

RCC User's Manual

Table of Contents

1. Introduction	3
1.1. General	3
1.2. Installation on host platform	3
1.2.1. Host requirements	3
1.2.2. Installing RCC on Windows platforms	4
1.2.3. Installing on Linux platform	5
1.3. Contents of /opt/rcc-cc-v1.3rc2	6
1.4. RCC tools	6
1.5. Documentation	7
1.6. RCC source Git access	8
1.7. Changes since RCC-1.2	8
1.8. Known limitations in the release candidate	8
1.9. Support	8
2. Using RCC	9
2.1. General development flow	9
2.2. GCC compiler toolchain	9
2.2.1. sparc-gaisler-rtems4.12-gcc options	9
2.2.2. LEON target compiler options	10
2.2.3. RCC multi-libs	10
2.2.4. Floating-point considerations	11
2.2.5. SPARC V8 instructions	11
2.2.6. LEON CASA instruction	11
2.2.7. LEON UMAC/SMAC instructions	11
2.3. RTEMS applications	11
2.4. Memory organisation	12
2.5. Board-support packages (BSPs)	12
2.5.1. LEON3 BSP	12
2.5.2. GR740 BSP	13
2.5.3. GR712RC BSP	13
2.5.4. UT699 BSP	13
2.5.5. UT699E/UT700 BSP	13
2.5.6. AT697F BSP	14
2.6. Driver Manager	14
2.6.1. Initialization	14
2.6.2. Configuration	14
2.6.3. Driver configuration	15
2.6.4. drvmgr command	16
2.7. Network configuration	18
2.8. PCI	18
2.9. LEON3 BSP multiprocessing configurations	18
2.9.1. Memory and device resource sharing	19
2.9.2. Interrupt considerations	19
2.9.3. Symmetric multiprocessing (SMP) configuration	19
2.9.4. Asymmetric multiprocessing (AMP) configuration	20
2.9.5. RTEMS SMP AMP example	22
2.10. Making boot-proms	22
3. Examples	23
3.1. Overview	23
3.2. Building	23
4. Execution and debugging	24
4.1. TSIM	24
4.2. GRMON	24
4.3. GDB with GRMON and TSIM	25
4.4. Using DDD graphical front-end to gdb	26

1. Introduction

1.1. General

This document describes the RTEMS LEON/ERC32 GNU cross-compiler system (RCC). This chapter covers the following topics:

- Overview
- Installing RCC
- Contents and directory structure of RCC
- Compiling and linking LEON and ERC32 RTEMS applications
- Debugging RTEMS application with GRMON, TSIM and GDB

RCC is a multi-platform development system based on the GNU family of freely available tools with additional tools developed by RTEMS Community and Cobham Gaisler. RCC consists of the following packages:

- GCC-7.2.0 C/C++ compiler
- GNU binary utilities 2.29
- RTEMS-4.11.99.0 C/C++ real-time kernel, precompiled BSPs for LEON2/3/4 and ERC32
- Newlib-2.5.0 (newlib-snapshot-20170818) standalone C-library
- GDB-6.8 SPARC cross-debugger

RCC includes precompiled BSPs for LEON2/3/4 and ERC32 in different configurations and some examples to easily get started using RTEMS on LEON/ERC32. The precompiled BSPs:

- UT699
- UT700 (also used for UT699E)
- GR712RC
- GR712RC SMP configuration
- GR740
- GR740 SMP configuration
- Generic LEON3
- Generic LEON3 SMP configuration

NOTE: In the RCC-1.3 release candidates the LEON2 and ERC32 are temporarily disabled but will be enabled in future releases.

1.2. Installation on host platform

1.2.1. Host requirements

RCC is provided for two host platforms: Linux/x86_64 and Windows/x86_64. The following are the platform system requirements:

Linux (GCC):	Linux-2.6.x, glibc-2.11.1 (or higher)
Windows (GCC):	MSYS base 2013.07.23 (or higher)

NOTE: In the RCC-1.3 release candidates the Windows toolchain is temporarily not available but will be enabled in future releases.

In order to build samples and recompile the RTEMS kernel sources an MSYS environment and some specific development tools are required. MSYS provides standard UNIX tools such as make, find, autoconf, etc. and MINGW provides GCC and BINUTILS built for Windows. The RTEMS-4.12 kernel source build system requires specific versions of automake-1.12.6 and autoconf-2.69 to work properly. The following links provides all tools necessary and section Section 1.2.2.1 describes the Windows MSYS setup flow briefly.

- <http://www.mingw.org>
- <http://sourceforge.net/projects/mingw/files/Installer/>

- <ftp://ftp.gnu.org/gnu/autoconf/>
- <ftp://ftp.gnu.org/gnu/automake/>

1.2.2. Installing RCC on Windows platforms

The toolchain installation zip file (`sparc-rtems-4.12-gcc-7.2.0-cc-v1.3rc2-mingw.zip`) must be extracted to `C:\opt` creating the directory `C:\opt\rcc-cc-v1.3rc2`. The toolchain executables can be invoked from the command prompt by adding the executable directory to the PATH environment variable. The directory `C:\opt\rcc-cc-v1.3rc2\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, the MSYS Base system can be installed to get a basic UNIX like development environment (including make).

The RTEMS build procedure rely on the autoconf and automake utilities to create Makefiles from the RTEMS sources. 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 is available as a self extracting installation application (`msys-get-installer.exe`). The following text assumes the MSYS has been successfully installed to `C:\MINGW`. The following tools, apart from the MSYS and MINGW base, are also required to be selected for installation from within the MSYS installer:

- `msys-findutils`
- `msys-m4`
- `msys-perl`

The directory where the toolchain is installed (`C:\opt\rcc-cc-v1.3rc2`) must be found in `/opt/rcc-cc-v1.3rc2` from the MSYS environment, this can be done by adding a mount entry similar to one of the examples below to the `/etc/fstab` file in the MSYS environment.

```
C:/opt/rcc-1.3.x /opt/rcc-1.3.x
```

or

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

The path to the toolchain binaries (`C:\opt\rcc-cc-v1.3rc2\bin`) must be 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/rcc-1.3.x/bin:$PATH
```

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

```
$ cd /opt/rcc-1.3.x/src/samples
$ make
```

1.2.2.2. Installing RTEMS source

Installing the RTEMS kernel sources are optional but recommended when debugging applications. The toolchain 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\rcc-cc-v1.3rc2\src\rcc-cc-v1.3rc2`. The RTEMS sources (`rtems-4.12-cc-v1.3rc2-src.tgz`) can be installed by extracting the source distribution to `C:\opt\rcc-cc-v1.3rc2\src` creating the directory `C:\opt\rcc-cc-v1.3rc2\src\rcc-cc-v1.3rc2`.

Alternatively the sources can be obtained from the Git repository, see Section 1.6.

1.2.2.3. Building RTEMS from source

The RCC toolchain comes with pre-built RTEMS kernel for the most common LEON BSP, thus this step is optional.

The RTEMS build environment can be set up by following the Windows instructions available www.rtems.org [http://www.rtems.org] and are briefly described here. In addition to the environment installed in Section 1.2.2.1 the RTEMS requires automake-1.12.6 and autoconf-2.69 to configure and build the RTEMS kernel. This section describes how to install autoconf, automake and building the RTEMS SPARC BSPs from source.

As previously mentioned the auto-tools can be downloaded from the GNU project's FTP. RCC comes with the tools and an install script found in the `/opt/rcc-cc-v1.3rc2/src/tools`. The script is invoked from the MSYS shell:

```
$ cd /opt/rcc-1.3.x/src/tools
$ sh autotools.sh
$ exit
```

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

Once the tools 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 `/opt/rcc-cc-v1.3rc2/src/build-rtems.sh`. See Section 2.2.1 for details on how to set the compiler options used when building a BSP. 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/rcc-1.3.x/src/rcc-1.3.x
$ ./bootstrap -c
$ ./bootstrap -p
$ ./bootstrap
```

The second step configures the RTEMS kernel and a build environment in `/opt/rcc-cc-v1.3rc2/src/build`,

```
$ cd /opt/rcc-1.3.x/src/build
$ ../rcc-1.3.x/configure --target=sparc-gaisler-rtems4.12 --enable-rtemsbsp="BSPs" ..
```

The third and fourth steps compiles and installs the new kernel to `/opt/rcc-cc-v1.3rc2/sparc-gaisler-rtems4.12`

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

1.2.3. Installing on Linux platform

The RCC directory tree is compiled to reside in the `/opt/rcc-cc-v1.3rc2` directory on all platforms. After obtaining the XZ compressed tarfile with the binary distribution, uncompress and untar it in a suitable location - if this is not `/opt/rcc-cc-v1.3rc2` 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
$ tar -Jxf sparc-rtems-4.12-gcc-7.2.x-1.3.y-linux.txz
```

After the compiler is installed, add `/opt/rcc-cc-v1.3rc2/bin` to the executables search path and `/opt/rcc-cc-v1.3rc2/man` to the man path.

1.2.3.1. Installing RTEMS source

The RTEMS sources used to (re)compile the SPARC BSPs included in RCC is prepared to be installed into `/opt/rcc-cc-v1.3rc2/src`, it can be done as follows.

