RedBoot User’s Guide

RedBoot User's Guide

Documentation licensing terms

Trademarks

Warranty

Table of Contents

List of Examples

Chapter 1. Getting Started with RedBoot

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

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

Here are some highlights of RedBoot's capabilities:

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

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

More information about RedBoot on the web

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

Installing RedBoot

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

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

User Interface

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

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

RedBoot Editing Commands

RedBoot uses the following line editing commands.

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

- Delete (0x7F) or Backspace (0x08) erases the character to the left of the cursor.
- ^A 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.

RedBoot Command History

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

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

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

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

RedBoot Startup Mode

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

ROM mode

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

ROMRAM mode

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

RAM mode

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

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

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

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

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

```
RedBoot>version

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

RedBoot Resource Usage

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

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

Flash Resources

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

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

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

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

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

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

RAM Resources

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

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

```
RedBoot> version

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

Platform: FooBar (SH 7615)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

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

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

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

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

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

Configuring the RedBoot Environment

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

Target Network Configuration

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

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

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

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

```
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.29
Default server IP address: 192.168.1.101
DNS server IP address: 192.168.1.1
GDB connection port: 9000
Network debug at boot time: false
```

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

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

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

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

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

Host Network Configuration

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

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

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

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

Enable TFTP on Red Hat Linux 6.2

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

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

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

   ```
   # service inet reload
   ```

Enable TFTP on Red Hat Linux 7 (or newer)

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

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

   Reload the xinetd configuration using the command:

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

   Create the directory /tftpboot using the command

   ```
   mkdir /tftpboot
   ```

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

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

Enable BOOTP/DHCP server on Red Hat Linux

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

Example 1-1. Sample DHCP configuration file

```
--------------- /etc/dhcpd.conf -----------------------------
default-lease-time 600;
max-lease-time 7200;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option domain-name-servers 198.41.0.4, 128.9.0.107;
option domain-name “bogus.com”;
allow bootp;
shared-network BOGUS {
subnet 192.168.1.0 netmask 255.255.255.0 {
         option routers 192.168.1.101;
         range 192.168.1.1 192.168.1.254;
}
 }
```

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

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

```
# service dhcpd start
```

Enable DNS server on Red Hat Linux

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

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

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

options {
        directory "/var/named";
        /*
         * If there is a firewall between you and nameservers you want
         * to talk to, you might need to uncomment the query-source
         * directive below.  Previous versions of BIND always asked
         * questions using port 53, but BIND 8.1 uses an unprivileged
         * port by default.
         */
        // query-source address * port 53;


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

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

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

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

Make sure the server is started with the command:

```
# service named start
```

and is started on next reboot with the command

```
# chkconfig named on
```

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

RedBoot network gateway

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

Verification

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

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

Chapter 2. RedBoot Commands and Examples

Introduction

RedBoot provides three basic classes of commands:

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

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

The basic format for commands is:

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

Commands may require additional information beyond the basic command name. In most cases this additional information is optional, with suitable default values provided if they are not present.

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

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

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

```
RedBoot> help
Manage aliases kept in FLASH memory
      alias name [value]
Set/Query the system console baud rate
      baudrate [-b <rate>]
Manage machine caches
      cache [ON | OFF]
Display/switch console channel
      channel [-1|<channel number>]
Compute a 32bit checksum [POSIX algorithm] for a range of memory
      cksum -b <location> -l <length>
Dump cpsr
      cpsr
Dump dcsr
      dcsr [-h] Hot Debug on;  [-c] off
      [-t] Trace     on;  [-n] off
Display (hex dump) a range of memory
      dump -b <location> [-l <length>] [-s] [-1|2|4]
Force an ECC failure
      eccf -b <base_address>
Execute an image - with MMU off
      exec [-w timeout] [-b <load addr> [-l <length>]]
           [-r <ramdisk addr> [-s <ramdisk length>]]
           [-c "kernel command line"] [<entry_point>]
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]
Run board diagnostics
      diag
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] [-h <host>] [-m <varies>] [-c <channel_number>]
           [-b <base_address>] <file_name>
Compare two blocks of memory
      mcmp -s <location> -d <location> -l <length> [-1|-2|-4]
Fill a block of memory with a pattern
      mfill -b <location> -l <length> -p <pattern> [-1|-2|-4]
Issue a PCI Config Read
      pcr -b <bus> -d <device> -f <function> -o <offset> [-l <length>] [-1|-2|-4]
Issue a PCI Config Write
      pcw -b <bus> -d <device> -f <function> -o <offset> -p <pattern> [-l <length>]
Network connectivity test
      ping [-v] [-n <count>] [-l <length>] [-t <timeout>] [-r <rate>]
           [-i <IP_addr>] -h <IP_addr>
Issue a PCI I/O Read
      pir -b <base address> [-l <length>] [-1|-2|-4]
Issue a PCI I/O Write
      piw -b <base address> -p <pattern> [-l <length>] [-1|-2|-4]
Issue a PCI Memory Read
      pmr [-u <upper base address>] -b <base address> [-l <length>] [-1|-2|-4]
Issue a PCI Memory Write
      pmw [-u <upper base address>] -b <base address> -p <pattern> [-l <length>] [-1|-2|-4]
Reset the system
      reset
Dump SDRAM SPD Contents
      spd
Write to SDRAM SPD
      spdw -a <address> -p <pattern>
Display RedBoot version information
      version
Display (hex dump) a range of memory
      x -b <location> [-l <length>] [-s] [-1|2|4]
```

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

There is one additional, special command. When RedBoot detects '$' or '+' (unless escaped via '\') in a command, it switches to GDB protocol mode. At this point, the eCos GDB stubs take over, allowing connections from a GDB host. The only way to get back to RedBoot from GDB mode is to restart the platform.

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

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

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

Common Commands

alias

Name

`alias` — Manipulate command line aliases

Synopsis

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

Arguments

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

Description

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

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

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

Examples

Set an alias.

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

Display an alias.

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

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

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

Aliases can be nested.

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

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

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

baudrate

Name

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

Synopsis

baudrate [-b *rate*]

Arguments

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

Description

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

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

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

Examples

Show the current baud rate.

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

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

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

baudrate

```
Update RedBoot non-volatile configuration - continue (y/n)? n
```

cache

Name

`cache` — Control hardware caches

Synopsis

cache [on | off]

Arguments

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

Description

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

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

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

Examples

Show the current cache state.

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

Disable the caches.

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

Enable the caches.

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

channel

Name

`channel` — Select the system console channel

Synopsis

channel [-1 | *`channel_number`*]

Arguments

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

Description

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

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

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

Examples

Show the current channel.

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

Change to an invalid channel.

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

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

```
RedBoot> **channel -1**
```

cksum

Name

`cksum` — Compute POSIX checksums

Synopsis

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

Arguments

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

Description

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

Examples

Checksum a buffer.

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

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

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

cpsr

Name

`cpsr` — Dump the current CPSR value

Synopsis

cpsr

Description

Dumps the current value of the CPSR.

Examples

Dump the current CPSR.

```
RedBoot> cpsr
cpsr = 0x200000d3
```

dcsr

Name

`dcsr` — Dump/modify the current DCSR

Synopsis

dcsr [-h *`Hot Debug on`*] [-c *`Hot Debug off`*] [-t *`Trace on`*] [-n *`Trace off`*]

Description

Displays and/or modifies the current value of the DCSR.

Examples

Dump the current DCSR.

```
RedBoot> dcsr
dcsr = 0x8000001c
```

diag

Name

`diag` — Execute board-specific diagnostics

Synopsis

diag

Description

The **diag** command is used to execute board-specific diagnostics.

diag

dump

Name

`dump` — Display memory.

Synopsis

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

Arguments

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

Description

Display a range of memory on the system console.

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

Note that this command could be detrimental if used on memory mapped hardware registers.

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

Examples

Display a buffer, one byte at a time.

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

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

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

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

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

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

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

Display a buffer, with visible ASCII strings.

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

eccf

Name

`eccf` — Force an ECC failure.

Synopsis

eccf {-b *`location`*}

Arguments

Name	Type	Description	Default
-b *`location`*	Memory address	Location in memory for ecc failure.	*none*

Description

Force and ECC failure at a given location.

Note that this command is used to test the ECC error interrupt processing in RedBoot

Examples

Force an ecc error at address 0x01000000.

```
RedBoot> eccf -b 0x01000000
ELOG0 = 0x00100001, ELOG1 = 0x00000000
ECAR0 = 0x01000000, ECAR1 = 0x00000000
MCISR = 0x00000000
```

help

Name

`help` — Display help on available commands

Synopsis

help [*topic*]

Arguments

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

Description

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

If the command has sub-commands, e.g. **fis**, then the topic specific display will print additional information about the available sub-commands. special (ICMP) packets to a specific host. These packets should be automatically returned by that host. The command will indicate how many of these round-trips were successfully completed.

Examples

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

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

```
Execute code at a location
   go [-w <timeout>] [entry]
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] [-h <host>] [-m {TFTP | HTTP | {x|y}MODEM -c <channel_number>}]
        [-b <base_address>] <file_name>
Compare two blocks of memory
   mcmp -s <location> -d <location> -l <length> [-1|-2|-4]
Fill a block of memory with a pattern
   mfill -b <location> -l <length> -p <pattern> [-1|-2|-4]
Network connectivity test
   ping [-v] [-n <count>] [-l <length>] [-t <timeout>] [-r <rate>]
        [-i <IP_addr>] -h <IP_addr>
Reset the system
   reset
Display RedBoot version information
   version
Display (hex dump) a range of memory
   x -b <location> [-l <length>] [-s] [-1|2|4]
```

Help about a command with sub-commands.

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

ip_address

Name

`ip_address` — Set IP addresses

Synopsis

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

Arguments

Name	Type	Description	Default
-l *`local_IP_address`*	Numeric IP or DNS name	The IP address RedBoot should use.	*none*
-h *`server_IP_address`*	Numeric IP or DNS name	The IP address of the default server. Use of this address is implied by other commands, such as **load**.	*none*
-d *`DNS_server_IP_address`*	Numeric IP or DNS name	The IP address of the DNS server.	*none*

Description

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

The `-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.

