RedBoot User's Guide

RedBoot User's Guide

Documentation licensing terms

Trademarks

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.

Table of Contents

List of Examples

Chapter 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 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.

More information about RedBoot on the web

The RedBoot Net Distribution web site (http://sources.redhat.com/redboot/) contains downloadable sources and documentation for all publically released targets, including the latest features and updates.

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 the Section called *Configuring the RedBoot Environment* for details.

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 the board is reset or the channel is manually changed by the user. 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 anywhere on a command line unless escaped using the '\' character. 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.

RedBoot Editing Commands

RedBoot uses the following line editing commands.

NOTE: In this description, ^A means the character formed by typing the letter “A” while holding down the control key.

- Delete (0x7F) or Backspace (0x08) erases the character to the left of the cursor.
- ^A or HOME moves the cursor (insertion point) to the beginning of the line.
- ^K erases all characters on the line from the cursor to the end.
- ^E or END positions the cursor to the end of the line.
- ^D or DELETE erases the character under the cursor.
- ^F or RIGHT-ARROW moves the cursor one character to the right.
- ^B or LEFT-ARROW moves the cursor one character to the left.
- ^P or UP-ARROW replaces the current line by a previous line from the history buffer. A small number of lines can be kept as history. Using ^P (and ^N), the current line can be replaced by any one of the previously typed lines.
- ^N or DOWN-ARROW replaces the current line by the next line from the history buffer.

In the case of the **fconfig** command, additional editing commands are possible. As data are entered for this command, the current/previous value will be displayed and the cursor placed at the end of that data. The user may use the editing keys (above) to move around in the data to modify it as appropriate. Additionally, when certain characters are entered at the end of the current value, i.e. entered separately, certain 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. Note: ^P and ^N do not have the same meaning while editing **fconfig** data and should not be used.
- . (period) stop editing any further items. This does not change the current item.
- Return 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.

RedBoot Command History

RedBoot provides support for listing and repeating previously entered commands. A list of previously entered commands may be obtained by typing **history** at the command line:

```
RedBoot> history
  0 fis list
  1 fconfig -l
  2 load -m ymodem
  3 history
```

The following history expansions may be used to execute commands in the history list:

- !! repeats last command.
- !**n** repeats command **n**.
- !**string** repeats most recent command starting with **string**.

RedBoot Startup Mode

RedBoot can normally be configured to run in a number of startup modes (or just "modes" for short), determining its location of residence and execution:

ROM mode

In this mode, RedBoot both resides and executes from ROM memory (flash or EPROM). This mode is used when there are limited RAM resources. The flash commands cannot update the region of flash where the RedBoot image resides. In order to update the RedBoot image in flash, it is necessary to run a RAM mode instance of RedBoot.

ROMRAM mode

In this mode, RedBoot resides in ROM memory (flash or EPROM), but is copied to RAM memory before it starts executing. The RAM footprint is larger than for ROM mode, but there are two advantages to make up for this: it normally runs faster (relevant only on slower boards) and it is able to update the flash region where the image resides.

RAM mode

In this mode, RedBoot both resides and executes from RAM memory. This is used for updating a primary ROM mode image in situ and sometimes as part of the RedBoot installation on the board when there's already an existing (non-RedBoot) boot monitor available.

You can only use ROM and ROMRAM mode images for booting a board - a RAM mode image cannot run unless loaded by another ROM monitor. There is no need for this startup mode if a RedBoot ROM-RAM mode image is the primary boot monitor. When this startup mode is programmed into flash (as a convenience as it's fast to load from flash) it will generally be named as "RedBoot[RAM]" in the FIS directory.

The chosen mode has influence on flash and RAM resource usage (see the Section called *RedBoot Resource Usage*) and the procedure of an in situ update of RedBoot in flash (see Chapter 4).

The startup mode is controlled by the option CYG_HAL_STARTUP which resides in the platform HAL. Some platforms provide only some of the RAM, ROM, and ROMRAM modes, others provide additional modes.

To see mode of a currently executing RedBoot, issue the **version** command, which prints the RedBoot banner, including the startup mode (here ROM):

```
RedBoot>version

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 13:31:57, May 17 2002
```

RedBoot Resource Usage

RedBoot takes up both flash and RAM resources depending on its startup mode and number of enabled features. There are also other resources used by RedBoot, such as timers. Platform-specific resources used by RedBoot are listed in the platform specific parts of this manual.

Both flash and RAM resources used by RedBoot depend to some degree on the features enabled in the RedBoot configuration. It is possible to reduce in particular the RAM resources used by RedBoot by removing features that are not needed. Flash resources can also be reduced, but due to the granularity of the flash (the block sizes), reductions in feature size do not always result in flash resource savings.

Flash Resources

On many platforms, a ROM mode RedBoot image resides in the first flash sectors, working as the board's primary boot monitor. On these platforms, it is also normal to reserve a similar amount of flash for a secondary RAM mode image, which is used when updating the primary ROM mode image.

On other platforms, a ROMRAM mode RedBoot image is used as the primary boot monitor. On these platforms there is not normally reserved space for a RAM mode RedBoot image, since the ROMRAM mode RedBoot is capable of updating the primary boot monitor image.

Most platforms also contain a FIS directory (keeping track of available flash space) and a RedBoot config block (containing RedBoot board configuration data).

To see the amount of reserved flash memory, run the **fis list** command:

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00020000  0x00000000
RedBoot[RAM]      0x00020000  0x06020000  0x00020000  0x060213C0
RedBoot config    0x0007F000  0x0007F000  0x00001000  0x00000000
FIS directory     0x00070000  0x00070000  0x0000F000  0x00000000
```

To save flash resources, use a ROMRAM mode RedBoot, or if using a ROM mode RedBoot, avoid reserving space for the RedBoot[RAM] image (this is done by changing the RedBoot configuration) and download the RAM mode RedBoot whenever it is needed. If the RedBoot image takes up a fraction of an extra flash block, it may be possible to reduce the image size enough to free this block by removing some features.

RAM Resources

RedBoot reserves RAM space for its run-time data, and such things as CPU exception/interrupt tables. It normally does so at the bottom of the memory map. It may also reserve space at the top of the memory map for configurable RedBoot features such as the net stack and zlib decompression support.

To see the actual amount of reserved space, issue the **version** command, which prints the RedBoot banner, including the RAM usage:

```
RedBoot> version

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 13:31:57, May 17 2002

Platform: FooBar (SH 7615)
Copyright (C) 2000, 2001, 2002, Free Software Foundation, Inc.

RAM: 0x06000000-0x06080000, 0x06012498-0x06061000 available
FLASH: 0x00000000 - 0x00080000, 8 blocks of 0x00010000 bytes each.
```

To simplify operations that temporarily need data in free memory, the limits of free RAM are also available as aliases (aligned to the nearest kilo-byte limit). These are named FREEMEMLO and FREEMEMHI, and can be used in commands like any user defined alias:

```
RedBoot> load -r -b %{FREEMEMLO} file
Raw file loaded 0x06012800-0x06013e53, assumed entry at 0x06012800

RedBoot> x -b %{FREEMEMHI}
06061000: 86 F5 EB D8 3D 11 51 F2  96 F4 B2 DC 76 76 8F 77  |....=.Q.....vv.w|
06061010: E6 55 DD DB F3 75 5D 15  E0 F3 FC D9 C8 73 1D DA  |.U...u]......s..|
```

To reduce RedBoot's RAM resource usage, use a ROM mode RedBoot. The RedBoot features that use most RAM are the net stack, the flash support and the gunzip support. These, and other features, can be disabled to reduce the RAM footprint, but obviously at the cost of lost functionality.

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.

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
DNS server IP address: 192.168.1.1
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.

The `DNS server IP address` option controls where RedBoot should make DNS lookups. A setting of 0.0.0.0 will disable DNS lookups. The DNS server IP address can also be set at runtime.

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>
```

Host Network Configuration

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

- dynamic IP address allocation, using BOOTP
- TFTP service for file downloading
- DNS server for hostname lookups

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.

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
   ```

Enable TFTP on Red Hat Linux 7 (or newer)

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
   ```

4. If you are using Red Hat 8 or newer, you may need to configure the built-in firewall to allow through TFTP. Either edit `/etc/sysconfig/iptables` or run **redhat-config-securitylevel** on the command line or from the menu as System Settings->Security Settings to lower the security level. You should only do this with the permission of your systems administrator and if you are already behind a separate firewall.

NOTE: Under Red Hat 7 you must address files by absolute pathnames, for example: `/tftpboot/boot.img` not `/boot.img`, as you may have done with other implementations. On systems newer than Red Hat 7 (7.1 and beyond), filenames are once again relative to the `/tftpboot` directory.

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.

Example 1-1. Sample DHCP configuration file

```
--------------- /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
```

Enable DNS server on Red Hat Linux

First, ensure that you have the proper RPM package, `caching-nameserver` installed. Then change the configuration (in `/etc/named.conf`) so that the `forwarders` point to the primary nameservers for your machine, normally using the nameservers listed in `/etc/resolv.conf`.

Example 1-2. Sample `/etc/named.conf` for Red Hat Linux 7.x

```
--------------- /etc/named.conf -----------------------------
// generated by named-bootconf.pl

options {
        directory "/var/named";
        /*
         * If there is a firewall between you and nameservers you want
         * to talk to, you might need to uncomment the query-source
         * directive below.  Previous versions of BIND always asked
```

```
         * questions using port 53, but BIND 8.1 uses an unprivileged
         * port by default.
         */
        // query-source address * port 53;


        forward first;
        forwarders {
                212.242.40.3;
                212.242.40.51;
        };
};

//
// a caching only nameserver config
//
// Uncomment the following for Red Hat Linux 7.2 or above:
// controls {
//         inet 127.0.0.1 allow { localhost; } keys { rndckey; };
// };
// include "/etc/rndc.key";
zone "." IN {
        type hint;
        file "named.ca";
};

zone "localhost" IN {
        type master;
        file "localhost.zone";
        allow-update { none; };
};

zone "0.0.127.in-addr.arpa" IN {
        type master;
        file "named.local";
        allow-update { none; };
};
```

Make sure the server is started with the command:

```
# service named start
```

and is started on next reboot with the command

```
# chkconfig named on
```

Finally, you may wish to change `/etc/resolv.conf` to use `127.0.0.1` as the nameserver for your local machine.

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.

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 to ping a host using the RedBoot ping command - both by IP address and hostname.
- Try using the RedBoot load command to download a file from a host.

Chapter 2. RedBoot Commands and Examples

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.

Format	Description	Example
-S	A boolean switch; the behavior of the command will differ, depending on the presence of the switch. In this example, the **`-f`** switch indicates that a complete initialization of the FIS data should be performed. There may be many such switches available for any given command and any or all of them may be present, in any order.	`RedBoot> fis init -f`
`-s` *val*	A qualified value; the letter "s" introduces the value, qualifying it's meaning. In the example, **`-b 0x100000`** specifies where the memory dump should begin. There may be many such switches available for any given command and any or all of them may be present, in any order.	`RedBoot> dump -b 0x100000 -l 0x20`

Format	Description	Example
operand	A simple value; some commands require a single parameter for which an additional **-x** switch would be redundant. In the example, **JFFS2** is the name of a flash image. The image name is always required, thus is no need to qualify it with a switch. Note that any un-qualified operand must always appear at the end of the command.	RedBoot> **fis delete JFFS2**

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

```
RedBoot> help
Manage aliases kept in FLASH memory
      alias name [value]
Set/Query the system console baud rate
      baudrate [-b <rate>]
Manage machine caches
      cache [ON | OFF]
Display/switch console channel
      channel [-1|<channel number>]
Display disk partitions
      disks
Display (hex dump) a range of memory
      dump -b <location> [-l <length>] [-s]
Manage flash images
      fis {cmds}
Manage configuration kept in FLASH memory
      fconfig [-i] [-l] [-n] [-f] [-d] | [-d] nickname [value]
Execute code at a location
      go [-w <timeout>] [-c] [-n] [entry]
Help about help?
      help [<topic>]
Set/change IP addresses
      ip_address [-l <local_ip_address>[/<mask_length>]] [-h <server_address>]
Load a file
      load [-r] [-v] [-d] [-c <channel>] [-h <host>] [-m {TFTP | HTTP | {x|y}MODEM | disk}]
      [-b <base_address>] <file_name>
Network connectivity test
      ping [-v] [-n <count>] [-t <timeout>] [-i <IP_addr]
      -h <host>
Reset the system
      reset
Display RedBoot version information
      version
Display (hex dump) a range of memory
      x -b <location> [-l <length>] [-s]
```

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 '$' or '+' (unless escaped via '\') 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.

NOTE: Multiple commands may be entered on a single line, separated by the semi-colon ";" character.

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.

Common Commands

alias

Name

`alias` — Manipulate command line aliases

Synopsis

alias { *`name`*} [*`value`*]

Arguments

Name	Type	Description	Default
`name`	Name	The name for this alias.	*none*
`value`	String	Replacement value for the alias.	*none*

Description

The **alias** command is used to maintain simple command line aliases. These aliases are shorthand for longer expressions. When the pattern %{name} appears in a command line, including in a script, the corresponding value will be substituted. Aliases may be nested.

If no value is provided, then the current value of the alias is displayed.

If the system supports non-volatile configuration data via the **fconfig** command (see the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2), then the value will be saved and used when

the system is reset.

Examples

Set an alias.

```
RedBoot> alias joe "This is Joe"
Update RedBoot non-volatile configuration - continue (y/n)? n
```

Display an alias.

```
RedBoot> alias joe
'joe' = 'This is Joe'
```

Use an alias. Note: the **"="** command simply echoes the command to to console.

```
RedBoot> = %{joe}
This is Joe
```

Aliases can be nested.

```
RedBoot> alias frank "Who are you? %{joe}"
Update RedBoot non-volatile configuration - continue (y/n)? n
RedBoot> = %{frank}
Who are you? This is Joe
```

Notice how the value of %{frank} changes when %{joe} is changed since the value of %{joe} is not evaluated until %{frank} is evaluated.

```
RedBoot> alias joe "This is now Josephine"
Update RedBoot non-volatile configuration - continue (y/n)? n
RedBoot> = %{frank}
Who are you? This is now Josephine
```

baudrate

Name

`baudrate` — Set the baud rate for the system serial console

Synopsis

baudrate [-b *rate*]

Arguments

Name	Type	Description	Default
-b *rate*	Number	The baud rate to use for the serial console.	*none*

Description

The **baudrate** command sets the baud rate for the system serial console.

If no value is provided, then the current value of the console baud rate is displayed.

If the system supports non-volatile configuration data via the **fconfig** command (see the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2), then the value will be saved and used when the system is reset.

Examples

Show the current baud rate.

```
RedBoot> baudrate
Baud rate = 38400
```

Change the console baud rate. In order to make this operation safer, there will be a slight pause after the first message to give you time to change to the new baud rate. If it doesn't work, or a less than affirmative answer is given to the "continue" prompt, then the baud rate will revert to the current value. Only after the baud rate has been firmly established will *RedBoot* give you an opportunity to save the value in persistent storage.

```
RedBoot> baudrate -b 57600
Baud rate will be changed to 57600 - update your settings
Device baud rate changed at this point
Baud rate changed to 57600 - continue (y/n)? y
Update RedBoot non-volatile configuration - continue (y/n)? n
```

cache

Name

`cache` — Control hardware caches

Synopsis

cache [on | off]

Arguments

Name	Type	Description	Default
on		Turn the caches on	*none*
off		Turn the caches off	*none*

Description

The **cache** command is used to manipulate the caches on the processor.

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.

Examples

Show the current cache state.

```
RedBoot> cache
Data cache: On, Instruction cache: On
```

Disable the caches.

```
RedBoot> cache off
RedBoot> cache
Data cache: Off, Instruction cache: Off
```

Enable the caches.

```
RedBoot> cache on
RedBoot> cache
Data cache: On, Instruction cache: On
```

channel

Name

`channel` — Select the system console channel

Synopsis

channel [-1 | *`channel_number`*]

Arguments

Name	Type	Description	Default
-1		Reset the console channel	*none*
channel_number	Number	Select a channel	*none*

Description

With no arguments, the **channel** command displays the current console channel number.

When passed an argument of 0 upward, this command switches the console channel to that channel number. The mapping between channel numbers and physical channels is platform specific but will typically be something like channel 0 is the first serial port, channel 1 is the second, etc.

When passed an argument of -1, this command reverts RedBoot to responding to whatever channel receives input first, as happens when RedBoot initially starts execution.

Examples

Show the current channel.

```
RedBoot> channel
Current console channel id: 0
```

Change to an invalid channel.

```
RedBoot> channel 99
**Error: bad channel number '99'
```

Revert to the default channel setting (any console mode).

```
RedBoot> channel -1
```

cksum

Name

`cksum` — Compute POSIX checksums

Synopsis

cksum {-b *location*} {-l *length*}

Arguments

Name	Type	Description	Default
-b *location*	Memory address	Location in memory for stat of data.	*none*
-l *length*	Number	Length of data	*none*

Description

Computes the POSIX checksum on a range of memory (either RAM or FLASH). The values printed (decimal cksum, decimal length, hexadecimal cksum, hexadecimal length) can be compared with the output from the Linux program 'cksum'.

Examples

Checksum a buffer.

```
RedBoot> cksum -b 0x100000 -l 0x100
POSIX cksum = 3286483632 256 (0xc3e3c2b0 0x00000100)
```

Checksum an area of memory after loading a file. Note that the base address and length parameters are provided by the preceding load command.

```
RedBoot> load -r -b %{FREEMEMLO} redboot.bin
Raw file loaded 0x06012800-0x0602f0a8
RedBoot> cksum
Computing cksum for area 0x06012800-0x0602f0a8
POSIX cksum = 2092197813 116904 (0x7cb467b5 0x0001c8a8)
```

disks

Name

`disks` — List available disk partitions.

Synopsis

disks

Arguments

None.

Description

The **disks** command is used to list disk partitions recognized by RedBoot.

Examples

Show what disk partitions are available.

```
RedBoot> disks
hda1     Linux Swap
hda2     Linux
00100000: 00 3E 00 06 00 06 00 06  00 00 00 00 00 00 00 00  |.>..............|
00100010: 00 00 00 78 00 70 00 60  00 60 00 60 00 60 00 60  |...x.p.`.`.`.`.`|
```

dump

Name

`dump` — Display memory.

Synopsis

dump {-b *`location`*} [-l *`length`*] [-s] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`location`*	Memory address	Location in memory for start of data.	*none*
-l *`length`*	Number	Length of data	32
-s	Boolean	Format data using Motorola S-records.	
-1		Access one byte (8 bits) at a time. Only the least significant 8 bits of the pattern will be used.	-1
-2		Access two bytes (16 bits) at a time. Only the least significant 16 bits of the pattern will be used.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Display a range of memory on the system console.

The **x** is a synonym for **dump**.

Note that this command 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.

Examples

Display a buffer, one byte at a time.

```
RedBoot> mfill -b 0x100000 -l 0x20 -p 0xDEADFACE
```

```
RedBoot> x -b 0x100000
00100000: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
00100010: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
```

Display a buffer, one short (16 bit) word at a time. Note in this case that the ASCII interpretation is suppressed.

```
RedBoot> dump -b 0x100000 -2
00100000: FACE DEAD FACE DEAD  FACE DEAD FACE DEAD
00100010: FACE DEAD FACE DEAD  FACE DEAD FACE DEAD
```

Display a buffer, one word (32 bit) word at a time. Note in this case that the ASCII interpretation is suppressed.

```
RedBoot> dump -b 0x100000 -4
00100000: DEADFACE DEADFACE DEADFACE DEADFACE
00100010: DEADFACE DEADFACE DEADFACE DEADFACE
```

Display the same buffer, using Motorola S-record format.

```
RedBoot> dump -b 0x100000 -s
S31500100000CEFAADDECEFAADDECEFAADDECEFAADDE8E
S31500100010CEFAADDECEFAADDECEFAADDECEFAADDE7E
```

Display a buffer, with visible ASCII strings.

```
RedBoot> d -b 0xfe00b000 -l 0x80
0xFE00B000: 20 25 70 0A 00 00 00 00  41 74 74 65 6D 70 74 20 | %p.....Attempt |
0xFE00B010: 74 6F 20 6C 6F 61 64 20  53 2D 72 65 63 6F 72 64 |to load S-record|
0xFE00B020: 20 64 61 74 61 20 74 6F  20 61 64 64 72 65 73 73 | data to address|
0xFE00B030: 3A 20 25 70 20 5B 6E 6F  74 20 69 6E 20 52 41 4D |: %p [not in RAM|
0xFE00B040: 5D 0A 00 00 2A 2A 2A 20  57 61 72 6E 69 6E 67 21 |]...*** Warning!|
0xFE00B050: 20 43 68 65 63 6B 73 75  6D 20 66 61 69 6C 75 72 | Checksum failur|
0xFE00B060: 65 20 2D 20 41 64 64 72  3A 20 25 6C 78 2C 20 25 |e - Addr: %lx, %|
0xFE00B070: 30 32 6C 58 20 3C 3E 20  25 30 32 6C 58 0A 00 00 |02lX <> %02lX...|
0xFE00B080: 45 6E 74 72 79 20 70 6F  69 6E 74 3A 20 25 70 2C |Entry point: %p,|
```

help

Name

`help` — Display help on available commands

Synopsis

help [*topic*]

Arguments

Name	Type	Description	Default
topic	String	Which command to provide help for.	All commands

Description

The **help** command displays information about the available RedBoot commands. If a *topic* is given, then the display is restricted to information about that specific command.

If the command has sub-commands, e.g. **fis**, then the topic specific display will print additional information about the available sub-commands. 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.

