RedBoot™ User's Guide

Document Version 2.0, October 2004

Copyright

Warranty

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

For non-customers, eCos and RedBoot software has NO WARRANTY.

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

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

How to Contact Red Hat

Red Hat Corporate Headquarters
1801 Varsity Drive
Raleigh NC 27606 USA
Telephone (toll free): +1 888 REDHAT 1 (+1 888 733 4281)
Telephone (main line): +1 919 547 0012
Telephone (FAX line): +1 919 547 0024
Website: http://www.redhat.com/

Contents

RedBoot™ User's Guide

1 Getting Started with RedBoot

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

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

Here are some highlights of RedBoot's capabilities:

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

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

1.1 More information about RedBoot on the web

Information about the RedBoot product, including information about details of porting, customization, training and technical support services from Red Hat, is available from the RedBoot Product web site.

The RedBoot Net Distribution web site contains downloadable sources and documentation for all publically released targets, including the latest features and updates.

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 '$' or '+' 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: `/tftpboot/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.

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.

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
S31503E00000803C04E980880000808800008088000046
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 (-l) value is assumed. If the value given by -s is less than -l, 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 Cyclone IQ80310

5.1.1 Overview

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

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

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

5.1.2 Initial Installation Method

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

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

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


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


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


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

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

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

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

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

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

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

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

5.1.3 Error codes

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

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

5.1.4 Using RedBoot with ARM Bootloader

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

5.1.5 Flash management

5.1.5.1 Updating the primary RedBoot image

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

ARM bootloader in flash boot sector

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

RedBoot in flash boot sector

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

5.1.5.2 Updating the secondary RedBoot image

ARM bootloader in flash boot sector

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

RedBoot in flash boot sector

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

5.1.6 Special RedBoot Commands

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

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

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

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

5.1.7 IQ80310 Hardware Tests

```
1 - Memory Tests
2 - Repeating Memory Tests
3 - 16C552 DUART Serial Port Tests
4 - Rotary Switch S1 Test for positions 0-3
5 - 7 Segment LED Tests
6 - Backplane Detection Test
7 - Battery Status Test
8 - External Timer Test
9 - i82559 Ethernet Configuration
10 - i82559 Ethernet Test
11 - i960Rx/303 PCI Interrupt Test
12 - Internal Timer Test
13 - Secondary PCI Bus Test
14 - Primary PCI Bus Test
15 - Battery Backup SDRAM Memory Test
16 - GPIO Test
17 - Repeat-On-Fail Memory Test
18 - Coyonosa Cache Loop (No return)
19 - Show Software and Hardware Revision
0 - quit
Enter the menu item number (0 to quit):
```

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

5.1.8 Rebuilding RedBoot

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

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

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

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

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

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

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

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

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

5.1.9 Interrupts

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

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

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

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


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


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


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

```
#define CYGNUM_HAL_INTERRUPT_BIST                26


// *** External board interrupts (XINT3) ***
#define CYGNUM_HAL_INTERRUPT_TIMER        27 // external timer
#define CYGNUM_HAL_INTERRUPT_ETHERNET     28 // onboard enet
#define CYGNUM_HAL_INTERRUPT_SERIAL_A     29 // 16x50 uart A
#define CYGNUM_HAL_INTERRUPT_SERIAL_B     30 // 16x50 uart B
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTD   31 // secondary PCI INTD
// The hardware doesn’t (yet?) provide masking or status for these
// even though they can trigger cpu interrupts. ISRs will need to
// poll the device to see if the device actually triggered the
// interrupt.
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTC   32 // secondary PCI INTC
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTB   33 // secondary PCI INTB
#define CYGNUM_HAL_INTERRUPT_PCI_S_INTA   34 // secondary PCI INTA


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


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

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

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

5.1.10 Memory Maps

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

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x00000fff    flash Memory
0x00001000 - 0x00001fff    80312 Internal Registers
0x00002000 - 0x007fffff    flash Memory
0x00800000 - 0x7fffffff    PCI ATU Outbound Direct Window
0x80000000 - 0x83ffffff    Primary PCI 32-bit Memory
0x84000000 - 0x87ffffff    Primary PCI 64-bit Memory
0x88000000 - 0x8bffffff    Secondary PCI 32-bit Memory
0x8c000000 - 0x8fffffff    Secondary PCI 64-bit Memory
0x90000000 - 0x9000ffff    Primary PCI IO Space
0x90010000 - 0x9001ffff    Secondary PCI IO Space
0x90020000 - 0x9fffffff    Unused
0xa0000000 - 0xbfffffff    SDRAM
0xc0000000 - 0xefffffff    Unused
0xf0000000 - 0xffffffff    80200 Internal Registers
```

```
Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x00000fff  Y Y  SDRAM
0x00001000 - 0x00001fff  N N  80312 Internal Registers
0x00002000 - 0x007fffff  Y N  flash Memory
0x00800000 - 0x7fffffff  N N  PCI ATU Outbound Direct Window
0x80000000 - 0x83ffffff  N N  Primary PCI 32-bit Memory
0x84000000 - 0x87ffffff  N N  Primary PCI 64-bit Memory
0x88000000 - 0x8bffffff  N N  Secondary PCI 32-bit Memory
0x8c000000 - 0x8fffffff  N N  Secondary PCI 64-bit Memory
0x90000000 - 0x9000ffff  N N  Primary PCI IO Space
0x90010000 - 0x9001ffff  N N  Secondary PCI IO Space
0xa0000000 - 0xbfffffff  Y Y  SDRAM
0xc0000000 - 0xcfffffff  Y Y  Cache Flush Region
0xd0000000 - 0xd0000fff  Y N  first 4k page of flash
0xf0000000 - 0xffffffff  N N  80200 Internal Registers
```

5.1.11 Resource Usage

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

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

Both RAM based RedBoot configurations are designed to run from RAM at addresses 0xa0020000 - 0xa005ffff. RAM addresses from 0xa0060000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images before they are written to flash.

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

5.2 Intel IQ80321

5.2.1 Overview

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

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

A special RedBoot command: `diag` is used to access a set of hardware diagnostics.

5.2.2 Initial Installation Method

The board manufacturer provides a DOS application which is capable of programming the flash over the PCI bus, and this is required for initial installations of RedBoot. Please see the board manual for information on using this utility. In general, the process involves programming the flash based RedBoot to flash. RedBoot should be programmed to flash address 0x00000000 using the DOS utility.

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

```
For RedBoot running from the flash boot sector:
loaders/iq80321/iq80321-rom.bin
loaders/iq80321/iq80321-rom.elf
loaders/iq80321/iq80321-rom.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
loaders/iq80321/iq80321-ram.bin
loaders/iq80321/iq80321-ram.elf
loaders/iq80321/iq80321-ram.srec
```

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

To install RedBoot to run from the flash boot sector, use the manufacturer's flash utility to install the `loaders/iq80321/iq80321-rom.bin` image at address zero.

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

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

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

```
RedBoot> fis init
About to initialize [format] FLASH image system - are you sure (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
    ... Unlock from 0xf07e0000-0xf0800000: .
    ... Erase from 0xf07e0000-0xf0800000: .
    ... Program from 0x01ddf000-0x01ddf400 at 0xf07e0000: .
    ... Lock from 0xf07e0000-0xf0800000: .
```

5.2.3 Switch Settings

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

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

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

5.2.4 LED Codes

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

```
LED     Actions
-------------------------------------------------------------
   Power-On/Reset
88
        Set the CPSR
        Enable coprocessor access
        Drain write and fill buffer
        Setup PBIU chip selects
A1
        Enable the Icache
```

```
A2
        Move FLASH chip select from 0x0 to 0xF0000000
        Jump to new FLASH location
A3
        Setup and enable the MMU
A4
        I2C interface initialization
90
        Wait for I2C initialization to complete
91
        Send address (via I2C) to the DIMM
92
        Wait for transmit complete
93
        Read SDRAM PD data from DIMM
94
        Read remainder of EEPROM data.
        An error will result in one of the following
        error codes on the LEDs:
 77 BAD EEPROM checksum
 55 I2C protocol error
 FF bank size error
A5
        Setup DDR memory interface
A6
        Enable branch target buffer
        Drain the write & fill buffers
        Flush Icache, Dcache and BTB
        Flush instuction and data TLBs
        Drain the write & fill buffers
SL
        ECC Scrub Loop
SE
A7
        Clean, drain, flush the main Dcache
A8
        Clean, drain, flush the mini Dcache
        Flush Dcache
        Drain the write & fill buffers
A9
        Enable ECC
AA
        Save SDRAM size
        Move MMU tables into RAM
AB
        Clean, drain, flush the main Dcache
        Clean, drain, flush the mini Dcache
        Drain the write & fill buffers
AC
        Set the TTB register to DRAM mmu_table
AD
        Set mode to IRQ mode
A7
        Move SWI & Undefined "vectors" to RAM (at 0x0)
A6
        Switch to supervisor mode
A5
        Move remaining "vectors" to RAM (at 0x0)
A4
        Copy DATA to RAM
        Initialize interrupt exception environment
        Initialize stack
        Clear BSS section
A3
        Call platform specific hardware initialization
A2
        Run through static constructors
A1
        Start up the eCos kernel or RedBoot
```

5.2.5 Flash management

5.2.5.1 Updating the primary RedBoot image

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

```
-f 0xf0000000
-b 0x100000
-l 0x40000
```

5.2.5.2 Updating the secondary RedBoot image

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

```
-f 0xf0040000
-b 0x20000
-r 0x20000
-l 0x40000
```

5.2.6 Special RedBoot Commands

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

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

  IQ80321 Hardware Tests

 1 - Memory Tests
 2 - Repeating Memory Tests
 3 - Repeat-On-Fail Memory Tests
 4 - Rotary Switch S1 Test
 5 - 7 Segment LED Tests
 6 - i82544 Ethernet Configuration
 7 - Baterry Status Test
 8 - Battery Backup SDRAM Memory Test
 9 - Timer Test
10 - PCI Bus test
11 - CPU Cache Loop (No Return)
 0 - quit
Enter the menu item number (0 to quit):
```

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

5.2.6.1 Memory Tests

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

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

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

Size of memory to test (in hex): 200000

Testing memory from 0x00100000 to 0x002fffff.

Walking 1’s test:
0000000100000002000000040000000800000010000000200000004000000080
0000010000000200000004000000080000001000000020000000400000008000
0001000000020000000400000008000000100000002000000040000000800000
0100000002000000040000000800000010000000200000004000000080000000
passed
32-bit address test: passed
32-bit address bar test: passed
8-bit address test: passed
Byte address bar test: passed
Memory test done.
```

5.2.6.2 Repeating Memory Tests

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

5.2.6.3 Repeat-On-Fail Memory Tests

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

5.2.6.4 Rotary Switch S1 Test

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

5.2.6.5 7 Segment LED Tests

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

5.2.6.6 i82544 Ethernet Configuration

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

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


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

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

5.2.6.7 Battery Status Test

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

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

5.2.6.8 Battery Backup SDRAM Memory Test

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

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

5.2.6.9 Timer Test

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

5.2.6.10 PCI Bus Test

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

5.2.6.11 CPU Cache Loop

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

5.2.7 Rebuilding RedBoot

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

```
% export TOPDIR=‘pwd‘
% export ECOS_REPOSITORY=\
    ${TOPDIR}/src/ecos-monitors/redboot-DATE-intel/packages
% mkdir ${TOPDIR}/build
% cd ${TOPDIR}/build
% ecosconfig new iq80321 redboot
% ecosconfig import \
   ${ECOS_REPOSITORY}/hal/arm/xscale/iq80321/VERSION/misc/redboot_ROM.ecm
% ecosconfig tree
% make
```

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

```
% ecosconfig import \
  ${ECOS_REPOSITORY}/hal/arm/xscale/iq80321/VERSION/misc/redboot_RAM.ecm
```

5.2.8 Interrupts

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

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

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

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

```
#define CYGNUM_HAL_INTERRUPT_RSVD_26       26
#define CYGNUM_HAL_INTERRUPT_XINT0         27
#define CYGNUM_HAL_INTERRUPT_XINT1         28
#define CYGNUM_HAL_INTERRUPT_XINT2         29
#define CYGNUM_HAL_INTERRUPT_XINT3         30
#define CYGNUM_HAL_INTERRUPT_HPI           31
```

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

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

5.2.9 Memory Maps

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

NOTE

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

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

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


Default Virtual Map      X C B  Description
-----------------------  - - -  ----------------------------------
0x00000000 - 0x1fffffff  1 1 1  SDRAM
0x20000000 - 0x9fffffff  0 0 0  ATU Outbound Direct Window
0xa0000000 - 0xb00fffff  0 0 0  ATU Outbound Translate Windows
```

```
0xc0000000 - 0xdfffffff  0 0 0  Uncached alias for SDRAM
0xe0000000 - 0xe00fffff  1 1 1  Cache flush region (no phys mem)
0xf0000000 - 0xf0800000  0 1 0  FLASH               (PBIU CS0)
0xfe800000 - 0xfe800fff  0 0 0  UART                (PBIU CS1)
0xfe840000 - 0xfe840fff  0 0 0  Left 7-segment LED  (PBIU CS3)
0xfe850000 - 0xfe850fff  0 0 0  Right 7-segment LED (PBIU CS2)
0xfe8d0000 - 0xfe8d0fff  0 0 0  Rotary Switch       (PBIU CS4)
0xfe8f0000 - 0xfe8f0fff  0 0 0  Baterry Status      (PBIU CS5)
0xfff00000 - 0xffffffff  0 0 0  Verde Memory mapped Registers

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

5.2.10 Resource Usage

The flash based RedBoot image occupies flash addresses 0xf0000000 - 0xf003ffff and RAM addresses (0x00000000 - 0x0001ffff).

