RCC User's Manual

Version 1.1.1

March 2008

1 Introduction

1.1 General

This document describes the RTEMS LEON/ERC32 GNU cross-compiler system (RCC). Discussions are provided for the following topics:

- installing RCC
- contents and directory structure of RCC
- compiling and linking LEON and ERC32 RTEMS applications
- usage of GRMON
- debugging application with GDB

RCC is a multi-platform development system based on the GNU family of freely available tools with additional 'point' tools developed by Cygnus, OAR and Gaisler Research. RCC consists of the following packages:

- GCC-4.2.4 C/C++ compiler
- GNU binary utilities 2.18 with support for LEON UMAC/SMAC instructions
- RTEMS-4.8 C/C++ real-time kernel with LEON2, LEON3 and ERC32 support
- Newlib-1.15.0 standalone C-library
- GDB-6.8 SPARC cross-debugger

1.2 Installation on host platform

1.2.1 Host requirements

RCC is provided for two host platforms: linux/x86 and MS Windows. The following are the platform system requirements:

Linux:	Linux-2.4.x, glibc-2.3 (or higher)
Windows:	MSYS-1.0.10 (or higher)

In order to recompile the RTEMS kernel sources automake-1.10 and autoconf-2.61 is required. MSYS-DTK-1.0.1 is needed on Windows platforms to build autoconf and automake. The sources of automake and autoconf can be found on the GNU ftp server:

ftp://ftp.gnu.org/gnu/autoconf/

ftp://ftp.gnu.org/gnu/automake/

MSYS and MSYS-DTK can be found at http://www.mingw.org.

1.2.2 Installing RCC on Windows platforms

The toolchain installation zip file (sparc-rtems-4.8-gcc-4.2.4-1.1.x-mingw.zip) must be extracted to C:\opt creating the directory C:\opt\rtems-4.8-mingw. The toolchain executables can be invoked from the command prompt by adding the executable directory to the PATH environment variable. The directory C:\opt\rtems-4.8-mingw\bin can be added to the PATH variable by selecting "My Computer->Properties->Advanced->Environment Variables".
Development often requires some basic utilities such as make, but is not required to compile, on Windows platforms the MSYS Base system can be installed to get a basic UNIX like development environment (including make). The RTEMS sources rely on the autoconf and automake utilities to create Makefiles. The MSYS Base

system doesn't include the required version of autoconf and automake, instead they can be compiled from sources as described below.

1.2.2.1 Installing MSYS

The MSYS package can be freely downloaded from http://www.mingw.org, it comes as an self extracting installation application (MSYS-1.0.10.exe). The following text assumes the MSYS has been successfully installed to C:\msys.
The directory where the toolchain is installed (C:\opt\rtems-4.8-mingw) must be found in /opt/rtems-4.8-mingw from the MSYS environment, this can done by adding an mount entry similar to one of the examples below to the /etc/fstab file in the MSYS environment.

```
C:/opt/rtems-4.8-mingw   /opt/rtems-4.8-mingw
```

or

```
C:/opt   /opt
```

The path to the toolchain binaries (C:\opt\rtems-4.8-mingw\bin) must added to the MSYS PATH environment variable. Below is an example of how to change the PATH variable in the MSYS shell.

```
export PATH=/opt/rtems-4.8-mingw/bin:$PATH
```

The toolchain installation can be tested by compiling the samples included in the toolchain,

```
$ cd /opt/rtems-4.8-mingw/src/samples
$ make
```

1.2.2.2 Installing RTEMS source

Installing the RTEMS kernel sources are optional but recommended when debugging applications. The tool-chain libraries are built with debugging symbols making it possible for GDB to find the source files. The RCC RTEMS sources is assumed to be located in C:\opt\rtems-4.8-mingw\src\rtems-4.8.
The RTEMS sources (sparc-rtems-4.8-1.1.x-src.tar.bz2) can be installed by extracting the source distribution to C:\opt\rtems-4.8-mingw\src creating the directory C:\opt\rtems-4.8-mingw\src\rtems-4.8.

