Redboot User's Guide

September 2000

Copying terms

Trademarks

Contents

1 Introduction

RedBoot is the new debug/bootstrap environment from RedHat. It is designed to replace the existing debug/boot tools CygMon and GDB stubs. It provides a complete bootstrap environment, including network downloading and debugging. Also provided is a simple flash file system for boot images.

Some host services may be necessary in order to run RedBoot, especially the first time. In particular:

- TFTP - RedBoot uses TFTP for downloading images. Most systems come with this disabled, so it may be necessary to enable a TFTP server. Also note that the files mentioned below are assumed to be in the TFTP directory, normally "/tftpboot".
- BOOTP/DHCP - RedBoot will use BOOTP to get its network address parameters unless setup otherwise. For the initial setup, BOOTP will be required.

Since RedBoot is still quite new, there are some limitations (which we expect to remove very soon):

- The only format supported for image download is Motorola S-records. Thus files to be downloaded would have to be converted using the "objcopy" utility for the platform in question; for EBSA this is arm-elf-objcopy. S-record versions of the boot images are produced as part of the normal RedBoot build process (see below).
- Serial downloads are not supported; TFTP only in this version.
- RedBoot "chatters" somewhat about network packets that it discards; this is to help with our debugging.
- Asynchronous interrupt (^C) of the board is not yet supported in RedBoot.

NOTES:

- Only one network interface is used by RedBoot. This will be the first interface found, which on the EBSA is the board in the highest numbered PCI slot.
- In order to use GDB, each network interface needs to have two IP addresses. One will be used for RedBoot/GDB and the other for eCos applications. After RedBoot is installed in the flash, we use the RedBoot configuration to force the IP address for the board's RedBoot persona, and then BOOTP for the eCos application's IP address.
- In case of difficulties with RedBoot, it is possible to go back to the "old way" of doing things, with built-in debug stubs and serial-only debug and so on; that way the eCos application and its built-in GDB stubs "takes over" the whole machine. Installing RedBoot is not a one-way street; it can be used just as CygMon or an eCos stub ROM was.

2 User Commands

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:

The general format of commands is:

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

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

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

Use the ‘help’ command to get limited help on command syntax.

```
==============================================================
RedBoot> help
Display (hex dump) a range of memory
        dump <location> [<length>]
Manage FLASH images
        fis {cmds}
Manage configuration kept in FLASH memory
        fconfig
Execute code at a location
```

```
        go [-w <timeout>] [entry]
Help about help?
        help <topic>
Load a file
        load [-raw] [-b <mem_addr>]]
RedBoot>
==========================================================================
```

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

Download Process

Currently, download is only supported using TFTP over a network. Files to be downloaded may either be executable images in SREC format or raw data. The format of the command is:

```
RedBoot> load <file> [-v] [-b <location>] [-r]
```

where:

<file> The name of the file on the TFTP server. Details of how this is specified are host-specific.

-v Display a small spinner (indicator) while the download is in progress. This is just for feedback, especially during long loads.

-b Specify the location in memory to which the file should be loaded. Executable images normally load at the location to which the file was linked. This option allows the file to be loaded to a specific memory location, possibly overriding any assumed location.

-r Download raw data. Normally, the load command is used to load executable images into memory. This option allows for raw data to be loaded. If this option is given, “-b” will also be required.

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

Flash Image System (FIS)

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

The available ‘fis’ commands are:

fis init [-f]

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

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

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

fis list

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

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

fis free

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

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

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

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

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

Options:

<name> The name of the file, as shown in the FIS directory.

-b The location in RAM used to obtain the image. This is a required option.

-l The length of the location. This is a required option.

-f The location in flash for the image. If this is not provided, the first freeVblock which is large enough will be used. See 'fis free'.

-e The execution entry address. This is used if the starting address for an image is not known, or needs to be overridden.

-r The location in RAM when the image is loaded via 'fis load'. This only needs to be specified for images which will eventually loaded via 'fis load'. Fixed images, such as RedBoot itself, will not need this.

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

fis load name

This command is used to transfer an image from flash memory to RAM. Once loaded, it may be executed using the 'go' command.

