BCC

Bare-C Cross-Compiler

2020 User's Manual

The most important thing we build is trust

BCC User's Manual

Table of Contents

1. Introduction

1.1. Scope

BCC is a cross-compiler for LEON3 processors. It is based one the GNU compiler tools and the Newlib standalone C-library. The cross-compiler system allows compilation of both tasking and non-tasking C and C++ applications. It supports hard and soft floating-point operations, as well as SPARC V8 multiply and divide instructions. BCC can also be used to compile the eCos kernel.

BCC consists of the following packages:

- GNU GCC C/C++ compiler 3.4.4 and 4.4.2
- GNU Binutils 2.19.51
- Newlib C-library 1.13.1
- Low-level I/O routines for LEON3, including interrupt support
- uIP light-weight TCP/IP stack
- GDB debugger 6.4 with DDD and Insight Graphical front-end
- Linux and Windows/Cygwin hosts

LEON2 support has been dropped since BCC release 1.0.36d. LEON2 support is available in BCC 2.

1.2. BCC 1.0 life cycle status

This document (BCC-UM 1.0.52) describes BCC version 1.0.52 which is part of the BCC 1.0 series (1.0.x).

- BCC 1.0.x with GCC 3.4.4 has reached *End of life status* as of year 2020 and is not recommended for new development.
- BCC 1.0.x with GCC 4.4.2 has reached *Legacy status* and is not recommended for new development.
- BCC 2 is the current *Production status* bare-metal tool chain for LEON and is recommended for new development.

For more information about LEON software life cycle and software options, please visit the Cobham Gaisler website or contact Cobham Gaisler support (Chapter 5).

1.3. Installation

1.3.1. Host requirements

BCC is provided for two host platforms: GNU Linux/x86 and Microsoft Windows. The following are the platform system requirements:

Linux:	Linux-2.6.x, glibc-2.11 (or higher)

1.3.2. Linux / Cygwin

BCC is provided as a bzipped tar-file. It should be uncompressed in the `/opt` directory of the host:

```
$ mkdir /opt
$ tar -C /opt -xjf sparc-elf-[version-number].tar.bz2
```

After installation, add `/opt/sparc-elf-[gcc-version-number]/bin` to the PATH variable. This should be done by adding the following line to the file `.profile` in the home directory:

```
export PATH=/opt/sparc-elf-[gcc-version-number]/bin:$PATH
```

On Cygwin hosts, all installation steps should be done in a cygwin shell window. See http://www.cygwin.com for information on Cygwin.

1.3.3. Windows

BCC for Windows is provided for native Windows (MinGW) and for the Cygwin environment. For the Cygwin version see previous section. The native version will not require any additional packages and can be run from a standard Command Prompt.

The native Windows version of BCC is packaged with zip. Use a tool like WinZip to uncompress it to a directory, e.g., `C:\opt`. Note that the directory must not contain spaces (or any other non-ASCII characters) as this will confuse the compiler.

To use the compiler the `bin` subdirectory, e.g., `C:\opt\bin`, must be added to the PATH environment variable. This can be done from the Control Panel:

```
System -> Advanced -> Environment Variables...
```

Se http://www.mingw.org for more information on MinGW and the optional MSYS environment.

1.4. Building from source

The source code for BCC is available from the Cobham Gaisler website. To build BCC from source, the following steps shall be performed:

- Untar the source archive to `[dir]`.
- Issue:

  ```
  $ cd [dir]; make download
  ```

  This will download the original GCC, binutils and newlibc sources.
- Issue:

  ```
  $ cd [dir]; make install
  ```

  This will untar all the downloaded original archives over the current sourcetree, preserving the LEON specific files.
- Issue

  ```
  $ cd [dir]; make all
  ```

  This will build the GCC 4.4.2 and 3.4.4 toolchains. The default prefix is `/opt`.

1.5. Support

BCC is provided freely without any warranties. Technical support can be obtained from Cobham Gaisler through the purchase of technical support contract. Please contact sales@gaisler.com for more details.

2. General development flow

2.1. Overview

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

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

BCC supports both tasking and non-tasking C/C++ programs. Compiling and linking is done in the same manner as with a host-based GCC, and will not be explained here. The produced binaries will run on LEON3 and LEON4 systems, without requiring any switches during compilation.

2.2. GCC options

All GCC options are described in detail in the GCC manual. Some useful options are:

`-g` generate debugging information - must be used for debugging with GDB.

`-msoft-float` emulate floating-point - must be used if no FPU exists in the system.

