GDB commands for newbie

Prasanth Rajagopal

Analog Devices, Inc.

Fact Sheet

What is GDB? http://shop.fsf.org/product/debugging-gdb-gnu-source-level-debugger/

"GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed. GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another. "

What is GDB Server? http://en.wikipedia.org/wiki/Gdbserver

"gdb-server is a computer program that makes it possible to remotely debug other programs. Running on the same system as the program to be debugged, it allows the GNU Debugger to connect from another system; that is, only the executable to be debugged needs to be resident on the target system, while the source code and a copy of the binary file to be debugged reside on the developer's local computer. The connection can be either TCP or a serial line"

What is GDB Proxy? https://docs.blackfin.uclinux.org/doku.php?id=toolchain:debug:gdbproxy

When debugging under the gdb environment, the debugger needs to know what target to connect to. This target is a tiny server daemon, typically running on the host computer. It is equivalent to a gdbserver, although it does not run on the target hardware. This server is called gdbproxy and is derived from rproxy-0.7.

What is Eclipse? http://en.wikipedia.org/wiki/Eclipse_(software)

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, R. Ruby (including Ruby on Rails framework), Scala, Clojure, and Scheme.

What is GDB/MI Interface? http://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI.html#GDB_002fMI

GDB/MI is a line based machine oriented text interface to GDB and is activated by specifying using the --interpreter command line option (see Mode Options). It is specifically intended to support the development of systems which use the debugger as just one small component as a larger system {Eclipse}.

Lets learn from a simple no-hardware code. All gdb commands are sequentially tested…

```
int global_data = 1111;

void func1();
void func2();
int main(void);

int main(void)
{
	func1();

	return 0;

}

void func1()
{
	int local_data = 1234;

	global_data = 5678;

	func2();

}

void func2()
{

}
```

Start the GDB first:

C:\Program Files\Analog Devices\GNU Toolchain\2010R1\elf\bin>bfin-elf-gdb.exe gdb_debugging

GNU gdb 6.6

Copyright (C) 2006 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=i386-mingw32msvc --target=bfin-elf"...

(gdb) target remote localhost:2000

Remote debugging using localhost:2000

main () at ../src/gdb_debugging.c:9

9 func1();

(gdb) load

Loading section .rodata, size 0x8 lma 0xff800000

Loading section .eh_frame, size 0x4 lma 0xff800008

Loading section .ctors, size 0x8 lma 0xff80000c

Loading section .dtors, size 0x8 lma 0xff800014

Loading section .jcr, size 0x4 lma 0xff80001c

Loading section .data, size 0x40c lma 0xff800020

Loading section .text, size 0x510 lma 0xffa00000

Loading section .init, size 0x12 lma 0xffa00510

Loading section .fini, size 0xe lma 0xffa00522

Start address 0xffa00000, load size 2396

Transfer rate: 140941 bits/sec, 266 bytes/write.

Insert some break points – at the entry of each function:

```
(gdb) break main
Breakpoint 1 at 0xffa00274: file ../src/gdb_debugging.c, line 9.
(gdb) break func1
Breakpoint 2 at 0xffa00284: file ../src/gdb_debugging.c, line 17.
(gdb) break func2
Breakpoint 3 at 0xffa002ac: file ../src/gdb_debugging.c, line 28.
```

Run the code and it should hit breakpoints.

```
(gdb) continue
Continuing.

Breakpoint 1, main () at ../src/gdb_debugging.c:9
9           func1();
(gdb) continue
Continuing.

Breakpoint 2, func1 () at ../src/gdb_debugging.c:17
17          int local_data = 1234;
```