```
$ cd /opt/rcc-1.3.x/src
$ tar -Jxf /path/to/rtems-4.12-1.3.x.txz
```

1.2.3.2. Building RTEMS from sources

The RTEMS libraries found in `sparc-gaisler-rtems4.12/BSP` can be built from the sources using the `src/build-rtems.sh`. The RTEMS build environment requires that autoconf-2.69 and automake-1.12.6 are installed. The auto tools and an example installation script comes with RCC in the `src/tools` directory, it is described in Section 1.2.2.3.

See Section 2.2.1 for details on how to set the BSP compiler options prior to building RTEMS.

Alternatively the sources can be obtained from the Git repository, see Section 1.6.

1.3. Contents of /opt/rcc-cc-v1.3rc2

The created RCC installation directory has the following sub-directories and files:

bin	Toolchain executables
doc	GNU, RCC and RTEMS documentation
include	Host includes
info	Info documents for GNU tools
lib	Libgcc, libstdc++, libgomp multi-libraries
libexec	Host toolchain executables
make	RTEMS make scripts
man	Man pages for GNU tools
sparc-gaisler-rtems4.12	Kernel, BSP, Newlib C/math target libraries and linker scripts
src	Various sources, examples and make scripts used to build kernel from source

1.4. RCC tools

The following GNU tools are included in RCC under the `bin/` directory:

sparc-gaisler-rtems4.12-addr2line	Convert address to C/C++ line number
sparc-gaisler-rtems4.12-ar	Library archiver
sparc-gaisler-rtems4.12-as	Cross-assembler
sparc-gaisler-rtems4.12-c++	C++ cross-compiler
sparc-gaisler-rtems4.12-c++filt	Utility to demangle C++ symbols
sparc-gaisler-rtems4.12-cpp	The C preprocessor
sparc-gaisler-rtems4.12-elfedit	Utility to update ELF header
sparc-gaisler-rtems4.12-g++	Same as sparc-gaisler-rtems4.12-c++
sparc-gaisler-rtems4.12-gcc	C/C++ cross-compiler
sparc-gaisler-rtems4.12-gcov	Coverage testing tool
sparc-gaisler-rtems4.12-gdb	GNU GDB C/C++ level Debugger
sparc-gaisler-rtems4.12-gprof	Profiling utility
sparc-gaisler-rtems4.12-ld	GNU linker
sparc-gaisler-rtems4.12-nm	Utility to print symbol table
sparc-gaisler-rtems4.12-objcopy	Utility to convert between binary formats
sparc-gaisler-rtems4.12-objdump	Utility to dump various parts of executables
sparc-gaisler-rtems4.12-ranlib	Library sorter
sparc-gaisler-rtems4.12-readelf	ELF file information utility
sparc-gaisler-rtems4.12-size	Utility to display segment sizes
sparc-gaisler-rtems4.12-strings	Utility to dump strings from executables
sparc-gaisler-rtems4.12-strip	Utility to remove symbol table

The following tools included in RCC comes from the RTEMS tools project:

rtems-bin2c	Utility to convert binary file to C source array
rtems-bsp-builder	Testing utility for building BSPs in various configurations
rtems-exeinfo	RTEMS Executable Information display tool
rtems-ld	RTEMS linker utility

rtems-ra	Part of the RTEMS Linker
rtems-rap	Part of the RTEMS Linker, manages RAP files
rtems-syms	RTEMS Linker tool to generate symbol tables
rtems-test	RTEMS Tester command line tool
rtems-tld	Part of the RTEMS Linker vcreating traceable executables

1.5. Documentation

The GNU, RCC and RTEMS documentation are distributed together with the toolchain. It consists of API, user's and tools manuals located in the `doc/` directory of the toolchain. The GRLIB drivers that Cobham Gaisler develops are documented in the RCC Drivers user's manual found in the same directory.

RCC specific documentation:

rcc-drivers-cc-v1.3rc2.pdf	GRLIB device driver documentation
rcc-cc-v1.3rc2.pdf	RCC User's Manual

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. RCC source Git access

The RCC RTEMS kernel sources is distributed from Cobham Gaisler homepage in a tar-file, the latest patches are also available using Git revision control system. It is possible to browse the code at <http://git.rtems.org/danielh/rcc.git> or checkout the repository issuing the below commands. The RCC sources are found in the rcc-1.3 branch.

