example, the version built using the commands above looks like:

```
RedBoot(tm) debug environment - built 07:54:25, Oct 16 2000
Platform: Assabet development system (StrongARM 1110)
Copyright (C) 2000, Red Hat, Inc.
RAM: 0x00000000-0x02000000
flash: 0x50000000 - 0x50400000, 32 blocks of 0x00020000 bytes ea.
Socket Communications, Inc: Low Power Ethernet CF Revision C
5V/3.3V 08/27/98
IP: 192.168.1.29, Default server: 192.168.1.101
RedBoot>
```

3.1.1 Configuration export files

To help with rebuilding RedBoot from source, some platforms HALs provide configuration export files. First locate the configuration export files for your platform in the eCos source repository. The RAM and ROM startup configuration exports can usually be found in a directory named "misc" in the platform HAL in the eCos source repository, named:

```
1432 Feb  1 13:27 misc/redboot_RAM.ecm
1487 Feb  1 14:38 misc/redboot_ROM.ecm
```

All dates and sizes are just examples.

3.1.1.1 Making RedBoot for RAM startup

Throughout the following instructions, several environmental variables are referred to:

$REDBOOTDIR

Full path to the toplevel RedBoot source release.

$BUILDDIR

Full path to where RedBoot will be built, e.g. `redboot.RAM`.

$ECOS_REPOSITORY

Full path to the RedBoot package source. Typically, this should be **$REDBOOTDIR**`/packages`.

$TARGET

e.g.atlas_mips32_4kc.

$ARCH_DIR

The directory for the architecture, e.g. mips.

$PLATFORM_DIR

The directory for the platform, e.g. atlas.

You must make sure these variables are correctly set in your environment before proceeding, or the build will fail. The values for **$TARGET**, **$ARCH_DIR** and **$PLATFORM_DIR** for each board are given in the specific installation details for each board in Chapter 5, *Installation and Testing*.

With the environment variables set, use the following sequence of commands to build a RedBoot image suitable for loading into RAM:

```
mkdir $BUILDDIR
cd $BUILDDIR
ecosconfig new $TARGET redboot
ecosconfig import \
  ${ECOS_REPOSITORY}/hal/${ARCH_DIR}/${PLATFORM_DIR}/current/misc/redboot_RAM.ecm
ecosconfig tree
make
```

To build the ROM version, in a different build/config directory, just use the configuration export file `redboot_ROM.ecm` instead.

The resulting files will be, in each of the ROM and RAM startup build places:

```
$BUILDDIR/install/bin/redboot.bin
$BUILDDIR/install/bin/redboot.elf
$BUILDDIR/install/bin/redboot.img
$BUILDDIR/install/bin/redboot.srec
```

Some targets may have variations, or extra files generated in addition.

3.1.2 Platform specific instructions

The platform specific information in Chapter 5, *Installation and Testing* should be consulted, as there may be other special instructions required to build RedBoot for particular boards.

4 Updating RedBoot

4.1 Introduction

RedBoot normally runs from flash or ROM. In the case of flash, it is possible to update RedBoot, that is, replace it with a newer version, in situ. This process is complicated by the fact that RedBoot is running from the very flash which is being updated. The following is an outline of the steps needed for updating RedBoot:

- Start RedBoot, running from flash.
- Load and start a different version of RedBoot, running from RAM.
- Update the primary RedBoot flash image.
- Reboot; run RedBoot from flash.

In order to execute this process, two versions of RedBoot are required; one which runs from flash, and a separate one which runs solely from RAM. Both of these images are typically provided as part of the RedBoot package, but they may also be rebuilt from source using the instructions provided for the platform.

The following is a more detailed look at these steps. For this process, it is assumed that the target is connected to a host system and that there is some sort of serial connection used for the RedBoot CLI.

4.1.1 Start RedBoot, Running from flash

To start RedBoot, reset the platform.

4.1.2 Load and start a different version of RedBoot, running from RAM

There are a number of choices here. The basic case is where the RAM based version has been stored in the FIS (flash Image System). To load and execute this version, use the commands:

```
RedBoot> fis load RedBoot[backup]
RedBoot> go
```

If this image is not available, or does not work, then an alternate RAM based image must be loaded. Using the load command:

```
RedBoot> load redboot_RAM.srec
RedBoot> go
```

NOTE

The details of how to load are installation specific. The file must be placed somewhere the host computer can provide it to the target RedBoot system. Either TFTP (shown) or X/Ymodem can be used to download the image into RAM.

Once the image is loaded into RAM, it may be used to update the secondary RedBoot image in flash using the FIS commands. Some platforms support locking (write protecting) certain regions of the flash, while others do not. If your platform does not support the lock/unlock commands, simply ignore these steps. Again, the details of these commands (in particular the numeric values) differ on each target platform, but the ideas are the same:

```
RedBoot> fis unlock -f <flash addr> -l <flash length>
RedBoot> fis create RedBoot[backup] -f <flash addr> -b <flash source>
         -r <image addr> -l <flash length>
RedBoot> fis lock -f <flash addr> -l <flash length>
```

4.1.3 Update the primary RedBoot flash image

At this point, a new version of RedBoot is running on the target, in RAM.

Using the `load` command, download the flash based version from the host.

Since the flash version is designed to load and run from flash, the image must be relocated into some suitable, available, RAM location. The details of this are target platform specific (found in the target appendix), but the command will look something like this:

```
RedBoot> load redboot_ROM.srec -b <flash source>
```

This command loads the flash image into RAM at **`flash_source`**, using the TFTP protocol via a network connection. Other options are available, refer to the command section on `load` for more details.

Once the image is loaded into RAM, it must be placed into flash using the FIS commands. Some platforms support locking (write protecting) certain regions of the flash, while others do not. If your platform does not support the lock/unlock commands, simply ignore these steps. Again, the details of these commands (in particular the numeric values) differ on each target platform, but the ideas are the same:

```
RedBoot> fis unlock -f <flash addr> -l <flash length>
RedBoot> fis create RedBoot -f <flash addr> -b <flash source> -l <flash length>
         -s <data length>
RedBoot> fis lock -f <flash addr> -l <flash addr>
```

NOTE

RedBoot will display a number of lines of information as it executes these commands. Also, the size (-s) value for the create operation should be determined from the output provided as part of the file download step.

It is not required, but it does allow for improved image validity checking in the form of an image checksum.

4.1.4 Reboot; run RedBoot from flash

Once the image has been successfully written into the flash, simply reboot the target and the new version of RedBoot will be running.

NOTE

There may be times when RedBoot does not exist on the hardware, thus making step 1 impossible to do. In these cases, it should be possible to get to step 2 by using GDB. If this is possible, the appropriate steps are provided with the target documentation.

5 Installation and Testing

5.1 Cyclone IQ80310

5.1.1 Overview

RedBoot supports both serial ports and the built-in ethernet port for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the onboard 8MB flash. Several basic RedBoot configurations are supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from flash address 0x40000, with ARM bootloader in flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.
- RedBoot running from RAM with ARM bootloader in flash boot sector.

A special RedBoot command: `diag` is used to access a set of hardware diagnostics provided by the board manufacturer.

5.1.2 Initial Installation Method

The board manufacturer provides a DOS application which is capable of programming the flash over the PCI bus, and this is required for initial installations of RedBoot. Please see the board manual for information on using this utility. In general, the process involves programming one of the two flash based RedBoot configurations to flash. The RedBoot which runs from the flash boot sector should be programmed to flash address 0x00000000. RedBoot that has been configured to be started by the ARM bootloader should be programmed to flash address 0x00004000.

Four sets of prebuilt files are provided in a tarball and zip format. Each set corresponds to one of the four supported configurations and includes an ELF file (.elf), a binary image (.bin), and an S-record file (.srec).

```
For RedBoot running from the flash boot sector:
bins/cyclone-rom.bin
bins/cyclone-rom.elf
bins/cyclone-rom.srec


For RedBoot running from flash address 0x40000:
bins/cyclone-roma.bin
bins/cyclone-roma.elf
bins/cyclone-roma.srec


For RedBoot running from RAM with RedBoot in the flash boot sector:
bins/cyclone-ram.bin
bins/cyclone-ram.elf
bins/cyclone-ram.srec


For RedBoot running from RAM with ARM bootloader in the flash boot sector:
bins/cyclone-rama.bin
bins/cyclone-rama.elf
bins/cyclone-rama.srec
```

Initial installations deal with the flash-based RedBoots. Installation and use of RAM based RedBoots is documented elsewhere.

To install RedBoot to run from the flash boot sector, use the manufacturer's flash utility to install the bins/cyclone-rom.bin image at address zero.

To install RedBoot to run from address 0x40000 with the ARM bootloader in the flash boot sector, use the manufacturer's flash utility to install the bins/cyclone-roma.bin image at address 0x40000.

