RedBoot™ User’s Guide

Document Version 2.04, February 2007

Copyright

Warranty

Contents

RedBoot™ User's Guide

1 Getting Started with RedBoot

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

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

Here are some highlights of RedBoot's capabilities:

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

Although RedBoot is derived from 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. 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.

1.1 More information about RedBoot on the web

Information about RedBoot, including downloadable sources and documentation with the latest features and updates is available from RedBoot Project Page.

1.2 Installing RedBoot

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

Although there are other possible configurations, RedBoot is usually run from the target platform's flash boot sector or boot ROM, and is designed to run when your system is initially powered on. The method used to install the RedBoot image into non-volatile storage varies from platform to platform. In general, it requires that the image be programmed into flash in situ or programmed into the flash or ROM using a device programmer. In some cases this will be done at manufacturing time; the platform being delivered with RedBoot already in place. In other cases, you will have to program RedBoot into the appropriate device(s) yourself. Installing to flash in situ may require special cabling or interface devices and software provided by the board manufacturer. The details of this installation process for a given platform will be found in Installation and Testing. Once installed, user-specific configuration options may be applied, using the `fconfig` command, providing that persistent data storage in flash is present in the relevant RedBoot version. See Section 1.4 for details.

1.3 User Interface

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

RedBoot also contains a set of GDB "stubs", consisting of code which supports the GDB remote protocol. GDB stub mode is automatically invoked when the '$' character appears 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.

1.4 Configuring the RedBoot Environment

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

1.4.1 Target Network Configuration

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

NOTE

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

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

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

```
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.29
Default server IP address: 192.168.1.101
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>
```

1.4.2 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.

1.4.2.1 Enable TFTP on Red Hat Linux 6.2

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

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

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

   ```
   # service inet reload
   ```

1.4.2.2 Enable TFTP on Red Hat Linux 7

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

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

   Reload the xinetd configuration using the command:

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

   Create the directory /tftpboot using the command

   ```
   mkdir /tftpboot
   ```

NOTE

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

1.4.2.3 Enable BOOTP/DHCP server on Red Hat Linux

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

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
```

1.4.2.4 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.

1.4.2.5 RedBoot network gateway

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

1.4.3 Verification

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

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

1.4.4 Multiple Network Devices

RedBoot may support more than one network device. For instance a dual port NIC may be used or multiple PCI based NICs may be supported. RedBoot will only use one network device no matter how many are configured into the system. Preference for the device to use may be specified with the `fconfig` command. Suppose that a RedBoot is configured for two i82559 based NICs. These devices will typically be named `i82559_eth0` and `i82559_eth1`. To tell RedBoot to try the eth1 device first, use:

```
RedBoot> fconfig net_device
net_device: i82559_eth1
Update RedBoot non-volatile configuration - continue (y/n)? y
```

If the default network device is not found, RedBoot will search for other network devices and use the first one found. Please see the platform specific documentation sections of this manual for details on the network devices and their names supported by a specific platform.

2 RedBoot Commands and Examples

2.1 Introduction

RedBoot provides three basic classes of commands:

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

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

The basic format for commands is:

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

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

```
 [-S]
```

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

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

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

```
 [-s val]
```

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

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

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

```
operand
```

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

```
RedBoot> go 0x10044
```

specifies executing the code starting at location 0x10044.

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

```
RedBoot> help
Manage aliases kept in FLASH memory
      alias name [value]
Set/Query the system console baud rate
      baud [-b <rate>]
Manage machine caches
      cache [ON | OFF]
Display/switch console channel
      channel [-1|<channel number>]
```

```
Display disk partitions
      disks
Set/Query DNS server IP address
      dns [IP]
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>] [entry]
Help about help?
      help [<topic>]
Set/change IP addresses
      ip_address [-l <local_ip_address>] [-h <server_address>]
Load a file
      load [-r] [-v] [-d] [-c <channel>] [-h <host>] [-m {TFTP | xyzMODEM | 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.

2.2 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** 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** positions the cursor to the end of the line.
- **^D** erases the character under the cursor.
- **^F** moves the cursor one character to the right.
- **^B** moves the cursor one character to the left.
- **^P** 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** 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.