1.2.2.3 Building RTEMS from source

The RTEMS build environment can be set up by following the Windows instructions available www.rtems.org, the environment requires MSYS-DTK-1.0.1, autoconf-2.61 and automake-1.10. This section describes how to install MSYS-DTK, autoconf, automake and building RTEMS SPARC BSPS from source.

MSYS-DTK can be downloaded from www.mingw.org, it comes as an self extracting installation application (msysDTK-1.0.1.exe). The following text assumes that MSYS-DTK has been installed successfully into the MSYS environment.

Autoconf and automake can be downloaded from ftp://ftp.gnu.org/gnu/autoconf and ftp://ftp.gnu.org/gnu/automake. Below is a example of how to build and install the required tools.

```
$ mkdir /src
$ cd /src
< ... download autoconf and automake to /src ... >
$ tar -jxf autoconf-2.61.tar.bz2
$ mkdir autoconf-2.61/build
```

```
$ cd autoconf-2.61/build
$ ../configure --prefix=/usr
$ make
$ make install
< ...autoconf-2.61 has been installed ... >
$ cd /src
$ tar -jxf automake-1.10.1.tar.bz2
$ mkdir automake-1.10.1/build
$ cd automake-1.10.1/build
$ ../configure --prefix=/usr
$ make
$ make install
< ... automake-1.10.1 has been installed ... >
$ exit
```

After installing automake and autoconf it may be required to restart the MSYS shell.

Once the tools required by RTEMS source tree has been installed and the MSYS shell has been restarted the installed RTEMS sources can be built manually or using the prepared Makefile available at C:\opt\rtems-4.8-mingw\src\Makefile. The build process is divided in four steps, in the first step the make scripts are generated this step is called bootstrap. The bootstrapping can be done with the make target boot as the examples shows below. The bootstrap step is only needed to be rerun when adding or removing files from the source tree.

```
$ cd /opt/rtems-4.8-mingw/src
$ make bootstrap
```

The second step configures a build environment in /opt/rtems-4.8-mingw/src/build,

```
$ make configure
```

The third and fourth steps compile and install the new kernel to /opt/rtems-4.8-mingw/sparc-rtems

```
$ make compile
$ make install
```

1.2.3 Installing on Linux platforms

The RCC directory tree is compiled to reside in the /opt/rtems-4.8 directory on all platforms. After obtaining the bzipped tarfile with the binary distribution, uncompress and untar it in a suitable location - if this is not /opt/rtems-4.8 then a link have to be created to point to the location of the RCC directory. The distribution can be installed with the following commands:

```
cd /opt
bunzip2 -c sparc-rtems-4.8-gcc-4.2.4-1.1.0-linux.tar.bz2 | tar xf -
```

After the compiler is installed, add /opt/rtems-4.8/bin to the executables search path and /opt/rtems-4.8/man to the man path.

1.2.3.1 Installing RTEMS source

The RTEMS sources used to compile the SPARC BSPS included in RCC is prepared to be installed into /opt/rtems-4.8/src, it can be done as follows.

$ cd /opt/rtems-4.8/src

$ tar -jxf /path/to/sparc-rtems-4.8-1.1.x.tar.bz2

1.2.3.2 Building RTEMS from sources

The RTEMS libraries found in /opt/rtems-4.8/sparc-rtems/BSP can be built from the sources using the Make-file found in /opt/rtems-4.8/src. The RTEMS build environment assumes that autoconf-2.61 and automake-1.10 are installed. Documentation on how to install autoconf and automake is included in respective source and an example can be found above in section 1.2.2.3.

1.3 Contents of /opt/rtems-4.8

The created rtems directory has the following sub-directories:

```
bin                Executables
doc                RCC and GNU documentation
include            Host includes
lib                Host libraries
make               RTEMS make scripts
man                Man pages for GNU tools
sparc-rtems        Sparc target libraries
src                Various sources
```