After booting the initial installation of RedBoot, this warning may be printed:

```
flash configuration checksum error or invalid key
```

This is normal, and indicates that the flash must be configured for use by RedBoot. Even if the above message is not printed, it may be a good idea to reinitialize the flash anyway. Do this with the `fis` command:

```
RedBoot> fis init
About to initialize [format] flash image system - are you sure (y/n)? y
*** Initialize flash Image System
Warning: device contents not erased, some blocks may not be usable
... Unlock from 0x007e0000-0x00800000: .
... Erase from 0x007e0000-0x00800000: .
... Program from 0xa1fd0000-0xa1fd0400 at 0x007e0000: .
... Lock from 0x007e0000-0x00800000: .
Followed by the fconfig command:
   RedBoot> fconfig
   Run script at boot: false
   Use BOOTP for network configuration: false
   Local IP address: 0.0.0.0 192.168.1.153
   Default server IP address: 0.0.0.0 192.168.1.10
   GDB connection port: 0 1000
   Network debug at boot time: false
   Update RedBoot non-volatile configuration - are you sure (y/n)? y
   ... Unlock from 0x007c0000-0x007e0000: .
   ... Erase from 0x007c0000-0x007e0000: .
   ... Program from 0xa0013018-0xa0013418 at 0x007c0000: .
   ... Lock from 0x007c0000-0x007e0000: .
```

5.1.3 Error codes

RedBoot uses the two digit LED display to indicate errors during board initialization. Possible error codes are:

```
88 - Unknown Error
55 - I2C Error
FF - SDRAM Error
01 - No Error
```

5.1.4 Using RedBoot with ARM Bootloader

RedBoot can coexist with ARM tools in flash on the IQ80310 board. In this configuration, the ARM bootloader will occupy the flash boot sector while RedBoot is located at flash address 0x40000. The sixteen position rotary switch is used to tell the ARM bootloader to jump to the RedBoot image located at address 0x40000. RedBoot is selected by switch position 0 or 1. Other switch positions are used by the ARM firmware and RedBoot will not be started.

5.1.5 Flash management

5.1.5.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

ARM bootloader in flash boot sector

```
-f 0x40000
-b 0xa0100000
-l 0x40000
```

RedBoot in flash boot sector

```
-f 0
-b 0xa0100000
-l 0x40000
```

5.1.5.2 Updating the secondary RedBoot image

ARM bootloader in flash boot sector

```
-f 0x80000
-b 0xa0020000
-r 0xa0020000
-l 0x40000
```

RedBoot in flash boot sector

```
-f 0x40000
-b 0xa0020000
-r 0xa0020000
-l 0x40000
```

5.1.6 Special RedBoot Commands

A special RedBoot command, diag, is used to access a set of hardware diagnostics provided by the board manufacturer. To access the diagnostic menu, enter diag at the RedBoot prompt:

```
RedBoot> diag
Entering Hardware Diagnostics - Disabling Data Cache!1 - Memory Tests
2 - Repeating Memory Tests
3 - 16C552 DUART Serial Port Tests
4 - Rotary Switch S1 Test for positions 0-3
5 - seven Segment LED Tests
6 - Backplane Detection Test
7 - Battery Status Test
8 - External Timer Test
9 - i82559 Ethernet Configuration
10 - i82559 Ethernet Test
11 - Secondary PCI Bus Test
12 - Primary PCI Bus Test
13 - i960Rx/303 PCI Interrupt Test
14 - Internal Timer Test
15 - GPIO Test
```

```
0 - quit Enter the menu item number (0 to quit):
```

Tests for various hardware subsystems are provided, and some tests require special hardware in order to execute normally. The Ethernet Configuration item may be used to set the board ethernet address.

5.1.7 IQ80310 Hardware Tests

```
1 - Memory Tests
2 - Repeating Memory Tests
3 - 16C552 DUART Serial Port Tests
4 - Rotary Switch S1 Test for positions 0-3
5 - seven Segment LED Tests
6 - Backplane Detection Test
7 - Battery Status Test
8 - External Timer Test
9 - i82559 Ethernet Configuration
10 - i82559 Ethernet Test
11 - Secondary PCI Bus Test
12 - Primary PCI Bus Test
13 - i960Rx/303 PCI Interrupt Test
14 - Internal Timer Test
15 - GPIO Test
0 - quit
Enter the menu item number (0 to quit):
```

Tests for various hardware subsystems are provided, and some tests require special hardware in order to execute normally. The Ethernet Configuration item may be used to set the board ethernet address.

5.1.8 Rebuilding RedBoot

The build process is nearly identical for the four supported configurations. Assuming that the provided RedBoot source tree is located in the current directory and that we want to build a RedBoot that runs from the flash boot sector, the build process is:

```
% export TOPDIR=‘pwd‘
% export ECOS_REPOSITORY=\
    ${TOPDIR}/src/ecos-monitors/redboot-DATE-intel/packages
% mkdir ${TOPDIR}/build
% cd ${TOPDIR}/build
% ecosconfig new iq80310 redboot
% ecosconfig import \
   ${ECOS_REPOSITORY}/hal/arm/iq80310/VERSION/misc/redboot_ROM.ecm
% ecosconfig tree
% make
```

If a different configuration is desired, simply use the above build process but substitute an alternate configuration file for the ecosconfig import command, e.g.:

For a RedBoot that runs from flash address 0x40000 with the ARM booloader in the flash boot sector, use:

```
% ecosconfig import \
  ${ECOS_REPOSITORY}/hal/arm/iq80310/VERSION/misc/redboot_ROMA.ecm
```

For a RedBoot which runs from RAM with RedBoot located in the flash boot sector, use:

```
% ecosconfig import \
  ${ECOS_REPOSITORY}/hal/arm/iq80310/VERSION/misc/redboot_RAM.ecm
```

For a RedBoot which runs from RAM with ARM bootloader located in the flash boot sector, use:

```
% ecosconfig import \
  ${ECOS_REPOSITORY}/hal/arm/iq80310/VERSION/misc/redboot_RAMA.ecm
```

5.1.9 Interrupts

RedBoot uses an interrupt vector table which is located at address 0xA000A004. Entries in this table are pointers to functions with this protoype::

```
int irq_handler( unsigned vector, unsigned data )
```

On an IQ80310 board, the vector argument is one of 49 interrupts defined in `hal/arm/iq80310/current/include/hal_platform_ints.h`::

```
// *** 80200 CPU ***
#define CYGNUM_HAL_INTERRUPT_reserved0     0
#define CYGNUM_HAL_INTERRUPT_PMU_PMN0_OVFL 1 // See Ch.12 - Performance Mon.
#define CYGNUM_HAL_INTERRUPT_PMU_PMN1_OVFL 2 // PMU counter 0/1 overflow
#define CYGNUM_HAL_INTERRUPT_PMU_CCNT_OVFL 3 // PMU clock overflow
#define CYGNUM_HAL_INTERRUPT_BCU_INTERRUPT 4 // See Ch.11 - Bus Control Unit
#define CYGNUM_HAL_INTERRUPT_NIRQ          5 // external IRQ
#define CYGNUM_HAL_INTERRUPT_NFIQ          6 // external FIQ


// *** XINT6 interrupts ***
#define CYGNUM_HAL_INTERRUPT_DMA_0         7
#define CYGNUM_HAL_INTERRUPT_DMA_1         8
#define CYGNUM_HAL_INTERRUPT_DMA_2         9
#define CYGNUM_HAL_INTERRUPT_GTSC         10 // Global Time Stamp Counter
#define CYGNUM_HAL_INTERRUPT_PEC          11 // Performance Event Counter
#define CYGNUM_HAL_INTERRUPT_AAIP         12 // application accelerator unit


// *** XINT7 interrupts ***
// I2C interrupts
#define CYGNUM_HAL_INTERRUPT_I2C_TX_EMPTY 13
#define CYGNUM_HAL_INTERRUPT_I2C_RX_FULL  14
#define CYGNUM_HAL_INTERRUPT_I2C_BUS_ERR  15
#define CYGNUM_HAL_INTERRUPT_I2C_STOP     16
#define CYGNUM_HAL_INTERRUPT_I2C_LOSS     17
#define CYGNUM_HAL_INTERRUPT_I2C_ADDRESS  18


// Messaging Unit interrupts
#define CYGNUM_HAL_INTERRUPT_MESSAGE_0           19
#define CYGNUM_HAL_INTERRUPT_MESSAGE_1           20
#define CYGNUM_HAL_INTERRUPT_DOORBELL            21
#define CYGNUM_HAL_INTERRUPT_NMI_DOORBELL        22
#define CYGNUM_HAL_INTERRUPT_QUEUE_POST          23
#define CYGNUM_HAL_INTERRUPT_OUTBOUND_QUEUE_FULL 24
#define CYGNUM_HAL_INTERRUPT_INDEX_REGISTER      25
// PCI Address Translation Unit
#define CYGNUM_HAL_INTERRUPT_BIST                26


// *** External board interrupts (XINT3) ***
#define CYGNUM_HAL_INTERRUPT_TIMER        27 // external timer
```

```
#define CYGNUM_HAL_INTERRUPT_ETHERNET     28 // onboard enet
#define CYGNUM_HAL_INTERRUPT_SERIAL_A     29 // 16x50 uart A
#define CYGNUM_HAL_INTERRUPT_SERIAL_B     30 // 16x50 uart B
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTD   31 // secondary PCI INTD
// The hardware doesn't (yet?) provide masking or status for these
// even though they can trigger cpu interrupts. ISRs will need to
// poll the device to see if the device actually triggered the
// interrupt.
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTC   32 // secondary PCI INTC
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTB   33 // secondary PCI INTB
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTA   34 // secondary PCI INTA


// *** NMI Interrupts go to FIQ ***
#define CYGNUM_HAL_INTERRUPT_MCU_ERR       35
#define CYGNUM_HAL_INTERRUPT_PATU_ERR      36
#define CYGNUM_HAL_INTERRUPT_SATU_ERR      37
#define CYGNUM_HAL_INTERRUPT_PBDG_ERR      38
#define CYGNUM_HAL_INTERRUPT_SBDG_ERR      39
#define CYGNUM_HAL_INTERRUPT_DMA0_ERR      40
#define CYGNUM_HAL_INTERRUPT_DMA1_ERR      41
#define CYGNUM_HAL_INTERRUPT_DMA2_ERR      42
#define CYGNUM_HAL_INTERRUPT_MU_ERR        43
#define CYGNUM_HAL_INTERRUPT_reserved52    44
#define CYGNUM_HAL_INTERRUPT_AAU_ERR       45
#define CYGNUM_HAL_INTERRUPT_BIU_ERR       46


// *** ATU FIQ sources ***
#define CYGNUM_HAL_INTERRUPT_P_SERR        47
#define CYGNUM_HAL_INTERRUPT_S_SERR        48
```