Let's play with breakpoints. See the information of the breakpoints, how it can be disabled.

```
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0xffa00274 in main at ../src/gdb_debugging.c:9
      breakpoint already hit 1 time
2   breakpoint     keep y   0xffa00284 in func1 at ../src/gdb_debugging.c:17
      breakpoint already hit 1 time
3   breakpoint     keep y   0xffa002ac in func2 at ../src/gdb_debugging.c:28
(gdb) disable 2
(gdb) info breakpoints
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0xffa00274 in main at ../src/gdb_debugging.c:9
      breakpoint already hit 1 time
2   breakpoint     keep n   0xffa00284 in func1 at ../src/gdb_debugging.c:17
      breakpoint already hit 1 time
3   breakpoint     keep y   0xffa002ac in func2 at ../src/gdb_debugging.c:28
```

Where are we now?

```
(gdb) where
#0  func1 () at ../src/gdb_debugging.c:17
#1  0xffa00278 in main () at ../src/gdb_debugging.c:9
```

Single stepping the code…

```
(gdb) step

19          global_data = 5678;
```

Let's print the local variable..

```
(gdb) info local

local_data = 1234
```

And the global variable…

```
(gdb) print/x global_data

$1 = 0x457
```

Check out the dis-assembly

```
(gdb) disassemble 0xffa00280 0xffa002a0

Dump of assembler code from 0xffa00280 to 0xffa002a0:

0xffa00280 <func1+0>:   LINK 0x10;              /* (16) */

0xffa00284 <func1+4>:   R0 = 0x4d2 (X);         /*          R0=0x0x4d2(1234) */

0xffa00288 <func1+8>:   [FP -0x4] = R0;

0xffa0028a <func1+10>:  P2.H = 0xff80;          /* (-128)     P2=0x0xff800000<_global_impure_ptr> */

0xffa0028e <func1+14>:  P2.L = 0x24;            /* ( 36)      P2=0x0xff800024<global_data> */

0xffa00292 <func1+18>:  R0 = 0x162e (X);    /*          R0=0x0x162e(5678) */

0xffa00296 <func1+22>:  [P2] = R0;

0xffa00298 <func1+24>:  CALL 0x0xffa002a4 <func2>;

0xffa0029c <func1+28>:  UNLINK;

End of assembler dump.
```

We run again…

```
(gdb) continue
Continuing.


Breakpoint 3, func2 () at ../src/gdb_debugging.c:28
28      }
```

Look at the back-trace info of function calls and frame info.

```
(gdb) bt
#0  func2 () at ../src/gdb_debugging.c:28
#1  0xffa0029c in func1 () at ../src/gdb_debugging.c:21
#2  0xffa00278 in main () at ../src/gdb_debugging.c:9
(gdb) backtrace full
#0  func2 () at ../src/gdb_debugging.c:28
No locals.
#1  0xffa0029c in func1 () at ../src/gdb_debugging.c:21
        local_data = 1234
#2  0xffa00278 in main () at ../src/gdb_debugging.c:9
No locals.
(gdb) frame
#0  func2 () at ../src/gdb_debugging.c:28
28      }
```

Core register information...

```
(gdb) info registers

r0          0x162e   5678
r1          0xff80042c      -8387540
r2          0x0      0A
r3          0x0      0
r4          0x8000   32768
r5          0x989680 10000000
r6          0x0      0
r7          0x0      0
p0          0xff800178      0xff800178
p1          0x1      0x1
p2          0xff800024      0xff800024
p3          0x3f1ff80       0x3f1ff80
p4          0x3f22148       0x3f22148
p5          0xffc03060      0xffc03060
sp          0xffb00f9c      0xffb00f9c
fp          0xffb00f9c      0xffb00f9c
i0          0xffe01300      -2092288
i1          0xffe00300      -2096384
i2          0x7d801048      2105544776
i3          0x7f900840      2140145728
m0          0x0      0
m1          0x0      0
m2          0xff807ffc      -8355844
m3          0x0      0
b0          0x7fa7e73e      2141710142
b1          0xb5025ff6      -1258135562
b2          0x5b4597f7      1531287543
b3          0x4c3dac51      1279110225
l0          0x0      0
l1          0x0      0
l2          0x0      0
l3          0x0      0
a0x         0x0      0
a0w         0x3200000       52428800
a1x         0x0      0
a1w         0x2b53   11091
astat       0x2001025       33558565
rets        0xffa0029c      0xffa0029c <func1+28>
lc0         0x0      0
lt0         0xffa0005c      -6291364
lb0         0xffa0005d      -6291363
lc1         0x0      0
lt1         0xffa00366      -6290586
lb1         0xffa00367      -6290585
```