Examples

Show generic help. Note that the contents of this display will depend on the various configuration options for RedBoot when it was built.

```
RedBoot> help
Manage aliases kept in FLASH memory
   alias name [value]
Manage machine caches
   cache [ON | OFF]
Display/switch console channel
   channel [-1|<channel number>]
Compute a 32bit checksum [POSIX algorithm] for a range of memory
   cksum -b <location> -l <length>
Display (hex dump) a range of memory
   dump -b <location> [-l <length>] [-s] [-1|-2|-4]
Manage FLASH images
   fis {cmds}
Manage configuration kept in FLASH memory
   fconfig [-i] [-l] [-n] [-f] [-d] | [-d] nickname [value]
Execute code at a location
   go [-w <timeout>] [entry]
```

```
Uncompress GZIP compressed data
   gunzip -s <location> -d <location>
Help about help?
   help [<topic>]
Read I/O location
   iopeek [-b <location>] [-1|2|4]
Write I/O location
   iopoke [-b <location>] [-1|2|4] -v <value>
Set/change IP addresses
   ip_address [-l <local_ip_address>[/<mask_length>]] [-h <server_address>]
Load a file
   load [-r] [-v] [-d] [-h <host>] [-m {TFTP | HTTP | {x|y}MODEM -c <channel_number>}]
        [-f <flash_address>] [-b <base_address>]  <file_name>
Compare two blocks of memory
   mcmp -s <location> -d <location> -l <length> [-1|-2|-4]
Fill a block of memory with a pattern
   mfill -b <location> -l <length> -p <pattern>
  [-1|-2|-4]
Network connectivity test
   ping [-v] [-n <count>] [-l <length>] [-t <timeout>] [-r <rate>]
        [-i <IP_addr>] -h <IP_addr>
Reset the system
   reset
Display RedBoot version information
   version
Display (hex dump) a range of memory
   x -b <location> [-l <length>] [-s] [-1|-2|-4]
```

Help about a command with sub-commands.

```
RedBoot> help fis
Manage FLASH images
   fis {cmds}
Create an image
  fis create -b <mem_base> -l <image_length> [-s <data_length>]
      [-f <flash_addr>] [-e <entry_point>] [-r <ram_addr>] [-n] <name>
Display an image from FLASH Image System [FIS]
  fis delete name
Erase FLASH contents
  fis erase -f <flash_addr> -l <length>
Display free [available] locations within FLASH Image System [FIS]
  fis free
Initialize FLASH Image System [FIS]
  fis init [-f]
Display contents of FLASH Image System [FIS]
  fis list [-c] [-d]
Load image from FLASH Image System [FIS] into RAM
  fis load [-d] [-b <memory_load_address>] [-c] name
Write raw data directly to FLASH
  fis write -f <flash_addr> -b <mem_base> -l <image_length>
```

iopeek

Name

`iopeek` — Read I/O location

Synopsis

iopeek [-b *`location`*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`location`*	I/O address	I/O Location.	*none*
-1		Access a one byte (8 bit) I/O location.	-1
-2		Access a two byte (16 bit) I/O location.	-1
-4		Access a one word (32 bit) I/O location.	-1

Description

Reads a value from the I/O address space.

Examples

Examine 8 bit value at I/O location 0x3F8.

```
RedBoot> iopeek -b 0x3f8
0x03f8 = 0x30
```

Examine 32 bit value at I/O location 0x3f8.

```
RedBoot> iopeek -b 0x3f8 -4
0x03f8 = 0x03c10065
```

iopoke

Name

`iopoke` — Write I/O location

Synopsis

iopoke [-b *`location`*] [-1 | -2 | -4] [-v *`value`*]

Arguments

Name	Type	Description	Default
-b *`location`*	I/O address	I/O Location.	*none*
-1		Access a one byte (8 bit) I/O location. Only the 8 least significant bits of value will be used	-1
-2		Access a two byte (16 bit) I/O location. Only the 16 least significant bits of value will be used	-1
-4		Access a one word (32 bit) I/O location.	-1

Description

Writes a value to the I/O address space.

Examples

Write 0x0123 to 16 bit I/O location 0x200.

```
RedBoot> iopoke -b 0x200 -v 0x123 -2
```

gunzip

Name

`gunzip` — Uncompress GZIP compressed data

Synopsis

gunzip {-s *source*} {-d *destination*}

Arguments

Name	Type	Description	Default
-s *location1*	Memory address	Location of GZIP compressed data to uncompress.	Value set by last **load** or **fis load** command.
-d *location2*	Memory address	Destination to write uncompressed data to.	*none*

Description

Uncompress GZIP compressed data.

Examples

Uncompress data at location 0x100000 to 0x200000.

```
RedBoot> gunzip -s 0x100000 -d 0x200000
Decompressed 38804 bytes
```

ip_address

Name

`ip_address` — Set IP addresses

Synopsis

ip_address [-b] [-l *`local_IP_address`* [/*`netmask_length`*]] [-h *`server_IP_address`*] [-d *`DNS_server_IP_address`*]

Arguments

Name	Type	Description	Default
-b	Boolean	Obtain an IP address using BOOTP or DHCP.	don't use BOOTP/DHCP
-l *`local_IP_address`*[*`/netmask_length`*]	Numeric IP or DNS name	The IP address RedBoot should use, optionally with the network mask length.	*none*
-h *`server_IP_address`*	Numeric IP or DNS name	The IP address of the default server. Use of this address is implied by other commands, such as **load**.	*none*
-d *`DNS_server_IP_address`*	Numeric IP or DNS name	The IP address of the DNS server.	*none*

Description

The **ip_address** command is used to show and/or change the basic IP addresses used by RedBoot. IP addresses may be given as numeric values, e.g. 192.168.1.67, or as symbolic names such as www.redhat.com if DNS support is enabled.

The `-b` option is used to cause the target to perform a bootp or dhcp negotiation to get an IP address.

The `-l` option is used to set the IP address used by the target device. The network mask length can also be specified

The `-h` option is used to set the default server address, such as is used by the **load** command.

The `-d` option is used to set the default DNS server address which is used for resolving symbolic network addresses. Note that an address of 0.0.0.0 will disable DNS lookups.

Examples

Display the current network settings.

```
RedBoot> ip_address
IP: 192.168.1.31, Default server: 192.168.1.101, DNS server IP: 0.0.0.0, DNS domain name:
```

Change the DNS server address.

```
RedBoot> ip_address -d 192.168.1.101
IP: 192.168.1.31, Default server: 192.168.1.101, DNS server IP: 192.168.1.101, DNS domain name:
```

Change the DNS domain name.

```
RedBoot> ip_address -D example.com
IP: 192.168.1.31, Default server: 192.168.1.101, DNS server IP: 192.168.1.101, DNS domain name: ex
```

Change the default server address.

```
RedBoot> ip_address -h 192.168.1.104
IP: 192.168.1.31, Default server: 192.168.1.104, DNS server IP: 192.168.1.101, DNS domain name:
```

Set the IP address to something new, with a 255.255.255.0 netmask

```
RedBoot> ip_address -l 192.168.1.32/24
IP: 192.168.1.32, Default server: 192.168.1.104, DNS server IP: 192.168.1.101, DNS domain name:
```

load

Name

`load` — Download programs or data to the RedBoot platform

Synopsis

load [-v] [-d] [-r] [-m [[xmodem | ymodem | tftp | disk | file]]] [-h *`server_IP_address`*] [-f *`location`*] [-b *`location`*] [-c *`channel`*] [*`file_name`*]

Arguments

Name	Type	Description	Default
-v	Boolean	Display a small spinner (indicator) while the download is in progress. This is just for feedback, especially during long loads. Note that the option has no effect when using a serial download method since it would interfere with the protocol.	*quiet*
-d	Boolean	Decompress data stream (gzip data)	*non-compressed data*
-r	Boolean	Raw (or binary) data. -b or -f must be used	*formatted (S-records, ELF image, etc)*
-m tftp		Transfer data via the network using TFTP protocol.	TFTP
-m http		Transfer data via the network using HTTP protocol.	TFTP
-m xmodem		Transfer data using *X-modem* protocol.	TFTP
-m ymodem		Transfer data using *Y-modem* protocol.	TFTP
-m disk		Transfer data from a local disk.	TFTP
-m file		Transfer data from a local filesystem such as JFFS2 or FAT.	TFTP

Name	Type	Description	Default
-h *server_IP_address*	Numeric IP or DNS name	The IP address of the TFTP or HTTP server.	Value set by **ip_address**
-b *location*	Number	Address in memory to load the data. Formatted data streams will have an implied load address which this option may override.	*Depends on data format*
-f *location*	Number	Address in flash to load the data. Formatted data streams will have an implied load address which this option may override.	*Depends on data format*
-c *channel*	Number	Specify which I/O channel to use for download. This option is only supported when using either xmodem or ymodem protocol.	*Depends on data format*
file_name	String	The name of the file on the TFTP or HTTP server or the local disk. Details of how this is specified for TFTP are host-specific. For local disk files, the name must be in *disk*: *filename* format. The disk portion must match one of the disk names listed by the **disks** command.	*None*

Description

The **load** command is used to download data into the target system. Data can be loaded via a network connection, using either the TFTP or HTTP protocols, or the console serial connection using the X/Y modem protocol. Files may also be loaded directly from local filesystems on disk. Files to be downloaded may either be executable images in ELF executable program format, Motorola S-record (SREC) format or raw data.

> **Note:** When downloading an ELF image, RedBoot will forcibly terminate the transfer once all the relevant (loadable) ELF sections have been received. This behaviour reduces download time when using the X/Y modem protocol over a slow serial connection. However, the terminal emulator may report that the transfer is incomplete and has been cancelled. Such messages are normal and may be ignored.

Examples

Download a Motorola S-record (or ELF) image, using TFTP, specifying the base memory address.

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

Download a Motorola S-record (or ELF) image, using HTTP, specifying the host [server] address.

```
RedBoot> load /redboot.ROM -m HTTP -h 192.168.1.104
Address offset = 0x0c400000
Entry point: 0x80000000, address range: 0x80000000-0x8000fe80
```

Load an ELF file from /dev/hda1 which should be an EXT2 partition:

```
RedBoot> load -mode disk hda1:hello.elf
Entry point: 0x00020000, address range: 0x00020000-0x0002fd70
```

Load an ELF file from /jffs2/applications which should be a directory in a JFFS2 filesystem:

```
RedBoot> load -mode file /jffs2/applications/hello.elf
Entry point: 0x00020000, address range: 0x00020000-0x0002fd70
```

mcmp

Name

`mcmp` — Compare two segments of memory

Synopsis

mcmp {-s *location1*} {-d *location1*} {-l *length*} [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-s *location1*	Memory address	Location for start of data.	*none*
-d *location2*	Memory address	Location for start of data.	*none*
-l *length*	Number	Length of data	*none*
-1		Access one byte (8 bits) at a time. Only the least significant 8 bits of the pattern will be used.	-4
-2		Access two bytes (16 bits) at a time. Only the least significant 16 bits of the pattern will be used.	-4
-4		Access one word (32 bits) at a time.	-4

Description

Compares the contents of two ranges of memory (RAM, ROM, FLASH, etc).

Examples

Compare two buffers which match (result is *quiet*).

```
RedBoot> mfill -b 0x100000 -l 0x20 -p 0xDEADFACE
RedBoot> mfill -b 0x200000 -l 0x20 -p 0xDEADFACE
RedBoot> mcmp -s 0x100000 -d 0x200000 -l 0x20
```

Compare two buffers which don't match. Only the first non-matching element is displayed.

```
RedBoot> mcmp -s 0x100000 -d 0x200000 -l 0x30 -2
```

```
Buffers don't match - 0x00100020=0x6000, 0x00200020=0x0000
```

mcopy

Name

`mcopy` — Copy memory

Synopsis

mcopy {-s *source*} {-d *destination*} {-l *length*} [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-s *location1*	Memory address	Location of data to copy.	*none*
-d *location2*	Memory address	Destination for copied data.	*none*
-l *length*	Number	Length of data	*none*
-1		Copy one byte (8 bits) at a time.	-4
-2		Copy two bytes (16 bits) at a time.	-4
-4		Copy one word (32 bits) at a time.	-4

Description

Copies memory (RAM, ROM, FLASH, etc) from one area to another.

Examples

Copy 16 bits at a time.

```
RedBoot> mfill -b 0x100000 -l 0x20 -2 -p 0xDEAD
RedBoot> mfill -b 0x200000 -l 0x20 -2 -p 0x0
RedBoot> dump  -b 0x200000 -l 0x20 -2
00200000: 0000 0000 0000 0000  0000 0000 0000 0000
00200010: 0000 0000 0000 0000  0000 0000 0000 0000
RedBoot> mcopy -s 0x100000 -d 0x200000 -2 -l 0x20
RedBoot> dump  -b 0x200000 -l 0x20 -2
00200000: DEAD DEAD DEAD DEAD  DEAD DEAD DEAD DEAD
00200010: DEAD DEAD DEAD DEAD  DEAD DEAD DEAD DEAD
```

mfill

Name

`mfill` — Fill RAM with a specified pattern

Synopsis

mfill {-b *`location`*} {-l *`length`*} {-p *`value`*} [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`location`*	Memory address	Location in memory for start of data.	*none*
-l *`length`*	Number	Length of data	*none*
-p *`pattern`*	Number	Data value to fill with	0
-1		Access one byte (8 bits) at a time. Only the least significant 8 bits of the pattern will be used.	-4
-2		Access two bytes (16 bits) at a time. Only the least significant 16 bits of the pattern will be used.	-4
-4		Access one word (32 bits) at a time.	-4

Description

Fills a range of memory with the given pattern.

Examples

Fill a buffer with zeros.

```
RedBoot> x -b 0x100000 -l 0x20
00100000: 00 3E 00 06 00 06 00 06  00 00 00 00 00 00 00 00  |.>..............|
00100010: 00 00 00 78 00 70 00 60  00 60 00 60 00 60 00 60  |...x.p.`.`.`.`.`|
RedBoot> mfill -b 0x100000 -l 0x20
RedBoot> x -b 0x100000 -l 0x20
00100000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00100010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Fill a buffer with a pattern.

```
RedBoot> mfill -b 0x100000 -l 0x20 -p 0xDEADFACE
RedBoot> x -b 0x100000 -l 0x20
00100000: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
00100010: CE FA AD DE CE FA AD DE  CE FA AD DE CE FA AD DE  |................|
```

ping

Name

`ping` — Verify network connectivity

Synopsis

ping [-v] [-i *local_IP_address*] [-l *length*] [-n *count*] [-t *timeout*] [-r *rate*] {-h *server_IP_address*}

Arguments

Name	Type	Description	Default
-v	Boolean	Be verbose, displaying information about each packet sent.	*quiet*
-n *local_IP_address*	Number	Controls the number of packets to be sent.	10
-i *local_IP_address*	Numeric IP or DNS name	The IP address RedBoot should use.	Value set by **ip_address**
-h *server_IP_address*	Numeric IP or DNS name	The IP address of the host to contact.	*none*
-l *length*	Number	The length of the ICMP data payload.	64
-r *length*	Number	How fast to deliver packets, i.e. time between successive sends. A value of 0 sends packets as quickly as possible.	1000ms (1 second)
-t *length*	Number	How long to wait for the round-trip to complete, specified in milliseconds.	1000ms (1 second)

Description

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.

Examples

Test connectivity to host 192.168.1.101.

```
RedBoot> ping -h 192.168.1.101
Network PING - from 192.168.1.31 to 192.168.1.101
PING - received 10 of 10 expected
```

Test connectivity to host 192.168.1.101, with verbose reporting.

```
RedBoot> ping -h 192.168.1.101 -v -n 4
Network PING - from 192.168.1.31 to 192.168.1.101
 seq: 1, time: 1 (ticks)
 seq: 2, time: 1 (ticks)
 seq: 3, time: 1 (ticks)
 seq: 4, time: 1 (ticks)
PING - received 10 of 10 expected
```

```
Test connectivity to a non-existent host (192.168.1.109).
RedBoot> ping -h 192.168.1.109 -v -n 4
PING: Cannot reach server '192.168.1.109' (192.168.1.109)
```

reset

Name

`reset` — Reset the device

Synopsis

reset

Arguments

None

Description

The **reset** command causes the target platform to be reset. Where possible (hardware support permitting), this will be equivalent to a power-on reset condition.

Examples

Reset the platform.

```
RedBoot> reset
... Resetting.+... Waiting for network card: .
Socket Communications, Inc: Low Power Ethernet CF Revision C 5V/3.3V 08/27/98
Ethernet eth0: MAC address 00:c0:1b:00:ba:28
IP: 192.168.1.29, Default server: 192.168.1.101

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 10:41:41, May 14 2002

Platform: Compaq iPAQ Pocket PC (StrongARM 1110)
Copyright (C) 2000, 2001, 2002, Free Software Foundation, Inc.

RAM: 0x00000000-0x01fc0000, 0x00014748-0x01f71000 available
FLASH: 0x50000000 - 0x51000000, 64 blocks of 0x00040000 bytes each.
RedBoot>
```

version

Name

`version` — Display RedBoot version information

Synopsis

version

Arguments

None

Description

The **version** command simply displays version information about RedBoot.

Examples

Display RedBoot's version.

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

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 (fis is short for Flash Image System) is used to manipulate and maintain flash images.

fis init

Name

`fis init` — Initialize Flash Image System (FIS)

Synopsis

fis init [*-f*]

Arguments

Name	Type	Description	Default
-f		All blocks of flash memory (except for the boot blocks) will be erased as part of the initialization procedure.	

Description

This command is used to initialize the Flash Image System (FIS). It should normally only be executed once, when RedBoot is first installed on the hardware. If the reserved images or their sizes in the FIS change, due to a different configuration of RedBoot being used, it may be necessary to issue the command again though.

Note: Subsequent executions will cause loss of previously stored information in the FIS.

Examples

Initialize the FIS directory.

```
RedBoot> fis init
About to initialize [format] flash image system - continue (y/n)? y
*** Initialize FLASH Image System
```

```
    Warning: device contents not erased, some blocks may not be usable
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

Initialize the FIS directory and all of flash memory, except for first blocks of the flash where the boot monitor resides.

```
RedBoot> fis init -f
About to initialize [format] flash image system - continue (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x00020000-0x00070000: .....
... Erase from 0x00080000-0x00080000:
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

fis list

Name

`fis list` — List Flash Image System directory

Synopsis

fis list [*-c*] [*-d*]

Arguments

Name	Type	Description	Default
-c		Show image checksum instead of memory address (column `Mem addr` is replaced by `Checksum`).	
-d		Show image data length instead of amount of flash occupied by image (column `Length` is replaced by `Datalen`).	

Description

This command lists the images currently available in the FIS. Certain images used by RedBoot have fixed names and have reserved slots in the FIS (these can be seen after using the **fis init** command). Other images can be manipulated by the user.

> **Note:** The images are listed in the order they appear in the FIS directory, not by name or creation time.

Examples

List the FIS directory.

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00020000  0x00000000
RedBoot config    0x0007F000  0x0007F000  0x00001000  0x00000000
FIS directory     0x00070000  0x00070000  0x0000F000  0x00000000
```

List the FIS directory, with image checksums substituted for memory addresses.

```
RedBoot> fis list -c
Name              FLASH addr  Checksum    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00020000  0x00000000
RedBoot config    0x0007F000  0x00000000  0x00001000  0x00000000
FIS directory     0x00070000  0x00000000  0x0000F000  0x00000000
```

List the FIS directory with image data lengths substituted for flash block reservation lengths.

```
RedBoot> fis list -d
Name              FLASH addr  Mem addr    Datalen     Entry point
RedBoot           0x00000000  0x00000000  0x00000000  0x00000000
RedBoot config    0x0007F000  0x0007F000  0x00000000  0x00000000
FIS directory     0x00070000  0x00070000  0x00000000  0x00000000
```

fis free

Name

`fis free` — Free flash image

Synopsis

fis free

Description

This command shows which areas of the flash memory are currently not in use. When a block contains non-erased contents it is considered in use. 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 FIS 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.

Examples

Show free flash areas.

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

fis create

Name

`fis create` — Create flash image

Synopsis

fis create {-b *data address*} {-l *length*} [-f *flash address*] [-e *entry*] [-r *relocation address*] [-s *data length*] [-n] [*name*]

Arguments

Name	Type	Description	Default
-b	Number	Address of data to be written to the flash.	Address of last loaded file. If not set in a load operation, it must be specified.
-l	Number	Length of flash area to occupy. If specified, and the named image already exists, the length must match the value in the FIS directory.	Length of area reserved in FIS directory if the image already exists, or the length of the last loaded file. If neither are set, it must be specified.
-f	Number	Address of flash area to occopy.	The address of an area reserved in the FIS directory for extant images. Otherwise the first free block which is large enough will be used.
-e	Number	Entry address for an executable image, used by the **fis load** command.	The entry address of last loaded file.
-r	Number	Address where the image should be relocated to by the **fis load** command. This is only relevant for images that will be loaded with the **fis load** command.	The load address of the last loaded file.
-s	Number	Actual length of data written to image. This is used to control the range over which the checksum is made.	It defaults to the length of the last loaded file.

Name	Type	Description	Default
-n		When set, no image data will be written to the flash. Only the FIS directory will be updated.	
name	String	Name of flash image.	

Description

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 file into RAM and then the **fis create** command to write it to a flash image.

Examples

Trying to create an extant image, will require the action to be verified.

```
RedBoot> fis create RedBoot -f 0xa0000000 -b 0x8c400000 -l 0x20000
An image named 'RedBoot' exists - continue (y/n)? n
```

Create a new test image, let the command find a suitable place.

```
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: .
```

Update the RedBoot[RAM] image.

```
RedBoot> load redboot_RAM.img
Entry point: 0x060213c0, address range: 0x06020000-0x06036cc0
RedBoot> fis create RedBoot[RAM]
No memory address set.
An image named 'RedBoot[RAM]' exists - continue (y/n)? y
* CAUTION * about to program 'RedBoot[RAM]'
            at 0x00020000..0x00036cbf from 0x06020000 - continue (y/n)? y
... Erase from 0x00020000-0x00040000: ..
... Program from 0x06020000-0x06036cc0 at 0x00020000: ..
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

fis load

Name

`fis load` — Load flash image

Synopsis

fis load [-b *load address*] [-c] [-d] [*name*]

Arguments

Name	Type	Description	Default
-b	Number	Address the image should be loaded to. Executable images normally load at the location to which the file was linked. This option allows the image to be loaded to a specific memory location, possibly overriding any assumed location.	If not specified, the address associated with the image in the FIS directory will be used.
-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.	
name	String	The name of the file, as shown in the FIS directory.	

Description

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

Once the image has been loaded, it may be executed using the **go** command.

Examples

Load and run RedBoot[RAM] image.