`-mcpu=v8` generate SPARC V8 mul/div instructions - needs hardware multiply and divide.

`-O2`, `-O3` or `-Os` optimize code for maximum performance or minimal code size.

`-qsvt` use the single-vector trap model.

`-mfix-b2bst` enable workarounds for GRLIB technical note GRLIB-TN-0009.

`-mfix-tn0013` enable workarounds for GRLIB technical note GRLIB-TN-0013.

`-mfix-gr712rc` enable workarounds applicable to GR712RC. `-mfix-gr712rc` enables workarounds for the following technical notes:

- GRLIB-TN-0009
- GRLIB-TN-0012
- GRLIB-TN-0013

`-mfix-ut700` enable workarounds applicable to UT700 and UT699E. `-mfix-ut700` enables workarounds for the following technical notes:

- GRLIB-TN-0009
- GRLIB-TN-0013

`-mtune=ut699` set UT699 specific parameters (gcc-3.4.4 and gcc-4.4.2).

`-qfix-tn0018` Enable workarounds for GRLIB technical note GRLIB-TN-0018.

Note that in GCC version 3.4.4 `-mcpu=v8` was called `-mv8` and `-mflat` is present:

`-mv8` generate SPARC V8 mul/div instructions - needs hardware multiply and divide.

`-mflat` do not use register windows (i.e. no save/restore instructions). This options is only available in gcc-3.4.4.

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

```
$ sparc-elf-gcc -msoft-float -g -O2 hello.c -o hello.exe
```

The default link address is start of RAM, i.e. 0x40000000 for LEON. Other link addresses can be specified through the `-Ttext` option (see GCC manual).

2.3. Floating-point considerations

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

2.4. LEON SPARC V8 instructions

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

Both LEON3 and LEON4 can also supports multiply and accumulate (MAC). The compiler will never issue those instructions, they have to be coded in assembly. Note that the BCC assembler and other utilities are based on a modified version of GNU binutils-2.15 that supports the LEON MAC instructions.

2.5. Alternate register windows organization (only for GCC 3.X)

The compiler normally produces binaries that assumes that the target processor has 8 register windows. However, by compiling and linking with the `-mflat` switch, it is possible to produce binaries that will run on processors with only 2 register windows.

`-mflat` affect performance and code size. Using `-mflat`, the code size will increase with ~10%, and the performance will decrease with the same amount. When creating boot proms (see below), it is essential that the same `-mflat` parameter is given to mkprom2, as was used when the binary was compiled. Any miss-match will produce a faulty prom image.

2.6. Single vector trapping

When the VHDL model is configured to support single vector trapping (SVT) the `-qsvt` switch can be used with the linker to build an image that uses a dispatcher rather than a static trap table. The saving amounts to ~4KiB for the trap table, however trap handling will be slower. The image will try to enable SVT on boot using `%asr17`.

2.7. Memory organization

The resulting executables are in ELF format and have three main segments; `text`, `data` and `bss`. The `text` segment is by default at address 0x40000000 for LEON3 and LEON4, followed immediately by the `data` and `bss` segments. The stack starts at top-of-ram and extends downwards. The area between the end of `bss` and the bottom of the stack is used for the heap.

2.8. NGMP, RAM applications located at address 0 and multibus systems

To create an application that is located at address 0, like when targeting a NGMP system, the option `-Wl,-msparcleon0` can be given to GCC or `-msparcleon0` to ld. (Until BCC version 1.0.40: On systems with multiple busses `-qambapp` can be given to GCC in the final link. This activates the AMBA PnP scan. From version 1.0.41 onward AMBA scanning is default).

2.9. Recommended compiler options for LEON systems

Table 2.1 contains recommended GCC 4.4.2 options related to code generation for LEON based systems. Options in the table apply also to GCC 3.4.4 when `-mcpu=v8` is changed to `-mv8`.

The recommendations in Table 2.1 apply to BCC version 1.0.52. Other toolchains and other versions of BCC may have other recommendations.

