

# Remote GDB debugging of AJIT Processor Cores

Titto Thomas, Madhav P. Desai  
Department of Electrical Engg.  
IIT Bombay  
Powai, Mumbai 400076

March 22, 2022

## 1 Introduction

The AJIT processor is available in a variety of configurations, ranging from a single core, single CPU thread setup to a multi-core setup with each core having two CPU threads. Each CPU thread implement either the SPARC-V8 ISA (the AJIT-32 thread), or the SPARC-V8 ISA with custom 64-bit extensions (the AJIT-64 thread). We describe the debug setup for the AJIT processor.

The GDB debugger is used as a remote client, while the role of the server is performed either by

- A C emulator model of the multi-core multi-thread processor
- An AJIT debug monitor utility which in turn controls the execution of the AJIT processor using a serial USB-UART link.

## 2 Using the debug setup with the AJIT C emulator model

Currently, the AJIT C emulator models the following system:

- Up to 4 cores, with each core either having 1 or 2 threads. The threads can either be AJIT-64 threads, or AJIT-32 threads.
- An interrupt controller, a timer, a serial device.
- A coherent memory.

### 2.1 Starting the emulation model

The AJIT C emulator model is invoked as

```
ajit_C_system_model ... options ....
```

The options relevant for debugging are summarized below

```
-n <number-of-cores>
    : specifies number-of-cores in the processor for this test.
    : default is 1, maximum is 4.
-t <number-of-threads-per-core>
    : specifies number-of-threads-per-core in the processor for this test.
    : default is 1, maximum is 2.
-u <32/64>
    : use -u 64 to run model in 64-bit mode [default is 32]
-m <mmap-file>
    : required, specifies memory-map of processor for this test.
-g, optional, to run the CORE in debug mode.
-p <gdb-port-number>, required with -g, to specify remote debug port.
-i <init-pc>, optional, for specifying initial value of PC (default=0). NPC is PC+4
```

For example, if we are debugging a machine with two cores, each of which has two AJIT-32 threads, we would use

```
ajit_C_system_model -m prog.mmap -n 2 -t -u 32\  
-g -p 8888 -p 8889 -p 8890 -p 8891
```

This will start four GDB servers listening on ports 8888-8891. Port 8888 is used for debugging (core=0, thread=0), port 8889 is used for debugging (core=0,thread=1) and so on.

## 2.2 Start the GDB clients

Since we are debugging four threads, we will start four clients in four shells (or in a TMUX session with four panes). For example, for core,thread (0,0), we start

```
sparc-linux-gdb prog.elf
```

Now a GDB shell will start and you can see the messages shown in Fig. 1



```
GNU gdb (GDB) 7.6.2  
Copyright (C) 2013 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law. Type "show copying"  
and "show warranty" for details.  
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=sparc-buildroot-linux-uclibc".  
For bug reporting instructions, please see:  
<http://www.gnu.org/software/gdb/bugs/>...  
Reading symbols from /home/titto/AJIT_Project/AjitRepoV2/processor/C/debugger/tests/output/watchpoints.elf...done.  
(gdb) █
```

Figure 1: GDB debug session

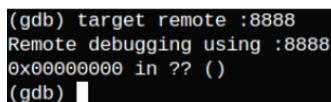
You will need to start a shell for each thread that you are debugging.

Now in the GDB shell for core,thread (0,0), type the same `port number`.

```
(gdb) target remote:8888
```

Do the same thing in the other four shells (with the appropriate port numbers).

The gdb side will also show the following messages confirming the connection establishment. Since the connection has now been established, you can control and monitor the program execution on the AJIT thread in real time.



```
(gdb) target remote :8888  
Remote debugging using :8888  
0x00000000 in ?? ()  
(gdb) █
```

## 3 Using the GDB debug setup with the `ajit_debug_monitor_mt` utility

TBD.