The data passed to the ISR is pulled from a data table (`hal_interrupt_data`) which immediately follows the interrupt vector table. With 49 interrupts, the data table starts at address 0xA000A0C8.

An application may create a normal C function with the above prototype to be an ISR. Just poke its address into the table at the correct index and enable the interrupt at its source. The return value of the ISR is ignored by RedBoot.

5.1.10 Memory Maps

The first level page table is located at 0xa0004000. Two second level tables are also used. One second level table is located at 0xa0008000 and maps the first 1MB of flash. The other second level table is at 0xa0008400, and maps the first 1MB of SDRAM.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ---------------------------------
0x00000000 - 0x00000fff    flash Memory
0x00001000 - 0x00001fff    80312 Internal Registers
0x00002000 - 0x007fffff    flash Memory
```

```
0x00800000 - 0x7fffffff    PCI ATU Outbound Direct Window
0x80000000 - 0x83ffffff    Primary PCI 32-bit Memory
0x84000000 - 0x87ffffff    Primary PCI 64-bit Memory
0x88000000 - 0x8bffffff    Secondary PCI 32-bit Memory
0x8c000000 - 0x8fffffff    Secondary PCI 64-bit Memory
0x90000000 - 0x9000ffff    Primary PCI IO Space
0x90010000 - 0x9001ffff    Secondary PCI IO Space
0x90020000 - 0x9fffffff    Unused
0xa0000000 - 0xbfffffff    SDRAM
0xc0000000 - 0xefffffff    Unused
0xf0000000 - 0xffffffff    80200 Internal Registers


Virtual Address Range    C B  Description
-----------------------  - -  ---------------------------------
0x00000000 - 0x00000fff  Y Y  SDRAM
0x00001000 - 0x00001fff  N N  80312 Internal Registers
0x00002000 - 0x007fffff  Y N  flash Memory
0x00800000 - 0x7fffffff  N N  PCI ATU Outbound Direct Window
0x80000000 - 0x83ffffff  N N  Primary PCI 32-bit Memory
0x84000000 - 0x87ffffff  N N  Primary PCI 64-bit Memory
0x88000000 - 0x8bffffff  N N  Secondary PCI 32-bit Memory
0x8c000000 - 0x8fffffff  N N  Secondary PCI 64-bit Memory
0x90000000 - 0x9000ffff  N N  Primary PCI IO Space
0x90010000 - 0x9001ffff  N N  Secondary PCI IO Space
0xa0000000 - 0xa0000fff  Y N  flash
0xa0001000 - 0xbfffffff  Y Y  SDRAM
0xc0000000 - 0xcfffffff  Y Y  Cache Flush Region
0xf0000000 - 0xffffffff  N N  80200 Internal Registers
```

5.1.11 Resource Usage

The standalone flash based RedBoot image (no ARM bootloader) occupies flash addresses 0x00000000 - 0x0003ffff.

The flash based RedBoot configured to be booted by the ARM bootloader occupies flash addresses 0x00040000 - 0x0007ffff. Both of these also reserve RAM (0xa0000000 - 0xa001ffff) for RedBoot runtime uses.

Both RAM based RedBoot configurations are designed to run from RAM at addresses 0xa0020000 - 0xa003ffff. RAM addresses from 0xa0040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images before they are written to flash.

The external timer is used as a polled timer to provide timeout support for networking and XModem file transfers.

5.2 Intel SA1100 (Brutus)

5.2.1 Overview

RedBoot supports both board serial ports on the Brutus board. The default serial port settings are 38400,8,N,1. flash management is not currently supported.

Two basic RedBoot configurations are supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.2.2 Initial Installation Method

Device programmer is used to program socketecflash parts.

5.2.3 Special RedBoot Commands

None.

5.2.4 Memory Maps

The first level page table is located at physical address 0xc0004000. No second level tables are used.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ---------------------------------
0x00000000 - 0x000fffff    Boot ROM
0x08000000 - 0x083fffff    Application flash
0x10000000 - 0x100fffff    SRAM
0x18000000 - 0x180fffff    Chip Select 3
0x20000000 - 0x3fffffff    PCMCIA
0x80000000 - 0xbfffffff    SA-1100 Internal Registers
0xc0000000 - 0xc7ffffff    DRAM Bank 0
0xc8000000 - 0xcfffffff    DRAM Bank 1
0xd0000000 - 0xd7ffffff    DRAM Bank 2
0xd8000000 - 0xdfffffff    DRAM Bank 3
0xe0000000 - 0xe7ffffff    Cache Clean


Virtual Address Range    C B  Description
-----------------------  - -  ---------------------------------
0x00000000 - 0x003fffff  Y Y  DRAM Bank 0
0x00400000 - 0x007fffff  Y Y  DRAM Bank 1
0x00800000 - 0x00bfffff  Y Y  DRAM Bank 2
0x00c00000 - 0x00ffffff  Y Y  DRAM Bank 3
0x08000000 - 0x083fffff  Y Y  Application flash
```

```
0x10000000 - 0x100fffff  Y N  SRAM
0x20000000 - 0x3fffffff  N N  PCMCIA
0x40000000 - 0x400fffff  Y Y  Boot ROM
0x80000000 - 0xbfffffff  N N  SA-1100 Internal Registers
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
```

5.2.5 Resource Usage

The flash based RedBoot image occupies flash addresses 0x40000000 - 0x4000ffff. The RAM based RedBoot image occupies RAM addresses 0x10000 - 0x2ffff. RAM addresses from 0x30000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash. The SA11x0 OS timer is used as a polled timer to provide timeout support for XModem file transfers.

5.2.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “brutus”, “arm” and “sa11x0/brutus” respectively. Note that the configuration export files supplied in the `hal/arm/sa11x0/brutus/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.3 Intel StrongArm EBSA 285

5.3.1 Overview

RedBoot uses the single EBSA-285 serial port. The default serial port settings are 38400,8,N,1. If the EBSA-285 is used as a host on a PCI backplane, ethernet is supported using an Intel PRO/100+ ethernet adapter.

Management of onboard flash is also supported. Two basic RedBoot configurations are supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.3.2 Initial Installation Method

A linux application is used to program the flash over the PCI bus. Sources and build instructions for this utility are located in the RedBoot sources in:

```
.../packages/hal/arm/ebsa285/current/support/linux/safl_util
```

5.3.3 Flash management

5.3.3.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x41000000
-b 0x100000
-l 0x20000
```

5.3.3.2 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x41040000
-b 0x20000
-r 0x20000
-l 0x20000
```

5.3.4 Communication Channels

Serial, Intel PRO 10/100+ 82559 PCI ethernet card.

5.3.5 Special RedBoot Commands

None.

