



DeviceMaster™ RTS Development Kit

How to Use this Document

Red underscored text link to Internet URLs. Blue underscored text link to sections within this document or to another document on the media. If you copy this document from the ftp/web or CD, make sure that you maintain the file structure or you will receive link errors.

Introduction

The DeviceMaster RTS Development Kit (DMDK) includes the basic tools required to develop eCos applications for the Control DeviceMaster RTS platform. It is assumed that the reader has some experience with embedded software development and is familiar with software development tools under the host environment (Windows/Cygwin or Linux). If you are unfamiliar with the tools mentioned in the DMDK documentation, the following references may be helpful:

- **Cygwin:**
located in the [WindowsTools/Cygwin/Doc/](#) directory or at the following sites:
 - <http://sources.redhat.com/cygwin/>
 - <http://sources.redhat.com/cygwin/cygwin-ug-net/cygwin-ug-net.html>
 - <http://sources.redhat.com/cygwin/faq/>
- **bash:**
 - <http://www.oreilly.com/catalog/bash2/>
 - <http://www.gnu.org/manual/bash/>
 - <http://www.slug-vt.org/bash.html>
- **make:**
<http://www.gnu.org/manual/make/>
- **tar:**
<http://www.gnu.org/manual/tar/>

Contents of the CD

The DMDK distribution contains the following directories:

- **eCos** contains both source and object distributions of eCos. This version of eCos is based on a CVS snapshot taken in December 2000 with some fixes and enhancements.
- **eCos Manuals** contains RedHat eCos manuals in PDF format. HTML version of the manuals are available online at <http://sources.redhat.com/ecos>. The available documents include:
 - [ecos-tutorial-arm.pdf](#) – Introduction to eCos and a tutorial on installing and using eCos and ARM tools.
 - [user-guides.pdf](#) – Users guide for eCos configuration utilities, the eCos package repository, and various tool options.
 - [ecos-ref.pdf](#) – Reference manual for eCos kernel and libraries APIs.
 - [ecos-tcpip.pdf](#) – Documentation on eCos TCP/IP and SNMP packages.
 - [Cmds_overview.pdf](#) – A summary of DeviceMaster RTS RedBoot (bootloader) commands.
 - [redboot.pdf](#) – RedBoot (bootloader) users manual.
 - [cdl-guide.pdf](#) – How to add components to the eCos configuration database. Only needed if you want to add a package to eCos or modify the structure of the configuration database.
- **eCos_Config_Tool** contains binaries for both Linux and Windows eCos configuration utilities.
- **Linux_Tools** contains RPM and SRPM files for ARM-ELF cross development toolchains.
- **Windows_Tools** contains a complete Cygwin binary distribution and binaries for ARM-ELF cross-development tools for Cygwin.

- **GNU_Docs** contains manuals in PDF format for the GNU development tools:
 - [gcc.pdf](#) – GNU C compiler manual.
 - [cpp.pdf](#) – GNU C pre-processor manual.
 - [gdb.pdf](#) – GNU debugger manual.
 - [as.pdf](#) – GNU assembler manual.
 - [ld.pdf](#) – GNU linker manual.
 - [bfd.pdf](#) – Documentation for the GNU binary file manipulation library.
 - [standards.pdf](#) – GNU project coding standards.
- **DeviceMaster_Apps** contains sources for two example DeviceMaster applications:
 - GoAhead [web-server](#) demonstration application with minimal changes required to build and run on RTS platform.
 - Control SocketServer application (which also includes the GoAhead web-server) that is shipped in flash ROM as the default application.
- **Sample_Apps** contains sources for sample eCos applications that demonstrate how to use Control's serial and Ethernet drivers.
- **Other_Docs** contains miscellaneous other documentation

Installing

In order to develop eCos applications, the user must install at least the ARM-ELF tools and the pre-compiled eCos libraries. The Windows ARM-ELF tools require that Cygwin be installed. If the user would like to modify the eCos configuration, then the eCos sources must be installed. Changes in eCos configuration require that the eCos libraries and include files be rebuilt from the eCos source tree.

Building at least one of the included sample applications is also suggested as a way to insure that the tools and libraries are operational. [Building a Program](#) on Page 3 shows how to install, build, and run the **serecho** program that is included in the **Sample_Apps** directory.

ARM-ELF Tools

GNU cross development tools (compiler, linker, assembler, debugger) are included for IA32 hosts running either Cygwin/Win32 or Linux. A complete Cygwin distribution is also included. Instructions for installing toolchains are in the files

- [Linux Tools/README.txt](#)
- [Windows Tools/README.txt](#)

Pre-Compiled eCos Libraries

The DMDK distribution contains a pre-compiled version of eCos configured for the DeviceMaster RTS. You may use this library to develop eCos applications without having to build eCos from its sources. Installation instructions and the pre-compiled libraries are in the following files:

- [eCos/Object/README.txt](#)
- [eCos/Object/install.tar.gz](#)

eCos Sources

The complete sources used to build the above eCos binaries and the configuration file that was used to build the above eCos binaries are in this directory. It also contains instructions for configuring and building eCos and a list of modifications that have been made to eCos since it was checked out of the RedHat eCos CVS repository.

- [eCos/Source/ecos.tar.gz](#) contains eCos source tree including Control serial and Ethernet drivers.
- [eCos/Source/ecos.ecc](#) is the eCos configuration used to build binaries included in the DMDK distribution.
- [eCos/Source/README.txt](#) contains instructions for building eCos for DM-RTS from the source tree.
- [eCos/Source/changes.txt](#) is the change list.

Sample Applications

Sample application sources are included in two directories: **DeviceMaster_Apps** (which contains two “full-up” applications that both include web servers) and **Sample_Apps** (which contains two small example applications that show how to use Control Corporation’s Ethernet and serial drivers). Detailed instructions on building and running one of the latter are in the [Building a Program](#) subsection.