2.3 Common Commands

The general format of commands is:

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

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

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

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

2.3.1 Connectivity

dns [IP]

This command is used to show/change the IP address used for DNS lookups. If an IP address of 0.0.0.0 is entered, DNS lookups are disabled.

ip_address [-l <local_ip_address>] [-h <server_address>]

This command is used to show/change the basic IP addresses used by RedBoot. The -l option is used to set the IP address used by the target device. The -h option is used to set the default server address, such as is used by the `load` command.

ping - Check network connectivity ping

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

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

Arguments

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

2.3.2 General

alias name [value]

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 a script, the corresponding value will be substituted.

Aliases are kept in RedBoot's non-volatile configuration area, i.e. Flash memory.

This is an example of setting an alias. Notice the use of a quoted string when the value contains spaces.

```
RedBoot> alias SBUF "-b 0x100000"
Update RedBoot non-volatile configuration - are you sure (y/n)? y
... Unlock from 0x50f80000-0x50fc0000: .
... Erase from 0x50f80000-0x50fc0000: .
... Program from 0x0000b9e8-0x0000c9e8 at 0x50f80000: .
... Lock from 0x50f80000-0x50fc0000: .
```

This example shows querying of an alias, as well as how it might be used.

```
RedBoot> alias SBUF
'SBUF' = '-b 0x100000'
RedBoot> d %{SBUF}
0x00100000: FE03 00EA 0000 0000  0000 0000 0000 0000   |................|
0x00100010: 0000 0000 0000 0000  0000 0000 0000 0000   |................|
```

baud [-b value]

This command sets the baud rate for the system serial console. If the platform supports non-volatile configuration data, then the new value will be saved and used when the system is reset.

cache [ON | OFF]

This 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.

channel [-1|<channel number>]

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

When passed an argument of 0 upwards, this command switches the console channel to that channel number. The mapping between channel numbers and physical channels is platform specific.

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.

cksum -b <location> -l <length>

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

mfill -b <location> -l <length> [-p <pattern>] [-1|-2|-4]

Fills a range of memory with the given pattern. If the pattern is ommitted, then a value of zero is used. The options `-1, -2, -4` are used to select the length of the objects used while filling. For example, `-2` selects to fill 16 bits at a time, etc.

mcmp -s <location> -d <location> -l <length> [-1|-2|-4]

Compares two ranges of memory. The options `-1, -2, -4` are used to select the length of the objects used while comparing. For example, `-2` selects to compare 16 bits at a time, etc.

swab -b <location> -l <length> [-2|-4]

Swap bytes in a block of 16-bit or 32-bit words. The options `-2, -4` are used to select the length of the words to be byte-swapped. For example, `-2` selects to swap bytes inside 16 bit words, etc.

disks

This command is used to list disk partitions recognized by RedBoot.

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

Display (hex dump) a range of memory.

This command displays the contents of memory in hexadecimal format. It is most useful for examining a segment of RAM or flash. If the optional -s switch is provided, then the dump will be formatted as Motorola S-records. The `x` command 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.

```
RedBoot> du -b 0x100 -l 0x80
0x00000100: 3C60 0004 6063 2000 7C68 03A6 4E80 0020 |<‘..‘c .|h..N.. |
0x00000110: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000120: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000130: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000140: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000150: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000160: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0x00000170: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
RedBoot> d -b 0xfe00b000 -l 0x80
0xFE00B000: 2025 700A 0000 0000 4174 7465 6D70 7420 | %p.....Attempt |
0xFE00B010: 746F 206C 6F61 6420 532D 7265 636F 7264 |to load S-record|
0xFE00B020: 2064 6174 6120 746F 2061 6464 7265 7373 | data to address|
0xFE00B030: 3A20 2570 205B 6E6F 7420 696E 2052 414D |: %p [not in RAM|
0xFE00B040: 5D0A 0000 2A2A 2A20 5761 726E 696E 6721 |]...*** Warning!|
0xFE00B050: 2043 6865 636B 7375 6D20 6661 696C 7572 | Checksum failur|
0xFE00B060: 6520 2D20 4164 6472 3A20 256C 782C 2025 |e - Addr: %lx, %|
0xFE00B070: 3032 6C58 203C 3E20 2530 326C 580A 0000 |02lX <> %02lX...|
0xFE00B080: 456E 7472 7920 706F 696E 743A 2025 702C |Entry point: %p,|
RedBoot> x -b 0x3e00000 -s -l 0x80
S31503E00000803C04E98088000080880000808800046
S31503E000108250101889481000884000001C01A040174
S31503E000208250101889481000884000002C01A03FD68
S31503E000308250101889481000884000003C01A03F95B
S31503E000408250101889481000884000004C01A03F54E
S31503E000508250101889481000884000005C01A03F141
S31503E000608250101889481000884000006C01A03ED34
S31503E000708250101889481000884000007C01A03E927
```

