The ERC32 GNU Cross-Compiler System

Version 2.0.1
March 1999

Jiri Gaisler
European Space Research and Technology Centre (ESA/ESTEC)

European Space Agency

jgais@ws.estec.esa.nl

The ERC32 GNU cross-compiler system

1 Introduction

1.1 General

This document describes the ERC32 GNU cross-compiler system version 2.0. Discussions are provided for the following topics:

- contents and directory structure of ERC32CCS
- compiling and linking ERC32 applications
- usage of SIS and MKPROM
- debugging ERC32 application with GDB/SIS

The ERC32 GNU cross-compiler system is a multi-platform development system based on the GNU family of freely available tools with additional 'point' tools developed by Cygnus, OAR and ESTEC. The ERC32CCS consists of the following packages:

- EGCS/GCC C/C++ compiler
- GNAT Ada95 compiler
- GNU binary utilities
- RTEMS C/C++ real-time kernel
- Newlib standalone C-library
- SIS ERC32 simulator
- GDB debugger with ERC32 remote debugging monitor (rdbmon)
- DDD graphical front-end for GDB
- MKPROM boot-prom builder

1.2 News in version 2.0.1

This version of ERC32CCS contains the following changes with respect to 1.3.2:

- DDD version 3.1.3
- Mkprom version 1.2.1
- Egcs-1.1.1 C/C++ compiler
- Newlib-1.8.1 C library
- RTEMS 4.0.0 official release with simplified compile procedure
- Ada95 compiler based on gnat-3.11p and rtems-4.0.0 (full tasking and interrupt support)
- GDB version 4.17 with gnat-3.11p support
- SIS version 3.0.2
- Updated documentation

2 Installation and directory structure

2.1 Obtaining ERC32CCS

ERC32CCS is only distributed via anonymous ftp. The primary home of ERC32CCS is ftp://ftp.estec.esa.nl/pub/ws/wsd/erc32/erc32ccs. Two platforms are supported: SPARC Solaris-2.5.1 (or higher), and x86 linux (libc5). Sources for rtems, rdbmon and mkprom are provided with ERC32CCS, the remaining sources can be found at the usual GNU sites or at the OAR home page.

2.2 Installation on a Linux host

The ERC32CCS directory tree is compiled to reside in `/usr/local/erc32` on Linux platforms. After obtaining the gzipped tarfile with the binary distribution, un-compress and un-tar it in a suitable location - if this is not `/usr/local/erc32` then a link have to be created to point to the location of the ERC32CCS directory. The distribution can be installed with the following command:

```
cd /usr/local
gunzip -c erc32ccs-2.0.1-linux-tar.gz | tar xf -
```

After the compiler is installed, add `/usr/local/erc32/bin` to your search path.

Note: ERC32CCS is compiled on slackware linux using libc5 libraries. RedHat linux and some versions of Debian might contain outdated libc5 libraries which causes the compiler to fail. In this case obtain file redhat-libc5.tar.gz which contains updated libc5 libraries. To install them on a Redhat system, do as follows:

```
cd /usr/i486-linux-libc5/lib
tar xzvf redhat-libc5.tar.gz
/sbin/ldconfig -v
```

2.3 Installation on a Solaris host

The ERC32CCS directory tree is compiled to reside in `/opt/gnu/erc32` on Solaris platforms. After obtaining the gzipped tarfile with the binary distribution, un-compress and un-tar it in a suitable location - if this is not `/usr/local/erc32` then a link have to be created to point to the location of the ERC32CCS directory. The distribution can be installed with the following command:

```
cd /opt/gnu
gunzip -c erc32ccs-2.0.1-solaris-tar.gz | tar xf -
```

After the compiler is installed, add `/opt/gnu/erc32/bin` to your search path.

2.4 Contents of `/usr/local/erc32 (/opt/gnu/erc32)`

The created `erc32` directory with the following sub-directories:

bin	executables
doc	documentation
include	host includes
lib	host libraries
man	man pages
rtems	rtems libraries
rtemsnp	rtems libraries (no posix)
sparc-rtems	target libraries (ERC32)
src	various sources

