Application Note: AN10034

How to debug a multicore program using XGDB

This application note is a short how-to on programming/using the xTIMEcomposer tools. It shows how to debug a multicore program using XGDB.

Required tools and libraries

This application note is based on the following components:

- xTIMEcomposer Tools - Version 14.0.0

Required hardware

Programming how-tos are generally not specific to any particular hardware and can usually run on all XMOS devices. See the contents of the note for full details.

1 How to debug a multicore program using XGDB

A single instance of XGDB can be used to debug multicore programs. As an example, consider the following 3 core program. This program passes a token in a ring between each core:

The *process* function waits for the token on the input channel. Once received, it prints its own core name to the console then passes the token on to the next core in the chain.

```
void process(chanend cIn, chanend cOut, char coreName[], int init) {
  if (init) {
    cOut <: 1;
  }
  while (1) {
    cIn :> int;
    printstrln(coreName);
    cOut <: 1;
  }
}
```

The 3 instances of the *process* function are instantiated here. Note: The final *init* parameter is used purely to get the ball rolling in the first place.

```
int main() {
  chan c01, c12, c20;
  par {
    process(c20, c01, "core0", 0);
    process(c01, c12, "core1", 0);
    process(c12, c20, "core2", 1);
  }
  return 0;
}
```

Compile the above program ensuring that debug is enabled (-g):

2 To debug using XGDB from xTIMEcomposer Studio

Create a new debug configuration via *Run->Debug Configurations->xCORE Applications*. Set a breakpoint on the *printstr* line in the *process* function. (Note: This will install the breakpoint for all cores). Now start running the project. Execution will break when the breakpoint is reached. Consider the *Debug* view. You can see that the program is suspended in tile[0], core[0]. Continue execution. You can see that you are now suspended in tile[0], core[1]. Note: When you are suspended, you can see the current position of any core by expanding its stack tree in the *Debug* view, and by clicking on a particular node in the stack tree; the *Registers* and *Variables* views will be refreshed with the relevant contents.

3 To debug using XGDB from the command line

Start XGDB, connect to the simulator and set a breakpoint on the *printstr* line in the *process* function. When run, execution will suspend when the breakpoint is hit. Continuing will cause each core in turn to hit the breakpoint:

```
> xgdb a.xe
...etc...
(gdb) connect -s
0xffffc070 in ?? ()
(gdb) b multicore_usage.xc:22
Breakpoint 1 at 0x100e6: file multicore_usage.xc, line 22.
(gdb) run
...etc...
Breakpoint 1, process (cIn=1026, cOut=2, coreName=@0x10568, init=0) at multicore_usage.xc:22
22      printstrln(coreName);
(gdb) c
core0
[Switching to tile[0] core[1]]

Breakpoint 1, process (cIn=258, cOut=514, coreName=@0x10558, init=0) at multicore_usage.xc:22
22      printstrln(coreName);
(gdb) c
core1
[Switching to tile[0] core[2]]

Breakpoint 1, process (cIn=770, cOut=1282, coreName=@0x10560, init=1) at multicore_usage.xc:22
22      printstrln(coreName);
(gdb) c
core2
[Switching to tile[0] core[0]]

Breakpoint 1, process (cIn=1026, cOut=2, coreName=@0x10568, init=0) at multicore_usage.xc:22
22      printstrln(coreName);
(gdb) info threads
  3  tile[0] core[2]  0x000100fa in process (cIn=770, cOut=1282, coreName=@0x10560, init=1)
    at multicore_usage.xc:23
  2  tile[0] core[1]  process (cIn=258, cOut=514, coreName=@0x10558, init=0) at multicore_usage.xc:21
* 1  tile[0] core[0]  process (cIn=1026, cOut=2, coreName=@0x10568, init=0) at multicore_usage.xc:22
```

Note: As can be seen in the above XGDB trace, the *info threads* command can be used to see the current location of each of the threads. The currently active thread is marked by the asterisk. To explicitly change the currently active thread, for example, to view the register/variable contents, the *thread* command can be used. This accepts a thread ID as an argument.