TECHNICAL NOTE	ONDEMAND MICROELECTRONICS
Project:	SVENm
Project No:	n.a.
Author:	L. Kervella
Document ID:	Svn_note034
Last modified:	2007-07-12, 5:17 PM
Version:	1.0
File: \\Odaxsun08\prj_svenm\pdoc\note\svm_note34_v1r1x.doc	

GDB chili short manual

1 Introduction

This document describes how to run the GNU debugger (GDB) with the CHILI simulator.

2 Revisions

Version	Date	Author	Description
1.0	2007-03-07	L. Kervella	Initial version
1.1	2007-07-12	L. Kervella	Updates

Compiling for the debugger

The obvious switch to set to gcc to produce a debuggeable binary is “-g”. Make sure that no optimization switches are on or set the “-O0” switch.

```
chilli-odm-elf-gcc –g test.c –o test.exe
```

Disassembling the DWARF2 debug information

You might never need to access this information but in case you might:

```
chilli-odm-elf-readelf –w test.exe
```

Starting the debugger

The order of the 3 following commands is important and can not exchange.

1. Starting gdb and loading the symbol table with:

```
chilli-odm-elf-gdb test.exe
```

2. Starting the chilli simulator (chilisim) and loaded the program in gdb:

```
(chili-odm-gdb) target sim "-z test.exe"
Connected to the simulator.

(chili-odm-gdb) load test.exe
```

The first command initiates the CHILI simulator and connects it to gdb. The arguments included between the double-quotes are chilisim arguments.

The second command is need to load the program into the gdb internal program representation.

Starting with DDD

```
ddd --debugger chili-odm-gdb
```

Debugging with the CHILI DMS simulation

The only difference with a standard simulation (outside gdb) is the double-quotes used to pass the arguments to the DMS engine(after the –x argument) need to be transform into single-quote.

```
(chili-odm-gdb) target sim "-d libdms2.so -e libdms2.so:createPort0 -
flibdms2.so:createPort1 –x '-c0:0x200000 -l0x1000000:0x100FFFF -
d0x10000000:0x10100000'  -z test.exe"
```

Debugging at Instruction-Level

The following commands are usefull to debug at instruction-level. They are all standard gdb commands. To get more informations about them, use the help command inside the debugger.

1. Register access:

To list the value of the 64 registers:

```
        (chili-odm-gdb) info regs
```

To display only one register

```
        (chili-odm-gdb) info regs 61
```

2. Disassembler:

```
        (chili-odm-gdb) disassembler
        (chili-odm-gdb) disas 0x000007e0 0x000007f0
```

With no argument, the current function is disassembled.

3. Frame information

```
        (chili-odm-gdb) info frame
        (chili-odm-gdb) frame
        (chili-odm-gdb) bt
```

4. Program counter

```
        (chili-odm-gdb) info program
```

5. Memory acces

```
        (chili-odm-gdb) print *(0x2000)
```

Debugging with optimized binaries

If you compile your application without the “-g” option but with some optimization switches (“-02” for example), gdb will have no symbol tables to display variables values or will not be able to show function stack frames. But debugging at instruction-level is still possible.

For example, you can set breakpoints with conditional expressions testing register values:

```
(chili-odm-gdb) b GetIntraPrediction4x4Luma
Breakpoint 2 at 0x544
(chili-odm-gdb) run
Starting program:
/local/users/lkervell/workspaces/head/chilitest/build/chili/bin/test012
...
...
Breakpoint 2, 0x00000544 in GetIntraPrediction4x4Luma ()

(chili-odm-gdb) cond 2 ($r1==6)
(chili-odm-gdb) c
Continuing.
```

```
Breakpoint 2, 0x00000544 in GetIntraPrediction4x4Luma ()
(chili-odm-gdb) p $r1
$3 = 6

(chili-odm-gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x0000ffcc <main>
        breakpoint already hit 1 time
2   breakpoint     keep y   0x00000544 <GetIntraPrediction4x4Luma>
        stop only if $r1 == 6
        breakpoint already hit 4 times
```

Notes

Please use the help command inside gdb to have a complete description of all available features. All commands have shortcuts (ex: “print” -> “p”).