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

Examples

Display the current network settings.

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

Change the DNS server address.

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

Change the default server address.

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

load

Name

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

Synopsis

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

Arguments

Name	Type	Description	Default
-v	Boolean	Display a small spinner (indicator) while the download is in progress. This is just for feedback, especially during long loads. Note that the option has no effect when using a serial download method since it would interfere with the protocol.	*quiet*
-d	Boolean	Decompress data stream (gzip data)	*non-compressed data*
-r	Boolean	Raw (or binary) data	*formatted (S-records, ELF image, etc)*
-m tftp		Transfer data via the network using TFTP protocol.	TFTP
-m http		Transfer data via the network using HTTP protocol.	TFTP
-m xmodem		Transfer data using *X-modem* protocol.	TFTP
-m ymodem		Transfer data using *Y-modem* protocol.	TFTP
-m disk		Transfer data from a local disk.	TFTP
-h *`server_IP_address`*	Numeric IP or DNS name	The IP address of the TFTP or HTTP server.	Value set by **ip_address**

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

Description

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

Examples

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

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

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

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

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

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

load

mcmp

Name

`mcmp` — Compare two segments of memory

Synopsis

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

Arguments

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

Description

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

Examples

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

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

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

```
RedBoot> mcmp -s 0x100000 -d 0x200000 -l 0x30 -2
Buffers don’t match - 0x00100020=0x6000, 0x00200020=0x0000
```

mfill

Name

`mfill` — Fill RAM with a specified pattern

Synopsis

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

Arguments

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

Description

Fills a range of memory with the given pattern.

Examples

Fill a buffer with zeros.

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

Fill a buffer with a pattern.

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

pcr

Name

`pcr` — Issue a PCI configuration read.

Synopsis

pcr {-b *`bus`*} {-d *`device`*} {-f *`function`*} {-o *`offset`*} [-l *`length`*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`bus`*	Bus Number	Bus Number to be used for configuration transaction.	*none*
-d *`device`*	Device Number	Device Number to be used for configuration transaction.	*none*
-f *`function`*	Function Number	Function Number to be used for configuration transaction.	*none*
-o *`offset`*	Offset	Offset to be read from.	*none*
-l *`length`*	Length	Number of bytes to be read.	*4*
-1		Access one byte (8 bits) at a time.	-1
-2		Access two bytes (16 bits) at a time.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Issue a PCI configuration read transaction to a given PCI bus/device/function.

Examples

Read the PCI vendor ID and device ID of the onboard ethernet controller at bus 0, device 2, function 0.

```
RedBoot> pcr -b 0 -d 2 -f 0 -o 0 -l 4 -2
0x000: 8086 1010
```

pcw

Name

`pcw` — Issue a PCI configuration write.

Synopsis

pcr {-b *bus*} {-d *device*} {-f *function*} {-o *offset*} {-p *pattern*} [-l *length*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *bus*	Bus Number	Bus Number to be used for configuration transaction.	*none*
-d *device*	Device Number	Device Number to be used for configuration transaction.	*none*
-f *function*	Function Number	Function Number to be used for configuration transaction.	*none*
-o *offset*	Offset	Offset to be written to.	*none*
-p *pattern*	Pattern	Pattern to be written.	*none*
-l *length*	Length	Number of bytes to be written.	*4*
-1		Access one byte (8 bits) at a time.	-1
-2		Access two bytes (16 bits) at a time.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Issue a PCI configuration write transaction to a given PCI bus/device/function.

Examples

Set BAR0 of the onboard ethernet controller to 0x80000000.

```
RedBoot> pcw -b 0 -d 2 -f 0 -o 0x10 -l 4 -4 -p 0x80000000
```

ping

Name

`ping` — Verify network connectivity

Synopsis

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

Arguments

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

Description

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

Examples

Test connectivity to host 192.168.1.101.

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

Test connectivity to host 192.168.1.101, with verbose reporting.

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

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

pir

Name

`pir` — Issue a PCI I/O read.

Synopsis

pir {-b *`PCI I/O Address`*} [-l *`length`*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`PCI I/O Address`*	PCI Address	PCI I/O Address of start of data.	*none*
-l *`length`*	Number	Length of data in bytes	4
-1		Access one byte (8 bits) at a time.	-1
-2		Access two bytes (16 bits) at a time.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Display a range of PCI I/O on the system console.

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

Examples

Display a PCI I/O buffer.

```
RedBoot> piw -b 0x4000 -l 0x20 -p 0xDEADFACE
RedBoot> pir -b 0x4000 -l 0x20
00004000: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
00004010: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
```

pir

piw

Name

`piw` — Issue a PCI I/O write.

Synopsis

piw {-b *`PCI I/O Address`*} {-p *`pattern`*} [-l *`length`*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-b *`PCI I/O Address`*	PCI Address	PCI I/o Address of start of data.	*none*
-l *`length`*	Number	Length of data in bytes	4
-p *`pattern`*	Pattern	Pattern to be written.	*none*
-1		Access one byte (8 bits) at a time.	-1
-2		Access two bytes (16 bits) at a time.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Fill a range of PCI memory with a constant 1, 2, or 4-byte pattern.

Examples

Fill a PCI I/O buffer.

```
RedBoot> piw -b 0x4000 -l 0x20 -p 0xDEADFACE
RedBoot> pir -b 0x4000 -l 0x20
00004000: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
00004010: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
```

pmr

Name

`pmr` — Issue a PCI memory read.

Synopsis

pmr [-u *`Upper PCI Address`*] {-b *`PCI Address`*} [-l *`length`*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-u *`Upper PCI Address`*	Upper PCI Address	Upper 32-bits of a 64-bit PCI Address of start of data.	*0*
-b *`PCI Address`*	PCI Address	Lower 32-bits of PCI Address of start of data.	*none*
-l *`length`*	Number	Length of data in bytes	4
-1		Access one byte (8 bits) at a time.	-1
-2		Access two bytes (16 bits) at a time.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Display a range of PCI memory on the system console.

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

Examples

Display a PCI memory buffer.

```
RedBoot> pmw -b 0x80000000 -l 0x20 -p 0xDEADFACE
RedBoot> pmr -b 0x80000000 -l 0x20
80000000: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
80000010: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
```

pmw

Name

`pmw` — Issue a PCI memory write.

Synopsis

pmw [-u *`Upper PCI Address`*] {-b *`PCI Address`*} {-p *`pattern`*} [-l *`length`*] [-1 | -2 | -4]

Arguments

Name	Type	Description	Default
-u *`Upper PCI Address`*	Upper PCI Address	Upper 32-bits of a 64-bit PCI Address of start of data.	*0*
-b *`PCI Address`*	PCI Address	Lower 32-bits of PCI Address of start of data.	*none*
-l *`length`*	Number	Length of data in bytes	4
-p *`pattern`*	Pattern	Pattern to be written.	*none*
-1		Access one byte (8 bits) at a time.	-1
-2		Access two bytes (16 bits) at a time.	-1
-4		Access one word (32 bits) at a time.	-1

Description

Fill a range of PCI memory with a constant 1, 2, or 4-byte pattern.

Examples

Fill a PCI memory buffer.

```
RedBoot> pmw -b 0x80000000 -l 0x20 -p 0xDEADFACE
RedBoot> pmr -b 0x80000000 -l 0x20
80000000: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
80000010: DEADFACE DEADFACE DEADFACE DEADFACE  |................|
```

reset

Name

`reset` — Reset the device

Synopsis

reset

Arguments

None

Description

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

Examples

Reset the platform.

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

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

Platform: Compaq iPAQ Pocket PC (StrongARM 1110)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

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

version

Name

`version` — Display RedBoot version information

Synopsis

version

Arguments

None

Description

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

Examples

Display RedBoot's version.

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

version

Flash Image System (FIS)

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

fis init

Name

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

Synopsis

fis init [-*f*]

Arguments

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

Description

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

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

Examples

Initialize the FIS directory.

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

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

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

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

fis list

Name

`fis list` — List Flash Image System directory

Synopsis

fis list [-*f*]

Arguments

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

Description

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

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

Examples

List the FIS directory.

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

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

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

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

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

fis free

Name

`fis free` — Free flash image

Synopsis

fis free

Description

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

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

Examples

Show free flash areas.

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

ﬁs free

fis create

Name

`fis create` — Create flash image

Synopsis

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

Arguments

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

Name	**Type**	**Description**	**Default**
-s	Number	Actual length of data written to image. This is used to control the range over which the checksum is made.	It defaults to the length of the last loaded file.
-n		When set, no image data will be written to the flash. Only the FIS directory will be updated.	
name	String	Name of flash image.	

Description

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

Examples

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

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

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

```
RedBoot> fis create junk -b 0x8c400000 -l 0x20000
... Erase from 0xa0040000-0xa0060000: .
... Program from 0x8c400000-0x8c420000 at 0xa0040000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
```

Update the RedBoot[RAM] image.

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

fis load

Name

`fis load` — Load flash image

Synopsis

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

Arguments

Name	**Type**	**Description**	**Default**
-b	Number	Address the image should be loaded to. Executable images normally load at the location to which the file was linked. This option allows the image to be loaded to a specific memory location, possibly overriding any assumed location.	If not specified, the address associated with the image in the FIS directory will be used.
-c		Compute and print the checksum of the image data after it has been loaded into memory.	
-d		Decompress gzipped image while copying it from flash to RAM.	
`name`	String	The name of the file, as shown in the FIS directory.	

Description

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

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

Examples

Load and run RedBoot[RAM] image.