5.3.6 Memory Maps

Physical and virtual mapping are mapped one to one on the EBSA-285 using a first level page table located at address 0x4000. No second level tables are used.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Address Range            C B  Description
-----------------------  - -  ---------------------------------
0x00000000 - 0x007fffff  Y Y  SDRAM
0x40000000 - 0x400fffff  N N  21285 Registers
0x41000000 - 0x413fffff  Y N  flash
0x42000000 - 0x420fffff  N N  21285 CSR Space
0x50000000 - 0x50ffffff  Y Y  Cache Clean
0x78000000 - 0x78ffffff  N N  Outbound Write Flush
0x79000000 - 0x7c0fffff  N N  PCI IACK/Config/IO
0x80000000 - 0xffffffff  N Y  PCI Memory
```

5.3.7 Resource Usage

The flash based RedBoot image occupies flash addresses 0x41000000 - 0x4101ffff. It also reserves the first 128K bytes of RAM for runtime uses. The RAM based RedBoot image occupies RAM addresses 0x20000 - 0x3ffff. RAM addresses from 0x40000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

Timer3 is used as a polled timer to provide timeout support for networking and XModem file transfers.

5.3.8 Building eCos Test Cases to run with old RedBoots

If using older versions of RedBoot, the default configuration for EBSA-285 will send diagnostic output to the serial line only, not over an ethernet connection. To allow eCos programs to use RedBoot to channel diagnostic output to GDB whether connected by net or serial, enable the configuration option

```
CYGSEM_HAL_VIRTUAL_VECTOR_DIAG
"Do diagnostic IO via virtual vector table"
```

located here in the common HAL configuration tree:

```
"eCos HAL"
     "ROM monitor support"
          "Enable use of virtual vector calling interface"
              "Do diagnostic IO via virtual vector table"
```

Other than that, no special configuration is required to use RedBoot.

If you have been using built-in stubs to acquire support for thread-aware debugging, you can still do that, but you must only use the serial device for GDB connection and you must not enable the

option mentioned above. However, it is no longer necessary to do that to get thread-awareness; RedBoot is thread aware.

5.3.9 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “ebsa285”, “arm” and “ebsa285” respectively. Note that the configuration export files supplied in the `hal/arm/ebsa285/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.4 Intel SA1100 Multimedia Board

5.4.1 Overview

RedBoot supports both board serial ports. The default serial port settings are 38400,8,N,1. flash management is also supported. Two basic RedBoot configurations are supported: n

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.4.2 Initial Installation Method

A device programmer is used to program socketed flash parts.

5.4.3 Special RedBoot Commands

None.

5.4.4 Memory Maps

The first level page table is located at physical address 0xc0004000. No second level tables are used.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x000fffff    Boot flash
0x08000000 - 0x083fffff    Application flash
0x10000000 - 0x107fffff    SA-1101 Board Registers
0x18000000 - 0x180fffff    Ct8020 DSP
0x18400000 - 0x184fffff    XBusReg
0x18800000 - 0x188fffff    SysRegA
0x18c00000 - 0x18cfffff    SysRegB
0x19000000 - 0x193fffff    Spare CPLD A
0x19400000 - 0x197fffff    Spare CPLD B
0x20000000 - 0x3fffffff    PCMCIA
0x80000000 - 0xbfffffff    SA1100 Internal Registers
0xc0000000 - 0xc07fffff    DRAM Bank 0
0xe0000000 - 0xe7ffffff    Cache Clean
Virtual Address Range    C B  Description


-----------------------  - -  ----------------------------------
0x00000000 - 0x007fffff  Y Y  DRAM Bank 0
0x08000000 - 0x083fffff  Y Y  Application flash
0x10000000 - 0x100fffff  N N  SA-1101 Registers
0x18000000 - 0x180fffff  N N  Ct8020 DSP
0x18400000 - 0x184fffff  N N  XBusReg
```

```
0x18800000 - 0x188fffff  N N  SysRegA
0x18c00000 - 0x18cfffff  N N  SysRegB
0x19000000 - 0x193fffff  N N  Spare CPLD A
0x19400000 - 0x197fffff  N N  Spare CPLD B
0x20000000 - 0x3fffffff  N N  PCMCIA
0x50000000 - 0x500fffff  Y Y  Boot flash
0x80000000 - 0xbfffffff  N N  SA1100 Internal Registers
0xc0000000 - 0xc07fffff  N Y  DRAM Bank 0
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
```

5.4.5 Resource Usage

The flash based RedBoot image occupies virtual addresses 0x50000000 - 0x5000ffff. The RAM based RedBoot image occupies virtual addresses 0x10000 - 0x2ffff. RAM addresses from 0x30000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

The SA11x0 OS timer is used as a polled timer to provide timeout support for XModem file transfers.

5.4.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “sa1100mm”, “arm” and “sa11x0/sa1100mm” respectively. Note that the configuration export files supplied in the `hal/arm/sa11x0/sa1100mm/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.5 Intel SA1110 (Assabet)

5.5.1 Overview

RedBoot supports the board serial port and the compact flash ethernet port. The default serial port settings are 38400,8,N,1. RedBoot also supports flash management on the Assabet. Two basic RedBoot configurationsare supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.5.2 Initial Installation Method

A Windows or Linux utility is used to program flash over parallel port driven JTAG interface. See board documentation for details on in situ flash programming.

The flash parts are also socketed and may be programmed in a suitable device programmer.

5.5.3 Flash management

5.5.3.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x50000000
-b 0x60000
-l 0x10000
```

5.5.3.2 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x50010000
-b 0x20000
-r 0x20000
-l 0x10000
```

5.5.4 Special RedBoot Commands

None.

5.5.5 Memory Maps

The first level page table is located at physical address 0xc0004000. No second level tables are used.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ---------------------------------
0x00000000 - 0x07ffffff    flash
0x08000000 - 0x0fffffff    SA-1111 Board flash
0x10000000 - 0x17ffffff    Board Registers
0x18000000 - 0x1fffffff    Ethernet
0x20000000 - 0x2fffffff    SA-1111 Board PCMCIA
0x30000000 - 0x3fffffff    Compact Flash
0x40000000 - 0x47ffffff    SA-1111 Board
0x48000000 - 0x4bffffff    GFX
0x80000000 - 0xbfffffff    SA-1110 Internal Registers
0xc0000000 - 0xc7ffffff    DRAM Bank 0
0xc8000000 - 0xcfffffff    DRAM Bank 1
0xd0000000 - 0xd7ffffff    DRAM Bank 2
0xd8000000 - 0xdfffffff    DRAM Bank 3
0xe0000000 - 0xe7ffffff    Cache Clean


Virtual Address Range    C B  Description
-----------------------  - -  ---------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM Bank 0
0x08000000 - 0x0fffffff  Y Y  SA-1111 Board flash
0x10000000 - 0x17ffffff  N N  Board Registers
0x18000000 - 0x1fffffff  N N  Ethernet
0x20000000 - 0x2fffffff  N N  SA-1111 Board PCMCIA
0x30000000 - 0x3fffffff  N N  Compact Flash
0x40000000 - 0x47ffffff  N N  SA-1111 Board
0x48000000 - 0x4bffffff  N N  GFX
0x50000000 - 0x57ffffff  Y Y  flash
0x80000000 - 0xbfffffff  N N  SA-1110 Internal Registers
0xc0000000 - 0xc1ffffff  N Y  DRAM Bank 0
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
The flash based RedBoot image occupies virtual addresses 0x50000000 - 0x5001ffff.
```

5.5.6 Resource Usage

The RAM based RedBoot image occupies RAM addresses 0x20000 - 0x5ffff. RAM addresses from 0x60000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

The SA11x0 OS timer is used as a polled timer to provide timeout support for network and XModem file transfers.

5.5.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “assabet”, “arm” and “sa11x0/assabet” respectively. Note that the configuration export files supplied in the `hal/arm/sa11x0/assabet/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.6 MIPS Atlas Board with CoreLV 4Kc and CoreLV 5Kc

5.6.1 Overview

RedBoot supports the DgbSer serial port and the built in ethernet port for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot runs from and supports flash management for the system flash region. These configurations are supported:

- RedBoot running from the system flash boot sector.
- RedBoot running from RAM with RedBoot in the system flash boot sector.

5.6.2 Initial Installation

RedBoot is installed using the code download facility built into the Atlas board. See the Atlas User manual for details, and also the Atlas download format in Section 5.6.2.2.

5.6.2.1 Quick download instructions

Here are quick start instructions for downloading the prebuilt RedBoot image.

1. Locate the prebuilt files in the bin directory: `deleteall.dl` and `redboot.dl`.
2. Make sure switch S1-1 is OFF and switch S5-1 is ON. Reset the board and verify that the LED display reads `Flash DL`.
3. Make sure your parallel port is connected to the 1284 port Of the Atlas board.
4. Send the deleteall.dl file to the parallel port to erase previous images:

   ```
   % cat deleteall.dl >/dev/lp0
   ```

   When this is complete, the LED display should read “Deleted.”
5. Send the RedBoot image to the board:

   ```
   % cat redboot.dl >/dev/lp0
   ```

   When this is complete, the LED display should show the last address programmed. This will be something like: `1fc17000`.
6. Change switch S5-1 to OFF and reset the board. The LED display should read “RedBoot”.
7. Run the RedBoot `fis init` and `fconfig` commands to initialize the flash. See Section 5.6.3.1, Section 2.4 and Section 2.5 for details.

5.6.2.2 Atlas download format

In order to download RedBoot to the Atlas board, it must be converted to the Atlas download format. There are different ways of doing this depending on which version of the developer’s kit is shipped with the board.

The *Atlas Developer’s Kit* CD contains an `srec2flash` utility. The source code for this utilty is part of the `yamon/yamon-src-01.01.tar.gz` tarball on the Dev Kit CD. The path in the expanded tarball is `yamon/bin/tools`. To use `srec2flash` to convert the S-record file:

```
% srec2flash -EL -S29 redboot.srec >redboot.dl
```

The *Atlas/Malta Developer's Kit* CD contains an `srecconv.pl` utility which requires Perl. This utilty is part of the `yamon/yamon-src-02.00.tar.gz` tarball on the Dev Kit CD. The path in the expanded tarball is `yamon/bin/tools`. To use `srecconv` to convert the S-record file:

```
% cp redboot.srec redboot.rec
% srecconv.pl -ES L -A 29 redboot
```

The resulting file is named redboot.fl.

5.6.3 Flash management

5.6.3.1 Additional config options

The ethernet MAC address is stored in flash manually using the `fconfig` command. You can use the YAMON `setenv ethaddr` command to print out the board ethernet address. Typically, it is:

```
00:0d:a0:00:xx:xx
```

where xx.xx is the hex representation of the board serial number.

5.6.3.2 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x9dc40000
-b 0x80020000
-r 0x80020000
-l 0x40000
```

5.6.3.3 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x9dc00000
-b 0x80080000
-l 0x40000
```

5.6.4 Interrupts

RedBoot uses an interrupt vector table which is located at address 0x80000400. Entries in this table are pointers to functions with this protoype:

```
int irq_handler( unsigned vector, unsigned data )
```

On an atlas board, the vector argument is one of 25 interrupts defined in `hal/mips/atlas/`*`VERSION`*`/include/plf_intr.h`:

```
#define CYGNUM_HAL_INTERRUPT_SER                0
#define CYGNUM_HAL_INTERRUPT_TIM0               1
#define CYGNUM_HAL_INTERRUPT_2                  2
#define CYGNUM_HAL_INTERRUPT_3                  3
#define CYGNUM_HAL_INTERRUPT_RTC                4
#define CYGNUM_HAL_INTERRUPT_COREHI             5
#define CYGNUM_HAL_INTERRUPT_CORELO             6
```

```
#define CYGNUM_HAL_INTERRUPT_7                   7
#define CYGNUM_HAL_INTERRUPT_PCIA                8
#define CYGNUM_HAL_INTERRUPT_PCIB                9
#define CYGNUM_HAL_INTERRUPT_PCIC               10
#define CYGNUM_HAL_INTERRUPT_PCID               11
#define CYGNUM_HAL_INTERRUPT_ENUM               12
#define CYGNUM_HAL_INTERRUPT_DEG                13
#define CYGNUM_HAL_INTERRUPT_ATXFAIL            14
#define CYGNUM_HAL_INTERRUPT_INTA               15
#define CYGNUM_HAL_INTERRUPT_INTB               16
#define CYGNUM_HAL_INTERRUPT_INTC               17
#define CYGNUM_HAL_INTERRUPT_INTD               18
#define CYGNUM_HAL_INTERRUPT_SERR               19
#define CYGNUM_HAL_INTERRUPT_HW1                20
#define CYGNUM_HAL_INTERRUPT_HW2                21
#define CYGNUM_HAL_INTERRUPT_HW3                22
#define CYGNUM_HAL_INTERRUPT_HW4                23
#define CYGNUM_HAL_INTERRUPT_HW5                24
```

The data passed to the ISR is pulled from a data table (`hal_interrupt_data`) which immediately follows the interrupt vector table. With 25 interrupts, the data table starts at address 0x80000464 on atlas.

An application may create a normal C function with the above prototype to be an ISR. Just poke its address into the table at the correct index and enable the interrupt at its source. The return value of the ISR is ignored by RedBoot.

5.6.5 Memory Maps

Memory Maps RedBoot sets up the following memory map on the Atlas board.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range Description
---------------------- -------------
0x00000000 - 0x07ffffff SDRAM
0x08000000 - 0x17ffffff PCI Memory Space
0x18000000 - 0x1bdfffff PCI I/O Space
0x1be00000 - 0x1bffffff System Controller
0x1c000000 - 0x1dffffff System flash
0x1e000000 - 0x1e3fffff Monitor flash
0x1f000000 - 0x1fbfffff FPGA
```

5.6.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x1fc00000 - 0x1fc1ffff. RedBoot also reserves RAM (0x00000000 - 0x0001ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x00020000 - 0x0003ffff. RAM physical addresses from 0x00040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.6.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “atlas_mips32_4kc”, “mips” and “atlas” respectively. Note that the configuration export files supplied in the `hal/mips/atlas/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.7 PMC-Sierra MIPS RM7000 Ocelot

5.7.1 Overview

RedBoot uses the front facing serial port. The default serial port settings are 38400,8,N,1. RedBoot also supports ethernet. Management of onboard flash is also supported. Two basic RedBoot configurations are supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.7.2 Initial Installation Method

Device programmer is used to program socketed flash parts.

5.7.3 Flash Management

5.7.3.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0xbfc00000
-b 0x80100000
-l 0x20000
```

5.7.3.2 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, but the actual numbers used with the flags in the sample commands should be:

```
-f 0xbdc20000
-b 0x80100000
-r 0x80100000
```

5.7.4 Special RedBoot Commands

None.

5.7.5 Memory Maps

RedBoot sets up the following memory map on the Ocelot board.

Note that these addresses are accessed through kseg0/1 and thus translate to the actual address range 0x80000000-0xbfffffff, depending on the need for caching/non-caching access to the bus.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range Description
---------------------- -----------
0x00000000 - 0x0fffffff SDRAM
0x10000000 - 0x10ffffff PCI I/O space
0x12000000 - 0x13ffffff PCI Memory space
0x14000000 - 0x1400ffff Galileo system controller
0x1c000000 - 0x1c0000ff PLD (board logic)
0x1fc00000 - 0x1fc7ffff flash
```

5.7.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x1fc00000 - 0x1fc1ffff. RedBoot also reserves RAM (0x00000000 - 0x0001ffff) for RedBoot runtime uses.

RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x00020000 - 0x0003ffff. RAM physical addresses from 0x00040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.7.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “ocelot”, “mips” and “rm7000/ocelot” respectively. Note that the configuration export files supplied in the `hal/mips/rm7000/ocelot/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.8 Motorola PowerPC MBX

5.8.1 Overview

RedBoot uses the SMC1/COM1 serial port. The default serial port settings are 38400,8,N,1. Ethernet is also supported using the 10-base T connector.

Management of onboard flash is also supported. Two basic RedBoot configurations are supported:

- RedBoot running from RAM with RedBoot in the flash boot sector.
- RedBoot running from the board's flash boot sector.

5.8.2 Initial Installation Method

Device programmer is used to program the XU1 socketed flash part (AM29F040B) with the ROM version of RedBoot. - Use the on-board EPPC-Bug monitor to update RedBoot.

This assumes that you have EPPC-Bug in the on-board flash. This can be determined by setting up the board according to the following instructions and powering up the board.

The EPPC-Bug prompt should appear on the SMC1 connector at 9600 baud, 8N1.

1. Set jumper 3 to 2-3 [allow XU1 flash to be programmed]
2. Set jumper 4 to 2-3 [boot EPPC-Bug]

If it is available, program the flash by following these steps:

1. Prepare EPPC-Bug for download:

   ```
   EPPC-Bug>lo 0
   ```

   At this point the monitor is ready for input. It will not return the prompt until the file has been downloaded.
2. Use the terminal emulator's ASCII download feature (or a simple clipboard copy/paste operation) to download the redboot.ppcbug file.

   Note that on Linux, Minicom's ASCII download feature seems to be broken. A workaround is to load the file into emacs (or another editor) and copy the full contents to the clipboard. Then press the mouse paste-button (usually the middle one) over the Minicom window.
3. Program the flash with the downloaded data:

   ```
   EPPC-Bug>pflash 40000 60000 fc000000
   ```
4. Switch off the power, and change jumper 4 to 1-2. Turn on the power again. The board should now boot using the newly programmed RedBoot.