Instructions for building sample applications are in:

- [DeviceMaster_Apps/SocketServer/Source/README.txt](#)
- [DeviceMaster_Apps/WebServer/Source/README.txt](#)
- [Sample_Apps/Source/README.txt](#)

Documentation for the Control serial and Ethernet drivers, and a commentary on the two small sample applications **netecho** and **serecho** are in:

- [Sample_Apps/Doc/nserial.pdf](#)
- [Sample_Apps/Doc/ether.pdf](#)

Hints and Tips

line-endings

Do not convert source or configuration files to MS-DOS line endings. Cygwin users: use a text editor that preserves UNIX line endings (jed, vim, PFE, textpad, VisualStudio, Wordpad, etc.). **Do not** use Notepad to edit source files or configuration files.

printf()

The **stdio printf()** function is not currently supported by the DMRTS HAL package. Use **diag_printf()** instead.

JTAG Debugging

Make sure that Pins 1 and 2 of on the three pin header are shorted with a jumper. The boards are shipped with the jumper on Pins 2 and 3. Move the jumper to Pins 1 and 2.

Printing eCos Manuals

If you want to print eCos manuals using Adobe Acrobat reader on an HP duplex printer, and you want the odd/even margins to line up, try unchecking “fit to page” and specifying a page size of 9.75 by 13 inches.

Disabling Auto-Load

If you want to prevent the bootloader on the DeviceMaster from loading and running the default application on start-up, follow the steps in [Sample Applications](#) on Page 3.

Building a Program

This section shows how to install and build two of the demo applications contained in the DMDK distribution. These demo programs are small example applications that show how to use the Control serial and Ethernet drivers included with the DMDK as eCos packages. The two demo programs will be built using the pre-compiled eCos object libraries and their associated include files.

The other demonstration applications can be built by following the instructions in the README.txt files listed in the [Sample Applications](#) subsection.

Install the Libraries and Sample Programs

Copy the gzipped tar files containing the pre-compiled eCos libraries ([install.tar.gz](#)) and the demo program source files ([demo.tar.gz](#)) to your hard disk. In this example, they were copied to the user’s home directory.

1. Create a work directory:

```
$ mkdir ecos-demo
$ cd ecos-demo
```

2. Unpack the pre-compiled eCos libraries:

Note: *If you already have the eCos libraries installed, or if you built them from sources, you can skip this step, but you must modify the **Makefile** to point to location of the eCos install directory.*

```
$ tar xzf ~/install.tar.gz
```

3. Unpack the demo sources

```
$ tar xzf ~/demo.tar.gz
```

Build the Sample Programs

```
$ cd demo
$ make clean
rm -f netecho serecho *.o *~ *.lst *.map \#\#\# *.bin *.elf10 core *.bak srcdeps
touch srcdeps
$ make depend
```

Generating source dependencies

```
arm-elf-gcc -mcpu=arm7tdmi -mbig-endian -fverbose-asm -g -Wa,-ahln=.lst -O0 -D__ECOS -I../install/include -M netecho.c serecho.c >>srcdeps
```

```
$ make
```

```
arm-elf-gcc -mcpu=arm7tdmi -mbig-endian -c -o netecho.o -fverbose-asm -g -Wa,-ahln=netecho.lst -O0 -D__ECOS -I../install/include -Wall netecho.c
arm-elf-as --gstabs -EB -m arm7tdmi -amhlsnd=boot.lst -o boot.o boot.s
arm-elf-gcc -mcpu=arm7tdmi -mbig-endian -Wl,-Map,netecho.map -g -Wl,--gc-sections -nostartfiles -L../install/lib -o netecho netecho.o boot.o -Ttarget.ld -nostdlib
arm-elf-gcc -mcpu=arm7tdmi -mbig-endian -c -o serecho.o -fverbose-asm -g -Wa,-ahln=serecho.lst -O0 -D__ECOS -I../install/include -Wall serecho.c
arm-elf-gcc -mcpu=arm7tdmi -mbig-endian -Wl,-Map,serecho.map -g -Wl,--gc-sections -nostartfiles -L../install/lib -o serecho serecho.o boot.o -Ttarget.ld -nostdlib
```

Using the Diagnostic Serial Port

The DeviceMaster RTS circuit board has two diagnostic serial ports. Each port is accessed via 4-Pin header near one end of the board. The signal levels on the headers are not RS-232, and a converter cable is required to convert them to RS-232 levels for use with a standard serial port.

Note: The convertor cable is included with the Developer's Kit.

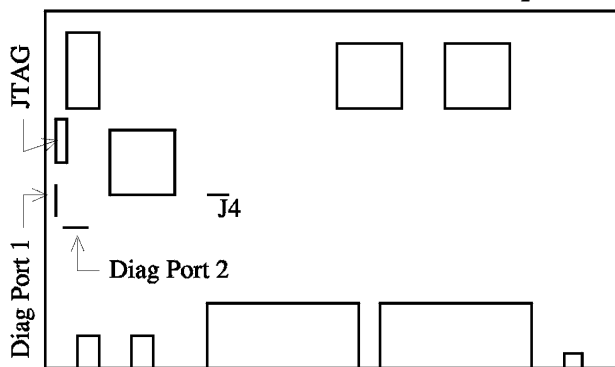
The first diagnostic port is used for diagnostic output from calls to `diag_printf()`. It is also used by [RedBoot](#) as a command console.

The second diagnostic port also functions as a RedBoot console port. The second port can be used for a serial *remote* GDB connection, see [Using GDB with Diagnostic Port 2](#) on Page 12. The positions of these connectors are shown in the figures at the right.

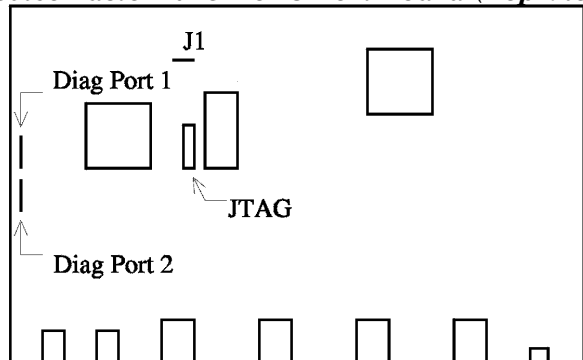
Use the following procedure to set up the diagnostic serial port:

1. Plug the diagnostic cable into the first port and connect it to a serial port configured for:
 - 57600 baud
 - 8 data bits
 - no parity
 - 1 stop bit
 - No flow control
2. Connect a null-modem cable from an available COM port on your PC to **Port 1** on the DeviceMaster RTS.

