Adding Debugger to AJIT Processor C model

Ashfaque Ahammed <ashfaque@iitb.ac.in>

Titto Thomas <tittothomas@iitb.ac.in>

Instructor: Prof. Madhav P. Desai <madhav@ee.iitb.ac.in>

Abstract

This document describes the hardware-side design of gdb compatible debugging interface for Ajit processor. The hardware server will control the processor execution when running on debug mode. It communicates with gdb software bridge[1], and according to commands, it may read/write processor registers or memory to perform the debugging operation. When ever the processor meets a break-point or watch-point, the processor is halted and it keeps the processor in the same state until it receives a continue form gdb(user).

1 Introduction

The debugger design for Ajit processor is divided as two main sections, GDB Server Bridge and Hardware Server. The design of GDB Server Bridge is described on the first document on this topic[1]. This document focuses more on the design of Hardware server. As described in the previous document Hardware server resides along with the processor core which receives debug link commands(GDB server bridge to processor interface) and executes them on hardware. The commands for the debug link have been decided and detailed on first document[1]. Hardware server will always be communicating with processor core, halting the processor or changing it's state if necessary.

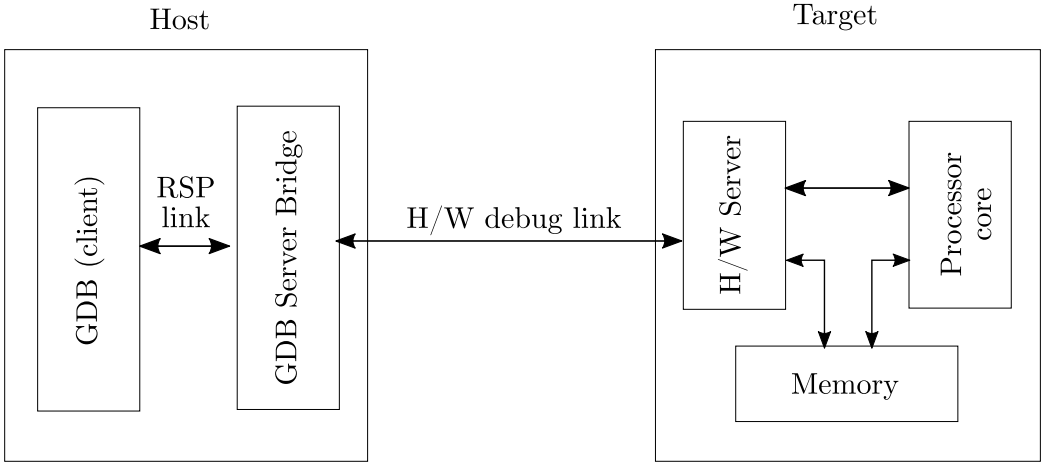

Figure 1: Overview of remote debugging interface for Ajit processor

2 Control flow

If debugging is enabled initially when the processor switches from reset mode to execute mode, it will inform that to hardware server and waits for continue signal back. Once hardware server establishes connection with gdb server bridge it will execute all the incoming commands (read/write registers, set/clear break-points etc). Then it will allow processor to run when it gets a continue command from gdb server bridge. In every processor cycle after it fetching the instruction, it will inform the state to the hardware server and wait for continue. If there was any interrupts, they are also conveyed along with the state. Now the hardware server checks for any break-point matches, if found it will inform the gdb server bridge. If there was no match the processor will be allowed to continue right away. After getting the continue signal the processor execution will happen normally and just before writing back results to memory, the address of the memories are send to the hardware server as a message to check any watch point occurrence, and processor waits till it gets a continue signal back.If there is a match with any watchpoints the hardware server will notify gdb server bridge. If there is no watch-point match it issues a continue signal right away. Once Processor gets a continue signal it continues executing as normal for the current cycle.