To install RedBoot on a target that already has eCos GDB stubs, download the RAM version of RedBoot and run it. Initialize the flash image directory:

```
RedBoot> fi init
```

Then download the ROM version of RedBoot and program it into flash:

```
RedBoot> load redboot_ROM.srec -b 0x80100000
RedBoot> fi cr RedBoot -f 0xFE000000 -b 0x00040000 -l 0x20000
```

5.8.3 Flash management

5.8.3.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0xfe000000
-b 0x50000
-l 0x20000
```

5.8.3.2 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, but the actual numbers used with the flags in the sample commands should be:

```
-f 0xfe020000
-b 0x20000
-r 0x20000
-l 0x20000
```

5.8.4 Special RedBoot Commands

None.

5.8.5 Memory Maps

Memory Maps RedBoot sets up the following memory map on the MBX board.

```
Physical Address Range Description
---------------------- -----------
0x00000000 - 0x003fffff DRAM
0xfa100000 - 0xfa100003 LEDs
0xfe000000 - 0xfe07ffff flash (AMD29F040B)
0xff000000 - 0xff0fffff MPC registers
```

5.8.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0xfe000000 - 0xfe01ffff. RedBoot also reserves RAM (0x00000000 - 0x0001ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x00020000 - 0x0004ffff. RAM physical addresses from 0x00050000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.8.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “mbx”, “powerpc” and “mbx” respectively. Note that the configuration export files supplied in the `hal/powerpc/mbx/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.9 ARM Evaluator7T (e7t) board with ARM7TDMI

5.9.1 Overview

RedBoot supports both serial ports for communication and downloads. The default serial port settings are 38400,8,N,1.

5.9.2 Initial Installation

RedBoot is installed using the on-board boot environment. See the user manual for full details.

5.9.3 Quick download instructions

Here are quick start instructions for downloading the prebuilt Redboot image:

- Boot the board and press ENTER:

```
ARM Evaluator7T Boot Monitor PreRelease 1.00
Press ENTER within 2 seconds to stop autoboot
Boot:
```

- Erase the part of the flash where RedBoot will get programmed:

```
Boot: flasherase 01820000 10000
```

- Prepare to download the UU-encoded version of the RedBoot image:

```
Boot: download 10000
Ready to download. Use 'transmit' option on terminal
emulator to download file.
```

- Either use ASCII transmit option in the terminal emulator, or on Linux, simply cat the file to the serial port:

```
$ cat redboot.UU > /dev/ttyS0
```

  When complete, you should see:

```
Loaded file redboot.bin at address 000100000, size = 41960
Boot:
```

- Program the flash:

```
Boot: flashwrite 01820000 10000 10000
```

- And verify that the module is available:

```
Boot: rommodules
Header   Base     Limit
018057c8 01800000 018059e7 BootStrapLoader v1.0 Apr 27 2000 10:33:58
01828f24 01820000 0182a3e8 RedBoot              Apr  5 2001
```

- Reboot the board and you should see the RedBoot banner.

5.9.4 Special RedBoot Commands

None.

5.9.5 Memory Maps

RedBoot sets up the following memory map on the E7T board.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range  C B  Description
----------------------- - -  -----------
0x00000000 - 0x0007ffff Y N  SDRAM
0x03ff0000 - 0x03ffffff N N  Microcontroller registers
0x01820000 - 0x0187ffff N N  System flash (mirrored)
```

5.9.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x0182000 - 0x0182ffff.

RedBoot also reserves RAM (0x00000000 - 0x0000bfff) for RedBoot runtime uses.

RAM physical addresses from 0x0000c000 to the end of RAM are available for general use.

5.9.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “e7t”, “arm” and “e7t” respectively. Note that the configuration export files supplied in the `hal/arm/e7t/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.10 ARM ARM7 PID, Dev7 and Dev9

5.10.1 Overview

RedBoot uses either of the serial ports. The default serial port settings are 38400,8,N,1. Management of onboard flash is also supported. Two basic RedBoot configurations are supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.10.2 Initial Installation Method

Device programmer is used to program socketed flash parts with ROM version of RedBoot.

Alternatively, to install RedBoot on a target that already has eCos GDB stubs, download the RAM version of RedBoot and run it. Initialize the flash image directory: `RedBoot> fi init` Then download the ROM version of RedBoot and program it into flash:

```
RedBoot> load -b 0x00040000 -m ymodem
RedBoot> fi cr RedBoot -f 0x04000000 -b 0x00040000 -l 0x20000
```

5.10.3 Special RedBoot Commands

None.

5.10.4 Memory Maps

RedBoot sets up the following memory map on the PID board.

```
Physical Address Range Description
---------------------- -----------
0x00000000 - 0x0007ffff DRAM
0x04000000 - 0x04080000 flash
0x08000000 - 0x09ffffff ASB Expansion
0x0a000000 - 0x0bffffff APB Reference Peripheral
0x0c000000 - 0x0fffffff NISA Serial, Parallel and PC Card ports
```

5.10.5 Resource Usage

The flash based RedBoot image occupies flash addresses 0x04000000 - 0x0401ffff.

RedBoot also reserves RAM (0x00000000 - 0x00007fff) for RedBoot runtime uses.

RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x00008000 - 0x0003ffff. RAM physical addresses from 0x00040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.10.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “pid”, “arm” and “pid” respectively. Note that the configuration export files supplied in the `hal/arm/pid/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.11 Compaq iPAQ PocketPC

5.11.1 Overview

RedBoot supports the serial port via cradle or cable, and Compact Flash ethernet cards if fitted for communication and downloads. The LCD touchscreen may also be used for the console, although by default RedBoot will switch exclusively to one channel once input arrives.

The default serial port settings are 38400,8,N,1. RedBoot runs from and supports flash management for the system flash region.

5.11.2 Initial Installation

Prebuilt images for the OSloader, redboot_Compaq.bin, redboot_ROM.bin and redboot_WinCE.bin images mentioned in the instructions below are provided.

5.11.2.1 Installing RedBoot on the iPAQ using Windows/CE

The first step in installing RedBoot on the iPAQ actually involves Windows/CE, since that is the native environment on the unit. Using WinCE, you need to install an application which will run a RAM based version of RedBoot. Once this is installed and running, RedBoot can be used to update the flash with a native/ROM version of RedBoot.

- Using ActiveSync, copy the file OSloader to your iPAQ.
- Using ActiveSync, copy the file redboot_WinCE.bin to the iPAQ as bootldr in it's root directory. Note: this is not the top level folder displayed by Windows (Mobile Device), but rather the 'My Pocket PC' folder within it.
- Execute OSloader. If you didn't create a shortcut, then you will have to poke around for it using the WinCE file explorer.
- Choose the **Tools->BootLdr->Run after loading from file** menu item.

At this point, the RAM based version of RedBoot should be running. You should be able to return to this point by just executing the last two steps of the previous process if necessary.

5.11.2.2 Installing RedBoot on the iPAQ - using the Compaq boot loader

If you already have the Compaq boot loader installed on the iPAQ, then you can install RedBoot this way.

- Install RedBoot to the flash, using the command:

```
boot> load flash 0x40000
```

- Send the file redboot_Compaq.bin using xmodem

```
Bytes Sent: 168320   BPS:2771
Transfer complete
                 Flash...erasing ...
```

```
Erasing sector 00040000
writing flash..
addr: 00040000 data: EA00000E
addr: 00050000 data: E5953054
addr: 00060000 data: 2B2B2B2E
verifying ... done.
boot>
```

- Execute RedBoot

```
boot> jump 0x4000
```

Note: sometimes it has been necessary to reset the iPAQ before issuing the 'jump' command.

5.11.2.3 Setting up and testing RedBoot

When RedBoot first comes up, it will want to initialize its LCD touch screen parameters. It does this by displaying a keyboard graphic and asks you to press certain keys. Using the stylus, press and hold until the prompt is withdrawn. When you lift the stylus, RedBoot will continue with the next calibration.

Once the LCD touchscreen has been calibrated, RedBoot will start. The calibration step can be skipped by pressing the **return/abort** button on the unit (right most button with a curved arrow icon). Additionally, the unit will assume default values if the screen is not touched within about 15 seconds.

Once RedBoot has started, you should get information similar to this on the LCD screen. It will also appear on the serial port at 38400,8,N,1.

```
RedBoot(tm) bootstrap and debug environment - built 06:17:41, Mar 19 2001
Platform: Compaq iPAQ Pocket PC (StrongARM 1110)

Copyright (C) 2000, 2001, Red Hat, Inc.

RAM: 0x00000000-0x01fc0000, 0x00022400-0x01f70000 available
FLASH: 0x50000000 - 0x51000000, 64 blocks of 0x00040000 bytes each.
```

Since the LCD touchscreen is only 30 characters wide, some of this data will be off the right hand side of the display. The joypad may be used to pan left and right in order to see the full lines.