reset

Reset the system.

This command resets the platform. On many targets this is equivalent to a power-on reset, but on others it may just cause a jump to the architecture's reset entry resulting in a reinitialization of the system.

version

Display RedBoot version information.

This command simply displays version information about RedBoot.

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

2.3.3 Download Process

load

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

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

Arguments

file	The name of the file on the TFTP 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.
-v	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.
-d	Decompress gzipped image during download.
-c	Specify which I/O channel to use for download. This option is only supported when using either xmodem or ymodem protocol.
-b	Specify the location in memory to which the file should be loaded. Executable images normally load at the location to which the file was linked. This option allows the file to be loaded to a specific memory location, possibly overriding any assumed location.
-r	Download raw data. Normally, the load command is used to load executable images into memory. This option allows for raw data to be loaded. If this option is given, -b will also be required.

-m The -m option is used to select the download method. The choices are:

xmodem*, *ymodem

serial download using standard protocols over a port. If no *-c* option is used, the current console port will be used, otherwise the protocol transfer will take place on the specified channel. When using this method, the *file* parameter is not required.

tftp

network based download using the TFTP protocol.

disk

load a file from local disk.

-h Used explicitly to name a host computer to contact for the download data. This works in TFTP mode only.

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

2.4 Flash Image System (FIS)

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

The available `fis` commands are:

fis init [-f]

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

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

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

fis [-c] [-d] list

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

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

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

```
RedBoot> fis list
Name           flash addr  Mem addr   Length   Entry point
RedBoot        0xA0000000  0xA0000000 0x020000 0x80000000
```

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

fis free

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

NOTE

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

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

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

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

Arguments

name	The name of the file, as shown in the FIS directory.
-b	The location in RAM used to obtain the image. This is a required option.
-l	The length of the image. If the image already exists, then the length is inferred from when the image was previously created. If specified, and the image exists, it must match the original value.
-f	The location in flash for the image, which will be inferred for extant images if not specified. If this is not provided, the first freeVblock which is large enough will be used. See `fis free`.
-e	The execution entry address. This is used if the starting address for an image is not known, or needs to be overridden.
-r	The location in RAM when the image is loaded via `fis load`. This only needs to be specified for images which will eventually loaded via `fis load`. Fixed images, such as RedBoot itself, will not need this.

-s	The length of the actual data to be written to flash. If not present then the image length (-1) value is assumed. If the value given by -s is less than -1, the remainder of the image in flash will be left in an erased state. Note that by using this option it is possible to create a completely empty flash image, for example to reserve space for use by applications other than RedBoot.
-n	If -n is specified, then only the FIS directory is updated, and no data is copied from RAM to flash. This feature can be used to recreate the FIS entry if it has been destroyed.

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

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

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

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

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

Arguments

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

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

fis delete name

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

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

```
junk             0xA0040000   0x8C400000  0x020000  0x80000000
RedBoot> fis delete junk
Delete image 'junk' - are you sure (y/n)? y
... Erase from 0xa0040000-0xa0060000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
```

NOTE

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

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

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

NOTE

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

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

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

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

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

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

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

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

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

Writes data from RAM at <location> to flash.

2.5 Persistent State Flash-based Configuration and Control

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

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

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

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

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

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

If the optional flag `-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”, Section 2.2.