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

fis delete name

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

```
==========================================================================
RedBoot> fis list
Name             FLASH addr   Mem addr    Length    Entry point
RedBoot          0xA0000000   0xA0000000  0x020000  0x80000000
RedBoot[backup]  0xA0020000   0x8C010000  0x020000  0x8C010000
```

```
RedBoot config   0xA0FC0000   0xA0FC0000  0x020000  0x00000000
FIS directory    0xA0FE0000   0xA0FE0000  0x020000  0x00000000
junk             0xA0040000   0x8C400000  0x020000  0x80000000
RedBoot> fis delete junk
Delete image 'junk' - are you sure (y/n)? y
... Erase from 0xa0040000-0xa0060000: .
... Erase from 0xa0fe0000-0xa1000000: .
... Program from 0x8c7d0000-0x8c7f0000 at 0xa0fe0000: .
==================================================================
```

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

```
This command is used to write-protect [lock] a portion of flash
memory, to prevent accidental overwriting of images. In order to
make make any modifications to the flash, a matching unlock command
must be issued. This command is optional and will only be provided
on hardware which can support write-protection of the flash space.
Note: depending on the system, attempting to write to write-
protected flash may generate errors or warnings, or be benignly
quiet.
==================================================================
RedBoot fis lock -f 0xa0040000 -l 0x20000
... Lock from 0xa0040000-0xa0060000: .
==================================================================
```

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

```
This command is used to unlock a portion of flash memory forcibly,
allowing it to be updated. It must be issued for regions which have
been locked before the FIS can reuse those portions of flash.
==================================================================
RedBoot fis unlock -f 0xa0040000 -l 0x20000
... Unlock from 0xa0040000-0xa0060000: .
==================================================================
```

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

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

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

Persistent state [flash-based configuration] control

Additionally, if the platform has flash memory, certain control and configuration information used by RedBoot can be stored in flash. The details of what information is maintained in flash differs, based on the platform and the configuration. However, the basic operation used to maintain this information is the same. Using the 'fconfig' command, the information may be displayed and/or changed.

Usage:

```
RedBoot> fconfig [-l]
```

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 just a '.' (period) on the line. Entry of the character '^' will move the editing back to the previous item.

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

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

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


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

==========================================================================
RedBoot> fconfig
         ^^^^^^^
Run script at boot: false t
                           ^
     Boot script:
```

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

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

NOTE: The '^' 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.

Program execution

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.

The format of the 'go' command is:

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

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

The “-w” option gives the user <time> seconds before execution begins. The execution may be aborted by typing ^C on the console. This mode would typically be used in startup scripts.

Installation and testing:

Intel StrongARM EBSA-285-Building Test Cases to run with RedBoot

The default configuration for EBSA-285 at present always sends diagnostic output to the serial line; to use RedBoot to channel diagnostic output to GDB whether connected by net or serial, enable the configuration option

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

located here in the common HAL configuration tree:

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

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

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

Installation on the Intel StrongARM EBSA-285 development system and equivalents.

Here’s how to install RedBoot, using the redboot images you should find in `loaders/arm-ebsa285/` in your installation directory:

```
431497 Aug 9 15:28 redboot-ram.elf
184802 Aug 9 15:28 redboot-ram.srec
433104 Aug 9 15:29 redboot-rom.elf (this one is not used)
194732 Aug 9 15:29 redboot-rom.srec
```

Copy the two '.srec' files to /tftpboot or where-ever they have to be for your TFTP server.

Briefly, we use whatever boot flash image you have in place already (CygMon or an eCos stub ROM) along with GDB, to execute a RAM based version of RedBoot. That is used, in its command-line mode, to fetch a ROM-based boot image of RedBoot and write it into the flash memory. "Fetching" the image means TFTP from a server; the

image must be in S-Record format. We then reset the target, thus running the newly-installed boot image of RedBoot. That in turn is used, in its command-line mode, to fetch a RAM-based boot image of RedBoot and write it into a different area of the flash memory, in order to make it easier to do the first part (running a RAM-based RedBoot in order to update the boot block) again in future.

Alternatively you can make a plain binary from the redboot-rom.elf and "blow" that into the boot flash using the means of your choice, as with previous systems.

1. Load a RedBoot, built for RAM startup, into RAM using existing GDB stubs. Note: do not run it yet!