The RAM based RedBoot configuration is designed to run from RAM at addresses 0x00020000 - 0x0005ffff. RAM addresses from 0x00060000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images before they are written to flash.

The Verde programmable timer0 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 for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MB boot flash on the mainboard.

The following RedBoot configurations are supported:

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

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 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
```

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

5.3.5 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 Mainstone 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.

5.3.6 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 (MB)	Description
0x00000000	0x00000000	010	256	SDRAM (cached)
0x10000000	0x10000000	010	256	SDRAM (alias)
0x20000000	0x00000000	000	256	SDRAM (uncached)
0x48000000	0x48000000	000	64	PCI Data
0x50000000	0x50000000	010	16	Flash (CS0)
0x51000000	0x51000000	000	112	CS1 - CS7
0x60000000	0x60000000	000	64	Queue Manager
0xC0000000	0xC0000000	000	1	PCI Controller
0xC4000000	0xC4000000	000	1	Exp. Bus Config

```
0xC8000000       0xC8000000      000       1     Misc IXP425 IO
0xCC000000       0xCC000000      000       1     SDRAM Config
```

5.3.7 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 for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot also supports flash management for the 16MB onboard flash.

The following RedBoot configurations are supported:

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

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
```

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 Mainstone 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 (MB)  Description
---------------   ----------------  ---  ---------  -----------
   0x00000000        0x00000000      010      32     SDRAM (cached)
   0x10000000        0x00000000      010      32     SDRAM (alias)
   0x20000000        0x00000000      000      32     SDRAM (uncached)
   0x48000000        0x48000000      000      64     PCI Data
   0x50000000        0x50000000      010      16     Flash (CS0)
   0x51000000        0x51000000      000     112     CS1 - CS7
   0x60000000        0x60000000      000      64     Queue Manager
   0xC0000000        0xC0000000      000       1     PCI Controller
   0xC4000000        0xC4000000      000       1     Exp. Bus Config
   0xC8000000        0xC8000000      000       1     Misc 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 Motorola PrPMC1100 CPU card

5.5.1 Overview

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

The following RedBoot configurations are supported:

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

5.5.2 Initial Installation Method

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

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

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

This is normal, and indicates that the flash should be configured for use by RedBoot. Even if this message is not seen, it is recommended that the `fconfig` be run to initialize the flash configuration area. See 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=prpmc1100
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/prpmc1100
```

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

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 Mainstone 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.

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

5.5.6 Platform Resource Usage

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

5.6 Intel SA1100 (Brutus)

5.6.1 Overview

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

Two basic RedBoot configurations are supported:

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

5.6.2 Initial Installation Method

Device programmer is used to program socketecflash parts.

5.6.3 Special RedBoot Commands

None.

5.6.4 Memory Maps

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

NOTE

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

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


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

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

5.6.5 Resource Usage

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

5.6.6 Rebuilding RedBoot

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

5.7 Intel StrongArm EBSA 285

5.7.1 Overview

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

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

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

5.7.2 Initial Installation Method

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

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

5.7.3 Flash management

5.7.3.1 Updating the primary RedBoot image

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

```
-f 0x41000000
-b 0x100000
-l 0x40000
```

5.7.3.2 Updating the secondary RedBoot image

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

```
-f 0x41040000
-b 0x20000
-r 0x20000
-l 0x40000
```

5.7.4 Communication Channels

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

5.7.5 Special RedBoot Commands

None.

5.7.6 Memory Maps

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

NOTE

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

```
Address Range            C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01ffffff  Y Y  SDRAM
0x40000000 - 0x400fffff  N N  21285 Registers
0x41000000 - 0x413fffff  Y N  flash
0x42000000 - 0x420fffff  N N  21285 CSR Space
0x50000000 - 0x50ffffff  Y Y  Cache Clean
0x78000000 - 0x78ffffff  N N  Outbound Write Flush
0x79000000 - 0x7c0fffff  N N  PCI IACK/Config/IO
0x80000000 - 0xffffffff  N Y  PCI Memory
```

5.7.7 Resource Usage

The flash based RedBoot image occupies flash addresses 0x41000000 - 0x4103ffff. It also reserves the first 192K bytes of RAM for runtime uses. The RAM based RedBoot image occupies RAM addresses 0x30000 - 0x5ffff. RAM addresses from 0x60000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

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

5.7.8 Building eCos Test Cases to run with old RedBoots

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

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

located here in the common HAL configuration tree:

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

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

If you have been using built-in stubs to acquire support for thread-aware debugging, you can still do that, but you must only use the serial device for GDB connection and you must not enable the option mentioned above. However, it is no longer necessary to do that to get thread-awareness; RedBoot is thread aware.

5.7.9 Rebuilding RedBoot

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

5.8 Intel SA1100 Multimedia Board

5.8.1 Overview

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

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

5.8.2 Initial Installation Method

A device programmer is used to program socketed flash parts.

5.8.3 Special RedBoot Commands

None.

5.8.4 Memory Maps

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

NOTE

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

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


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

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

5.8.5 Resource Usage

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

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

5.8.6 Rebuilding RedBoot

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

5.9 Intel SA1110 (Assabet)

5.9.1 Overview

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

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

5.9.2 Initial Installation Method

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

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

5.9.3 Flash management

5.9.3.1 Updating the primary RedBoot image

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

```
-f 0x50000000
-b 0x60000
-l 0x40000
```

5.9.3.2 Updating the secondary RedBoot image

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

```
-f 0x50040000
-b 0x20000
-r 0x20000
-l 0x40000
```

5.9.4 Special RedBoot Commands

None.

5.9.5 Memory Maps

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

NOTE

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

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


Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM Bank 0
0x08000000 - 0x0fffffff  Y Y  SA-1111 Board flash
0x10000000 - 0x17ffffff  N N  Board Registers
0x18000000 - 0x1fffffff  N N  Ethernet
0x20000000 - 0x2fffffff  N N  SA-1111 Board PCMCIA
0x30000000 - 0x3fffffff  N N  Compact Flash
0x40000000 - 0x47ffffff  N N  SA-1111 Board
0x48000000 - 0x4bffffff  N N  GFX
0x50000000 - 0x57ffffff  Y Y  flash
0x80000000 - 0xbfffffff  N N  SA-1110 Internal Registers
0xc0000000 - 0xc1ffffff  N Y  DRAM Bank 0
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
The flash based RedBoot image occupies virtual addresses 0x50000000 - 0x5003ffff.
```

5.9.6 Resource Usage

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

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

5.9.7 Rebuilding RedBoot

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

5.10 NEC uPD985xx ASCOM LAKI Board

5.10.1 Overview

RedBoot uses the single LAKI serial port. The default serial port settings are 38400,8,N,1.

Management of onboard flash is also supported.

Three basic RedBoot configurations are supported:

- RedBoot running from the board's flash boot sector. ("ROM" startup)
- RedBoot running at a higher address in the flash ("POST" startup) - for cooperation with customer POST code or other boot image.
- RedBoot running in RAM ("RAM" startup).

By default ROM startup lives at the base of flash at 0xBFC00000; POST at 0xBFC80000 and RAM applications including RedBoot are loaded at 0x80020000. Suitable default entries are provided by `fis init` as usual.

Configuration fragment ".ecm" files are provided to build RedBoot in each of these startup types.

5.10.2 Initial Installation Method

This assumes that you will install all 3 variants. Should you wish to preserve the customer bootloader behaviour, don't bother with any mention of ROM startup RedBoot below; just install POST and RAM startup. You would then have to use the customer bootloader to start RedBoot if/when you want to use RedBoot and eCos.

The ASCOM boot image supports SRecord downloading over the serial line (as does RedBoot itself). Obtain SRecord versions of each RedBoot image using commands such as **`mips64vr4100el-elf-objcopy -O srec install/bin/redboot.elf /tftpboot/redboot-RAM.srec`**

See below for information about rebuilding RedBoot from sources.

5.10.2.1 Run a RAM-startup RedBoot

Load an SRecord RAM-startup RedBoot using the "SRecord download and execute" option in the ASCOM Boot image menu.

This is the initial splashscreen:

```
ascom PLC Bootloader                                [Ver. 2.0]
Jul 04 2001 - 14:51:39
System Information
    Top Memory address : 0xA07FFFE0
    Top Flash address : 0xBFFFFF80

           **************************************************
           Press any key to enter menu, or wait for countdown
                   to start your default application!
           **************************************************
                                      2
```

So press a key, to get the menu screen. Navigate to start S3 download and execute thus, then press enter:

```
PLC Bootloader [Ver. 2.0] Jul 04 2001 - 14:51:39

        Boot Menu
        =========

         Configuration
         view memory
         jump in memory
         boot from sdram
        [START S3 DOWNLOAD AND EXECUTE]
         start S3 download
         start gdb-stub
         jump at 0xbfc20000 (start Green Hills monserv)

[use : n or N - next, p or P - previous, x, X or enter - valid]
```

The bootloader will show this screen, with a rotating caret as is usual when the download starts:

```
ascom PLC Bootloader                                   [Ver. 2.0]
Jul 04 2001 - 14:51:39

Bootloader starts ...
```

Send the SRecord version of RedBoot down the serial line. On LINUX `cat /tftpboot/redboot-RAM.srec > /dev/ttyS0` for example.

You should now have a running RedBoot.

5.10.2.2 Initialize the Flash

Initialize the flash image system and RedBoot’s flash configuration data.

```
RedBoot> fis init
....
```

Place a valid fconfig record in the flash:

```
RedBoot> fco
....
```

(just change something, say yes to update, then change it back and say yes to update to make it go into the flash)

For more information about these commands, see Section 2.4 and Section 2.5.

5.10.2.3 Install POST RedBoot in flash

Load a POST-startup RedBoot into RAM then save it in flash:

```
RedBoot> load -m y -b 0x80100000
```

Use minicom’s “send” function ^AS with ymodem to send your file eg.`/tftpboot/redboot-POST.srec`

See Section 2.3.3 for details of the load command.

```
RedBoot> fis cre RedBoot[post] -b 0x80100000 -r 0x9fc80000
An image named ’RedBoot[post]’ exists - are you sure (y/n)? y
```

```
* CAUTION * about to program 'RedBoot[post]'
   at 0xbfc80000..0xbfcbffff from 0x80100000 - are you sure (y/n)? y
```

5.10.2.4 Install RedBoot in the boot block

Load a ROM-startup RedBoot into RAM then save it in flash:

```
RedBoot> load -m y -b 0x80100000
```

Use minicom's "send" function ^AS with ymodem to send your file eg.`/tftpboot/redboot-ROM.srec`

See Section 2.3.3 for details of the load command.

```
RedBoot> fis cre RedBoot -b 0x80100000 -r 0x9fc00000
An image named 'RedBoot' exists - are you sure (y/n)? y
* CAUTION * about to program 'RedBoot'
   at 0xbfc00000..0xbfc3ffff from 0x80100000 - are you sure (y/n)? y
```

Reset RedBoot which should run the flash-based RedBoot:

```
RedBoot> reset
....
```

To check that all is well in the fis, you can use the `fis list` command. You should see something like this:

```
RedBoot> fis lis
Name               FLASH addr  Mem addr    Length      Entry point
RedBoot            0xBFC00000  0xBFC00000  0x00040000  0x9FC00000
RedBoot[post]      0xBFC80000  0xBFC80000  0x00040000  0x9FC80000
RedBoot[backup]    0xBFCC0000  0x80020000  0x00040000  0x800200A4
RedBoot config     0xBFFC0000  0xBFFC0000  0x00020000  0x00000000
FIS directory      0xBFFE0000  0xBFFE0000  0x00020000  0x00000000
RedBoot>
```

For more information about the fis, see Section 2.4.

NOTE

If you chose not to load a ROM-startup RedBoot, rather than `reset` above, type `go 0xbfc80000` to jump into the POST-startup RedBoot.

5.10.2.5 Install RAM-based RedBoot in flash

You can now set up RedBoot's networking parameters following the instructions in Section 1.4.1 and then use standard methods to update RedBoot in flash, including the RAM-startup RedBoot image called `RedBoot[backup]`. See the next section and Section 4.1.3 for details.