```
RedBoot> fis load RedBoot[RAM]
```

```
RedBoot> go
```

fis delete

Name

`fis delete` — Delete flash image

Synopsis

fis delete {*name*}

Arguments

Name	Type	Description	Default
name	Number	Name of image that should be deleted.	

Description

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.

> **Note:** Certain images are reserved by RedBoot and cannot be deleted. RedBoot will issue a warning if this is attempted.

Examples

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

fis lock

Name

`fis lock` — Lock flash area

Synopsis

fis lock {-f *flash_address*} {-l *length*}

Arguments

Name	Type	Description	Default
flash_address	Number	Address of area to be locked.	
length	Number	Length of area to be locked.	

Description

This command is used to write-protect (lock) a portion of flash memory, to prevent accidental overwriting of images. In order to make any modifications to the flash, a matching **fis 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.

Examples

Lock an area of the flash

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

fis unlock

Name

`fis unlock` — Unlock flash area

Synopsis

fis unlock {-f *flash_address*} {-l *length*}

Arguments

Name	Type	Description	Default
flash_address	Number	Address of area to be unlocked.	
length	Number	Length of area to be unlocked.	

Description

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.

> **Note:** Some flash devices power up in locked state and always need to be manually unlocked before they can be written to.

Examples

Unlock an area of the flash

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

fis erase

Name

`fis erase` — Erase flash area

Synopsis

fis erase {-f *flash_address*} {-l *length*}

Arguments

Name	Type	Description	Default
flash_address	Number	Address of area to be erased.	
length	Number	Length of area to be erased.	

Description

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 an existing image.

Examples

Erase an area of the flash

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

fis write

Name

`fis write` — Write flash area

Synopsis

fis write {-b *mem_address*} {-l *length*} {-f *flash_address*}

Arguments

Name	Type	Description	Default
mem_address	Number	Address of data to be written to flash.	
length	Number	Length of data to be writtem.	
flash_address	Number	Address of flash to write to.	

Description

This command is used to write data from memory to flash. There is no cross-checking to ensure that the area being written to does not correspond to an existing image.

Examples

Write an area of data to the flash

```
RedBoot> fis write -b 0x0606f000 -l 0x1000 -f 0x00020000
* CAUTION * about to program FLASH
            at 0x00020000..0x0002ffff from 0x0606f000 - continue (y/n)? y
... Erase from 0x00020000-0x00030000: .
... Program from 0x0606f000-0x0607f000 at 0x00020000: .
```

Filesystem Interface

If the platform has access to secondary storage, then RedBoot may be able to access a filesystem stored on this device. RedBoot can access FAT filesystems stored on IDE disks or CompactFlash devices and can use JFFS2 filesystems stored in FLASH memory. The **fs** command is used to manipulate files on filesystems. Applications may be loaded into memory using the *file* mode of the **load** command.

fs info

Name

`fs info` — Print filesystem information

Synopsis

fs info

Arguments

The command takes no arguments.

Description

This command prints information about the filesystems that are available. Three lists are produced. The first is a list of the filsystem implementations available in RedBoot; names from this list may be used in the `-t` option to the **fs mount** command. The second list describes the block devices that are available for mounting a filesystem; names from this list may be used in the `-d` option to the **fs mount** command. The last list describes the filesystems that are already mounted.

Examples

```
RedBoot> fs info
Filesystems available:
ramfs
jffs2

Devices available:
/dev/flash1

Mounted filesystems:
         Device Filesystem Mounted on
    <undefined>      ramfs /
    /dev/flash1      jffs2 /flash
RedBoot>
```

fs mount

Name

`fs mount` — Mount a filesystem

Synopsis

fs mount [-d *device*] {-t *fstype*} {mountpoint}

Arguments

Name	Type	Description	Default
device	Number	Device containing filsystem to mount.	undefined
fstype	Number	Filesystem type.	
mountpoint	String	Pathname for filesystem root.	/

Description

This command is used make a filesystem available for access with the filesystem access commands. Three things need to be defined to do this. First, the name of the device on which the filesystem is stored needs to be given to the *-d* option. Secondly, the type of filesystem it is needs to be given to the *-t* option. Finally, the pathname by which the new filesystem will be accessed needs to be supplied. Following a successful mount, the root of the filesystem will be accessible at the mountpoint.

Examples

Mount a JFF2 partititon:

```
RedBoot> fs info
Filesystems available:
ramfs
jffs2

Devices available:
/dev/flash1

Mounted filesystems:
         Device Filesystem Mounted on
    <undefined>      ramfs /
RedBoot> fs mount -d /dev/flash1 -t jffs2 /flash
RedBoot> fs info
Filesystems available:
ramfs
jffs2
```

```
Devices available:
/dev/flash1

Mounted filesystems:
         Device Filesystem Mounted on
    <undefined>      ramfs /
    /dev/flash1      jffs2 /flash
RedBoot>
```

fs umount

Name

`fs umount` — Unmount filesystem

Synopsis

fs umount {*mountpoint*}

Arguments

Name	Type	Description	Default
mountpoint	String	Mountpoint of filesystem to unmount.	

Description

This command removes a filesystem from being accessible using the filesystem commands. The single argument needs to be the mountpoint that was used when mounting the filesystem. This command will fail if the current directory is currently within the filesystem to be unmounted.

Examples

Unmount a JFF2 partititon:

```
RedBoot> fs info
Filesystems available:
ramfs
jffs2

Devices available:
/dev/flash1

Mounted filesystems:
         Device Filesystem Mounted on
    <undefined>      ramfs /
    /dev/flash1      jffs2 /flash
RedBoot> fs umount /flash
RedBoot> fs info
Filesystems available:
ramfs
jffs2

Devices available:
/dev/flash1

Mounted filesystems:
         Device Filesystem Mounted on
```

```
            <undefined>       ramfs /
          RedBoot>
```

fs cd

Name

`fs cd` — Change filesystem directory

Synopsis

fs cd [*directory*]

Arguments

Name	Type	Description	Default
directory	String	Pathname to directory to change to.	Root directory

Description

This command changes the current filesystem directory. Subsequent filesystem commands will be executed in the new directory. If no argument is given, then the current directory is set back to the root of the filesystem name space.

Examples

Change current directory:

```
RedBoot> fs list
212416 d---------  3 size    128 .
212416 d---------  3 size    128 ..
211392 d---------  2 size     96 tests
210368 ----------  1 size   4096 image
RedBoot> fs cd tests
RedBoot> fs list
211392 d---------  2 size     96 .
212416 d---------  3 size    128 ..
205760 ----------  1 size  16384 test1
RedBoot>
```

fs mkdir

Name

`fs mkdir` — Create filesystem directory

Synopsis

fs mkdir {*directory*}

Arguments

Name	Type	Description	Default
directory	String	Pathname to directory to delete.	

Description

This command creates (makes) a directory in the filesystem.

Examples

Create directory:

```
RedBoot> fs list
212416 d---------  2 size    128 .
212416 d---------  2 size    128 ..
210368 ----------  1 size   4096 image
RedBoot> fs mkdir tests
RedBoot> fs list
212416 d---------  3 size    128 .
212416 d---------  3 size    128 ..
211392 d---------  2 size     64 tests
210368 ----------  1 size   4096 image
RedBoot>
```

fs deldir

Name

`fs deldir` — Delete filesystem directory

Synopsis

fs deldir {*directory*}

Arguments

Name	Type	Description	Default
directory	String	Pathname to directory to delete.	

Description

This command deletes a directory from the filesystem. If the directory contains files or other directories then this command will fail.

Examples

Delete directory:

```
RedBoot> fs list
212416 d---------  3 size    128 .
212416 d---------  3 size    128 ..
211392 d---------  2 size     96 tests
210368 ----------  1 size   4096 image
RedBoot> fs deldir tests
RedBoot> fs list
212416 d---------  2 size    128 .
212416 d---------  2 size    128 ..
210368 ----------  1 size   4096 image
RedBoot>
```

fs del

Name

`fs del` — Delete file

Synopsis

fs del {*file*}

Arguments

Name	Type	Description	Default
file	String	Pathname of file to delete.	

Description

This command deletes a file from the filesystem.

Examples

Change current directory:

```
RedBoot> fs list tests
211392 d---------  2 size     96 .
212416 d---------  3 size    128 ..
205760 ----------  1 size  16384 test1
RedBoot> fs del tests/test1
RedBoot> fs list tests
211392 d---------  2 size     96 .
212416 d---------  3 size    128 ..
RedBoot>
```

fs move

Name

`fs move` — Move file

Synopsis

fs move {*source*} {*dest*}

Arguments

Name	Type	Description	Default
source	String	Pathname of file to move.	
dest	String	Pathname to new file location.	

Description

This command moves a file within a filesystem. This command will fail if the destination file already exists, or is in a different filesystem.

Examples

Rename a file:

```
RedBoot> fs list tests
211392 d---------  2 size     96 .
212416 d---------  3 size    128 ..
205760 ----------  1 size  12288 test1
RedBoot> fs move tests/test1 tests/test2
RedBoot> fs list tests
211392 d---------  2 size    128 .
212416 d---------  3 size    128 ..
205760 ----------  1 size  12288 test2
RedBoot>
```

fs list

Name

`fs list` — List filesystem directory

Synopsis

fs list [*directory*]

Arguments

Name	Type	Description	Default
directory	String	Pathname to directory to list.	Current directory

Description

This command prints a list of the contents of the named directory. Each line of the listing starts with the file's inode number, which is its address in the filesystem. Following is a set of UNIX-like access flags, the first character of this will be a "d" if this entry is a directory. The third item indicates the number of links to the file. Following this is the size of the file in bytes and the last item is its name.

Examples

List the current directory:

```
RedBoot> fs list
212416 d---------  3 size    128 .
212416 d---------  3 size    128 ..
211392 ----------  1 size   4096 image
206784 d---------  2 size     96 tests
RedBoot>
```

List a subdirectory:

```
RedBoot> fs list tests
206784 d---------  2 size     96 .
212416 d---------  3 size    128 ..
205760 ----------  1 size  16384 test1
RedBoot>
```

fs write

Name

`fs write` — Write to filesystem

Synopsis

fs write [-b *mem_address*] [-l *length*] {name}

Arguments

Name	Type	Description	Default
mem_address	Number	Address of data to be written to flash.	Address of last loaded file. If not set by a load operation it must be specified.
length	Number	Length of data to be written.	Length of last loaded file.
name	String	Name of file to create.	

Description

This command is used to write data from memory to a file. If the file does not exist it will be created. If it does exist, then it will be overwritten with the new contents.

Examples

Write an area of data to a file

```
RedBoot> fs write -b 0x0606f000 -l 0x1000 image
RedBoot> fs list
212416 d---------  3 size    128 .
212416 d---------  3 size    128 ..
211392 ----------  1 size   4096 image
206784 d---------  2 size     96 tests
RedBoot>
```

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 the Section called *Flash Image System (FIS)*.

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 `-i` is specified, then the configuration database will be reset to its default state. This is also needed the first time RedBoot is installed on the target, or when updating to a newer RedBoot with different configuration keys.

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", the Section called *RedBoot Editing Commands* in Chapter 1.

If any changes are made in the configuration, then the updated data will be written back to flash after getting acknowledgment 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
dns_ip: 10.16.19.1
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 - continue (y/n)? y
... Unlock from 0x507c0000-0x507e0000: .
... Erase from 0x507c0000-0x507e0000: .
... Program from 0x0000a8d0-0x0000acd0 at 0x507c0000: .
... Lock from 0x507c0000-0x507e0000: .
```

```
RedBoot>
```

Additionally, nicknames can be used like aliases via the format %{nickname}. This allows the values stored by **fconfig** to be used directly by scripts and commands.

Depending on how your terminal program is connected and its capabilities, you might find that you are unable to use line-editing to delete the 'old' value when using the default behaviour of **fconfig *nickname*** or just plain **fconfig**, as shown in this example:

```
RedBoot> fco bootp
bootp: false_
```

The user deletes the word "false;" and enters "true" so the display looks like this:

```
RedBoot> fco bootp
bootp: true
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from ...
RedBoot> _
```

To edit when you cannot backspace, use the optional flag `-d` (for "dumb terminal") to provide a simpler interface thus:

```
RedBoot> fco -d bootp
bootp: false ? _
```

and you enter the value in the obvious manner thus:

```
RedBoot> fco -d bootp
bootp: false ? true
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from ...
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
DNS server IP address: 192.168.1.1
DNS domain name: example.com
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 - continue (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, Free Software Foundation, 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 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.

NOTE: RedBoot supports the notion of a boot script timeout, i.e. a period of time that RedBoot waits before executing the boot time script. This period is primarily to allow the possibility of canceling the script. Since a timeout value of zero (0) seconds would never allow the script to be aborted or canceled, this value is not allowed. If the timeout value is zero, then RedBoot will abort the script execution immediately.

On many targets, RedBoot may be configured to run from ROM or it may be configured to run from RAM. Other configurations are also possible. All RedBoot configurations will execute the boot script, but in certain cases it may be desirable to limit the execution of certain script commands to one RedBoot configuration or the other. This can be accomplished by prepending `{<startup type>}` to the commands which should be executed only by the RedBoot configured for the specified startup type. The following boot script illustrates this concept by having the ROM based RedBoot load and run the RAM based RedBoot. The RAM based RedBoot will then list flash images.

```
RedBoot> fco
Run script at boot: false t
Boot script:
Enter script, terminate with empty line
>> {ROM}fis load RedBoot[RAM]
>> {ROM}go
>> {RAM}fis li
>>
Boot script timeout (1000ms resolution): 2
Use BOOTP for network configuration: false
```

```
 ...
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from 0x007c0000-0x007e0000: .
... Erase from 0x007c0000-0x007e0000: .
... Program from 0xa0015030-0xa0016030 at 0x007df000: .
... Lock from 0x007c0000-0x007e0000: .
RedBoot> reset
... Resetting.
+Ethernet eth0: MAC address 00:80:4d:46:01:05
IP: 192.168.1.153, Default server: 192.168.1.10

RedBoot(tm) bootstrap and debug environment [ROM]
Red Hat certified release, version R1.xx - built 17:37:36, Aug 14 2001

Platform: IQ80310 (XScale)
Copyright (C) 2000, 2001, Free Software Foundation, Inc.

RAM: 0xa0000000-0xa2000000, 0xa001b088-0xa1fdf000 available
FLASH: 0x00000000 - 0x00800000, 64 blocks of 0x00020000 bytes each.
== Executing boot script in 2.000 seconds - enter ^C to abort
RedBoot> fis load RedBoot[RAM]
RedBoot> go
+Ethernet eth0: MAC address 00:80:4d:46:01:05
IP: 192.168.1.153, Default server: 192.168.1.10

RedBoot(tm) bootstrap and debug environment [RAM]
Red Hat certified release, version R1.xx - built 13:03:47, Aug 14 2001

Platform: IQ80310 (XScale)
Copyright (C) 2000, 2001, Free Software Foundation, Inc.

RAM: 0xa0000000-0xa2000000, 0xa0057fe8-0xa1fdf000 available
FLASH: 0x00000000 - 0x00800000, 64 blocks of 0x00020000 bytes each.
== Executing boot script in 2.000 seconds - enter ^C to abort
RedBoot> fis li
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x00000000  0x00000000  0x00040000  0x00002000
RedBoot config    0x007DF000  0x007DF000  0x00001000  0x00000000
FIS directory     0x007E0000  0x007E0000  0x00020000  0x00000000
RedBoot>
```

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

Name

`go` — Execute a program

Synopsis

go [-w *timeout*] [-c] [-n] [*start_address*]

Arguments

Name	Type	Description	Default
-w *timeout*	Number	How long to wait before starting execution.	0
-c	Boolean	Go with caches enabled.	caches off
-n	Boolean	Go with network interface stopped.	network enabled
start_address	Number	Address in memory to begin execution.	Value set by last **load** or **fis load** command.

Description

The **go** command causes RedBoot to give control of the target platform to another program. This program must execute stand alone, e.g. an eCos application or a Linux kernel.

If the -w option is used, RedBoot will print a message and then wait for a period of time before starting the execution. This is most useful in a script, giving the user a chance to abort executing a program and move on in the script.

Examples

Execute a program - *no explicit output from RedBoot*.

```
RedBoot> go 0x40040
```

Execute a program with a timeout.

```
RedBoot> go -w 10
About to start execution at 0x00000000 - abort with ^C within 10 seconds
^C
RedBoot>
```

Note that the starting address was implied (0x00000000 in this example). The user is prompted that execution will commence in 10 seconds. At anytime within that 10 seconds the user may type Ctrl+C on the console and RedBoot will abort execution and return for the next command, either from a script or the console.

exec

Name

`exec` — Execute a Linux kernel

Synopsis

exec [-w *timeout*] [-r *ramdisk_address*] [-s *ramdisk_length*] [-b *load_address* {-l *load_length*}] [-c *kernel_command_line*] [*entry_point*]

Arguments

Name	Type	Description	Default
-w *timeout*	Number	Time to wait before starting execution.	0
-r *ramdisk_address*	Number	Address in memory of "initrd"-style ramdisk - passed to Linux kernel.	*None*
-s *ramdisk_length*	Number	Length of ramdisk image - passed to Linux kernel.	*None*
-b *load_address*	Number	Address in memory of the Linux kernel image.	Value set by **load** or **fis load**
-l *load_length*	Number	Length of Linux kernel image.	*none*
-c *kernel_command_line*	String	Command line to pass to the Linux kernel.	*None*
-x		Boot kernel with endianess opposite of RedBoot endianess.	Boot kernel with same endianess as RedBoot
entry_address	Number	Starting address for Linux kernel execution	Implied by architecture

Description

The **exec** 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 which is architecture dependent(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 punctuation (spaces, etc), then it must be quoted using the double-quote character '"'. If the quote character is required, it should be written as '\"'.

The "-x" option is optionally available on some bi-endian platforms. It is used to boot a kernel built with an endianess opposite of RedBoot.

Examples

Execute a Linux kernel, passing a command line, which needs relocation. The result from RedBoot is normally quiet, with the target platform being passed over to Linux immediately.

```
RedBoot> exec -b 0x100000 -l 0x80000 -c "noinitrd root=/dev/mtdblock3 console=ttySA0"
```

Execute a Linux kernel, default entry address and no relocation required, with a timeout. The *emphasized lines* are output from the loaded kernel.

```
RedBoot> exec -c "console=ttyS0,38400 ip=dhcp nfsroot=/export/elfs-sh" -w 5
Now booting linux kernel:
Base address 0x8c001000 Entry 0x8c210000
Cmdline : console=ttyS0,38400 ip=dhcp nfsroot=/export/elfs-sh
About to start execution at 0x8x210000 - abort with ^C within 5 seconds
Linux version 2.4.10-pre6 (...) (gcc version 3.1-stdsh-010931) #3 Thu Sep 27 11:04:23 BST 2001
```

Chapter 3. Rebuilding RedBoot

Introduction

RedBoot is built as an application on top of eCos. The makefile rules for building RedBoot are part of the eCos CDL package, so it's possible to build eCos from the Configuration Tool, as well as from the command line using ecosconfig.

Building RedBoot requires only a few steps: selecting the platform and the RedBoot template, importing a platform specific configuration file, and finally starting the build.

The platform specific configuration file makes sure the settings are correct for building RedBoot on the given platform. Each platform should provide at least two of these configuration files: `redboot_RAM.ecm` for a RAM mode RedBoot configuration and `redboot_ROM.ecm` or `redboot_ROMRAM.ecm` for a ROM or ROMRAM mode RedBoot configuration. There may be additional configuration files according to the requirements of the particular platform.

The RedBoot build process results in a number of files in the install `bin` directory. The ELF file `redboot.elf` is the pricipal result. Depending on the platform CDL, there will also be generated versions of RedBoot in other file formats, such as `redboot.bin` (binary format, good when doing an update of a primary RedBoot image, see the Section called *Update the primary RedBoot flash image* in Chapter 4), `redboot.srec` (Motorola S-record format, good when downloading a RAM mode image for execution), and `redboot.img` (stripped ELF format, good when downloading a RAM mode image for execution, smaller than the .srec file). Some platforms may provide additional file formats and also relocate some of these files to a particular address making them more suitable for downloading using a different boot monitor or flash programming tools.

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

Rebuilding RedBoot using ecosconfig

To rebuild RedBoot using the ecosconfig tool, create a temporary directory for building RedBoot, name it according to the desired configuration of RedBoot, here RAM:

```
$ mkdir /tmp/redboot_RAM
$ cd /tmp/redboot_RAM
```

Create the build tree according to the chosen platform, here using the Hitachi Solution Engine 7751 board as an example:

> **Note:** It is assumed that the environment variable ECOS_REPOSITORY points to the eCos/RedBoot source tree.

```
$ ecosconfig new se7751 redboot
U CYGPKG_HAL_SH_7750, new inferred value 0
U CYGPKG_HAL_SH_7751, new inferred value 1
U CYGHWR_HAL_SH_IRQ_USE_IRQLVL, new inferred value 1
U CYGSEM_HAL_USE_ROM_MONITOR, new inferred value 0
U CYGDBG_HAL_COMMON_CONTEXT_SAVE_MINIMUM, new inferred value 0
U CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS, new inferred value 1
U CYGFUN_LIBC_STRING_BSD_FUNCS, new inferred value 0
```

```
U CYGPKG_NS_DNS_BUILD, new inferred value 0
```

Replace the platform name ("se7751") with the appropriate name for the chosen platform.

Then import the appropriate platform RedBoot configuration file, here for RAM configuration:

```
$ ecosconfig import ${ECOS_REPOSITORY}/hal/sh/se7751/VERSION/misc/redboot_RAM.ecm
$ ecosconfig tree
```

Replace architecture ("sh"), platform ("se7751") and version ("*VERSION*") with those appropriate for the chosen platform and the version number of its HAL package. Also replace the configuration name ("redboot_RAM.ecm") with that of the appropriate configuration file.

RedBoot can now be built:

```
$ make
```

The resulting RedBoot files will be in the associated install directory, in this example, `./install/bin`.

In Chapter 5 each platform's details are described in the form of shell variables. Using those, the steps to build RedBoot are:

```
export REDBOOT_CFG=redboot_ROM
export VERSION=VERSION
mkdir /tmp/${REDBOOT_CFG}
cd /tmp/${REDBOOT_CFG}
ecosconfig new ${TARGET} redboot
ecosconfig import ${ECOS_REPOSITORY}/hal/${ARCH_DIR}/${PLATFORM_DIR}/${VERSION}/misc/${REDBOOT_CFG
ecosconfig tree
make
```

To build for another configuration, simply change the *REDBOOT_CFG* definition accordingly. Also make sure the *VERSION* variable matches the version of the platform package.

Rebuilding RedBoot from the Configuration Tool

To rebuild RedBoot from the Configuration Tool, open the template window (Build->Templates) and select the appropriate Hardware target and in Packages select "redboot". Then press OK. Depending on the platform, a number of conflicts may need to be resolved before the build can be started; select "Continue".

Import the desired RedBoot configuration file from the platform HAL (File->Import...). Depending on the platform, a number of conflicts may need to be resolved before the build can be started; select "Continue". For example, if the platform selected is Hitachi SE7751 board and the RAM configuration RedBoot should be built, import the file `hal/sh/se7751/`*VERSION*`/misc/redboot_RAM.ecm`.

Save the configuration somewhere suitable with enough disk space for building RedBoot (File->Save...). Choose the name according to the RedBoot configuration, for example `redboot_RAM.ecc`.

Then start the build (Build->Library) and wait for it to complete. The resulting RedBoot files will be in the associated install directory, for the example this would be `redboot_RAM_install/bin`.

As noted above, each platform's details are described in Chapter 5. Use the information provided in the shell variables to find the configuration file - the path to it is `${ECOS_REPOSITORY}/hal/${ARCH_DIR}/${PLATFORM_DIR}/${VERSION}/misc/${REDBOOT_CFG}.ecm`, where *ECOS_REPOSITORY* points to the eCos/RedBoot sources, *VERSION* is the version of the package (usually "current") and *REDBOOT_CFG* is the desired configuration, e.g. redboot_RAM.

Chapter 4. Updating RedBoot

Introduction

RedBoot normally resides in an EPROM or, more common these days, a flash on the board. In the former case, updating RedBoot necessitates physically removing the part and reprogramming a new RedBoot image into it using prommer hardware. In the latter case, it is often possible to update RedBoot in situ using Redboot's flash management commands.

The process of updating RedBoot in situ is documented in this section. For this process, it is assumed that the target is connected to a host system and that there is a serial connection giving access to the RedBoot CLI. For platforms with a ROMRAM mode RedBoot, skip to the Section called *Update the primary RedBoot flash image*.

Note: The addresses and sizes included in the below are examples only, and will differ from those you will see. This is normal and should not cause concern.

Load and start a RedBoot RAM instance

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

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

If this image is not available, or does not work, then an alternate RAM mode image must be loaded:

```
RedBoot> load redboot_RAM.img
Entry point: 0x060213c0, address range: 0x06020000-0x060369c8
RedBoot> go
```

Note: This command loads the RedBoot image using the TFTP protocol via a network connection. Other methods of loading are available, refer to the **load** command for more details.

Note: If you expect to be doing this more than once, it is a good idea to program the RAM mode image into the flash. You do this using the **fis create** command after having downloaded the RAM mode image, but before you start it.

Some platforms support locking (write protecting) certain regions of the flash, while others do not. If your platform does not support locking, simply ignore the **fis unlock** and **fis lock** steps (the commands will not be recognized by RedBoot).