```
% arm-elf-gdb -nw loaders/arm-ebsa285/redboot-ram.elf

GNU gdb 4.18-ecos-99r1-991015
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it
under certain conditions. Type "show copying" to see the
conditions. This version of GDB is supported for customers of
Cygnus Solutions. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu
--target=arm-elf"...(no debugging symbols found)...
(gdb) set remotebaud 38400
(gdb) tar rem /dev/ttyS0
Remote debugging using /dev/ttyS0
0x41000838 in ?? ()
(gdb) load
Loading section .rom_vectors, size 0x44 lma 0x20000
Loading section .text, size 0xf06c lma 0x20044
Loading section .rodata, size 0x19a8 lma 0x2f0b0
Loading section .data, size 0x474 lma 0x30a58
Start address 0x20044 , load size 69324
Transfer rate: 25208 bits/sec.
(gdb) detach
Ending remote debugging.
(gdb) q
```

2. Execute RedBoot from RAM, and initialize the flash filing system.

NOTES:

The key here is the "-o" option which keeps minicom from sending junk.

The magic phrase "$c#63" is important: you must type it in exactly thus. It is the packet which a "continue" command in GDB would send to the target. If you get no response, try "+$c#63" instead. The IP and server info comes from BOOTP, which is how this RedBoot will start up if the flash does not contain good config info.

```
% minicom -o ttyS0
$c#63
RedBoot(tm) debug environment - built 07:45:57, Aug 7 2000
Copyright (C) 2000, Red Hat, Inc.

RAM: 0x00000000-0x01000000
FLASH: 0x41000000 - 0x41400000, 16 blocks of 0x00040000 bytes ea.
IP: 192.168.1.25, Default server: 192.168.1.101
```

```
show tcp = 0x00030d48
RedBoot fi init
About to initialize [format] FLASH image system - are you sure (y/
n)? y
*** Initialize FLASH Image System
Warning: device contents not erased, some blocks may not be usable
... Erase from 0x413c0000-0x41400000: .
... Program from 0x00fb0000-0x00fb0400 at 0x413c0000: .
```

3. Program RedBoot image into FLASH: This expects the file redboot-rom.srec to exist in the TFTP server space on the server that answered the BOOTP request.

```
Redboot> lo -v redboot-rom.srec -b 0x00100000
Address offset = bf100000
Entry point: 0x41000044, address range: 0x41000000-0x41011384
Redboot> fi cr RedBoot -f 0x41000000 -b 0x00100000 -l 0x20000
An image named 'RedBoot' exists - are you sure (y/n)? y
... Erase from 0x41000000-0x41020000: .
... Program from 0x00100000-0x00120000 at 0x41000000: .
... Erase from 0x413c0000-0x41400000: .
... Program from 0x00fb0000-0x00ff0000 at 0x413c0000: .
RedBoot>
****reset the board here, leaving your terminal program
connected****
RedBoot(tm) debug environment - built 07:47:35, Aug 7 2000
Copyright (C) 2000, Red Hat, Inc.

RAM: 0x00000000-0x01000000
FLASH: 0x41000000 - 0x41400000, 16 blocks of 0x00040000 bytes ea.
IP: 192.168.1.25, Default server: 192.168.1.101
show tcp = 0x000082f0
RedBoot>
```

4. Install RAM based RedBoot for backup/update: Similar considerations apply: redboot-ram.srec must be an S-record version of RedBoot built for RAM startup.

```
RedBoot> lo -v redboot-ram.srec
Entry point: 0x00020044, address range: 0x00020000-0x00030ecc
RedBoot> fi cr RedBoot[backup] -f 0x41040000 -b 0x20000 -r 0x20000
-l 0x20000
An image named 'RedBoot[backup]' exists - are you sure (y/n)? y
... Erase from 0x41040000-0x41060000: .
... Program from 0x00020000-0x00040000 at 0x41040000: .
... Erase from 0x413c0000-0x41400000: .
... Program from 0x00fb0000-0x00ff0000 at 0x413c0000: .
RedBoot>
```

You have now updated your board completely.

5. To update RedBoot with a new version of RedBoot, it is necessary to run a RAM-based version of RedBoot which itself re-writes the ROM-based one, because you can't re-write the code that is executing at the time.

```
Redboot> fi lo RedBoot[backup]
Redboot> g
+
RedBoot(tm)
debug environment - built 07:45:57, Aug 7 2000
Copyright (C) 2000, Red Hat, Inc.
```

```
RAM: 0x00000000-0x01000000
FLASH: 0x41000000 - 0x41400000, 16 blocks of 0x00040000 bytes ea.
IP: 192.168.1.25, Default server: 192.168.1.101
show tcp = 0x00030d48
Redboot>
```

.. continue with step 3, using whatever your new boot image is called in the TFTP-place, in .srec format.

You probably also want to set up then environment with your own IP addresses and so on. Recall that this IP address is the one you use for GDB to talk to the board, not the IP address which the eCos application will take on (by BOOTP/DHCP or whatever means according to configury as usual).