1.4 RCC tools

The following tools are included in RCC:

```
sparc-rtems-addr2line  Convert address to C/C++ line number
sparc-rtems-ar         Library archiver
sparc-rtems-as         Cross-assembler
sparc-rtems-c++        C++ cross-compiler
sparc-rtems-c++filt    Utility to demangle C++ symbols
sparc-rtems-cpp        The C preprocessor
sparc-rtems-g++        Same as sparc-rtems-c++
sparc-rtems-gcc        C/C++ cross-compiler
sparc-rtems-gcov       Coverage testing tool
sparc-rtems-gdb        Debugger
sparc-rtems-gprof      Profiling utility
sparc-rtems-ld         GNU linker
sparc-rtems-nm         Utility to print symbol table
sparc-rtems-objcopy    Utility to convert between binary formats
sparc-rtems-objdump    Utility to dump various parts of executables
sparc-rtems-ranlib     Library sorter
sparc-rtems-readelf    ELF file information utility
sparc-rtems-size       Utility to display segment sizes
sparc-rtems-strings    Utility to dump strings from executables
sparc-rtems-strip      Utility to remove symbol table
```

1.5 Documentation

The RCC and GNU documentation are distributed together with the toolchain, it consists of RTEMS manuals and GNU tools manuals.

GNU manuals:

```
as.pdf             Using as - the GNU assembler
binutils.pdf       The GNU binary utilities
cpp.pdf            The C Preprocessor
gcc.pdf            Using and porting GCC
gdb.pdf            Debugging with GDB
gprof.pdf          the GNU profiling utility
ld.pdf             The GNU linker
```

Newlib C library:

```
libc.pdf           Newlib C Library
libm.pdf           Newlib Math Library
```

RTEMS manuals:

```
bsp_howto.pdf      BSP and Device Driver Development Guide
c_user.pdf         RTEMS C User's Guide (this is the one you want!)
cpu_supplement.pdf RTEMS SPARC CPU Application Supplement
develenv.pdf       RTEMS Development environment guide
filesystem.pdf     RTEMS Filesystem Design Guide
itron.pdf          RTEMS ITRON 3.0 User's Guide
networking.pdf     RTEMS Network Supplement
new_chapters.pdf   RTEMS Newly added features
porting.pdf        RTEMS Porting Guide
posix1003-1.pdf    RTEMS POSIX 1003.1 Compliance Guide
posix_users.pdf    RTEMS POSIX API User's Guide
relnotes.pdf       RTEMS Release Notes
started.pdf        Getting Started with RTEMS for C/C++ Users
```

The documents are all provided in PDF format, with searchable indexes.

1.6 Support

The RCC compiler system is provided freely without any warranties. Technical support can be obtained from Gaisler Research through the purchase of a technical support contract. See www.gaisler.com for more details.

2 Using RCC

2.1 General development flow

Compilation and debugging of applications is typically done in the following steps:

1. Compile and link program with gcc
2. Debug program using a simulator (gdb connected to TSIM/GRSIM)
3. Debug program on remote target (gdb connected to GRMON)
4. Create boot-prom for a standalone application with mkprom2

RCC supports multi-tasking real-time C/C++ programs based on the RTEMS kernel. Compiling and linking is done in much the same manner as with a host-based gcc.

2.2 sparc-rtems-gcc options

The gcc compiler has been modified to support the following addtional options:

-qleon2	generate LEON2 executable.
-tsc691	generate ERC32 executable.

Other usefull (standard) options are:

-g	generate debugging information - must be used for debugging with gdb
-msoft-float	emulate floating-point - must be used if no FPU exists in the system
-mcpu=v8	generate SPARC V8 mul/div instructions- only for LEON with hardware multiply and divide configured
-O2	optimize code - should be used for optimum performance and minimal code size

Other gcc options are explained in the gcc manual (gcc.pdf).

2.3 RTEMS applications

To compile and link an RTEMS application, use 'sparc-rtems-gcc':

```
sparc-rtems-gcc -g -O2 rtems-hello.c -o rtems-hello
```

RCC creates executables for LEON3 by default. To generate executables for LEON2 or ERC32 add -qleon2 or -tsc691 switches during both compile and link stages. The default load address is start of RAM, i.e. 0x40000000 for LEON2/3 and 0x2000000 for ERC32. Other load addresses can be specified through the use of the -Ttext option (see gcc manual).

