GDB for OR1k

Author: Marko Mlinar
marko.mlinar@opencores.org
Damjan Lampret
Igor Mohor

Rev. 0.2
May 22, 2001

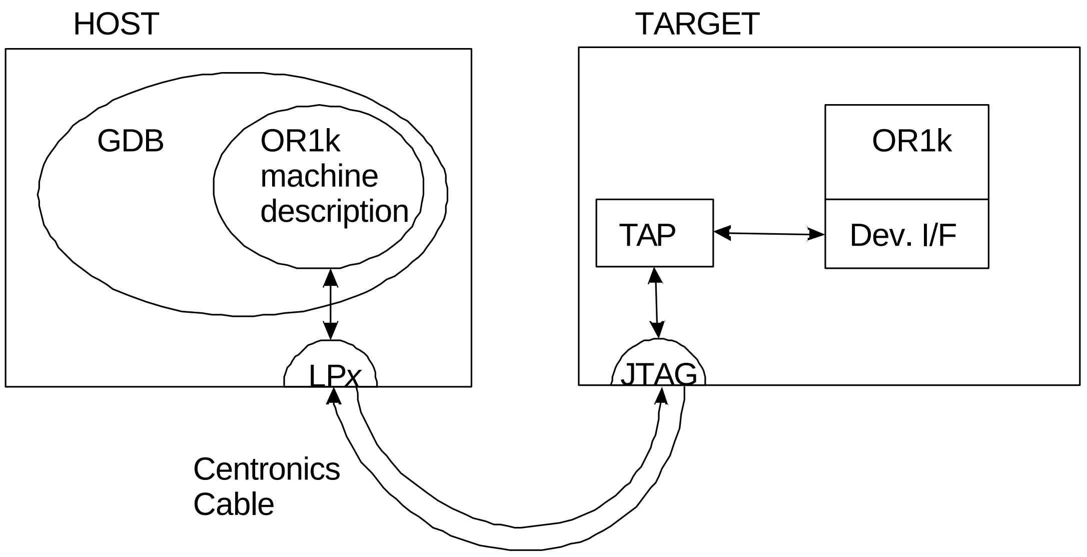
Revision History

Rev.	Date	Author	Description
0.1	27/4/01	Marko Mlinar	Initial document
0.2	22/5/01	MM	Added more descriptions to software operations

Contents

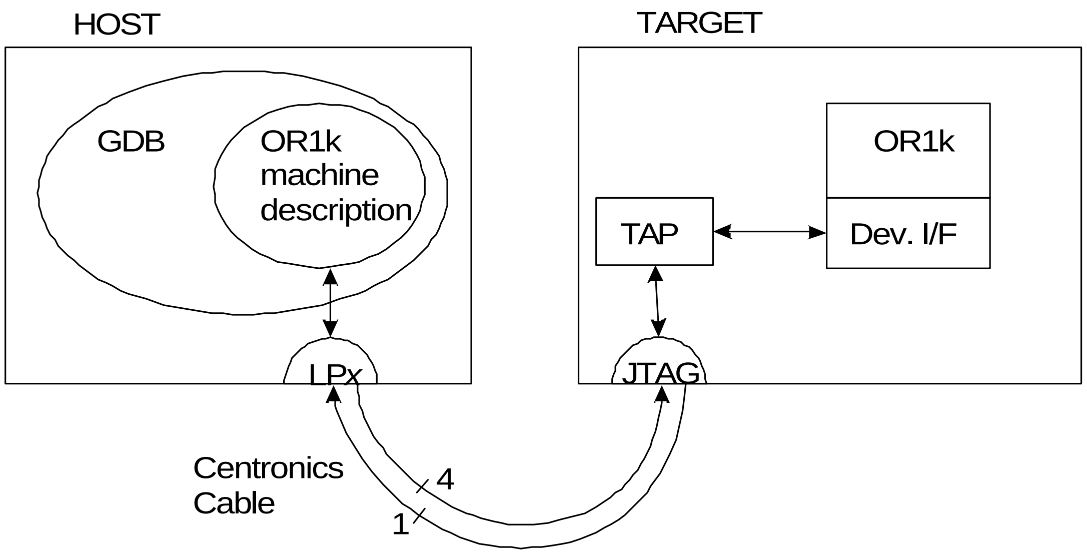
1 Introduction

This document describes communication protocol between GDB (GNU Debugger) and JTAG Test-Access-Port and lists supported capabilities in GDB.



2 JP1 Protocol

JP1 protocol is simple JTAG compatible protocol. It does not need any extra hardware.