DeviceMaster RTS 16-Port Board (Top View)



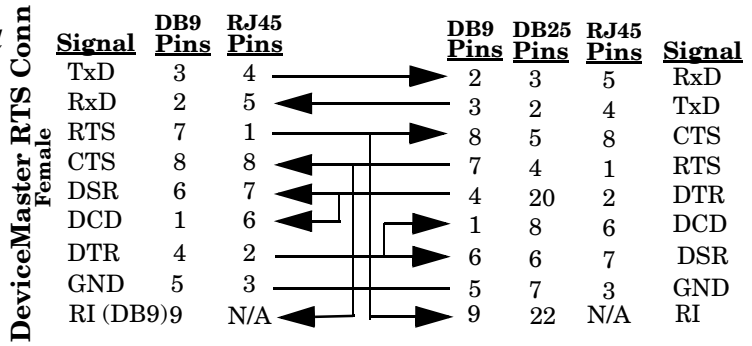
DeviceMaster RTS 4 or 8-Port Board (Top View)



Note: You may want to purchase or build a straight-through cable and purchase a null-modem adapter.

- If you monitor the data from that port and cycle power on the DeviceMaster RTS, you should see startup messages from the bootloader and then a bootloader prompt:

```
RAM OK
Sum OK
+
Control DeviceMaster RTS Boot
Version 1.06
```



```
ks32c5000_eth_init()
  found MAC address 00:C0:4E:0B:FF:F9
ks5000_ether: installInterrupts()
EthInit(00:C0:4E:0B:FF:F9)
ks32C5000 eth: 00:C0:4E:0B:FF:F9 Hardware CRC
ks32c5000_eth_start()
IP: 10.0.0.5, Mask: 255.0.0.0, Gateway: 10.0.0.1, Server: 0.0.0.0
RedBoot>
```

- At this point you can type bootloader commands. If a default application is configured in flash, and the default application time-out is set (factory setting: 10 seconds), you will see the default application start up after the time-out period:

```
ARM DeviceMaster HAL (no virtual vectors)
AIOPIC serial driver: 16 channels
ks32c5000 old_init_handler()
Network stack using 262144 bytes for misc space
                    262144 bytes for mbufs
                    524288 bytes for mbuf clusters
ks32c5000_eth_init()
  found MAC address 00:C0:4E:0B:FF:F9
ks5000_ether: installInterrupts()
EthInit(00:C0:4E:0B:FF:F9)
ks32C5000 eth: 00:C0:4E:0B:FF:F9 Hardware CRC
SocketServer 1.13
Copyright Control Corp. 2001
Build date: Tue Jun 19 12:21:54 CDT 2001

Free RAM at 280700, Len = 53F900
starting madmin
Bg started
madmin running
madmin MyMacAddr=00:C0:4E:0B:FF:F9
ks32c500 old_set_config_handler()
ks32c5000_eth_start()
  IP = 0A000005
  Mask = FF000000
  Gate = 0A000001
```

```
Net init complete.
Net init done
socket server init
socket server active config: 05002000
TcpRx0: waiting on semaphore
TcpTx0 started
[...]
```

Downloading a Program

There are several methods available for downloading and running a DeviceMaster RTS application:

- RedBoot
 - TFTP via Ethernet (binary or S-Record file)
 - X-Modem via RS-232 (binary or S-Record file)
- DeviceMaster Utility (binary or S-Record file, Windows only)
- **nslinkadmin** program (binary file, Linux only)
- GDB via JTAG interface
- GDB via *remote* protocol on Diag Port 2

Each of these will be described. It is also possible to download a program and save it in flash ROM so that it is executed automatically when the DeviceMaster RTS starts.

Disabling Auto-Load

When shipped, the DeviceMaster RTS will be configured so that RedBoot waits for 10 seconds after start-up before loading and running the default application from flash ROM. Since several of the download methods require that RedBoot be running, it may be convenient to disable auto-loading of the default application.

You may configure RedBoot to not load a default application from flash ROM either of two ways:

1. At the RedBoot command prompt, disable default application loading by setting the time-out parameter to zero:

```
RedBoot> timeout 0
Timeout 0 seconds
RedBoot>
```

2. Delete the **default** application from flash using the **fis delete** command at the RedBoot prompt:

```
RedBoot> fis delete default
Delete image 'default' - are you sure (y/n)? y
... Erase from 0x05030000-0x050c0000: .....
... Erase from 0x053f0000-0x05400000: .
... Program from 0x007a0000-0x007b0000 at 0x053f0000: .
+
RedBoot>
```

Download Using RedBoot

The DeviceMaster RTS platform includes a customized version of the [RedBoot](#) bootloader from RedHat. RedBoot is described in detail in [The RedBoot Bootloader](#) on Page 13. RedBoot can be used to download application programs using either TFTP (via Ethernet) or X-Modem (via serial port). Examples of loading using the RedBoot **load** command can be found in [Loading a File](#) on Page 17. Once the file is loaded, it may be run using the RedBoot **go** command as shown in [Executing a Program](#) on Page 18.

For more information on using RedBoot on the DeviceMaster RTS platform, see the documents listed below:

- [Cmds overview.pdf](#) – Summary of DeviceMaster RTS RedBoot commands.
- [redboot.pdf](#) – RedBoot users manual.

Download Using One of the DeviceMaster RTS Utilities

There are two DeviceMaster RTS administration utilities (Linux and Windows) that can also be used to download and run a DeviceMaster RTS applications. Instructions for using the utilities to download are found in the help files.