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

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

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

```
RedBoot> fconfig nickname value
```

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

```
RedBoot> fconfig -l -n
boot_script: false
bootp: false
bootp_my_ip: 10.16.19.176
bootp_server_ip: 10.16.19.66
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 - are you sure (y/n)? y
... Unlock from 0x507c0000-0x507e0000: .
... Erase from 0x507c0000-0x507e0000: .
... Program from 0x0000a8d0-0x0000acd0 at 0x507c0000: .
... Lock from 0x507c0000-0x507e0000: .
```

```
     RedBoot>
```

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 - are you sure (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 - are you sure (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
GDB connection port: 9000
Network debug at boot time: false
```

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

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

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


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

NOTE

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

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 cancelling the script. Since a timeout value of zero (0) seconds would never allow the script to be aborted or cancelled, 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[backup]
>> {ROM}go
>> {RAM}fis li
>>
Boot script timeout (1000ms resolution): 2
Use BOOTP for network configuration: false
 ...
Update RedBoot non-volatile configuration - are you sure (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, Red Hat, 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[backup]
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, Red Hat, 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[backup]   0x00040000  0xA0020000  0x00040000  0xA0020040
RedBoot config    0x007DF000  0x007DF000  0x00001000  0x00000000
FIS directory     0x007E0000  0x007E0000  0x00020000  0x00000000
RedBoot>
```

2.6 Executing Programs from RedBoot

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

NOTE

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

go - Execute a program

The format of the `go` command is:

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

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

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

The `-c` option is used to allow execution with caches enabled. Normally, the `go` command will disable caches before execution.

The `-n` option is only available when RedBoot supports a network device. It causes the network interface to be disabled before execution begins.

exec - Execute a Linux kernel image

NOTE

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

Arguments

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

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

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

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

3 Rebuilding RedBoot

3.1 Introduction

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

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

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

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

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

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

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

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

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

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

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

```
cat >RedBoot_ROM.ecm <<EOF
cdl_component CYG_HAL_STARTUP {
       user_value ROM
};
EOF
ecosconfig import RedBoot_ROM.ecm
ecosconfig tree
make
```

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

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

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

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

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

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

```
RedBoot(tm) debug environment [ROM]
Red Hat certified release, version R1.xx - built 07:54:25, Oct 16 2000
Platform: Assabet development system (StrongARM 1110)
Copyright (C) 2000, Red Hat, Inc.
RAM: 0x00000000-0x02000000
flash: 0x50000000 - 0x50400000, 32 blocks of 0x00020000 bytes ea.
Socket Communications, Inc: Low Power Ethernet CF Revision C
5V/3.3V 08/27/98
IP: 192.168.1.29, Default server: 192.168.1.101
RedBoot>
```

3.1.1 Configuration export files

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

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

All dates and sizes are just examples.

3.1.1.1 Making RedBoot for RAM startup

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

$REDBOOTDIR

Full path to the toplevel RedBoot source release.

$BUILDDIR

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

$ECOS_REPOSITORY

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

$TARGET

e.g.atlas_mips32_4kc.

$ARCH_DIR

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

$PLATFORM_DIR

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

$VERSION

The version of the release, e.g. current.

You must make sure these variables are correctly set in your environment before proceeding, or the build will fail. The values for **$TARGET**, **$ARCH_DIR** and **$PLATFORM_DIR** for each board are given in the specific installation details for each board in Chapter 5, *Installation and Testing*. The value for **$VERSION** is the name of the package subdirectories - usually 'current' for sources checked out of CVS, or something like 'vX_Y' for a regular X.Y release.

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

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

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

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

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

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

3.1.2 Platform specific instructions

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

4 Updating RedBoot

4.1 Introduction

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

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

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

On some platforms, RedBoot runs in a ROMRAM-startup configuration: RedBoot is stored in the flash or ROM, but when the board is reset, it is copied to RAM and executes from there. For these platforms where RedBoot is in flash, the update in-situ process is simplified since the ROMRAM-startup configuration of RedBoot can update the flash content. The update procedure becomes:

- Start ROMRAM RedBoot, running from RAM.
- Update the primary RedBoot flash image.
- Reboot; run the new ROMRAM RedBoot from ram.

In order to execute this process, only one version of RedBoot is required; a ROMRAM-startup configuration. This image is typically provided as part of the RedBoot package, but it may also be rebuilt from source using the instructions provided for the platform.