RCC uses the sources of RTEMS-4.8 with minor patches, and allows re-compilation if a modification has been made to a bsp or the kernel. Install the RTEMS sources in /opt/rtems-4.8/src, and re-compile and install with:

```
cd /opt/rtems-4.8/src
make install
```

2.4 Floating-point considerations

If the targeted LEON2/3 (or ERC32) processor has no floating-point hardware, then all applications must be compiled (and linked) with the -msoft-float option to enable floating-point emulation. When running the program on the TSIM simulator, the simulator should be started with the -nfp option (no floating-point) to disable the FPU.

2.5 LEON SPARC V8 instructions

Both LEON2 and LEON3 processors can be configured to implement the SPARC V8 multiply and divide instructions. The RCC compiler does by default NOT issue those instructions, but emulates them trough a library. To enable generation of mul/div instruction, use the -mcpu=v8 switch during both compilation and linking. The -mcpu=v8 switch improves performance on compute-intensive applications and floating-point emulation.

LEON2/3 also supports multiply and accumulate (MAC). The compiler will never issue those instructions, they have to be coded in assembly. Note that the RCC assembler and other utilities are based on a modified version of GNU binutils-2.11 that supports the LEON MAC instructions.

2.6 Memory organisation

The resulting RTEMS executables are in elf format and has three main segments; text, data and bss. The text segment is by default at address 0x40000000 for LEON2/3 and 0x2000000 for ERC32, followed immediately by the data and bss segments. The stack starts at top-of-ram and extends downwards.

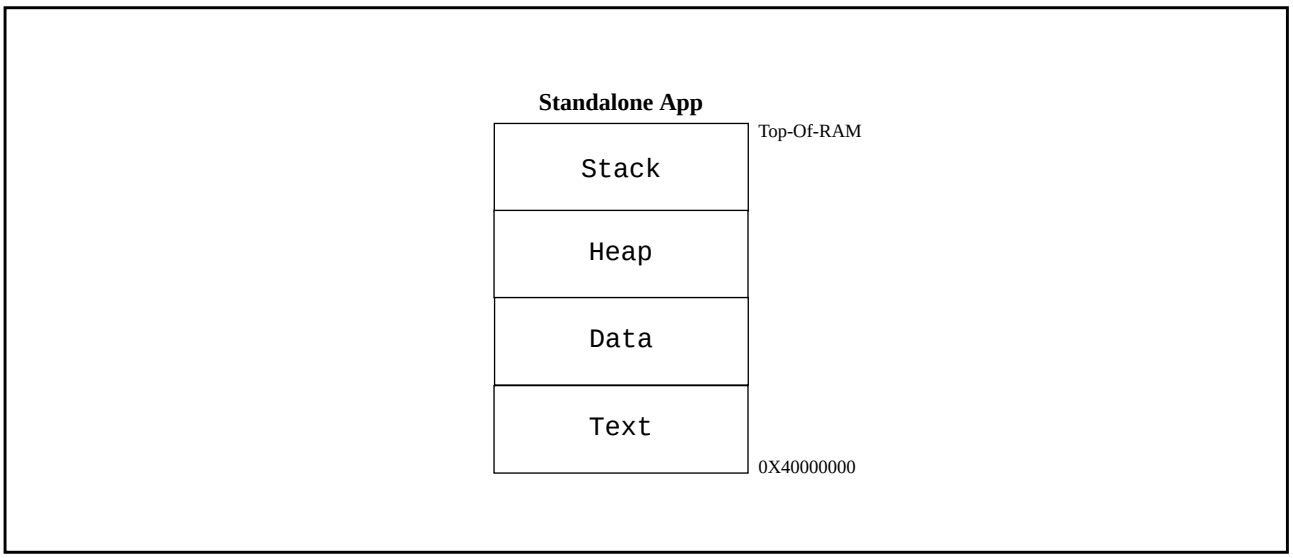

Figure 1: RCC RAM applications memory map

The SPARC trap table always occupies the first 4 Kbyte of the .text segment.

2.7 Board-support packages (BSPs)

RCC includes board-support packages for LEON2, LEON3 and ERC32. BSPs provide interface between RTEMS and target hardware through initialization code specific to target processor and a number of device drivers. Console and timer drivers are supported for all three processors.