```
$ git clone git://git.rtems.org/danielh/rcc.git
```

1.7. Changes since RCC-1.2

This section lists some of the changes going from RCC-1.2 to RCC-1.3.

- Prebuilt BSPs are now available for UT699, GR712RC, UT700 and GR740 devices. The prebuilt BSPs are using applicable work arounds and optimal ISA configuration.
- Starting with RCC-1.3, GCC uses same compiler flags as the mainline GCC. For example the `-mv8` and the `-mtune=ut699` flags are no longer available.

1.8. Known limitations in the release candidate

This section lists known limitations in RCC-1.3 release candidates that will be addressed in later future RCC-1.3.

- ERC2, LEON2, AT697 BSP are not yet available as pre-built BSPs.
- The toolchain is not yet available for Windows.
- The LEON3 multiprocessor BSP configuration (`-qbsp=leon3_mp`) is not yet available as pre-built.
- The LEON3 BSP configuration without driver manager (`-qbsp=leon3_std`) is not yet available as pre-built.

1.9. Support

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

When contacting the support team at support@gaisler.com, please identify yourself in full, including company affiliation and site name and address. Please identify exactly what product that is used, specifying if it is an IP core (with full name of the library distribution archive file), component, software version, compiler version, operating system version, debug tool version, simulator tool version, board version, etc.

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 specifying a precompiled BSP with `-qbsp=BSP` flag
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. GCC compiler toolchain

2.2.1. sparc-gaisler-rtems4.12-gcc options

The gcc compiler has been modified to support selecting a precompiled BSP or a user built BSP using the flag: `"-qbsp=BSP"`. BSPs are compiled using a specific set of target GCC compiler flags and RTEMS kernel configuration. For details about the prebuilt BSPs please see Section 2.5. Below are the predefined RTEMS BSP variants available in RCC. Unless otherwise stated the RTEMS BSPs are configured single-core with driver manager startup initialization.

<code>-qbsp=leon2</code>	generic LEON2 BSP (without driver manager startup initialization).]
<code>-qbsp=at697f</code>	generic LEON2 BSP with AT697F compiler flags (without driver manager startup initialization).]
<code>-qbsp=erc32</code>	ERC32 BSP (without driver manager startup initialization).]
<code>-qbsp=leon3</code>	generic LEON3/4 BSP (default if no other option given).
<code>-qbsp=leon3std</code>	generic LEON3/4 BSP (without driver manager startup initialization).]
<code>-qbsp=leon3_smp</code>	generic LEON3/4 BSP in RTEMS SMP configuration.
<code>-qbsp=leon3_mp</code>	generic LEON3/4 BSP in RTEMS Multiprocessor executable (AMP).]
<code>-qbsp=gr712rc</code>	GR712RC LEON3/4 BSP.
<code>-qbsp=gr712rc_smp</code>	GR712RC LEON3/4 BSP in RTEMS SMP configuration.
<code>-qbsp=gr740</code>	GR740 LEON3/4 BSP.
<code>-qbsp=gr740_smp</code>	GR740 LEON3/4 BSP in RTEMS SMP configuration.
<code>-qbsp=ut699</code>	UT699 LEON3/4 BSP.
<code>-qbsp=ut700</code>	UT700/UT699 LEON3/4 BSP.
<code>-qbsp=CUSTOM</code>	User defined BSP configuration and compiler flags matching the RTEMS cfg-file in <code>c/src/lib/libbsp/sparc/BSP/make/custom/CUSTOM.cfg</code> .

The prebuilt BSPs **are** built using target specific optimizations and with specific errata work arounds enabled. It is important that the same compiler flags are used in all build steps compiling kernel and application and when linking the final binary (this selects the correct libgcc/libc/libm from the target specific multilibs).

The RTEMS BSP build configuration file is found relative the RTEMS source tree `c/src/lib/libbsp/sparc/BSP/make/custom/BSP.cfg`. When setting custom build flags it is copied/renamed and updated to reflect the compiler flags used when building a custom RTEMS kernel/BSP. The name of the configuration file is used as input to the RTEMS configure script's `--enable-rtemsbsp="CUSTOM"`. See Section 1.2 for build and installation instructions.

The GCC `mcpu=` options determines the instruction set (ISA) generated by the compiler whereas the `-mfix-TARGET` options determine target specific work arounds and compiler configurations. Below are the SPARC/LEON specific and some other commonly used GCC options:

<code>-g</code>	generate debugging information - must be used for debugging with gdb.
<code>-msoft-float</code>	emulate floating-point - must be used if no FPU exists in the system.
<code>-mcpu=v7/none</code>	generate SPARC V7 instructions. This ISA should be compliant with all LEON targets. (default)
<code>-mcpu=v8</code>	generate SPARC V8 mul/div instructions and SUN SPARC timings.
<code>-mcpu=leon</code>	generate SPARC V8 mul/div instructions and LEON timings.
<code>-mcpu=leon3</code>	generate SPARC V8 mul/div and CAS/CASA instructions and LEON3 timings.
<code>-mcpu=leon3v7</code>	generate SPARC V7 with CAS/CASA instructions and LEON3 timings.
<code>-mfix-ut699</code>	Enables work arounds for the UT699, for example the UT699 floating-point errata, the UT699 data cache nullify errata and the LEON3FT B2BST errata.
<code>-mfix-ut700</code>	Enables work arounds for the UT700/UT699e, including the LEON3FT B2BST errata.
<code>-mfix-gr712rc</code>	Enables work arounds for the GR712RC, including the LEON3FT B2BST errata.
<code>-mfix-b2bst</code>	Enables B2BST Errata work around (present in UT699/699e/700 and GR712RC). <code>__FIX_B2BST</code> is predefined by the preprocessor so that written assembly code can adapt to errata. The work around is automatically activated when using one of the targets flags affected (<code>-mfix-ut699</code> , <code>-mfix-ut700</code> , <code>-mfix-gr712rc</code>) and therefore the <code>-mfix-b2bst</code> shall not be used together with them.
<code>-O2</code>	optimize code - should be used for optimum performance and combination of small code size.
<code>-Os</code>	optimize code for size - should be used for minimum code size.

For a full explanation and listing of GNU GCC options see the gcc manual ([gcc.pdf](#)) referenced from Section 1.5.

2.2.2. LEON target compiler options

Below is a summary of commonly used LEON chips and their specific compiler options.

AT697F	<code>-mcpu=leon -mfix-at697f</code>
GR712RC	<code>-mcpu=leon3 -mfix-gr712rc</code>
GR740	<code>-mcpu=leon3</code>
UT699	<code>-mcpu=leon -mfix-ut699</code>
UT700/UT699E	<code>-mcpu=leon3 -mfix-ut700</code>

2.2.3. RCC multi-libs

Below is a summary of all prebuilt variants (multi-libs) of the libgcc/libc/libm available prebuilt in the RCC toolchain. The compiler flags during linking selects which library variant will be linked in. It is possible using either Map-file from the linker or GCC `-v` flag to verify that the correct libraries are used in the final executable.

Table 2.1. RCC multi-lib variants selected during linking

Directory	Compiler linker flags	CPU	FPU	V8	CAS
.	(none)	SPARCv7	Y	N	N
soft	<code>-msoft-float</code>	SPARCv7	N	N	N
v8	<code>-mcpu=v8</code>	SPARCv8	Y	Y	N
leon	<code>-mcpu=leon</code>	LEON2, UT699	Y	Y	N
leon3	<code>-mcpu=leon3</code>	LEON3/4	Y	Y	Y
leon3v7	<code>-mcpu=leon3v7</code>	LEON3/4	Y	N	Y

Directory	Compiler linker flags	CPU	FPU	V8	CAS
leon3/gr712rc	<code>-mcpu=leon3</code> <code>-mfix-gr712rc / -mfix-ut700</code>	GR712RC, UT699E, UT700	Y	Y	Y
leon3v7/gr712rc	<code>-mcpu=leon3v7</code> <code>-mfix-gr712rc / -mfix-ut700</code>	GR712RC, UT699E, UT700	Y	N	Y
leon/ut699	<code>-cpu=leon -mfix-ut699</code>	UT699	Y	Y	N
leon/at697f	<code>-mcpu=leon -mfix-at697f</code>	AT697F	Y	Y	N
soft/v8	<code>-msoft-float -mcpu=v8</code>	SPARCV8	N	Y	N
soft/leon3	<code>-msoft-float -mcpu=leon3</code>	LEON3/4	N	Y	Y
soft/leon3v7	<code>-msoft-float -mcpu=leon3v7</code>	LEON3/4	N	N	Y
soft/leon/ut699	<code>-msoft-float -mcpu=leon -mfix-ut699</code>	UT699	N	Y	N
soft/leon/at697f	<code>-msoft-float -mcpu=leon -mfix-at697f</code>	AT697	N	Y	N

2.2.4. Floating-point considerations

If the targeted processor has no floating-point (FP) hardware, then all code must be compiled (and linked) with the `-msoft-float` option to enable floating-point emulation by SW. When running the program on the TSIM simulator, the simulator should be started with the `-nfp` option (no floating-point) to disable the FPU.

RTEMS saves the FPU register file and FSR register cross context switches and disables the FPU temporarily during interrupts to avoid that a faulty ISR trash the FPU state. If an ISR needs to use FPU it is responsible to save and restore the FPU context itself using the RTEMS API. Due to the SPARC ABI the OS only needs to save the FPU context on interrupts since the ABI states that FPU context is clobbered on function calls.

When creating RTEMS classic tasks the `FLOATING_POINT` option must be set if the task will execute FP instructions. Otherwise the CPU will generate a `fp_disabled` trap (trap type `tt=0x04`) on the first FP instruction executed by the task.

2.2.5. SPARC V8 instructions

LEON2/3/4 processors can be configured to implement the SPARC V8 multiply and divide instructions. Depending on the compiler options (see above sections) the compiler will either generate or not those instructions or emulate them through a SW library part of `libgcc`. Using the `MUL/DIV` improves performance significantly on compute-intensive applications and floating-point emulation.

RTEMS saves the `MUL/DIV (%y)` register state cross context switches and on interrupts to protect against ISR overwriting the `MUL/DIV` state.

2.2.6. LEON CASA instruction

Recent LEON3 and all LEON4 processors can be configured to implement the CASA instruction using the same instruction definitions as SPARC V9. The instruction is used to implement atomic Compare-And-Swap (CAS) operations. The compiler typically generates CAS instruction when compiling atomic C11/C++11 code otherwise they are not generated. The spin-lock implementation of RTEMS SMP is implemented using C11 atomics. Therefore RTEMS SMP requires CAS hardware support.

2.2.7. LEON UMAC/SMAC instructions

LEON2/3/4 models support optionally the 32-bit multiply and accumulate (MAC). The compiler never issues those instructions, they have to be coded in assembly. The GNU `binutils-2.29` assembler supports the LEON MAC instructions.

The RTEMS OS does not save the HW state of the MAC registers cross context switches.

2.3. RTEMS applications

To compile and link an RTEMS application, use `sparc-gaisler-rtems4.12-gcc`:

```
$ sparc-gaisler-rtems4.12-gcc -g -O2 rtems-hello.c -o rtems-hello
```

RCC creates executables for LEON3/4 by default. To generate executables for another target/BSP add the `-qbsp=BSP` switch during both the compile and link stages, see Section 2.2 for options. The load start address is specified by the BSP. The default BSP load address is start of RAM, i.e. 0x40000000 LEON2/3/4 or 0x00000000 for GR740 or 0x20000000 for ERC32. Other load addresses can be specified through the use of the `-Ttext` option (see GCC manual).

RCC uses the sources of RTEMS-4.11.99.0 with minor patches, and allows recompilation when user has modifications or configuration changes to the BSP or the kernel. Install the RTEMS sources in `/opt/rcc-cc-v1.3rc2/src`, and rebuild and install with:

```
$ cd /opt/rtems-4.10/src
$ build-rtems.sh
```

2.4. 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 LEON2/3/4, or 0x00000000 for GR740 or 0x20000000 for ERC32, followed immediately by the data and bss segments. The GR740/LEON4 system has main memory at 0x00000000. The stack starts at top-of-ram and grows downwards.

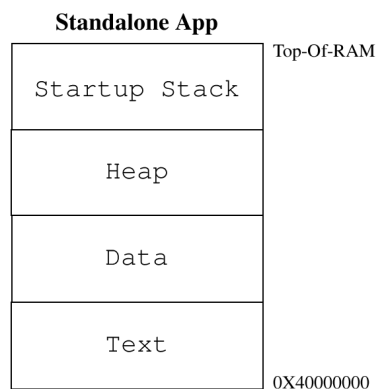


Figure 2.1. RCC RAM applications memory map

The SPARC trap table always occupies the first 4 Kbytes of the `.text` segment and is modified during run-time.

The LEON BSPs auto-detects end-of-ram by looking at the stack pointer provided by the bootloader or GRMON at early boot. Hence the heap will be sized by the loader.

2.5. Board-support packages (BSPs)

RCC includes board support packages for LEON2, LEON3 and ERC32. LEON4 is supported by the LEON3 BSP. BSPs provide interface between RTEMS and target hardware through initialization code specific to the target processor and a number of device drivers. Console and timer drivers are supported for all processors.

LEON2 and ERC32 BSPs assume a default system resource configuration such as memory mapping of on-chip devices and usage of interrupt resources. LEON3/4 systems are based on GRLIB Plug & Play configuration, and are thereby highly configurable regarding memory mapping and interrupt routing. At start-up, the 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. Plug and play makes it possible to use the same BSP for GR712RC, UT699, UT699E/UT700 and GR740 but compiled with different flags.

See Section 2.2.1 on how to select which BSP and compiler flags.

See `doc/rcc-drivers-cc-v1.3rc2.pdf` for GRLIB/LEON device driver API documentation.

2.5.1. LEON3 BSP

The LEON3 BSP includes two different console and timer drivers, 1) standard RTEMS drivers (`-qbsp=leon3std`) and 2) drivers which rely on the driver manager. The latter drivers are possible to config-

ure from the project configuration using standard driver manager configuration options, for example which AP-BUART device is mapped to `/dev/console` and which timer is used as system clock (configuration required for AMP systems).

The APBUART console driver registers the first UART under name `/dev/console`, the second and third UARTs get names `/dev/console_b` and `dev/console_c` and so on. The LEON3 BSP requires at least one APBUART to implement system console and `printf` support.

The timer driver uses the General Purpose Timer (GPTIMER and GRTIMER). The driver handles GPTIMER timer 0 and the lowest interrupt request line used by GPTIMER. 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 1-6.

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

The rest of the GRLIB/LEON device drivers are independent of the BSP but dependent on driver manager API for initialization order etc. The RTEMS project configuration selects which drivers are linked in to final executable image. Please see separate drivers documentation for details.

2.5.1.1. Multi processing (ASMP and SMP) configurations

The LEON3 BSP supports the RTEMS SMP and ASMP configurations. See the multi-processing Section 2.9 for more information.

2.5.2. GR740 BSP

GR740 systems are supported by the LEON3 Plug & Play BSP, a custom linker script, specific compiler flags and the LEON3 BSP GR740 initialization code. At start-up for example, the RTEMS counter API initialization code checks the timers/counters present in HW and selects the best option. For the GR740 the best counter is the ASR22:23 up-counter which is selected for optimum performance. This is an example of how the LEON3 BSP is adapted to the GR740.

Both SMP and uni-processor GR740 configurations are supported by RTEMS. See Section 2.2.1 how to select BSP and set compiler flags for the GR740 (`-qbsp=gr740` or `-qbsp=gr740_smp`).

2.5.3. GR712RC BSP

GR712RC systems are supported by the LEON3 Plug & Play BSP, specific compiler flags and specific GR712RC device drivers.

Prebuilt version of both SMP and uni-processor GR712RC configurations are available in RCC. See Section 2.2.1 how to select BSP and set compiler flags for the GR712RC (`-qbsp=gr712rc` or `-qbsp=gr712rc_smp`).

NOTE: IRQ14 is the default interrupt for servicing multi processing (ASMP and SMP) IPIs. The 1553, Ethernet and Telecommand I/O also generates IRQ14 which will be in conflict is used simultaneously. If one of these I/O will generate interrupts it is required to move the IPI to another IRQ, see Section 2.9.2.2 on how to setup the IPI configuration.

2.5.4. UT699 BSP

UT699 systems are supported by the LEON3 Plug & Play BSP, specific compiler flags and BSP/driver adaptations that works around errata (forces cache miss when accessing DMA areas).

See Section 2.2.1 how to select BSP and set compiler flags for the UT699 (`-qbsp=ut699`).

2.5.5. UT699E/UT700 BSP

UT699E/UT700 systems are supported by the LEON3 Plug & Play BSP, specific compiler flags and drivers. The same build settings are always used for UT699E and UT700.

See Section 2.2.1 how to select BSP and set compiler flags for the UT699E/UT700 (`-qbsp=ut700`).

2.5.6. AT697F BSP

AT697 systems are supported by the LEON2 BSP, specific compiler flags and drivers. The AT697 PCI support requires the driver manager and the PCI library to set up PCI peripherals.

See Section 2.2.1 how to select BSP and set compiler flags for the AT697 (`-qbsp=at697f`).

2.6. Driver Manager

The LEON3 BSP uses an optional Driver Manager that handles drivers and devices on the AMBA and PCI Plug & Play buses. The drivers are automatically assigned to one or more hardware devices. The Driver Manager is either initialized by the user from the `Init()` thread after RTEMS has started up, or during startup of RTEMS. The precompiled LEON3 BSP has by default (see Section 2.2.1) the driver manager enabled (`--enable-drvmgr` was given to RTEMS configure during compile-time) that means that no extra initialization calls from `Init()` is needed, however which drivers to be included must be configured uniquely per project. By default the Timer and UART drivers are included for system clock and console. One can use `-qbsp=leon3std` to avoid using the driver manager. In most cases the GPTIMER and the APBUART drivers are required by the application.

If the driver manager was configured to be initialized by the BSP, the `RTEMS_DRVMGR_STARTUP` define is defined. If not configured the define is not set and the user can choose to initialize the driver manager manually from for example the `Init()` task or not use it at all.

LEON2 systems are divided into two different systems, standard LEON2 systems and GRLIB-LEON2 systems where the AMBA Plug & Play bus is available. Both systems can use the LEON2 hardcoded bus with the Driver Manager, however it's primary intention is to provide a root bus for a second bus supporting Plug & Play. For example a GRLIB-LEON2 system has hardcoded peripherals (the standard LEON2 peripherals) and GRLIB cores attached available from the AMBA Plug & Play information, the setup for a system like that would be a LEON2 hardcoded bus and a LEON2 AMBA Plug & Play sub bus. Once the AMBA Plug & Play bus is initialized all device and their drivers can be used the same way as in LEON3/4 systems.

For AT697 PCI systems the driver manager can be used to scan the PCI bus.

The ERC32 BSP does not support the driver manager.

2.6.1. Initialization

Before the driver manager is initialized one must register a root bus driver so that the driver manager knows which bus to start search for devices at. For a LEON3 system this means calling `ambapp_grlib_root_register()` with a configuration structure. The driver manager itself must also be initialized by calling `drvMgr_init()` before any driver relying on the driver manager can be accessed. The manager then calls each individual driver's register function one by one for initialization and registration. The driver functions are typically named `DRIVER_register()`.

NOTE: As described previously this step is taken care of by the BSP when `--enable-drvmgr` was used at RTEMS configuration/build time.

2.6.2. Configuration

The driver manager is configured by defining the array `drvMgr_drivers`, it contains one function pointer per driver that is responsible to register one or more drivers. The `drvMgr_drivers` can be set up by defining `CONFIGURE_INIT`, selecting the appropriate drivers and including `drvMgr/drvMgr_confdefs.h`. The approach is similar to configuring a standard RTEMS project using `rtems/confdefs.h`. Below is an example how to select drivers.