Download Using `nslinkadmin`

The `nslinkadmin` utility that is included with the Linux NS-Link driver can be used to download and run a DeviceMaster RTS application that has been converted to a binary file with an entry point at Address 0. The `nslinkadmin` program must run as root in order to access an Ethernet interface in raw mode. Therefore, you must be logged in as root or have `nslinkadmin` `suid` root. There are two ways to run `nslinkadmin`: interactive mode and command-line mode. For more information, see the `nslinkadmin(8)` manual page.

Interactive Mode

If your DeviceMaster RTS is connected to an Ethernet interface other than `eth0`, use the `-d` option to specify the interface:

```
# /usr/sbin/nslinkadmin -d eth1
Using network device 'eth1'
Checking network for any possible remotes.
  1) 00:c0:4e:0b:ff:f9 DeviceMaster      (idle,free)
  q) quit
->_
```

If you have more than one DeviceMaster RTS, RocketPort Serial Hub *Si* or RocketPort Serial Hub *ia* connected to the Ethernet network, the list will show each DeviceMaster RTS (or RPSH) device found. Select the appropriate device by entering the corresponding number:

```
->1
Other Control Hub: 00:c0:4e:0b:ff:f9
      IP Utility
?          Display help information.
ig         Get/display flash IP config
ic         Get/display current IP config
is <addr> <mask> <gate> Set IP info
ie         Erase IP info
l          Load the remote.
s          Start the remote.
z          Shortcut for l, s
r          Reset the remote.
x <filename> Load and start the specified file.
q          Quit the test.
->_
```

To download and start a DeviceMaster RTS application, enter `x` followed by the file name of the binary file:

```
->x serecho.bin
Loading remote
.....
Starting remote
->_
```

While loading, a “.” will be displayed for each packet downloaded. Once the file is downloaded, it will be started automatically. Once the application is started, NS-Link will no longer be able to communicate with the DeviceMaster RTS. To exit `nslinkadmin`, enter `q` and press the return key.

Command Line Mode

Downloading using `nslinkadmin` in non-interactive command-line mode is done by specifying command line switches:

- **-d <ethdev>** controls the ethernet interface used: **eth0** (default), **eth1**, etc.
- **-e <macaddr>** specifies the Ethernet address of the DeviceMaster RTS device in `xx:xx:xx:xx:xx:xx` format.
- **-f <binfile>** specifies the path of the file to be downloaded.
- **-l** tells `nslinkadmin` to load and run the file.

For example:

```
nslinkadmin -d eth1 -e 00:c0:4e:0b:ff:f9 -l -f serecho.bin
```

Using GDB with JTAG

This subsection will show how to use **GDB** to run a program via the **JTAG** interface. The `.gdbinit` file that is included in the demo application package assumes that we are using a **JTAG** interface that uses the **RDI** protocol over **UDP/IP** (such as the [EPI Jeeni](#)), and that host name for the **JTAG** interface is **jeeni**. If you are using a serial or serial/parallel connection to your **JTAG** interface, or the host name for your **JTAG** interface is not **jeeni**, you will have to edit the `.gdbinit` file and modify the **target** command before running **GDB**.

To use **JTAG** debugging, make sure that the shorting jumper is placed on Pins 1 and 2 of the three pin header on the DeviceMaster RTS board (J1 on 4/8-port boards, J4 on 16-port boards). Location of the jumpers is shown in [Using the Diagnostic Serial Port](#) on Page 4.

For notes on using **GDB** via **Diag Port 2**, see [Using GDB with Diagnostic Port 2](#).

Running GDB

First we will start **GDB** in non-window mode so that we can show the commands and their output.

1. If you are using the insight GUI for **GDB**, enter the commands shown below in the console window.

```
$ arm-elf-gdb -nw
GNU gdb 5.0
Copyright 2000 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.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i586-pc-linux-gnu --target=arm-elf".
JEENI (ADP,ARM7TDI,RST) Rev 2.2
Rebuilt on Jan 12 2001 at 14:22:34
SN=0102J069 ENET=00:80:CF:00:0C:CD IP=10.0.0.100 (255.255.0.0)
Connected to ARM RDI target.
(gdb)
```

2. Just to be safe, we'll run the **resetcpu** macro which will put the CPU into a known state:

```
(gdb) resetcpu
```

3. Load the **serecho** program using the **reload** macro.

```
(gdb) reload serecho
Warning: the current language does not match this frame.
Loading section .rom_vectors, size 0x44 lma 0x340
Loading section .text, size 0x27cf4 lma 0x384
Loading section .rodata, size 0x167f lma 0x28078
Loading section .data, size 0x1610 lma 0x296f8
Loading section .boot, size 0x4 lma 0x0
Start address 0x384 , load size 174539
Transfer rate: 465437 bits/sec, 505 bytes/write.
```

4. Set a temporary breakpoint at the `cyg_user_start()` function:

```
(gdb) tbreak cyg_user_start
Breakpoint 1 at 0x99c: file serecho.c, line 72.
```

5. Start execution with the `continue` command:

```
(gdb) c
Continuing.
cyg_user_start () at serecho.c:72
72      diag_printf("Entering cyg_user_start() function\n");
1: *(char *) 123764751 &= 252 = 128 '\200'
Current language: auto; currently c
```

We hit the breakpoint and stopped (at Line 72 in `serecho.c`). If we were using a GUI, the source window would show `serecho.c` with Line 72 highlighted.

6. Continue (no breakpoints are set, so it will run until we hit **Ctrl-C** to stop it).

```
(gdb) c
Continuing.
```

7. Hit **Ctrl-C**, and the emulator stops the processor:

```
JEENI: halt request
JEENI: halted
RDI_execute: you pressed Escape

Program received signal SIGINT, Interrupt.
idle_thread_main (data=1040807) at /opt/ecos/ecos-cvs/ecos/packages/kernel/current/src/
common/thread.cxx:1148
1148 /opt/ecos/ecos-cvs/ecos/packages/kernel/current/src/common/thread.cxx: No such
file or directory.
1: *(char *) 123764751 &= 252 = 128 '\200'
Current language: auto; currently c++
(gdb)
```

In the example, the program was stopped in the built-in eCos idle task. This is typical, since the program does not do much and will spend most of its time idle. Since we don't have the eCos source files installed, the debugger can not display the source file line and warns us that it can not find the source file.