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

fis delete

Name

`fis delete` — Delete flash image

Synopsis

fis delete {*name*}

Arguments

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

Description

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

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

Examples

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

fis lock

Name

`fis lock` — Lock flash area

Synopsis

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

Arguments

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

Description

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

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

Examples

Lock an area of the flash

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

fis lock

fis unlock

Name

`fis unlock` — Unlock flash area

Synopsis

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

Arguments

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

Description

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

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

Examples

Unlock an area of the flash

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

fis unlock

fis erase

Name

`fis erase` — Erase flash area

Synopsis

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

Arguments

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

Description

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

Examples

Erase an area of the flash

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

ﬁs erase

fis write

Name

`fis write` — Write flash area

Synopsis

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

Arguments

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

Description

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

Examples

Write an area of data to the flash

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

Persistent State Flash-based Configuration and Control

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

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

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

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

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

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

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

If the optional flag `-l` is specified, the configuration data is simply listed. Otherwise, each configuration parameter will be displayed and you are given a chance to change it. The entire value must be typed - typing just carriage return will leave a value unchanged. Boolean values may be entered using the first letter (`t` for true, `f` for false). At any time the editing process may be stopped simply by entering a period (.) on the line. Entering the caret (^) moves the editing back to the previous item. See “RedBoot Editing Commands”, the Section called *RedBoot Editing Commands* in Chapter 1.

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

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

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

```
RedBoot> fconfig nickname value
```

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

```
RedBoot> fconfig -l -n
boot_script: false
bootp: false
bootp_my_ip: 10.16.19.176
bootp_server_ip: 10.16.19.66
dns_ip: 10.16.19.1
gdb_port: 9000
net_debug: false
RedBoot> fconfig bootp_my_ip 10.16.19.177
bootp_my_ip: 10.16.19.176 Setting to 10.16.19.177
Update RedBoot non-volatile configuration - continue (y/n)? y
```

```
  ... Unlock from 0x507c0000-0x507e0000: .
  ... Erase from 0x507c0000-0x507e0000: .
  ... Program from 0x0000a8d0-0x0000acd0 at 0x507c0000: .
  ... Lock from 0x507c0000-0x507e0000: .
  RedBoot>
```

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

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

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

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

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

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

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

and you enter the value in the obvious manner thus:

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

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

```
RedBoot> fconfig -l
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.29
Default server IP address: 192.168.1.101
DNS server IP address: 192.168.1.1
```

```
GDB connection port: 9000
Network debug at boot time: false
```

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

```
RedBoot> fconfig
Run script at boot: false t
      Boot script:
Enter script, terminate with empty line
>> fi li
    Boot script timeout: 0 10
Use BOOTP for network configuration: false .
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0xa0fc0000-0xa0fe0000: .
... Program from 0x8c021f60-0x8c022360 at 0xa0fc0000: .
RedBoot>
RedBoot(tm) debug environment - built 08:22:24, Aug 23 2000
Copyright (C) 2000, 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 config   0xA0FC0000   0xA0FC0000  0x020000  0x00000000
FIS directory    0xA0FE0000   0xA0FE0000  0x020000  0x00000000
RedBoot>
```

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

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

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

```
RedBoot> fco
Run script at boot: false t
Boot script:
Enter script, terminate with empty line
>> {ROM}fis load RedBoot[RAM]
>> {ROM}go
>> {RAM}fis li
>>
Boot script timeout (1000ms resolution): 2
Use BOOTP for network configuration: false
 ...
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from 0x007c0000-0x007e0000: .
... Erase from 0x007c0000-0x007e0000: .
... Program from 0xa0015030-0xa0016030 at 0x007df000: .
... Lock from 0x007c0000-0x007e0000: .
RedBoot> reset
... Resetting.
+Ethernet eth0: MAC address 00:80:4d:46:01:05
IP: 192.168.1.153, Default server: 192.168.1.10

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

Platform: IQ80310 (XScale)
Copyright (C) 2000, 2001, 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[RAM]
RedBoot> go
+Ethernet eth0: MAC address 00:80:4d:46:01:05
IP: 192.168.1.153, Default server: 192.168.1.10

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

Platform: IQ80310 (XScale)
Copyright (C) 2000, 2001, 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 config    0x007DF000  0x007DF000  0x00001000  0x00000000
FIS directory     0x007E0000  0x007E0000  0x00020000  0x00000000
RedBoot>
```

Executing Programs from RedBoot

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

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

go

Name

`go` — Execute a program

Synopsis

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

Arguments

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

Description

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

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

Examples

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

```
RedBoot> go 0x40040
```

Execute a program with a timeout.

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

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

exec

Name

`exec` — Execute a Linux kernel

Synopsis

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

Arguments

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

Description

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

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

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

Examples

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

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

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

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

Chapter 3. Rebuilding RedBoot

Introduction

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

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

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

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

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

Rebuilding RedBoot using ecosconfig

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

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

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

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

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

```
U CYGDBG_HAL_COMMON_CONTEXT_SAVE_MINIMUM, new inferred value 0
U CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS, new inferred value 1
U CYGFUN_LIBC_STRING_BSD_FUNCS, new inferred value 0
U CYGPKG_NS_DNS_BUILD, new inferred value 0
```

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

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

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

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

RedBoot can now be built:

```
$ make
```

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

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

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

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

Rebuilding RedBoot from the Configuration Tool

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

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

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

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

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

Chapter 4. Updating RedBoot

Introduction

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

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

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

Load and start a RedBoot RAM instance

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

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

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

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

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

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

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

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

```
... Erase from 0x00020000-0x00040000: ..
... Program from 0x06020000-0x060369c8 at 0x00020000: ..
... Erase from 0x00070000-0x00080000: .
... Program from 0x0606f000-0x0607f000 at 0x00070000: .
RedBoot> fis lock RedBoot[RAM]
  ... Lock from 0x00000000-0x00020000: ..
```

Update the primary RedBoot flash image

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

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

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

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

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

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

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

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

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

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

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

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

Assuming the cksum matches, the next step is programming the image into flash using the FIS commands.

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

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

Reboot; run the new RedBoot image

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

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

Chapter 5. Installation and Testing

ARM/XScale Intel IQ80331

Overview

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

The following RedBoot configurations are supported:

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/redboot_ROM.bin
bin/redboot_ROM.elf
bin/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/redboot_RAM.bin
bin/redboot_RAM.elf
bin/redboot_RAM.srec
```

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

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

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

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

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

Switch Settings

The 80331 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 S7C1-3 to ON when using the board in a backplane.
- For the remaining switch settings, the following is a known good configuration:

S7C1	1 is closed, rest are open
J1B1	Open
J1B2	1-2, 7-8
J1D2	Open
J9C1	Open

LED Codes

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

```
LED     Actions
-------------------------------------------------------------
   Power-On/Reset
0,1:  PBI Initialization Complete

0,2:  ICache Enabled

0,3:  Flash remapped to high address
```

```
0,4:  MMU enabled

0,5:  Begin platform specific initialization

0,6:  MCU initialization

0,7:  Timeout resetting SPD EEPROM read pointer

0,8:  Timeout reading SPD EEPROM contents

0,9:  Checksum error in SPD EEPROM

0,A:  Unknown SDRAM Type (SPD byte #2)

0,B:  Unsupported number of SDRAM rows (SPD byte #3)

0,C:  Unsupported number of SDRAM columns (SPD byte #4)

0,D:  Unsupported number of SDRAM banks (SPD byte #5)

0,E:  Unsupported SDRAM module width (SPD byte #6)

0,F:  Unsupported SDRAM module width (SPD byte #7)

1,0:  Unsupported SDRAM Config (SPD byte #11)

1,1:  Unsupported DDR SDRAM Refresh rate (SPD byte #12)

1,2:  Unsupported DDR-II SDRAM Refresh rate (SPD byte #12)

1,3:  Unsupported SDRAM width (SPD byte #13)

1,4:  Unsupported DDR SDRAM tCAS setting (SPD byte #18)

1,5:  Unsupported DDR-II SDRAM tCAS setting (SPD byte #18)

1,6:  Unsupported DDR Memory Module Attributes (SPD byte #21)

1,7: Unsupported tRP value (SPD byte #27)

1,8:  Unsupported tRCD value (SPD byte #29)

1,9:  Unsupported tRAS value (SPD byte #30)

1,A:  Unsupported DDR bank sizes (SPD byte #31)

1,B:  Unsupported DDR-II bank sizes (SPD byte #31)

1,C:  Unsupported tRC value (SPD byte #41)

1,D:  Unsupported tRFC value (SPD bytes #40 and #42)

1,E:  Error initializing bank registers

1,F:  Error initializing bank register address translation bits
```

```
2,0:  MCU Register initialization complete

2,1:  MCU BIU Dual-porting enabled

2,2:  SDRAM JEDEC initialization complete

2,3:  MCU Initialization complete

2,4:  Setup ATU

2,5:  ATU BAR’s, LR’s initialized for host BIOS configuration

2,6:  Dobson Bridge A Device Hiding Enabled

2,7:  ATU Config Retry released

2,8:  Dobson Bridge B Config Retry released

2,9:  Dobson Bridge A Config Retry released

2,A:  ATU/Bridge Initializaton for host BIOS configuration complete

2,B:  End of C hal_platform_setup routine

2,C:  Cache unlocked, cleared and disabled

2,D:  Cache locked as SRAM at address 0

2,E:  ICache, DCache, BTB, MMU initialized

2,F:  Start of loop to move page tables to RAM

3,0:  End of loop to move page tables to RAM

3,1:  RAM-based page tables active

A,D:  End of platform_setup1 macro

A,7:  Switch to IRQ mode

A,6:  Move SWI & Undefined "vectors" to RAM (at 0x0)

A,5:  Switch to supervisor mode

A,4:  Move remaining "vectors" to RAM (at 0x0)

A,3:  Copy DATA to RAM,
      Initialize interrupt exception environment
      Initialize stack
      Clear BSS section

A,2:  Platform specific hardware initialization
      Initialize debug stub
```

```
A,1:  Run through static constructors
      Start up the eCos kernel or RedBoot

S,L:  MCU Memory Scrub Loop start

S,E:  MCU Memory Scrub Loop end
```

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!

  iq80331/iq80332 CRB Hardware Tests

 1 - Memory Tests
 2 - Random-write Memory Tests
 3 - Repeating Memory Tests
 4 - Repeat-On-Fail Memory Test
 5 - Enhanced Memory Tests
 6 - Rotary Switch S1 Test
 7 - 7 Segment LED Tests
 8 - i82545 Ethernet Configuration
 9 - Battery Status Test
10 - Battery Backup SDRAM Memory Test
11 - Timer Test
12 - PCI Bus test
13 - CPU Cache Loop (No return)
14 - Execute Selected Tests
 0 - quit
Enter the menu item number (0 to quit):
```