```
#include <rtems.h>
#define CONFIGURE_INIT
#include <bsp.h>

/* Standard RTEMS setup */
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_MAXIMUM_DRIVERS
#include <rtems/confdefs.h>
```

32


```

/* Driver manager setup */
#if defined(RTEMS_DRVMGR_STARTUP)
/* if --enable-drvmgr was given to configure (-qbsp=leon3, ..) include GPTIMER and APBUART drivers
 * that rely on the driver manager
 */
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_GPTIMER
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_APBUART
#endif

/* Select additional drivers */
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_GRETH
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_GRSPW
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_GRCAN
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_OCCAN
#define CONFIGURE_DRIVER_AMBAPP_MCTRL
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_PCIF
#define CONFIGURE_DRIVER_AMBAPP_GAISLER_GRPIC
#define CONFIGURE_DRIVER_PCI_GR_RASTA_IO
#define CONFIGURE_DRIVER_PCI_GR_RASTA_TMTC
#define CONFIGURE_DRIVER_PCI_GR_701

#include <drvmgr/drvmgr_confdefs.h>

```

By default the BSP automatically includes the timer and console drivers when the driver manager is initialized on startup, whereas when using the standard built BSP (-qbsp=leon3std) the timer and console drivers are included based on the `CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER` and `CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER`. However it is possible to remove the BSP default drivers using `#undef` after including `bsp.h` from the RTEMS project configuration.

2.6.3. Driver configuration

Device drivers are configured per device using driver resources. The driver manager provides driver with an API to easily extract the information using a common way for all driver to configure a device. Driver resources are described by an array of different data types which are assigned names for flexibility. The name is searched for by the driver once initialized, if resource name is found the resource value replaces the default value internal to the driver. The resources are provided by the bus driver. It is up to the bus driver how the resources are assigned to the bus. The LEON BSPs bus drivers use a default weak array (`gplib_drv_resources` on the LEON3 BSP) that can be overridden by the project configuration without any function calls. The driver parameters are documented separately for each driver in the drivers manual. The example below sets up GRSPW0 and GRSPW1 descriptor count driver resources for the AMBA Plug & Play bus on two different GR-RASTA-IO PCI boards and the root bus.

```

/* ROOT AMBA PnP Bus: GRSPW0 and GRSPW1 resources */
struct drvmgr_key gplib_grspw01_res[] =
{
    {"txDesc", DRVMGR_KT_INT, {(unsigned int)32}},
    {"rxDesc", DRVMGR_KT_INT, {(unsigned int)32}},
    DRVMGR_KEY_EMPTY
};

/* If RTEMS_DRVMGR_STARTUP is defined we override the "weak defaults" that
 * is defined by the LEON3 BSP.
 */
struct drvmgr_bus_res gplib_drv_resources =
{
    .next = NULL,
    .resource = {
        {DRIVER_AMBAPP_GAISLER_GRSPW_ID, 0, &gplib_grspw01_res[0]},
        {DRIVER_AMBAPP_GAISLER_GRSPW_ID, 1, &gplib_grspw01_res[0]},
        DRVMGR_RES_EMPTY
    },
};

#ifdef RTEMS_DRVMGR_STARTUP
struct gplib_config gplib_bus_config = {
    &ambapp_plb, /* AMBAPP bus setup */
    &gplib_drv_resources, /* Driver configuration */
};
#endif

/* GR-RASTA-IO 0: GRSPW0 resources */
struct drvmgr_key rastao0_grspw0_res[] = {
    {"txDesc", DRVMGR_KT_INT, {(unsigned int)8}},
    {"rxDesc", DRVMGR_KT_INT, {(unsigned int)32}},
    DRVMGR_KEY_EMPTY
};

```

```

/* GR-RASTA-IO 1: GRSPW1 resources */
struct drvmgr_key rastai0_grspw0_res[] = {
    {"txDesc", DRVMMGR_KT_INT, {(unsigned int)16}},
    {"rxDesc", DRVMMGR_KT_INT, {(unsigned int)16}},
    DRVMMGR_KEY_EMPTY
};

/* GR-RASTA-IO 1: GRSPW0 and GRSPW1 resources use same configuration */
struct drvmgr_key rastai0l_grspw0l_res[] = {
    {"txDesc", DRVMMGR_KT_INT, {(unsigned int)16}},
    {"rxDesc", DRVMMGR_KT_INT, {(unsigned int)64}},
    DRVMMGR_KEY_EMPTY
};

/** Driver resources for GR-RASTA-IO 0 AMBA PnP bus */
struct drvmgr_bus_res gr_rasta_io0_res = {
    .next = NULL,
    .resource = {
        {DRIVER_AMBAPP_GAISLER_GRSPW_ID, 0, &rastai0_grspw0_res[0]},
        {DRIVER_AMBAPP_GAISLER_GRSPW_ID, 1, &rastai0_grspw1_res[0]},
        DRVMMGR_RES_EMPTY
    },
};

/** Driver resources for GR-RASTA-IO 1 AMBA PnP bus */
struct drvmgr_bus_res gr_rasta_iol_res = {
    .next = NULL,
    .resource = {
        {DRIVER_AMBAPP_GAISLER_GRSPW_ID, 0, &rastai0l_grspw0l_res[0]},
        {DRIVER_AMBAPP_GAISLER_GRSPW_ID, 1, &rastai0l_grspw0l_res[0]},
        DRVMMGR_RES_EMPTY
    },
};

/* Tell GR-RASTA-IO driver about the bus resources.
 * Resources for one GR-RASTA-IO board are available.
 * AMBAPP->PCI->GR-RASTA-IO->AMBAPP bus resources
 *
 * The resources will be used by the drivers for the
 * cores found on the GR-RASTA-IO->AMBAPP bus.
 *
 * The "weak defaults" are overridden here.
 */
struct drvmgr_bus_res *gr_rasta_io_resources[] = {
    &gr_rasta_io0_res, /* GR-RASTA-IO board 1 resources */
    &gr_rasta_iol_res, /* GR-RASTA-IO board 2 resources */
    NULL, /* End of table */
};

rtems_task Init( rtems_task_argument argument)
{
    /* Manual driver manager initialization only required when driver manager not initialized during
     * startup (-qleon2, -qleon3std)
     */
#ifdef RTEMS_DRVMGR_STARTUP
    /* Register GRLIB root bus (LEON3/4) */
    ambapp_grib_root_register(&grib_bus_config);

    /* Initialize Driver Manager */
    drvmgr_init();
#endif
    ...
}

```

2.6.4. drvmgr command

The RTEMS shell comes with a number of commands, the drvmgr command can be used to extract information about the current setup and hardware. Please see the `rtems-shell.c` sample application that comes with RCC. The rtems-shell on a GR712RC ASIC:

```

Creating /etc/passwd and group with three useable accounts
root/pwd , test/pwd, rtems/NO PASSWORD

RTEMS SHELL (Ver.1.0-FRC):dev/console. Oct 3 2011. 'help' to list commands.
[/] # drvmgr --help
usage:
drvmgr buses      List bus specific information on all buses
drvmgr devs       List general and driver specific information
                  about all devices
drvmgr drvs       List driver specific information on all drivers

```