The following is a more detailed look at these steps. For this process, it is assumed that the target is connected to a host system and that there is some sort of serial connection used for the RedBoot CLI. For platforms with a ROMRAM-startup configuration of RedBoot, skip to Section 4.1.3.

4.1.1 Start RedBoot, Running from flash

To start RedBoot, reset the platform.

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

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

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

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

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

NOTE

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

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

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

4.1.3 Update the primary RedBoot flash image

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

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

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

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

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

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

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

NOTE

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

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

NOTE

After the flash image directory has been initialized with the `fis init` command it is possible to use a shorthand version of the `fis create` command since it can get the necessary information from the flash image directory:

```
RedBoot> fis create RedBoot -b <flash source>
```

4.1.4 Reboot; run RedBoot from flash

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

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 -i` command.

NOTE

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

5 Installation and Testing

5.1 Intel Xscale IXDP465 Evaluation Board

5.1.1 Overview

RedBoot supports the builtin high-speed and console UARTs, a single PCI based i82559 ethernet card (i82559_eth0), and all three NPE ethernet ports (npe_eth0, npe_eth1, and npe_eth2) for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 32MiB boot flash on the mainboard.

The following RedBoot configurations are supported:

Configuration	**Mode**	**Description**	**File**
ROM	[ROM]	RedBoot running from 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 booting from flash, running from RAM.	redboot_ROM-RAM.ecm

5.1.2 Initial Installation Method

The IXDP465 flash is not socketed, so initial installation must be done using an appropriate JTAG based solution. The ROM or ROMRAM mode RedBoot is programmed into the boot flash at address 0x00000000.

NOTE

Make sure that jumpers on the CPU card connect the EEPROM I2C lines to the IXP based, not GPIO based I2C bus lines. Also, make sure that baseboard is jumpered to enable writes to the EEPROM.

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 Section 2.5 for more details.

5.1.3 LED Codes

RedBoot uses the LCD 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
        Setup LCD
1001
        Enable Icache
1002
        Initialize SDRAM controller
1003
        Switch flash (CS0) from 0x00000000 to 0x50000000
1004
        Fill SDRAM with zero to initialize ECC codes
1005
        [ROMRAM only] Copy RedBoot to SDRAM and execute from there
1006
        Build MMU table in SDRAM
1007
        Setup TTB to point to page table
1008
        Turn on MMU
1009
        Enable DCache
100A
        Enable branch target buffer
100B
        Drain write and fill buffer
        Flush caches
100C
        Enable ECC support
100D
        Start up the eCos kernel or RedBoot
0001
```

5.1.4 Special RedBoot Commands

The `set_npe_mac` command allows the printing and setting of the ethernet MAC address of the three NPE ports. To print the current MAC addresses of all ethernet ports:

```
RedBoot> set_npe_mac
NPE eth0 mac: 00:02:b3:3c:15:ab
NPE eth1 mac: 00:02:b3:3c:16:46
NPE eth2 mac: 00:02:b3:3c:16:48
```

To set the mac address for NPE eth1 (NPEC), use something like:

```
RedBoot> set_npe_mac -p 1 00:02:b3:3c:16:46
```

but with the desired mac address.

5.1.5 Rebuilding RedBoot

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

```
export TARGET=ixdp465
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/ixdp465
```

Optionally,

```
export TARGET=ixdp465_npe
```

could be used to include NPE ethernet support.

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

5.1.6 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 IXDP465 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
#define CYGNUM_HAL_INTERRUPT_USB_HOST     32
#define CYGNUM_HAL_INTERRUPT_I2C          33
#define CYGNUM_HAL_INTERRUPT_SPI          34
#define CYGNUM_HAL_INTERRUPT_TIMESYNC     35
#define CYGNUM_HAL_INTERRUPT_EAU_DONE     36
#define CYGNUM_HAL_INTERRUPT_SHA_DONE     37
#define CYGNUM_HAL_INTERRUPT_SWCP_PERR    58
#define CYGNUM_HAL_INTERRUPT_QMGR_PERR    60
#define CYGNUM_HAL_INTERRUPT_MCU_ERR      61
#define CYGNUM_HAL_INTERRUPT_EXP_PERR     62
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table. With 64 interrupts, the data table starts at address 0x8104.

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

5.1.7 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 (MiB) Description
---------------   ----------------  ---  ---------  -----------
   0x00000000       0x00000000      010     128     SDRAM (cached)
   0x20000000       0x00000000      000     128     SDRAM (uncached)
   0x30000000       0x00000000      010     128     SDRAM (cached data coherent)
   0x48000000       0x48000000      000      64     PCI Data
   0x50000000       0x50000000      010      16     Flash (CS0)
   0x51000000       0x51000000      000     112     CS1 - CS7
   0x60000000       0x60000000      000      64     Queue Manager
   0xA0000000       0x50000000      010      16     Flash (CS0, data coherent)
   0xC0000000       0xC0000000      000       1     PCI Controller
   0xC4000000       0xC4000000      000       1     Exp. Bus Config
   0xC8000000       0xC8000000      000       1     Misc IXP4xx IO
   0xCC000000       0xCC000000      000       1     SDRAM Config
```