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

Memory Tests

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

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

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

Size of memory to test (in hex): 200000

Testing memory from 0x00100000 to 0x002fffff.
```

```
Walking 1's test:
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.
```

Random-write Memory Tests

The random test writes pseudo-random values to memory looking for ECC failures.

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.

Enhanced Memory Tests

The enhanced memory test performs more intensive memory testing. It requires the user to specify the number of iterations the test will execute till.

Repeat-On-Fail Memory Tests

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

Rotary Switch S1 Test

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

7 Segment LED Tests

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

i82544 Ethernet Configuration

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

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

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

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

Battery Status Test

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

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

Battery Backup SDRAM Memory Test

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

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

Timer Test

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

PCI Bus Test

This tests the secondary PCI-X bus and socket. This test requires that an IQ803xx board be plugged into the secondary slot of the IQ80331 board. The test assumes at least 32MB of installed memory on the IQ803xx. That memory is mapped into the IQ80331 address space and the memory tests are run on that memory.

CPU Cache Loop

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

Execute Selected Tests

This executes the timer test and the memory test automatically. It requires no user intervention.

Rebuilding RedBoot

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

```
export TARGET=iq80331
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/iq80331
```

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

Interrupts

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

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

On an IQ80331 board, the vector argument is one of 64 interrupts defined in `hal/arm/xscale/iop33x/current/include/hal_var_ints.h`::

```
#define CYGNUM_HAL_INTERRUPT_NONE          -1
#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_0_4       4
#define CYGNUM_HAL_INTERRUPT_RSVD_0_5       5
#define CYGNUM_HAL_INTERRUPT_AA_EOT         6
#define CYGNUM_HAL_INTERRUPT_AA_EOC        7
#define CYGNUM_HAL_INTERRUPT_TIMER0         8
#define CYGNUM_HAL_INTERRUPT_TIMER1         9
#define CYGNUM_HAL_INTERRUPT_I2C_0          10
#define CYGNUM_HAL_INTERRUPT_I2C_1          11
#define CYGNUM_HAL_INTERRUPT_MU             12
#define CYGNUM_HAL_INTERRUPT_MU_IN_POST_Q   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_WDT            17
#define CYGNUM_HAL_INTERRUPT_RSVD_0_18      18
#define CYGNUM_HAL_INTERRUPT_RSVD_0_19      19
#define CYGNUM_HAL_INTERRUPT_RSVD_0_20      20
#define CYGNUM_HAL_INTERRUPT_RSVD_0_21      21
```

```
#define CYGNUM_HAL_INTERRUPT_RSVD_0_22      22
#define CYGNUM_HAL_INTERRUPT_RSVD_0_23      23
#define CYGNUM_HAL_INTERRUPT_XINT0          24
#define CYGNUM_HAL_INTERRUPT_XINT1          25
#define CYGNUM_HAL_INTERRUPT_XINT2          26
#define CYGNUM_HAL_INTERRUPT_XINT3          27
#define CYGNUM_HAL_INTERRUPT_XINT4          28
#define CYGNUM_HAL_INTERRUPT_XINT5          29
#define CYGNUM_HAL_INTERRUPT_XINT6          30
#define CYGNUM_HAL_INTERRUPT_XINT7          31
#define CYGNUM_HAL_INTERRUPT_XINT8         32
#define CYGNUM_HAL_INTERRUPT_XINT9         33
#define CYGNUM_HAL_INTERRUPT_XINT10        34
#define CYGNUM_HAL_INTERRUPT_XINT11        35
#define CYGNUM_HAL_INTERRUPT_XINT12        36
#define CYGNUM_HAL_INTERRUPT_XINT13        37
#define CYGNUM_HAL_INTERRUPT_XINT14        38
#define CYGNUM_HAL_INTERRUPT_XINT15        39
#define CYGNUM_HAL_INTERRUPT_RSVD_1_8      40
#define CYGNUM_HAL_INTERRUPT_RSVD_1_9      41
#define CYGNUM_HAL_INTERRUPT_RSVD_1_10     42
#define CYGNUM_HAL_INTERRUPT_RSVD_1_11     43
#define CYGNUM_HAL_INTERRUPT_RSVD_1_12     44
#define CYGNUM_HAL_INTERRUPT_RSVD_1_13     45
#define CYGNUM_HAL_INTERRUPT_RSVD_1_14     46
#define CYGNUM_HAL_INTERRUPT_RSVD_1_15     47
#define CYGNUM_HAL_INTERRUPT_RSVD_1_16     48
#define CYGNUM_HAL_INTERRUPT_RSVD_1_17     49
#define CYGNUM_HAL_INTERRUPT_RSVD_1_18     50
#define CYGNUM_HAL_INTERRUPT_UART0         51
#define CYGNUM_HAL_INTERRUPT_UART1         52
#define CYGNUM_HAL_INTERRUPT_PBUS_UNIT     53
#define CYGNUM_HAL_INTERRUPT_ATUCR_WRITE   54
#define CYGNUM_HAL_INTERRUPT_ATU_ERR       55
#define CYGNUM_HAL_INTERRUPT_MCU_ERR       56
#define CYGNUM_HAL_INTERRUPT_DMA0_ERR      57
#define CYGNUM_HAL_INTERRUPT_DMA1_ERR      58
#define CYGNUM_HAL_INTERRUPT_RSVD_1_27    59
#define CYGNUM_HAL_INTERRUPT_AA_ERR        60
#define CYGNUM_HAL_INTERRUPT_RSVD_1_29     61
#define CYGNUM_HAL_INTERRUPT_MU_ERR        62
#define CYGNUM_HAL_INTERRUPT_HPI           63
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table.

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

Memory Maps

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x7fffffff  SDRAM
0x80000000 - 0x87ffffff  ATU Outbound Memory Translation Windows
0x90000000 - 0x900fffff  ATU Outbound I/O Translation Window
0xc0000000 - 0xc07fffff  Flash (PCE0#)
0xce800000 - 0xce8fffff  PCE1# - Uncached
0x0e000000 - 0x0e0fffff  Cache Flush
0xfff00000 - 0xffffffff  80331 I/O Processer Memory Mapped Registers.

Default Virtual Map     Description
----------------------- ----------------------------------
0x00000000 - 0x7fffffff  SDRAM
0x80000000 - 0x87ffffff  ATU Outbound Memory Translation Windows
0x90000000 - 0x900fffff  ATU Outbound I/O Translation Window
0xc0000000 - 0xc07fffff  Flash (PCE0#)
0xce800000 - 0xce8fffff  PCE1# - Uncached
0x0e000000 - 0x0e0fffff  Cache Flush
0xfff00000 - 0xffffffff  80331 I/O Processer Memory Mapped Registers.
```

Platform Resource Usage

The IQ80331 programmable timer0 is used for timeout support for networking and XModem file transfers.

ARM/XScale Intel IQ80332

Overview

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

The following RedBoot configurations are supported:

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/redboot_ROM.bin
bin/redboot_ROM.elf
bin/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/redboot_RAM.bin
bin/redboot_RAM.elf
bin/redboot_RAM.srec
```

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

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

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

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

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

Switch Settings

The 80332 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 S7A1-3 to ON when using the board in a backplane.

- For the remaining switch settings, the following is a known good configuration:

S7A1	1,4,5,6 are closed, rest are open
J1C1	1-2
J1D2	Open
J7B4	1-2, 3-4
J7D1	Open
J9D3	Open

LED Codes

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

```
LED     Actions
-------------------------------------------------------------
   Power-On/Reset
0,1:  PBI Initialization Complete

0,2:  ICache Enabled

0,3:  Flash remapped to high address

0,4:  MMU enabled

0,5:  Begin platform specific initialization

0,6:  MCU initialization

0,7:  Timeout resetting SPD EEPROM read pointer

0,8:  Timeout reading SPD EEPROM contents

0,9:  Checksum error in SPD EEPROM

0,A:  Unknown SDRAM Type (SPD byte #2)

0,B:  Unsupported number of SDRAM rows (SPD byte #3)

0,C:  Unsupported number of SDRAM columns (SPD byte #4)

0,D:  Unsupported number of SDRAM banks (SPD byte #5)

0,E:  Unsupported SDRAM module width (SPD byte #6)

0,F:  Unsupported SDRAM module width (SPD byte #7)
```

```
1,0:  Unsupported SDRAM Config (SPD byte #11)

1,1:  Unsupported DDR SDRAM Refresh rate (SPD byte #12)

1,2:  Unsupported DDR-II SDRAM Refresh rate (SPD byte #12)

1,3:  Unsupported SDRAM width (SPD byte #13)

1,4:  Unsupported DDR SDRAM tCAS setting (SPD byte #18)

1,5:  Unsupported DDR-II SDRAM tCAS setting (SPD byte #18)

1,6:  Unsupported DDR Memory Module Attributes (SPD byte #21)

1,7: Unsupported tRP value (SPD byte #27)

1,8:  Unsupported tRCD value (SPD byte #29)

1,9:  Unsupported tRAS value (SPD byte #30)

1,A:  Unsupported DDR bank sizes (SPD byte #31)

1,B:  Unsupported DDR-II bank sizes (SPD byte #31)

1,C:  Unsupported tRC value (SPD byte #41)

1,D:  Unsupported tRFC value (SPD bytes #40 and #42)

1,E:  Error initializing bank registers

1,F:  Error initializing bank register address translation bits

2,0:  MCU Register initialization complete

2,1:  MCU BIU Dual-porting enabled

2,2:  SDRAM JEDEC initialization complete

2,3:  MCU Initialization complete

2,4:  Setup ATU

2,5:  ATU BAR’s, LR’s initialized for host BIOS configuration

2,6:  Dobson Bridge A Device Hiding Enabled

2,7:  ATU Config Retry released

2,8:  Dobson Bridge B Config Retry released

2,9:  Dobson Bridge A Config Retry released

2,A:  ATU/Bridge Initializaton for host BIOS configuration complete

2,B:  End of C hal_platform_setup routine
```

```
2,C:  Cache unlocked, cleared and disabled

2,D:  Cache locked as SRAM at address 0

2,E:  ICache, DCache, BTB, MMU initialized

2,F:  Start of loop to move page tables to RAM

3,0:  End of loop to move page tables to RAM

3,1:  RAM-based page tables active

A,D:  End of platform_setup1 macro

A,7:  Switch to IRQ mode

A,6:  Move SWI & Undefined "vectors" to RAM (at 0x0)

A,5:  Switch to supervisor mode

A,4:  Move remaining "vectors" to RAM (at 0x0)

A,3:  Copy DATA to RAM,
      Initialize interrupt exception environment
      Initialize stack
      Clear BSS section

A,2:  Platform specific hardware initialization
      Initialize debug stub

A,1:  Run through static constructors
      Start up the eCos kernel or RedBoot

S,L:  MCU Memory Scrub Loop start

S,E:  MCU Memory Scrub Loop end
```