```

drvmgr info [ID]      List general and driver specific information
                      about all devices or one device, bus or driver
drvmgr mem            Dynamically memory usage
drvmgr parent ID      Short info about parent bus of a device
drvmgr remove ID      Remove a device or a bus
drvmgr res ID         List Resources of a device or bus
drvmgr short [ID]     Short info about all devices/buses or one
                      device/bus
drvmgr topo           Show bus topology with all devices
drvmgr tr ID OPT ADR  Translate hw(0)/cpu(1) (OPT bit0) address ADR
                      down(0)/up(1) streams (OPT bit1) for device

drvmgr --help

```

```

[/] # drvmgr topo
--- BUS TOPOLOGY ---
|-> DEV 0x400fd3a0 GRLIB AMBA PnP
|  |-> DEV 0x400fd450 GAISLER_LEON3FT
|  |-> DEV 0x400fd4a8 GAISLER_LEON3FT
|  |-> DEV 0x400fd500 GAISLER_AHBJTAG
|  |-> DEV 0x400fd558 GAISLER_ETHMAC
|  |-> DEV 0x400fd5b0 GAISLER_SATCAN
|  |-> DEV 0x400fd608 GAISLER_SPW2
|  |-> DEV 0x400fd660 GAISLER_SPW2
|  |-> DEV 0x400fd6b8 GAISLER_SPW2
|  |-> DEV 0x400fd710 GAISLER_SPW2
|  |-> DEV 0x400fd768 GAISLER_SPW2
|  |-> DEV 0x400fd7c0 GAISLER_SPW2
|  |-> DEV 0x400fd818 GAISLER_B1553BRM
|  |-> DEV 0x400fd870 GAISLER_GRTC
|  |-> DEV 0x400fd8c8 GAISLER_GRTM
|  |-> DEV 0x400fd920 GAISLER_SLINK
|  |-> DEV 0x400fd978 GAISLER_FTMCTRL
|  |-> DEV 0x400fd9d0 GAISLER_APBMS
|  |-> DEV 0x400fd928 GAISLER_LEON3DSU
|  |-> DEV 0x400fd980 GAISLER_APBMS
|  |-> DEV 0x400fdb30 GAISLER_CANAHB
|  |-> DEV 0x400fdad8 GAISLER_CANAHB
|  |-> DEV 0x400fdb88 GAISLER_FTAHBRAM
|  |-> DEV 0x400fdb0 GAISLER_APBUART
|  |-> DEV 0x400fdc38 GAISLER_IRQMP
|  |-> DEV 0x400fdc90 GAISLER_GPTIMER
|  |-> DEV 0x400fdce8 GAISLER_SPICTRL
|  |-> DEV 0x400fdd40 GAISLER_CANMUX
|  |-> DEV 0x400fdd98 NO_NAME
|  |-> DEV 0x400fddf0 GAISLER_ASCS
|  |-> DEV 0x400fde48 GAISLER_GPIO
|  |-> DEV 0x400fdea0 GAISLER_GPIO
|  |-> DEV 0x400def8 GAISLER_I2CMST
|  |-> DEV 0x400fdf50 GAISLER_CLKGATE
|  |-> DEV 0x400fdfa8 GAISLER_AHBSTAT
|  |-> DEV 0x400fe000 GAISLER_APBUART
|  |-> DEV 0x400fe058 GAISLER_APBUART
|  |-> DEV 0x400fe0b0 GAISLER_APBUART
|  |-> DEV 0x400fe108 GAISLER_APBUART
|  |-> DEV 0x400fe160 GAISLER_APBUART
|  |-> DEV 0x400fe1b8 GAISLER_GRTIMER

```

```

[/] # drvmgr info 0x400fdb0
-- DEVICE 0x400fdb0 --
PARENT BUS: 0x400fd408
NAME:      GAISLER_APBUART
STATE:     0x00000100
INIT LEVEL: 4
ERROR:     0
MINOR BUS: 0
MINOR DRV: 0
DRIVER:    0x400a2198 (APBUART_DRV)
PRIVATE:   0x400fe210
--- DEVICE INFO FROM BUS DRIVER ---
AMBA PnP DEVICE
VENDOR ID: 0x0001 (VENDOR_GAISLER)
DEVICE ID: 0x000c (GAISLER_APBUART)
IRQ:       2
VERSION:   0x1
ambapp_core: 0x400fdc1c
interfaces: APBSLV
APBSLV_FREQ: 80000kHz
          apb: 0x80000100-0x800001ff
--- DEVICE INFO FROM DEVICE DRIVER ---
UART Mode:  TERMIO
STATUS REG: 0x100082
CTRL REG:   0x80000803
SCALER REG: 0x103 baud rate 38610

```

2.7. Network configuration

The LEON2/3 BSPs support three network devices: the Cobham Gaisler GRETH MAC, GRETH_GBIT MAC and the LAN91C111 (note: 91C111 only in single-core configuration). The GRETH driver comes in two editions, one that needs the driver manager (in libbsp) to operate and the standard driver (in libchip). The driver manager dependent GRETH driver adds the network interface automatically to the `rtems_bsdnet_config` network interface configuration using `network_interface_add()` function. It also supports SMP. The drivers under `libchip/` directory does not support SMP currently. Since the LAN91C111 chip is supported over the I/O bus it cannot be found by Plug & Play so it has to be manually set up by either hardcoding an entry in the `rtems_bsdnet_config` interface list or dynamically registered by calling `network_interface_add()`. The LAN91C111 attach routine is defined by `RTEMS_BSP_NETWORK_DRIVER_ATTACH_SMC91111` in `bsp.h`. The standard GRETH device is setup in a similar way, see `bsp.h`.

See `src/samples/rtems-ttcp.c` for a sample networking application.

2.8. PCI

Cobham Gaisler provides a PCI Library together with RTEMS located in `cpukit/libpci` in the sources. The documentation for the PCI Library is located in the RTEMS documentation `doc/usr/libpci.t` and available prebuilt into PDF named `c_user.pdf`, see Section 1.5.

All the LEON PCI host bridge drivers have been written using the driver manager and the PCI library is initialized using the PCI bus layer for the driver manager. Even if the PCI Library does not require the driver manager the PCI host drivers does.

The RTEMS shell has been extended with a `pci` command can be used to extract information about the current setup and hardware. Please see the `rtems-shell.c` sample application that comes with RCC. A non-PCI system:

```
reating /etc/passwd and group with three useable accounts
root/pwd , test/pwd, rtems/NO PASSWORD
```

```
RTEMS SHELL (Ver.1.0-FRC):dev/console. Oct 3 2011. 'help' to list commands.
```

```
[/] # pci --help
usage:
pci ls [bus:dev:fun|PCIID]      List one or all devices
pci r{8|16|32} bus:dev:fun ofs  Configuration space read
pci r{8|16|32} PCIID ofs       Configuration space read
                                access by PCIID
pci w{8|16|32} bus:dev:fun ofs d Configuration space write
pci w{8|16|32} PCIID ofs d     Configuration space write
                                access by PCIID
pci pciid bus:dev:fun          Print PCIID for bus:dev:fun
pci pciid PCIID               Print bus:dev:fun for PCIID
pci pcfg                     Print current PCI config for
                                static configuration library
pci getdev {PCIID|bus:dev:fun} Get PCI Device from RAM tree
pci infodev DEV_ADR           Info about a PCI RAM Device
pci --help
```

