GDB Server for SoCLib

The GdbServer tool is a software debugger for !SoCLib.

Overview

The GdbServer is able to manage all processors in a !SoCLib platform. It listens for TCP connection from <u>?Gnu</u> <u>GDB</u> clients. Once connected, clients can be used to freeze, run, step every processor in the platform, add breakpoints, catch exceptions and dump registers and memory content.

Implementation

The GdbServer contains no processor specific code and can be used to manage any !SoCLib processor model using the generic Iss interface. It is implemented as an Iss wrapper class. When the GdbServer is in use, it intercepts all events between the processor Iss model and the !SoCLib platform. This enables the GdbServer to access platform resources as viewed from the processor without modifying platform components or processor model source code. The GdbServer is able to freeze the wrapped processor model while the platform is still running.

In order to simplify the debug in a multi-processor context, all processors wrapped in a GdbServer will be frozen when a breakpoint is detected in one single processor.

Usage

Adding GdbServer support to your platform

Adding the GdbServer to your topcell is easy. First include the header:

#include "gdbserver.h"

Then replace processor instantiation:

```
// Without GdbServer
// soclib::caba::VciXcacheWrapper<soclib::common::Mips32ElIss> cpu0("cpu0", 0, maptab, IntTab(0)
// With GdbServer
soclib::caba::VciXcacheWrapper<soclib::common::GdbServer<soclib::common::Mips32ElIss> > cpu0(
```

Finally do not forget to update the platform description file:

Uses('caba:iss_wrapper', iss_t = 'common:gdb_iss', gdb_iss_t = 'common:mips32el'),

Iss v1 and XCacheWrapper example

For using the GdbServer with the legacy Iss v1 simulators (like mipsel) models, the platform description file should contain:

```
Uses('caba:vci_xcache_wrapper', iss_t = 'common:gdb_iss', gdb_iss_t = 'common:ississ2', iss2_t =
```

The topcell description (top.cpp) should contain:

soclib::caba::VciXcacheWrapper<soclib::common::GdbServer<vci_param, soclib::common::IssIss2<socl</pre>

Connecting with a GDB client

When the simulation is running, the GDB Server listen for client connections on TCP port 2346.

```
$ ./system.x mutekh/kernel-soclib-mips.out
```

Its easy to connect to the simulation with a suitable gdb client:

• First launch the gdb client

```
$ mipsel-unknown-elf-gdb mutekh/kernel-soclib-mips.out
GNU gdb 6.7
Copyright (C) 2007 Free Software Foundation, Inc.
```

• Then enter this first command at the prompt

```
(gdb) target remote localhost:2346
Remote debugging using localhost:2346
0xe010cef4 in cpu_atomic_bit_waitset (a=0x602002cc, n=<error type>) at /home/diaxen/proje
99 {
```

Note that you can avoid typing this command every time: you just have to copy it in a .gdbinit file in the directory where gdb is executed.

Processor state analysis

The processors are now frozen. Each processor is seen as a thread by the GDB client:

```
(gdb) info threads
4 Thread 4 (Processor mips_iss3) 0xe010ceec in cpu_atomic_bit_waitset (a=0x602002cc, n=<error
at /home/diaxen/projets/mutekh/cpu/mips/include/cpu/hexo/atomic.h:99
3 Thread 3 (Processor mips_iss2) 0xe010ce64 in lock_spin (lock=0x602002cc) at /home/diaxen/pr
2 Thread 2 (Processor mips_iss1) 0xe010d110 in gpct_lock_HEXO_SPIN_unlock (lock=0x602061e8) at
* 1 Thread 1 (Processor mips_iss0) 0xe010cef4 in cpu_atomic_bit_waitset (a=0x602002cc, n=<error
at /home/diaxen/projets/mutekh/cpu/mips/include/cpu/hexo/atomic.h:99</pre>
```

The first processor has thread id 1. A specific processor can be selected for registers examination with the thread command.

Note this does change processor used for single step execution though. (see advanced commands sections)

```
(gdb) thread 1
[Switching to thread 1 (Thread 1)]#0 0x6011d370 in sched_context_stop_unlock ()
```

Classical GDB debugging session takes place. Here is a register dump of the processor 0 (thread 1):

```
(qdb) info registers
                      v1
                                      a2
                v0
                           a0
                                           аЗ
     zero
           at
                                a1
R0
   00000000 0000ff00 0000001 00000000 60200338 00000001 00000000 e010e74c
      t0
           t1
                t2
                     t3
                           t4
                                t5
                                     t6
                                           t7
   R8
                s2
                      s3
      sO
           s1
                           s4
                                s5
                                     s6
                                           s7
t9
                k0
                      k1
      t8
                           gp
                                sp
                                     s8
                                           ra
R24 00000000 00000000 0000000 602007fc 60207ff0 60205ce8 60205ce8 e0101134
      sr
           lo
                hi
                     bad
                         cause
                                рс
   fsr
           fir
   0000000 0000000
```