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!

  iq80331/iq80332 CRB Hardware Tests

 1 - Memory Tests
 2 - Random-write Memory Tests
 3 - Repeating Memory Tests
 4 - Repeat-On-Fail Memory Test
 5 - Enhanced Memory Tests
 6 - Rotary Switch S1 Test
 7 - 7 Segment LED Tests
 8 - i82545 Ethernet Configuration
 9 - Battery Status Test
10 - Battery Backup SDRAM Memory Test
11 - Timer Test
12 - PCI Bus test
13 - CPU Cache Loop (No return)
14 - Execute Selected Tests
 0 - quit
Enter the menu item number (0 to quit):
```

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

Memory Tests

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

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

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

Size of memory to test (in hex): 200000

Testing memory from 0x00100000 to 0x002fffff.

Walking 1's test:
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.
```

Random-write Memory Tests

The random test writes pseudo-random values to memory looking for ECC failures.

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.

Enhanced Memory Tests

The enhanced memory test performs more intensive memory testing. It requires the user to specify the number of iterations the test will execute till.

Repeat-On-Fail Memory Tests

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

Rotary Switch S1 Test

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

7 Segment LED Tests

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

i82544 Ethernet Configuration

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

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


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

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

Battery Status Test

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

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

Battery Backup SDRAM Memory Test

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

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

Timer Test

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

PCI Bus Test

This tests the secondary PCI-X bus and socket. This test requires that an IQ803xx board be plugged into the secondary slot of the IQ80332 board. The test assumes at least 32MB of installed memory on the IQ803xx. That memory is mapped into the IQ80332 address space and the memory tests are run on that memory.

CPU Cache Loop

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

Execute Selected Tests

This executes the timer test and the memory test automatically. It requires no user intervention.

Rebuilding RedBoot

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

```
export TARGET=iq80332
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/iq80332
```

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

Interrupts

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

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

On an IQ80332 board, the vector argument is one of 64 interrupts defined in `hal/arm/xscale/iop33x/current/include/hal_var_ints.h`::

```
#define CYGNUM_HAL_INTERRUPT_NONE          -1
#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_0_4       4
#define CYGNUM_HAL_INTERRUPT_RSVD_0_5       5
#define CYGNUM_HAL_INTERRUPT_AA_EOT         6
#define CYGNUM_HAL_INTERRUPT_AA_EOC        7
#define CYGNUM_HAL_INTERRUPT_TIMER0         8
#define CYGNUM_HAL_INTERRUPT_TIMER1         9
#define CYGNUM_HAL_INTERRUPT_I2C_0          10
#define CYGNUM_HAL_INTERRUPT_I2C_1          11
#define CYGNUM_HAL_INTERRUPT_MU             12
#define CYGNUM_HAL_INTERRUPT_MU_IN_POST_Q   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_WDT            17
#define CYGNUM_HAL_INTERRUPT_RSVD_0_18      18
#define CYGNUM_HAL_INTERRUPT_RSVD_0_19      19
#define CYGNUM_HAL_INTERRUPT_RSVD_0_20      20
#define CYGNUM_HAL_INTERRUPT_RSVD_0_21      21
#define CYGNUM_HAL_INTERRUPT_RSVD_0_22      22
#define CYGNUM_HAL_INTERRUPT_RSVD_0_23      23
#define CYGNUM_HAL_INTERRUPT_XINT0          24
#define CYGNUM_HAL_INTERRUPT_XINT1          25
#define CYGNUM_HAL_INTERRUPT_XINT2          26
#define CYGNUM_HAL_INTERRUPT_XINT3          27
#define CYGNUM_HAL_INTERRUPT_XINT4          28
#define CYGNUM_HAL_INTERRUPT_XINT5          29
#define CYGNUM_HAL_INTERRUPT_XINT6          30
#define CYGNUM_HAL_INTERRUPT_XINT7          31
#define CYGNUM_HAL_INTERRUPT_XINT8         32
#define CYGNUM_HAL_INTERRUPT_XINT9         33
#define CYGNUM_HAL_INTERRUPT_XINT10        34
#define CYGNUM_HAL_INTERRUPT_XINT11        35
#define CYGNUM_HAL_INTERRUPT_XINT12        36
#define CYGNUM_HAL_INTERRUPT_XINT13        37
#define CYGNUM_HAL_INTERRUPT_XINT14        38
#define CYGNUM_HAL_INTERRUPT_XINT15        39
#define CYGNUM_HAL_INTERRUPT_RSVD_1_8      40
#define CYGNUM_HAL_INTERRUPT_RSVD_1_9      41
#define CYGNUM_HAL_INTERRUPT_RSVD_1_10     42
#define CYGNUM_HAL_INTERRUPT_RSVD_1_11     43
#define CYGNUM_HAL_INTERRUPT_RSVD_1_12     44
#define CYGNUM_HAL_INTERRUPT_RSVD_1_13     45
#define CYGNUM_HAL_INTERRUPT_RSVD_1_14     46
#define CYGNUM_HAL_INTERRUPT_RSVD_1_15     47
#define CYGNUM_HAL_INTERRUPT_RSVD_1_16     48
```

```
#define CYGNUM_HAL_INTERRUPT_RSVD_1_17     49
#define CYGNUM_HAL_INTERRUPT_RSVD_1_18     50
#define CYGNUM_HAL_INTERRUPT_UART0         51
#define CYGNUM_HAL_INTERRUPT_UART1         52
#define CYGNUM_HAL_INTERRUPT_PBUS_UNIT     53
#define CYGNUM_HAL_INTERRUPT_ATUCR_WRITE   54
#define CYGNUM_HAL_INTERRUPT_ATU_ERR       55
#define CYGNUM_HAL_INTERRUPT_MCU_ERR       56
#define CYGNUM_HAL_INTERRUPT_DMA0_ERR      57
#define CYGNUM_HAL_INTERRUPT_DMA1_ERR      58
#define CYGNUM_HAL_INTERRUPT_RSVD_1_27    59
#define CYGNUM_HAL_INTERRUPT_AA_ERR        60
#define CYGNUM_HAL_INTERRUPT_RSVD_1_29     61
#define CYGNUM_HAL_INTERRUPT_MU_ERR        62
#define CYGNUM_HAL_INTERRUPT_HPI           63
```

The data passed to the ISR is pulled from a data table (`hal_interrupt_data`) which immediately follows the interrupt vector table.

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

Memory Maps

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

```
Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x7fffffff  SDRAM - 64 bit ECC
0x80000000 - 0x87ffffff  ATU Outbound Memory Translation Windows
0x90000000 - 0x900fffff  ATU Outbound I/O Translation Window
0xa0000000 - 0xbfffffff  SDRAM - 64 bit ECC Uncached
0xc0000000 - 0xc07fffff  Flash (PCE0#)
0xce800000 - 0xce8fffff  PCE1# - Uncached
0x0f000000 - 0x0f0fffff  Cache Flush
0xfff00000 - 0xffffffff  80332 I/O Processer Memory Mapped Registers.

Default Virtual Map      Description
-----------------------  ----------------------------------
0x00000000 - 0x7fffffff  SDRAM - 64 bit ECC
0x80000000 - 0x87ffffff  ATU Outbound Memory Translation Windows
0x90000000 - 0x900fffff  ATU Outbound I/O Translation Window
0xa0000000 - 0xbfffffff  SDRAM - 64 bit ECC Uncached
0xc0000000 - 0xc07fffff  Flash (PCE0#)
0xce800000 - 0xce8fffff  PCE1# - Uncached
0x0f000000 - 0x0f0fffff  Cache Flush
0xfff00000 - 0xffffffff  80332 I/O Processer Memory Mapped Registers.
```

Platform Resource Usage

The IQ80332 programmable timer0 is used for timeout support for networking and XModem file transfers.