```
[/] # pci
SYSTEM:          UNKNOWN / UNINITIALIZED
CFG LIBRARY:     AUTO
NO. PCI BUSES:   0 buses
PCI ENDIAN:      Little
MACHINE ENDIAN:  Big
```

2.9. LEON3 BSP multiprocessing configurations

RTEMS supports uni-processor, asymmetric multiprocessing (ASMP) and Symmetric multiprocessing (SMP) configurations. The LEON3 BSP support all three modes. The ASMP mode can be accomplished in one of two possible setups:

- Uni-processor configuration and custom synchronization/communication protocols between CPUs implemented by user. (`-qbsp=BSP`)
- RTEMS ASMP configuration where RTEMS services are used to synchronize and communicate between the CPU nodes. For example using global objects like global semaphores and global tasks and Inter Processor Communication services implemented on top the shared memory (SHM) driver. (`-qbsp=BSP_mp`)

Note that in a multi-processor system with more than two CPUs RTEMS supports running SMP configuration on two or more processor in parallel with another OS instance on a different set of processors. This means SMP and ASMP can coexist in the same system.

2.9.1. Memory and device resource sharing

An ASMP application typically assigns different hardware resources to different operating system instances without overlap at compile-time. For example UART0/IRQ2 is used by CPU0 and UART1/IRQ3 by CPU1. In uni-processor and ASMP configuration the resource sharing between operating system instances are configured using the driver manager or by using the default configuration. Linker scripts or linker arguments are typically used to separate address spaces between ASMP CPUs.

In SMP configuration RTEMS has typically access to all resources. Instead it uses locks and scheduler to share/protect resources between processors during run-time. This means that no specific driver manager configuration related to resource sharing is needed for SMP.

2.9.2. Interrupt considerations

2.9.2.1. Interrupt Controller IRQ(A)MP

The GRLIB IRQ(A)MP interrupt controller has extended ASMP support which allows for assigning individual isolated interrupt controllers on a processor basis. An individual isolated interrupt controller is also referred to as an *internal interrupt controller* in the IRQ(A)MP documentation. For an RTEMS SMP application running in an AMP setting, it is recommended to dedicate one of the virtual interrupt controllers to RTEMS SMP. RTEMS will detect which interrupt controller its processor is assigned to by probing the IRQ(A)MP Asymmetric multiprocessing control register and the Interrupt controller select registers.

In SMP configuration, RTEMS assumes that the boot loader (MKPROM or GRMON) has assigned all the processor part of the SMP system to the same (internal) interrupt controller. For example, the GR740 has a IRQ(A)MP with four internal controllers but only one is used when RTEMS SMP control all four CPUs.

2.9.2.2. Inter processor interrupt (IPI)

When RTEMS AMP/SMP requests to notify another CPU an interrupt is generated on the target CPU by writing the interrupt controller's registers. The ISR handling IPI requests are typically associated with a "free" interrupt number, i.e. not used for I/O interrupts. The interrupt is by default the highest interrupt number available, IRQ14. It is recommended to change to a dedicated IRQ if other (I/O) interrupt sources is also generating IRQ14.

The IPI interrupt is configurable on project compile time by overriding the weak variable `LEON3_mp_irq` with a custom IRQ number. For example include this code in the project to handle IPIs on IRQ11 on all RTEMS CPUs:

```
const unsigned char LEON3_mp_irq = 11;
```

2.9.2.3. Interrupt affinity (SMP only)

By default the boot CPU handles all interrupts (except for IPI and per-cpu iimer) in SMP configuration, thus normally CPU0 handles all I/O interrupts. The LEON3 BSP uses a static interrupt affinity configuration table to select which CPU will service a particular IRQ. `LEON3_irq_to_cpu` is a weak variable which translates `IRQ[I]` to `CPU[J]`. It allows the user to override the default BSP behaviour during compile time. The table is used when registering a new interrupt handler.

2.9.3. Symmetric multiprocessing (SMP) configuration

2.9.3.1. Processor selection

RTEMS SMP allows for a wide range of processor configuration arrangements in a multiprocessor system. By default, an SMP application will execute on the processors with index 0 up to `CONFIGURE_MAXIMUM_PROCESSORS-1`, where `CONFIGURE_MAXIMUM_PROCESSORS` is a preprocessor define which can be provided to the compiler using the `-D` option. The boot processor is normally CPU 0, but there is no restriction on this: the first processor reaching the kernel entry point is assigned as the boot processor.

An RTEMS SMP application can be configured at compile time to assign different schedulers to specific sets of processors. This is described in detail in the *RTEMS Classic API Guide 4.12* in the sections named *Scheduling concepts* and *Scheduler algorithm Configuration*. In this RCC User's Manual we will assume usage of the the default scheduler. Processor assignments will be setup using a *scheduler assignment table*.

Processors are assigned to the application (scheduler) with the scheduler assignment table, specified using the `CONFIGURE_SMP_SCHEDULER_ASSIGNMENTS` preprocessor define. The define shall contain a list of `RTEMS_SCHEDULER_ASSIGN(index, attr)` and `RTEMS_SCHEDULER_ASSIGN_NO_SCHEDULER` entries. There is one entry for each processor in the range 0 to `CONFIGURE_MAXIMUM_PROCESSORS-1`.

The following example will assign RTEMS SMP to CPU1 and CPU2. CPU0 and CPU3 will not be used in the RTEMS SMP instance and can run operating system(s).

```
#define CONFIGURE_MAXIMUM_PROCESSORS 4
/* RTEMS SMP on CPU[0,1,2]. Something else on CPU[3]. */
#define CONFIGURE_SMP_SCHEDULER_ASSIGNMENTS \
    RTEMS_SCHEDULER_ASSIGN_NO_SCHEDULER, \
    RTEMS_SCHEDULER_ASSIGN(0, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
    RTEMS_SCHEDULER_ASSIGN(1, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
    RTEMS_SCHEDULER_ASSIGN(2, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
    RTEMS_SCHEDULER_ASSIGN_NO_SCHEDULER
```

For `RTEMS_SCHEDULER_ASSIGN(index, attr)`, the `index` parameter selects one of the optionally configured schedulers, and the `attr` parameter is set to `RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY` or `RTEMS_SCHEDULER_ASSIGN_PROCESSOR_OPTIONAL`. A processor is not used in the application if its entry in the list is `RTEMS_SCHEDULER_ASSIGN_NO_SCHEDULER`.

2.9.4. Asymmetric multiprocessing (AMP) configuration

RTEMS supports asymmetric multiprocessing (AMP), the LEON3 BSP supports AMP in two different setups. Either the RTEMS kernel is compiled with multiprocessing support (`-qbsp=BSP_mp`) or the user setup custom resource sharing with driver manager resources (`-qbsp=BSP`), the difference is that RTEMS provide multiprocessing objects and communication channels in the former case and in the latter case the user is responsible for all synchronization itself which in many cases are sufficient. All nodes in a asymmetric multiprocessor system executes thier own program image. Messages are passed between the nodes containing synchronization information, for example take global semaphore A. Messages are sent over memory using the Shared Memory Support Driver in the LEON3 BSP, and interrupts are used to alert the receiving CPU.

The kernel must be compiled with multiprocessing support in order for the RTEMS AMP support to be available, the toolchain includes a precompiled LEON3 MP kernel in `rcc-cc-v1.3rc2/sparc-gaisler-rtems4.12/leon3_mp`, it is the LEON3 BSP compiled with multiprocessing support. The MP kernel is selected when the `-qbsp=BSP_mp` argument is given to `sparc-gaisler-rtems4.12-gcc`.

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, the application running on CPU 0 could run in memory area `0x40100000 - 0x401fffff`, while CPU 1 runs in memory area `0x40200000 - 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 GPT 0 timer 1 to CPU 0, UART 2 and GPT 0 timer 2 to CPU 1 and so on. When the default configuration does not meet the requirements or hardware setup a custom resource allocation can be setup using the driver manager, see below.

The shared memory driver's default memory layout configuration can be overridden without recompiling the kernel. The default settings are set in the variable weak variable `BSP_shm_cfgtbl`, it can be overridden by defining `BSP_shm_cfgtbl` once in the project as in the below example. The parameters that has an effect in changing is the fields base and length.