5.10.3 Flash management

5.10.3.1 Updating the primary ROM RedBoot image

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

```
-f 0xbfc00000
-b 0x80100000
-l    0x40000
-r 0x9fc00000
```

5.10.3.2 Updating the secondary RAM RedBoot image

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

```
-f 0xbfcc0000
-b 0x80100000
-l    0x40000
-r 0x80020000
```

5.10.3.3 Updating the POST RedBoot image

To update the POST RedBoot image, follow the procedures detailed in Section 4.1.3, but use a POST RedBoot image and the actual numbers used with the flags in the sample commands should be (also see above for example):

```
-f 0xbfc80000
-b 0x80100000
-l    0x40000
-r 0x9fc80000
```

5.10.3.4 Starting RedBoot from ASCOM Boot image

If you chose not to load a ROM-startup RedBoot, then you will need to select “Jump in Memory” and type in `bfc80000` (Bee Eff Cee Eight oh oh oh oh, not the default) then “yes” to confirm. RedBoot should start, all happy.

```
PLC Bootloader [Ver. 2.0] Jul 04 2001 - 14:51:39

        Boot Menu
        =========

         Configuration
         view memory
        [JUMP IN MEMORY]
         boot from sdram
         start S3 download and execute
         start S3 download
         start gdb-stub
         jump at 0xbfc20000 (start Green Hills monserv)

[use : n or N - next, p or P - previous, x, X or enter - valid]
```

You will need to do this every time the board is power cycled.

5.10.4 Communication Channels

Serial, Ethernet.

The default serial setup is 38400 8N1.

The ethernet acquires its ESA from the Serial EEPROM that is automatically managed by the uPD985xx MicroWire subsystem.

5.10.5 Special RedBoot Commands

None. But this board is unusual in that it might be fitted with an oversized Intel StrataFlash device. In this case the flash device driver takes special care to use the flash correctly, and reports as such during startup:

```
FLASH: Oversized device!  End addr 0xc0400000 changed to 0xc0000000
FLASH: 0xbfc00000 - 0xc0000000, 32 blocks of 0x00020000 bytes each.
....
```

The CPU is unable to address the upper half of the flash device which would live in addresses `0xc0000000 - 0xc0400000` so it adjusts the description of the flash device to accommodate this limitation.

5.10.6 Memory Maps

This is all dictated by the MIPS architecture of the VR4120 MIPS Core at the heart of the uPD985xx devices. Addresses `0x80000000 - 0x9fffffff` are cachable access to physical address space, and addresses `0xa0000000 - 0xbfffffff` are non-cachable access to physical address space. RedBoot and eCos uses only these two areas. Flash and I/O areas are accessed through the non-cachable area; RAM is accessed (mainly) through the cachable space.

5.10.7 Resource Usage

The flash based RedBoot image occupies flash addresses `0xbfc00000 - 0xbfc3ffff`. It also reserves the first 128K bytes of RAM for runtime uses. The RAM based RedBoot image occupies RAM addresses `0x80020000 - 0x8003ffff`. RAM addresses from 0x40000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

5.10.8 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “laki1”, “mips” and “laki1” respectively. Note that the configuration export files supplied in the `hal/mips/laki1/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.11 Intel SA1110 ASCOM/ACN PLC2 Board

5.11.1 Overview

RedBoot supports the board serial ports attached to UART1 and UART3. Both channels are used for the RedBoot prompt et al, and you can connect using GDB to either.

The default serial port settings are 38400,8,N,1.

RedBoot supports FLASH management on the Plc2. Three basic RedBoot configurations are supported: RedBoot running from the board's FLASH boot sector ("ROM"), RedBoot running at address 0x40000 in the FLASH ("POST") - for cooperation with customer POST code - and normal eCos RAM startup.

5.11.2 Initial Installation Method

A Linux utility "Jflash" is used to program FLASH over a parallel port driven JTAG interface. A special hardware box from ACN is required. After that, RedBoot can be managed and upgraded in the usual manner using its own flash management commands in the usual manner; "fis init" will fill in appropriate defaults for the flash addresses concerned.

5.11.3 Special RedBoot Commands

None.

5.11.4 Memory Maps

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

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x007fffff    8Mb flash (nCS0)
0x18000000 - 0x180fffff    Altera FPGA and X-Bus (nCS3)
0x20000000 - 0x2fffffff    PCMCIA slot 0 - LAN91C96 Ethernet
0x80000000 - 0xbfffffff    SA-1110 Internal Registers
0xc0000000 - 0xc7ffffff    DRAM Bank 0
0xc8000000 - 0xcfffffff    DRAM Bank 1
0xe0000000 - 0xe7ffffff    Cache Clean


Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x001fffff  Y Y  DRAM - 8Mb to 32Mb
0x18000000 - 0x180fffff  N N  Altera FPGA and X-Bus (nCS3)
0x20000000 - 0x2fffffff  N N  PCMCIA slot 0 - LAN91C96 Ethernet
```

```
0x50000000 - 0x507fffff  Y Y  8Mb flash (nCS0)
0x80000000 - 0xbfffffff  N N  SA-1110 Internal Registers
0xc0000000 - 0xc0ffffff  N Y  DRAM Bank 0:8 or 16Mb
0xc8000000 - 0xc8ffffff  N Y  DRAM Bank 1:8 or 16Mb or absent
0xe0000000 - 0xe7ffffff  Y Y  Cache Clean
```

The flash based RedBoot image occupies virtual addresses 0x50000000 - 0x5001ffff.

SDRAM can be any of

1 x 8Mb = 8Mb 2 x 8Mb = 16Mb

1 x 16Mb = 16Mb 2 x 16Mb = 32Mb

All are programmed the same way in the memory controller. Startup code detects which is fitted and programs the memory map accordingly. If the device(s) is 8Mb, then there are gaps in the physical memory map, because a high order address bit is not connected. The gaps are the higher 2Mb out of every 4Mb.

The SA11x0 OS timer is used as a polled timer to provide timeout support within RedBoot.

5.11.5 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “plc2”, “arm” and “sa11x0/plc2” respectively. Note that the configuration export files supplied in the `hal/arm/sa11x0/plc2/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

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

5.12.1 Overview

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

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

5.12.2 Initial Installation

RedBoot is installed using the code download facility built into the Atlas board. See the Atlas User manual for details, and also the Atlas download format in Section 5.12.2.2.

5.12.2.1 Quick download instructions

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

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

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

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

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

   When this is complete, the LED display should show the last address programmed. This will be something like: `1fc17000`.
6. Change switch S5-1 to OFF and reset the board. The LED display should read “RedBoot”.
7. Run the RedBoot `fis init` and `fconfig` commands to initialize the flash. See Section 5.12.3.1, Section 2.4 and Section 2.5 for details.

5.12.2.2 Atlas download format

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

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

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

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

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

The resulting file is named redboot_ROM.fl.

5.12.3 Flash management

5.12.3.1 Additional config options

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

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

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

5.12.3.2 Updating the secondary RedBoot image

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

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

5.12.3.3 Updating the primary RedBoot image

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

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

5.12.4 Additional commands

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

-b ***<addr>***

Location to store command line and environment passed to kernel

-w ***<time>***

Wait time in seconds before starting kernel

-c ***"params"***

Parameters passed to kernel

<addr>

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

Linux kernels on MIPS platforms expect the entry point to be called with arguments in the registers equivalent to a C call with prototype:

```
void Linux(int argc, char **argv, char **envp);
```

RedBoot will place the appropriate data at the offset specified by the *-b* parameter, or by default at address 0x80080000, and will set the arguments accordingly when calling into the kernel.

The default entry point, if no image with explicit entry point has been loaded and none is specified, is 0x80000750.

5.12.5 Interrupts

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

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

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

```
#define CYGNUM_HAL_INTERRUPT_SER                0
#define CYGNUM_HAL_INTERRUPT_TIM0               1
#define CYGNUM_HAL_INTERRUPT_2                  2
#define CYGNUM_HAL_INTERRUPT_3                  3
#define CYGNUM_HAL_INTERRUPT_RTC                4
#define CYGNUM_HAL_INTERRUPT_COREHI             5
#define CYGNUM_HAL_INTERRUPT_CORELO             6
#define CYGNUM_HAL_INTERRUPT_7                  7
#define CYGNUM_HAL_INTERRUPT_PCIA               8
#define CYGNUM_HAL_INTERRUPT_PCIB               9
#define CYGNUM_HAL_INTERRUPT_PCIC              10
#define CYGNUM_HAL_INTERRUPT_PCID              11
#define CYGNUM_HAL_INTERRUPT_ENUM              12
#define CYGNUM_HAL_INTERRUPT_DEG               13
#define CYGNUM_HAL_INTERRUPT_ATXFAIL           14
#define CYGNUM_HAL_INTERRUPT_INTA              15
#define CYGNUM_HAL_INTERRUPT_INTB              16
#define CYGNUM_HAL_INTERRUPT_INTC              17
#define CYGNUM_HAL_INTERRUPT_INTD              18
#define CYGNUM_HAL_INTERRUPT_SERR              19
#define CYGNUM_HAL_INTERRUPT_HW1               20
#define CYGNUM_HAL_INTERRUPT_HW2               21
#define CYGNUM_HAL_INTERRUPT_HW3               22
#define CYGNUM_HAL_INTERRUPT_HW4               23
#define CYGNUM_HAL_INTERRUPT_HW5               24
```

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

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

5.12.6 Memory Maps

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

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

5.12.7 Resource Usage

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

5.12.8 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “atlas_mips32_4kc” or “atlas_mips64_5kc”, “mips” and “atlas” respectively. Note that the configuration export files supplied in the `hal/mips/atlas/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.13 MIPS Malta Board with CoreLV 4Kc and CoreLV 5Kc

5.13.1 Overview

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

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

5.13.2 Initial Installation

RedBoot is installed using the code download facility built into the Malta board. See the Malta User manual for details, and also the Malta download format in Section 5.13.2.2.

5.13.2.1 Quick download instructions

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

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

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

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

   ```
   % cat redboot_ROM.fl >/dev/lp0
   ```

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

5.13.2.2 Malta download format

In order to download RedBoot to the Malta board, it must be converted to the Malta download format.

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

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

The resulting file is named redboot_ROM.fl.

5.13.3 Flash management

5.13.3.1 Updating the secondary RedBoot image

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

```
-f 0xBE020000
-b 0x80020000
-r 0x80020000
-l 0x20000
```

5.13.3.2 Updating the primary RedBoot image

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

```
-f 0xBE000000
-b 0x80080000
-l 0x20000
```

5.13.4 Additional commands

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

-b ***<addr>***

Location to store command line and environment passed to kernel

-w ***<time>***

Wait time in seconds before starting kernel

-c ***"params"***

Parameters passed to kernel

<addr>

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

Linux kernels on MIPS platforms expect the entry point to be called with arguments in the registers equivalent to a C call with prototype:

```
void Linux(int argc, char **argv, char **envp);
```

RedBoot will place the appropriate data at the offset specified by the *-b* parameter, or by default at address 0x80080000, and will set the arguments accordingly when calling into the kernel.

The default entry point, if no image with explicit entry point has been loaded and none is specified, is 0x80000750.

5.13.5 Interrupts

RedBoot uses an interrupt vector table which is located at address 0x80000200. Entries in this table are pointers to functions with this protoype:

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

On the malta board, the vector argument is one of 22 interrupts defined in `hal/mips/malta/`*`VERSION`*`/include/plf_intr.h`:

```
#define CYGNUM_HAL_INTERRUPT_SOUTH_BRIDGE_INTR   0
#define CYGNUM_HAL_INTERRUPT_SOUTH_BRIDGE_SMI    1
#define CYGNUM_HAL_INTERRUPT_CBUS_UART           2
#define CYGNUM_HAL_INTERRUPT_COREHI              3
#define CYGNUM_HAL_INTERRUPT_CORELO              4
#define CYGNUM_HAL_INTERRUPT_COMPARE             5
#define CYGNUM_HAL_INTERRUPT_TIMER               6
#define CYGNUM_HAL_INTERRUPT_KEYBOARD            7
#define CYGNUM_HAL_INTERRUPT_CASCADE             8
#define CYGNUM_HAL_INTERRUPT_TTY1                9
#define CYGNUM_HAL_INTERRUPT_TTY0               10
#define CYGNUM_HAL_INTERRUPT_11                 11
#define CYGNUM_HAL_INTERRUPT_FLOPPY             12
#define CYGNUM_HAL_INTERRUPT_PARALLEL           13
#define CYGNUM_HAL_INTERRUPT_REAL_TIME_CLOCK    14
#define CYGNUM_HAL_INTERRUPT_I2C                15
#define CYGNUM_HAL_INTERRUPT_PCI_AB             16
#define CYGNUM_HAL_INTERRUPT_PCI_CD             17
#define CYGNUM_HAL_INTERRUPT_MOUSE              18
#define CYGNUM_HAL_INTERRUPT_19                 19
#define CYGNUM_HAL_INTERRUPT_IDE_PRIMARY        20
#define CYGNUM_HAL_INTERRUPT_IDE_SECONDARY      21
```

The data passed to the ISR is pulled from a data table (`hal_interrupt_data`) which immediately follows the interrupt vector table. With 22 interrupts, the data table starts at address 0x80000258.

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

5.13.6 Memory Maps

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

NOTE

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