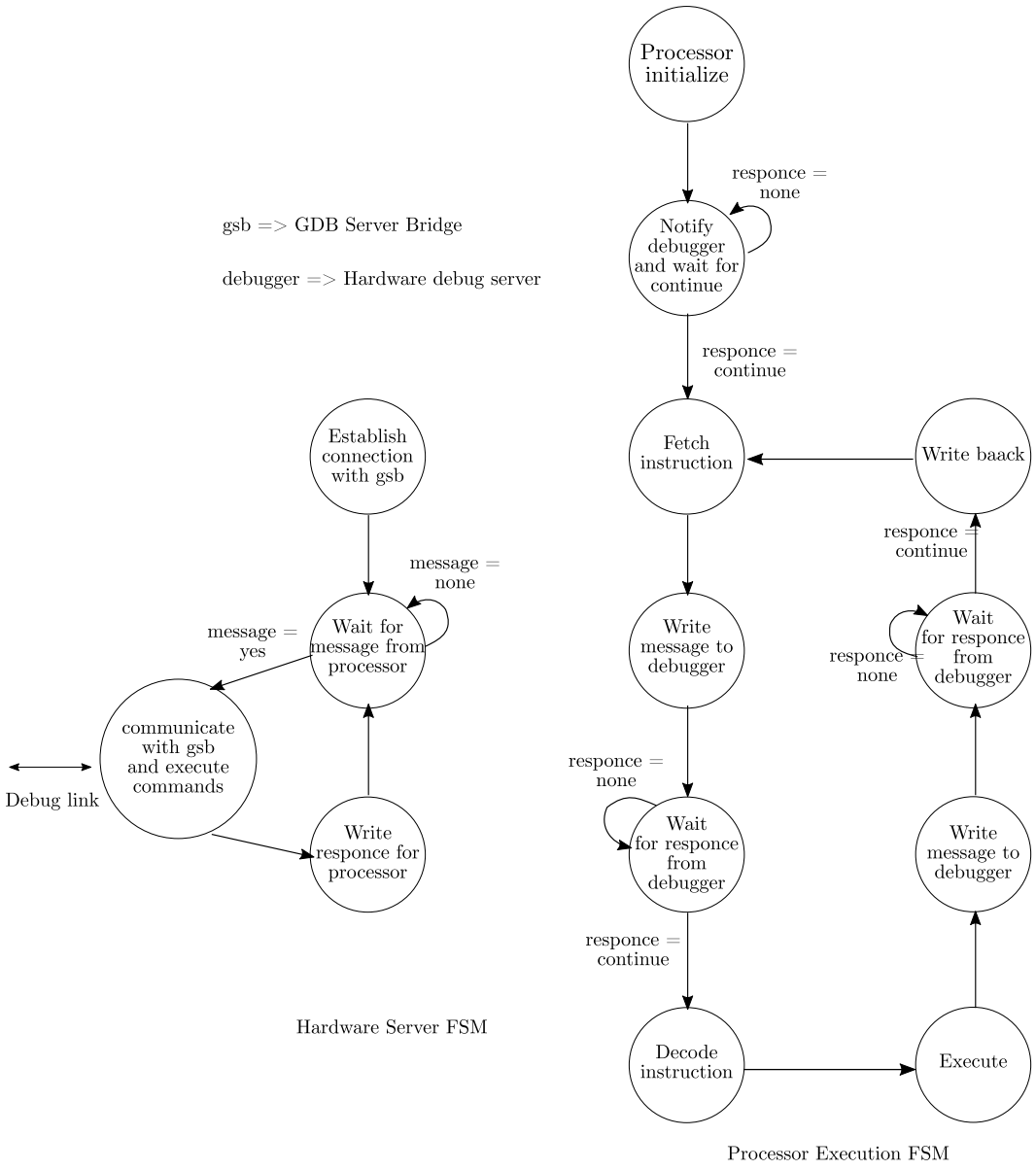

Figure 2: FSM shows execution of processor and hardware server

2.1 Processor execution flow

Notify debugger and wait for continue: Processor initialize event must be notified to gdb, so processor sends a message to hardware server and wait for response. After getting this message, gdb sets initial break-points and watch-points and sends a response back to processor saying continue.

Fetch instruction, Decode instruction, Execute, Write back: These all states of FSM describes the normal instruction execution flow of processor.

Write message to debugger: Just after fetching each instruction processor execution is paused and sends a message to processor which essentially consists of a pointer which indicates processor state, from which hardware server can retrieve PC, status register values, etc.

Wait for response from debugger: Processor keeps wait state till it gets a response (continue signal) from hardware server. Meantime hardware server will be communicating with gdb server-bridge and executing corresponding operations, as it finishes sends continue signal to processor.

2.2 Hardware server server execution flow

Establish connection with gsb: This is initialization of hardware debug server, as it started it will establish connection with gdb server bridge. It will remain on same state till it able to do so.