```
RedBoot> fis unlock RedBoot[RAM]
  ... Unlock from 0x00000000-0x00020000: ..
RedBoot> fis create RedBoot[RAM]
An image named 'RedBoot[RAM]' exists - continue (y/n)? y
* CAUTION * about to program 'RedBoot[RAM]'
            at 0x00020000..0x000369c7 from 0x06020000 - continue (y/n)?y
... Erase from 0x00020000-0x00040000: ..
... Program from 0x06020000-0x060369c8 at 0x00020000: ..
... Erase from 0x00070000-0x00080000: .
```

```
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
RedBoot> fis lock RedBoot[RAM]
  ... Lock from 0x00000000-0x00020000: ..
```

Update the primary RedBoot flash image

An instance of RedBoot should now be running on the target from RAM. This can be verified by looking for the mode identifier in the banner. It should be either [RAM] or [ROMRAM].

If this is the first time RedBoot is running on the board or if the flash contents has been damaged, initialize the FIS directory:

```
RedBoot> fis init -f
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x00020000-0x00070000: .....
... Erase from 0x00080000-0x00080000:
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
```

It is important to understand that the presence of a correctly initialized FIS directory allows RedBoot to automatically determine the flash parameters. Additionally, executing the steps below as stated without loading other data or using other flash commands (than possibly **fis list**) allows RedBoot to automatically determine the image location and size parameters. This greatly reduces the risk of potential critical mistakes due to typographical errors. It is still always possible to explicitly specify parameters, and indeed override these, but it is not advised.

> **Note:** If the new RedBoot image has grown beyond the slot in flash reserved for it, it is necessary to change the RedBoot configuration option CYGBLD_REDBOOT_MIN_IMAGE_SIZE so the FIS is created with adequate space reserved for RedBoot images. In this case, it is necessary to re-initialize the FIS directory as described above, using a RAM mode RedBoot compiled with the updated configuration.

Using the **load** command, download the new flash based image from the host, relocating the image to RAM::

```
RedBoot> load -r -b %{FREEMEMLO} redboot_ROM.bin
Raw file loaded 0x06046800-0x06062fe8, assumed entry at 0x06046800
```

> **Note:** This command loads the RedBoot image using the TFTP protocol via a network connection. Other methods of loading are available, refer to the load command for more details.

> **Note:** Note that the binary version of the image is being downloaded. This is to ensure that the memory after the image is loaded should match the contents of the file on the host. Loading SREC or ELF versions of the image does not guarantee this since these formats may contain holes, leaving bytes in these holes in an unknown state after the load, and thus causing a likely cksum difference. It is possible to use these, but then the step verifying the cksum below may fail.

Once the image is loaded into RAM, it should be checksummed, thus verifying that the image on the target is indeed the image intended to be loaded, and that no corruption of the image has happened. This is done using the cksum command:

```
RedBoot> cksum
Computing cksum for area 0x06046800-0x06062fe8
POSIX cksum = 2535322412 116712 (0x971df32c 0x0001c7e8)
```

Compare the numbers with those for the binary version of the image on the host. If they do not match, try downloading the image again.

Assuming the cksum matches, the next step is programming the image 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 locking, simply ignore the **fis unlock** and **fis lock** steps (the commands will not be recognized by RedBoot).

```
RedBoot> fis unlock RedBoot
  ... Unlock from 0x00000000-0x00020000: ..
RedBoot> fis create RedBoot
An image named 'RedBoot' exists - continue (y/n)? y
* CAUTION * about to program 'RedBoot'
            at 0x00000000..0x0001c7e7 from 0x06046800 - continue (y/n)? y
... Erase from 0x00000000-0x00020000: ..
... Program from 0x06046800-0x06062fe8 at 0x00000000: ..
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
RedBoot> fis lock RedBoot
  ... Lock from 0x00000000-0x00020000: ..
```

Reboot; run the new RedBoot image

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

When installing RedBoot for the first time, or after updating to a newer RedBoot with different configuration keys, it is necessary to update the configuration directory in the flash using the **fconfig** command. See the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2.

Chapter 5. Installation and Testing

AM3x/MN103E010 Matsushita MN103E010 (AM33/2.0) ASB2305 Board

Overview

RedBoot supports the debug serial port and the built in ethernet port for communication and downloads. The default serial port settings are 115200,8,N,1 with RTS/CTS flow control. RedBoot can run from either flash, and can support flash management for either the boot PROM or the system flash regions.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
PROM	[ROM]	RedBoot running from the boot PROM and able to access the system flash.	redboot_ROM.ecm
FLASH	[ROM]	RedBoot running from the system flash and able to access the boot PROM.	redboot_FLASH.ecm
RAM	[RAM]	RedBoot running from RAM and able to access the boot PROM.	redboot_RAM.ecm

Initial Installation

Unless a pre-programmed system flash module is available to be plugged into a new board, RedBoot must be installed with the aid of a JTAG interface unit. To achieve this, the RAM mode RedBoot must be loaded directly into RAM by JTAG and started, and then *that* must be used to store the ROM mode RedBoot into the boot PROM.

These instructions assume that you have binary images of the RAM-based and boot PROM-based RedBoot images available.

Preparing to program the board

If the board is to be programmed, whether via JTAG or RedBoot, some hardware settings need to be changed:

- Jumper across ST18 on the board to allow write access to the boot PROM.
- Set DIP switch S1-3 to OFF to allow RedBoot to write to the system flash.
- Set the switch S5 (on the front of the board) to boot from whichever flash is *not* being programmed. Note that the RedBoot image cannot access the flash from which it is currently executing (it can only access the

other flash).

The RedBoot binary image files should also be copied to the TFTP pickup area on the host providing TFTP services if that is how RedBoot should pick up the images it is going to program into the flash. Alternatively, the images can be passed by YMODEM over the serial link.

Preparing to use the JTAG debugger

The JTAG debugger will also need setting up:

1. Install the JTAG debugger software (WICE103E) on a PC running Windows (WinNT is probably the best choice for this) in "C:/PanaX".
2. Install the Matsushita provided "project" into the "C:/Panax/wice103e/prj" directory.
3. Install the RedBoot image files into the "C:/Panax/wice103e/prj" directory under the names redboot.ram and redboot.prom.
4. Make sure the PC's BIOS has the parallel port set to full bidirectional mode.
5. Connect the JTAG debugger to the PC's parallel port.
6. Connect the JTAG debugger to the board.
7. Set the switch on the front of the board to boot from "boot PROM".
8. Power up the JTAG debugger and then power up the board.
9. Connect the board's Debug Serial port to a computer by a null modem cable.
10. Start minicom or some other serial communication software and set for 115200 baud, 1-N-8 with hardware (RTS/CTS) flow control.

Loading the RAM-based RedBoot via JTAG

To perform the first half of the operation, the following steps should be followed:

1. Start the JTAG debugger software.
2. Run the following commands at the JTAG debugger's prompt to set up the MMU registers on the CPU.

```
ed 0xc0002000, 0x12000580

ed 0xd8c00100, 0x8000fe01
ed 0xd8c00200, 0x21111000
ed 0xd8c00204, 0x00100200
ed 0xd8c00208, 0x00000004

ed 0xd8c00110, 0x8400fe01
ed 0xd8c00210, 0x21111000
ed 0xd8c00214, 0x00100200
ed 0xd8c00218, 0x00000004

ed 0xd8c00120, 0x8600ff81
ed 0xd8c00220, 0x21111000
ed 0xd8c00224, 0x00100200
ed 0xd8c00228, 0x00000004

ed 0xd8c00130, 0x8680ff81
ed 0xd8c00230, 0x21111000
ed 0xd8c00234, 0x00100200
```

```
ed 0xd8c00238, 0x00000004

ed 0xd8c00140, 0x9800f801
ed 0xd8c00240, 0x00140000
ed 0xd8c00244, 0x11011100
ed 0xd8c00248, 0x01000001

ed 0xda000000, 0x55561645
ed 0xda000004, 0x000003c0
ed 0xda000008, 0x9000fe01
ed 0xda00000c, 0x9200fe01
ed 0xda000000, 0xa89b0654
```

3. Run the following commands at the JTAG debugger's prompt to tell it what regions of the CPU's address space it can access:

```
ex 0x80000000,0x81ffffff,/mexram
ex 0x84000000,0x85ffffff,/mexram
ex 0x86000000,0x867fffff,/mexram
ex 0x86800000,0x87ffffff,/mexram
ex 0x8c000000,0x8cffffff,/mexram
ex 0x90000000,0x93ffffff,/mexram
```

4. Instruct the debugger to load the RAM RedBoot image into RAM:

```
_pc=90000000
u_pc
rd redboot.ram,90000000
```

5. Load the boot PROM RedBoot into RAM:

```
rd redboot.prom,91020000
```

6. Start RedBoot in RAM:

```
g
```

   Note that RedBoot may take some time to start up, as it will attempt to query a BOOTP or DHCP server to try and automatically get an IP address for the board. Note, however, that it should send a plus over the serial port immediately, and the 7-segment LEDs should display "rh 8".

Loading the boot PROM-based RedBoot via the RAM mode RedBoot

Once the RAM mode RedBoot is up and running, it can be communicated with by way of the serial port. Commands can now be entered directly to RedBoot for flashing the boot PROM.

1. Instruct RedBoot to initialise the boot PROM:

```
RedBoot> fi init
```

2. Write the previously loaded redboot.prom image into the boot PROM:

```
RedBoot> fi write -f 0x80000000 -b 0x91020000 -l 0x00020000
```

3. Check that RedBoot has written the image:

```
RedBoot> dump -b 0x91020000
RedBoot> dump -b 0x80000000
```

   Barring the difference in address, the two dumps should be the same.

4. Close the JTAG software and power-cycle the board. The RedBoot banners should be displayed again over the serial port, followed by the RedBoot prompt. The boot PROM-based RedBoot will now be running.

5. Power off the board and unjumper ST18 to write-protect the contents of the boot PROM. Then power the board back up.

6. Run the following command to initialise the system flash:

```
RedBoot> fi init
```

   Then program the system flash based RedBoot into the system flash:

```
RedBoot> load -r -b %{FREEMEMLO} redboot_FLASH.bin
RedBoot> fi write -f 0x84000000 -b %{FREEMEMLO} -l 0x00020000
```

   NOTE: RedBoot arranges the flashes on booting such that they always appear at the same addresses, no matter which one was booted from.

7. A similar sequence of commands can be used to program the boot PROM when RedBoot has been booted from an image stored in the system flash.

```
RedBoot> load -r -b %{FREEMEMLO} /tftpboot/redboot_ROM.bin
RedBoot> fi write -f 0x80000000 -b %{FREEMEMLO} -l 0x00020000
```

   See the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2 for details on configuring the RedBoot in general, and also the Section called *Flash Image System (FIS)* in Chapter 2 for more details on programming the system flash.

Additional Commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this architecture (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for ASB2305 board are:

-w *<time>*

Wait time in seconds before starting kernel

-c *"params"*

Parameters passed to kernel

<addr>

Kernel entry point, defaulting to the entry point of the last image loaded

The parameter string is stored in the on-chip memory at location 0x8C001000, and is prefixed by “cmdline:” if it was supplied.

Memory Maps

RedBoot sets up the following memory map on the ASB2305 board.

NOTE: The regions mapped between 0x80000000-0x9FFFFFFF are cached by the CPU. However, all those regions can be accessed uncached by adding 0x20000000 to the address.

```
Physical Address Range   Description
----------------------   -----------
0x80000000 - 0x9FFFFFFF  Cached Region
0x80000000 - 0x81FFFFFF  Boot PROM
0x84000000 - 0x85FFFFFF  System Flash
0x86000000 - 0x86007FFF  64Kbit Sys Config EEPROM
0x86F90000 - 0x86F90003  4x 7-segment LEDs
0x86FA0000 - 0x86FA0003  Software DIP Switches
0x86FB0000 - 0x86FB001F  PC16550 Debug Serial Port
0x8C000000 - 0x8FFFFFFF  On-Chip Memory (repeated 16Kb SRAM)
0x90000000 - 0x93FFFFFF  SDRAM
0x98000000 - 0x9BFFFFFF  Paged PCI Memory Space (64Mb)
0x9C000000 - 0x9DFFFFFF  PCI Local SRAM (32Mb)
0x9E000000 - 0x9E03FFFF  PCI I/O Space
0x9E040000 - 0x9E0400FF  AM33-PCI Bridge Registers
0x9FFFFFF4 - 0x9FFFFFF7  PCI Memory Page Register
0x9FFFFFF8 - 0x9FFFFFFF  PCI Config Registers
0xA0000000 - 0xBFFFFFFF  Uncached Mirror Region
0xC0000000 - 0xDFFFFFFF  CPU Control Registers
```

The ASB2305 HAL makes use of the on-chip memory in the following way:

```
0x8C000000 - 0x8C0000FF  hal_vsr_table
0x8C000100 - 0x8C0001FF  hal_virtual_vector_table
0x8C001000 -             Linux command line (RedBoot exec command)
           - 0x8C003FFF  Emergency DoubleFault Exception Stack
```

Currently the CPU's interrupt table lies at the beginning of the RedBoot image, which must therefore be aligned to a 0xFF000000 mask.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=asb2305
export ARCH_DIR=mn10300
export PLATFORM_DIR=asb2305
```

The names of configuration files are listed above with the description of the associated modes.

ARM/ARM7 ARM Evaluator7T

Overview

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

The following RedBoot configurations are supported:

Configuration	Mode	Description	File

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from flash address 0x20000, with ARM Boot Monitor in flash boot sector.	redboot_ROMA.ecm

Initial Installation

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

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.

Special RedBoot Commands

None.

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)
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=e7t
export ARCH_DIR=arm
export PLATFORM_DIR=e7t
```

The names of configuration files are listed above with the description of the associated modes.

ARM/ARM7+ARM9 ARM Integrator

Overview

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

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm

Initial Installation

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

Quick download instructions

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

- Set DIP switch S1[1] to the ON position and reset or power the board up. You will see the bootPROM startup message on serial port A (J14):

```
Initialising...


ARM bootPROM [Version 1.3] Rebuilt on Jun 26 2001 at 22:04:10
Running on a Integrator Evaluation Board
Board Revision V1.0, ARM966E-S Processor
Memory Size is 16MBytes, Flash Size is 32MBytes
Copyright (c) ARM Limited 1999 - 2001. All rights reserved.
Board designed by ARM Limited
Hardware support provided at http://www.arm.com/
For help on the available commands type ? or h
boot Monitor >
```

- Issue the FLASH ROM load command:

```
boot Monitor > L
Load Motorola S-Records into flash

Deleting Image 0

The S-Record loader only accepts input on the serial port.
Type Ctrl/C to exit loader.
```

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

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

  When complete, type Ctrl-C and you should see something similar to:

```
.............................
.............................
................
Downloaded 5,394 records in 81 seconds.

Overwritten block/s
    0

boot Monitor >
```

- Set DIP switch S1[1] to the OFF position and reboot the board and you should see the RedBoot banner.

Special RedBoot Commands

None.

Memory Maps

RedBoot sets up the following memory map on the Integrator 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).

```
ARM7TDMI
--------

Physical Address Range  C B  Description
----------------------- - -  -----------
0x00000000 - 0x0007ffff N N  SSRAM
0x00080000 - 0x0fffffff N N  SDRAM (depends on part fitted)
0x10000000 - 0x1fffffff N N  System control and peripheral registers
0x20000000 - 0x23ffffff N N  Boot ROM (contains boot Monitor)
0x24000000 - 0x27ffffff N N  FLASH ROM (contains RedBoot)
0x28000000 - 0x2bffffff N N  SSRAM echo area
0x40000000 - 0x5fffffff N N  PCI Memory access windows
0x60000000 - 0x60ffffff N N  PCI IO access window
0x61000000 - 0x61ffffff N N  PCI config space window
0x62000000 - 0x6200ffff N N  PCI bridge register window
0x80000000 - 0x8fffffff N N  SDRAM echo area (used for PCI accesses)


ARM966E
-------

Physical Address Range  C B  Description
----------------------- - -  -----------
0x00000000 - 0x000fffff N N  SSRAM
0x00100000 - 0x0fffffff N N  SDRAM (depends on part fitted)
0x10000000 - 0x1fffffff N N  System control and peripheral registers
0x20000000 - 0x23ffffff N N  Boot ROM (contains boot Monitor)
0x24000000 - 0x27ffffff N N  FLASH ROM (contains RedBoot)
0x28000000 - 0x2bffffff N N  SSRAM echo area
0x40000000 - 0x5fffffff N N  PCI Memory access windows
0x60000000 - 0x60ffffff N N  PCI IO access window
0x61000000 - 0x61ffffff N N  PCI config space window
0x62000000 - 0x6200ffff N N  PCI bridge register window
0x80000000 - 0x8fffffff N N  SDRAM echo area (used for PCI accesses)
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=integrator
export ARCH_DIR=arm
```

```
export PLATFORM_DIR=integrator
```

The names of configuration files are listed above with the description of the associated modes.

ARM/ARM7+ARM9 ARM PID Board and EPI Dev7+Dev9

Overview

RedBoot uses either of the serial ports. The default serial port settings are 38400,8,N,1. Management of on-board flash is also supported.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

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 mode image of RedBoot and run it. Initialize the flash image directory: **fis init** Then download the ROM version of RedBoot and program it into flash:

```
RedBoot> load -b %{FREEMEMLO} -m ymodem
RedBoot> fi cr RedBoot
```

Special RedBoot Commands

None.

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
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=pid
export ARCH_DIR=arm
export PLATFORM_DIR=pid
```

The names of configuration files are listed above with the description of the associated modes.

ARM/ARM7 Atmel AT91 Evaluation Boards (EBXX)

Overview

RedBoot support is available for the EB40, EB40A, EB42 and EB55 boards. By default all these boards are shipped with only 256Kbytes of RAM. To minimize the amount of RAM used by RedBoot, only very basic flash management is provided, comprising of just the **fis erase** and **fis write** commands.

RedBoot supports both serial ports. On all AT91 evaluation boards, serial port A requires a straight through cable to connect with a PC, whereas serial port B requires a null modem cable. If you fail to be able to connect to Angel in the instructions below when installing RedBoot, be sure to verify you are using the appropriate cable for the serial port. The default serial port settings for RedBoot are 38400,8,N,1.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm

Initial Installation Method

RedBoot installation is essentially the same for all boards, however the details differ slightly. Please make sure you follow the directions from the correct section below. Any errors could result in an unusable board.

Installing RedBoot on the EB40

This development board comes with ARM's debug tool, Angel, installed in flash. At this time, Angel will not be replaced. Rather, RedBoot will be placed in the alternate half of flash. Switch SW1 is used to select which monitor to boot. Once RedBoot is installed, selecting SW1 to `lower mem` will choose Angel, whereas selecting SW1 to `upper mem` will choose RedBoot.

Set SW1 to `lower mem` and connect serial port A to a host computer. Using GDB from the host and Angel on the board, download and run the RAM mode image of RedBoot to the board.

```
arm-elf-gdb redboot_RAM.elf
(gdb) tar rdi s=/dev/ttyS0
Angel Debug Monitor (serial) 1.04 (Advanced RISC Machines SDT 2.5) for
AT91EB40 (2.00)
Angel Debug Monitor rebuilt on Apr 07 2000 at 12:40:31
Serial Rate:   9600
Connected to ARM RDI target.
(gdb) set $cpsr=0xd3
(gdb) load
Loading section .rom_vectors, size 0x40 lma 0x2020000
Loading section .text, size 0x7fd8 lma 0x2020040
Loading section .rodata, size 0x15a0 lma 0x2028018
Loading section .data, size 0x2e4 lma 0x20295b8
Start address 0x2020040 , load size 39068
Transfer rate: 6250 bits/sec, 500 bytes/write.
(gdb) cont
Continuing.
```

Once RedBoot is started, the GDB session connected with Angel must be suspended (this can be done using Ctrl-Z) or terminated (with Ctrl-C or the Windows task manager). Follow this by connecting to the board using a terminal emulator such as hyperterminal or minicom at 38400-8N1. At this point, RedBoot will be running on the board in RAM.

```
RedBoot> version

RedBoot(tm) bootstrap and debug environment [RAM]
Non-certified release, version UNKNOWN - built 14:09:27, Jul 20 2001

Platform: Atmel AT91/EB40 (ARM7TDMI)
Copyright (C) 2000, 2001, Free Software Foundation, Inc.

RAM: 0x02000000-0x02080000, 0x020116d8-0x0207fd00 available
FLASH: 0x01010000 - 0x01020000, 256 blocks of 0x00000100 bytes each.

RedBoot>
```

Now, download the ROM mode image.

```
RedBoot> load -m ymodem -b %{FREEMEMLO}
```

Use your terminal emulator to send the file redboot_ROM.srec via YModem. e.g. `Transfer->Send File` in Hyperterminal, or `Ctrl-A S` in minicom. Finally, program it to flash.

```
RedBoot> fi wr -f 0x01010000 -b %{FREEMEMLO} -l 0xe100
```

SW1 should now be set to `upper mem` to select booting with RedBoot rather than Angel. Finally, press the "reset" pushbutton and RedBoot should come up on the board.