2.5 ERC32CCS tools

The following tools are included in ERC32CCS:

ddd	graphic X11 front-end to GDB
fcheck	utility to check for FPU rev.B bugs
mkprom	boot-prom builder
protoize	GNU protoize utility
sis	ERC32 simulator
sis64	ERC32 simulator (with 64-bit time)
sparc-rtems-ar	library archiver
sparc-rtems-as	cross-assembler (with some FPU rev.B fixes)
sparc-rtems-c++	C++ cross-compiler
sparc-rtems-c++filt	utility to demangle C++ symbols
sparc-rtems-g++	same as sparc-rtems-c++
sparc-rtems-gasp	assembler pre-processor
sparc-rtems-gcc	C/C++ cross-compiler
sparc-rtems-gdb	debugger with ERC32 simulator and remote target interface
sparc-rtems-gdb64	debugger with ERC32 simulator (64-bit time)
sparc-rtems-gnatcmd	Utility to print all GNAT command switches
sparc-rtems-gnatmake	Ada make utility
sparc-rtems-gnatbind	Ada binder
sparc-rtems-gnatf	Ada syntax checker and cross-reference generator
sparc-rtems-gnatprep	Ada pre-processor
sparc-rtems-gnatbl	Ada bind and link
sparc-rtems-gnatkr	Ada file name kruncher
sparc-rtems-gnatpsta	Utility to print the *Standard* package
sparc-rtems-gnatchop	Ada source code splitter
sparc-rtems-gnatlink	Ada linker
sparc-rtems-gnatpsys	Utility to display the *System* package
sparc-rtems-gnatchp	Ada source code splitter
sparc-rtems-gnatls	Ada library lister
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-size	utility to display segment sizes
sparc-rtems-strings	utility to dump strings from executables
sparc-rtems-strip	utility to remove symbol table
unprotoize	GNU unprotoize utility

2.6 Documentation

An extensive set of documentation for all tools can be found in `doc` and `man`. The following documents are provided:

aarm.pdf	Annotated Ada 95 Reference Manual
as.bdf	Using **as** - the GNU assembler
bfd.pdf	Libbfd - the binary file description
binutils.pdf	The GNU binary utilities

cpp.pdf	The C Preprocessor
ddd.pdf	DDD - The Data Display Debugger
gcc.pdf	Using and porting GCC
gdb.pdf	Debugging with GDB
gnat_rm.pdf	GNAT reference manual
gnat_ug.pdf	GNAT User's guide
ld.pdf	Using ld - the GNU linker
mkprom.pdf	Mkprom manual page
rtems_dev.pdf	RTEMS Development environment guide
rtems_relnotes.pdf	RTEMS Release notes
rtems_sparc.pdf	RTEMS SPARC Applications supplement
rtems_user.pdf	RTEMS C User's manual (this is the one you want!)
sis.pdf	SIS - SPARC instruction set simulator manual
sparcv7.pdf	SPARC V7 Instruction set manual

Data sheets for the ERC32 chip-set are also provided:

mecspec.pdf	MEC rev.A Device specification
tsc961e.pdf	TSC961 Integer Unit User's manual
tsc962e.pdf	TSC962 Floating-point Unit user's manual
sysover.pdf	ERC32 System overview
erc32cba.pdf	ERC32 revision CBA bug list
erc32cca.pdf	ERC32 revision CCA bug list

The documents are all provided in PDF format, with searchable indexes. The GNU documents have embedded hyper-links and searchable document text. A free PDF viewer ('acrobat') can be downloaded from Adobe (http://www.adobe.com/).

2.7 Support

Additional information and commercial technical support for the various tools is available as follows:

rtems, gcc, newlib	OAR	http://www.oarcorp.com/
sis, rdbmon, mkprom	J.Gaisler (ESTEC)	jgais@ws.estec.esa.nl
ddd	-	http://www.cs.tu-bs.de/softech/ddd/
gnat	ACT	http://www.act-europe.fr/
gcc, gdb, newlib	Cygnus Support	http://www.cygnus.com/

3 Using ERC32CCS

3.1 General development flow

The compilation and debugging of an ERC32-based applications is done in the following steps:

1. Compile and link program with gcc
2. Debug program in standalone simulator (SIS) or with gdb
3. Debug program on remote target with gdb
4. Create boot-prom for a standalone application

The ERC32CCS-2.0 supports three types of applications; ordinary sequential C/C++ programs, multi-tasking real-time C/C++ programs based on the RTEMS kernel, and Ada-95 programs. Compiling and linking is done in much the same manner as with the host-based gcc and GNAT.