```
Physical Address Range  C B  Description
----------------------- - -  -----------
0x80000000 - 0x81ffffff Y Y  SDRAM
0x9e000000 - 0x9e3fffff Y N  System flash (cached)
```

```
0x9fc00000 - 0x9fffffff Y N  System flash (mirrored)
0xa8000000 - 0xb7ffffff N N  PCI Memory Space
0xb4000000 - 0xb40fffff N N  Galileo System Controller
0xb8000000 - 0xb80fffff N N  Southbridge / ISA
0xb8100000 - 0xbbdfffff N N  PCI I/O Space
0xbe000000 - 0xbe3fffff N N  System flash (noncached)
0xbf000000 - 0xbfffffff N N  Board logic FPGA
```

5.13.7 Resource Usage

The flash based RedBoot image occupies flash addresses 0xbe000000 - 0xbe01ffff. RedBoot also reserves RAM (0x00000000 - 0x0001ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x00020000 - 0x0004ffff. RAM physical addresses from 0x00050000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.13.8 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “malta_mips32_4kc”, “mips” and “malta” respectively. Note that the configuration export files supplied in the `hal/mips/malta/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.14 PMC-Sierra MIPS RM7000 Ocelot

5.14.1 Overview

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

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

5.14.2 Initial Installation Method

Device programmer is used to program socketed flash parts.

5.14.3 Flash Management

5.14.3.1 Updating the primary RedBoot image

To update the primary RedBoot images, follow the procedures detailed in Section 4.1.3, loading the primary image into RAM at 0x80100000. The actual numbers used with the flags in the sample commands are then:

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

5.14.3.2 Updating the secondary RedBoot image

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

```
-f 0xbfc20000
-b 0x80020000
-r 0x80020000
-l 0x20000
```

5.14.4 Additional commands

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

-b ***<addr>***

Location to store command line and environment passed to kernel

-w ***<time>***

Wait time in seconds before starting kernel

-c ***"params"***

Parameters passed to kernel

<addr>

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

Linux kernels on MIPS platforms expect the entry point to be called with arguments in the registers equivalent to a C call with prototype:

```
void Linux(int argc, char **argv, char **envp);
```

RedBoot will place the appropriate data at the offset specified by the *-b* parameter, or by default at address 0x80080000, and will set the arguments accordingly when calling into the kernel.

The default entry point, if no image with explicit entry point has been loaded and none is specified, is 0x80000750.

5.14.5 Memory Maps

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

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

NOTE

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

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

5.14.6 Resource Usage

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

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

5.14.7 Rebuilding RedBoot

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

5.15 Motorola PowerPC MBX

5.15.1 Overview

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

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

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

5.15.2 Initial Installation Method

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

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

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

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

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

1. Prepare EPPC-Bug for download:

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

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

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

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

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

```
RedBoot> fi init
```

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

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

5.15.3 Flash management

5.15.3.1 Updating the primary RedBoot image

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

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

5.15.3.2 Updating the secondary RedBoot image

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

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

5.15.4 Special RedBoot Commands

None.

5.15.5 Memory Maps

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

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

5.15.6 Resource Usage

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

5.15.7 Rebuilding RedBoot

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

5.16 Analogue & Micro PowerPC 860T

5.16.1 Overview

RedBoot uses the SMC1 serial port. The default serial port settings are 38400,8,N,1. Ethernet is also supported using the RJ-45 connector.

Management of onboard flash is also supported. A single RedBoot configuration is supported:

- RedBoot running from RAM using an image copied from the flash boot sector (ROMRAM mode).

5.16.2 Initial Installation Method

RedBoot must be installed at the A & M factory.

5.16.3 Flash management

5.16.3.1 Updating the primary RedBoot image

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

```
-f 0xfe000000
-b 0x50000
-l 0x30000
```

5.16.4 Special RedBoot Commands

None.

5.16.5 Memory Maps

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

```
Physical Address Range Description
----------------------- -----------
0x00000000 - 0x007fffff DRAM
0xfe000000 - 0xfe0fffff flash (AMD29LV8008B)
0xff000000 - 0xff0fffff MPC registers
```

5.16.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0xfe000000 - 0xfe02ffff. RedBoot also reserves RAM (0x00000000 - 0x0003ffff) for RedBoot runtime uses. RAM physical addresses from 0x00040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.16.7 Rebuilding RedBoot

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

5.17 ARM Evaluator7T (e7t) board with ARM7TDMI

5.17.1 Overview

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

5.17.2 Initial Installation

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

5.17.3 Quick download instructions

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

- Boot the board and press ENTER:

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

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

```
Boot: flasherase 01820000 10000
```

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

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

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

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

  When complete, you should see:

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

- Program the flash:

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

- And verify that the module is available:

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

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

5.17.4 Special RedBoot Commands

None.

5.17.5 Memory Maps

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

NOTE

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

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

5.17.6 Resource Usage

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

RedBoot also reserves RAM (0x00000000 - 0x0000ffff) for RedBoot runtime uses.

RAM physical addresses from 0x00010000 to the end of RAM are available for general use.

5.17.7 Rebuilding RedBoot

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

5.18 ARM Integrator board with ARM7TDMI or ARM966E

5.18.1 Overview

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

5.18.2 Initial Installation

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

5.18.3 Quick download instructions

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

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

```
Initialising...


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

- Issue the FLASH ROM load command:

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

Deleting Image 0

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

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

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

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

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

Overwritten block/s
    0
```

```
boot Monitor >
```

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

5.18.4 Special RedBoot Commands

None.

5.18.5 Memory Maps

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

NOTE

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

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

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


ARM966E
-------

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

5.18.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x24000000 - 0x2401ffff.

RedBoot also reserves RAM (0x00000000 - 0x0003ffff) for RedBoot runtime uses. If ethernet support is included, then the address range 0x00f00000 to 0x00ffffff are reserved for use by the driver. This may be moved using the MLT.

RAM physical addresses from 0x00040000 to 0x00efffff and from 0x00100000 to the end of SDRAM are available for general use.

5.18.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “integrator” or “integrator_arm9”, “arm” and “integrator” respectively. Note that the configuration export files supplied in the `hal/arm/integrator/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.19 ARM ARM7 PID, Dev7 and Dev9

5.19.1 Overview

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

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

5.19.2 Initial Installation Method

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

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

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

5.19.3 Special RedBoot Commands

None.

5.19.4 Memory Maps

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

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

5.19.5 Resource Usage

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

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

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

5.19.6 Rebuilding RedBoot

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

5.20 Compaq iPAQ PocketPC

5.20.1 Overview

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

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

5.20.2 Initial Installation

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

5.20.2.1 Installing RedBoot on the iPAQ using Windows/CE

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

NOTE

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

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

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

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

- Using ActiveSync, copy the file OSloader to your iPAQ.
- Using ActiveSync, copy the file redboot_WinCE.bin to the iPAQ as bootldr in its root directory. Note: this is not the top level folder displayed by Windows (Mobile Device), but rather the ’My Pocket PC’ folder within it.

- Execute OSloader. If you didn't create a shortcut, then you will have to poke around for it using the WinCE file explorer.
- Choose the **Tools->BootLdr->Run after loading from file** menu item.

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

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

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

5.20.2.3 Setting up and testing RedBoot

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

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

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

```
RedBoot(tm) bootstrap and debug environment [ROM]
Red Hat certified release, version R1.xx - built 06:17:41, Mar 19 2001
Platform: Compaq iPAQ Pocket PC (StrongARM 1110)

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

RAM: 0x00000000-0x01fc0000, 0x0001f200-0x01f70000 available
FLASH: 0x50000000 - 0x51000000, 64 blocks of 0x00040000 bytes each.
```

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

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

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

5.20.2.4 Installing RedBoot permanently

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

CAUTION

This step will destroy your Windows/CE environment.

Before you take this step, it is strongly recommended you save your WinCE FLASH contents as outlined above using the "parrot" loader, or by using the Compaq OSloader:

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

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

```
RedBoot> lo -r -b 0x100000 /tftpboot/redboot_ROM.bin
RedBoot> fi loc -f 0x50000000 -l 0x40000
RedBoot> fis init
RedBoot> fi unl -f 0x50040000 -l 0x40000
RedBoot> fi cr RedBoot -b 0x100000
RedBoot> fi loc -f 0x50040000 -l 0x40000
RedBoot> reset
```

WARNING

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

5.20.2.5 Restoring Windows/CE

To restore Windows/CE from the backup taken in Section 5.20.2.4, visit http://www.handhelds.org/projects/wincerestoration.html for directions.

5.20.3 Flash Management

5.20.3.1 Updating the secondary RedBoot image

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

```
-f 0x50080000
-b 0x00020000
-r 0x00020000
-e 0x00020040
-l 0x40000
```

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

```
fis unlock -f 0x50080000 -l 0x40000
```

and

```
fis lock -f 0x50080000 -l 0x40000
```

5.20.3.2 Updating the primary RedBoot image

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

```
-f 0x50040000
-b 0x00100000
-l 0x40000
```

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

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

and

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

5.20.4 Additional commands

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

-b ***<addr>***

Location Linux kernel was loaded to

-l ***<len>***

Length of kernel

-c ***"params"***

Parameters passed to kernel

-r ***<addr>***

'initrd' ramdisk location

-s ***<len>***

Length of initrd ramdisk

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

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

5.20.5 Memory Maps

RedBoot sets up the following memory map on the iPAQ: The first level page table is located at physical address 0xC0004000. No second level tables are used.

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x01ffffff    16Mb to 32Mb FLASH (nCS0) [organized as below]
  0x000000 - 0x0003ffff      Parrot Loader
  0x040000 - 0x0007ffff      RedBoot
  0xf80000 - 0x00fbffff      Fconfig data
  0xfc0000 - 0x00ffffff      FIS directory
0x30000000 - 0x3fffffff    Compact Flash
0x48000000 - 0x4bffffff    iPAQ internal registers
0x80000000 - 0xbfffffff    SA-1110 Internal Registers
0xc0000000 - 0xc1ffffff    DRAM Bank 0 - 32Mb SDRAM
0xe0000000 - 0xe7ffffff    Cache Clean


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

5.20.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x50040000 - 0x5007ffff. RedBoot also reserves RAM (0x00000000 - 0x0001ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at virtual addresses 0x00020000 - 0x0005ffff. RAM virtual addresses from 0x00060000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash. An exception is RAM from 0x01F70000 - 0x01FFFFFF which is reserved for use by the LCD display.

5.20.7 Rebuilding RedBoot

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

5.21 Cirrus Logic EP7xxx (EDB7211, EDB7212, EDB7312)

5.21.1 Overview

RedBoot supports both serial ports on the board and the ethernet port. The default serial port settings are 38400,8,N,1. RedBoot also supports flash management on the EDB7xxx for the NOR flash only. Two basic RedBoot configurations are supported:

- RedBoot running from the board’s flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.
- EDB7312 only: RedBoot running from RAM copied directly from the flash boot sector.

5.21.2 Initial Installation Method

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

5.21.3 Flash management

5.21.3.1 Updating the primary RedBoot image

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

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

5.21.3.2 Updating the secondary RedBoot image

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