The LEON2/3 BSPs support three network drivers: the Gaisler Research GRETH MAC, OpenCores Ethernet MAC and the LAN91C111. The default driver for LEON3 is the GRETH MAC. To select a different network driver, one of following defines should be set before including the bsp.h file:

```
#define RTEMS_BSP_NETWORK_DRIVER_ATTACH RTEMS_BSP_NETWORK_DRIVER_ATTACH_OPENETH

#define RTEMS_BSP_NETWORK_DRIVER_ATTACH RTEMS_BSP_NETWORK_DRIVER_ATTACH_SMC91111
```

See src/samples/rtems_ttcp.c or rtems_http.c for sample networking applications.

LEON2 and ERC32 BSPs assume a default system resource configuration such as memory mapping of on-chip devices and usage of interrupt resources. LEON3 systems are based on GRLIB plug&play configuration, and are thereby highly configurable regarding memory mapping and interrupt routing. At start-up, LEON3 BSP scans the system bus to obtain system configuration information. Device drivers support a number of devices which are automatically recognized, initiated and handled by the device drivers.

The console driver supports one to eight GR APB UARTs. The first UART is registered under name "/dev/console", second and third UARTs get names "/dev/console_b" and "dev/console_c" and so on. LEON3 BSP requires at least one GR APB UART.

The timer driver uses GR General Purpose Timer (GPT). The driver handles GPT timer 0 and the lowest interrupt request line used by GPT. GPT timer 0 and lowest request line should never be used by an RTEMS application. If an application needs to use more timers GPT should be configured to have two or more timers using separate request lines. Timer 0 interrupt can not be shared with other devices or GPT timers 2-7.

For more information on how to configure a system based on GRLIB see GRLIB IP Library User's Manual.

2.8 Making boot-proms

RTEMS applications are linked to run from beginning of RAM. To make a boot-PROM that will run from the PROM on a standalone target, use the mkprom2 utility freely available from www.gaisler.com. The mkprom utility is documented in a separate document that is distributed together with the mkprom2 utility. Mkprom will create a compressed boot image that will load the application into RAM, initiate various processor registers, and finally start the application. Mkprom will set all target dependent parameters, such as memory sizes, waitstates, baudrate, and system clock. The applications do not set these parameters themselves, and thus do not need to be re-linked for different board architectures.

The example below creates a LEON3 boot-prom for a system with 1 Mbyte RAM, one waitstate during write, 3 waitstates for rom access, and 40 MHz system clock. For more details see the mkprom manual

```
mkprom2 -ramsz 1024 -ramwws 1 -romws 3 hello.exe -freq 40 hello.exe
```

Note that mkprom creates binaries for LEON2/3 and for ERC32, select processor type with the mkprom options -leon3, -leon2 or -erc32 flag. To create an SRECORD file for a prom programmer, use objcopy:

```
sparc-rtems-objcopy -O srec rtems-hello rtems-hello.srec
```

2.9 Simple examples

Following example compiles the famous "hello world" program and creates a boot-prom in SRECORD format:

```
bash-2.04$ sparc-rtems-gcc -mcpu=v8 -msoft-float -O2 rtems-hello.c -o rtems-hello

bash-2.04$ mkprom2 -leon3 -freq 40 -dump -baud 38400 -ramsize 1024 -rmw rtems-hello
bash-2.04$ sparc-rtems-objcopy -O srec rtems-hello rtems-hello.srec
bash-2.04$
```

Several example C programs can be found in /opt/rtems-4.8/src/samples.

2.10 Multiprocessing

RTEMS supports asymmetric multiprocessing (AMP). Each node in a multiprocessor system executes its own program image. Message passing is used for communication between nodes. Shared Memory Support Driver has been ported for LEON3 to provide MP support.

Since each CPU executes its own program image, a memory area has to be allocated for each CPU's program image and stack. This is achieved by linking each CPU's RTEMS program at the start addresses of the CPU's memory area and setting stack pointer to the top of the memory area. E.g. for two CPU system, application running on CPU 0 could run in memory area 0x40100000 - 0x401fffff, while CPU 1 runs in memory area 0x4020000 - 0x402fffff. Shared Memory Support Driver allocates 4 KB buffer at address 0x40000000 for message passing (this area can not be used for applications).

Each CPU requires its own set of standard peripherals such as UARTs and timers. In an MP system the BSP will automatically allocate UART 1 and timer 1 to CPU 0, UART 2 and timer 2 to CPU 1 and so on.