If you have a Compact Flash ethernet card, RedBoot should find it. You'll need to have BOOTP enabled for this unit (see your sysadmin for details). If it does, it will print a message like:

```
... Waiting for network card: .Ready!
Socket Communications Inc: CF+ LPE Revision E 08/04/99
IP: 192.168.1.34, Default server: 192.168.1.101
```

5.11.2.3 Installing RedBoot permanently

Once you are satisfied with the setup and that RedBoot is operating properly in your environment, you can set up your iPAQ unit to have RedBoot be the bootstrap application.

CAUTION

This step will destroy your Windows/CE environment.

Before you take this step, you might want to save your WinCE FLASH contents. This can be done using the Compaq OSloader:

- Using OSloader on the iPAQ, select the **Tools->Flash->Save to files....** menu item.
- Four (4) files, 4MB each in size will be created.
- After each file is created, copy the file to your computer, then delete the file from the iPAQ to make room in the WinCE ramdisk for the next file.

You will need to download the version of RedBoot designed as the ROM bootstrap. Then install it permanently using these commands:

```
RedBoot> lo -r -b 0x100000 /tftpboot/redboot_ROM.bin
RedBoot> fi unl -f 0x50000000 -l 0x40000
RedBoot> fis init
RedBoot> fi cr RedBoot -b 0x100000
RedBoot> fi loc -f 0x50000000 -l 0x40000
RedBoot> reboot
```

WARNING

You must type these commands exactly! Failure to do so may render your iPAQ totally useless. Once you've done this, RedBoot should come up every time you reset.

5.11.2.4 Restoring Windows/CE

To restore Windows/CE from the backup taken in Section 5.11.2.3, visit http://www.handhelds.org/projects/wincerestoration.html for directions.

5.11.3 Flash Management

5.11.3.1 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, relying on default location and size of the image. If is also possible to explicitly specify the options - the appropriate options for the iPAQ are:

```
-f 0x50040000
-b 0x00020000
-r 0x00020000
-e 0x00020040
-l 0x40000
```

When updating the image, the flash should be unlocked before programming, and relocked afterwards. This is done with the commands:

```
fis unlock -f 0x50040000 -l 0x40000
```

and

```
fis lock -f 0x50040000 -l 0x40000
```

5.11.3.2 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, relying on default location and size of the image. If is also possible to explicitly specify the options - the appropriate options for the iPAQ are:

```
-f 0x50000000
-b 0x00100000
-l 0x40000
```

When updating the image, the flash should be unlocked before programming, and relocked afterwards. This is done with the commands:

```
fis unlock -f 0x50000000 -l 0x40000
```

and

```
fis lock -f 0x50000000 -l 0x40000
```

5.11.4 Additional commands

The `exec` command which allows the loading and execution of Linux kernels, is supported for this board (see Section 2.6). The `exec` parameters used for the iPAQ are:

-b <*addr*>

Location Linux kernel was loaded to

-l <*len*>

Length of kernel

-c *"params"*

Parameters passed to kernel

-r <*addr*>

'initrd' ramdisk location

-s <*len*>

Length of initrd ramdisk

Linux kernels may be run on the iPAQ using the sources from the anonymous CVS repository at the Handhelds project (http://www.handhelds.org/) with the `elinux.patch` patch file applied. This file can be found in the `misc/` subdirectory of the iPAQ platform HAL in the RedBoot sources, normally `hal/arm/sa11x0/ipaq/`*`VERSION`*`/misc/`

On the iPAQ (and indeed all SA11x0 platforms), Linux expects to be loaded at address 0xC0008000 and the entry point is also at 0xC0008000.

5.11.5 Memory Maps

RedBoot sets up the following memory map on the iPAQ: The first level page table is located at physical address 0xc0004000. No second level tables are used.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x01ffffff    16Mb to 32Mb FLASH (nCS0)
0x30000000 - 0x3fffffff    Compact Flash
0x48000000 - 0x4bffffff    iPAQ internal registers
0x80000000 - 0xbfffffff    SA-1110 Internal Registers
0xc0000000 - 0xc1ffffff    DRAM Bank 0 - 32Mb SDRAM
0xe0000000 - 0xe7ffffff    Cache Clean


Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM - 32Mb
0x30000000 - 0x3fffffff  N N  Compact Flash
0x48000000 - 0x4bffffff  N N  iPAQ internal registers
0x50000000 - 0x51ffffff  Y Y  Up to 32Mb FLASH (nCS0)
0x80000000 - 0xbfffffff  N N  SA-1110 Internal Registers
0xc0000000 - 0xc1ffffff  N Y  DRAM Bank 0: 32Mb
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
```

5.11.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x50000000 - 0x5003fff. RedBoot also reserves RAM (0x00000000 - 0x0001ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at virtual addresses 0x00020000 - 0x0003ffff. RAM virtual addresses from 0x00040000 to the end of RAM arae available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash. An exception is RAM from 0x01F70000 - 0x01FFFFFF which is reserved for use by the LCD display.

5.11.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “ipaq”, “arm” and “sa11x0/ipaq” respectively. Note that the configuration export files supplied in the `hal/arm/sa11x0/ipaq/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.12 Cirrus Logic EP72xx (EDB7211, EDB7212)

5.12.1 Overview

RedBoot supports both serial ports on the board and the ethernet port. The default serial port settings are 38400,8,N,1. RedBoot also supports flash management on the EDB72xx for the NOR flash only. Two basic RedBoot configurationsare supported:

- RedBoot running from the board's flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.12.2 Initial Installation Method

A Windows or Linux utility is used to program flash using serial port #1 via on-chip programming firmware. See board documentation for details on in situ flash programming.

5.12.3 Flash management

5.12.3.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0xE0000000
-b 0x40000
-l 0x10000
```

5.12.3.2 Updating the secondary RedBoot image

To update the secondary RedBoot images, follow the procedures detailed in Section 4.1.2, but the actual numbers used with the flags in the sample commands should be:

```
-f 0xE0010000

-b 0x20000

-r 0x20000

-l 0x10000
```

5.12.4 Special RedBoot Commands

None.

5.12.5 Memory Maps

The MMU page tables are located at the end of DRAM.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ---------------------------------
0x00000000 - 0x01ffffff    NOR Flash
0x10000000 - 0x11ffffff    NAND Flash
0x20000000 - 0x2fffffff    Expansion 2
0x30000000 - 0x3fffffff    Expansion 3
0x40000000 - 0x4fffffff    PCMCIA 0
0x50000000 - 0x5fffffff    PCMCIA 1
0x60000000 - 0x600007ff    On-chip SRAM
0x80000000 - 0x8fffffff    I/O registers
0xc0000000 - 0xc1ffffff    DRAM

Virtual Address Range    C B  Description
-----------------------  - -  ---------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM
0x20000000 - 0x2fffffff  N N  Expansion 2
0x30000000 - 0x3fffffff  N N  Expansion 3
0x40000000 - 0x4fffffff  N N  PCMCIA 0
0x50000000 - 0x5fffffff  N N  PCMCIA 1
0x60000000 - 0x600007ff  Y Y  On-chip SRAM
0x80000000 - 0x8fffffff  N N  I/O registers
0xc0000000 - 0xc001ffff  N Y  LCD buffer (if configured)
0xe0000000 - 0xe1ffffff  Y Y  NOR Flash
0xf0000000 - 0xf1ffffff  Y Y  NAND Flash

The flash based RedBoot image occupies virtual addresses 0xe0000000 - 0xe001ffff.
```

5.12.6 Resource Usage

The RAM based RedBoot image occupies RAM addresses 0x20000 - 0x3ffff. RAM addresses from 0x40000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

The EP72xx timer #2 is used as a polled timer to provide timeout support for network and XModem file transfers.

5.12.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for ARCH_DIR and PLATFORM_DIR on this platform are “arm” and “edb7xxx” respectively. The value for TARGET is either “edb7211” or “edb7212”, depending on the desired platform. Note that the configuration export files supplied in the `hal/arm/sa11x0/brutus/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used, and the correct edb7211 vs. edb7212 variant chosen.

5.13 Bright Star Engineering commEngine and nanoEngine

5.13.1 Overview

RedBoot supports a serial port and the built in ethernet port for communication and downloads. The default serial port settings are 38400,8,N,1. RedBoot runs from and supports flash management for the system flash region. These configurations are supported:

- RedBoot running from the first free block at 0x40000
- RedBoot running from RAM

5.13.2 Initial Installation

Unlike other targets, the nanoEngine comes equipped with boot firmware which you cannot modify. See chapter 5, "nanoEngine Firmware" of the *nanoEngine Hardware Reference Manual* (we refer to "July 17, 2000 Rev 0.6") from Bright Star Engineering.

Because of this, eCos, and therefore Redboot, only supports RAM and POST startup types, rather than the more usual ROM, RAM and optionally, POST.

Briefly, the POST-startup RedBoot image lives in flash following the BSE firmware. The BSE firmware is configured, using its standard `bootcmd` parameter, to jump into the RedBoot image at startup.

5.13.3 Download Instructions

You can perform the initial load of the POST-startup RedBoot image into flash using the BSE firmware's `load` command. This will load a binary file, using TFTP, and program it into flash in one operation. Because no memory management is used in the BSE firmware, flash is mapped from address zero upwards, so the address for the RedBoot POST image is 0x40000. You must use the binary version of RedBoot for this, `redboot-post.bin`.

This assumes you have set up the other BSE firmware config parameters such that it can communicate over your network to your TFTP server.