```
-f 0xE0040000
-b 0x40000
-r 0x40000
-l 0x40000
```

NOTE

> On the EDB7312, because the primary RedBoot image runs in RAM and not FLASH, it can be updated directly without use of the separate RAM based version.

5.21.4 Special RedBoot Commands

None.

5.21.5 Memory Maps

The MMU page tables and LCD display buffer, if enabled, are located at the end of DRAM.

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x01ffffff    NOR Flash (EDB7211, EDB7212)
0x00000000 - 0x00ffffff    NOR Flash (EDB7312)
0x10000000 - 0x11ffffff    NAND Flash
0x20000000 - 0x2fffffff    Expansion 2
0x30000000 - 0x3fffffff    Expansion 3
0x40000000 - 0x4fffffff    PCMCIA 0
0x50000000 - 0x5fffffff    PCMCIA 1
0x60000000 - 0x600007ff    On-chip SRAM
0x80000000 - 0x8fffffff    I/O registers
0xc0000000 - 0xc1ffffff    DRAM (EDB7211, EDB7212)
0xc0000000 - 0xc0ffffff    DRAM (EDB7312)

Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM
0x00000000 - 0x00fcffff  Y Y  DRAM (EDB7312)
0x20000000 - 0x2fffffff  N N  Expansion 2
0x30000000 - 0x3fffffff  N N  Expansion 3
0x40000000 - 0x4fffffff  N N  PCMCIA 0
0x50000000 - 0x5fffffff  N N  PCMCIA 1
0x60000000 - 0x600007ff  Y Y  On-chip SRAM
0x80000000 - 0x8fffffff  N N  I/O registers
0xc0000000 - 0xc001ffff  N Y  LCD buffer (if configured)
0xe0000000 - 0xe1ffffff  Y Y  NOR Flash (EDB7211, EDB7212)
0xe0000000 - 0xe0ffffff  Y Y  NOR Flash (EDB7312)
0xf0000000 - 0xf1ffffff  Y Y  NAND Flash

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

5.21.6 Resource Usage

The RAM based RedBoot image occupies RAM addresses `0x40000 - 0x7ffff`. The ROMRAM based RedBoot image (EDB7312 only) occupies RAM addresses `0x1000 - 0x3ffff`. RAM addresses start at `0x80000` and continue up to the top of the installed physical RAM size, less the memory reserved for MMU page tables (0x9000 bytes) and the LCD display buffer, if enabled (0x20000 bytes). The RAM is available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

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

5.21.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for ARCH_DIR and PLATFORM_DIR on this platform are “arm” and “edb7xxx” respectively. The value for TARGET is either “edb7211” or “edb7212” or “edb7312”, depending on the desired platform. Note that the configuration export files supplied in the `hal/arm/edb7xxx/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used, and the correct edb7XXX variant chosen.

5.22 Bright Star Engineering commEngine and nanoEngine

5.22.1 Overview

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

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

5.22.2 Initial Installation

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

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

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

5.22.3 Download Instructions

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

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

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

NOTE

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

After a reset, the BSE firmware will print

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

and then RedBoot starts, switching to 38400 Baud.

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

5.22.4 Cohabiting with POST in Flash

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

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

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

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

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

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

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

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

5.22.5 Special RedBoot Commands

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

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

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

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

5.22.6 Memory Maps

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

NOTE

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

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

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

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

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

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

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

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

All are programmed the same in the memory controller.

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

5.22.7 Nano Platform Port

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

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

respectively.

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

5.22.8 Ethernet Driver

The ethernet driver is in two parts:

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

The platform-specific ether driver is `devs/eth/arm/nano`. Its package is `CYGPKG_DEVS_ETH_ARM_NANO`. This tells the generic driver the address in IO memory of the chip, for example, and other configuration details. This driver picks up the ESA from RedBoot's configuration data - unless configured to use a static ESA in the usual manner.

5.22.9 Rebuilding RedBoot

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

5.23 x86 Based PC

5.23.1 Overview

RedBoot supports two serial ports and an Intel i82559 based ethernet card (for example an Intel EtherExpress Pro 10/100) for communication and downloads. The default serial port settings are 38400,8,N,1. RedBoot runs from a boot floppy disk installed in the A: drive of the PC.

5.23.2 Initial Installation

RedBoot takes the form of a self-booting image that must be written onto a formatted floppy disk. The process will erase any file system or data that already exists on that disk, so proceed with caution.

For Red Hat Linux users, this can be done by:

```
$ dd conv=sync if=install/bin/redboot.bin of=/dev/fd0H1440
```

For NT Cygwin users, this can be done by first ensuring that the raw floppy device is mounted as `/dev/fd0`. To check if this is the case, type the command **`mount`** at the Cygwin bash prompt. If the floppy drive is already mounted, it will be listed as something similar to the following line:

```
\\.\a: /dev/fd0 user binmode
```

If this line is not listed, then mount the floppy drive using the command:

```
$ mount -f -b //./a: /dev/fd0
```

To actually install the boot image on the floppy, use the command:

```
$ dd conv=sync if=install/bin/redboot.bin of=/dev/fd0
```

Insert this floppy in the A: drive of the PC to be used as a target and ensure that the BIOS is configured to boot from A: by default. On reset, the PC will boot from the floppy and be ready to be debugged via either serial line, or via the ethernet interface if it is installed.

NOTE

Unreliable floppy media may cause the write to silently fail. This can be determined if the RedBoot image does not correctly boot. In such cases, the floppy should be (unconditionally) reformatted using the `fdformat` command on Linux, or `format a:  /u` on DOS/Windows.

5.23.3 Flash management

PC RedBoot does not support any FLASH commands.

5.23.4 Special RedBoot Commands

None.

5.23.5 Memory Maps

All selectors are initialized to map the entire 32-bit address space in the familiar protected mode flat model. Page translation is not used. RAM up to 640K is mapped to 0x0 to 0xa0000. RAM above 640K is mapped from address 0x100000 upwards. Space is reserved between 0xa0000 and 0x100000 for option ROMs and the BIOS.

5.23.6 Resource Usage

RedBoot is loaded into RAM at address 0x2000 and reserves all RAM below 0xa0000 for its own use. RAM applications should load from address 0x100000 upwards.

5.23.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “pc”, “i386” and “pc” respectively. Note that the configuration export files supplied in the `hal/i386/pc/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used. In particular, the `redboot_FLOPPY.ecm` file is used for building a version of RedBoot suitable for booting off a floppy disk.

5.24 Samsung CalmRISC16 Core Evaluation Board

5.24.1 Overview

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

Only a ROM startup RedBoot configuration is supported.

5.24.2 Initial Installation Method

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

```
.../packages/hal/calmrisc16/ceb/current/support
```

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

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

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

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

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

5.24.3 Special RedBoot Commands

None.

5.24.4 Special Note on Serial Channel

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

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

5.24.5 Resource Usage

The RedBoot image occupies program addresses 0x000000 - 0x00ffff and data addresses 0x000000 - 0x00ffff.

5.24.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “calm16_ceb”, “calmrisc16” and “ceb” respectively. Note that the configuration export files supplied in the `hal/calmrisc16/ceb/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.25 Samsung CalmRISC32 Core Evaluation Board

5.25.1 Overview

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

Only a ROM startup RedBoot configuration is supported.

5.25.2 Initial Installation Method

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

```
.../packages/hal/calmrisc32/ceb/current/support
```

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

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

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

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

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

5.25.3 Special RedBoot Commands

None.

5.25.4 Special Note on Serial Channel

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

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

5.25.5 Resource Usage

The RedBoot image occupies program addresses 0x00000000 - 0x0000ffff and data addresses 0x00000000 - 0x0000ffff.

5.25.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “calm32_ceb”, “calmrisc32” and “ceb” respectively. Note that the configuration export files supplied in the `hal/calmrisc32/ceb/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.26 Hitachi EDK7708 (edk7708)

5.26.1 Overview

RedBoot uses the serial port. The default serial port settings are 38400,8,N,1.

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

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

5.26.2 Initial Installation Method

Program the ROM RedBoot image into flash using an eprom programmer.

5.26.3 Flash management

5.26.3.1 Updating the primary RedBoot image

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

```
-f 0x80000000
-b 0x88040000
-l 0x20000
```

5.26.4 Memory Maps

RedBoot sets up the following memory map on the EDK7708 board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x8001ffff Flash (AT29LV1024)
0x88000000 - 0x881fffff DRAM
0xa4000000 - 0xa40000ff LED ON
0xb8000000 - 0xb80000ff LED ON
```

5.26.5 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x88000000 - 0x8800ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x88010000 - 0x8803ffff. RAM physical addresses from 0x88040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.26.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are "edk7708", "sh" and "edk7708" respectively. Note that the configuration export files supplied in the `hal/sh/edk7708/VERSION/misc` directory in the RedBoot source tree should be used.

5.27 Hitachi Solution Engine 77X9 (SE77X9)

5.27.1 Overview

This description covers the MS7729SE01 and MS7709SSE0101 variants. See Section 5.28 for instructions for the MS7709SE01 variant.

RedBoot uses the COM1 and COM2 serial ports. The default serial port settings are 38400,8,N,1. Ethernet is also supported using the 10-base T connector.

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

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

5.27.2 Initial Installation Method

The Solution Engine ships with the Hitachi boot monitor in EPROM which allows for initial programming of RedBoot:

1. Set switches SW4-3 and SW4-4 to ON [boot from EPROM]
2. Connect a serial cable to COM2 and power up the board.
3. After the boot monitor banner, invoke the flash download/program command:

   ```
   Ready >fl
   ```

4. The monitor should now ask for input:

   ```
   Flash ROM data copy to RAM
   Please Send A S-format Record
   ```

   At this point copy the RedBoot ROM SREC file to the serial port:

   ```
   $ cat redboot_ROM.eprom.srec > /dev/ttyS0
   ```

   Eventually you should see something like

   ```
   Start Addrs = A1000000
   End Addrs = A1xxxxxx
   Transfer complete
   ```

   from the monitor.

5. Set switch SW4-3 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner.

5.27.3 Flash management

5.27.3.1 Updating the primary RedBoot image

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

```
-f 0x80000000
-b 0x8c080000
-l 0x20000
```

5.27.3.2 Updating the secondary RedBoot image

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

```
-f 0x80020000
-b 0x8c020000
-r 0x8c020000
-l 0x20000
```

5.27.4 Special RedBoot Commands

The `exec` command which allows the loading and execution of Linux kernels is supported for this board (see Section 2.6). The `exec` parameters used for the SE77x9 are:

-b ***<addr>***

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i ***<addr>***

Start address of initrd image

-j ***<size>***

Size of initrd image

-c ***"args"***

Kernel arguments string

-m ***<flags>***

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f ***<flags>***

RAM disk flags. Should normally be 0x4000

-r ***<device number>***

Root device specification. /dev/ram is 0x0101

-l ***<type>***

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

On the SE77x9, Linux expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000. This is configurable in the kernel using the CONFIG_MEMORY_START option.

5.27.5 Memory Maps

RedBoot sets up the following memory map on the SE77x9 board.

```
Physical Address Range  Description
----------------------- -----------
```

```
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x81000000 - 0x813fffff EPROM (M27C800)
0x8c000000 - 0x8dffffff SDRAM
0xb0000000 - 0xb03fffff Ethernet (DP83902A)
0xb0400000 - 0xb07fffff SuperIO (FDC37C935A)
0xb0800000 - 0xb0bfffff Switches
0xb0c00000 - 0xbfffffff LEDs
0xb1800000 - 0xb1bfffff PCMCIA (MaruBun)
```

5.27.6 Ethernet Driver

The ethernet driver uses a hardwired ESA which can, at present, only be changed in CDL.

5.27.7 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x8c000000 - 0x8c01ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x8c020000 - 0x8c07ffff. RAM physical addresses from 0x8c080000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.27.8 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “se77x9”, “sh” and “se77x9” respectively. Note that the configuration export files supplied in the `hal/sh/se77x9/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.28 Hitachi Solution Engine 7709 (SE77X9)

5.28.1 Overview

This description covers the MS7709SE01 variant. See Section 5.27 for instructions for the MS7729SE01 and MS7709SSE0101 variants.

RedBoot uses the COM1 and COM2 serial ports. The default serial port settings are 38400,8,N,1. Ethernet is also supported using the 10-base T connector.

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

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

5.28.2 Initial Installation Method

The Solution Engine ships with the Hitachi boot monitor in EPROM which allows for initial programming of RedBoot:

1. Set switch SW4-1 to ON [boot from EPROM]
2. Connect a serial cable to CN1 (SCI) and power up the board.
3. After the boot monitor banner, invoke the flash download/program command:

   ```
   Ready >fl
   ```

4. The monitor should now ask for input:

   ```
   Flash ROM data copy to RAM
   Please Send A S-format Record
   ```

   At this point copy the RedBoot ROM SREC file to the serial port:

   ```
   $ cat redboot_SE7709RP_ROM.eprom.srec > /dev/ttyS0
   ```

   Eventually you should see something like

   ```
   Start Addrs = A1000000
   End Addrs = A1xxxxxx
   Transfer complete
   ```

   from the monitor.

5. Set switch SW4-1 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner.

5.28.3 Flash management

5.28.3.1 Updating the primary RedBoot image

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

```
-f 0x80000000
-b 0x8c080000
-l 0x20000
```

5.28.3.2 Updating the secondary RedBoot image

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

```
-f 0x80020000
-b 0x8c020000
-r 0x8c020000
-l 0x20000
```

5.28.4 Special RedBoot Commands

The `exec` command which allows the loading and execution of Linux kernels is supported for this board (see Section 2.6). The `exec` parameters used for the SE77x9 are:

-b ***<addr>***

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i ***<addr>***

Start address of initrd image

-j ***<size>***

Size of initrd image

-c ***"args"***

Kernel arguments string

-m ***<flags>***

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f ***<flags>***

RAM disk flags. Should normally be 0x4000

-r ***<device number>***

Root device specification. /dev/ram is 0x0101

-l ***<type>***

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

For the the SE77x9, Linux by default expects to be loaded at 0x8c001000 which conflicts with the data space used by RedBoot. To work around this, either change the CONFIG_MEMORY_START kernel option to a higher address, or use the compressed kernel image and load it at a higher address. For example, setting CONFIG_MEMORY_START to 0x8c100000, the kernel expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000.

5.28.5 Memory Maps

RedBoot sets up the following memory map on the SE77x9 board.

```
Physical Address Range  Description
----------------------- -----------
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x81000000 - 0x813fffff EPROM (M27C800)
0x8c000000 - 0x8dffffff DRAM
0xb0000000 - 0xb03fffff Ethernet (DP83902A)
0xb0800000 - 0xb08fffff 16C552A
0xb1000000 - 0xb100ffff Switches
0xb1800000 - 0xb18fffff LEDs
0xb8000000 - 0xbbffffff PCMCIA (MaruBun)
```

5.28.6 Ethernet Driver

The ethernet driver uses a hardwired ESA which can, at present, only be changed in CDL.