```
/* Override default SHM configuration */
shm_config_table BSP_shm_cfgtbl = {
    .base = (void *)0x40000000,
    .length = 0x00010000
}
```

```
};
```

Hardware resource allocation is done by the BSP for UART, IRQ controller and System Clock Timer. Devices which has a driver that is implemented using the driver manager can be ignored by a specific CPU by assigning the keys value NULL in the driver resources. The driver manager simply ignores the device when a NULL resource is detect. An example is given below where CPU0 is assigned GRGPIO0 and CPU1 GRGPIO1. GPTIMER driver have options that limit number of timers and which timer is used for system clock, the system console and debug output can be selected to a specific UART with the APBUART driver.

CPU0 Application:

```
struct rtems_drvmgr_drv_res grlib_drv_resources[] =
{
    {DRIVER_AMBAPP_GAISLER_GRGPIO_ID, 1, NULL} /* Used by CPU1 */
};
```

CPU1 Application:

```
struct rtems_drvmgr_drv_res grlib_drv_resources[] =
{
    {DRIVER_AMBAPP_GAISLER_GRGPIO_ID, 0, NULL}, /* Used by CPU0 */
    {DRIVER_AMBAPP_GAISLER_GRGPIO_ID, 1, &grlib_drv_res_grgpio1[0]}
};
```

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 0x60000000 - 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.

2.9.4.1. MP testsuite

The MP testsuite is located in the sources under testsuite/mptests, it requires modifications to the make scripts in order to select a unique image RAM location. The default shared memory area is at 0x40000000-0x40000fff, the two images for node1 and node2 needs to be located on a unique address and the heap/stack must also fit. The argument -Wl,-Ttext,0x40001000 for node1 and -Wl,-Ttext,0x40200000 for node2 can be added to the link stage, and the entry point (0x40001000 and 0x40200000) and stacks (0x401ffff0 and 0x403ffff0) must also be set by the loader (GRMON or mkprom for example). Depending on where the RAM memory is located and how much memory is available the parameters may vary.

2.9.5. RTEMS SMP AMP example

An AMP demonstration including RTEMS SMP targeting GR740 is distributed with RCC and is installed in the directory `src/samples/gr740/amdemo`. The demonstration consists of two applications: one single-processor RTEMS application and one RTEMS SMP application. Two different AMP configurations are demonstrated:

`ampdemo-smp0`

CPU0, CPU1 and CPU2 execute an RTEMS SMP application. CPU3 is started from the RTEMS SMP application and executes an independent single-processor application.

`ampdemo-sp0`

CPU0 executes a single-processor application. CPU0 starts (wakes up) CPU1 which together with CPU2 and CPU3 execute an RTEMS SMP application.

The purpose of the example is to demonstrate how RTEMS SMP can be used to start other operating system instances as well as itself being a slave instance in an AMP setting. Initialization of the interrupt controller by means of GRMON commands is also demonstrated: this is typically performed by a boot loader when the system is deployed. The preparation script configures a set of hardware breakpoints to assert that the operating system instances do not interfere with each others GPTIMER and APBUART and.

More information on how to run the RTEMS SMP AMP example is provided in `src/samples/gr740/amdemo/README`.

2.10. 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, wait-states, 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 -ramwvs 1 -romws 3 -freq 40 hello.exe
```

Note that `mkprom` creates binaries for LEON2/3/4 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-gaisler-rtems4.12-objcopy -O srec hello.exe hello.srec
```


3. Examples

3.1. Overview

There is a samples collection distributed within the RCC under `rcc-cc-v1.3rc2/src/samples`. It contains some general RTEMS examples and some more specific LEON or boards specific examples.

3.2. Building

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

```
$ sparc-gaisler-rtems4.12-gcc -qbsp=leon3 -mcpu=leon3 -O2 rtems-hello.c -o rtems-hello
$ mkprom2 -leon3 -freq 40 -dump -baud 38400 -ramsize 1024 -rmw rtems-hello
$ sparc-gaisler-rtems4.12-objcopy -O srec rtems-hello rtems-hello.srec
```

Several example C programs can be found in `/opt/rcc-cc-v1.3rc2/src/samples`. This folder also includes a Makefile that can be used to build the examples. Building the examples for a predefined BSP configuration is done by calling **make <bsp-target>**. Calling **make** (without a target) will compile all the examples for all the pre-built BSPs. The CPU compiler flags are set to the same as when building the RTEMS BSP. The executables will be stored `bin/<bsp-target>`. Valid bsp-targets are the same listed in Section 2.2.1 `leon3`, `leon3_smp`, `gr712rc`, `gr712rc_smp` etc. Below is an example building for GR712RC

```
bash$ make gr712rc
make BSP=gr712rc ODIR=bin/gr712rc gr712rc_build
make[1]: Entering directory '/home/daniel/git/rtems/build-1.3/src/samples'
sparc-gaisler-rtems4.12-gcc -Wall -g -O2 -Werror -I./ -mcpu=leon3 -mfix-gr712rc \
-qbsp=gr712rc -DBRM_BM_TEST rtems-brm.c brm_lib.c -o bin/gr712rc/rtems-brm_bm
sparc-gaisler-rtems4.12-gcc -Wall -g -O2 -Werror -I./ -mcpu=leon3 -mfix-gr712rc \
-qbsp=gr712rc rtems-brm.c brm_lib.c -o bin/gr712rc/rtems-brm_rt
sparc-gaisler-rtems4.12-gcc -Wall -g -O2 -Werror -I./ -mcpu=leon3 -mfix-gr712rc \
-qbsp=gr712rc -DBRM_BC_TEST rtems-brm.c brm_lib.c -o bin/gr712rc/rtems-brm_bc
sparc-gaisler-rtems4.12-gcc -Wall -g -O2 -Werror -I./ -mcpu=leon3 -mfix-gr712rc \
-qbsp=gr712rc -DTASK_TX -DTASK_RX rtems-occan.c occan_lib.c -o bin/gr712rc/rtems-occan
sparc-gaisler-rtems4.12-gcc -Wall -g -O2 -Werror -I./ -mcpu=leon3 -mfix-gr712rc \
-qbsp=gr712rc -DMULTI_BOARD -DTASK_TX rtems-occan.c occan_lib.c -o bin/gr712rc/rtems-occan_tx
...
make[1]: Leaving directory '/opt/rcc-1.3-rc2/src/samples'
```

It is also possible to build a single example by calling **make <example>** or to build a prom image by calling **make <example>.mkprom**. In this case the executables will be stored in the root samples directory. When building individual examples it is possible to control the behaviour by setting the following variables.

BSP

By setting the BSP variable to one of the bsp-targets, then the hardware specific compilation flags for that cpu-target will be added when compiling and the matchin `-qbsp=BSP` flag will be added.

CFLAGS

Override common compilation flags

CPUFLAGS

Override the hardware specific compilation flags

MKPROMFLAGS

Override mkprom2 flags

ODIR

Executables output directory. By default `bin/BSP/`.

Below is the command line for building only the `hello-world` example using the BSP option to make the binary target a GR712RC system:

```
src/samples$ make BSP=gr712rc CFLAGS="-O0 -g" rtems-hello
sparc-gaisler-rtems4.12-gcc -O0 -g -I./ -mcpu=leon3 -mfix-gr712rc -qbsp=gr712rc \
rtems-hello.c -o rtems-hello
```

4. 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.

4.1. 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 support@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>
```

4.2. 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' 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                                Aeroflex Gaisler
LEON3 SPARC V8 Processor                  Aeroflex Gaisler
AHB Debug UART                           Aeroflex Gaisler
AHB Debug JTAG TAP                       Aeroflex Gaisler
GR Ethernet MAC                           Aeroflex Gaisler
LEON2 Memory Controller                  European Space Agency
AHB/APB Bridge                           Aeroflex Gaisler
LEON3 Debug Support Unit                 Aeroflex Gaisler
Nuhorizons Spartan3 I/O interfaced       Aeroflex Gaisler
OC CAN controller                        Aeroflex Gaisler
Generic APB UART                         Aeroflex Gaisler
Multi-processor Interrupt Ctrl            Aeroflex Gaisler
Modular Timer Unit                       Aeroflex Gaisler

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

grib> 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
gplib> run
Hello World
gplib>
```

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

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

4.3. 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 necessarily 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-4.10/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.

4.4. 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-gaisler-rtems4.12-gdb debugger do:

```
ddd --debugger sparc-gaisler-rtems4.12-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.

Cobham Gaisler AB
Kungsgatan 12
411 19 Gothenburg
Sweden
www.cobham.com/gaisler
sales@gaisler.com
T: +46 31 7758650
F: +46 31 421407

Cobham Gaisler AB, reserves the right to make changes to any products and services described herein at any time without notice. Consult Cobham or an authorized sales representative to verify that the information in this document is current before using this product. Cobham does not assume any responsibility or liability arising out of the application or use of any product or service described herein, except as expressly agreed to in writing by Cobham; nor does the purchase, lease, or use of a product or service from Cobham convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual rights of Cobham or of third parties. All information is provided as is. There is no warranty that it is correct or suitable for any purpose, neither implicit nor explicit.

Copyright © 2017 Cobham Gaisler AB