Table 2.1. Recommended compiler options for GCC 4.4.2

System	Recommended options for GCC 4.4.2
GR740 silicon revision 1	`-mcpu=v8 -Wl,-msparcleon0`
GR740 silicon revision 0, LEON4-N2X	`-mcpu=v8 -Wl,-msparcleon0 -mfix-tn0013`
GR712RC	`-mcpu=v8 -mfix-gr712rc -qfix-tn0018`
UT699E, UT700	`-mcpu=v8 -mfix-ut700 -qfix-tn0018`

System	Recommended options for GCC 4.4.2
UT699/EPICA-NEXT, SCOC3	`-mcpu=v8 -mtune=ut699 -qfix-tn0018`
LEON3FT and LEON3FT-RTAX systems with SPARC V8 `mul/div` based on GRLIB versions up to and including build 4174.	`-mcpu=v8 -mfix-b2bst -mfix-tn0013 -qfix-tn0018`
LEON3FT and LEON3FT-RTAX systems with SPARC V8 `mul/div` based on GRLIB version 4175 to 4206	`-mcpu=v8 -mfix-tn0013 -qfix-tn0018`
LEON3FT and LEON3FT-RTAX systems with SPARC V8 `mul/div` based on GRLIB version 4207 to 4248.	`-mcpu=v8 -qfix-tn0018`
LEON3FT and LEON3FT-RTAX systems with SPARC V8 `mul/div` based on GRLIB version 4249 and later.	`-mcpu=v8`
LEON3 systems with SPARC V8 `mul/div` implemented without cache parity protection.	`-mcpu=v8` For GRLIB version up to and including 4206, also add • `-mfix-tn0013`
LEON3/LEON3FT systems without SPARC V8 `mul/div`.	Do not use `-mcpu=v8`, but otherwise follow the recommendations in this table.
LEON2 systems (AT697)	Not supported

2.10. Making LEON boot-proms

To make a boot-prom that will run from the prom on a standalone LEON3 or LEON4 target, use the mkprom2 utility freely available at the Cobham Gaisler website. It will create a compressed boot image that will load the application to the RAM, initialize various LEON registers, and finally start the application. mkprom2 will set all target dependent parameters, such as memory sizes, memory waitstates, UART baudrate, and system clock. The applications compiled with sparc-elf-gcc do not set these parameters themselves, and thus do not need to be re-linked for different board architectures.

The example below creates a boot-prom for a system with 1 Mbyte RAM, one RAM waitstate, 3 waitstates for ROM access, and 25 MHz system clock.

```
$ mkprom2 -ramsize 1024 -ramws 1 -romws 3 -freq 25 hello.exe -msoft-float
```

Note that mkprom2 creates ELF files. To create an SRECORD file for a prom programmer, use objcopy:

```
$ sparc-elf-objcopy -O srec hello.prom hello.srec
```

It is essential that the same `-mflat`, `-qsvt` and `-msoft-float` parameters are given to mkprom2, as was used when the binary was compiled. Any miss-match will produce a faulty PROM image.

For more information on how to use mkprom2, see the mkprom2 users manual available at Cobham Gaisler website.

3. Libraries

3.1. Newlib and Stdio

BCC applications use Newlib, which is a POSIX compatible C-library with full math support. However, no file or other I/O related functions are supported, with the exception of I/O to stdin/stdout. Stdin/stdout are mapped on UART A, accessible via the usual stdio functions.

3.2. Time functions

The LEON timers are used to generate the system time. The function clock() will return the time expired in microseconds. The gettimeofday(), time() and times() can also be used to get the time. Before the time functions can be used, leonbare_init_ticks() should be called to start the LEON timers and install the timer interrupt handler:

```
#include <asm-leon/timer.h>
void leonbare_init_ticks();
```

This will initialize Timer1 and Timer2. Timer1 is used to generate ticks at 100Hz while Timer2 is used to create high resolution timer events. Timer1 ticks can be used by installing a ticker callback at:

```
tickerhandler ticker_callback;
```

Timer2 timer events can be generated by initializing a struct timerevent structure and calling

```
#include <asm-leon/timer.h>
int addtimer(struct timerevent *e);
```

struct timerevent 'expire' field is the timeposition at which the event should be triggered. The current time can be retrieved using int gettimeofday(struct timeval *tv, struct timezone *tz);

3.3. Task switching

Task switching is supported by the functions:

```
#include <contextswitch.h>
int thread_setjmp(threadctx_t env, int val);
void thread_longjmp(threadctx_t env, int val);
```

thread_longjmp() will save the current register windows to the stack and jump to the stack previously saved by thread_setjmp() similar to clib's setjmp and longjmp construct. You can create your own scheduler by using a construct like:

```
void sched() {
...
thread_longjmp(next());
}
...
if (!thread_setjmp(self()))
sched();
...
```

3.4. Interrupt handling

Installing an interrupt handler is done by initializing member handler of a global variable struct irqaction and calling:

```
#include <asm-leon/irq.h>
void chained_catch_interrupt (int irq, struct irqaction *a );
```

where irq is the irq number (1 - 15). The supplied struct irqaction will be inserted in a list and therefore should be global. The simple void *catch_interrupt(void func(int irq), int irq); is also supported which uses chained_catch_interrupt internally.