5.28.7 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x8c000000 - 0x8c01ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x8c020000 - 0x8c07ffff. RAM physical addresses from 0x8c080000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.28.8 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “se77x9”, “sh” and “se77x9” respectively. Note that the configuration export files (containing SE7709RP substring) supplied in the `hal/sh/se77x9/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.29 Hitachi Solution Engine 7751 (SE7751)

5.29.1 Overview

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

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

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

5.29.2 Initial Installation Method

The Solution Engine ships with the Hitachi boot monitor in EPROM which allows for initial programming of RedBoot:

1. Set switches SW5-3 and SW5-4 to ON [boot from EPROM]
2. Connect a serial cable to COM1 and power up the board.
3. After the boot monitor banner, invoke the flash download/program command:

   ```
   Ready >fl
   ```

4. The monitor should now ask for input:

   ```
   Flash ROM data copy to RAM
   Please Send A S-format Record
   ```

   At this point copy the RedBoot ROM SREC file to the serial port:

   ```
   $ cat redboot_ROM.eprom.srec > /dev/ttyS0
   ```

   Eventually you should see something like

   ```
   Start Addrs = A1000000
   End Addrs = A1xxxxxx
   Transfer complete
   ```

   from the monitor.

5. Set switch SW5-3 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner.

5.29.3 Flash management

5.29.3.1 Updating the primary RedBoot image

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

```
-f 0x80000000
-b 0x8c080000
-l 0x20000
```

5.29.3.2 Updating the secondary RedBoot image

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

```
-f 0x80020000
-b 0x8c020000
-r 0x8c020000
-l 0x20000
```

5.29.4 Special RedBoot Commands

The `exec` command which allows the loading and execution of Linux kernels is supported for this board (see Section 2.6). The `exec` parameters used for the SE7751 are:

-b ***<addr>***

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i ***<addr>***

Start address of initrd image

-j ***<size>***

Size of initrd image

-c ***"args"***

Kernel arguments string

-m ***<flags>***

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f ***<flags>***

RAM disk flags. Should normally be 0x4000

-r ***<device number>***

Root device specification. /dev/ram is 0x0101

-l ***<type>***

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

On the SE7751, Linux expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000. This is configurable in the kernel using the CONFIG_MEMORY_START option.

5.29.5 Memory Maps

RedBoot sets up the following memory map on the SE7751 board.

```
Physical Address Range  Description
----------------------- -----------
```

```
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x81000000 - 0x813fffff EPROM (M27C800)
0x8c000000 - 0x8fffffff SDRAM
0xb8000000 - 0xb8ffffff PCMCIA (MaruBun)
0xb9000000 - 0xb9ffffff Switches
0xba000000 - 0xbaffffff LEDs
0xbd000000 - 0xbdffffff PCI MEM space
0xbe200000 - 0xbe23ffff PCI Ctrl space
0xbe240000 - 0xbe27ffff PCI IO space
```

5.29.6 Ethernet Driver

The ethernet driver uses a hardwired ESA which can, at present, only be changed in CDL.

5.29.7 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x8c000000 - 0x8c01ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x8c020000 - 0x8c07ffff. RAM physical addresses from 0x8c080000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.29.8 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “se7751”, “sh” and “se7751” respectively. Note that the configuration export files supplied in the `hal/sh/se7751/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.30 Hitachi HS7729PCI

5.30.1 Overview

RedBoot uses the COM1 and COM2 serial ports (and the debug port on the motherboard). The default serial port settings are 38400,8,N,1. Ethernet is also supported using a D-Link DFE-530TX PCI plugin card.

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

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

5.30.2 Initial Installation Method

A copy of the ROM startup version of RedBoot must be programmed into the two EPROMs. Two files with a split version of the ROM image is provided: it is also possible to recreate these from the redboot.bin file, but requires the split_word.c program in hal/sh/hs7729pci/*VERSION*/misc to be built and executed with the redboot.bin filename as sole argument.

After doing this it is advised that another ROM version of RedBoot is programmed into the flash, and that copy be used for booting the board. This allows for software programmed updates of RedBoot instead of having to reprogram the EPROMs.

1. Program the EPROMs with RedBoot. The .lo image should go in socket M1 and the .hi image in socket M2.
2. Set switch SW1-6 to ON [boot from EPROM]
3. Follow the instructions under Flash management for updating the flash copy of RedBoot, but use

   ```
   -f 0x80400000
   ```

   due to setting of the SW1-6 switch.
4. Set switch SW1-6 to OFF [boot from flash] and reboot the board. You should now see the RedBoot banner. At this time you may want to issue the command

   ```
   fis init
   ```

   to initialize the flash table with the correct addresses.

5.30.3 Flash management

5.30.3.1 Updating the primary RedBoot image

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

```
-f 0x80000000
-b 0x8c080000
-l 0x20000
```

5.30.3.2 Updating the secondary RedBoot image

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

```
-f 0x80020000
-b 0x8c020000
-r 0x8c020000
-l 0x20000
```

5.30.4 Special RedBoot Commands

The `exec` command which allows the loading and execution of Linux kernels is supported for this board (see Section 2.6). The `exec` parameters used for the HS7729PCI are:

-b ***<addr>***

Parameter block address. This is normally the first page of the kernel image and defaults to 0x8c101000

-i ***<addr>***

Start address of initrd image

-j ***<size>***

Size of initrd image

-c ***"args"***

Kernel arguments string

-m ***<flags>***

Mount rdonly flags. If set to a non-zero value the root partition will be mounted read-only.

-f ***<flags>***

RAM disk flags. Should normally be 0x4000

-r ***<device number>***

Root device specification. /dev/ram is 0x0101

-l ***<type>***

Loader type

Finally the kernel entry address can be specified as an optional argument. The default is 0x8c102000

On the HS7729PCI, Linux expects to be loaded at address 0x8c101000 with the entry point at 0x8c102000. This is configurable in the kernel using the CONFIG_MEMORY_START option.

5.30.5 Memory Maps

RedBoot sets up the following memory map on the HS7729PCI board.

```
Physical Address Range  Description
----------------------- -----------
```

```
0x80000000 - 0x803fffff Flash (MBM29LV160)
0x80400000 - 0x807fffff EPROM (M27C800)
0x82000000 - 0x82ffffff SRAM
0x89000000 - 0x89ffffff SRAM
0x8c000000 - 0x8fffffff SDRAM
0xa8000000 - 0xa800ffff SuperIO (FDC37C935A)
0xa8400000 - 0xa87fffff USB function (ML60851C)
0xa8800000 - 0xa8bfffff USB host (SL11HT)
0xa8c00000 - 0xa8c3ffff Switches
0xa8c40000 - 0xa8c7ffff LEDs
0xa8c80000 - 0xa8cfffff Interrupt controller
0xb0000000 - 0xb3ffffff PCI (SD0001)
0xb8000000 - 0xbbffffff PCMCIA (MaruBun)
```

5.30.6 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x8c000000 - 0x8c01ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x8c020000 - 0x8c07ffff. RAM physical addresses from 0x8c080000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.30.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “hs7729pci”, “sh” and “hs7729pci” respectively. Note that the configuration export files supplied in the `hal/sh/hs7729pci/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.31 Cirrus Logic SEB9312 (EP93xx) (aka SEB9213)

5.31.1 Overview

RedBoot supports the serial port labelled ’modem inteface’ on the board and the ethernet port. The default serial port settings are 38400,8,N,1. RedBoot also supports flash management on the SEB9312. Two basic RedBoot configurations are supported:

- RedBoot running from RAM, but contained in the board’s flash boot sector (ROMRAM mode).
- RedBoot running standalone from RAM.

NOTE

The cache is currently disabled as this feature does not seem to work properly with the beta silicon; it interacts badly with flash management activities and network usage.

5.31.2 Initial Installation Method

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

5.31.3 Flash management

5.31.3.1 Updating the primary RedBoot image

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

```
-f 0x24000000
-b 0x00100000
-l 0x20000
```

Note that since the primary RedBoot image copies itself to RAM and runs from RAM (“ROMRAM” startup), it is not necessary to run a separate RAM-only RedBoot during this update process; you can use the primary RedBoot image to update the primary RedBoot image in situ.

5.31.3.2 Updating the secondary RedBoot image

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

```
-f 0x24020000
-b 0x40000
-r 0x40000
-l 0x20000
-e 0x40040
```

5.31.4 Special RedBoot Commands

No special commands.

One special `fconfig` option is provided to control the ethernet station address (ESA or 'MAC address').

```
RedBoot> fco -l
....
Local IP address: 10.16.19.10
Default server IP address: 10.16.19.66
Network hardware address [MAC]: 0x00:0x12:0x34:0x56:0x78:0xAB
...
RedBoot> fco ep93xx_esa
ep93xx_esa: 0x00:0x12:0x34:0x56:0x78:0xAB
```

5.31.5 Memory Maps

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

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x01ffffff    DRAM
0x24000000 - 0x247fffff    Flash
0x28000000 - 0x280fffff    SRAM
0x80000000 - 0x8fffffff    I/O registers

Virtual Address Range    C B  Description
-----------------------  - -  ----------------------------------
0x00000000 - 0x01ffffff  Y Y  DRAM
0x24000000 - 0x247fffff  Y Y  Flash
0x28000000 - 0x2800ffff  Y Y  On-chip SRAM
0x80000000 - 0x8fffffff  N N  I/O registers
0xC0000000 - 0xC1ffffff  N N  Non-cachable access to DRAM

The flash based RedBoot image occupies virtual addresses 0x24000000 - 0x2401ffff.
```

5.31.6 Resource Usage

The ROMRAM startup RedBoot image occupies RAM addresses 0x08000 - 0x3ffff. RAM addresses from 0x40000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

The RAM based RedBoot image occupies RAM addresses 0x40000 - 0x68000. RAM addresses from 0x68000 to the end of RAM are available for general use.

In either case, memory from 0x0 - 0x07fff is reserved for system vectors and the VM translation tables.

5.31.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for ARCH_DIR and PLATFORM_DIR on this platform are "arm" and "arm9/ep93xx" respectively.

The value for TARGET is “seb9312”. Note that the configuration export files supplied in the `hal/arm/arm9/ep93xx/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used. Use only the ROMRAM and RAM type configuration export files.

5.32 Atmel AT91 Evaluation Board (EB40)

5.32.1 Overview

RedBoot supports both serial ports. The default serial port settings are 38400,8,N,1. RedBoot also supports minimal flash management on the EB40. However, since the flash device (AT29LV1024) is so small (only the upper 64K is available for general use), only ’fconfig’ is supported, along with the simple flash write command, ’fis write’. Two basic RedBoot configurations are supported:

- RedBoot running from RAM, but contained in the board’s flash boot sector (ROMRAM mode).
- RedBoot running from RAM with RedBoot in the flash boot sector.

The RAM version is only used during the initialization of RedBoot the first time.

5.32.2 Initial Installation Method

This development board comes with ARM’s debug tool, Angel, installed in flash. At this time, Angel will not be replaced. Rather, RedBoot will be placed in the alternate half of flash. Switch SW1 is used which monitor to boot. Selecting SW1 to "lower mem" will choose Angel. Select SW1 to "Upper mem" for RedBoot once it has been installed.

Set SW1 to "lower mem" and connect serial port A to a host computer. Using GDB from the host and Angel on the board, download the RAM based version of RedBoot to the board. Once this is started, the Angel session must be interrupted (on Linux this can be done using ^Z). Follow this by connecting to the board using minicom at 38400-8N1. At this point, RedBoot will be running on the board in RAM. Now, download the ROMRAM version and program it to flash. Be sure and first set SW1 to "upper mem".

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

At this point, interrupt the Angel session and start minicom.

```
RedBoot> ve

RedBoot(tm) bootstrap and debug environment [RAM]
Red Hat certified release, version R1.xx - built 14:09:27, Jul 20 2001

Platform: Atmel AT91/EB40 (ARM7TDMI)
Copyright (C) 2000, 2001, Red Hat, Inc.
```

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

RedBoot> load -m ymodem -b 0x02040000
```

Use minicom to send the file redboot_ROMRAM.srec via YModem.

```
RedBoot> fi wr -f 0x01010000 -b 0x02040000 -l 0xe000
```

Set switch SW1 to "upper mem", press the "reset" pushbutton and RedBoot should come up on the board.

5.32.3 Flash management

5.32.3.1 Updating the RedBoot image in flash

Since the primary RedBoot runs from RAM, it can be used to update itself directly. Simply follow the steps above, starting with a connection to RedBoot running on the board.

5.32.4 Special RedBoot Commands

None.

5.32.5 Memory Maps

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

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

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

5.32.6 Resource Usage

The RAM based RedBoot image occupies RAM addresses 0x02020000 - 0x0203ffff. RAM addresses from 0x02040000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

5.32.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for PLATFORM_DIR on this platform is “at91”. The value for TARGET is “eb40”. Note that the configuration export files supplied in the `hal/arm/at91/VERSION/misc` directory in the RedBoot source tree should be used. The ROMRAM configuration should be used to build the RedBoot image to be programmed into flash. The RAM configuration is used for the initial download of RedBoot.