```
>load /tftpboot/redboot-post.bin 40000
loading ... erasing blk at 00040000
erasing blk at 00050000
94168 bytes loaded cksum 00008579
done
>
> set bootcmd "go 40000"
> get
myip = 10.16.19.198
netmask = 255.255.255.0
eth = 0
gateway = 10.16.19.66
serverip = 10.16.19.66
bootcmd = go 40000
>
```

NOTE

the BSE firmware runs its serial IO at 9600 Baud; RedBoot runs instead at 38400 Baud. You must select the right baud rate in your terminal program to be able to set up the BSE firmware.

After a reset, the BSE firmware will print

```
Boot: BSE 2000 Sep 12 2000 14:00:30
autoboot: "go 40000" [hit ESC to abort]
```

and then RedBoot starts, switching to 38400 Baud.

Once you have installed a bootable RedBoot in the system in this manner, we advise re-installing using the generic method described in Chapter 4, *Updating RedBoot* in order that the Flash Image System contains an appropriate description of the flash entries.

5.13.4 Cohabiting with POST in Flash

The configuration export file named `redboot_POST.ecm` configures redboot to build for execution at address 0x50040000 (or, during bootup, 0x00040000). This is to allow power-on self-test (POST) code or immutable firmware to live in the lower addresses of the flash and to run before RedBoot gets control. The assumption is that RedBoot will be entered at its base address in physical memory, that is 0x00040000.

Alternatively, for testing, you can call it in an already running system by using **`go 0x50040040`** at another RedBoot prompt, or a branch to that address. The address is where the reset vector points, and is reported by RedBoot's **`tftp load`** command and listed by the **`fis list`** command, amongst other places.

Using the POST configuration enables a normal config option which causes linking and initialization against memory layout files called "...post..." rather than "...rom..." or "...ram..." in the `include/pkgconf` directory. Specifically:

```
665 Feb  9 17:57 include/pkgconf/mlt_arm_sa11x0_nano_post.h
839 Feb  9 17:57 include/pkgconf/mlt_arm_sa11x0_nano_post.ldi
585 Feb  9 17:57 include/pkgconf/mlt_arm_sa11x0_nano_post.mlt
```

It is these you should edit if you wish to move that execution address from 0x50040000 in the POST configuration. Startup type naturally remains ROM in this configuration.

Because the nanoEngine contains immutable boot firmware at the start of flash, RedBoot for this target is configured to reserve that area in the Flash Image System, and to create by default an entry for the POST startup RedBoot.

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
(reserved)        0x50000000  0x50000000  0x00040000  0x00000000
RedBoot[post]     0x50040000  0x00100000  0x00020000  0x50040040
RedBoot[backup]   0x50060000  0x00020000  0x00020000  0x00020040
RedBoot config    0x503E0000  0x503E0000  0x00010000  0x00000000
FIS directory     0x503F0000  0x503F0000  0x00010000  0x00000000
RedBoot>
```

The entry "(reserved)" ensures that the FIS cannot attempt to overwrite the BSE firmware, thus ensuring that the board remains bootable and recoverable even after installing a broken RedBoot image.

5.13.5 Special RedBoot Commands

The nanoEngine/commEngine has one or two Intel i82559 Ethernet controllers installed, but these have no associated serial EEPROM in which to record their Ethernet Station Address (ESA, or MAC address). The BSE firmware records an ESA for the device it uses, but this information is not available to RedBoot; we cannot share it.

To keep the ESAs for the two ethernet interfaces, two new items of RedBoot configuration data are introduced. You can list them with the RedBoot command `fconfig -l` thus:

```
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 10.16.19.91
Default server IP address: 10.16.19.66
Network hardware address [MAC] for eth0: 0x00:0xB5:0xE0:0xB5:0xE0:0x99
Network hardware address [MAC] for eth1: 0x00:0xB5:0xE0:0xB5:0xE0:0x9A
GDB connection port: 9000
Network debug at boot time: false
RedBoot>
```

You should set them before running RedBoot or eCos applications with the board connected to a network. The `fconfig` command can be used as for any configuration data item; the entire ESA is entered in one line.

5.13.6 Memory Maps

The first level page table is located at physical address 0xc0004000. No second level tables are used.

NOTE

The virtual memory maps in this section use a C and B column to indicate whether or not the region is cached (C) or buffered (B).

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x003fffff    4Mb FLASH (nCS0)
0x18000000 - 0x18ffffff    Internal PCI bus - 2 x i82559 ethernet
0x40000000 - 0x4fffffff    External IO or PCI bus
0x80000000 - 0xbfffffff    SA-1110 Internal Registers
0xc0000000 - 0xc7ffffff    DRAM Bank 0 - 32Mb SDRAM
0xc8000000 - 0xcfffffff    DRAM Bank 1 - empty
0xe0000000 - 0xe7ffffff    Cache Clean

Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x001fffff  Y Y  DRAM - 8Mb to 32Mb
0x18000000 - 0x180fffff  N N  Internal PCI bus - 2 x i82559 ethernet
0x40000000 - 0x4fffffff  N N  External IO or PCI bus
```

```
0x50000000 - 0x51ffffff  Y Y  Up to 32Mb FLASH (nCS0)
0x80000000 - 0xbfffffff  N N  SA-1110 Internal Registers
0xc0000000 - 0xc0ffffff  N Y  DRAM Bank 0: 8 or 16Mb
0xc8000000 - 0xc8ffffff  N Y  DRAM Bank 1: 8 or 16Mb or absent
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
```

The FLASH based RedBoot POST-startup image occupies virtual addresses 0x50040000 - 0x5005ffff.

The ethernet devices use a "PCI window" to communicate with the CPU. This is 1Mb of SDRAM which is shared with the ethernet devices that are on the PCI bus. It is neither cached nor buffered, to ensure that CPU and PCI accesses see correct data in the correct order. By default it is configured to be megabyte number 30, at addresses 0x01e00000-0x01efffff. This can be modified, and indeed must be, if less than 32Mb of SDRAM is installed, via the memory layout tool, or by moving the section `__pci_window` referred to by symbols `CYGMEM_SECTION_pci_window*` in the linker script.

Though the nanoEngine ships with 32Mb of SDRAM all attached to DRAM bank 0, the code can cope with any of these combinations also; "2 x " in this context means one device in each DRAM Bank.

```
1 x 8Mb = 8Mb     2 x 8Mb = 16Mb
1 x 16Mb = 16Mb   2 x 16Mb = 32Mb
```

All are programmed the same in the memory controller.

Startup code detects which is fitted and programs the memory map accordingly. If the device(s) is 8Mb, then there are gaps in the physical memory map, because a high order address bit is not connected. The gaps are the higher 2Mb out of every 4Mb. The SA11x0 OS timer is used as a polled timer to provide timeout support within RedBoot.

5.13.7 Nano Platform Port

The nano is in the set of SA11X0-based platforms. It uses the arm architectural HAL, the sa11x0 variant HAL, plus the nano platform hal. These are components

```
CYGPKG_HAL_ARM                  hal/arm/arch/
CYGPKG_HAL_ARM_SA11X0           hal/arm/sa11x0/var
CYGPKG_HAL_ARM_SA11X0_NANO      hal/arm/sa11x0/nano
```

respectively.

The target name is "nano" which includes all these, plus the ethernet driver packages, flash driver, and so on.

5.13.8 Ethernet Driver

The ethernet driver is in two parts:

A generic ether driver for Intel i8255x series devices, specifically the i82559, is `devs/eth/intel/i82559`. Its package name is `CYGPKG_DEVS_ETH_INTEL_I82559`.

The platform-specific ether driver is `devs/eth/arm/nano`. Its package is `CYGPKG_DEVS_ETH_ARM_NANO`. This tells the generic driver the address in IO memory of the

chip, for example, and other configuration details. This driver picks up the ESA from RedBoot's configuration data - unless configured to use a static ESA in the usual manner.

5.13.9 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are "nano", "arm" and "sa11x0/nano" respectively. Note that the configuration export files supplied in the `hal/arm/sa11x0/nano/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

Index

A

B

C

D

E

F

G

H

I

M

N

P

R

T

U

V