The source code for libgloss (`libleonbare.a`) can be found in the `src/libgloss` directory.

For systems using the extended LEON3 interrupt controller with support for up to 31 interrupts it is possible to use irq 1-31 with catch_interrupt() and chained_catch_interrupt().

An example on how to install an interrupt handler is supplied in the `src/examples/c-irq.c` example of the BCC distribution.

Low-level interrupt processing takes around 40 instructions to set up the C enviroment for the interrupt handler and another ~25 instruction to dispatch irq to the associated handler. If very fast processing is required, a custom lowlevel assembly irqroutine can be installed using:

```
#include <asm-leon/irq.h>
void lolevelirqinstall(int irqnr,void (*handler)());
```

This will install the instructions:

```
sethi    %hi(handler), %l4;
jmpl    %l4 + %lo(handler), %g0;
nop
```

at address traptable+0x100+(irqnr*16). The callers low-level interrupt routine has to ensure proper enviroment setup before calling a C routine. This includes saving volatile register, checking for invalid windows and avoiding nested irqs. An appropriate routine would be written in assembler.

In case of single vector trap schemes (`-qsvt`) you have to use the following funtion to insert an irq handler:

```
int svtlolevelirqinstall(int trap,void (*handler)())
```

In case of `-qsvt` a table is used to dispatch the traps:

```
struct svt_trap_entry {
int start,end;
void (*func)(void);
};
extern struct svt_trap_entry trap_table[28];
```

Where start and end specify the range of traps that handler func should process. The last entry in the table should be {0,0,0}. You can modify the table by hand or use svtlolevelirqinstall to install a interrupt handler for you. Note that the irq number is trap number + 0x10. The symbol svt_trap_table_ext_end marks the end of the trap dispatch table. To insert a trap handler in `-qsvt` mode you can use the function:

```
int svtloleveltrapinstall(int trap,void (*handler)());
```

Using svtlolevelirqinstall(irq,handler) is equivalent to svtloleveltrapinstall(irq+0x10,handler).

```
Trap pre-amble
--------------
1572  400001a0  ae10200a  mov      10, %l7
1579  400001a4  a1480000  mov      %psr, %l0
1580  400001a8  108022fc  ba       0x40008d98
1581  400001ac  a7500000  mov      %wim, %l3

1582  40008d98  2d000004  sethi    %hi(0x1000), %l6
1587  40008d9c  a02c0016  andn     %l0, %l6, %l0
1588  40008da0  2d100023  sethi    %hi(0x40008c00), %l6
1595  40008da4  ac15a1a8  or       %l6, 0x1a8, %l6
1596  40008da8  29100025  sethi    %hi(0x40009400), %l4
```

```
etraps.s save state
--------
1597  40008dac  81c52170  jmp      %l4 + 0x170
1599  40008db0  932de008  sll      %l7, 8, %o1
1606  40009570  aa27a138  sub      %fp, 312, %l5
1613  40009574  c2256074  st       %g1, [%l5 + 0x74]
1616  40009578  c43d6078  std      %g2, [%l5 + 0x78]
1620  4000957c  c83d6080  std      %g4, [%l5 + 0x80]
1624  40009580  cc3d6088  std      %g6, [%l5 + 0x88]
1634  40009584  15100029  sethi    %hi(0x4000a400), %o2
1635  40009588  d602a050  ld       [%o2 + 0x50], %o3
1639  4000958c  d6256134  st       %o3, [%l5 + 0x134]
1644  40009590  960560b0  add      %l5, 176, %o3
1651  40009594  d622a050  st       %o3, [%o2 + 0x50]

check for invalid window:

1654  40009598  a8102001  mov      1, %l4
1655  4000959c  a92d0010  sll      %l4, %l0, %l4
1656  400095a0  808d0013  andcc    %l4, %l3, %g0
1663  400095a4  02800013  be       0x400095f0
1664  400095a8  01000000  nop
1665  400095f0  81c5a008  jmp      %l6 + 0x8
1673  400095f4  9c100015  mov      %l5, %sp

back in irqtrap_fast.s:
check for nested_irq flag + set pil

1674  40008db0  932de008  sll      %l7, 8, %o1
1675  40008db4  92140009  or       %l0, %o1, %o1
1676  40008db8  11100029  sethi    %hi(0x4000a400), %o0
1677  40008dbc  90122054  or       %o0, 0x54, %o0
1678  40008dc0  d0020000  ld       [%o0], %o0
1688  40008dc4  80a00008  cmp      %o0
1691  40008dc8  22800002  be,a     0x40008dd0
1692  40008dcc  92126f00  or       %o1, 0xf00, %o1
1693  40008dd0  818a6020  mov      %o1, 0x20, %psr
1700  40008dd4  01000000  nop
1701  40008dd8  01000000  nop
1702  40008ddc  01000000  nop


--------------------------------------------------------------


call routine catch_interrupt.c: handler_irq():

1703  40008de0  90100017  mov      %l7, %o0
1710  40008de4  40000028  call     0x40008e84
1711  40008de8  9203a0f0  add      %sp, 240, %o1
1712  40008e84  9de3bf98  save     %sp, -104, %sp
1713  40008e88  03100029  sethi    %hi(0x4000a400), %g1
1714  40008e8c  9b2e2002  sll      %i0, 2, %o5
1715  40008e90  82106228  or       %g1, 0x228, %g1
1722  40008e94  e000400d  ld       [%g1 + %o5], %l0
1723  40008e98  80a42000  cmp      %l0
1726  40008e9c  02800018  be       0x40008efc
1727  40008ea0  a4102001  mov      1, %l2
1734  40008ea4  10800007  ba       0x40008ec0
1735  40008ea8  da040000  ld       [%l0], %o5
1739  40008ec0  80a36000  cmp      %o5
1748  40008ec4  02bffffa  be       0x40008eac
1749  40008ec8  23100029  sethi    %hi(0x4000a400), %l1
1750  40008ecc  c2046124  ld       [%l1 + 0x124], %g1
1754  40008ed0  90100018  mov      %i0, %o0
1761  40008ed4  80a06000  cmp      %g1
1762  40008ed8  12bffff5  bne      0x40008eac
1764  40008edc  94100019  mov      %i1, %o2
1765  40008ee0  d2042008  ld       [%l0 + 0x8], %o1
1775  40008ee4  9fc34000  call     %o5
1777  40008ee8  e4246124  st       %l2, [%l1 + 0x124]

-- installed irq handler
1780  40001260  9de3bf98  save     %sp, -104, %sp
```