Following example shows how to run RTEMS MP application on a two CPU system using GRMON. CPU 0 executes image node1.exe in address space 0x6000000 - 0x600fffff while CPU 1 executes image node2.exe in address space 0x60100000 - 0x601fffff.

```
GRMON LEON debug monitor v1.1.22

Copyright (C) 2004,2005 Gaisler Research - all rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com
...

grlib> lo node1.exe
section: .text at 0x60000000, size 143616 bytes
section: .data at 0x60023100, size 3200 bytes
total size: 146816 bytes (174.4 kbit/s)
read 852 symbols
entry point: 0x60000000
grlib> lo node2.exe
section: .text at 0x60100000, size 143616 bytes
section: .data at 0x60123100, size 3200 bytes
total size: 146816 bytes (172.7 kbit/s)
read 852 symbols
entry point: 0x60100000
grlib> cpu act 0
active cpu: 0
```

```
grlib> ep 0x60000000
entry point: 0x60000000
grlib> stack 0x600fff00
 stack pointer: 0x600fff00
grlib> cpu act 1
active cpu: 1
grlib> ep 0x60100000
entry point: 0x60100000
grlib> stack 0x601fff00
 stack pointer: 0x601fff00
grlib> cpu act 0
active cpu: 0
grlib> run
```

RTEMS MP applications can not be run directly in GRSIM (using load and run commands). Instead a boot image containing several RTEMS MP applications should be created and simulated.

3 Execution and debugging

Applications built by RCC can be debugged on the TSIM LEON/ERC32 simulator, or on target hardware using the GRMON debug monitor (LEON only). Both TSIM and GRMON can be connected to the GNU debugger (gdb) for full source-level debugging.

3.1 TSIM

The TSIM simulator can emulate a full ERC32 and LEON2/3 system with on-chip peripherals and external memories. For full details on how to use TSIM, see the TSIM User's Manual. Below is a simple example that shows how the 'hello world' program is run in the simulator:

```
$ tsim-leon3 rtems-hello

 TSIM/LEON3 SPARC simulator, version 2.0.4a (professional version)

 Copyright (C) 2001, Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to tsim@gaisler.com

using 64-bit time
serial port A on stdin/stdout
allocated 4096 K RAM memory, in 1 bank(s)
allocated 2048 K ROM memory
icache: 1 * 4 kbytes, 16 bytes/line (4 kbytes total)
dcache: 1 * 4 kbytes, 16 bytes/line (4 kbytes total)
section: .text, addr: 0x40000000, size 92096 bytes
section: .data, addr: 0x400167c0, size 2752 bytes
read 463 symbols
tsim> go
resuming at 0x40000000
Hello World


Program exited normally.
tsim>
```

3.2 GRMON

GRMON is used to download, run and debug LEON2/3 software on target hardware. For full details on how to use GRMON, see the GRMON User's Manual. Below is a simple example that shows how the 'hello world' program is downloaded and run:

```
$ grmon -u -jtag

 GRMON LEON debug monitor v1.1.11

 Copyright (C) 2004,2005 Gaisler Research - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com

 using JTAG cable on parallel port
 JTAG chain: xc3s1500 xcf04s xcf04s

 initialising ............
 detected frequency:  41 MHz
 GRLIB build version: 1347
```

```
 Component                            Vendor
 LEON3 SPARC V8 Processor             Gaisler Research
 AHB Debug UART                       Gaisler Research
 AHB Debug JTAG TAP                   Gaisler Research
 GR Ethernet MAC                      Gaisler Research
 LEON2 Memory Controller              European Space Agency
 AHB/APB Bridge                       Gaisler Research
 LEON3 Debug Support Unit             Gaisler Research
 Nuhorizons Spartan3 I/O interfac     Gaisler Research
 OC CAN controller                    Gaisler Research
 Generic APB UART                     Gaisler Research
 Multi-processor Interrupt Ctrl       Gaisler Research
 Modular Timer Unit                   Gaisler Research

 Use command 'info sys' to print a detailed report of attached cores

grlib> lo rtems-hello
section: .text at 0x40000000, size 92096 bytes
section: .data at 0x400167c0, size 2752 bytes
total size: 94848 bytes (339.7 kbit/s)
read 463 symbols
entry point: 0x40000000
grlib> run
Hello World
grlib>
```

Note that the program was started from address 0x40000000, the default start address.