info registers contd...

```
cycles         0xa07d46a4      -1602402652
cycles2        0x0     0
usp            0xffb00ff4      0xffb00ff4
seqstat        0x0     0
syscfg         0x32    50
reti           0xffa000e8      -6291224
retx           0xffa004d6      -6290218
retn           0x986b21f       159822367
rete           0xffa002ac      -6290772
pc             0xffa002ac      0xffa002ac <func2+8>
cc             0x1     1
text_addr      0x0     0
text_end_addr  0x0     0
data_addr      0x0     0
fdpic_exec     0x0     0
fdpic_interp   0x0     0
ipend          0x0     0
```

Set some global data

```
(gdb) set global_data = 9999

(gdb) print global_data

$2 = 9999
```

GDB/MI interface is meant for IDDE such as Eclipse. There are of course some simple useful commands. But beware – the text out is terribly bad….

Setting up as usual.

```
C:\Program Files\Analog Devices\GNU Toolchain\2010R1\elf\bin>bfin-elf-gdb.exe --interpreter=mi
gdb_debugging
~"GNU gdb 6.6\n"
~"Copyright (C) 2006 Free Software Foundation, Inc.\n"
~"GDB is free software, covered by the GNU General Public License, and you are\n"
~"welcome to change it and/or distribute copies of it under certain conditions.\n"
~"Type \"show copying\" to see the conditions.\n"
~"There is absolutely no warranty for GDB.  Type \"show warranty\" for details.\n"
~"This GDB was configured as \"--host=i386-mingw32msvc --target=bfin-elf\"...\n"

(gdb)
target remote localhost:2000
&"target remote localhost:2000\n"
~"Remote debugging using localhost:2000\n"
~"main () at ../src/gdb_debugging.c:9\n"
~"9\t\tfunc1();\n"
^done
```

Dis-assembly

```
(gdb)
-data-disassemble -f gdb_debugging.c -l 15 -n 3 -- 0
^done,asm_insns=
[{address="0xffa00280",func-name="func1",offset="0",inst="LINK 0x10;\t\t/* (16) */"},
{address="0xffa00284",func-name="func1",offset="4",inst="R0 = 0x4d2 (X);\t\t/*\t\tR0=0x0x4d2(1234)
*/"},
{address="0xffa00288",func-name="func1",offset="8",inst="[FP -0x4] = R0;"}]
```

Reading and writing data

```
(gdb)
-data-read-memory 0xff800000 x 1 3 3
^done,addr="0xff800000",nr-bytes="9",total-bytes="9",next-row="0xff800003",prev-
row="0xff7ffffd",next-page="0xff800009",prev-page="0xff7ffff7",
memory=[{addr="0xff800000",data=["0x00","0x00","0x00"]},
{addr="0xff800003",data=["0x00","0x43","0x00"]},
{addr="0xff800006",data=["0x00","0x00","0xff"]}]
(gdb)
-data-write-memory 0xff800000 x 1 0x11
^done
(gdb)
-data-write-memory 0xff800001 x 1 0x22
^done
(gdb)
-data-write-memory 0xff800001 x 1 0x33
^done
(gdb)
-data-read-memory 0xff800000 x 1 3 3
^done,addr="0xff800000",nr-bytes="9",total-bytes="9",next-row="0xff800003",prev-
row="0xff7ffffd",next-page="0xff800009",prev-page="0xff7ffff7",
memory=[{addr="0xff800000",data=["0x11","0x33","0x00"]},
{addr="0xff800003",data=["0x00","0x43","0x00"]},
{addr="0xff800006",data=["0x00","0x00","0xff"]}]
(gdb)
```