Installing RedBoot on the EB40A, EB42 or EB55

These development boards come with ARM's debug tool, Angel, installed in flash. At this time, Angel will not be replaced. Rather, RedBoot will be placed in the alternate half of flash. Jumper JP1 is used to select which monitor to boot. Once RedBoot is installed, setting JP1 to `STD` will choose Angel, whereas setting JP1 to `USER` will choose RedBoot.

Set JP1 to `STD` and connect serial port A to a host computer. Using GDB from the host and Angel on the board, download the RAM mode image of RedBoot to the board, and start it using the 'cont' command.

```
arm-elf-gdb redboot_RAM.elf
(gdb) tar rdi s=/dev/ttyS0
Angel Debug Monitor (serial) 1.04 (Advanced RISC Machines SDT 2.5) for AT91EB55 (2.20)
Angel Debug Monitor rebuilt on Feb 03 2002 at 16:10:20
Serial Rate:   9600
Connected to ARM RDI target.
(gdb) set $cpsr=0xd3
(gdb) load
Loading section .rom_vectors, size 0x40 lma 0x2008000
Loading section .text, size 0xb0b8 lma 0x2008040
Loading section .rodata, size 0x1c27 lma 0x20130f8
Loading section .data, size 0x5f0 lma 0x2014d20
Start address 0x2008040, load size 54031
Transfer rate: 6264 bits/sec, 500 bytes/write.
(gdb) cont
Continuing.
```

Once RedBoot is started, the GDB session connected with Angel must be suspended (this can be done using Ctrl-Z) or terminated (with Ctrl-C or the Windows task manager). Follow this by connecting to the board using a terminal emulator such as hyperterminal or minicom at 38400-8N1. At this point, RedBoot will be running on the board in RAM.

```
RedBoot> version

RedBoot(tm) bootstrap and debug environment [RAM]
Non-certified release, version UNKNOWN - built 16:58:52, May  7 2003

Platform: Atmel AT91/EB55 (ARM7TDMI)
Copyright (C) 2000, 2001, 2002, Free Software Foundation, Inc.

RAM: 0x02000000-0x02040000, 0x020068a8-0x0203f000 available
FLASH: 0x01010000 - 0x01200000, 31 blocks of 0x00010000 bytes each.

RedBoot>
```

Now, download the ROM mode image.

```
RedBoot> load -m ymodem -b %{FREEMEMLO}
```

Use your terminal emulator to send the file redboot_ROM.srec via YModem. e.g. `Transfer->Send File` in Hyperterminal, or `Ctrl-A S` in minicom. Finally, program it to flash.

```
RedBoot> fi wr -f 0x01100000 -b %{FREEMEMLO} -l 0x10000
```

Set JP1 to the `USER` setting, press the "reset" pushbutton and RedBoot should come up on the board.

Special RedBoot Commands

None.

Memory Maps

This processor has no MMU, so the only memory map is for physical addresses.

The memory layout of the EB40 is as follows:

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x00000fff    On-chip SRAM
0x01000000 - 0x0101ffff    Flash
0x02000000 - 0x0207ffff    RAM
0xffe00000 - 0xffffffff    I/O registers
```

The flash based RedBoot image occupies virtual addresses 0x01010000 - 0x0101dfff.

The memory layout of the EB40A is as follows:

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x0003ffff    On-chip SRAM
0x01000000 - 0x011fffff    Flash
0x02000000 - 0x02ffffff    External SRAM (optional)
0xffe00000 - 0xffffffff    I/O registers
```

The flash based RedBoot image occupies virtual addresses 0x01100000 - 0x0110ffff.

The memory layout of the EB42 and EB55 is as follows:

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x00001fff    On-chip SRAM
0x01000000 - 0x011fffff    Flash
0x02000000 - 0x0203ffff    RAM
0xffe00000 - 0xffffffff    I/O registers
```

The flash based RedBoot image occupies virtual addresses 0x01100000 - 0x0110ffff.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export ARCH_DIR=arm

export TARGET=eb40
export PLATFORM_DIR=at91/eb40

export TARGET=eb40a
export PLATFORM_DIR=at91/eb40a

export TARGET=eb42
export PLATFORM_DIR=at91/eb42

export TARGET=eb55
```

```
export PLATFORM_DIR=at91/eb55
```

Use just one of the `TARGET` and `PLATFORM_DIR` variable pairs only.

The names of configuration files are listed above with the description of the associated modes.

When reprogramming RedBoot using RedBoot itself, you should load a RedBoot RAM image as normal, and load the new ROM image into RAM. However before programming the new image into Flash you *must* switch SW1 to lower mem (EB40) or set JP1 to STD (EB40A, EB42, EB55) before writing to Flash.

Warning!

Failure to set SW1 to `lower mem` (EB40) or JP1 to `STD` (EB40A, EB42, EB55) will cause the installation of RedBoot to overwrite Angel, thus making the board *unbootable*. Only hardware JTAG can restore the board once in this state.

ARM/ARM7 Atmel JTST Evaluation Board (AT572D740-DK1)

Overview

RedBoot support is available for the JTST board. By default this board is shipped with 256Kbytes of external SRAM. To minimize the amount of RAM used by RedBoot, only very basic flash management is provided, comprising of just the **fis erase** and **fis write** commands.

RedBoot supports two serial ports. The default serial port settings for RedBoot are 115200,8,N,1.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Installing a RedBoot image on the JTST

This development board comes with RedBoot installed on flash. To install a new version of RedBoot or another binary image in flash you must start a GDB session setting a remote target and load and run the **jtstflashd.elf** diopsis application. This is a daemon that listens on JTST serial port 1. On the PC side you must use the **jtstflash.exe** (both linux and windows PC are supported) to flash the image on JTST. The sources for win32 and linux/cygwin versions of this host tool can be found in the support directory of the jtst hal. The binaries can be found along with the binaries for redboot on the eCos website at http://ecos.sourceware.org/ecos/boards/redbootbins/at91jtst/ (http://ecos.sourceware.org/ecos/boards/redbootbins/at91jtst/index.html)

When the jtstflashd.elf is started, the user should open the jumper JP5 to write in the second half (512Kbytes) of the flash, in this way the original RedBoot image is preserved.

GDB console

```
arm-elf-gdb jtstflash.elf
(gdb) set remotebaud 115200
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00502a44 in ?? ()
(gdb) load
Loading section .internal_vectors, size 0x1c4 lma 0x160
Loading section .rom_vectors, size 0x40 lma 0x606000
Loading section .text, size 0x14198 lma 0x606040
Loading section .rodata, size 0xb6c lma 0x61a1d8
Loading section .data, size 0x498 lma 0x61ad44
Start address 0x606040, load size 86944
Transfer rate: 77283 bits/sec, 301 bytes/write.
(gdb) c
Continuing.
* JTST FLASH PROGRAMMER
* opening usart port 1
...
```

PC console

```
jtstflash mybinaryimage.bin
* binary len 79536 bytes flash add 0x500000..
* flash id check ok
* erasing space address 0x500000... please wait
* flash erase check ok
* start programming 79536 bytes.
```

Special RedBoot Commands

None.

Memory Maps

This processor has no MMU, so the only memory map is for physical addresses.

The memory layout of the JTST after bootstrap is as follows:

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x00007fff    On-chip SRAM
0x00500000 - 0x0057ffff    Flash
0x00600000 - 0x0063ffff    External SRAM
0x00410000 - 0x0042fffc    On-chip Magic Data Memory Left
0x00430000 - 0x0043fffc    On-chip Magic Data Memory Right
0x00430000 - 0x0044fffc    On-chip Magic Program Memory
0x00490000 - 0x00490ffc    On-chip Arm/Magic Data Exchange Left
0x004A0000 - 0x004A0ffc    On-chip Arm/Magic Data Exchange Right
```

```
0x00450000 - 0x0045003c    Magic I/O registers
0x00460000 - 0x0046000c    Magic Control registers
0xffe00000 - 0xffffffff    I/O registers
```

ARM/ARM7 Cirrus Logic EP7xxx (EDB7211, EDB7212, EDB7312)

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 EDB7xxx for the NOR flash only.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector (EDB7312 only).	redboot_ROMRAM.ecm

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.

Special RedBoot Commands

None.

Memory Maps

The MMU page tables and LCD display buffer, if enabled, 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 (EDB7211, EDB7212)
0x00000000 - 0x00ffffff    NOR Flash (EDB7312)
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 (EDB7211, EDB7212)
0xc0000000 - 0xc0ffffff    DRAM (EDB7312)

Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM
0x00000000 - 0x00fcffff  Y Y  DRAM (EDB7312)
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 (EDB7211, EDB7212)
0xe0000000 - 0xe0ffffff  Y Y  NOR Flash (EDB7312)
0xf0000000 - 0xf1ffffff  Y Y  NAND Flash

The flash based RedBoot image occupies virtual addresses 0xe0000000 - 0xe003ffff.
```

Platform Resource Usage

The EP7xxx timer #2 is used as a polled timer to provide timeout support for network and XModem file transfers.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=edb7211
export TARGET=edb7212
export TARGET=edb7312
export ARCH_DIR=arm
export PLATFORM_DIR=edb7xxx
```

Use one of the TARGET settings only.

The names of configuration files are listed above with the description of the associated modes.

ARM/ARM9 Agilent AAED2000

Overview

RedBoot supports the serial and ethernet ports on the board. The default serial port settings are 38400,8,N,1. RedBoot also supports flash management on the AAED2000.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_primary_ROMRAM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_primary_RAM.ecm

Initial Installation Method

It is possible to install RedBoot in one of two ways. Either as the primary bootmonitor on the board (installed to blocks 0-1 of the flash) or as the secondary bootmonitor on the board (installed to blocks 1-2 of the flash).

Presently, only the former method is supported.

RedBoot as Primary Bootmonitor

RedBoot is installed in flash using the on-board ARM Boot Monitor.

Boot the board while pressing SPACE. This should bring up the Boot Monitor:

```
ARM bootPROM [Version 1.3] Rebuilt on Jul 16 2001 at 16:21:36
Running on a P920 board Evaluation Board
Board Revision V1.0, ARM920T processor Processor
Memory Size is 32MBytes, Flash Size is 32MBytes
Copyright (c) ARM Limited 1999 - 2001. All rights reserved.
Board designed by ARM Limited
Hardware support provided at http://www.arm.com/
For help on the available commands type ? or h
boot Monitor >
```

Download the RAM mode image of RedBoot configured as a primary bootmonitor using the ARM bootmonitor's SREC-download command:

```
boot Monitor > m
Load Motorola S-Record image into memory and execute it
The S-Record loader only accepts input on the serial port.
Record addresses must be between 0x00008000 and 0x01E0F510.
Type Ctrl/C to exit loader.
```

Use the terminal emulator's ASCII upload command, or (on Linux) simply cat the file to the serial port:

```
$ cat redboot_primary_RAM/redboot.srec >/dev/ttyS1
```

You should see RedBoot start up:

```
FLASH configuration checksum error or invalid key
Ethernet eth0: MAC address 00:30:d3:03:04:99
IP: 192.168.42.111, Default server: 192.168.42.3

RedBoot(tm) bootstrap and debug environment [RAM]
Non-certified release, version UNKNOWN - built 13:15:40, Nov  9 2001

Platform: AAED2000 system (ARM9) [Primary]
Copyright (C) 2000, 2001, Free Software Foundation, Inc.

RAM: 0x00000000-0x01f80000, 0x0006f208-0x01f51000 available
FLASH: 0x60000000 - 0x62000000, 256 blocks of 0x00020000 bytes each.
RedBoot>
```

As can be seen from the output above, the network has been configured to give the board an IP address and information about the default server. If things are not set up on your network, you can still continue, but use the Y-modem download method when loading the RedBoot ROMRAM mode image. Now initialize RedBoot's FIS:

```
RedBoot> fis init
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
... Erase from 0x61fe0000-0x62000000: .
... Program from 0x01f5f000-0x01f5f300 at 0x61fe0000: .
```

Download the ROMRAM mode image of RedBoot via ethernet:

```
RedBoot> load -b %{FREEMEMLO} redboot_primary_ROMRAM/redboot.srec
```

or using serial Y-modem protocol:

```
RedBoot> load -mode ymodem -b %{FREEMEMLO}
```

(Use the terminal emulator's Y-modem upload command to send the file `redboot_primary_ROMRAM/redboot.srec`.) When the image has been downloaded, program it into flash:

```
Address offset = 0x00ff8000
Entry point: 0x00008040, address range: 0x00008000-0x0002da80
RedBoot> fi cr RedBoot
An image named 'RedBoot' exists - continue (y/n)? y
* CAUTION * about to program 'RedBoot'
            at 0x60000000..0x6003ffff from 0x00100000 - continue (y/n)? y
... Erase from 0x60000000-0x60040000: ..
... Program from 0x00100000-0x00140000 at 0x60000000: ..
... Erase from 0x61fe0000-0x62000000: .
... Program from 0x01f5f000-0x01f7f000 at 0x61fe0000: .
```

Now reset the board. You should see the RedBoot banner.

Special RedBoot Commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the AAED2000 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

The parameters for kernel image base and size are automatically set after a load operation. So one way of starting the kernel would be:

```
RedBoot> load -r -b 0x100000 zImage
Raw file loaded 0x00100000-0x001a3d6c
RedBoot> exec -c "console=ttyAC0,38400"
Using base address 0x00100000 and length 0x000a3d6c
Uncompressing Linux.....
```

An image could also be put in flash and started directly:

```
RedBoot> exec -b 0x60040000 -l 0xc0000 -c "console=ttyAC0,38400"
Uncompressing Linux.....
```

Memory Maps

The MMU page tables are located at 0x4000.

> **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    Flash
0x10000000 - 0x100fffff    Ethernet
0x30000000 - 0x300fffff    Board registers
0x40000000 - 0x4fffffff    PCMCIA Slot (0)
0x50000000 - 0x5fffffff    Compact Flash Slot (1)
0x80000000 - 0x800037ff    I/O registers
0xb0060000 - 0xb00fffff    On-chip SRAM
0xf0000000 - 0xfd3fffff    SDRAM

Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01f7ffff  Y Y  SDRAM
0x01f80000 - 0x01ffffff  Y Y  SDRAM (used for LCD frame buffer)
0x10000000 - 0x100fffff  N N  Ethernet
```

```
0x30000000 - 0x300fffff  N N  Board registers
0x40000000 - 0x4fffffff  N N  PCMCIA Slot (0)
0x50000000 - 0x5fffffff  N N  Compact Flash Slot (1)
0x60000000 - 0x61ffffff  N N  Flash
0x80000000 - 0x800037ff  N N  I/O registers
0xf0000000 - 0xffffffff  N N  SDRAM (uncached)
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=aaed
export ARCH_DIR=arm
export PLATFORM_DIR=arm9/aaed2000
```

The names of configuration files are listed above with the description of the associated modes.

ARM/ARM9 Altera Excalibur

Overview

RedBoot supports the serial port labelled P2 on the board. The default serial port settings are 57600,8,N,1. RedBoot also supports flash management on the Excalibur.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm
REDBOOT	[ROMRAM]	RedBoot running from top of RAM, but contained in the board's flash boot sector.	redboot_REDBOOT.ecm

NOTE: RedBoot is currently hardwired to use a 128MB SDRAM SIMM module.

Initial Installation Method

A Windows utility (exc_flash_programmer.exe) is used to program flash using the ByteBlasterMV JTAG unit. See board documentation for details on in situ flash programming.

For ethernet to work (under Linux) the following jumper settings should be used on a REV 2 board:

SW2-9 : OFF
U179 : 2-3
JP14-18 : OPEN
JP40-41 : 2-3
JP51-55 : 2-3

Flash management

The ROMRAM and REDBOOT configurations supported on this platform differ only in the memory layout (ROMRAM configuration runs RedBoot from 0x00008000 while REDBOOT configuration runs RedBoot from 0x07f80000). The REDBOOT configuration allows applications to be loaded and run from address 0x00008000.

Special RedBoot Commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the Excalibur 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

The parameters for kernel image base and size are automatically set after a load operation. So one way of starting the kernel would be:

```
RedBoot> load -r -b 0x100000 zImage
Raw file loaded 0x00100000-0x001a3d6c
RedBoot> exec -c "console=ttyUA0,57600"
Using base address 0x00100000 and length 0x000a3d6c
Uncompressing Linux.....
```

An image could also be put in flash and started directly:

```
RedBoot> exec -b 0x40400000 -l 0xc0000 -c "console=ttyUA0,57600"
Uncompressing Linux.....
```

Memory Maps

The MMU page tables are located at 0x4000.

> **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 - 0x0805ffff    On-chip SRAM
0x40000000 - 0x40ffffff    Flash
0x7fffc000 - 0x7fffffff    I/O registers
0x80000000 - 0x8001ffff    PLD

Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x07ffffff  Y Y  SDRAM
0x08000000 - 0x0805ffff  Y Y  On-chip SRAM
0x40000000 - 0x403fffff  N Y  Flash
0x7fffc000 - 0x7fffffff  N N  I/O registers
0x80000000 - 0x8001ffff  N N  PLD
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=excalibur_arm9
export ARCH_DIR=arm
export PLATFORM_DIR=arm9/excalibur
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA110) Intel EBSA 285

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

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`

Communication Channels

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

Special RedBoot Commands

None.

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 - 0x01ffffff  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
```

Platform Resource Usage

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

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=ebsa285
export ARCH_DIR=arm
export PLATFORM_DIR=ebsa285
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA1100) Intel Brutus

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

Device programmer is used to program socketed flash parts.

Special RedBoot Commands

None.

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
```

Platform Resource Usage

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

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=brutus
export ARCH_DIR=arm
export PLATFORM_DIR=sa11x0/brutus
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA1100) Intel SA1100 Multimedia Board

Overview

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

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

A device programmer is used to program socketed flash parts.

Special RedBoot Commands

None.

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
```

Platform Resource Usage

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

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=sa1100mm
export ARCH_DIR=arm
export PLATFORM_DIR=sa11x0/sa1100mm
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA1110) Intel SA1110 (Assabet)

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

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.

Special RedBoot Commands

None.

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
```

Platform Resource Usage

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

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=assabet
export ARCH_DIR=arm
export PLATFORM_DIR=sa11x0/assabet
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA11X0) Bright Star Engineering commEngine and nanoEngine

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
POST	[ROM]	RedBoot running from the first free flash block at 0x40000.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

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 a special configuration of the ROM mode, starting at offset 0x40000 in the flash.

Briefly, the POST-configuration RedBoot image lives in flash following the BSE firmware. The BSE firmware is configured, using its standard **bootcmd** command, to run RedBoot at startup.

Download Instructions

You can perform the initial load of the POST-configuration 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 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 in order that the Flash Image System contains an appropriate description of the flash entries.

Cohabiting with POST in Flash

The configuration 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. It is reported by RedBoot's **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:

```
include/pkgconf/mlt_arm_sa11x0_nano_post.h
include/pkgconf/mlt_arm_sa11x0_nano_post.ldi
include/pkgconf/mlt_arm_sa11x0_nano_post.mlt
```

It is these you should edit if you wish to move the execution address from 0x50040000 in the POST configuration. Startup mode 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 mode RedBoot.

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
(reserved)        0x50000000  0x50000000  0x00040000  0x00000000
RedBoot[post]     0x50040000  0x00100000  0x00020000  0x50040040
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.

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.

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 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 0x01e00000-0x01effffff. 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.

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.

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.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=nano
export ARCH_DIR=arm
export PLATFORM_DIR=sa11x0/nano
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA11X0) Compaq iPAQ PocketPC

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm
WinCE	[RAM]	RedBoot running from RAM, started from OSloader.	redboot_WinCE.ecm

Initial Installation

RedBoot ROM and WinCE mode images are needed by the installation process.

Installing RedBoot on the iPAQ using Windows/CE

The Windows/CE environment originally shipped with the iPAQ contains a hidden mini-loader, sometimes referred to as the "Parrot" loader. This loader can be started by holding down the action button (the joypad) while resetting the unit or when powering on. At this point, a blue bird will appear on the LCD screen. Also at this point, a simple loader can be accessed over the serial port at 115200/8N1. Using this loader, the contents of the iPAQ flash memory can be saved to a Compact Flash memory card.

> **NOTE:** We have only tested this operation with a 32Mbyte CF memory card. Given that the backup will take 16MBytes + 1KByte, something more than a 16MByte card will be required.

Use the "r2c" command to dump Flash contents to the CF memory card. Once this completes, RedBoot can be installed with no fear since the Parrot loader can be used to restore the Flash contents at a later time.

If you expect to completely recover the state of the iPAQ Win/CE environment, then HotSync should be run to backup all "RAM" files as well before installing RedBoot.

The next step in installing RedBoot on the iPAQ actually involves Windows/CE, which 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 its 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.

Installing RedBoot on the iPAQ - using the Compaq boot loader

This method of installation is no longer supported. If you have previously installed either the Compaq boot loader or older versions of RedBoot, restore the Win/CE environment and proceed as outlined above.

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 [ROM]
Non-certified release, version UNKNOWN - built 06:17:41, Mar 19 2001
Platform: Compaq iPAQ Pocket PC (StrongARM 1110)

Copyright (C) 2000, 2001, Free Software Foundation, Inc.

RAM: 0x00000000-0x01fc0000, 0x0001f200-0x01f70000 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
```

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, it is strongly recommended you save your WinCE FLASH contents as outlined above using the "parrot" loader, or by 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 redboot_ROM.bin
RedBoot> fi loc -f 0x50000000 -l 0x40000
RedBoot> fis init
RedBoot> fi unl -f 0x50040000 -l 0x40000
RedBoot> fi cr RedBoot -b 0x100000
RedBoot> fi loc -f 0x50040000 -l 0x40000
RedBoot> reset
```

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.

Restoring Windows/CE

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

Additional commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). 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.

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) [organized as below]
  0x000000 - 0x0003ffff      Parrot Loader
  0x040000 - 0x0007ffff      RedBoot
  0xf80000 - 0x00fbffff      Fconfig data
  0xfc0000 - 0x00ffffff      FIS directory
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
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=ipaq
export ARCH_DIR=arm
export PLATFORM_DIR=sa11x0/ipaq
```

The names of configuration files are listed above with the description of the associated modes.

ARM/StrongARM(SA11X0) Intrinsyc CerfCube

Overview

RedBoot supports the serial port and the builtin ethernet connection 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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation

The original boot loader supplied with the CerfCube can be used to install RedBoot. Connect to the device using a serial port at 38400/8N1. Copy the binary RedBoot ROM mode image to an available TFTP server. Issue these commands to the Instrinsyc loader:

```
download tftp:x.x.x.x redboot_ROM.bin 0xc0000000
flashloader 0x00000000 0xc0000000 0x20000
```

where *x.x.x.x* is the IP address of the TFTP server.

NOTE: Other installation methods may be available via the Intrinsyc loader. Contact Intrinsyc for details.

Additional commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the CerfCube 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

Memory Maps

RedBoot sets up the following memory map on the CerfCube: 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) [organized as below]
  0x000000 - 0x0001ffff      RedBoot
  0x020000 - 0x0003ffff      RedBoot [RAM version]
  0xfc0000 - 0x00fdfffff     Fconfig data
  0xfe0000 - 0x00ffffff      FIS directory
0x0f000000 - 0x0fffffff    Onboard ethernet
0x10000000 - 0x17ffffff    CerfCube internal registers
0x20000000 - 0x3fffffff    PCMCIA / Compact Flash
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
0x08000000 - 0x0fffffff  N N  Onboard ethernet controller
0x10000000 - 0x17ffffff  N N  CerfCube internal registers
0x20000000 - 0x3fffffff  N N  PCMCIA / Compact Flash
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
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=cerf
export ARCH_DIR=arm
export PLATFORM_DIR=sa11x0/cerf
```

The names of configuration files are listed above with the description of the associated modes.