3.2 RTEMS applications

As of ERC32CCS-2.0, compiling and linking of RTEMS applications is again done by adding the rtems compiler switch to gcc. This will instruct gcc compiler driver to add RTEMS specific include paths and libraries. To compile and link a RTEMS application, use 'sparc-rtems-gcc':

```
sparc-rtems-gcc -g -rtems -O3 rtems-hello.c -o rtems-hello.exe
```

The various compilation switches are explained in the gcc manual (gcc.pdf) and the man-pages. The default load address is start of RAM, i.e. 0x2000000. Any load address can be specified through the -Ttext option (see gcc manual).

RTEMS is provided in two versions; with and without POSIX threads interface. If applications are written with the POSIX interface, add the -posix switch during compilation and linking:

```
sparc-rtems-gcc -posix -g -O3 posix-app.c -o posix-app.exe
```

Extensive documentation is provided on `RTEMS in doc/rtems_user.pdf.`

3.3 Sequential C-programs

Ordinary sequential C programs can be compiled without any particular switches to the compiler driver:

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

3.4 Ada95 programs

Compiling and linking an Ada95 program is easiest done through gnatmake:

```
sparc-rtems-gnatmake -g -O3 -gnatp dais.adb -largs -qgnat -rtems -bargs -r
```

Note that the binder and linker arguments have to be provided to enable the correct libraries.

Individual units can be compiled through gcc:

```
sparc-rtems-gcc -c -g -O3 -gnatp dais.adb
```

Binding and linking can also be done separately:

```
sparc-rtems-gnatbind -r dais.ali
sparc-rtems-gnatlink -qgnat -rtems -g -O3 dais.ali
```

For details on how to use gnat, see the GNAT User's Manual (`gnat_us.pdf`) and GNAT Reference Manual (`gnat_rm.pdf`). ERC32 interrupts can be attached using the Ada95 interrupt_attach method. The ERC32 interrupts (numbered 1 - 15), are mapped on Ada interrupt 17 - 31. Avoid ERC32 interrupt 13 which is used for the real-time clock. See the `irqtest` program in the examples directory on how to attach interrupts. The compiler is configured for a maximum of 20 tasks and 30 mutexes. If you need a different configuration, go to the erc32/src/libio directory, edit gnatinit.c and do a make install'. Alternatively, linking with a local gnatinit.c file is also possible. In this case the global file will not be used.

NOTE: the Ada compiler is only provided for testing purposes, and is not by any means validated or guaranteed. A commercial, validated version is available from Ada Core Technology (http://www.act-europe.fr/).

3.5 Making boot-proms

Both sequential C-programs and RTEMS applications are linked to run from beginning of ram at address 0x2000000. To make a boot-prom that will run on a standalone target, use the `mkprom` utility. This will create a compressed boot image that will load the application to the beginning of ram, initiate various MEC register, and finally start the application. `mkprom` will set all target dependent parameters, such as memory sizes, number of memory banks, waitstates, baudrate, and system clock. The applications do not set these parameters themselves, and thus do not need to relinked for different board architectures. The example below creates a boot-prom for a system with 1 Mbyte RAM, one waitstate during write, 3 waitstates for rom access, and 12 MHz system clock. For more details see the mkprom manual

```
mkprom -ramsz 1024 -ramws 1 -romws 3 hello.exe -freq 12 hello.srec
```

3.6 Simple examples

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

```
> sparc-rtems-gcc -nortems -g -O2 hello.c -o hello
> mkprom hello -o hello.exe
> sparc-rtems-objcopy --adjust-vma=0x2000000 -O srec hello.exe hello.srec
```

An Ada application compiled through gnatmake:

```
> sparc-rtems-gnatmake -g -O3 -gnatp dais.adb -bargs -r -largs -qgnat -rtems
sparc-rtems-gcc -c -g -O3 -gnatp dais.adb
sparc-rtems-gnatbind -aO./ -I- -r -x dais.ali
sparc-rtems-gnatlink -g -rtems -qgnat dais.ali
> sparc-rtems-size dais
text    data    bss     dec     hex     filename
204720  6392    32268   243380  3b6b4   dais
```

Several example C, C++ and Ada program can be found in `src/examples`. The RTEMS validation tests can be found in `src/examples/RTEMS`.

3.7 FPU rev.B bugs

The FPU rev.B have a bug that will make certain lddf/stdf sequences fail. The compiler only rarely emits these sequences. The occurrence of such sequence can be check with the provided fcheck program. A modified assembler is also provided which will automatically insert NOPs in the failing sequences to correct this problem. The modified assembler also emits NOPs between ldf/fpop sequences with dependencies to circumvent a second FPU bug which is only occur if waitstates are used.