5.33 Matsushita MN103E010 (AM33/2.0) ASB2303 Board

5.33.1 Overview

RedBoot supports both serial ports for communication and downloads. The default serial port settings are 115200,8,N,1. RedBoot can run from either flash, and can support flash management for either the boot PROM or the system flash regions. These configurations are supported:

- RedBoot running from the boot PROM and able to access the system flash (the redboot_ROM.bin image should be used for this).
- RedBoot running from the system flash and able to access the boot PROM (the redboot_FLASH.bin image should be used for this).
- RedBoot running from RAM and able to access the boot PROM (the redboot_RAM.bin image should be used for this).
- JTAG loadable RedBoot running from RAM and able to access the boot PROM (the redboot_FULLRAM.bin image should be used for this).

5.33.2 Initial Installation

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

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

5.33.2.1 Preparing to program the board

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

- Jumper across ST10 on the board to allow write access to the boot PROM.
- Set the switch ST6 (on the front of the board) to boot from whichever flash is *not* being programmed. Note that the RedBoot image cannot access the flash from which it is currently executing (it can only access the other flash).

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

5.33.2.2 Preparing to use the JTAG debugger

The JTAG debugger will also need setting up:

1. Install the JTAG debugger software (WICE103E) on a PC running Windows (WinNT is probably the best choice for this) in “C:/PanaX”.
2. Install the Matsushita provided “project” into the “C:/Panax/wice103e/prj” directory.

3. Install the RedBoot image files into the “C:/Panax/wice103e/prj” directory under the names redboot.ram and redboot.prom.
4. Make sure the PC’s BIOS has the parallel port set to full bidirectional mode.
5. Connect the JTAG debugger to the PC’s parallel port.
6. Connect the JTAG debugger to the board.
7. Set the switch on the front of the board to boot from “boot PROM”.
8. Power up the JTAG debugger and then power up the board.
9. Connect the board’s Debug Serial port to a computer by a null modem cable.
10. Start minicom or some other serial communication software and set for 115200 baud, 1-N-8 with no flow control.

5.33.2.3 Loading the RAM-based RedBoot via JTAG

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

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

```
ed 0xc0002000, 0x12040580

ed 0xd8c00100, 0x8400fe01
ed 0xd8c00200, 0x21111000
ed 0xd8c00204, 0x00100200
ed 0xd8c00208, 0x00000004
ed 0xd8c00110, 0x8000fc01
ed 0xd8c00210, 0x21111000
ed 0xd8c00214, 0x00100200
ed 0xd8c00218, 0x04000004
ed 0xd8c00120, 0x8600fff1
ed 0xd8c00220, 0x21111000
ed 0xd8c00224, 0x00100200
ed 0xd8c00228, 0x00000004

ed 0xda000000,0x55561645
ed 0xda000004,0x00000c30
ed 0xda000008,0x9000fe01
ed 0xda00000c,0x9200fe01
ed 0xda000000,0xa89b0654
```

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

```
ex 0x80000000,0x81ffffff,/mexram
ex 0x84000000,0x85ffffff,/mexram
ex 0x86000000,0x867fffff,/mexram
ex 0x90000000,0x93ffffff,/mexram
```

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

```
_pc=90000000
u_pc
rd redboot_FULLRAM.bin,90000000
```

5. Load the boot PROM RedBoot into RAM:

```
rd redboot_ROM.bin,90100000
```

6. Start RedBoot in RAM:

```
g
```

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

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

1. Instruct RedBoot to initialise the boot PROM:

```
fis init
```

2. Write the previously loaded redboot_ROM.bin image into the boot PROM:

```
fis create RedBoot -b 0x90100000
```

3. Check that RedBoot has written the image:

```
dump -b 0x90100000
dump -b 0x80000000
```

   Other than the difference in address, the two dumps should be the same.

4. Close the JTAG software and power-cycle the board. The RedBoot banners should be displayed again over the serial port, followed by the RedBoot prompt. The boot PROM-based RedBoot will now be running.
5. Run the following command to initialise the system flash:

```
fi init
```

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

```
load -r -b 0x90100000 -m ymodem
```

   Use the terminal software to download redboot_FLASH.bin using YMODEM protocol.

```
fis create RedBoot -b 0x90100000
```

   NOTE

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

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

```
load -r -b 0x90100000 -m ymodem
```

   Use the terminal software to download redboot_ROM.bin using YMODEM protocol.

```
fis cre RedBoot -b 0x90100000
```

   See Section 2.5 for details on configuring the RedBoot in general, and also Section 2.4 for more details on programming the system flash.

5.33.3 Additional Commands

The `exec` command which allows the loading and execution of Linux kernels, is supported for this architecture (see Section 2.6). The `exec` parameters used for ASB2303 board are:

-w ***<time>***

Wait time in seconds before starting kernel

-c ***"params"***

Parameters passed to kernel

<addr>

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

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

5.33.4 Memory Maps

RedBoot sets up the following memory map on the ASB2303 board.

NOTE

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

```
Physical Address Range   Description
-----------------------  -----------
0x80000000 - 0x9FFFFFFF  Cached Region
0x80000000 - 0x81FFFFFF  Boot PROM
0x84000000 - 0x85FFFFFF  System Flash
0x86000000 - 0x867FFFFF  8KB Configuration ROM
0x8C000000 - 0x8FFFFFFF  On-Chip Memory (repeated 16Kb SRAM)
0x90000000 - 0x93FFFFFF  SDRAM
0xDB000008               7-segment LED
```

The ASB2303 HAL makes use of the on-chip memory in the following way:

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

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

5.33.5 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x90000000 - 0x9001ffff) for RedBoot runtime uses. FULLRAM based RedBoot configurations are designed to run from RAM at physical addresses 0x90000000 - 0x9001ffff. RAM based RedBoot configurations are designed to run from RAM at physical addresses

0x90020000 - 0x9003ffff. RAM physical addresses from 0x90040000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

NOTE

The location at which RedBoot can be started is highly restricted due to the way in which the address of the Trap Vector Table is specified to the CPU.

5.33.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “asb”, “mn10300” and “asb” respectively. Note that the configuration export files supplied in the `hal/mn10300/asb/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.34 Matsushita MN103E010 (AM33/2.0) ASB2305 Board

5.34.1 Overview

RedBoot supports the debug serial port and the built in ethernet port for communication and downloads. The default serial port settings are 115200,8,N,1 with no flow control. RedBoot can run from either flash, and can support flash management for either the boot PROM or the system flash regions. These configurations are supported:

- RedBoot running from the boot PROM and able to access the system flash (the redboot_ROM.bin image should be used for this).
- RedBoot running from the system flash and able to access the boot PROM (the redboot_FLASH.bin image should be used for this).
- RedBoot running from RAM and able to access the boot PROM (the redboot_RAM.bin image should be used for this).

5.34.2 Initial Installation

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

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

5.34.2.1 Preparing to program the board

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

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

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

5.34.2.2 Preparing to use the JTAG debugger

The JTAG debugger will also need setting up:

1. Install the JTAG debugger software (WICE103E) on a PC running Windows (WinNT is probably the best choice for this) in “C:/PanaX”.
2. Install the Matsushita provided “project” into the “C:/Panax/wice103e/prj” directory.

3. Install the RedBoot image files into the “C:/Panax/wice103e/prj” directory under the names redboot.ram and redboot.prom.
4. Make sure the PC’s BIOS has the parallel port set to full bidirectional mode.
5. Connect the JTAG debugger to the PC’s parallel port.
6. Connect the JTAG debugger to the board.
7. Set the switch on the front of the board to boot from “boot PROM”.
8. Power up the JTAG debugger and then power up the board.
9. Connect the board’s Debug Serial port to a computer by a null modem cable.
10. Start minicom or some other serial communication software and set for 115200 baud, 1-N-8 with no flow control.

5.34.2.3 Loading the RAM-based RedBoot via JTAG

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

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

```
ed 0xc0002000, 0x12000580

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

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

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

ed 0xd8c00130, 0x8680ff81
ed 0xd8c00230, 0x21111000
ed 0xd8c00234, 0x00100200
ed 0xd8c00238, 0x00000004

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

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

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

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

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

   ```
   _pc=90000000
   u_pc
   rd redboot_FULLRAM.bin,90000000
   ```

5. Load the boot PROM RedBoot into RAM:

   ```
   rd redboot_ROM,90100000
   ```

6. Start RedBoot in RAM:

   ```
   g
   ```

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

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

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

1. Instruct RedBoot to initialise the boot PROM:

   ```
   fis init
   ```

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

   ```
   fis create RedBoot -b 0x90100000
   ```

3. Check that RedBoot has written the image:

   ```
   dump -b 0x90100000
   dump -b 0x80000000
   ```

   Other than the difference in address, the two dumps should be the same.

4. Close the JTAG software and power-cycle the board. The RedBoot banners should be displayed again over the serial port, followed by the RedBoot prompt. The boot PROM-based RedBoot will now be running.
5. Power off the board and unjumper ST18 to write-protect the contents of the boot PROM. Then power the board back up.
6. Run the following command to initialise the system flash:

   ```
   fi init
   ```

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

   ```
   load -r -b 0x90100000 redboot_FLASH.bin
   fis create RedBoot -b 0x90100000
   ```

NOTE

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

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

```
load -r -b 0x90100000 redboot_ROM.bin
fis create RedBoot -b 0x90100000
```

   See Section 2.5 for details on configuring the RedBoot in general, and also Section 2.4 for more details on programming the system flash.

5.34.3 Additional Commands

The `exec` command which allows the loading and execution of Linux kernels, is supported for this architecture (see Section 2.6). The `exec` parameters used for ASB2305 board are:

-w ***<time>***

Wait time in seconds before starting kernel

-c ***"params"***

Parameters passed to kernel

<addr>

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

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

5.34.4 Memory Maps

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

NOTE

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

Physical Address Range	Description
0x80000000 - 0x9FFFFFFF	Cached Region
0x80000000 - 0x81FFFFFF	Boot PROM
0x84000000 - 0x85FFFFFF	System Flash
0x86000000 - 0x86007FFF	64Kbit Sys Config EEPROM
0x86F90000 - 0x86F90003	4x 7-segment LEDs
0x86FA0000 - 0x86FA0003	Software DIP Switches
0x86FB0000 - 0x86FB001F	PC16550 Debug Serial Port
0x8C000000 - 0x8FFFFFFF	On-Chip Memory (repeated 16Kb SRAM)
0x90000000 - 0x93FFFFFF	SDRAM

```
0x98000000 - 0x9BFFFFFF  Paged PCI Memory Space (64Mb)
0x9C000000 - 0x9DFFFFFF  PCI Local SRAM (32Mb)
0x9E000000 - 0x9E03FFFF  PCI I/O Space
0x9E040000 - 0x9E0400FF  AM33-PCI Bridge Registers
0x9FFFFFF4 - 0x9FFFFFF7  PCI Memory Page Register
0x9FFFFFF8 - 0x9FFFFFFF  PCI Config Registers
0xA0000000 - 0xBFFFFFFF  Uncached Mirror Region
0xC0000000 - 0xDFFFFFFF  CPU Control Registers
```

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

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

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

5.34.5 Resource Usage

The flash based RedBoot image occupies flash addresses 0x80000000 - 0x8001ffff. RedBoot also reserves RAM (0x90000000 - 0x9001ffff) for RedBoot runtime uses. RAM based RedBoot configurations are designed to run from RAM at physical addresses 0x90000000 - 0x9001ffff. RAM physical addresses from 0x90050000 to the end of RAM are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

NOTE

The location at which RedBoot can be started is highly restricted due to the way in which the address of the Trap Vector Table is specified to the CPU.

5.34.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are "asb2305", "mn10300" and "asb2305" respectively. Note that the configuration export files supplied in the `hal/mn10300/asb2305/VERSION/misc` directory in the RedBoot source tree should be used.

5.35 Altera Excalibur ARM9 (excalibur_arm9)

5.35.1 Overview

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

- RedBoot running from the board's flash boot sector (ROM).
- RedBoot running from RAM with RedBoot in the flash boot sector (ROMRAM/REDBOOT).

NOTE

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

5.35.2 Initial Installation Method

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

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

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

5.35.3 Flash management

5.35.3.1 Updating the primary RedBoot image

To update the primary RedBoot image (of startup type ROMRAM or REDBOOT), follow the procedures detailed in Section 4.1.3, but the actual numbers used with the flags in the sample commands should be:

```
-f 0x40000000
-b 0x40000
-l 0x20000
```

When updating the image, the flash should be unlocked before programming, and relocked afterwards (a board reset will also cause the flash to be relocked). This is done with the commands:

```
fis unlock -f 0x40000000 -l 0x20000
```

and

```
fis lock -f 0x40000000 -l 0x20000
```

NOTE

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

NOTE

Since the default RedBoot flash image is of startup type ROMRAM or REDBOOT, it is not necessary to use the two-step update process with the RAM startup type of RedBoot; it is possible to update the image in flash directly.

5.35.4 Special RedBoot Commands

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

-b *<addr>*

Location Linux kernel was loaded to

-l *<len>*

Length of kernel

-c *"params"*

Parameters passed to kernel

-r *<addr>*

’initrd’ ramdisk location

-s *<len>*

Length of initrd ramdisk

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

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

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

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

5.35.5 Memory Maps

The MMU page tables are located at 0x4000.

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x07ffffff    SDRAM
0x08000000 - 0x0805ffff    On-chip SRAM
0x40000000 - 0x40ffffff    Flash
0x7fffc000 - 0x7fffffff    I/O registers
0x80000000 - 0x8001ffff    PLD

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

The flash based RedBoot image occupies virtual addresses 0x40000000 - 0x4001ffff.
```

5.35.6 Resource Usage

The ROMRAM startup type RedBoot image occupies RAM addresses 0x00000 - 0x3ffff. RAM addresses from 0x40000 to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

The REDBOOT startup type RedBoot image occupies RAM addresses 0x07f80000 - 0x07ffffff. RAM addresses from 0x8000 to about 0x07f00000 are available for general use such as a temporary scratchpad for downloaded images before they are written to flash.