ARM/XScale Cyclone IQ80310

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm
ROMA	[ROM]	RedBoot running from flash address 0x40000, with ARM bootloader in flash boot sector.	redboot_ROMA.ecm
RAMA	[RAM]	RedBoot running from RAM with ARM bootloader in flash boot sector.	redboot_RAMA.ecm

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 images to flash. The ROM mode RedBoot (which runs from the flash boot sector) should be programmed to flash address 0x00000000. The ROMA RedBoot mode (which is started by the ARM bootloader) should be programmed to flash address 0x00004000.

To install RedBoot to run from the flash boot sector, use the manufacturer's flash utility to install the ROM mode 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 ROMA mode 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 - continue (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: 192.168.1.153
   Default server IP address: 192.168.1.10
   GDB connection port: 1000
   Network debug at boot time: false
   Update RedBoot non-volatile configuration - continue (y/n)? y
   ... Unlock from 0x007c0000-0x007e0000: .
   ... Erase from 0x007c0000-0x007e0000: .
   ... Program from 0xa0013018-0xa0013418 at 0x007c0000: .
   ... Lock from 0x007c0000-0x007e0000: .
```

Note: When later updating RedBoot in situ, it is important to use a matching ROM and RAM mode pair of images. So use either RAM/ROM or RAMA/ROMA images. Do not mix them.

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

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.

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.

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 - 7 Segment LED Tests
6 - Backplane Detection Test
7 - Battery Status Test
8 - External Timer Test
9 - i82559 Ethernet Configuration
10 - i82559 Ethernet Test
11 - i960Rx/303 PCI Interrupt Test
12 - Internal Timer Test
13 - Secondary PCI Bus Test
14 - Primary PCI Bus Test
15 - Battery Backup SDRAM Memory Test
16 - GPIO Test
17 - Repeat-On-Fail Memory Test
18 - Coyonosa Cache Loop (No return)
19 - Show Software and Hardware Revision
```

```
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.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=iq80310
export ARCH_DIR=arm
export PLATFORM_DIR=iq80310
```

The names of configuration files are listed above with the description of the associated modes.

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.

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 - 0xbfffffff	Y	Y	SDRAM
0xc0000000 - 0xcfffffff	Y	Y	Cache Flush Region
0xd0000000 - 0xd0000fff	Y	N	first 4k page of flash
0xf0000000 - 0xffffffff	N	N	80200 Internal Registers

Platform Resource Usage

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

ARM/XScale Intel IQ80321

Overview

RedBoot supports the serial port 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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

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 the ROM mode RedBoot image to flash. RedBoot should be programmed to flash address 0x00000000 using the DOS utility.

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 - continue (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
    ... Unlock from 0xf07e0000-0xf0800000: .
    ... Erase from 0xf07e0000-0xf0800000: .
    ... Program from 0x01ddf000-0x01ddf400 at 0xf07e0000: .
    ... Lock from 0xf07e0000-0xf0800000: .
```

Switch Settings

The 80321 board is highly configurable through a number of switches and jumpers. RedBoot makes some assumptions about board configuration and attention must be paid to these assumptions for reliable RedBoot

operation:

- The onboard ethernet and the secondary slot may be placed in a private space so that they are not seen by a PC BIOS. If the board is to be used in a PC with BIOS, then the ethernet should be placed in this private space so that RedBoot and the BIOS do not conflict.
- RedBoot assumes that the board is plugged into a PC with BIOS. This requires RedBoot to detect when the BIOS has configured the PCI-X secondary bus. If the board is placed in a backplane, RedBoot will never see the BIOS configure the secondary bus. To prevent this wait, set switch S7E1-3 to ON when using the board in a backplane.
- For the remaining switch settings, the following is a known good configuration:

S1D1	All OFF
S7E1	7 is ON, all others OFF
S8E1	2,3,5,6 are ON, all others OFF
S8E2	2,3 are ON, all others OFF
S9E1	3 is ON, all others OFF
S4D1	1,3 are ON, all others OFF
J9E1	2,3 jumpered
J9F1	2,3 jumpered
J3F1	Nothing jumpered
J3G1	2,3 jumpered
J1G2	2,3 jumpered

LED Codes

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

```
LED     Actions
-----------------------------------------------------------
  Power-On/Reset
88
      Set the CPSR
      Enable coprocessor access
      Drain write and fill buffer
      Setup PBIU chip selects
A1
      Enable the Icache
A2
      Move FLASH chip select from 0x0 to 0xF0000000
      Jump to new FLASH location
A3
      Setup and enable the MMU
A4
      I2C interface initialization
90
```

Wait for I2C initialization to complete

91

Send address (via I2C) to the DIMM

92

Wait for transmit complete

93

Read SDRAM PD data from DIMM

94

Read remainder of EEPROM data.
An error will result in one of the following
error codes on the LEDs:
77 BAD EEPROM checksum
55 I2C protocol error
FF bank size error

A5

Setup DDR memory interface

A6

Enable branch target buffer
Drain the write & fill buffers
Flush Icache, Dcache and BTB
Flush instuction and data TLBs
Drain the write & fill buffers

SL

ECC Scrub Loop

SE

A7

Clean, drain, flush the main Dcache

A8

Clean, drain, flush the mini Dcache
Flush Dcache
Drain the write & fill buffers

A9

Enable ECC

AA

Save SDRAM size
Move MMU tables into RAM

AB

Clean, drain, flush the main Dcache
Clean, drain, flush the mini Dcache
Drain the write & fill buffers

AC

Set the TTB register to DRAM mmu_table

AD

Set mode to IRQ mode

A7

Move SWI & Undefined "vectors" to RAM (at 0x0)

A6

Switch to supervisor mode

A5

Move remaining "vectors" to RAM (at 0x0)

A4

Copy DATA to RAM

Initialize interrupt exception environment
Initialize stack
Clear BSS section

A3
Call platform specific hardware initialization

A2
Run through static constructors

A1
Start up the eCos kernel or RedBoot

Special RedBoot Commands

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

```
RedBoot> diag
Entering Hardware Diagnostics - Disabling Data Cache!

  IQ80321 Hardware Tests

 1 - Memory Tests
 2 - Repeating Memory Tests
 3 - Repeat-On-Fail Memory Tests
 4 - Rotary Switch S1 Test
 5 - 7 Segment LED Tests
 6 - i82544 Ethernet Configuration
 7 - Baterry Status Test
 8 - Battery Backup SDRAM Memory Test
 9 - Timer Test
10 - PCI Bus test
11 - CPU Cache Loop (No Return)
 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.

Memory Tests

This test is used to test installed DDR SDRAM memory. Five different tests are run over the given address ranges. If errors are encountered, the test is aborted and information about the failure is printed. When selected, the user will be prompted to enter the base address of the test range and its size. The numbers must be in hex with no leading “0x”

```
Enter the menu item number (0 to quit): 1

Base address of memory to test (in hex): 100000

Size of memory to test (in hex): 200000

Testing memory from 0x00100000 to 0x002fffff.

Walking 1's test:
00000001000000020000000400000008000000100000002000000040000000 80
00000100000002000000040000000800000010000000200000004000000080 00
00010000000200000004000000080000001000000020000000400000008000 00
```

```
01000000020000000400000008000000100000002000000040000000800000000
passed
32-bit address test: passed
32-bit address bar test: passed
8-bit address test: passed
Byte address bar test: passed
Memory test done.
```

Repeating Memory Tests

The repeating memory tests are exactly the same as the above memory tests, except that the tests are automatically rerun after completion. The only way out of this test is to reset the board.

Repeat-On-Fail Memory Tests

This is similar to the repeating memory tests except that when an error is found, the failing test continuously retries on the failing address.

Rotary Switch S1 Test

This tests the operation of the sixteen position rotary switch. When run, this test will display the current position of the rotary switch on the LED display. Slowly dial through each position and confirm reading on LED.

7 Segment LED Tests

This tests the operation of the seven segment displays. When run, each LED cycles through 0 through F and a decimal point.

i82544 Ethernet Configuration

This test initializes the ethernet controller's serial EEPROM if the current contents are invalid. In any case, this test will also allow the user to enter a six byte ethernet MAC address into the serial EEPROM.

```
Enter the menu item number (0 to quit): 6


Current MAC address: 00:80:4d:46:00:02
Enter desired MAC address: 00:80:4d:46:00:01
Writing to the Serial EEPROM... Done

******** Reset The Board To Have Changes Take Effect ********
```

Battery Status Test

This tests the current status of the battery. First, the test checks to see if the battery is installed and reports that finding. If the battery is installed, the test further determines whether the battery status is one or more of the following:

- Battery is charging.

- Battery is fully discharged.
- Battery voltage measures within normal operating range.

Battery Backup SDRAM Memory Test

This tests the battery backup of SDRAM memory. This test is a three step process:

1. Select Battery backup test from main diag menu, then write data to SDRAM.
2. Turn off power for 60 seconds, then repower the board.
3. Select Battery backup test from main diag menu, then check data that was written in step 1.

Timer Test

This tests the internal timer by printing a number of dots at one second intervals.

PCI Bus Test

This tests the secondary PCI-X bus and socket. This test requires that an IQ80310 board be plugged into the secondary slot of the IOP80321 board. The test assumes at least 32MB of installed memory on the IQ80310. That memory is mapped into the IOP80321 address space and the memory tests are run on that memory.

CPU Cache Loop

This test puts the CPU into a tight loop run entirely from the ICache. This should prevent all external bus accesses.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=iq80321
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/iq80321
```

The names of configuration files are listed above with the description of the associated modes.

Interrupts

RedBoot uses an interrupt vector table which is located at address 0x8004. Entries in this table are pointers to functions with this protoype::

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

On an IQ80321 board, the vector argument is one of 32 interrupts defined in `hal/arm/xscale/verde/current/include/hal_var_ints.h`::

```
// *** 80200 CPU ***
#define CYGNUM_HAL_INTERRUPT_DMA0_EOT      0
#define CYGNUM_HAL_INTERRUPT_DMA0_EOC      1
#define CYGNUM_HAL_INTERRUPT_DMA1_EOT      2
#define CYGNUM_HAL_INTERRUPT_DMA1_EOC      3
#define CYGNUM_HAL_INTERRUPT_RSVD_4        4
#define CYGNUM_HAL_INTERRUPT_RSVD_5        5
#define CYGNUM_HAL_INTERRUPT_AA_EOT        6
#define CYGNUM_HAL_INTERRUPT_AA_EOC        7
#define CYGNUM_HAL_INTERRUPT_CORE_PMON     8
#define CYGNUM_HAL_INTERRUPT_TIMER0        9
#define CYGNUM_HAL_INTERRUPT_TIMER1        10
#define CYGNUM_HAL_INTERRUPT_I2C_0         11
#define CYGNUM_HAL_INTERRUPT_I2C_1         12
#define CYGNUM_HAL_INTERRUPT_MESSAGING     13
#define CYGNUM_HAL_INTERRUPT_ATU_BIST      14
#define CYGNUM_HAL_INTERRUPT_PERFMON       15
#define CYGNUM_HAL_INTERRUPT_CORE_PMU      16
#define CYGNUM_HAL_INTERRUPT_BIU_ERR       17
#define CYGNUM_HAL_INTERRUPT_ATU_ERR       18
#define CYGNUM_HAL_INTERRUPT_MCU_ERR       19
#define CYGNUM_HAL_INTERRUPT_DMA0_ERR      20
#define CYGNUM_HAL_INTERRUPT_DMA1_ERR      22
#define CYGNUM_HAL_INTERRUPT_AA_ERR        23
#define CYGNUM_HAL_INTERRUPT_MSG_ERR       24
#define CYGNUM_HAL_INTERRUPT_SSP           25
#define CYGNUM_HAL_INTERRUPT_RSVD_26       26
#define CYGNUM_HAL_INTERRUPT_XINT0         27
#define CYGNUM_HAL_INTERRUPT_XINT1         28
#define CYGNUM_HAL_INTERRUPT_XINT2         29
#define CYGNUM_HAL_INTERRUPT_XINT3         30
#define CYGNUM_HAL_INTERRUPT_HPI           31
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table. With 32 interrupts, the data table starts at address 0x8084.

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.

Memory Maps

The RAM based page table is located at RAM start + 0x4000. RedBoot may be configured for one of two memory maps. The difference between them is the location of RAM and the PCI outbound windows. The alternative memory map may be used when building RedBoot or eCos by using the `RAM_ALTMAP` and `ROM_ALTMAP` startup types in the configuration.

> **NOTE:** The virtual memory maps in this section use a C, B, and X column to indicate the caching policy for the region..

```
X C B  Description
```

```
- - -  ---------------------------------------------
0 0 0  Uncached/Unbuffered
0 0 1  Uncached/Buffered
0 1 0  Cached/Buffered    Write Through, Read Allocate
0 1 1  Cached/Buffered    Write Back, Read Allocate
1 0 0  Invalid -- not used
1 0 1  Uncached/Buffered  No write buffer coalescing
1 1 0  Mini DCache - Policy set by Aux Ctl Register
1 1 1  Cached/Buffered    Write Back, Read/Write Allocate

Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x7fffffff    ATU Outbound Direct Window
0x80000000 - 0x900fffff    ATU Outbound Translate Windows
0xa0000000 - 0xbfffffff    SDRAM
0xf0000000 - 0xf0800000    FLASH               (PBIU CS0)
0xfe800000 - 0xfe800fff    UART                (PBIU CS1)
0xfe840000 - 0xfe840fff    Left 7-segment LED  (PBIU CS3)
0xfe850000 - 0xfe850fff    Right 7-segment LED (PBIU CS2)
0xfe8d0000 - 0xfe8d0fff    Rotary Switch       (PBIU CS4)
0xfe8f0000 - 0xfe8f0fff    Baterry Status      (PBIU CS5)
0xfff00000 - 0xffffffff    Verde Memory mapped Registers


Default Virtual Map      X C B  Description
-----------------------  - - -  ----------------------------------
0x00000000 - 0x1fffffff  1 1 1  SDRAM
0x20000000 - 0x9fffffff  0 0 0  ATU Outbound Direct Window
0xa0000000 - 0xb00fffff  0 0 0  ATU Outbound Translate Windows
0xc0000000 - 0xdfffffff  0 0 0  Uncached alias for SDRAM
0xe0000000 - 0xe00fffff  1 1 1  Cache flush region (no phys mem)
0xf0000000 - 0xf0800000  0 1 0  FLASH               (PBIU CS0)
0xfe800000 - 0xfe800fff  0 0 0  UART                (PBIU CS1)
0xfe840000 - 0xfe840fff  0 0 0  Left 7-segment LED  (PBIU CS3)
0xfe850000 - 0xfe850fff  0 0 0  Right 7-segment LED (PBIU CS2)
0xfe8d0000 - 0xfe8d0fff  0 0 0  Rotary Switch       (PBIU CS4)
0xfe8f0000 - 0xfe8f0fff  0 0 0  Baterry Status      (PBIU CS5)
0xfff00000 - 0xffffffff  0 0 0  Verde Memory mapped Registers

Alternate Virtual Map    X C B  Description
-----------------------  - - -  ----------------------------------
0x00000000 - 0x000fffff  1 1 1  Alias for 1st MB of SDRAM
0x00100000 - 0x7fffffff  0 0 0  ATU Outbound Direct Window
0x80000000 - 0x900fffff  0 0 0  ATU Outbound Translate Windows
0xa0000000 - 0xbfffffff  1 1 1  SDRAM
0xc0000000 - 0xdfffffff  0 0 0  Uncached alias for SDRAM
0xe0000000 - 0xe00fffff  1 1 1  Cache flush region (no phys mem)
0xf0000000 - 0xf0800000  0 1 0  FLASH               (PBIU CS0)
0xfe800000 - 0xfe800fff  0 0 0  UART                (PBIU CS1)
0xfe840000 - 0xfe840fff  0 0 0  Left 7-segment LED  (PBIU CS3)
0xfe850000 - 0xfe850fff  0 0 0  Right 7-segment LED (PBIU CS2)
0xfe8d0000 - 0xfe8d0fff  0 0 0  Rotary Switch       (PBIU CS4)
0xfe8f0000 - 0xfe8f0fff  0 0 0  Baterry Status      (PBIU CS5)
0xfff00000 - 0xffffffff  0 0 0  Verde Memory mapped Registers
```

Platform Resource Usage

The Verde programmable timer0 is used for timeout support for networking and XModem file transfers.

ARM/Intel XScale IXDP425 Network Processor Evaluation Board

Overview

RedBoot supports the builtin high-speed and console UARTs and a PCI based i82559 ethernet card for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MB boot flash on the mainboard.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from flash sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

The IXDP425 flash is socketed, so initial installation may be done using an appropriate device programmer. JTAG based initial may also be used. In either case, the ROM mode RedBoot is programmed into the boot flash at address 0x00000000.

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 should be configured for use by RedBoot. See the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2 for more details.

LED Codes

RedBoot uses the 4 digit LED display to indicate status during board initialization. Possible codes are:

LED Actions

Power-On/Reset
- Set the CPSR
- Enable coprocessor access
- Drain write and fill buffer
- Setup expansion bus chip selects

1001
: Enable Icache

1002
: Initialize SDRAM controller

1003
: Switch flash (CS0) from 0x00000000 to 0x50000000

1004
: Copy MMU table to RAM

1005
: Setup TTB and domain permissions

1006
: Enable MMU

1007
: Enable DCache

1008
: Enable branch target buffer

1009
: Drain write and fill buffer
: Flush caches

100A
: Start up the eCos kernel or RedBoot

0001

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=ixdp425
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/ixdp425
```

The names of configuration files are listed above with the description of the associated modes.

Interrupts

RedBoot uses an interrupt vector table which is located at address 0x8004. Entries in this table are pointers to functions with this protoype::

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

On the IXDP425 board, the vector argument is one of many interrupts defined in `hal/arm/xscale/ixp425/current/include/hal_var_ints.h`::

```
#define CYGNUM_HAL_INTERRUPT_NPEA         0
#define CYGNUM_HAL_INTERRUPT_NPEB         1
#define CYGNUM_HAL_INTERRUPT_NPEC         2
#define CYGNUM_HAL_INTERRUPT_QM1          3
#define CYGNUM_HAL_INTERRUPT_QM2          4
#define CYGNUM_HAL_INTERRUPT_TIMER0       5
#define CYGNUM_HAL_INTERRUPT_GPIO0        6
```

```
#define CYGNUM_HAL_INTERRUPT_GPIO1        7
#define CYGNUM_HAL_INTERRUPT_PCI_INT      8
#define CYGNUM_HAL_INTERRUPT_PCI_DMA1     9
#define CYGNUM_HAL_INTERRUPT_PCI_DMA2     10
#define CYGNUM_HAL_INTERRUPT_TIMER1       11
#define CYGNUM_HAL_INTERRUPT_USB          12
#define CYGNUM_HAL_INTERRUPT_UART2        13
#define CYGNUM_HAL_INTERRUPT_TIMESTAMP    14
#define CYGNUM_HAL_INTERRUPT_UART1        15
#define CYGNUM_HAL_INTERRUPT_WDOG         16
#define CYGNUM_HAL_INTERRUPT_AHB_PMU      17
#define CYGNUM_HAL_INTERRUPT_XSCALE_PMU   18
#define CYGNUM_HAL_INTERRUPT_GPIO2        19
#define CYGNUM_HAL_INTERRUPT_GPIO3        20
#define CYGNUM_HAL_INTERRUPT_GPIO4        21
#define CYGNUM_HAL_INTERRUPT_GPIO5        22
#define CYGNUM_HAL_INTERRUPT_GPIO6        23
#define CYGNUM_HAL_INTERRUPT_GPIO7        24
#define CYGNUM_HAL_INTERRUPT_GPIO8        25
#define CYGNUM_HAL_INTERRUPT_GPIO9        26
#define CYGNUM_HAL_INTERRUPT_GPIO10       27
#define CYGNUM_HAL_INTERRUPT_GPIO11       28
#define CYGNUM_HAL_INTERRUPT_GPIO12       29
#define CYGNUM_HAL_INTERRUPT_SW_INT1      30
#define CYGNUM_HAL_INTERRUPT_SW_INT2      31
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table. With 32 interrupts, the data table starts at address 0x8084.

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.

Memory Maps

The RAM based page table is located at RAM start + 0x4000.

> **NOTE:** The virtual memory maps in this section use a C, B, and X column to indicate the caching policy for the region..

```
X C B  Description
- - -  ---------------------------------------------
0 0 0  Uncached/Unbuffered
0 0 1  Uncached/Buffered
0 1 0  Cached/Buffered    Write Through, Read Allocate
0 1 1  Cached/Buffered    Write Back, Read Allocate
1 0 0  Invalid -- not used
1 0 1  Uncached/Buffered  No write buffer coalescing
1 1 0  Mini DCache - Policy set by Aux Ctl Register
1 1 1  Cached/Buffered    Write Back, Read/Write Allocate

Virtual Address   Physical Address  XCB  Size (MB)  Description
---------------   ----------------  ---  ---------  -----------
   0x00000000       0x00000000      010     256     SDRAM (cached)
```

```
0x10000000      0x10000000      010     256     SDRAM (alias)
0x20000000      0x00000000      000     256     SDRAM (uncached)
0x48000000      0x48000000      000      64     PCI Data
0x50000000      0x50000000      010      16     Flash (CS0)
0x51000000      0x51000000      000     112     CS1 - CS7
0x60000000      0x60000000      000      64     Queue Manager
0xC0000000      0xC0000000      000       1     PCI Controller
0xC4000000      0xC4000000      000       1     Exp. Bus Config
0xC8000000      0xC8000000      000       1     Misc IXP425 IO
0xCC000000      0xCC000000      000       1     SDRAM Config
```

Platform Resource Usage

The IXP425 programmable OStimer0 is used for timeout support for networking and XModem file transfers.