5.1.8 Platform Resource Usage

The IXP4xx programmable OStimer0 is used for timeout support for networking and XModem file transfers.

5.2 Intel(r) Next Generation of Network Processors (Code Name Hamoa) Reference Platform

5.2.1 Overview

RedBoot supports the builtin high-speed UART, a single PCI based E100 (i82559_eth0), a single PCI based E1000 ethernet card (e1000_eth0), and both NPE ethernet ports (npe_wan and npe_lan) for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MiB boot flash on the board.

The following RedBoot configurations are supported:

Configuration	Mode	Description	File
ROM	[ROM]	RedBoot running from 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 booting from flash, running from RAM.	redboot_ROM-RAM.ecm

5.2.2 Initial Installation Method

The onboard flash is not socketed, so initial installation must be done using an appropriate JTAG based solution. The ROM or ROMRAM mode RedBoot images are programmed into the boot flash at offset 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 `fconfig` be run to initialize the flash configuration area. See Section 2.5 for more details.

5.2.3 LED Codes

RedBoot uses 8 discrete LEDs to indicate an 8 bit status code during board initialization. The LEDs are arranged as two rows of four LEDs. The top row of LEDs are the most significant 4 bits of the status code and the bottom row are the least significant 4 bits. Possible codes are:

```
LED    Actions
-------------------------------------------------------------
  Power-On/Reset
       Set the CPSR
       Enable coprocessor access
       Drain write and fill buffer
       Setup expansion bus chip selects
F1
       Enable Icache
F2
       Initialize SDRAM controller
F3
       Initialize hardware registers.
F4
       Switch flash (CS0) from 0x00000000 to 0x50000000
F5
       [ROMRAM only] Copy RedBoot to SDRAM and execute from there
F6
       Build MMU table in SDRAM
F7
       Setup TTB to point to page table
F8
       Turn on MMU
F9
       Enable DCache
FA
       Enable branch target buffer
FB
       Drain write and fill buffer
       Flush caches
FC
       Set up low level vectors.
03
       initialize PCI bus.
01
       Start RedBoot command shell.
```

5.2.4 Rebuilding RedBoot

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

```
export TARGET=kixrp435
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/kixrp435
```

Optionally,

```
export TARGET=kixrp435_npe
```

could be used to include NPE ethernet support.