Wait for message from processor: Hardware server will wait for message from processor.

communicate with gsb and execute commands: This is the core part of hardware server. it retrieve PC and status register values, current fetched instruction etc. First the PC value is matched with break-point registers, if no matches found it checks for watch-point matches. if any break point or watch point is met, it will inform gdb server-bridge and execute subsequent commands till it gets a continue command. If there is no match with break-point then current instruction is command is issued right after.

Write response for processor: writes response (continue signal) to processor.

3 Implementation

The hardware server will be implemented as a single thread. It has three main functional blocks, one to handle the debug_link communications(debug_link side), second to handle the processor side communications(processor side) and an execution unit, which executes the debug_link commands. Necessary modifications are made in the processor core `sparcCore.c, execute.c` to enable communication with hardware server.

3.1 Modifications to Ajit C model

Processor and hardware server use Request acknowledgment mode communication through unidirectional pipes. When processor has to communicate something, it sends a corresponding id through `CPU_HW_id`(8 bit pipe) and the current processor state pointer through `CPU_to_HWSERVER_state`(64 bit pipe). After processor initialization it notifies hardware hardware server by sending an id "0" and and waits till it gets continue signal back. At each instruction execution cycle just after fetching instruction processor sends an id '1'(if the processor has been reset) or '2'(normal instruction fetch) to hardware server and waits till it gets a continue signal back on `HWSERVER_to_CPU` (32 bit pipe). While executing memory access instructions (memory read/memory write), just after calculating the memory address, processor sends an id '3' to hardware server and waits till it gets a continue back. When processor sends id ='3', it will also send calculated address through `CPU_to_HWSERVER_state`

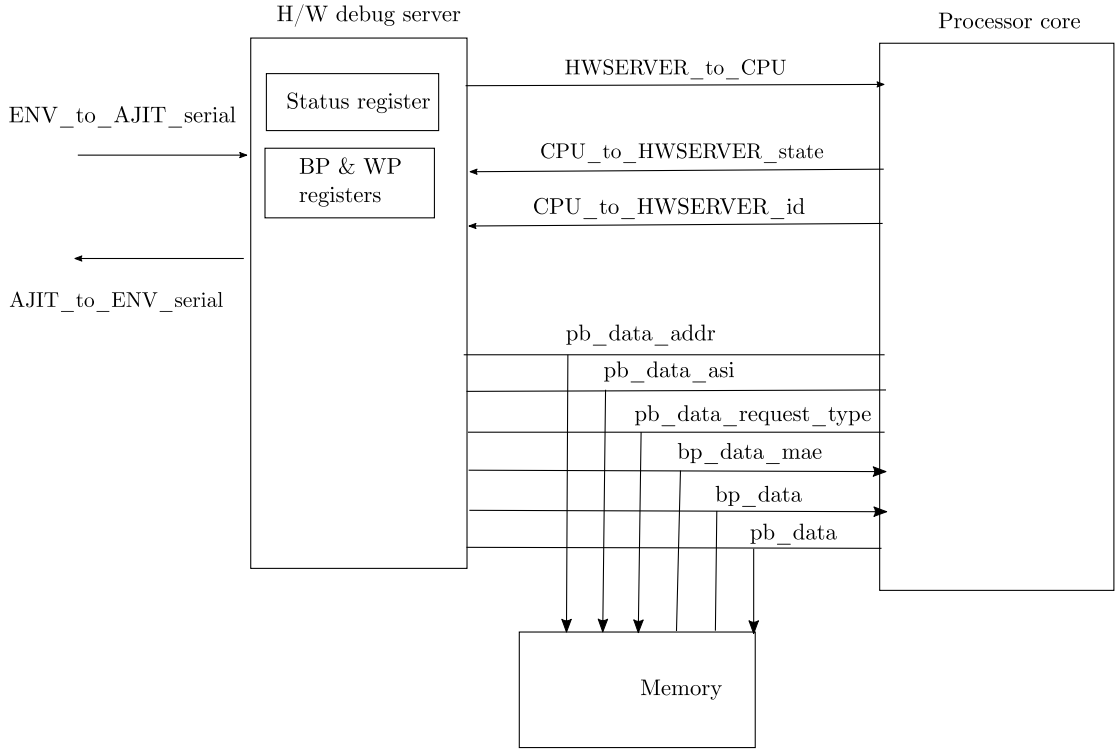

Figure 3: Hardware server connections

pipe. Finally when complete execution finishes(ta0) it sends id ='4'. The continue in all these cases is 0x434F4E54("CONT" in ASCII).