```
Redboot> fconfig
Network debug at boot time: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.25
Default server IP address: 192.168.1.101
GDB connection port: 1000
Run script at boot: false
Redboot>
```

Building RedBoot

To rebuild RedBoot from source, first cut out the attached configuration export file "redboot.RAM" and save it somewhere, say /tmp/redboot.RAM

```
mkdir redboot
cd redboot
ecosconfig new ebsa redboot
ecosconfig import /tmp/redboot.RAM
ecosconfig tree
make
```

To build the ROM version, in a different build/config directory, just change the startup mode (at the end of the export file) and add an enable of CYGSEM_HAL_ROM_MONITOR, or use the attached config export redboot.ROM.

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

```
70456 ..../install/bin/redboot.bin
433104 ..../install/bin/redboot.elf
91407 ..../install/bin/redboot.img
194732 ..../install/bin/redboot.srec
```

The .elf and .srec files have the obvious relationship to those supplied in the "loaders/arm-ebsa285" directory in the install.

Here are the redboot.RAM and .ROM configuration exports:

```
================== redboot.RAM ========================
cdl_savefile_version 1;
cdl_savefile_command cdl_savefile_version {};
cdl_savefile_command cdl_savefile_command {};
cdl_savefile_command cdl_configuration { description hardware
template package };
cdl_savefile_command cdl_package { value_source user_value
wizard_value inferred_value };
```

```
cdl_savefile_command cdl_component { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_option { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_interface { value_source user_value
wizard_value inferred_value };

cdl_configuration eCos {
description "" ;
hardware ebsa285 ;
template redboot ;
package -hardware CYGPKG_HAL_ARM current ;
package -hardware CYGPKG_HAL_ARM_EBSA285 current ;
package -hardware CYGPKG_IO_PCI current ;
package -hardware CYGPKG_DEVS_ETH_ARM_EBSA285 current ;
package -hardware CYGPKG_IO_SERIAL_ARM_EBSA285 current ;
package -hardware CYGPKG_DEVICES_WATCHDOG_ARM_EBSA285 current ;
package -template CYGPKG_HAL current ;
package -template CYGPKG_INFRA current ;
package -template CYGPKG_REDBOOT current ;
package CYGPKG_IO_FLASH current ;
package CYGPKG_IO_ETH_DRIVERS current ;
package CYGPKG_DEVS_FLASH_EBSA285 current ;
};
cdl_option CYGBLD_BUILD_GDB_STUBS {
  user_value 0 };
cdl_option CYGDBG_DEVS_ETH_ARM_EBSA285_CHATTER {
  user_value 0 };
cdl_option CYGNUM_DEVS_ETH_ARM_EBSA285_DEV_COUNT {
  user_value 1 };
cdl_option CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS {
  inferred_value 1 };
cdl_option CYGDBG_HAL_DEBUG_GDB_CTRLC_SUPPORT {
  inferred_value 0 };
cdl_option CYGSEM_HAL_VIRTUAL_VECTOR_DIAG {
  user_value 1 };
cdl_option CYGSEM_HAL_USE_ROM_MONITOR {
  inferred_value 0 0 };
cdl_option CYGBLD_BUILD_REDBOOT {
  user_value 1 };
cdl_component CYG_HAL_STARTUP {
  user_value RAM };
================== redboot.RAM ====================================
================== redboot.ROM ====================================
cdl_savefile_version 1;
cdl_savefile_command
cdl_savefile_version {};
cdl_savefile_command
cdl_savefile_command {};
cdl_savefile_command
cdl_configuration { description hardware template package };
cdl_savefile_command cdl_package { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_component { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_option { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_interface { value_source user_value
wizard_value inferred_value };
```