ARM/XScale Intel IQ80315

Overview

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

The following RedBoot configurations are supported:

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/redboot_ROM.bin
bin/redboot_ROM.elf
bin/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/redboot_RAM.bin
bin/redboot_RAM.elf
bin/redboot_RAM.srec
```

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

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

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

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

```
RedBoot> fis init
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
    Warning: device contents not erased, some blocks may not be usable
... Unlock from 0x407e0000-0x40800000: .
... Erase from 0x407e0000-0x40800000: .
... Program from 0x0ffdf000-0x0ffff000 at 0x407e0000: .
... Lock from 0x407e0000-0x40800000: .
```

Switch Settings

The 80315 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:

- For the IQ80315 switch settings, the following is a known good configuration:

S5C1-1	On
S5C1-4	Off
S5C1-2	Off
S5C1-3	Off
S5C2-1	On
S5C2-2	On
S5C2-3	On
S5C2-4	On
S6C2-1	On
S6C2-2	On
S6C2-3	On
S6C2-4	On

LED Codes

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

```
LED      Actions
-------------------------------------------------------------
IC + ErrorCode I2C Error reading SPD device - see below for I2C error codes
```

```
01  Reading ByteCount from first populated DIMM populated

02  Reading ByteCount from Bank1 (if populated)

03  Reading ByteCount from Bank2 (if populated)

04  SDRAM registers initialized

05  ECC Enabled

SL  Scrub Loop - initializing RAM to 0x0

08  Vector Code copied to RAM

09  Cache Flushed after vector copy

10  MMU Page Tables copied to RAM

11  Cache flushed after page table copy

PP  32Meg memory test passed

FF  32Meg memory test failed - RedBoot will not run if this test fails

EC  ECC error occurred during 32Meg Memory Test - RedBoot will not run if this test fails

A1  Boot complete

--------I2C Error Codes----------------------------------

01  I2C timed out reading an SPD byte

22  No DDR Module installed

02  Invalid Refresh Value found in SPD

03  Neither CAS1.5 or CAS2.5 supported by this DIMM

04  Burst length of 2 not supported by this DIMM

05  x4, x8, x16 not supported by DIMM

06  x4, x8, x16 for ECC_SZ not supported by DIMM

07  Buffered/Registered invalid

08  Number of PHY banks invalid

09  Number of Logical banks invalid

10  Number of Columns invalid

11  TRas invalid

12  TRcd invalid
```

`13`	`Trp invalid`
`14`	`Trfc invalid`
`15`	`Density invalid in first populated DDR slot`
`16`	`Density invalid for DDR_1`
`17`	`Dimm size invalid for DDR_1`
`18`	`Num banks invalid for DDR_1`
`19`	`Density invalid for DDR_2`
`20`	`Dimm size invalid for DDR_2`
`21`	`Num banks invalid for DDR_2`

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!

 IQ80315 Diags

 1 - Memory Test Sub-Menu
 2 - Rotary Switch Test
 3 - 7 Segment LED Tests
 4 - IOC80314 Ethernet Configuration
 5 - Battery Status Test
 6 - Battery Backup SDRAM Memory Test
 7 - Timer Test
 8 - PCI Bus test
 9 - CPU Cache Loop (No return)
10 - Read DDR0 and DDR1 Size on I2C
11 - Read Temp Sensor(s) on I2C
12 - Basic Sanity Tests
13 - Loop Version of Basic Sanity Tests
14 - EEPROM menu
15 - RTC menu
16 - Fan menu
17 - Test Buzzer
18 - CompactFlash Test
19 - LCD Test Menu
 0 - quit
Enter the menu item number (0 to quit):
```

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

Memory Tests Sub Menu

This test is used to test installed DDR SDRAM memory. Selecting this option presents the user with the following menu.

```
============================================
|         Memory Test Sub Menu             |
============================================
|              Parameters                  |
| 0) Select Test Base and Range            |
| 1) Select # of Itterations               |
| 2) Enable or Disable Data Cache          |
| 3) Disable ECC                           |
|                                          |
|---------------Tests---------------------|
| 4) Basic Memory Tests                    |
| 5) Pattern Memory Tests                  |
| 6) Pattern Memory Tests Variation        |
| 7) Address as Data Memory Tests          |
| 8) Run all Tests Sequentially            |
| 9) Return to Main Menu                   |
============================================
```

The first four options allow the user to set/enable/disable the various parameters that are required for the actual memory diagnostics which are presented as the remaining six options.

Selecting the Basic Memory Tests options executes the following test.

```
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.
```

Selecting the Patter Memory Tests executes the following test.

```
Enabling data cache
Starting Iteration: 1
Testing memory from 0x00000000 to 0xffffffff.
Filling Pattern: 0x00000000
Verifying Pattern: 0x00000000
Filling Pattern: 0xffffffff
Verifying Pattern: 0xffffffff
Filling Pattern: 0xffff0000
Verifying Pattern: 0xffff0000
Filling Pattern: 0x0000ffff
Verifying Pattern: 0x0000ffff
Filling Pattern: 0xff00ff00
```

```
Verifying Pattern: 0xff00ff00
Filling Pattern: 0x00ff00ff
Verifying Pattern: 0x00ff00ff
Filling Pattern: 0xf0f0f0f0
Verifying Pattern: 0xf0f0f0f0
Filling Pattern: 0x0f0f0f0f
Verifying Pattern: 0x0f0f0f0f
Filling Pattern: 0xaaaa5555
Verifying Pattern: 0xaaaa5555
Filling Pattern: 0x5555aaaa
Verifying Pattern: 0x5555aaaa
Filling Pattern: 0x5a5a5a5a
Verifying Pattern: 0x5a5a5a5a
Filling Pattern: 0x55aa55aa
Verifying Pattern: 0x55aa55aa
Filling Pattern: 0xaa55aa55
Verifying Pattern: 0xaa55aa55
Filling Pattern: 0xa5a5a5a5
Verifying Pattern: 0xa5a5a5a5
Filling Pattern: 0xcccc3333
Verifying Pattern: 0xcccc3333
Filling Pattern: 0x3333cccc
Verifying Pattern: 0x3333cccc
Filling Pattern: 0x33cc33cc
Verifying Pattern: 0x33cc33cc
Filling Pattern: 0xcc33cc33
Verifying Pattern: 0xcc33cc33
Filling Pattern: 0x3c3c3c3c
Verifying Pattern: 0x3c3c3c3c
Filling Pattern: 0xc3c3c3c3
Verifying Pattern: 0xc3c3c3c3

Memory test done.

Checking for any ECC SingleBit Errors
No ECC Errors Found
********** SUMMARY ************
Iterations: 1
Failed Runs:  0
Pass/Fail: Pass
*******************************
```

This test basically writes and verifies a pattern to a user defined range of memory.

Selecting the Pattern Memory Tests Variation performs the following operation

```
** STARTING Pattern Memory Tests Variation **
Enabling data cache
Starting Iteration: 1
Testing memory from 0x00000000 to 0xffffffff.
Clearing SDRAM Test Range
Filling Pattern: 0x00000000 and immediately verifying
Verifying Pattern over full range: 0x00000000
Filling Pattern: 0xffffffff and immediately verifying
Verifying Pattern over full range: 0xffffffff
```

```
Filling Pattern: 0xffff0000 and immediately verifying
Verifying Pattern over full range: 0xffff0000
Filling Pattern: 0x0000ffff and immediately verifying
Verifying Pattern over full range: 0x0000ffff
Filling Pattern: 0xff00ff00 and immediately verifying
Verifying Pattern over full range: 0xff00ff00
Filling Pattern: 0x00ff00ff and immediately verifying
Verifying Pattern over full range: 0x00ff00ff
Filling Pattern: 0xf0f0f0f0 and immediately verifying
Verifying Pattern over full range: 0xf0f0f0f0
Filling Pattern: 0x0f0f0f0f and immediately verifying
Verifying Pattern over full range: 0x0f0f0f0f
Filling Pattern: 0xaaaa5555 and immediately verifying
Verifying Pattern over full range: 0xaaaa5555
Filling Pattern: 0x5555aaaa and immediately verifying
Verifying Pattern over full range: 0x5555aaaa
Filling Pattern: 0x5a5a5a5a and immediately verifying
Verifying Pattern over full range: 0x5a5a5a5a
Filling Pattern: 0x55aa55aa and immediately verifying
Verifying Pattern over full range: 0x55aa55aa
Filling Pattern: 0xaa55aa55 and immediately verifying
Verifying Pattern over full range: 0xaa55aa55
Filling Pattern: 0xa5a5a5a5 and immediately verifying
Verifying Pattern over full range: 0xa5a5a5a5
Filling Pattern: 0xcccc3333 and immediately verifying
Verifying Pattern over full range: 0xcccc3333
Filling Pattern: 0x3333cccc and immediately verifying
Verifying Pattern over full range: 0x3333cccc
Filling Pattern: 0x33cc33cc and immediately verifying
Verifying Pattern over full range: 0x33cc33cc
Filling Pattern: 0xcc33cc33 and immediately verifying
Verifying Pattern over full range: 0xcc33cc33
Filling Pattern: 0x3c3c3c3c and immediately verifying
Verifying Pattern over full range: 0x3c3c3c3c
Filling Pattern: 0xc3c3c3c3 and immediately verifying
Verifying Pattern over full range: 0xc3c3c3c3

Memory test done.

Checking for any ECC SingleBit Errors
No ECC Errors Found
********** SUMMARY ************
Iterations: 1
Failed Runs:  0
Pass/Fail: Pass
*******************************
```

This test is just a simple variation of the preceding memory tests.

Address as Data Memory Tests This test verifies teh data in the memory range on the SRDRAM as addresses.

```
** STARTING Address as Data Memory Tests **
Enabling data cache
Starting Iteration: 1
Testing memory from 0x00000000 to 0xffffffff.
```

```
Verifying Data as Addresses
Now Verifying from Top to Bottom
Verifying Data as Addresses

Memory test done.

Checking for any ECC SingleBit Errors
No ECC Errors Found
********** SUMMARY ************
Iterations: 1
Failed Runs:  0
Pass/Fail: Pass
*******************************
```

Run all Tests Sequentially (final option on the memory diagnostics sub-menu) This test runs all above mentioned memory diagnostic tests in a sequential order.

Rotary Switch S1 Test

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

7 Segment LED Tests

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

IOC80314 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): 4

IOC80314 Ethernet Configuration
Total Data Pairs in EEPROM: 0x2
Address: 0xfffe6044
Found PORT0 Lower Address in EEPROM at location: 0xc
Data: 0x0200ffff
Address: 0xfffe6040
Found PORT0 Upper Address in EEPROM at location: 0x14
Data: 0x2ecb00b3
Port0 MAC Address from EEPROM: 00:02:b3:00:cb:2e
1) Set MAC Address for Port0
2) Set MAC Address for Port1
1

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

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

Battery Status Test

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

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

Battery Backup SDRAM Memory Test

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

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

Timer Test

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

PCI Bus Test

This tests the secondary PCI-X bus and socket. This test requires that an IQ80310 board be plugged into the secondary slot of the IQ80321 board. The test assumes at least 32MB of installed memory on the IQ80310. That memory is mapped into the IQ80321 address space and the memory tests are run on that memory.