GRMON can also be used to program the boot-PROM image created by sparc-rtems-mkprom into the target's flash PROM.

```
grmon[grlib]> flash unlock all
grmon[grlib]> flash erase all
Erase in progress
Block @ 0x00000000 : code = 0x00800080  OK
Block @ 0x00004000 : code = 0x00800080  OK
                ...
grmon[grlib]> flash load prom.out
section: .text at 0x0, size 54272 bytes
total size: 54272 bytes (93.2 kbit/s)
read 45 symbols
grmon[grlib]> flash lock all
```

When boot-PROM is run (i.e. after reset) it will initialize various LEON registers, unpack the application to the RAM and start the application. The output on console when running "hello world" from PROM is shown below:

```
 MkProm LEON3 boot loader v1.2
 Copyright Gaisler Research - all right reserved

 system clock    : 40.0 MHz
 baud rate       : 38352 baud
```

```
  prom           : 512 K, (2/2) ws (r/w)
  sram           : 1024 K, 1 bank(s), 0/0 ws (r/w)

  decompressing .text
  decompressing .data

  starting rtems-hello

Hello World
```

The application must be re-loaded with the 'load' command before it is possible to re-execute it.

When running multiple RTEMS programs on a multiprocessing system, entry point and stack pointer have to be set up individually for each CPU. E.g. when running app1.exe (link address 0x40100000) on CPU0 and app2.exe (link address 0x40200000) on CPU1:

```
grlib> lo app1.exe
grlib> lo app2.exe
grlib> cpu act 0
grlib> ep 0x40100000
grlib> stack 0x401fff00
grlib> cpu act 1
grlib> ep 0x40200000
grlib> stack 0x402fff00
grlib> cpu act 0
grlib> run
```

3.3 GDB with GRMON and TSIM

To perform source-level debugging with gdb, start TSIM or GRMON with -gdb or enter the 'gdb'command at the prompt. Then, attach gdb by giving command 'tar extended-remote localhost:2222' to gdb when connecting to GRMON or 'tar extended-remote localhost:1234' when connecting to TSIM. Note that RTEMS applications do not have a user-defined main() function as ordinary C-programs. Instead, put a breakpoint on Init(), which is the default user-defined start-up function.

```
jupiter> sparc-rtems-gdb rtems-hello
GNU gdb 6.7.1
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=sparc-rtems".
(gdb) tar extended-remote localhost:2222
Remote debugging using localhost:2222
(gdb) load
Loading section .text, size 0x164e0 lma 0x40000000
Loading section .jcr, size 0x4 lma 0x400164e0
Loading section .data, size 0xaa8 lma 0x400164e8
Start address 0x40000000, load size 94092
Transfer rate: 57902 bits/sec, 277 bytes/write.
(gdb) break Init
Breakpoint 2 at 0x400011f8: file rtems-hello.c, line 33.
(gdb) run
```

```
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /opt/rtems-4.8/src/samples/rtems-hello

Breakpoint 2, Init (ignored=0) at rtems-hello.c:33
33        printf("Hello World\n");
(gdb) cont
Continuing.
Hello World
Program exited with code 0363.
```

The application must be re-loaded with the 'load' command before it is possible to re-execute it.

3.4 Using DDD graphical front-end to gdb

DDD is a graphical front-end to gdb, and can be used regardless of target. The DDD graphical debugger is freely available from http://www.gnu.org/software/ddd. To start DDD with the sparc-rtems-gdb debugger do:

```
ddd  --debugger sparc-rtems-gdb
```

The required gdb commands to connect to a target can be entered in the command window. See the GDB and DDD manuals for how to set the default settings. If you have problems with getting DDD to run, run it with --check-configuration to probe for necessary libraries etc. DDD has many advanced features, see the on-line manual under the 'Help'menu.

On windows/cygwin hosts, DDD must be started from an xterm shell. First launch the cygwin X-server by issuing 'startx' in a cygwin shell, and the launch DDD in the newly created xterm shell.