3.5. Extended IrqCtrl

The extended irq functionality is activated by the following code. Extended irq number is 13 in this example.

```
#include <asm-leon/irq.h>
extern struct irqmp_type irqmp;
...
irqmp.addr = 0x80000200;
irqmp.eirq = 13;
enable_irq(13);
...
```

irqmp.addr is the address of the irq controller, irqmp.eirq is the extended irq number. Having initialized the application like this you can register an irq handler for an irq > 15 using catch_interrupt(). Note that the extended irq number's interrupt handler itself is not called but the handler of the irq indicated by the extended irq ctrl's extended irq acknowledge register. Another possibility is of course to implement the extended irq handling yourself.

3.6. Interrupt nesting

The variable

```
extern unsigned int nestedirq;
```

can be set to 1 if irq nesteing is desired. It is set to 0 by default. In case of 0 the PSR's PIL will be set to 15 (highest) to keep the irq processing uninterrupted. If nestedirq is set to 1 the PSR's PIL will be set to the incoming irq's level, therefore causing higher level irq's to interrupt the current irq processing.

3.7. Installing custom irq handlers

To overwrite a compile-time generated traptable entry the function `traptable_genjmp()` can be used:

```
#include <asm-leon/leon3.h>
extern void traptable_genjmp(unsigned long *p, int i, int arg, unsigned int fn);
extern unsigned int sparc_leon23_get_tbr_base(void);
```

where *`p`* is the traptable base, *`i`* the traptable index, *`arg`* a 13 bit value in %l7 at the time of the traphandler call and *`fn`* the assembly function address to be called. The routine `sparc_leon23_get_tbr_base()` can be used to retrieve the current %tbr base value.

Below is a simple example that routes the `window_overflow` (0x5) trap call through `mynewhandler`:

```
...
#include <asm-leon/leon3.h>
...
void wrap(void) {
  __asm__ __volatile__("\n.global mynewhandler\nmynewhandler:\n"
                       "mov %%psr, %%l0\n"
                       "ba _window_overflow;nop\n"
                       ::);
}
extern void mynewhandler();
main () {
  traptable_genjmp((void *)sparc_leon23_get_tbr_base(), 5, 0, (int)&mynewhandler);
...
```

3.8. Small binary

Newlib `atexit()` introduces a dependency on `malloc()` which will add ~10KiB extra code. If you want to avoid this you can link against `libsmall.a` (`-lsmall`). `libsmall.a`'s `atexit()` supports only static 32 exit-function entries. The C library newlib `atexit()` function is declared weak and can be overridden.