Running code

The following rules apply:

- Managed processors begin executing code at simulation startup until a gdb client connect on port 2346.
- Processors may be forced to start in frozen state waiting for incoming gdb connection by adding the F flag to the SOCLIB_GDB environment variable.
- All the managed processors are frozen at the same time when the gdb client prompt is displayed.
- When using the continue command, all processors resume at the same time.
- Single step execution is **only** performed on the processor which was interrupted. User selection of a different processor for data examination with the thread command does **not** change this. (see advanced commands section below)

Exceptions catching rules:

- Processors are stopped when an exception occurs.
- The X flag can be added to the SOCLIB_GDB environment variable to globally disable exception catching.
- Some monitor commands can be used to tweak exception catching for each processors separately (see below).
- The S flag can be added to the SOCLIB_GDB environment variable to pause the simulation waiting for connection when an exception is catched.

Advanced commands

The gdb client offers a easy way to send server specific data though the monitor command. Our GdbServer takes advantages of the monitor command to provide useful advanced features:

• The GdbServer may be instructed to dump every inter-function branch to produce a calltrace on stderr. The set_loader function must be used on Gdb iss to enable this feature. This can be enabled globally by adding the C flag to the SOCLIB_GDB environment variable (or the Z flag for dumping only call to function's entrypoint); or on a per processor basis using the calltrace command:

```
(gdb) monitor calltrace 0  # disable for all processors
(gdb) monitor calltrace 1 2  # enable for thread 2 (processor 1)
```

• The processor (thread id) used for step by step execution may be forced for the next **single step** operation:

(gdb) monitor stepcpu 1

• The GdbServer may be instructed to break on processor exception or to let the processor jump in its exception handler transparently. When used with an extra parameter, this setting can apply to a single processor instead of all.

```
(gdb) monitor except 1# enable for all processors(gdb) monitor except 0 2# disable for thread 2 (processor 1)
```

Exception catching is enabled by default but can be disabled globally by adding the X flag to the the SOCLIB_GDB environment variable.

• An alternative way to set hardware watch point range is provided to bypass the sometime annoying gdb client watch point feature. It can be used to modify directly the read and write watching intervals. The following commands set a 4 bytes (default is cpu register width) read/write watch interval at 0x12345678 and then excludes read watching for 32 bytes range at 0x12345000. These watch points will be unknown to the gdb client and will be lost when the simulation terminates.

(gdb) monitor watch rw 0x12345678 (gdb) monitor watch -r 0x12345000 32

This kind of watch points can be added using the SOCLIB_GDB_WATCH environment variable too:

export SOCLIB_GDB_WATCH=address[,size][r][w]:...
export SOCLIB_GDB_WATCH=0x44440000,32w
 # write watch [0x44440000, 0x444400
export SOCLIB_GDB_WATCH=0x12340000r:0x45870000rw
 # read watch [0x12340000, 0x1234000

The W flag can be added to the the SOCLIB_GDB variable to just report watchpoint hit on stderr and avoid stopping the simulation to be less intrusive.

• The gdb protocol debug mode may be enabled to dump interaction between client and server:

(gdb) monitor debug 1

• The gdb server almost stops the simulation process when the instrumented virtual processors are frozen. This saves resources of the host machine during debugging sessions. However this behavior may be an issue when freezing other platform components is not desirable (Use of multiple GDB servers with different processors, critical I/O device latency, multi-threaded simulation...). The sleepms command can be used to tweak the simulator sleep time between each execution cycle when the processors are in frozen state. This value may be set to 0 to let the simulation running at full speed or to -1 to completely stop the simulation while processors are frozen. The SOCLIB_GDB_SLEEPMS environment variable can also be used to set this value. An integer ms value is expected. The default value is 100ms.

```
(gdb) monitor sleepms 10
```

• Executable files can be added during simulation, so that gdb_server is aware of new symbols and is then able to continue performing a correct calltrace:

(gdb) monitor load /path/to/my/file.exe

It can be view as the counterpart of the command add-symbol-file except it is done on server side (and the "load address" can not be specified, the file is loaded according to its internal information, e.g. LMA/VMA)

Flags memo

The following flags can be concatenated in the SOCLIB_GDB environment variable (eg, \$ export SOCLIB_GDB=SZ)

- F: start the simulation in a frozen state so it can be attached with a gdb client
- X: disable automatic break whenever an exception is caught, the exception handler will be called transparently
- S: make the simulation stop and wait for a gdb attachment whenever an exception is caught
- C: dump a trace of every inter-functions branch
- Z: same as C but display only function's entrypoint
- W: disable automatic break whenever a watchpoint is hit, just report it on stderr (the watchpoints are to be defined in SOCLIB_GDB_WATCH)

More informations on using the GDB client can be found on the <u>?The GNU Project Debugger</u> home page.