## 4 Basic GDB commands for use

### Continue / Next

When the execution is stopped due to some reason, you can let the processor continue with these commands.

```
(gdb) continue  
(gdb) c
```

Next command also lets the processor continue, but stops it after executing one more line of code.

```
(gdb) next  
(gdb) n
```

## 4.1 Breakpoints

Breakpoints are useful for stopping the execution on reaching specific points in the source code, and probe the processor for information. They can be set using the following commands.

```
(gdb) break <function_name>
(gdb) break <line_number>
(gdb) break <filename : function_name>
(gdb) break <filename : line_number>
(gdb) break *<address>
```

Example:

```
(gdb) break 24
Breakpoint 1 at 0xf0004044: file Watchpoints.c, line 24.
```

The AJIT thread being debugged will stop execution when a breakpoint is hit, and you will be notified in the corresponding GDB shell.

```
(gdb) continue
Continuing.

Breakpoint 1, main () at Watchpoints.c:24
24          j = 25;
(gdb) █
```

Breakpoints can be listed by the following command.

```
(gdb) info breakpoints
```

This command can be used to delete specific breakpoints.

```
(gdb) delete break <breakpoint_number>
```

## 4.2 Watchpoints

Watchpoints are useful for stopping the execution on modification of specific variables in the source code, and probe the processor for information. They can be set using the following commands. This command can be used to delete specific breakpoints.

```
(gdb) watch [-l|-location] <variable>
(gdb) break [-l|-location] <expression>
```

Ordinarily a watchpoint respects the scope of variables in expression. The `-location` argument tells gdb to instead watch the memory referred to by expression.

Example:

```
(gdb) watch t
Hardware watchpoint 2: t
```

The AJIT thread being debugged will stop execution when a watchpoint is hit, and you will be notified with their values in gdb.

```
(gdb) continue
Continuing.
Hardware watchpoint 2: t

Old value = 20
New value = 5
0xf0004058 in main () at Watchpoints.c:25
25          t = j - t;
(gdb) █
```

Watchpoints can be listed by the following command.

```
(gdb) info watchpoints
```

This command can be used to delete specific watchpoints.

```
(gdb) delete watch <watchpoint_number>
```

### 4.3 Traps

When the AJIT thread is executing a program, it will stop and inform the GDB client whenever there is a trap. After a trap, you can read the TBR register to get information about the trap type etc.

Example:

```
(gdb) c
Continuing.

AJIT: Trap Occured

Program received signal SIGTRAP, Trace/breakpoint trap.
halt () at /home/titto/AJIT_Project/AjitRepoV2/os/kernels/pico/src/sparc_stdio.c:7
7  __asm__ __volatile__("ta 0\n\t"
(gdb)
```

### 4.4 Setting variables / memory / registers

You can modify the values of variables, memory contents and register values with the set command.

```
(gdb) set var <variable> = <value>
(gdb) set $ <register> = <value>
(gdb) set *(int)<memory_address> = <value>
```

Example:

```
(gdb) set *(int)0xffffffff8 = 0x11
(gdb) set $l1=0x55
(gdb) set var t=0x50
```

### Printing variables / memory / registers

You can display the values of variables, memory contents and register values with the print or p command.

```
(gdb) print var <variable>
(gdb) print $ <register>
(gdb) print *(int)<memory_address>
```

Example:

```
(gdb) p/x $l1
$5 = 0x55
(gdb) p/x *(int)0xffffffff8
$6 = 0x50
(gdb) p/x t
$7 = 0x50
```

### Detach

You can stop debugging the program and let the processor continue till it finishes execution, using this command.

```
(gdb) detach
```

Example:

```
(gdb) detach
Detaching from program: /home/titto/AJIT_Project/AjitRepoV2/processor/C/debugger/tests/output/Watchpoints.elf, Remote target
Ending remote debugging.
```

## 4.5 Other useful commands

You can get help on any commands using

```
(gdb) h[elp]
```

```
(gdb) h[elp] <command>
```

Finish current function, loop, etc. with

```
(gdb) fin[ish]
```

Show lines of code surrounding the current point

```
(gdb) l[ist]
```

Delete all breakpoints

```
(gdb) d[ele]te
```

Add a list of gdb commands to execute each time a breakpoint is hit

```
(gdb) comm[ands] <breakpoint_number>
```