8. If you had been watching the output from the diagnostic serial port, you should have seen something like this:

```
ARM DeviceMaster HAL (no virtual vectors)
AIOPIC serial driver: 16 channels
ks32c5000 old_init_handler()
Network stack using 262144 bytes for misc space
                    262144 bytes for mbufs
                    524288 bytes for mbuf clusters

ks32c5000_eth_init()
  found MAC address 00:C0:4E:0B:FF:F9
ks5000_ether: installInterrupts()
EthInit(00:C0:4E:0B:FF:F9)
ks32C5000 eth: 00:C0:4E:0B:FF:F9 Hardware CRC
Entering cyg_user_start() function
0: Beginning execution
2: Beginning execution
3: Beginning execution
4: Beginning execution
5: Beginning execution
6: Beginning execution
7: Beginning execution
8: Beginning execution
9: Beginning execution
```



```

27     set *0x7ffa004 = 0
28     set *0x7ffa008 = 0
29     set *0x7ffa010 = 0
30
31     set *0x7ff9000 = 0
32     set *0x7ff9004 = 0
33     set *0x7ff9008 = 0
34     set *0x7ff900c = 0
35
36     # reset UARTS
37     set *0x7ffc000 = 0
38     set *0x7ffc004 = 0
39     set *0x7ffd000 = 0
40     set *0x7ffd004 = 0
41
42     # disable/clear interrupts
43     set *0x7ff4000 = 0
44     set *0x7ff4008 = 0xffffffff
45     set *0x7ff4004 = 0xffffffff
46 end
47

```

Define the **memconfig** macro. The **memconfig** program is linked so that it runs in the 8K SRAM that is built into the Samsung uController. This will not work if the cache is enabled, so do a **resetcpu** first to disable cache.

```

48 # macro to load memory config program into SRAM and run it
49
50 define memconfig
51     delete
52     symbol-file memconfig
53     load memconfig
54     tbreak __memoryConfigDone
55     cont
56     symbol-file
57 end
58

```

Define a macro that loads a program and sets breakpoints on the interrupt vectors. Note the **display** command at the end of the macro – that will tell **GDB** to evaluate and display the expression ***(char*)0x760800f &= 0xfc** every time execution is stopped. That expression will disable the watchdog timer in the Dallas DS1511W – thus preventing the board from being reset while you’re trying to decide what to do next after a breakpoint has been hit.

```

59 define reload-with-break
60     resetcpu
61     delete
62     symbol-file
63     symbol-file $arg0
64     load $arg0
65     break *0x00
66     break *0x04
67     break *0x08
68     break *0x0c
69     break *0x10
70     break *0x14
71     break *0x18
72     break *0x1c
73     display *(char*)0x760800f &= 0xfc

```

```
74 end
75
```

The same macro without the breakpoints on the interrupt vectors.

```
76 define reload
77     resetcpu
78     delete
79     symbol-file
80     symbol-file $arg0
81     load $arg0
82     display *(char*)0x760800f &= 0xfc
83 end
84
85 # the "display *(char*)0x760800f &= 0xfc" will disable the
86 # watchdog timer in the Dallas Semi part whenever execution
87 # stops (when breakpoint is hit or when the user stops the
88 # program.) If we don't do this, the board will reset after
89 # a breakpoint has been hit.
90
```

Pick which set of register names you like:

```
91 set dis std
92
```

Make sure the debugger knows we're running big-endian:

```
93 set endian big
```

The **target** command tells the debugger what protocol (**rdi**) and communications link (ethernet) we are going to use.

```
94 # target rdi /dev/ttyS0 19200
95 # target rdi s=/dev/ttyS0,p=/dev/par0 19200
96 target rdi e=jeeni
97
```

Finally, force the CPU into a benign state using the macro we defined above:

```
98 resetcpu
99
```

Using GDB with Diagnostic Port 2

If you do not have a JTAG interface, it is possible to use the GDB stubs via the GDB *remote* protocol that is implemented by the RedBoot bootloader. Using GDB via Diag Port 2 is similar to using GDB via JTAG, but changes to the GDB initialization file (**.gdbinit** or **gdb.ini**) will need to be made. The differences are:

- Use the **target remote <dev>** command instead of the **target rdi** command. Replace **<dev>** with the device name of the serial port connected to the second diag port (e.g. **/dev/ttyS0**).
- Start GDB with the command line flag **\tt -b 57600** to specify the proper baud rate.
- Do not execute the **resetcpu** or **memconfig** macros that were shown in the sample **.gdbinit** file. The GDB stubs in the bootloader will insure that the processor and memory controller are in the proper states.
- When starting an eCos program, you must set a breakpoint and stop execution somewhere after the eCos initialization code. The **cyg_user_start** function is a good place. If you load an eCos program and do a **continue** without any breakpoints, you will not be able to stop execution of the eCos application. After you have stopped at a breakpoint and then resumed execution you can interrupt the eCos application and return control to the debugger at any time by pressing **control-C**, sending GDB a **SIGINT** signal, or pressing the **stop** button in Insight.

The easiest way to do this is to add the command **tbreak cyg_user_start if 0** to the end of the **reload** macro (or whatever macro you use to load an eCos application). This will place a temporary breakpoint at the **cyg_user_start** function, but when that breakpoint is reached, execution will continue immediately because the condition **if 0** is not true.

Saving a Program to Flash ROM

Once you have debugged your DeviceMaster RTS application, you may want to save it in the DeviceMaster RTS flash ROM so that it can be automatically executed when the DeviceMaster RTS starts up. If you do this, your application will replace the existing **SocketServer** application that is present in the DeviceMaster RTS flash ROM when it is shipped from the factory.

Note: *You can change back to the SocketServer default application if you want to later.*

The procedure for downloading a DeviceMaster RTS application and saving it to flash as the default application is shown in [Loading a File](#) on Page 17.