ARM/Intel XScale Generic Residential Gateway

Overview

RedBoot supports the console UART and a PCI based i82559 ethernet card for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MB onboard flash.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from flash sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

The GRG flash is socketed, so initial installation may be done using an appropriate device programmer. JTAG based initial may also be used. In either case, the ROM mode RedBoot is programmed into the boot flash at address 0x00000000.

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 should be configured for use by RedBoot. See the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2 for more details.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=grg
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/grg
```

The names of configuration files are listed above with the description of the associated modes.

Interrupts

RedBoot uses an interrupt vector table which is located at address 0x8004. Entries in this table are pointers to functions with this protoype::

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

On the GRG board, the vector argument is one of many interrupts defined in `hal/arm/xscale/ixp425/current/include/hal_var_ints.h`::

```
#define CYGNUM_HAL_INTERRUPT_NPEA         0
#define CYGNUM_HAL_INTERRUPT_NPEB         1
#define CYGNUM_HAL_INTERRUPT_NPEC         2
#define CYGNUM_HAL_INTERRUPT_QM1          3
#define CYGNUM_HAL_INTERRUPT_QM2          4
#define CYGNUM_HAL_INTERRUPT_TIMER0       5
#define CYGNUM_HAL_INTERRUPT_GPIO0        6
#define CYGNUM_HAL_INTERRUPT_GPIO1        7
#define CYGNUM_HAL_INTERRUPT_PCI_INT      8
#define CYGNUM_HAL_INTERRUPT_PCI_DMA1     9
#define CYGNUM_HAL_INTERRUPT_PCI_DMA2     10
#define CYGNUM_HAL_INTERRUPT_TIMER1       11
#define CYGNUM_HAL_INTERRUPT_USB          12
#define CYGNUM_HAL_INTERRUPT_UART2        13
#define CYGNUM_HAL_INTERRUPT_TIMESTAMP    14
#define CYGNUM_HAL_INTERRUPT_UART1        15
#define CYGNUM_HAL_INTERRUPT_WDOG         16
#define CYGNUM_HAL_INTERRUPT_AHB_PMU      17
#define CYGNUM_HAL_INTERRUPT_XSCALE_PMU   18
#define CYGNUM_HAL_INTERRUPT_GPIO2        19
#define CYGNUM_HAL_INTERRUPT_GPIO3        20
#define CYGNUM_HAL_INTERRUPT_GPIO4        21
#define CYGNUM_HAL_INTERRUPT_GPIO5        22
#define CYGNUM_HAL_INTERRUPT_GPIO6        23
#define CYGNUM_HAL_INTERRUPT_GPIO7        24
#define CYGNUM_HAL_INTERRUPT_GPIO8        25
#define CYGNUM_HAL_INTERRUPT_GPIO9        26
#define CYGNUM_HAL_INTERRUPT_GPIO10       27
#define CYGNUM_HAL_INTERRUPT_GPIO11       28
#define CYGNUM_HAL_INTERRUPT_GPIO12       29
#define CYGNUM_HAL_INTERRUPT_SW_INT1      30
#define CYGNUM_HAL_INTERRUPT_SW_INT2      31
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table. With 32 interrupts, the data table starts at address 0x8084.

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.

Memory Maps

The RAM based page table is located at RAM start + 0x4000.

> **NOTE:** The virtual memory maps in this section use a C, B, and X column to indicate the caching policy for the region..

```
X C B  Description
- - -  ---------------------------------------------
0 0 0  Uncached/Unbuffered
0 0 1  Uncached/Buffered
0 1 0  Cached/Buffered    Write Through, Read Allocate
0 1 1  Cached/Buffered    Write Back, Read Allocate
1 0 0  Invalid -- not used
1 0 1  Uncached/Buffered  No write buffer coalescing
1 1 0  Mini DCache - Policy set by Aux Ctl Register
1 1 1  Cached/Buffered    Write Back, Read/Write Allocate
```

Virtual Address	Physical Address	XCB	Size (MB)	Description
0x00000000	0x00000000	010	32	SDRAM (cached)
0x10000000	0x00000000	010	32	SDRAM (alias)
0x20000000	0x00000000	000	32	SDRAM (uncached)
0x48000000	0x48000000	000	64	PCI Data
0x50000000	0x50000000	010	16	Flash (CS0)
0x51000000	0x51000000	000	112	CS1 - CS7
0x60000000	0x60000000	000	64	Queue Manager
0xC0000000	0xC0000000	000	1	PCI Controller
0xC4000000	0xC4000000	000	1	Exp. Bus Config
0xC8000000	0xC8000000	000	1	Misc IXP425 IO
0xCC000000	0xCC000000	000	1	SDRAM Config

Platform Resource Usage

The IXP425 programmable OStimer0 is used for timeout support for networking and XModem file transfers.

Motorola PrPMC1100 CPU card

Overview

RedBoot supports the builtin high-speed and console UARTs . The console UART is the default and feeds the front panel COM1 connector. The high-speed UART signals are only available from the PN4 IO connector. Therefore, usability of this port depends on the carrier board used. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MB boot flash on the mainboard.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from flash sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

The PrPMC1100 flash is socketed, so initial installation may be done using an appropriate device programmer. JTAG based flash programming may also be used. In either case, the ROM mode RedBoot is programmed into the boot flash at address 0x00000000.

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 should be configured for use by RedBoot. Even if this message is not seen, it is recommended that the **fconfig** be run to initialize the flash configuration area. See the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2 for more details.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=prpmc1100
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/prpmc1100
```

The names of configuration files are listed above with the description of the associated modes.

Interrupts

RedBoot uses an interrupt vector table which is located at address 0x8004. Entries in this table are pointers to functions with this protoype::

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

On the PrPMC1100 board, the vector argument is one of many interrupts defined in `hal/arm/xscale/ixp425/current/include/hal_var_ints.h`::

```
#define CYGNUM_HAL_INTERRUPT_NPEA         0
#define CYGNUM_HAL_INTERRUPT_NPEB         1
#define CYGNUM_HAL_INTERRUPT_NPEC         2
#define CYGNUM_HAL_INTERRUPT_QM1          3
#define CYGNUM_HAL_INTERRUPT_QM2          4
#define CYGNUM_HAL_INTERRUPT_TIMER0       5
#define CYGNUM_HAL_INTERRUPT_GPIO0        6
#define CYGNUM_HAL_INTERRUPT_GPIO1        7
#define CYGNUM_HAL_INTERRUPT_PCI_INT      8
#define CYGNUM_HAL_INTERRUPT_PCI_DMA1     9
#define CYGNUM_HAL_INTERRUPT_PCI_DMA2     10
#define CYGNUM_HAL_INTERRUPT_TIMER1       11
#define CYGNUM_HAL_INTERRUPT_USB          12
#define CYGNUM_HAL_INTERRUPT_UART2        13
#define CYGNUM_HAL_INTERRUPT_TIMESTAMP    14
#define CYGNUM_HAL_INTERRUPT_UART1        15
#define CYGNUM_HAL_INTERRUPT_WDOG         16
#define CYGNUM_HAL_INTERRUPT_AHB_PMU      17
#define CYGNUM_HAL_INTERRUPT_XSCALE_PMU   18
#define CYGNUM_HAL_INTERRUPT_GPIO2        19
#define CYGNUM_HAL_INTERRUPT_GPIO3        20
#define CYGNUM_HAL_INTERRUPT_GPIO4        21
#define CYGNUM_HAL_INTERRUPT_GPIO5        22
#define CYGNUM_HAL_INTERRUPT_GPIO6        23
#define CYGNUM_HAL_INTERRUPT_GPIO7        24
#define CYGNUM_HAL_INTERRUPT_GPIO8        25
#define CYGNUM_HAL_INTERRUPT_GPIO9        26
#define CYGNUM_HAL_INTERRUPT_GPIO10       27
#define CYGNUM_HAL_INTERRUPT_GPIO11       28
#define CYGNUM_HAL_INTERRUPT_GPIO12       29
#define CYGNUM_HAL_INTERRUPT_SW_INT1      30
#define CYGNUM_HAL_INTERRUPT_SW_INT2      31
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table. With 32 interrupts, the data table starts at address 0x8084.

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.

Memory Maps

The RAM based page table is located at RAM start + 0x4000.

> **NOTE:** The virtual memory maps in this section use a C, B, and X column to indicate the caching policy for the region..

```
X C B  Description
- - -  ---------------------------------------------
0 0 0  Uncached/Unbuffered
0 0 1  Uncached/Buffered
0 1 0  Cached/Buffered    Write Through, Read Allocate
0 1 1  Cached/Buffered    Write Back, Read Allocate
1 0 0  Invalid -- not used
1 0 1  Uncached/Buffered  No write buffer coalescing
1 1 0  Mini DCache - Policy set by Aux Ctl Register
1 1 1  Cached/Buffered    Write Back, Read/Write Allocate

Virtual Address   Physical Address  XCB  Size (MB)  Description
---------------   ----------------  ---  ---------  -----------
   0x00000000       0x00000000      010     256     SDRAM (cached)
   0x10000000       0x10000000      010     256     SDRAM (alias)
   0x20000000       0x00000000      000     256     SDRAM (uncached)
   0x48000000       0x48000000      000      64     PCI Data
   0x50000000       0x50000000      010      16     Flash (CS0)
   0x51000000       0x51000000      000     112     CS1 - CS7
   0x60000000       0x60000000      000      64     Queue Manager
   0xC0000000       0xC0000000      000       1     PCI Controller
   0xC4000000       0xC4000000      000       1     Exp. Bus Config
   0xC8000000       0xC8000000      000       1     Misc CPU IO
   0xCC000000       0xCC000000      000       1     SDRAM Config
```

Platform Resource Usage

The CPU programmable OStimer0 is used for timeout support for networking and XModem file transfers.

CalmRISC/CalmRISC16 Samsung CalmRISC16 Core Evaluation Board

Overview

The Samsung CalmRISC16 evaluation platform consists of two boards connected by a ribbon cable. One board contains the CPU core and memory. The other board is called the MDSChip board and provides the host interface. The calmRISC16 is a harvard architecture with separate 22-bit program and data addresses. The instruction set provides no instruction for writing to program memory. The MDSChip board firmware (called CalmBreaker) provides a pseudo register interface so that code running on the core has access to a serial channel and a mechanism to write to program memory. The serial channel is fixed at 57600-8-N-1 by the firmware. The CalmBreaker firmware also provides a serial protocol which allows a host to download a program and to start or stop the core board.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running via the MDSChip board.	redboot_ROM.ecm

Initial Installation Method

The CalmRISC16 core is controlled through the MDSChip board. There is no non-volatile storage available for RedBoot, so RedBoot must be downloaded to the board on every power cycle. A small utility program is used to download S-record files to the eval board. Sources and build instructions for this utility are located in the RedBoot sources in: `packages/hal/calmrisc16/ceb/current/support`

To download the RedBoot image, first press the reset button on the MDSChip board. The green 'Run' LED on the core board should go off. Now, use the utility to download the RedBoot image with:

```
$ calmbreaker -p /dev/term/b --reset --srec-code -f redboot.elf
```

Note that the '-p /dev/term/b' specifies the serial port to use and will vary from system to system. The download will take about two minutes. After it finishes, start RedBoot with:

```
$ calmbreaker -p /dev/term/b --run
```

The 'Run' LED on the core board should be on. Connecting to the MDSboard with a terminal and typing enter should result in RedBoot reprinting the command prompt.

Special RedBoot Commands

None.

Special Note on Serial Channel

The MDSChip board uses a relatively slow microcontroller to provide the pseudo-register interface to the core board. This pseudo-register interface provides access to the serial channel and write access to program memory. Those interfaces are slow and the serial channel is easily overrun by a fast host. For this reason, GDB must be told to limit the size of code download packets to avoid serial overrun. This is done with the following GDB command:

```
(gdb) set download-write-size 25
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=calm16_ceb
export ARCH_DIR=calmrisc16
export PLATFORM_DIR=ceb
```

The names of configuration files are listed above with the description of the associated modes.

CalmRISC/CalmRISC32 Samsung CalmRISC32 Core Evaluation Board

Overview

The Samsung CalmRISC32 evaluation platform consists of two boards connected by a ribbon cable. One board contains the CPU core and memory. The other board is called the MDSChip board and provides the host interface. The calmRISC32 is a harvard architecture with separate 32-bit program and data addresses. The instruction set provides no instruction for writing to program memory. The MDSChip board firmware (called CalmBreaker) provides a pseudo register interface so that code running on the core has access to a serial channel and a mechanism to write to program memory. The serial channel is fixed at 57600-8-N-1 by the firmware. The CalmBreaker firmware also provides a serial protocol which allows a host to download a program and to start or stop the core board.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running via the MDSChip board.	redboot_ROM.ecm

Initial Installation Method

The calmRISC32 core is controlled through the MDSChip board. There is no non-volatile storage available for RedBoot, so RedBoot must be downloaded to the board on every power cycle. A small utility program is used to download S-record files to the eval board. Sources and build instructions for this utility are located in the RedBoot sources in: `packages/hal/calmrisc32/ceb/current/support`

To download the RedBoot image, first press the reset button on the MDSChip board. The green 'Run' LED on the core board should go off. Now, use the utility to download the RedBoot image with:

```
$ calmbreaker -p /dev/term/b --reset --srec-code -f redboot.elf
```

Note that the '-p /dev/term/b' specifies the serial port to use and will vary from system to syetm. The download will take about two minutes. After it finishes, start RedBoot with:

```
$ calmbreaker -p /dev/term/b --run
```

The 'Run' LED on the core board should be on. Connecting to the MDSboard with a terminal and typing enter should result in RedBoot reprinting the command prompt.

Special RedBoot Commands

None.

Special Note on Serial Channel

The MDSChip board uses a relatively slow microcontroller to provide the pseudo-register interface to the core board. This pseudo-register interface provides access to the serial channel and write access to program memory. Those interfaces are slow and the serial channel is easily overrun by a fast host. For this reason, GDB must be told to limit the size of code download packets to avoid serial overrun. This is done with the following GDB command:

```
(gdb) set download-write-size 25
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=calm32_ceb
export ARCH_DIR=calmrisc32
export PLATFORM_DIR=ceb
```

The names of configuration files are listed above with the description of the associated modes.

FRV/FRV400 Fujitsu FR-V 400 (MB-93091)

Overview

RedBoot supports both serial ports, which are available via the stacked serial connectors on the mother board. The topmost port is the default and is considered to be port 0 by RedBoot. The bottommost port is serial port 1. The default serial port settings are 38400,8,N,1.

FLASH management is also supported, but only for the FLASH device in IC7. This arrangement allows for IC8 to retain either the original Fujitsu board firmware, or some application specific contents.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm

Configuration	Mode	Description	File
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

RedBoot can be installed by directly programming the FLASH device in IC7 or by using the Fujitsu provided software to download and install a version into the FLASH device. Complete instructions are provided separately.

Special RedBoot Commands

None.

Memory Maps

The memory map of this platform is fixed by the hardware (cannot be changed by software). The only attributes which can be modified are control over cacheability, as noted below.

```
Address                 Cache?      Resource
00000000-03EFFFFF         Yes       SDRAM (via plugin DIMM)
03F00000-03FFFFFF         No        SDRAM (used for PCI window)
10000000-1FFFFFFF         No        MB86943 PCI bridge
20000000-201FFFFF         No        SRAM
21000000-23FFFFFF         No        Motherboard resources
24000000-25FFFFFF         No        PCI I/O space
26000000-2FFFFFFF         No        PCI Memory space
30000000-FDFFFFFF         ??        Unused
FE000000-FEFFFFFF         No        I/O devices
FF000000-FF1FFFFF         No        IC7 - RedBoot FLASH
FF200000-FF3FFFFF         No        IC8 - unused FLASH
FF400000-FFFFFFFF         No        Misc other I/O
```

NOTE: The only configuration currently suppored requires a 64MB SDRAM DIMM to be present on the CPU card. No other memory configuration is supported at this time.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=frv400
export ARCH_DIR=frv
export PLATFORM_DIR=frv400
```

The names of configuration files are listed above with the description of the associated modes.

Fujitsu FR-V Design Kit (MB93091-CBxx)

Overview

RedBoot supports both serial ports, which are available via the stacked serial connectors on the mother board in the case of the FR400 CPU board, and via serial connectors present on the other supported CPU boards themselves. The topmost port is the default and is considered to be port 0 by RedBoot. The bottommost port is serial port 1. The default serial port settings are 115200,8,N,1. The serial port supports baud rates up to 460800, which can be set using the **baud** command as described in Chapter 2.

FLASH management is also supported, but only for the FLASH device in IC7. This arrangement allows for IC8 to retain either the original Fujitsu board firmware, or some application specific contents. Two basic RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Since the normal RedBoot configuration does not use the FLASH ROM except during startup, it is unnecessary to load a RAM-based RedBoot before reprogramming the FLASH.

Initial Installation Method

RedBoot can be installed by directly programming the FLASH device in IC7 or by using the Fujitsu provided software to download and install a version into the FLASH device. Complete instructions are provided separately.

Special RedBoot Commands

The **exec** command as described in Chapter 2 is supported by RedBoot on this target, for executing Linux kernels. Only the command line and timeout options are relevant to this platform.

Memory Maps

The memory map of this platform is fixed by the hardware (cannot be changed by software). The only attributes

which can be modified are control over cacheability, as noted below.

```
Address                 Cache?      Resource
00000000-03EFFFFF         Yes       SDRAM (via plugin DIMM)
03F00000-03FFFFFF         No        SDRAM (used for PCI window)
10000000-1FFFFFFF         No        MB86943 PCI bridge
20000000-201FFFFF         No        SRAM
21000000-23FFFFFF         No        Motherboard resources
24000000-25FFFFFF         No        PCI I/O space
26000000-2FFFFFFF         No        PCI Memory space
30000000-FDFFFFFF         ??        Unused
FE000000-FEFFFFFF         No        I/O devices
FF000000-FF1FFFFF         No        IC7 - RedBoot FLASH
FF200000-FF3FFFFF         No        IC8 - unused FLASH
FF400000-FFFFFFFF         No        Misc other I/O
```

NOTE: The only configuration currently suppored requires a 64MiB SDRAM DIMM to be present on the CPU card. No other memory configuration is supported at this time.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=mb93091
export ARCH_DIR=frv
export PLATFORM_DIR=mb93091
```

The names of configuration files are listed above with the description of the associated modes.

Resource Usage

The RedBoot image occupies flash addresses 0xFF000000 - 0xFF03FFFF. To execute it copies itself out of there to RAM at 0x03E00000. RedBoot reserves memory from 0x00000000 to 0x0001FFFF for its own use. User programs can use memory from 0x00020000 to 0x03DFFFFF. RAM based RedBoot configurations are designed to run from RAM at 0x00020000.

Fujitsu FR-V Portable Demonstration Kit (MB93093-PD00)

Overview

RedBoot supports the serial port which is available via a special cable connected to the CON_UART connector on the board. The default serial port settings are 115200,8,N,1. The serial port supports baud rates up to 460800, which can be set using the **baud** command as described in Chapter 2.

FLASH management is also supported. Two basic RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Since the normal RedBoot configuration does not use the FLASH ROM except during startup, it is unnecessary to load a RAM-based RedBoot before reprogramming the FLASH.

Initial Installation Method

The Portable Demonstration Kit should have been shipped with an existing version of RedBoot, which can be upgraded to the current version using the instructions below.

Special RedBoot Commands

The **exec** command as described in Chapter 2 is supported by RedBoot on this target, for executing Linux kernels. Only the command line and timeout options are relevant to this platform.

Memory Maps

The memory map of this platform is fixed by the hardware (cannot be changed by software). The only attributes which can be modified are control over cacheability, as noted below.

```
Address                 Cache?      Resource
00000000-03EFFFFF         Yes       SDRAM (via plugin DIMM)
03F00000-03FFFFFF         No        Unused (SDRAM)
10000000-1FFFFFFF         No        AX88796 Ethernet
20000000-2FFFFFFF         No        System FPGA
30000000-3FFFFFFF         No        MB93493 companion chip (unused)
40000000-FCFFFFFF         ??        Unused
FD000000-FDFFFFFF         ??        FLASH (ROM3,ROM4) (unused)
FE000000-FEFFFFFF         No        Miscellaneous on-chip I/O
FF000000-FFFFFFFF         No        RedBoot FLASH (16MiB)
```

NOTE: The only configuration currently suppored requires a 64MiB SDRAM DIMM to be present on the CPU card. No other memory configuration is supported at this time.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=mb93093
export ARCH_DIR=frv
export PLATFORM_DIR=mb93093
```

Resource Usage

The RedBoot image occupies flash addresses 0xFF000000 - 0xFF03FFFF. To execute it copies itself out of there to RAM at 0x03E00000. RedBoot reserves memory from 0x00000000 to 0x0001FFFF for its own use. User programs can use memory from 0x00020000 to 0x03DFFFFF. RAM based RedBoot configurations are designed to run from RAM at 0x00020000.

IA32/x86 x86-Based PC

Overview

RedBoot supports two serial ports and an Intel i82559 based ethernet card (for example an Intel EtherExpress Pro 10/100) for communication and downloads. The default serial port settings are 38400,8,N,1.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
Floppy	[Floppy]	RedBoot running from a boot floppy disk installed in the A: drive of the PC.	redboot_ROM.ecm

Initial Installation

RedBoot takes the form of a self-booting image that must be written onto a formatted floppy disk. The process will erase any file system or data that already exists on that disk, so proceed with caution.

For Red Hat Linux users, this can be done by:

```
$ dd conv=sync if=install/bin/redboot.bin of=/dev/fd0H1440
```

For NT Cygwin users, this can be done by first ensuring that the raw floppy device is mounted as `/dev/fd0`. To check if this is the case, type the command **mount** at the Cygwin bash prompt. If the floppy drive is already mounted, it will be listed as something similar to the following line:

```
\\.\a: /dev/fd0 user binmode
```

If this line is not listed, then mount the floppy drive using the command:

```
$ mount -f -b //./a: /dev/fd0
```

To actually install the boot image on the floppy, use the command:

```
$ dd conv=sync if=install/bin/redboot.bin of=/dev/fd0
```

Insert this floppy in the A: drive of the PC to be used as a target and ensure that the BIOS is configured to boot from A: by default. On reset, the PC will boot from the floppy and be ready to be debugged via either serial line, or via the ethernet interface if it is installed.

> **NOTE:** Unreliable floppy media may cause the write to silently fail. This can be determined if the RedBoot image does not correctly boot. In such cases, the floppy should be (unconditionally) reformatted using the **fdformat** command on Linux, or **format a: /u** on DOS/Windows.

Flash management

PC RedBoot does not support any FLASH commands.

Special RedBoot Commands

None.

Memory Maps

All selectors are initialized to map the entire 32-bit address space in the familiar protected mode flat model. Page translation is not used. RAM up to 640K is mapped to 0x0 to 0xa0000. RAM above 640K is mapped from address 0x100000 upwards. Space is reserved between 0xa0000 and 0x100000 for option ROMs and the BIOS.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=pc
export ARCH_DIR=i386
export PLATFORM_DIR=pc
```

The names of configuration files are listed above with the description of the associated modes.