The compiler option `-lsmall` removes references to `malloc()` by overriding the newlib `atexit()` function.

3.9. Amba PLUG and PLAY

Up to BCC 1.0.40: The option -qambapp can be given to GCC to enable PLUG and PLAY scanning for UART, timer and irq-ctrl across AHB2AHB bridges. The default setup only scanns the main BUS's configuration area at 0xfff00000.

From BCC 1.0.41 and upward: recursive scanning is enabled per default, -qnoambapp can be given to disable recursive scaning.

3.10. FreeRTOS

The sheduling library FreeRTOS is included in the BCC distribution. The precompiled library `libfreertos.a` was compiled using the configuration file supplied in `[installdir]/sparc-elf/include/freertos/FreeRTOSConfig.h`.

To recompile it with another configuration, goto [installdir]/src/freertos/, update FreeRTOSConfig.h and issue

```
$ make recompile
```

Additional sources can be added to `$(LIBOBJ)`.

Refer to the documentation available on the FreeRTOS website http://www.freertos.org for information on how to use the FreeRTOS API.

4. Execution and debugging

4.1. TSIM simulator and GRMON debug monitor

LEON applications can be debugged on either the TSIM simulator or on a hardware target connected with the GRMON debug monitor. Both TSIM and GRMON can be connected to the GNU debugger (sparc-elf-gdb) to perform source-level symbolic debugging.

For more information on GRMON and TSIM, see their respective user manuals.

4.2. Running on the TSIM simulator

To execute an application in the TSIM LEON simulator, use the **load** command to load the binary, and the **run** command to execute the application:

```
$ tsim-leon3

TSIM LEON SPARC simulator, version 2.0.3 (professional version)

Copyright (C) 2001, Gaisler Research - all rights reserved.
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)
tsim> load hello.exe
section: .text at 0x40000000, size 35120 bytes
section: .data at 0x40008930, size 2080 bytes
section: .jcr at 0x400091b4, size 4 bytes
tsim> run

starting at 0x40000000
Hello world!

Program exited normally.
tsim>
```

4.3. Debugging with GDB

To debug an application with GDB, start TSIM with the `-gdb` option (or issue the **gdb** command inside TSIM). TSIM by default listens on port 1234 for a GDB connection. This can be changed to any port using the TSIM `-port` switch at start-up.

```
$ tsim-leon3 -gdb

TSIM LEON SPARC simulator, version 2.0.3 (professional version)

Copyright (C) 2001, Gaisler Research - all rights reserved.
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)
gdb interface: using port 1234
```

Then, start GDB in a separate shell, load the application to the target, add optional breakpoints, and finally execute the application using the GDB **run** command:

```
$ sparc-elf-gdb hello.exe
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=sparc-tsim-elf"...
(gdb) target extended-remote localhost:1234
Remote debugging using localhost:1234
```

```
0x00000000 in ?? ()
(gdb) load
Loading section .text, size 0x8930 lma 0x40000000
Loading section .data, size 0x820 lma 0x40008930
Loading section .jcr, size 0x4 lma 0x400091b4
Start address 0x40000000, load size 37204
Transfer rate: 297632 bits in <1 sec, 275 bytes/write.
(gdb) break main
Breakpoint 1 at 0x40001384: file hello.c, line 4.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/jiri/samples/hello.exe

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

To re-execute the application, first re-load it to the target using the GDB load command and the issue **run** again.

4.4. Debugging on target hardware

To connect GRMON to a LEON system, start GRMON on the command line in a terminal shell. By default, GRMON will connect to the processor debug support unit (DSU) using a serial port of the host (ttyS0 or com1). See the GRMON manual for more information on how to connect via JTAG, PCI, ethernet or Spacewire. Once connected, the application can be downloaded and executed using the same procedure as when the simulator is used:

```
$ grmon -u

GRMON - The LEON multi purpose monitor v1.0.7

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

using port /dev/ttyS0 @ 115200 baud

initialising .........

Component                            Vendor
Leon3 SPARC V8 Processor             Gaisler Research
AHB Debug UART                       Gaisler Research
Ethernet DSU interface               Gaisler Research
LEON2 Memory Controller              European Space Agency
AHB/APB Bridge                       Gaisler Research
Leon3 Debug Support Unit             Gaisler Research
Generic APB UART                     Gaisler Research
Multi-processor Interrupt Ctrl       Gaisler Research
Modular Timer Unit                   Gaisler Research

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

grmon[grlib]> load hello.exe
section: .text at 0x40000000, size 35120 bytes
section: .data at 0x40008930, size 2080 bytes
section: .jcr at 0x400091b4, size 4 bytes
total size: 37204 bytes (99.4 kbit/s)
read 110 symbols
entry point: 0x40000000

grmon[grlib]> run
Hello world!