Note: *Alternative methods for loading using RedBoot are shown in [Default Application](#) on Page 19 and in [DeviceMaster Apps/SocketServer/Doc/Update SocketSVR.pdf](#).*

The RedBoot Bootloader

The DeviceMaster RTS boards run a bootloader based on RedHat's **RedBoot** program. In addition to the RPSH-Si compatible TCP and MAC mode network interfaces RedBoot provides a boot console interface that can be used to perform various functions:

- View/modify board configuration:
 - Model number
 - Board revision
 - MAC address
 - IP configuration (address/netmask/gateway)
 - Password
 - Autoload time-out
 - Telnet enable/disable
 - HTTP authentication method
- Load file via serial port ([xyz]modem) or Ethernet (TFTP).
- Load file from flash.
- Save file to flash.
- Test serial port.

Important Differences From Older Products

There are several differences between the DeviceMaster RTS and the RPSH-Si 2/4/8 port products:

RPSH-Si bootloader sends out a DHCP request and later passes the DHCP reply information to the downloaded application.

DeviceMaster bootloader sends out a DHCP request to obtain basic IP address information. When the application program starts it sends out another DHCP request further configuration information (netmask, name-server, etc.).

1. **RPSH-Si** bootloader will continue to send DHCP requests once per second if it gets no response. **DeviceMaster** bootloader will give up after a two attempts and disables IP networking.
2. **RPSH-Si 2-Port** does not implement DHCP lease renewal. **DeviceMaster** and RPSH-Si 4/8 port implements DHCP lease renewal.
3. **RPSH-Si** will always send DHCP requests unless a static IP address is configured. **DeviceMaster** can be configured to disable IP networking (no BOOTP/DHCP requests will be sent).

Note: *We have had complaints about DHCP requests from customers using MAC addressing mode.*

4. **RPSH-Si** ignores the server and filename fields in the BOOTP/DHCP response. **DeviceMaster** will attempt to load via TFTP the specified file from the specified server if a filename and server address is present in the BOOTP/DHCP response. If the file is loaded successfully, it will then be executed.

Board Configuration Commands

These functions are performed via a command-line interface that is accessible via external serial Port 0, the 4-pin debug header (57.6K, 8, none), or by telnet. When connected to the debug header or telnet you should see a **RedBoot>** prompt

Note: *The start-up messages are displayed only on the 4-Pin diagnostic head and will not be visible via telnet unless the **version** command is entered.*

The external console port will be disabled on power-up. In order to start console services on that port, the string **#!DM** must be the first thing received on that port after power-up. When that string has been seen, the port will be enabled and a prompt will be displayed.

A password is required for telnet access, but no password is required for serial port access.

```
Control DeviceMaster Boot Version 0.01
RedBoot(tm) debug environment - built 17:28:09, Mar  7 2001
Platform: Control DeviceMaster (ARM 7TDMI)
Portions Copyright (C) 2000, Red Hat, Inc.
Portions Copyright (C) 2001, Control Corp.
RAM: 0-7C0000
Id=0089,8897
FLASH: 0x05030000 - 0x05400000, 61 blocks of 0x00010000 bytes
each.
ks32C5000 eth: 00:C0:4E:0B:FF:FA  Hardware CRC
IP: 192.168.1.23, Default server: 0.0.0.0
RedBoot>
```

To see a list of available commands, type **help** followed by a carriage-return:

```
RedBoot> help
  auth [noaccess,none,basic,md5,invalid]
    Set/show web authentication
  boardrev [rev-number]
    Show/set Board revision
  cache [ON | OFF]
    Manage machine caches
  disable
    Disable autoload of default app
  dump -b <location> [-l <length>]
    Display (hex dump) a range of memory
  fis {cmds}
    Manage FLASH images
  go [-w <timeout>] [entry]
    Execute code at a location
  help
    Display list of bootloader commands
  ip [addr mask gateway]
    Show/set IP address config
  load [-r] [-v] [-h <host>] [-m {TFTP | xyzMODEM | direct}] [-b <base_address>]
<file_name>
    Load a file
  loop 232|422|int port-number
    Run loopback test on port
  mac [XX XX XX XX XX XX XX]
    Show/set Ethernet MAC address
  model [model-number]
    Show/set Model number
  password [password]
    Set/Delete password
```

```

reset
    Reset the system
telnet [disable|enable]
    Set/Show telnet server enable
terse
    Terse command response mode
t485 port-number1 port-number2
    Run port-to-port 485 test
timeout [seconds]
    Show/set bootloader timeout
version
    Display RedBoot version information

```

The board configuration commands are **ip**, **mac**, **model**, **boardrev**, **password**, **auth**, **timeout**, **telnet**. Typing the command will display the current value of that configuration item. The command with a parameter will set the configuration item value.

Note: For security reasons, the password command will not display the current password. If the password command is used with no parameters, the password will be set to the empty string.

Model Number

The model number is used by applications to determine what hardware features are available on the board. If the model number is set incorrectly, applications may not function correctly.

```

RedBoot> model
Model 5002120
RedBoot> model 5002113
Model 5002113
RedBoot>

```

IP Configuration

Setting the IP address to 255.255.255.255 will disable IP networking. Setting the IP address to 0.0.0.0 will cause the bootloader to use BOOTP to request an IP address.

```

RedBoot> ip
IP Config: IpAddr=192.168.1.23 IpMask=255.255.255.0 IpGate=192.168.1.1
RedBoot> ip 10.23.4.12 255.255.0.0 10.23.1.1
IP Config: IpAddr=10.23.4.12 IpMask=255.255.0.0 IpGate=10.23.1.1
RedBoot>

```

If only one parameter is provided to the **IP** command, the IP address value will be changed and the mask and gateway will be unaffected. Changes to IP configuration will take effect after reset.