CPU Cache Loop

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

Read DDR0 and DDR1 Size on I2C

This test probes the SPD devices on BANK0 and BANK1 of the IQ80315 and reports how memory is inserted into the banks.

Read Temp Sensor(s) on I2C

The two temperature sensors located on the I2C bus can be probed using this diagnostic. The output continuously reports the temperature of the two devices (in C).

Basic Sanity Tests

This executes the memory test on SRAM and SDRAM, the seven segment display test and the PHY Register test, and finally also reads from an internal register of the IOC80314.

Loop Version of Basic Sanity Tests

Performs the above mentioned tests, except in a loop. The test can be stopped by hitting the ’enter’ key on the keyboard.

EEPROM menu

The EEPROM menu option presents a menu of EEPROM diagnostics that can be run for the IQ80315.

```
=======================================
|  I2C EEPROM menu for IQ80315         |
=======================================
| 1) Read  EEPROM                      |
| 2) Write EEPROM                      |
| 3) Erase EEPROM                      |
---------------------------------------
```

Selecting Read EEPROM presents the folling menu.

```
Total of 2 address/data pairs in EEPROM
Choose Read Method:
1) Address Data Pairs
2) Dump All Valid Bytes
3) Dump Select Bytes
```

The first option will print all the address data pairs in EEPROM. The second option prints out all valid bytes and the final option queries the user for an address and dumps the data at that location.

Selecting Write queries for a location where user queried data will be written to.

Selecting Erase EEPROM option erases the entire eeprom.

RTC Menu

This diagnostic allows the programming and the reading of the RTC on the I2C bus.

```
=======================================
|  Real Time Clock Menu for IQ80315    |
=======================================
| 1) Display Current Time              |
| 2) Set Time                          |
---------------------------------------
```

The names of these options are pretty self-explanatory. The first displays the time the clock is currently set to, and the final option allows the user to set the time.

Fan menu

This menu allows the user to toggle the front and the rear fans.

```
=======================================
|  I2C Case Fan menu for IQ80315       |
=======================================
| 1) Turn Front Case Fans On           |
| 2) Turn Front Case Fans Off          |
| 3) Turn Rear  Case Fans On           |
| 4) Turn Rear  Case Fans Off          |
---------------------------------------
```

Test Buzzer

This option outputs a square wave from teh RTC to the buzzer or speaker.

CompactFlash Test

This option performs a read/write test on the compact flash device.

LCD Test Menu

This menu provides further more options how to manupilate the LCD.

```
+======================================+
|        LCD Menu for IQ80315          |
+======================================+
| 1) Fill 1x16 Display                 |
| 2) Fill 2x16 Display                 |
| 3) Fill 1x20 Display                 |
| 4) Fill 2x20 Display                 |
| 5) Fill 4x20 Display                 |
| 6) Fill 1x40 Display                 |
| 7) Fill 2x40 Display                 |
| 8) Clear Display                     |
+--------------------------------------+
```

Rebuilding RedBoot

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

```
export TARGET=iq80315
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/iq80315
```

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

Interrupts

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

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

On an IQ80315 board, the vector argument is one of 34 interrupts defined in `hal/arm/xscale/ioc80314/current/include/hal_var_ints.h`::

```
#define CYGNUM_HAL_INTERRUPT_NONE         -1
#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      21
#define CYGNUM_HAL_INTERRUPT_RSVD_22       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
#define CYGNUM_HAL_INTERRUPT_SERIAL_A      32 // 16x50 uart A
#define CYGNUM_HAL_INTERRUPT_SERIAL_B      33 // 16x50 uart B
```

The data passed to the ISR is pulled from a data table `(hal_interrupt_data)` which immediately follows the interrupt vector table.

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

Memory Maps

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

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

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

Physical Address Range     Description
-----------------------    ----------------------------------
0x00000000 - 0x40000000  SDRAM
0x40000000 - 0x50000000  FLASH/PBI
0x50000000 - 0x50100000  SRAM
0x50100000 - 0x50110000  Control Registers
0x60000000 - 0x80000000  Un-cached SDRAM Alias
0x80000000 - 0x9EFF0000  PCI1 MEM32
0x9EFF0000 - 0x9F000000  PCI1 I/O
0x9F000000 - 0xA0000000  PCI1 CFG
0xA0000000 - 0xB0000000  PCI1 PFM1
0xB0000000 - 0xC0000000  PCI1 PFM2
0xC0000000 - 0xDEFF0000  PCI2 MEM32
0xDEFF0000 - 0xDF000000  PCI2 I/O
0xDF000000 - 0xE0000000  PCI2 CFG
0xE0000000 - 0xF0000000  PCI2 PFM1
0xF0000000 - 0xFFFFFFFF  PCI2 PFM2
```

```
Default Virtual Map      X C B  Description
----------------------  - - -  ----------------------------------
0x00000000 - 0x40000000 1 1 1 SDRAM
0x40000000 - 0x50000000 0 1 0 FLASH/PBI
0x50000000 - 0x50100000 1 1 1 SRAM
0x50100000 - 0x50110000 0 0 0 Control Registers
0x50110000 - 0x60000000 0 0 0 No access
0x60000000 - 0x80000000 0 0 0 Un-cached SDRAM Alias
0x80000000 - 0x9EFF0000 0 0 0 PCI1 MEM32
0x9EFF0000 - 0x9F000000 0 0 0 PCI1 I/O
0x9F000000 - 0xA0000000 0 0 0 PCI1 CFG
0xA0000000 - 0xB0000000 0 0 0 PCI1 PFM1
0xB0000000 - 0xC0000000 0 0 0 PCI1 PFM2
0xC0000000 - 0xDEFF0000 0 0 0 PCI2 MEM32
0xDEFF0000 - 0xDF000000 0 0 0 PCI2 I/O
0xDF000000 - 0xE0000000 0 0 0 PCI2 CFG
0xE0000000 - 0xF0000000 0 0 0 PCI2 PFM1
0xF0000000 - 0xFFFFFFFF 0 0 0 PCI2 PFM2
```

ARM/Xscale Intel IQ31244

Overview

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

The following RedBoot configurations are supported:

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/iq31244/redboot_ROM.bin
bin/iq31244/redboot_ROM.elf
bin/iq31244/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/iq31244/redboot_RAM.bin
bin/iq31244/redboot_RAM.elf
bin/iq31244/redboot_RAM.srec
```

Initial installations use 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 instructions to install the bin/iq31244/redboot_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 - continue (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
    ... Unlock from 0xf07e0000-0xf0800000: .
    ... Erase from 0xf07e0000-0xf0800000: .
    ... Program from 0x01ddf000-0x01ddf400 at 0xf07e0000: .
    ... Lock from 0xf07e0000-0xf0800000: .
```

LED Codes

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

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

```
        Setup and enable the MMU
A4
        I2C interface initialization
90
        Wait for I2C initialization to complete
91
        Send address (via I2C) to the DIMM
92
        Wait for transmit complete
93
        Read SDRAM PD data from DIMM
94
        Read remainder of EEPROM data.
        An error will result in one of the following
        error codes on the LEDs:
 77 BAD EEPROM checksum
 55 I2C protocol error
 FF bank size error
A5
        Setup DDR memory interface
A6
        Enable branch target buffer
        Drain the write & fill buffers
        Flush Icache, Dcache and BTB
        Flush instuction and data TLBs
        Drain the write & fill buffers
SL
        ECC Scrub Loop
SE
A7
        Clean, drain, flush the main Dcache
A8
        Clean, drain, flush the mini Dcache
        Flush Dcache
        Drain the write & fill buffers
A9
        Enable ECC
AA
        Save SDRAM size
        Move MMU tables into RAM
AB
        Clean, drain, flush the main Dcache
        Clean, drain, flush the mini Dcache
        Drain the write & fill buffers
AC
        Set the TTB register to DRAM mmu_table
AD
        Set mode to IRQ mode
A7
        Move SWI & Undefined "vectors" to RAM (at 0x0)
A6
        Switch to supervisor mode
A5
        Move remaining "vectors" to RAM (at 0x0)
A4
```

```
        Copy DATA to RAM
        Initialize interrupt exception environment
        Initialize stack
        Clear BSS section
A3
        Call platform specific hardware initialization
A2
        Run through static constructors
A1
        Start up the eCos kernel or RedBoot
```

Special RedBoot Commands

`cf info`

Returns Compact Flash information. If a card is installed, the number of blocks and the size of the card is printed.

`cf read -s <sector_number> -b <mem_base> -n <num_sectors>`

Reads num_sectors starting at sector_number and places data at mem_base.

`cf write -s <sector_number> -b <mem_base> -n <num_sectors>`

Writes num_sectors of data from mem_base to Compact Flash starting at sector_number.

`bexec [-w timeout] [-c "kernel commandline"] <entry_point>`

Execute a BSD boot image.

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!

  IQ31244 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 - i82546 Ethernet Configuration
 7 - i82546 Ethernet Test
 8 - Timer Test
 9 - LM75 Test
10 - PCI Bus test
11 - Compact Flash Test
12 - DMA Test
13 - CPU Cache Loop (No Return)
 0 - quit
Enter the menu item number (0 to quit):
```

Tesfor various hardware subsystems are prov[a] Tided, and some tests require special hardware in order to execute normally. The Ethernet Configuration item may be used to set the board ethernet address.

Memory Tests

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

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

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

Size of memory to test (in hex): 200000

Testing memory from 0x00100000 to 0x002fffff.

Walking 1’s test:
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.
```

Repeating Memory Tests

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

Repeat-On-Fail Memory Tests

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

Rotary Switch S1 Test

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

7 Segment LED Tests

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

i82546 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. The least significant bit of the address must be zero as the MAC address for the second port on the 82546 is formed by setting this bit.

