Application Note: AN10017

How to find the location of a crash using XGDB

This application note is a short how-to on programming/using the xTIMEcomposer tools. It shows how to find the location of a crash 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 find the location of a crash using XGDB

If an exception occurs in the target code, XGDB can be used to find the location of the exception and diagnose its root cause. For example, compile the following code ensuring that debug is enabled (-g):

```
int divide(int x, int y) {
  return x / y;
}

int main() {
  divide(5, 0);
  return 0;
}
```

2 From within xTIMEcomposer Studio

Create a new debug configuration via *Run->Debug Configurations->xCORE Applications*, then start debugging. Execution will now break when the exception occurs. The *Debug* view will display the trap type information, in this case, ET_ARITHMETIC (caused by the divide by zero). It will also show the pc of the excepting instruction. For more details the *Registers* view can be used to find the contents of the exception data register, which depending on the type of exception that occurred, can provide more information as to its root cause.

3 From the command line

For example, start XGDB, connect to the simulator and start debugging. Execution will now break when the exception occurs, and the exception type will be displayed on the console. The stack trace of how this location in the program was reached can be found using the *backtrace* command. You can then look at the register contents to attempt to diagnose the root cause of the trap:

```
> xgdb a.xe
...etc...
(gdb) connect -s
0xffffc04e in ?? ()
(gdb) run
...etc...
Program received signal ET_ARITHMETIC, Arithmetic exception.
0x000100ba in divide (x=5, y=0) at find_the_location_of_a_crash.xc:10
10    return x / y;
(gdb) backtrace
#0  0x000100ba in divide (x=5, y=0) at find_the_location_of_a_crash.xc:10
#1  0x000100ce in main () at find_the_location_of_a_crash.xc:14
(gdb) print /x $spc
$1 = 0x100ba
(gdb) print /x $ssr
$2 = 0x0
(gdb) print /x $et
$3 = 0x7
(gdb) print /x $ed
$4 = 0x0
```

Note: In the above, the *spc* (saved program counter) and the *ssr* (saved status register) will contain the values in the *pc* and the *sr* at the time of the exception. The *et* and *ed* registers contain the exception type and exception data details. In this case, the exception type is 0x7, which corresponds to *ET_ARITHMETIC* (see xs1b_user.h). The exception data register contents in general depend on the type of exception. For example, for an *ET_RESOURCE* exception the exception data will contain the id of the resource causing the trap. However, in the case of arithmetic exceptions, the *ed* will be set to zero.