MAC address

Sets or displays the MAC (Ethernet) address. Changes will take effect after reset.

```

RedBoot> mac
MAC: 00 C0 4E 0B FF FA
RedBoot> mac 00 c0 4e 0c 34 ea
MAC: 00 C0 4E 0C 34 EA
RedBoot>

```

Board Revision

Sets or displays the board revision.

```

RedBoot> boardrev
BoardRev 7
RedBoot> boardrev 3
BoardRev 3
RedBoot>

```

Telnet Enable

Allows the user to enable or disable telnet access to the DeviceMaster RTS. Accepted values are **enable** and **disable**:

```
RedBoot> telnet
Telnet enable
RedBoot> telnet disable
Telnet disable
RedBoot>
```

HTTP Authentication

Controls the type of authentication required for HTTP access. The types of authentication are:

- **noaccess** – No HTTP access will be allowed – access forbidden error will be returned.
- **none** – No authentication will be required.
- **basic** – Plain text password authentication required.
- **md5** – MD5 encrypted password authentication required.
- **invalid** – No HTTP access will be allowed – invalid URL error will be returned.

```
RedBoot> auth
Auth: none
RedBoot> auth basic
Auth: basic
RedBoot>
```

The value of this setting is not used by RedBoot since it does not contain a web server. This setting is merely saved in I2C EPROM as a service to applications.

Password

Changes the DeviceMaster RTS password used to authenticate telnet and HTTP access. Unlike other commands, the password command will not display the current value of the password. The password command *always* sets the value of the password. The password length is limited to 15 characters.

```
RedBoot> password
Password ''
RedBoot> password FooBar1
Password 'FooBar1'
RedBoot>
```

Autoload Timeout

Sets the number of seconds after network initialization that the bootloader will wait before starting the default application program from flash ROM. A time-out value of 0 will disable auto-loading of the default application. The maximum value is 255 seconds.

```
RedBoot> timeout
Timeout 9 seconds
RedBoot> timeout 5
Timeout 5 seconds
RedBoot>
```

Terse Mode

In order to provide a console interface more amenable to programmatic control, the bootloader may be put into **terse** mode. The significant features of terse mode are:

- No prompt is issued.
- Command strings are not echoed.
- All responses consist of a single line. If the command was successful, the response string begins with “+”. If the command failed, the response string begins with “-”.

We recommend that supervisory programs written to the interface with the DeviceMaster RTS boot loader ignore lines that do not begin with “+” or “-” in case bugs in some commands improperly display diagnostic information.

Loading a File

It is possible to load a program or data file into RAM from three sources: serial port (the 4-Pin diagnostic header or external Port 0), Ethernet (from a TFTP server or via TCP), or from flash ROM on the DeviceMaster RTS board. Loading from flash ROM is described in [fis load](#) on Page 18. Loading from serial port or Ethernet is done using the **load** command. Files loaded with the **load** command default to Motorola S-Record format (which must end with a single “Entry-Point” record).

It is possible to load a binary file with the **load** command by using the **-r** and **-b** options.

Loading via Ethernet

TFTP

Loading a file from a TFTP server is done by using the **-h** option to load:

```
RedBoot> load -v -h 192.168.1.2 socket.srec
Entry point: 0x00000384, address range: 0x00000000-0x000718a8
RedBoot>
```

The **-v** option will cause a spinning status indicator to be displayed as the file is loaded. Do not use the **-v** option when connected via telnet. If you are loading a raw binary file, use the **-r** option.

TCP

Loading a file via TCP can be done by connecting to the telnet server (TCP Port 23), logging in, and issuing a **load -m d** command. The telnet server will then expect to read an S-record file as input. Each line read will be acknowledged with a single line feed character. When the last line in the S-record file (which must be an Entry-Point record) has been processed, a load summary will be printed.

Loading via serial port

Loading an S-record file via a serial port is done with a **load** command specify either xmodem or ymodem protocol:

```
RedBoot> load -m ymodem
Start ymodem transfer program.
Entry point: 0x00000384, address range: 0x00000000-0x000718a8
RedBoot>
```

It is also possible to load a binary file via x-modem. Binary download will be approximately 2-3 times faster except for sparse files. They entry point for a binary file will be the base address specified in the load command (or a default of 0 if no entry point is specified). For DeviceMaster RTS eCos executables, the base address should be normally 0:

```
RedBoot> load -b 0 -r -m x
start x-modem download
CRC mode, 4085(SOH)/0(STX)/0(CAN) packets, 2 retries
RedBoot>
```

Flash Image System

RedBoot implements a rudimentary file system that allows program and data files to be stored in flash. There are various **fis** commands that can be used to manipulate this file system. Typing **fis** followed by a carriage-return will display a list of sub-commands:

```
RedBoot> fis
*** invalid 'fis' command: too few arguments
```

Usage:

```
fis create -b <mem_base> -l <image_length> [-s <data_length>]
           [-f <flash_addr>] [-e <entry_point>] [-r <ram_addr>] [-n] <name>
fis delete name
fis erase -f <flash_addr> -l <length>
fis free
fis init [-f]
fis list [-c]
fis load [-b <memory_load_address>] [-c] name
```

fis list

The **fis list** command displays a directly of the files currently stored in flash ROM:

```
RedBoot> fis list
Name                FLASH addr  Mem addr   Length    Entry point
FIS directory       0x053F0000 0x053F0000 0x00010000 0x00000000
default             0x05030000 0x00000000 0x00080000 0x00000384
RedBoot>
```

The **FIS directory** entry will always be there and is the file in which flash ROM bookkeeping information is stored. The **default** file is the program that will be run by RedBoot on startup. This will be described further in [Default Application](#) on Page 19. The **list**, **delete**, and **create** commands are the most frequently used:

fis delete

The **fis delete** command is used to delete a file from flash:

```
RedBoot> fis delete default
Delete image 'default' - are you sure (y/n)? y
... Erase from 0x05030000-0x050b0000: .....
... Erase from 0x053f0000-0x05400000: .
... Program from 0x007a0000-0x007b0000 at 0x053f0000: .
RedBoot> fis list
Name                FLASH addr  Mem addr   Length    Entry point
FIS directory       0x053F0000 0x053F0000 0x00010000 0x00000000
RedBoot>
```

fis create

The **fis create** command is used to store a region of RAM as a file in flash ROM. The basic form of the command is:

```
RedBoot> fis create -b 0 -l 0x10000 foobar
... Erase from 0x05030000-0x05040000: .
... Program from 0x00000000-0x00010000 at 0x05030000: .
... Erase from 0x053f0000-0x05400000: .
... Program from 0x007a0000-0x007b0000 at 0x053f0000: .
RedBoot> fis list
Name                FLASH addr  Mem addr   Length    Entry point
FIS directory       0x053F0000 0x053F0000 0x00010000 0x00000000
foobar              0x05030000 0x00000000 0x00010000 0x00000000
RedBoot>
```