5.35.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for ARCH_DIR and PLATFORM_DIR on this platform are “arm” and “arm9/excalibur” respectively. The value for TARGET is “excalibur_arm9”. Note that the configuration export files supplied in the `hal/arm/arm9/excalibur/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.36 Agilent AAED2000 ARM9 (aaed)

5.36.1 Overview

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

- RedBoot being automatically executed by the on-board ARM Boot Monitor, running from RAM.
- RedBoot running from RAM with RedBoot in the flash boot sector.

5.36.2 Initial Installation Method

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

Presently, only the former method is supported.

5.36.2.1 RedBoot as Primary Bootmonitor

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

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

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

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

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

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

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

You should see RedBoot start up:

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

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

Platform: AAED2000 system (ARM9) [Primary]
Copyright (C) 2000, 2001, Red Hat, Inc.

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

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

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

Download the ROMRAM version of RedBoot via ethernet:

```
RedBoot> load -b 0x100000 redboot_primary_ROMRAM/redboot.srec
```

or using serial Y-modem protocol:

```
RedBoot> load -mode ymodem -b 0x100000
```

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

```
Address offset = 0x00ff8000
Entry point: 0x00008040, address range: 0x00008000-0x0002da80
RedBoot> fi cr RedBoot -b 0x100000
An image named 'RedBoot' exists - are you sure (y/n)? y
* CAUTION * about to program 'RedBoot'
            at 0x60000000..0x6003ffff from 0x00100000 - are you sure (y/n)? y
... Erase from 0x60000000-0x60040000: ..
... Program from 0x00100000-0x00140000 at 0x60000000: ..
... Erase from 0x61fe0000-0x62000000: .
... Program from 0x01f5f000-0x01f7f000 at 0x61fe0000: .
```

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

5.36.3 Flash management

5.36.3.1 Updating the RedBoot image

Since the RedBoot image is a ROMRAM-startup type, it is not necessary to load and run the RAM startup RedBoot image to update the image in flash. Doing so causes no harm though - but ignoring the step saves some time. For the same reason, there has not been reserved space in the flash for a "RedBoot[backup]" image.

To update the primary RedBoot image, follow the procedures detailed in Section 4.1.3, but let RedBoot find the correct flash address (the -f option) since this is different between the bootmonitor configurations. If specifying it, be sure to get it right - the actual numbers used with the flags in the sample commands should for a RedBoot image configured to be the primary bootmonitor be:

```
-f 0x60000000
-b 0x100000
-l 0x40000
```

5.36.4 Special RedBoot Commands

The `exec` command which allows the loading and execution of Linux kernels, is supported for this board (see Section 2.6). The `exec` parameters used for the AAED2000 are:

-b ***<addr>***

Location Linux kernel was loaded to

-l ***<len>***

Length of kernel

-c ***"params"***

Parameters passed to kernel

-r ***<addr>***

’initrd’ ramdisk location

-s ***<len>***

Length of initrd ramdisk

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

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

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

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

5.36.5 Memory Maps

The MMU page tables are located at 0x4000.

NOTE

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x01ffffff    Flash
0x10000000 - 0x100fffff    Ethernet
0x30000000 - 0x300fffff    Board registers
0x40000000 - 0x4fffffff    PCMCIA Slot (0)
0x50000000 - 0x5fffffff    Compact Flash Slot (1)
0x80000000 - 0x800037ff    I/O registers
0xb0060000 - 0xb00fffff    On-chip SRAM
0xf0000000 - 0xfd3fffff    SDRAM

Virtual Address Range     C B  Description
-----------------------   - -  ----------------------------------
0x00000000 - 0x01f7ffff   Y Y  SDRAM
0x01f80000 - 0x01ffffff   Y Y  SDRAM (used for LCD frame buffer)
0x10000000 - 0x100fffff   N N  Ethernet
0x30000000 - 0x300fffff   N N  Board registers
0x40000000 - 0x4fffffff   N N  PCMCIA Slot (0)
0x50000000 - 0x5fffffff   N N  Compact Flash Slot (1)
0x60000000 - 0x61ffffff   N N  Flash
0x80000000 - 0x800037ff   N N  I/O registers
0xf0000000 - 0xffffffff   N N  SDRAM (uncached)
```

5.36.6 Resource Usage

The RAM based RedBoot image occupies RAM addresses `0x40000 - 0x7ffff`. The flash based RedBoot image occupies RAM addresses `0x00000000 - 0x0003ffff`. RAM addresses from `0x80000` to the end of RAM are available for general use such as a temporary scratchpad for downloaded images before they are written to flash. If configured to use the LCD screen, additional DRAM from `0x01f80000 - 0x01ffffff` is used for the LCD frame buffer.

5.36.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for ARCH_DIR and PLATFORM_DIR on this platform are “arm” and “arm9/aaed2000” respectively. The value for TARGET is “aaed”. Note that the configuration export files supplied in the `hal/arm/arm9/aaed2000/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used.

5.37 NEC DDB-VRC4375

5.37.1 Overview

RedBoot supports only serial port 1, which is connected to the upper of the stacked serial connectors on the board. The default serial port settings are 38400,8,N,1. FLASH management is also supported. Two basic RedBoot configurations are supported:

- RedBoot running from RAM which has been relocated from the board’s flash boot sector.
- RedBoot running from RAM with RedBoot in the flash boot sector.

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

5.37.2 Initial Installation Method

A device programmer should be used to program a socketed FLASH part (AMD 29F040). The board as delivered is configured for a 512K EPROM. To install a FLASH ROM, Jumpers J30, J31 and J36 need to be changed as described in the board’s User Manual.

Since RedBoot for this board relocates itself from ROM to RAM at startup, it is not necessary to run a secondary RAM based version of RedBoot to update the main FLASH image. Instead this can be done from the primary version of RedBoot.

5.37.2.1 Updating the primary RedBoot image

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

```
-b 0x80100000
```

Flash locking and unlocking is not required. Note that these values are inferred when updating the RedBoot image once the `fis create` has been run.

5.37.3 Special RedBoot Commands

None.

5.37.4 Memory Maps

RedBoot sets up the memory map primarily as described in the board’s User Manual. There are some minor differences, noted in the following table:

Physical Addresses	Virtual Addresses	Resource
00000000-01FFFFFF	80000000-81FFFFFF	Base SDRAM (cached)
00000000-01FFFFFF	A0000000-A1FFFFFF	Base SDRAM (uncached)
0C000000-0C0BFFFF	AC000000-AC0B0000	PCI IO space
0F000000-0F0001FF	AF000000-AF0001FF	VRC4375 Registers
1C000000-1C0FFFFF	BC000000-BC0FFFFF	VRC4372 Registers
1C100000-1DFFFFFF	BC100000-BDFFFFFF	PCI Memory space

```
1FC00000-1FC7FFFF       BFC00000-BFC7FFFF       FLASH ROM
80000000-8000000D       C0000000-C000000D       RTC
8000000E-80007FFF       C000000E-C0007FFF       NVRAM
81000000-81FFFFFF       C1000000-C1FFFFFF       Z85C30 DUART
82000000-82FFFFFF       C2000000-C2FFFFFF       Z8536 Timer
83000000-83FFFFFF       C3000000-C3FFFFFF       8255 Parallel port
87000000-87FFFFFF       C7000000-C7FFFFFF       Seven segment display
```

NOTE

By default the VRC4375 SIMM control registers are not programmed since the values used must depend on the SIMMs installed. If SIMMs are to be used, correct values must be placed in these registers before accessing the SIMM address range.

NOTE

The allocation of address ranges to devices in the PCI IO and memory spaces is handled by the eCos PCI support library. They do not correspond to those described in the board User Manual.

NOTE

The MMU has been set up to relocate the VRC4372 supported devices mapped at physical addresses 0x8xxxxxxx to virtual addresses 0xCxxxxxxx.

5.37.5 Resource Usage

The RedBoot image occupies flash addresses 0x1fc00000 - 0x1fc1ffff. To execute it copies itself out of there to RAM at 0x80000000. RedBoot reserves 1MB of RAM from 0x80000000 to 0x800FFFFF for its own use. The top 1MB of RAM from 0x81F00000 to 0x81FFFFFF is reserved for use by the PCI Ethernet device. RAM based RedBoot configurations are designed to run from RAM at virtual addresses 0x80100000 - 0x8011ffff. RAM virtual addresses from 0x80020000 to the start of the PCI window are available for general use, such as a temporary scratchpad for downloaded images, before they are written to flash.

5.37.6 Ethernet Driver

The ethernet driver is in two parts:

A generic ether driver for the Intel i21143 device is located in `devs/eth/intel/i21143`. Its package name is `CYGPKG_DEVS_ETH_INTEL_I21143`.

The platform-specific ether driver is `devs/eth/mips/vrc4375`. Its package is `CYGPKG_DEVS_ETH_MIPS_VRC4375`. This tells the generic driver the address in IO memory of the chip, for example, and other configuration details. The ESA (MAC address) is by default collected from on-board serial EEPROM, unless configured statically within this package.

5.37.7 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “vrc4375”, “mips” and “vrc4375” respectively. The configuration export files supplied in the `hal/mips/vrc4375/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used. In general only the ROMRAM variant should need to be used.

5.38 Fujitsu FR-V Design Kit (MB93091-CBxx)

5.38.1 Overview

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

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

- RedBoot running from RAM which has been relocated from the board's flash boot sector. This mode is known as ROMRAM.
- RedBoot running from RAM, loaded by some other means.

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

5.38.2 Initial Installation Method

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

5.38.2.1 Updating the primary RedBoot image

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

```
-f 0xFF000000
-b 0x100000
-l 0x40000
```

Note that these values are inferred when updating the RedBoot image once the `fis create` has been run.

5.38.3 Special RedBoot Commands

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

5.38.4 Memory Maps

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

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

NOTE

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

5.38.5 Resource Usage

The RedBoot image occupies flash addresses 0xFF000000 - 0xFF03FFFF. To execute it copies itself out of there to RAM at 0x03FC0000. RedBoot reserves memory from 0x00000000 to 0x0001FFFF for its own use. User programs can use memory from 0x00020000 to 0x03FBFFFF. RAM based RedBoot configurations are designed to run from RAM at 0x00020000.

5.38.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “mb93091”, “frv” and “mb93091” respectively. The configuration export files supplied in the `hal/frv/mb93091/`*`VERSION`*`/misc` directory in the RedBoot source tree should be used. In general only the ROMRAM variant should need to be used.

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

5.39.1 Overview

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

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

- RedBoot running from RAM which has been relocated from the board's flash boot sector. This mode is known as ROMRAM.
- RedBoot running from RAM, loaded by some other means.

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

5.39.2 Initial Installation Method

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

5.39.2.1 Updating the primary RedBoot image

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

```
-f 0xFF000000
-b 0x100000
-l 0x40000
```

Note that these values are inferred when updating the RedBoot image once the `fis create` has been run.

5.39.3 Special RedBoot Commands

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

5.39.4 Memory Maps

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

Address	Cache?	Resource
00000000-03EFFFFF	Yes	SDRAM (via plugin DIMM)
03F00000-03FFFFFF	No	Unused (SDRAM)
10000000-1FFFFFFF	No	AX88796 Ethernet

```
20000000-2FFFFFFF          No         System FPGA
30000000-3FFFFFFF          No         MB93493 companion chip (unused)
40000000-FCFFFFFF          ??         Unused
FD000000-FDFFFFFF          ??         FLASH (ROM3,ROM4) (unused)
FE000000-FEFFFFFF          No         Miscellaneous on-chip I/O
FF000000-FFFFFFFF          No         RedBoot FLASH (16MiB)
```

5.39.5 Resource Usage

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

5.39.6 Rebuilding RedBoot

The instructions in Chapter 3, *Rebuilding RedBoot* should be followed. The values for TARGET, ARCH_DIR and PLATFORM_DIR on this platform are “mb93093”, “frv” and “mb93093” respectively. The configuration export files supplied in the `hal/frv/mb93093/VERSION/misc` directory in the RedBoot source tree should be used. In general only the ROMRAM variant should need to be used.

Index