```
cdl_configuration eCos {
  description "" ;
  hardware ebsa285 ;
  template redboot ;
  package -hardware CYGPKG_HAL_ARM current ;
  package -hardware CYGPKG_HAL_ARM_EBSA285 current ;
  package -hardware CYGPKG_IO_PCI current ;
  package -hardware CYGPKG_DEVS_ETH_ARM_EBSA285 current ;
  package -hardware CYGPKG_IO_SERIAL_ARM_EBSA285 current ;
  package -hardware CYGPKG_DEVICES_WATCHDOG_ARM_EBSA285 current ;
  package -template CYGPKG_HAL current ; package -template
CYGPKG_INFRA current ;
  package -template CYGPKG_REDBOOT current ;
  package CYGPKG_IO_FLASH current ;
  package CYGPKG_IO_ETH_DRIVERS current ;
  package CYGPKG_DEVS_FLASH_EBSA285 current ;
};
cdl_option CYGBLD_BUILD_GDB_STUBS {
  user_value 0 };
cdl_option CYGDBG_DEVS_ETH_ARM_EBSA285_CHATTER {
  user_value 0 };
cdl_option CYGNUM_DEVS_ETH_ARM_EBSA285_DEV_COUNT {
  user_value 1 };
cdl_option CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS {
  inferred_value 1 };
cdl_option CYGDBG_HAL_DEBUG_GDB_CTRLC_SUPPORT {
  inferred_value 0 };
cdl_option CYGSEM_HAL_VIRTUAL_VECTOR_DIAG {
  user_value 1 };
cdl_option CYGSEM_HAL_USE_ROM_MONITOR {
  inferred_value 0 0 };
cdl_option CYGBLD_BUILD_REDBOOT {
  user_value 1 };
cdl_component CYG_HAL_STARTUP {
  user_value ROM };
cdl_option CYGSEM_HAL_ROM_MONITOR {
  user_value 1 };
================ redboot.ROM ====================================
```

Installation on the the Cirrus Logic ARM-based EDB7xxx evaluation boards.

Copy the two '.srec' files to /tftpboot or where-ever they have to be for your tftp server.

Briefly, we use whatever boot flash image you have in place already (CygMon or an eCos stub ROM) along with GDB, to execute a RAM based version of RedBoot. That is used, in its command-line mode, to fetch a ROM-based boot image of RedBoot and write it into the flash memory. "Fetching" the image means tftp from a server; the image must be in S-Record format. We then reset the target, thus running the newly-installed boot image of RedBoot.

That in turn is used, in its command-line mode, to fetch a RAM-based boot

image of RedBoot and write it into a different area of the flash memory, in order to make it easier to do the first part (running a RAM-based RedBoot in order to update the boot block) again in future.

1. Program RedBoot image into FLASH:

Using the 'dl_edb7xxx' tool (or it's Windows equivalent), program the file 'edb7212_redboot_ROM.bin' into the flash.

2. Execute RedBoot from ROM, and initialize the flash filing system.

Notes: the key here is the "-o" option which keeps minicom from sending junk.

```
% minicom -o ttyS0

RedBoot(tm) debug environment - built 08:36:10, Aug 15 2000
Copyright (C) 2000, Red Hat, Inc.

RAM: 0x00000000-0x00fd7000
FLASH: 0xe0000000 - 0xe1000000, 128 blocks of 0x00020000 bytes ea.
IP: 192.168.1.23, Default server: 192.168.1.101
show tcp = 0x0001ed58
RedBoot>

RedBoot> fi in
About to initialize [format] FLASH image system - are you sure (y/
n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be
usable
... Erase from 0xe0fe0000-0xe1000000: .
... Program from 0x00fa7000-0x00fa7400 at 0xe0fe0000: .
```

3. Install RAM based RedBoot for backup/update: Similar considerations apply: redboot-ram.srec must be an S-record version of RedBoot built for RAM startup.

```
RedBoot> lo -v edb7212_redboot.srec
Entry point: 0x00010044, address range: 0x00010000-0x0001ee88
RedBoot> fi cr RedBoot[backup] -f 0xe0020000 -b 0x10000 -l 0x20000
An image named 'RedBoot[backup]' exists - are you sure (y/n)? y
... Erase from 0xe0020000-0xe0040000: .
... Program from 0x00010000-0x00030000 at 0xe0020000: .
... Erase from 0xe0fe0000-0xe1000000: .
... Program from 0x00fa7000-0x00fc7000 at 0xe0fe0000: .
RedBoot> fi li
Name              FLASH addr   Mem addr    Length    Entry point
RedBoot           0xE0000000   0xE0000000  0x020000  0xE0000044
RedBoot[backup]   0xE0020000   0xE0020000  0x020000  0x00010044
RedBoot config    0xE0FC0000   0xE0FC0000  0x020000  0x00000000
FIS directory     0xE0FE0000   0xE0FE0000  0x020000  0x00000000
RedBoot>
```

You have now updated your board completely.

4. To update RedBoot with a new version of RedBoot, it is necessary to run a RAM-based version of RedBoot which itself re-writes the ROM-based one, because you can't re-write the code that is executing at the time.