```
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 ********
```

i82546 Ethernet Configuration

This runs internal and external loopback tests on the 82546 ports.

Timer Test

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

LM75 Test

This tests the two LM75 temperature sensors.

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 IQ31244 board. The test assumes at least 32MB of installed memory on the IQ80310. That memory is mapped into the IQ31244 address space and the memory tests are run on that memory.

DMA Test

This tests the IOP321 DMA transfers. This test requires that an IQ80321 board be plugged into the secondary slot of the IQ31244 board. That IQ80321 memory is used as a target and source for DMA transfers.

CPU Cache Loop

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

Rebuilding RedBoot

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

```
export TARGET=iq31244
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/iq31244
```

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

Interrupts

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

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

On an IQ31244 board, the vector argument is one of 32 interrupts defined in `hal/arm/xscale/verde/current/include/hal_var_ints.h`::

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

```
#define CYGNUM_HAL_INTERRUPT_XINT2          29
#define CYGNUM_HAL_INTERRUPT_XINT3          30
#define CYGNUM_HAL_INTERRUPT_HPI            31
```

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

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

Memory Maps

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

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
```

Platform 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 IOP321 programmable timer0 is used for timeout support for networking and XModem file transfers.

ARM/Xscale Intel EP80219

Overview

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

The following RedBoot configurations are supported:

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/ep80219/redboot_ROM.bin
bin/ep80219/redboot_ROM.elf
bin/ep80219/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/ep80219/redboot_RAM.bin
bin/ep80219/redboot_RAM.elf
bin/ep80219/redboot_RAM.srec
```

Initial installations use 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 instructions to install the bin/ep80219/redboot_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 - continue (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
    ... Unlock from 0xf07e0000-0xf0800000: .
    ... Erase from 0xf07e0000-0xf0800000: .
    ... Program from 0x01ddf000-0x01ddf400 at 0xf07e0000: .
    ... Lock from 0xf07e0000-0xf0800000: .
```

LED Codes

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

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

```
        Setup and enable the MMU
A4
        I2C interface initialization
90
        Wait for I2C initialization to complete
91
        Send address (via I2C) to the DIMM
92
        Wait for transmit complete
93
        Read SDRAM PD data from DIMM
94
        Read remainder of EEPROM data.
        An error will result in one of the following
        error codes on the LEDs:
 77 BAD EEPROM checksum
 55 I2C protocol error
 FF bank size error
A5
        Setup DDR memory interface
A6
        Enable branch target buffer
        Drain the write & fill buffers
        Flush Icache, Dcache and BTB
        Flush instuction and data TLBs
        Drain the write & fill buffers
SL
        ECC Scrub Loop
SE
A7
        Clean, drain, flush the main Dcache
A8
        Clean, drain, flush the mini Dcache
        Flush Dcache
        Drain the write & fill buffers
A9
        Enable ECC
AA
        Save SDRAM size
        Move MMU tables into RAM
AB
        Clean, drain, flush the main Dcache
        Clean, drain, flush the mini Dcache
        Drain the write & fill buffers
AC
        Set the TTB register to DRAM mmu_table
AD
        Set mode to IRQ mode
A7
        Move SWI & Undefined "vectors" to RAM (at 0x0)
A6
        Switch to supervisor mode
A5
        Move remaining "vectors" to RAM (at 0x0)
A4
```

```
        Copy DATA to RAM
        Initialize interrupt exception environment
        Initialize stack
        Clear BSS section
A3
        Call platform specific hardware initialization
A2
        Run through static constructors
A1
        Start up the eCos kernel or RedBoot
```

Special RedBoot Commands

`cf info`

Returns Compact Flash information. If a card is installed, the number of blocks and the size of the card is printed.

`cf read -s <sector_number> -b <mem_base> -n <num_sectors>`

Reads num_sectors starting at sector_number and places data at mem_base.

`cf write -s <sector_number> -b <mem_base> -n <num_sectors>`

Writes num_sectors of data from mem_base to Compact Flash starting at sector_number.

`bexec [-w timeout] [-c "kernel commandline"] <entry_point>`

Execute a BSD boot image.

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!

  EP80219 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 - i82546 Ethernet Configuration
 7 - i82546 Ethernet Test
 8 - Timer Test
 9 - LM75 Test
10 - PCI Bus test
11 - Compact Flash Test
12 - DMA Test
13 - CPU Cache Loop (No Return)
 0 - quit
Enter the menu item number (0 to quit):
```

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

Memory Tests

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

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

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

Size of memory to test (in hex): 200000

Testing memory from 0x00100000 to 0x002fffff.

Walking 1’s test:
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.
```

Repeating Memory Tests

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

Repeat-On-Fail Memory Tests

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

Rotary Switch S1 Test

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

7 Segment LED Tests

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

i82546 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. The least significant bit of the address must be zero as the MAC address for the second port on the 82546 is formed by setting this bit.

```
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 ********
```

i82546 Ethernet Configuration

This runs internal and external loopback tests on the 82546 ports.

Timer Test

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

LM75 Test

This tests the two LM75 temperature sensors.

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 EP80219 board. The test assumes at least 32MB of installed memory on the IQ80310. That memory is mapped into the EP80219 address space and the memory tests are run on that memory.

DMA Test

This tests the IOP321 DMA transfers. This test requires that an IQ80321 board be plugged into the secondary slot of the EP80219 board. That IQ80321 memory is used as a target and source for DMA transfers.

CPU Cache Loop

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

Rebuilding RedBoot

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

```
export TARGET=ep80219
export ARCH_DIR=arm
export PLATFORM_DIR=xscale/ep80219
```

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

Interrupts

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

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

On an EP80219 board, the vector argument is one of 32 interrupts defined in `hal/arm/xscale/verde/current/include/hal_var_ints.h`::

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

```
#define CYGNUM_HAL_INTERRUPT_XINT2         29
#define CYGNUM_HAL_INTERRUPT_XINT3         30
#define CYGNUM_HAL_INTERRUPT_HPI           31
```

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

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

Memory Maps

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

> **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
```

Platform 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 IOP321 programmable timer0 is used for timeout support for networking and XModem file transfers.

ARM/XScale Cyclone IQ80310

Overview

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

The following RedBoot configurations are supported:

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/redboot_ROM.bin
bin/redboot_ROM.elf
bin/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/redboot_RAM.bin
bin/redboot_RAM.elf
bin/redboot_RAM.srec
```

Initial installations use 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 ROM mode image at address zero.

To install RedBoot to run from address 0x40000 with the ARM bootloader in the flash boot sector, use the manufacturer's flash utility to install the ROMA mode image at address 0x40000.

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

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

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

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

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

Error codes

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

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

Using RedBoot with ARM Bootloader

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

Special RedBoot Commands

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

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

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

IQ80310 Hardware Tests

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

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

Rebuilding RedBoot

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

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

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

Interrupts

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

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

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

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


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


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


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


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

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


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


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

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

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

Memory Maps

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

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

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

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


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

Platform Resource Usage

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

ARM/XScale Intel IQ80321

Overview

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

The following RedBoot configurations are supported:

Configuration	Mode	Description	File

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

Initial Installation Method

The initial flash programming must be done through the JTAG port. See board manufacturers documentation for more information on programming flash through JTAG. RedBoot should be programmed to flash offset 0x00000000 using the JTAG flash utility.

Two sets of prebuilt files are provided in a tarball. 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:
bin/redboot_ROM.bin
bin/redboot_ROM.elf
bin/redboot_ROM.srec

For RedBoot running from RAM with RedBoot in the flash boot sector:
bin/redboot_RAM.bin
bin/redboot_RAM.elf
bin/redboot_RAM.srec
```

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

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

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

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

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

Switch Settings

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

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

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

LED Codes

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

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

A3	Setup and enable the MMU
A4	I2C interface initialization
90	Wait for I2C initialization to complete
91	Send address (via I2C) to the DIMM
92	Wait for transmit complete
93	Read SDRAM PD data from DIMM
94	Read remainder of EEPROM data. An error will result in one of the following error codes on the LEDs: 77 BAD EEPROM checksum 55 I2C protocol error FF bank size error
A5	Setup DDR memory interface
A6	Enable branch target buffer Drain the write & fill buffers Flush Icache, Dcache and BTB Flush instuction and data TLBs Drain the write & fill buffers
SL	ECC Scrub Loop
SE	
A7	Clean, drain, flush the main Dcache
A8	Clean, drain, flush the mini Dcache Flush Dcache Drain the write & fill buffers
A9	Enable ECC
AA	Save SDRAM size Move MMU tables into RAM
AB	Clean, drain, flush the main Dcache Clean, drain, flush the mini Dcache Drain the write & fill buffers
AC	Set the TTB register to DRAM mmu_table
AD	Set mode to IRQ mode
A7	Move SWI & Undefined "vectors" to RAM (at 0x0)
A6	Switch to supervisor mode
A5	Move remaining "vectors" to RAM (at 0x0)

```
A4
        Copy DATA to RAM
        Initialize interrupt exception environment
        Initialize stack
        Clear BSS section
A3
        Call platform specific hardware initialization
A2
        Run through static constructors
A1
        Start up the eCos kernel or RedBoot
```

Special RedBoot Commands

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

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

  IQ80321 Hardware Tests

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

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

Memory Tests

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

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

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

Size of memory to test (in hex): 200000
```

```
Testing memory from 0x00100000 to 0x002fffff.

Walking 1's test:
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.
```

Repeating Memory Tests

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

Repeat-On-Fail Memory Tests

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

Rotary Switch S1 Test

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

7 Segment LED Tests

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

i82544 Ethernet Configuration

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

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


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

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

Battery Status Test

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

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

Battery Backup SDRAM Memory Test

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

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

Timer Test

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

PCI Bus Test

This tests the secondary PCI-X bus and socket. This test requires that an IQ80310 board be plugged into the secondary slot of the IQ80321 board. The test assumes at least 32MB of installed memory on the IQ80310. That memory is mapped into the IQ80321 address space and the memory tests are run on that memory.

CPU Cache Loop

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

Rebuilding RedBoot

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

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

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

Interrupts

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

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

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

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

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

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

Memory Maps

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

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

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

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


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

Alternate Virtual Map    X C B  Description
```

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

Platform Resource Usage

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