`pd_rsp_id` from processor to hardware server in each case

Processor initialization	[0]
Processor has been reset	[1]
Memory access:	[2]
When completes execution(ta0):	[3]

3.2 Hardware server implimentation

Hardware server is implemented as 4 functional blocks. Processor side, Debug_link side and Execution unit.

3.2.1 Processor side

This block handles all the communications with processor through the three pipes `HWSERVER_to_CPU`, `CPU_to_HWSERVER_state` and `CPU_to_HWSERVER_id`. It listens to `CPU_to_HWSERVER_id` pipe, and when gets id is "0' (which indicates processor is now initialized), the processor is not let to continue execute further. NOW hardware server sets break-points and watch-points requested by gdb server bridge, and when it is done lets the processor continue. If the id is '2', then the processor has halted after a normal instruction fetch, and the hardware server will check for break-point conditions. If conditions didn't met if conditions didn't met, a continue signal is given to processor right away. If the conditions are met the control will be transferred to debug_link interface block till it returns back. After getting control back it will send continue signal to processor through `HWSERVER_to_CPU`. If the `CPU_to_HWSERVER_id` is '3', it will check for match between watch-point registers and memory address

being accessed. If CPU_to_HWSERVER_id 2, then it indicates that there is an exception. The exceptions will be handled by passing them to the debug_link interface block.

For passing the current processor state will be communicated using a pointer to the structure ProcessorState, which also contains the status register values(a pointer to the structure StatusRegisters).

3.2.2 Debug_link side

GDB server bridge and hardware server communicates through two 32 bit unidirectional pipes AJIT_to_ENV_serial, ENV_to_AJIT_serial). This block receives all the debug_link commands(form gdb server bridge) through ENV_to_AJIT_serial, executes them and sends their responses back through AJIT_to_ENV_serial pipe. When ever the processor goes to halt mode this block will takes the control, and each time when it comes to control it will send details about why the processor halted to gdb server bridge. It consists of two functional units, Packet translate unit and Execution unit.

Packet translate unit

This unit handles all the communication between gdb sever bridge and hardware server. The block will always listen to the serial port expecting a start character. When the start character arrives, the buffer will be cleared and subsequent characters will be stored till receiving the end character, the buffer contents will be decoded and decoded commands passed to the execution unit. They are then passed on to the Execution unit. Similarly when the Execution unit needs to send data back to the gdb server bridge, it passes them on the Packet translate unit. Then packet translate unit will add start and stop characters to the message and will be sent through the serial line.

The basic functions required for implementing this unit will be,

`recv_spi()`
: Clear the buffer on receiving the start character, and then enter the upcoming characters in it. Calculate checksum of the packet on receiving the end character and check for error. Acknowledge host machine and return the content of the packet if it was received correctly, and return error code if it was not.

`send_spi()`
: Insert start and stop characters to the message, and then calculate and append the checksum to it. Finally , send the packet character by character through the debug serial link.

3.2.3 Execution unit

Execution unit will receive commands received from packet translate unit and will do the corresponding actions depending on the received commands.Execute unit have 16 break-point and 16 watch point registers along with a status register, which indicates the content of each register is valid or not. The possible commands are listed below

3.2.4 Validation unit

If the hardware-server is being used for validation, then this unit takes care of that process. When hardware-server gets id = '4' on CPU_to_HWSERVER_id, the unit runs the validation scripts. The validation function checks all the project files given, checks all the memory/register contents to finally generate a log file for each of them.

- *Read or write registers*
  This command is to read or write register contents. if it is a status register, the content can be retrieved by reading from pointer to the structure StatusRegisters, or if it is general purpose register we can use writeRegister(), writeFRegister(), readRegister(), readFRegister()

- *Read or write memory*
  Memory read or memory write commands will will read or write to memory by using same functions as used by processor.

- *Set/Clear brak-point or Set/Clear watch-point*
  The set break-point/watch-point will load the corresponding values to corresponding register bank if there is a free register is available(as shown by status register), if all the break-point/watch-point registers are full it will send an error message to gdb server bridge. Similarly the clear watch-point/break-point will force the corresponding entry on status register to zero.

References

[1] Prof. Madhav Desai Titto Thomas, Ashfaque Ahammed. *Design Of Debugger For AJIT Processor.* 2015.