```
RedBoot> fi lo RedBoot[backup]
RedBoot> g
+
RedBoot(tm) debug environment - built 07:45:57, Aug  7 2000
Copyright (C) 2000, Red Hat, Inc.

RAM: 0x00000000-0x00fd7000
FLASH: 0xe0000000 - 0xe1000000, 128 blocks of 0x00020000 bytes ea.
IP: 192.168.1.25, Default server: 192.168.1.101
show tcp = 0x00030d48
RedBoot>
```

.. continue with step 3, using whatever your new boot image is called in the tftp-place, in .srec format.

You probably also want to set up then environment with your own IP addresses and so on. Recall that this IP address is the one you use for GDB to talk to the board, not the IP address which the eCos application will take on (by bootp/dhcp or whatever means according to configury as usual).

```
RedBoot> fconfig
Network debug at boot time: false
Use BOOTP for network configuration: false
Local IP address: 192.168.1.25
Default server IP address: 192.168.1.101
Network hardware address [MAC]: 0x08:0x88:0x12:0x34:0x56:0x79
GDB connection port: 1000
Run script at boot: false
RedBoot>
```

Building RedBoot

To rebuild RedBoot from source, first cut out the attached configuration export file "redboot.RAM" and save it somewhere, say /tmp/redboot.RAM

```
mkdir redboot
cd redboot
ecosconfig new edb7212 redboot
ecosconfig import /tmp/redboot.RAM
ecosconfig tree
make
```

To build the ROM version, in a different build/config directory, just change the startup mode (at the end of the export file).

Here is the redboot.RAM configuration export:

```
==================== redboot.RAM ================================
cdl_savefile_version 1;
cdl_savefile_command cdl_savefile_version {};
cdl_savefile_command cdl_savefile_command {};
cdl_savefile_command cdl_configuration { description hardware
template package };
cdl_savefile_command cdl_package { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_component { value_source user_value
wizard_value inferred_value };
cdl_savefile_command cdl_option { value_source user_value
```

```
wizard_value inferred_value };
cdl_savefile_command cdl_interface { value_source user_value
wizard_value inferred_value };

cdl_configuration eCos {
    description "" ;
    hardware    edb7212 ;
    template    redboot ;
    package -hardware CYGPKG_HAL_ARM current ;
    package -hardware CYGPKG_HAL_ARM_EDB7XXX current ;
    package -hardware CYGPKG_DEVS_ETH_ARM_EDB7XXX current ;
    package -hardware CYGPKG_IO_SERIAL_ARM_EDB7XXX current ;
    package -template CYGPKG_HAL current ;
    package -template CYGPKG_INFRA current ;
    package -template CYGPKG_REDBOOT current ;
    package CYGPKG_IO_ETH_DRIVERS current ;
    package CYGPKG_IO_FLASH current ;
    package CYGPKG_DEVS_FLASH_EDB7XXX current ;
};

cdl_option CYGBLD_BUILD_GDB_STUBS {
    user_value 0
};

cdl_option CYGDBG_HAL_COMMON_INTERRUPTS_SAVE_MINIMUM_CONTEXT {
    user_value 0
};

cdl_option CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS {
    inferred_value 1
};

cdl_option CYGDBG_HAL_DEBUG_GDB_CTRLC_SUPPORT {
    inferred_value 0
};

cdl_option CYGSEM_HAL_USE_ROM_MONITOR {
    inferred_value 0 0
};

cdl_option CYGHWR_HAL_ARM_EDB7XXX_VARIANT {
    user_value EP7212
};

cdl_option CYGHWR_HAL_ARM_EDB7XXX_PROCESSOR_CLOCK {
    user_value 73728
};

cdl_option CYGBLD_BUILD_REDBOOT {
    user_value 1
};

cdl_component CYG_HAL_STARTUP {
    user_value RAM
};
==================== redboot.RAM ================================
```