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

5.2.5 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 Intel(r) Next Generation of Network Processors (Code Name Hamoa) Reference Platform, 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
#define CYGNUM_HAL_INTERRUPT_USB_HOST     32
#define CYGNUM_HAL_INTERRUPT_I2C          33
#define CYGNUM_HAL_INTERRUPT_SPI          34
#define CYGNUM_HAL_INTERRUPT_TIMESYNC     35
#define CYGNUM_HAL_INTERRUPT_EAU_DONE     36
#define CYGNUM_HAL_INTERRUPT_SHA_DONE     37
#define CYGNUM_HAL_INTERRUPT_SWCP_PERR    58
#define CYGNUM_HAL_INTERRUPT_QMGR_PERR    60
#define CYGNUM_HAL_INTERRUPT_MCU_ERR      61
#define CYGNUM_HAL_INTERRUPT_EXP_PERR     62
```

The data passed to the ISR is pulled from a data table (`hal_interrupt_data`) which immediately follows the interrupt vector table. With 64 interrupts, the data table starts at address 0x8104.

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

5.2.6 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 (MiB)	Description
0x00000000	0x00000000	010	128	SDRAM (cached)
0x20000000	0x00000000	000	128	SDRAM (uncached)
0x30000000	0x00000000	010	128	SDRAM (cached data coherent)
0x48000000	0x48000000	000	64	PCI Data
0x50000000	0x50000000	010	16	Flash (CS0)
0x51000000	0x51000000	000	112	CS1 - CS4
0x60000000	0x60000000	000	64	Queue Manager
0xA0000000	0x50000000	010	16	Flash (CS0, data coherent)
0xC0000000	0xC0000000	000	1	PCI Controller
0xC4000000	0xC4000000	000	1	Exp. Bus Config
0xC8000000	0xC8000000	000	1	Misc IXP4xx IO
0xCC000000	0xCC000000	000	1	SDRAM Config

5.2.7 Platform Resource Usage

The IXP4xx programmable OStimer0 is used for timeout support for networking and XModem file transfers.

5.3 Intel Xscale IXDP425 Network Processor Evaluation Board

5.3.1 Overview

RedBoot supports the builtin high-speed and console UARTs, a PCI based i82559 ethernet card (i82559_eth0), and the NPE ethernet ports (npe_eth0 and npe_eth1) for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MiB boot flash on the mainboard.

The following RedBoot configurations are supported:

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

5.3.2 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. Even if this message is not seen, it is recommended that the `fconfig` be run to initialize the flash configuration area. See Section 2.5 for more details.

5.3.3 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

5.3.4 Special RedBoot Commands

The `set_npe_mac` command allows the printing and setting of the ethernet MAC address of the two NPE ports. To print the current MAC addresses of both ports:

```
RedBoot> set_npe_mac
NPE eth0 mac: 00:02:b3:3c:15:ab
NPE eth1 mac: 00:02:b3:3c:16:46
```

To set the mac address for NPE eth1 (NPEC), use something like:

```
RedBoot> set_npe_mac -p 1 00:02:b3:3c:16:46
```

but with the desired mac address.

5.3.5 Rebuilding RedBoot

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

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

Optionally,

```
export TARGET=ixdp425_npe
```

could be used to include NPE ethernet support.

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

5.3.6 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/bulverde/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.

5.3.7 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 (MiB)	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

5.3.8 Platform Resource Usage

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

5.4 Intel Xscale Generic Residential Gateway

5.4.1 Overview

RedBoot supports the console UART, a PCI based i82559 ethernet card (i82559_eth0), and both NPE ethernet ports (npe_eth0 and npe_eth1) for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MiB onboard flash.

The following RedBoot configurations are supported:

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

5.4.2 Initial Installation Method

The GRG 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 Section 2.5 for more details.

5.4.3 Rebuilding RedBoot

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

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

Optionally,

```
export TARGET=grg_npe
```

could be used to include NPE ethernet support.

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

5.4.4 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/bulverde/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.

5.4.5 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 (MiB) 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 CPU IO
   0xCC000000        0xCC000000     000      1      SDRAM Config
```

5.4.6 Platform Resource Usage

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

5.5 Intel IXDPG425 Network Gateway Reference Platform

5.5.1 Overview

RedBoot supports the high-speed UART and NPE ethernet ports (npe_eth0 and npe_eth1) for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MiB onboard flash.

The following RedBoot configurations are supported:

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

5.5.2 Initial Installation Method

The IXDPG425 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 Section 2.5 for more details.

5.5.3 Rebuilding RedBoot

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

```
export TARGET=ixdpg425
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/ixdpg425
```

Optionally,

```
export TARGET=ixdpg425_npe
```

could be used to include NPE ethernet support.

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

5.5.4 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 IXDPG425 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.

5.5.5 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 (MiB) Description
---------------   ----------------  ---  ---------  -----------
   0x00000000        0x00000000     010      32     SDRAM (cached)
   0x10000000        0x00000000     010      32     SDRAM (alias)
   0x20000000        0x00000000     000      32     SDRAM (uncached)
   0x30000000        0x00000000     010      32     SDRAM (data coherent)
   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
```

5.5.6 Platform Resource Usage

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

Index