Program exited normally.
grmon[grlib]>
```

Connecting GDB to GRMON when attached to a real LEON target is done in the same way as when using the simulator. GRMON uses port 2222 by default to communicate with GDB.

4.5. Using the DDD graphical front-end to GDB

DDD is a graphical front-end to GDB, and can be used regardless of target. DDD must be started with the `--debugger` switch to select the sparc debugger, rather than the native GDB:

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

For further details on DDD operation, see the DDD web site: http://www.gnu.org/software/ddd/. DDD also has a built-in manual under the HELP menu in the main window.

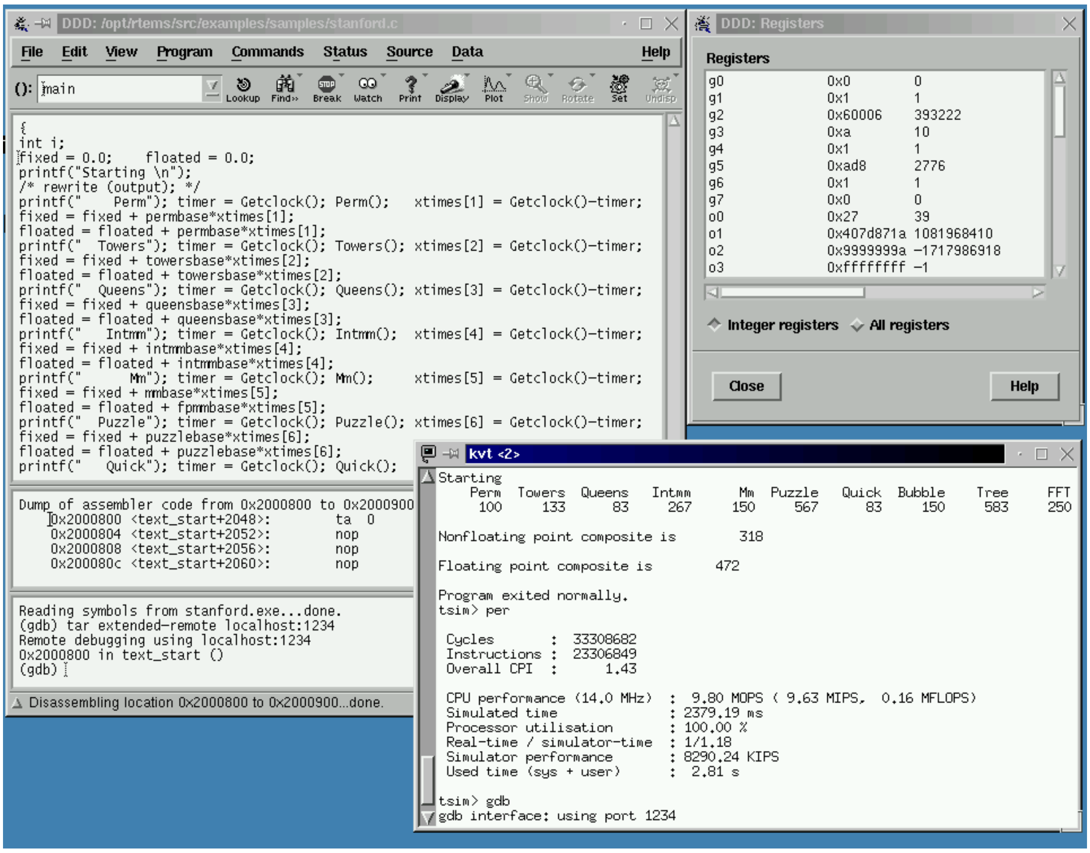

Figure 4.1. DDD with TSIM

Attaching to TSIM or GRMON is done in the same manner as when using sparc-elf-gdb without DDD. The GDB commands are entered in the bottom command window. Remember to load the application first, before issuing a **run** command. On Cygwin hosts, the Cygwin X-server must first be started by issuing startx in a Cygwin terminal. This will open an Xterm window, from which DDD should be started with the options mentioned above.

4.6. Using the Insight debugger

The Insight debugger is based on GDB-6.4 with an TCL/TK based graphical front-end. It can be used on both Linux and Cygwin hosts. The debugger is started with:

```
sparc-elf-insight app.exe
```

This will create the Insight main window:

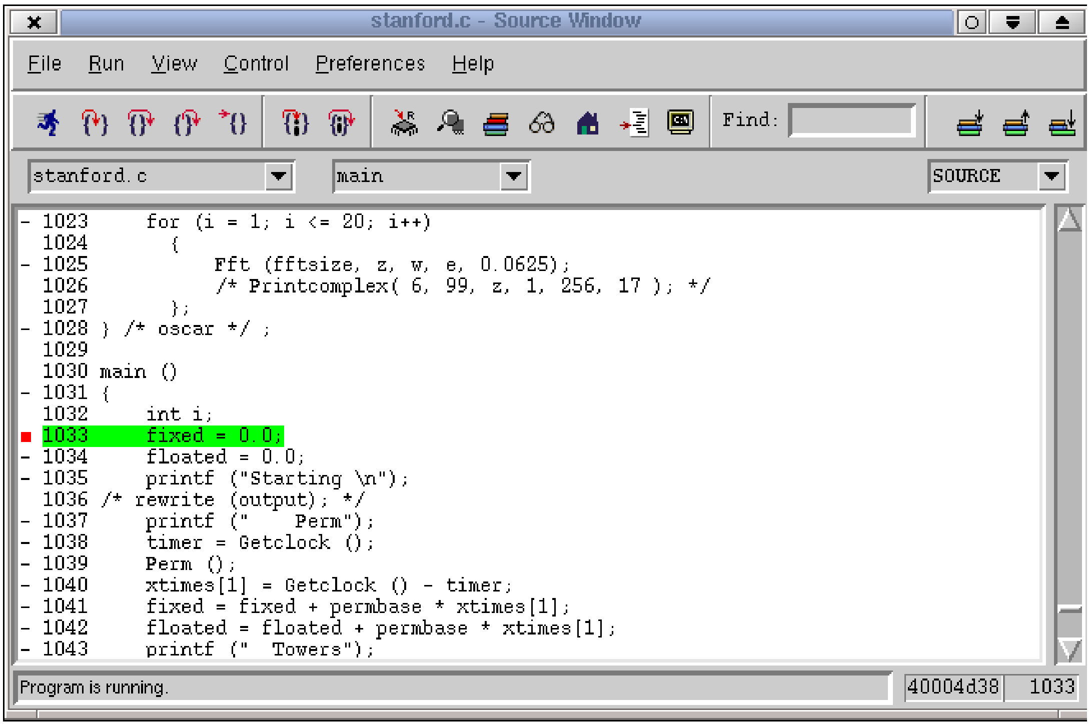

Figure 4.2. Insight main window

Clicking on the RUN button (or selecting Run->Connect) will open the **Connect to target** menu:

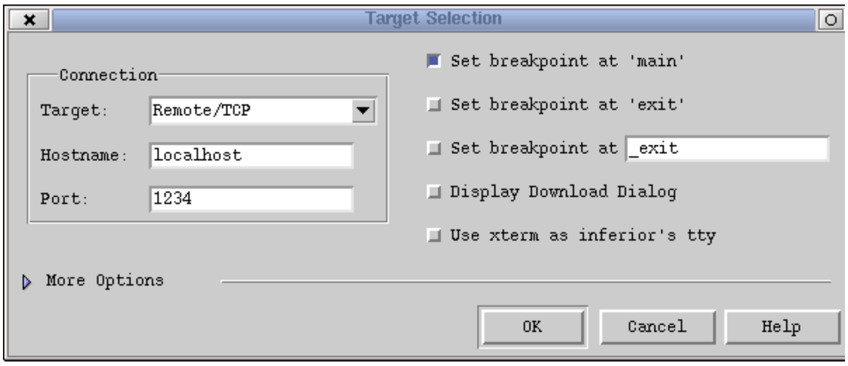

Figure 4.3. Insight target selection window

To connect to TSIM, select Remote/TCP and port 1234. To connect to GRMON, select port 2222. Enable the breakpoint on `main`, but disable the breakpoint on `exit`. Before clicking on OK, make sure that you have started TSIM or GRMON in a separate terminal, and entered GDB mode. Insight automatically downloads the application to the target when needed, so the load command does not have to be issued manually. To restart the application, just click on the **run** button again.

Insight requires at least TSIM version 2.0.5 or GRMON version 1.1.12.

5. Support

For support contact the Cobham Gaisler support team at support@gaisler.com.

When contacting support, 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.

The support service is only for paying customers with a support contract.

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