MIPS/MIPS32(CoreLV 4Kc)+MIPS64(CoreLV 5Kc) Atlas Board

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

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 the Section called *Atlas download format*.

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 ROM mode 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 the Section called *Additional config options*, the Section called *Flash Image System (FIS)* in Chapter 2 and the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2 for details.

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 utility 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_ROM.srec redboot_ROM.rec
$ srecconv.pl -ES L -A 29 redboot_ROM
```

The resulting file is named `redboot_ROM.fl`.

Flash management

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.

Additional commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this architecture (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for MIPS boards are:

-b *<addr>*

Location to store command line and environment passed to kernel

-w *<time>*

Wait time in seconds before starting kernel

-c *"params"*

Parameters passed to kernel

<addr>

Kernel entry point, defaulting to the entry point of the last image loaded

Linux kernels on MIPS platforms expect the entry point to be called with arguments in the registers equivalent to a C call with prototype:

```
void Linux(int argc, char **argv, char **envp);
```

RedBoot will place the appropriate data at the offset specified by the *-b* parameter, or by default at address 0x80080000, and will set the arguments accordingly when calling into the kernel.

The default entry point, if no image with explicit entry point has been loaded and none is specified, is 0x80000750.

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.

Memory Maps

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

```
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
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=atlas_mips32_4kc
export TARGET=atlas_mips64_5kc
export ARCH_DIR=mips
export PLATFORM_DIR=atlas
```

Use one of the TARGET settings only.

The names of configuration files are listed above with the description of the associated modes.

MIPS/MIPS32(CoreLV 4Kc)+MIPS64(CoreLV 5Kc) Malta Board

Overview

RedBoot supports both front facing serial ports 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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation

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

Quick download instructions

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

1. Locate the prebuilt files in the bin directory: `deleteall.fl` and `redboot_ROM.fl`.
2. Make sure 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.fl` file to the parallel port to erase previous images:

   ```
   $ cat deleteall.fl >/dev/lp0
   ```

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

   ```
   $ cat redboot_ROM.fl >/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 the Section called *Flash Image System (FIS)* in Chapter 2 and the Section called *Persistent State Flash-based Configuration and Control* in Chapter 2 for details.

Malta download format

In order to download RedBoot to the Malta board, it must be converted to the Malta download format.

The *Atlas/Malta Developer's Kit* CD contains an srecconv.pl utility which requires Perl. This utility 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_ROM.srec redboot_ROM.rec
$ srecconv.pl -ES L -A 29 redboot_ROM
```

The resulting file is named `redboot_ROM.fl`.

Additional commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this architecture (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for MIPS boards are:

-b *<addr>*

Location to store command line and environment passed to kernel

-w *<time>*

Wait time in seconds before starting kernel

-c *"params"*

Parameters passed to kernel

<addr>

Kernel entry point, defaulting to the entry point of the last image loaded

Linux kernels on MIPS platforms expect the entry point to be called with arguments in the registers equivalent to a C call with prototype:

```
void Linux(int argc, char **argv, char **envp);
```

RedBoot will place the appropriate data at the offset specified by the *-b* parameter, or by default at address 0x80080000, and will set the arguments accordingly when calling into the kernel.

The default entry point, if no image with explicit entry point has been loaded and none is specified, is 0x80000750.

Interrupts

RedBoot uses an interrupt vector table which is located at address 0x80000200. Entries in this table are pointers to functions with this protoype:

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

On the malta board, the vector argument is one of 22 interrupts defined in `hal/mips/malta/`*VERSION*`/include/plf_intr.h`:

```
#define CYGNUM_HAL_INTERRUPT_SOUTH_BRIDGE_INTR   0
#define CYGNUM_HAL_INTERRUPT_SOUTH_BRIDGE_SMI    1
#define CYGNUM_HAL_INTERRUPT_CBUS_UART           2
#define CYGNUM_HAL_INTERRUPT_COREHI              3
#define CYGNUM_HAL_INTERRUPT_CORELO              4
#define CYGNUM_HAL_INTERRUPT_COMPARE             5
#define CYGNUM_HAL_INTERRUPT_TIMER               6
#define CYGNUM_HAL_INTERRUPT_KEYBOARD            7
#define CYGNUM_HAL_INTERRUPT_CASCADE             8
#define CYGNUM_HAL_INTERRUPT_TTY1                9
#define CYGNUM_HAL_INTERRUPT_TTY0               10
#define CYGNUM_HAL_INTERRUPT_11                 11
#define CYGNUM_HAL_INTERRUPT_FLOPPY             12
#define CYGNUM_HAL_INTERRUPT_PARALLEL           13
#define CYGNUM_HAL_INTERRUPT_REAL_TIME_CLOCK    14
#define CYGNUM_HAL_INTERRUPT_I2C                15
#define CYGNUM_HAL_INTERRUPT_PCI_AB             16
#define CYGNUM_HAL_INTERRUPT_PCI_CD             17
#define CYGNUM_HAL_INTERRUPT_MOUSE              18
#define CYGNUM_HAL_INTERRUPT_19                 19
#define CYGNUM_HAL_INTERRUPT_IDE_PRIMARY        20
#define CYGNUM_HAL_INTERRUPT_IDE_SECONDARY      21
```

The data passed to the ISR is pulled from a data table (`hal_interrupt_data`) which immediately follows the interrupt vector table. With 22 interrupts, the data table starts at address 0x80000258.

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.

Memory Maps

Memory Maps RedBoot sets up the following memory map on the Malta 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
----------------------- - -  -----------
0x80000000 - 0x81ffffff Y Y  SDRAM
0x9e000000 - 0x9e3fffff Y N  System flash (cached)
0x9fc00000 - 0x9fffffff Y N  System flash (mirrored)
0xa8000000 - 0xb7ffffff N N  PCI Memory Space
0xb4000000 - 0xb40fffff N N  Galileo System Controller
0xb8000000 - 0xb80fffff N N  Southbridge / ISA
0xb8100000 - 0xbbdfffff N N  PCI I/O Space
0xbe000000 - 0xbe3fffff N N  System flash (noncached)
0xbf000000 - 0xbfffffff N N  Board logic FPGA
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=malta_mips32_4kc
export ARCH_DIR=mips
export PLATFORM_DIR=malta
```

The names of configuration files are listed above with the description of the associated modes.

MIPS/RM7000 PMC-Sierra Ocelot

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Additional commands

The **exec** command which allows the loading and execution of Linux kernels, is supported for this architecture (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for MIPS boards are:

-b *<addr>*

Location to store command line and environment passed to kernel

-w *<time>*

Wait time in seconds before starting kernel

-c *"params"*

Parameters passed to kernel

<addr>

Kernel entry point, defaulting to the entry point of the last image loaded

Linux kernels on MIPS platforms expect the entry point to be called with arguments in the registers equivalent to a C call with prototype:

```
void Linux(int argc, char **argv, char **envp);
```

RedBoot will place the appropriate data at the offset specified by the *-b* parameter, or by default at address 0x80080000, and will set the arguments accordingly when calling into the kernel.

The default entry point, if no image with explicit entry point has been loaded and none is specified, is 0x80000750.

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
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=ocelot
export ARCH_DIR=mips
export PLATFORM_DIR=rm7000/ocelot
```

The names of configuration files are listed above with the description of the associated modes.

MIPS/VR4375 NEC DDB-VRC4375

Overview

RedBoot supports only serial port 1, which is connected to the upper of the stacked serial connectors on the board. The default serial port settings are 38400,8,N,1. FLASH management is also supported.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

A device programmer should be used to program a socketed FLASH part (AMD 29F040). The board as delivered is configured for a 512K EPROM. To install a FLASH ROM, Jumpers J30, J31 and J36 need to be changed as described in the board's User Manual.

Special RedBoot Commands

None.

Memory Maps

RedBoot sets up the memory map primarily as described in the board's User Manual. There are some minor differences, noted in the following table:

Physical Addresses	Virtual Addresses	Resource
00000000-01FFFFFF	80000000-81FFFFFF	Base SDRAM (cached)
00000000-01FFFFFF	A0000000-A1FFFFFF	Base SDRAM (uncached)
0C000000-0C0BFFFF	AC000000-AC0B0000	PCI IO space
0F000000-0F0001FF	AF000000-AF0001FF	VRC4375 Registers
1C000000-1C0FFFFF	BC000000-BC0FFFFF	VRC4372 Registers
1C100000-1DFFFFFF	BC100000-BDFFFFFF	PCI Memory space
1FC00000-1FC7FFFF	BFC00000-BFC7FFFF	FLASH ROM
80000000-8000000D	C0000000-C000000D	RTC
8000000E-80007FFF	C000000E-C0007FFF	NVRAM
81000000-81FFFFFF	C1000000-C1FFFFFF	Z85C30 DUART
82000000-82FFFFFF	C2000000-C2FFFFFF	Z8536 Timer
83000000-83FFFFFF	C3000000-C3FFFFFF	8255 Parallel port
87000000-87FFFFFF	C7000000-C7FFFFFF	Seven segment display

NOTE: By default the VRC4375 SIMM control registers are not programmed since the values used must depend on the SIMMs installed. If SIMMs are to be used, correct values must be placed in these registers before accessing the SIMM address range.

NOTE: The allocation of address ranges to devices in the PCI IO and memory spaces is handled by the eCos PCI support library. They do not correspond to those described in the board User Manual.

NOTE: The MMU has been set up to relocate the VRC4372 supported devices mapped at physical addresses 0x8xxxxxxx to virtual addresses 0xCxxxxxxx.

Ethernet Driver

The ethernet driver is in two parts:

A generic ether driver for the Intel i21143 device is located in `devs/eth/intel/i21143`. Its package name is `CYGPKG_DEVS_ETH_INTEL_I21143`.

The platform-specific ether driver is `devs/eth/mips/vrc4375`. Its package is `CYGPKG_DEVS_ETH_MIPS_VRC4375`. This tells the generic driver the address in IO memory of the chip, for example, and other configuration details. The ESA (MAC address) is by default collected from on-board serial EEPROM, unless configured statically within this package.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=vrc4373
export ARCH_DIR=mips
export PLATFORM_DIR=vrc4373
```

The names of configuration files are listed above with the description of the associated modes.

PowerPC/MPC860T Analogue & Micro PowerPC 860T

Overview

RedBoot uses the SMC1 serial port. The default serial port settings are 38400,8,N,1. Ethernet is also supported using the RJ-45 connector. Management of onboard flash is also supported.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROMRAM	[ROMRAM]	RedBoot running from RAM, but contained in the board's flash boot sector.	redboot_ROMRAM.ecm

Initial Installation Method

RedBoot must be installed at the A & M factory.

Special RedBoot Commands

None.

Memory Maps

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

```
Physical Address Range Description
---------------------- -----------
0x00000000 - 0x007fffff DRAM
0xfe000000 - 0xfe0fffff flash (AMD29LV8008B)
0xff000000 - 0xff0fffff MPC registers
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=viper
export ARCH_DIR=powerpc
export PLATFORM_DIR=viper
```

The names of configuration files are listed above with the description of the associated modes.

PowerPC/MPC8XX Motorola MBX

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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

Device programmer is used to program the XU1 socketed flash part (AM29F040B) with the ROM mode image 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.

Special RedBoot Commands

None.

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
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=mbx
export ARCH_DIR=powerpc
export PLATFORM_DIR=mbx
```

The names of configuration files are listed above with the description of the associated modes.

SuperH/SH3(SH7708) Hitachi EDK7708

Overview

RedBoot uses the serial port. The default serial port settings are 38400,8,N,1.

Management of onboard flash is also supported.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

Program the ROM RedBoot image into flash using an eprom programmer.

Memory Maps

RedBoot sets up the following memory map on the EDK7708 board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x8001ffff Flash (AT29LV1024)
0x88000000 - 0x881fffff DRAM
0xa4000000 - 0xa40000ff LED ON
0xb8000000 - 0xb80000ff LED ON
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=edk7708
export ARCH_DIR=sh
export PLATFORM_DIR=edk7708
```

The names of configuration files are listed above with the description of the associated modes.

SuperH/SH3(SH7709) Hitachi Solution Engine 7709

Overview

This description covers the MS7709SE01 variant. See the Section called *SuperH/SH3(SH77X9) Hitachi Solution Engine 77X9* for instructions for the MS7729SE01 and MS7709SSE0101 variants.

RedBoot uses the COM1 and COM2 serial ports. 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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

The Solution Engine ships with the Hitachi boot monitor in EPROM which allows for initial programming of RedBoot:

1. Set switch SW4-1 to ON [boot from EPROM]
2. Connect a serial cable to CN1 (SCI) and power up the board.
3. After the boot monitor banner, invoke the flash download/program command:

   ```
   Ready >fl
   ```

4. The monitor should now ask for input:

   ```
   Flash ROM data copy to RAM
   Please Send A S-format Record
   ```

   At this point copy the RedBoot ROM SREC file to the serial port:

   ```
   $ cat redboot_SE7709RP_ROM.eprom.srec > /dev/ttyS0
   ```

   Eventually you should see something like

   ```
   Start Addrs = A1000000
   End Addrs = A1xxxxxx
   Transfer complete
   ```

   from the monitor.
5. Set switch SW4-1 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner.

Special RedBoot Commands

The **exec** command which allows the loading and execution of Linux kernels is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the SE77x9 are:

-b *<addr>*

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i *<addr>*

Start address of initrd image

-j *<size>*

Size of initrd image

-c *"args"*

Kernel arguments string

-m *<flags>*

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f *<flags>*

RAM disk flags. Should normally be 0x4000

-r *<device number>*

Root device specification. /dev/ram is 0x0101

-l *<type>*

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

For the the SE77x9, Linux by default expects to be loaded at 0x8c001000 which conflicts with the data space used by RedBoot. To work around this, either change the CONFIG_MEMORY_START kernel option to a higher address, or use the compressed kernel image and load it at a higher address. For example, setting CONFIG_MEMORY_START to 0x8c100000, the kernel expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000.

Memory Maps

RedBoot sets up the following memory map on the SE77x9 board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x81000000 - 0x813fffff EPROM (M27C800)
0x8c000000 - 0x8dffffff DRAM
0xb0000000 - 0xb03fffff Ethernet (DP83902A)
0xb0800000 - 0xb08fffff 16C552A
0xb1000000 - 0xb100ffff Switches
0xb1800000 - 0xb18fffff LEDs
0xb8000000 - 0xbbffffff PCMCIA (MaruBun)
```

Ethernet Driver

The ethernet driver uses a hardwired ESA which can, at present, only be changed in CDL.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=se77x9
export ARCH_DIR=sh
export PLATFORM_DIR=se77x9
```

The names of configuration files are listed above with the description of the associated modes.

SuperH/SH3(SH7729) Hitachi HS7729PCI

Overview

RedBoot uses the COM1 and COM2 serial ports (and the debug port on the motherboard). The default serial port settings are 38400,8,N,1. Ethernet is also supported using a D-Link DFE-530TX PCI plugin card. Management of onboard flash is also supported.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

A ROM mode RedBoot image must be programmed into the two EPROMs. Two files with a split version of the ROM mode image is provided: it is also possible to recreate these from the `redboot.bin` file, but requires the split_word.c program in `hal/sh/hs7729pci/`*`VERSION`*`/misc` to be built and executed with the `redboot.bin` filename as sole argument.

After doing this it is advised that another ROM mode image of RedBoot is programmed into the on-board flash, and that copy be used for booting the board. This allows for software programmed updates of RedBoot

instead of having to reprogram the EPROMs.

1. Program the EPROMs with RedBoot. The .lo image should go in socket M1 and the .hi image in socket M2.
2. Set switch SW1-6 to ON [boot from EPROM]
3. Follow the instructions under Flash management for updating the flash copy of RedBoot, but force the flash destination address with

   `-f 0x80400000`
   due to setting of the SW1-6 switch.
4. Set switch SW1-6 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner. At this time you may want to issue the command **fis init** to initialize the flash table with the correct addresses.

Special RedBoot Commands

The **exec** command which allows the loading and execution of Linux kernels is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the HS7729PCI are:

-b *<addr>*

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i *<addr>*

Start address of initrd image

-j *<size>*

Size of initrd image

-c *"args"*

Kernel arguments string

-m *<flags>*

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f *<flags>*

RAM disk flags. Should normally be 0x4000

-r *<device number>*

Root device specification. /dev/ram is 0x0101

-l *<type>*

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

On the HS7729PCI, Linux expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000. This is configurable in the kernel using the CONFIG_MEMORY_START option.

Memory Maps

RedBoot sets up the following memory map on the HS7729PCI board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x80400000 - 0x807fffff EPROM (M27C800)
0x82000000 - 0x82ffffff SRAM
0x89000000 - 0x89ffffff SRAM
0x8c000000 - 0x8fffffff SDRAM
0xa8000000 - 0xa800ffff SuperIO (FDC37C935A)
0xa8400000 - 0xa87fffff USB function (ML60851C)
0xa8800000 - 0xa8bfffff USB host (SL11HT)
0xa8c00000 - 0xa8c3ffff Switches
0xa8c40000 - 0xa8c7ffff LEDs
0xa8c80000 - 0xa8cfffff Interrupt controller
0xb0000000 - 0xb3ffffff PCI (SD0001)
0xb8000000 - 0xbbffffff PCMCIA (MaruBun)
```

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=hs7729pci
export ARCH_DIR=sh
export PLATFORM_DIR=hs7729pci
```

The names of configuration files are listed above with the description of the associated modes.

SuperH/SH3(SH77X9) Hitachi Solution Engine 77X9

Overview

This description covers the MS7729SE01 and MS7709SSE0101 variants. See the Section called *SuperH/SH3(SH7709) Hitachi Solution Engine 7709* for instructions for the MS7709SE01 variant.

RedBoot uses the COM1 and COM2 serial ports. 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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm

Configuration	Mode	Description	File
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

The Solution Engine ships with the Hitachi boot monitor in EPROM which allows for initial programming of RedBoot:

1. Set switches SW4-3 and SW4-4 to ON [boot from EPROM]
2. Connect a serial cable to COM2 and power up the board.
3. After the boot monitor banner, invoke the flash download/program command:

   ```
   Ready >fl
   ```

4. The monitor should now ask for input:

   ```
   Flash ROM data copy to RAM
   Please Send A S-format Record
   ```

   At this point copy the RedBoot ROM SREC file to the serial port:

   ```
   $ cat redboot_ROM.eprom.srec > /dev/ttyS0
   ```

   Eventually you should see something like

   ```
   Start Addrs = A1000000
   End Addrs = A1xxxxxx
   Transfer complete
   ```

   from the monitor.
5. Set switch SW4-3 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner.

Special RedBoot Commands

The **exec** command which allows the loading and execution of Linux kernels is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the SE77x9 are:

-b *<addr>*

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i *<addr>*

Start address of initrd image

-j *<size>*

Size of initrd image

-c *"args"*

Kernel arguments string

-m *<flags>*

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f *<flags>*

RAM disk flags. Should normally be 0x4000

-r *<device number>*

Root device specification. /dev/ram is 0x0101

-l *<type>*

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

On the SE77x9, Linux expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000. This is configurable in the kernel using the CONFIG_MEMORY_START option.

Memory Maps

RedBoot sets up the following memory map on the SE77x9 board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x81000000 - 0x813fffff EPROM (M27C800)
0x8c000000 - 0x8dffffff SDRAM
0xb0000000 - 0xb03fffff Ethernet (DP83902A)
0xb0400000 - 0xb07fffff SuperIO (FDC37C935A)
0xb0800000 - 0xb0bfffff Switches
0xb0c00000 - 0xbfffffff LEDs
0xb1800000 - 0xb1bfffff PCMCIA (MaruBun)
```

Ethernet Driver

The ethernet driver uses a hardwired ESA which can, at present, only be changed in CDL.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=se77x9
export ARCH_DIR=sh
export PLATFORM_DIR=se77x9
```

The names of configuration files are listed above with the description of the associated modes.

SuperH/SH4(SH7751) Hitachi Solution Engine 7751

Overview

RedBoot uses the 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.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from the board's flash boot sector.	redboot_ROM.ecm
RAM	[RAM]	RedBoot running from RAM with RedBoot in the flash boot sector.	redboot_RAM.ecm

Initial Installation Method

The Solution Engine ships with the Hitachi boot monitor in EPROM which allows for initial programming of RedBoot:

1. Set switches SW5-3 and SW5-4 to ON [boot from EPROM]
2. Connect a serial cable to COM1 and power up the board.
3. After the boot monitor banner, invoke the flash download/program command:

   ```
   Ready >fl
   ```

4. The monitor should now ask for input:

   ```
   Flash ROM data copy to RAM
   Please Send A S-format Record
   ```

   At this point copy the RedBoot ROM SREC file to the serial port:

   ```
   $ cat redboot_ROM.eprom.srec > /dev/ttyS0
   ```

   Eventually you should see something like

   ```
   Start Addrs = A1000000
   End Addrs = A1xxxxxx
   Transfer complete
   ```

   from the monitor.
5. Set switch SW5-3 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner.

Special RedBoot Commands

The **exec** command which allows the loading and execution of Linux kernels is supported for this board (see the Section called *Executing Programs from RedBoot* in Chapter 2). The **exec** parameters used for the SE7751 are:

-b *<addr>*

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i *<addr>*

Start address of initrd image

-j *<size>*

Size of initrd image

-c *"args"*

Kernel arguments string

-m *<flags>*

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f *<flags>*

RAM disk flags. Should normally be 0x4000

-r *<device number>*

Root device specification. /dev/ram is 0x0101

-l *<type>*

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

On the SE7751, Linux expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000. This is configurable in the kernel using the CONFIG_MEMORY_START option.

Memory Maps

RedBoot sets up the following memory map on the SE7751 board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x81000000 - 0x813fffff EPROM (M27C800)
0x8c000000 - 0x8fffffff SDRAM
0xb8000000 - 0xb8ffffff PCMCIA (MaruBun)
0xb9000000 - 0xb9ffffff Switches
0xba000000 - 0xbaffffff LEDs
0xbd000000 - 0xbdffffff PCI MEM space
0xbe200000 - 0xbe23ffff PCI Ctrl space
0xbe240000 - 0xbe27ffff PCI IO space
```

Ethernet Driver

The ethernet driver uses a hardwired ESA which can, at present, only be changed in CDL.

Rebuilding RedBoot

These shell variables provide the platform-specific information needed for building RedBoot according to the procedure described in Chapter 3:

```
export TARGET=se7751
export ARCH_DIR=sh
export PLATFORM_DIR=se7751
```

The names of configuration files are listed above with the description of the associated modes.