4 Execution and debugging

The applications built by ERC32CCS can be executed in four different ways; on the standalone simulator, on gdb with integrated simulator, on a remote target connected to gdb and on a standalone target board from prom.

4.1 Standalone simulator

The standalone simulator can run both application produced by the compiler and srecord images produced by MkProm. The following example shows how the 'hello world' program is run:

```
tellus > sis hello

 SIS - SPARC instruction simulator 3.0.2,  copyright Jiri Gaisler 1995-1998
 Bug-reports to jgais@ws.estec.esa.nl

loading hello:
section .text at 0x02000000 (26032 bytes)
section .data at 0x020065b0 (1304 bytes)
section .bss at 0x02006ac8 (40 bytes)(not loaded)
serial port A on stdin/stdout
sis> go
Hello world
IU in error mode (257)
     2567  02000800  91d02000  ta  0
sis>
```

Note that the program was started from address 0x2000000, the default start address. Programs always halt the IU after they have terminated, that is why the IU goes into error mode. The boot-prom image can also be simulated:

```
tellus > sparc-rtems-sis hello.srec

SIS - SPARC instruction simulator 3.0.2,  copyright Jiri Gaisler 1995-1998
 Bug-reports to jgais@ws.estec.esa.nl

loading hello.srec:
section .sec1 at 0x00000000 (16784 bytes)
serial port A on stdin/stdout
sis> run
ERC32 boot loader v1.0

  initialising RAM
  decompressing .text
  decompressing .data

  starting hello

Hello world!
IU in error mode (257)
sis>
```

4.2 GDB with simulator

To do symbolic debugging of both C and Ada applications, use gdb. After gdb is started, the simulator has to attached and the program loaded. It is important that the applications have been compiled with the -g switch. Below is a sample session:

```
tellus > sparc-rtems-gdb hello
(gdb)tar sim

 SIS - SPARC instruction simulator 3.0.2
 Bug-reports to Jiri Gaisler ESA/ESTEC (jgais@ws.estec.esa.nl)
serial port A on stdin/stdout
Connected to the simulator.
(gdb)
(gdb) load
(gdb) break main
Breakpoint 1 at 0x20014e4: file hello.c, line 4.
(gdb) run
Starting program: /home/jgais/erc32/src/examples/hello

Breakpoint 1, main () at hello.c:4
4               printf("Hello world!\n");
(gdb) cont
Continuing.
Hello world!

Program exited normally.
(gdb)
```

4.3 GDB with remote target

To attach gdb to a remote targets similar to attaching to the simulator. The baud rate for the serial port has to be specified and the remote target monitor has to run on the target. Also, a tip window should be connected to UART A to see the application output. Below is a sample session with a remote target:

```
tellus> xterm -e tip -38400 /dev/ttya &
[234]
tellus > sparc-rtems-gdb hello
(gdb) set remotebaud 38400
(gdb) tar erc32 /dev/ttyb
Remote debugging using /dev/ttyb
0x2000000 in trap_table ()
(gdb) lo
Loading section .text, size 0x65e8 vma 0x2000000
Loading section .data, size 0x4d0 vma 0x20065e8
(gdb) bre main
Breakpoint 1 at 0x20014e4: file hello.c, line 3.
(gdb) run
Starting program: /home/jgais/erc32/src/examples/hello
main () at hello.c:4
```

```
3           printf("Hello world!\n");
(gdb) cont
Continuing.

Program exited with code 03.
(gdb)
```

Note that the program has to be loaded each time before it is started with 'run'. This is to initialise the data segment to the proper start values. It is possible to switch between several targets (real or simulated) in the same GDB session. Use the GDB command **detach** to disconnect from the present target before attaching a new one.

4.4 Using DDD

DDD is a graphical front-end to gdb, and can be used regardless of target. To start DDD with the debugger use:

```
ddd --debugger sparc-rtems-gdb --attach-window
```

A small script, `dddx`, is provided to start DDD in this configuration. You might need the full path in front of DDD if you already have a version of ddd installed. To get the source code displayed in the ddd window, click on **locate()**. 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 manual in erc32/doc (ddd.pdf) or the on-line manual under the 'Help' menu.