It is also possible to specify a program entry point with the **-e** option and a specific location for the file in flash ROM with the **-f** option. If no address or length is specified, it will use the address and length of the S-Record file most recently loaded to RAM via serial port or Ethernet.

fis load

The **fis load** command loads a file from flash ROM into RAM:

```
RedBoot> fis load foobar
```

Executing a Program

The **go** command is used to execute a program. If no starting address is provided as a parameter to the **go** command, the entry address of the last file loaded into RAM will be used.

```
RedBoot> load -r -h 10.0.0.2 socket.bin
Defaulting to entry point of 0x00000000.
Raw load done: 542101 bytes read
Address range: 00000000-00084594, Entry point: 00000000,
RedBoot> go
ARM DeviceMaster HAL (no virtual vectors)
AIOPIC serial driver: 16 channels
[...]
```

If you wish to specify an entry point, you may do so:

```
RedBoot> load -r -h 10.0.0.2 socket.bin
Defaulting to entry point of 0x00000000.
Raw load done: 542101 bytes read
Address range: 00000000-00084594, Entry point: 00000000,
RedBoot> go 0x384
ARM DeviceMaster HAL (no virtual vectors)
AIOPIC serial driver: 16 channels
[...]
```

Note: *eCos* application have two entry points: 0 and 0x384. The instruction at address 0 is a jump to 0x384.

Default Application

The DeviceMaster RTS version of RedBoot will wait for a configured number of seconds after startup for connections from a host. If no connection from a host is made, the bootloader will look for a file named **default** in the flash file system. If a file named **default** exists, it will be loaded and executed. If a host initiates a download or if the **dis** command is entered during the waiting period, the **default** program will not be loaded.

Here is an example of the commands used to load a program from a TFTP server and save it in flash as the default application:

```
RedBoot> fis delete default
Delete image 'default' - are you sure (y/n)? y
... Erase from 0x05030000-0x050c0000: .....
... Erase from 0x053f0000-0x05400000: .
... Program from 0x007a0000-0x007b0000 at 0x053f0000: .
RedBoot> load -v -h 192.168.1.2 socket.srec
Entry point: 0x00000384, address range: 0x00000000-0x000718a8
RedBoot> fis create default
... Erase from 0x05030000-0x050b0000: .....
... Program from 0x00000000-0x000718a9 at 0x05030000: .....
... Erase from 0x053f0000-0x05400000: .
... Program from 0x007a0000-0x007b0000 at 0x053f0000: .
RedBoot>
```

If you wish to permanently disable the default application, use the **fis delete** command to delete it from the flash file system (example shown in [fis delete](#) on Page 18).

Updating RedBoot

To update the bootloader, load and execute the **burn-redboot** program that contains code to update the bootloader flash. For example:

```
RedBoot> load -v -h 192.168.4.3 burn-redboot.srec
Entry point: 0x00000000, address range: 0x00000000-0x000228d8
/
RedBoot> go
Diag Startup
burn
Id=0089,8897
flash_erase_region(05010000,65536)
sector erase 05010000
sector erase 05020000
flash_program_buf(05010000,000028D8,65536)
ROM = 05000000
done -- resetting...
```

At this point the bootloader has been updated, and the board should reset and run the new bootloader.

Testing Serial Ports

There are two commands that run loopback (internal, RS-232, or RS-422) or port-to-port (RS-485) serial tests.

Loopback Tests

With a loop-back plug connected to a port, the **loop** command may be used to run a loop-back test in either internal loop-back mode, RS-232 mode or RS-422 mode. In RS-232 mode, modem control lines are also tested. In RS-422 mode and internal loop-back mode, only data paths are tested. If the test fails, a hexadecimal error code is displayed. The first digit of the error code indicates what portion of the test failed. Loop-back error codes are shown in [Table 1. Loopback Error Codes](#).

Examples of loopback test commands are shown below.

- RS-232 loopback test on Port 2:

```
RedBoot> loop 232 2
Loopback pass
```
- RS-232 loopback test on Port 3:

```
RedBoot> loop 232 3
- Loopback failed RS-232: 10680
```
- Internal loopback test on Port 1:

```
RedBoot> loop int 1
Loopback pass
```

Table 1. Loopback Error Codes

Loopback Error Codes	Code Description
0x1ssss	No data was present in the receive FIFO when there should have been. The channel status register is displayed in the lower 4 digits (ssss).
0x2ssss	An error flag was present when receive data was read.
0x3ttrr	Receive data byte did not match transmit data byte. The transmit data byte is tt and the receive data byte is rr .
0x40000	Receive data was present after the expected last byte.
0x5ssss	CTS signal did not match expected value.
0x6ssss	RI signal did not match expected value.
0x7ssss	DSR signal did not match expected value.
0x8ssss	CD signal did not match expected value.

Port-to-Port RS-485 Test

Since RS-485 is a half-duplex physical layer, it is not possible to perform a loopback test. Instead an RS-485 cross-over cable must be connected between two different ports on the DeviceMaster RTS.

Trademark Notices

Comtrol and DeviceMaster are trademarks of Comtrol Corporation. Other product names mentioned herein may be trademarks and/or registered trademarks of their respective owners.

First Edition, September 17, 2001
Copyright © 2001. Comtrol Corporation.
All Rights Reserved.

Comtrol Corporation makes no representations or warranties with regard to the contents of this document or to the suitability of the Comtrol product for any particular purpose. Specifications subject to change without notice. Some software or features may not be available at the time of publication. Contact your reseller for current product information.

Document Number: 2000238 Rev. A



e-mail: info@direktronik.se

tel: 08-52 400 700 fax: 08-520 18121