4.5 Remote target monitor

The directory `src/rdbmon` contains the remote monitor which needs to be running on the target board to allow remote target debugging with gdb. The monitor supports 'break-in' into a running program by pressing Ctrl-C in GDB or **interrupt** in DDD. The two timers are stopped during monitor operation to preserve the notion of time for the application. Note that the remote debugger monitor only works with programs compiled with ERC32CCS, and thus NOT with programs compiled with Aonix Ada, VxWorks or similar.

Type make to build the monitor. Depending on desired baudrate type either 'make m38k4', 'make m19k2' or 'make m9k6'. Program the resulting *.srec file to you boot-prom. The remote debugger will be attached via UART B, console is on UART A. The maximum baudrate depends on the system clock of the target, 38K4 has been successfully used with a zero-waitstate ERC32 system running at 10 MHz. The monitor installs it self into the top 32K of ram. It therefore needs to know how large the ram is. The default ram size for the monitor is 2 Mbyte, adjust the Makefile if your system has different size.

5 Internals

5.1 Memory allocation

The resulting executables are in a.out format and has three segments; text, data and bss. The text segment is by default at address 0x2000000, followed immediately by the data and bss segments. The stack starts at top-of-ram and extends downwards. When the remote debugger is used, a part of the stack is automatically reserved for the debugger monitor.

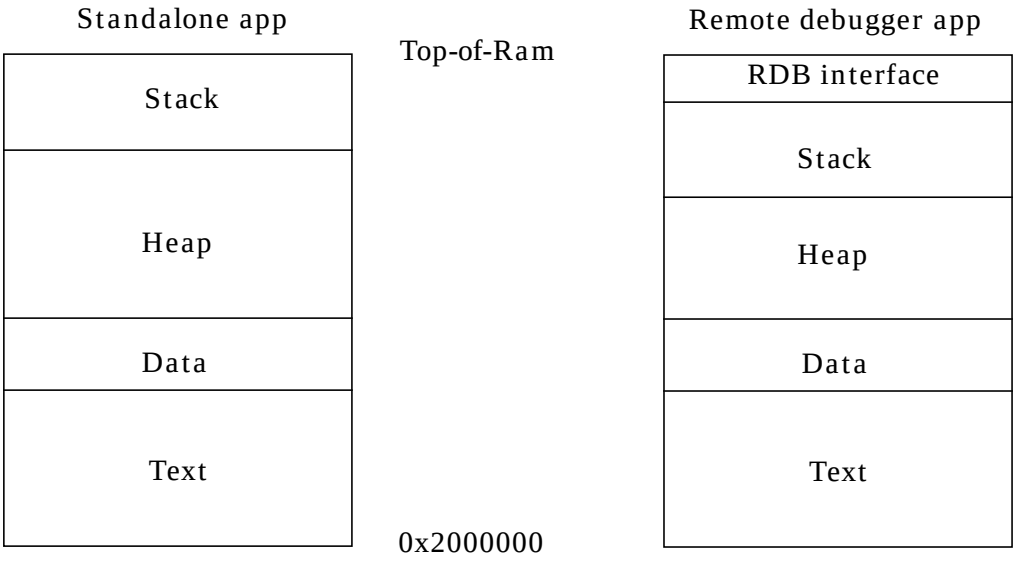

Figure 1: ERC32CCS applications memory map

By default, programs are linked assuming that the target system has 512Kbyte prom and 2 Mbyte ram. These settings can be overridden at compile time using the linker's --defsym option. On a system with 1 Mbyte prom and 8 Mbyte ram, use:

```
sparc-rtems-gcc -Xlinker --defsym -Xlinker _PROM_SIZE=1M -Xlinker --defsym
-Xlinker _RAM_SIZE=8M ....
```

When using gnatmake, these options should be added after the -cargs switch.

5.2 Sequential C-programs

For sequential C-programs, a posix compatible C-library and math library is provided. However, no file or other I/O related functions will work, with the exception of I/O to stdin/stdout. Stdin/stdout are mapped on UART A, accessible via the usual stdio functions. UART B can be accessed via file handle 3 (input) or 4 (output) using the read() and write() functions:

```
write(4,buf,size);
```

At startup of a program, the MEC real-time counter is programmed to increment one per microsecond. The function `clock()` will return the value of the counter. The sources to the board-specific part of the C-library is provided in `erc32/src/libio`. A user can modify the I/O functions according to his needs and install them into the C-library location (`erc32/sparc-rtems/lib`).