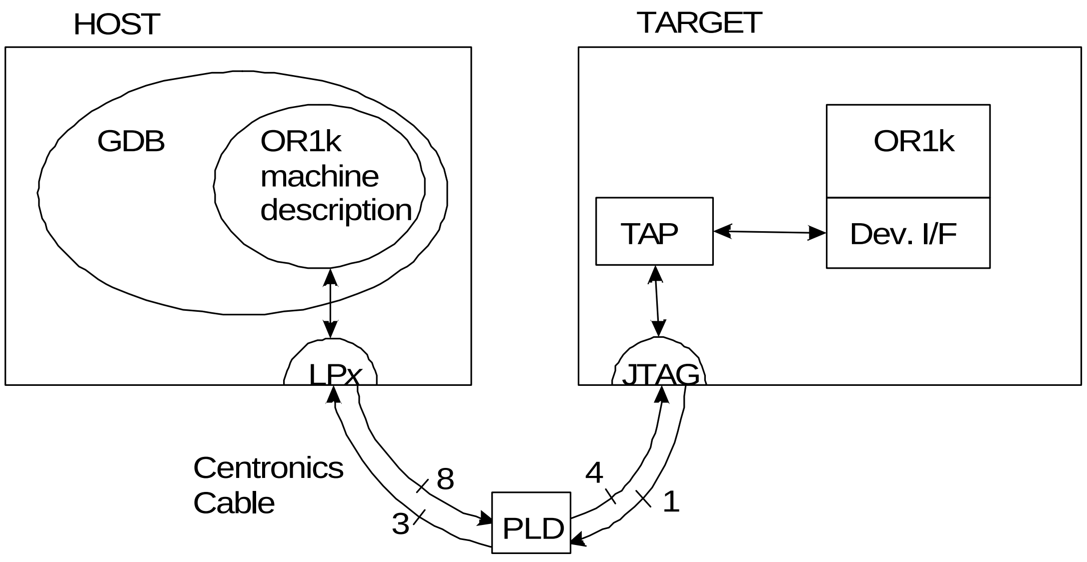


Each JTAG cycle requires 2 parallel port writes and one read (if necessary) from host. First one lowers the clock and sets the data (RSTn, TMS and TDI). Second write does not modify the data, but raises the clock. See JTAG specification for more info. Then one bit is read from CENTRONICS_BUSY signal, using IOCTL.

Port	Description	Width	Direction (relative to host)	Assigned centronics pin
TCLK	Clock	1	Output	D0
TRSTn	Reset	1	Output	D1
TMS	Mode Select	1	Output	D2
TDI	Data Input	1	Output	D3
TDO	Data Output	1	Input	BUSY

3 JP3 Protocol

Unlike JP1 it requires small amout of extra logic (e.g. PLD) on the board, but is six times faster.



This protocol does not directly change signals of JTAG port, but instead sends three pairs (TMS, TDI) and receives three TDO signals. CLK signal has different meaning: both clock positive and clock negative edge represents data valid. If bitstream length is not of modulo 3, then zeros are appended to TMS, data is x. This way JTAG stays in RUN_TEST/IDLE state.
Shortly, PLD circuit should translate JP3 protocol to JP1 for each data.

Port	Description	Width	Direction (relative to host)	Assigned centronics pin(s)
TCLK	Clock	1	Output	D0
TRSTn	Reset	1	Output	D1
TMS	Mode Select	3	Output	D2, D4, D6
TDI	Data Input	3	Output	D3, D5, D7
TDO	Data Output	3	Input	BUSY, PAPER_ERR, SELECT

4 Software Operation

This section deals with the software operation.

Communication example: Setting SW Breakpoint

It all starts when setting breakpoint in gdb prompt:
`(gdb) breakpoint 0x1234`
GDB then internally searches for target specific macros, like (INSERT_BREAKPOINT, TARGET_XCHG_MEMORY, BREAKPOINT_FROM_PC, …) to replace instruction at address `0x1234` with `l.brk`. Previous instruction is backed into host buffer. When or1k encounters `l.brk` instruction it halts. GDB meanwhile continuously polls processor status. Note that processor can be stopped using access to or1k registers.

GDB (remote) target uses JP1/3 protocol via parallel port driver (e.g. /dev/lp0) and JTAG I/F to access or1k registers, as specified in Bender Development Interface Document. For each 32b memory or register access we have to send 65 bits for data (and address), 8 bit CRC and some control bits (for JTAG purposes). See the RISC Development document for details. Using JP3 protocol we don't need to send extra dummy bits (one transfer requires exactly 24 parallel port writes, and for reading extra 11 reads).

4.1 Reset and Initialization of Remote Target

In order to debug the target, program has to be transferred to a stable environment. Since after the chip reset the processor is surely in stable and well defined state, it is naturally to stall processor right after the reset. Implementation specific processor info is then read, and program data is transferred. Right after that remote debugging can start.
More accurately - following steps are taken:

1. set processor reset
2. set processor stall
3. unset processor reset
4. read implementation specific registers and configure gdb (e.g. UPR)
5. set debug specific registers to idle state
6. transfer data (when user executes `load` command)
7. unstall processor

4.2 Communication with Target

It is not smart to do complex operations while processor runs, since we can enter unpredicted state. During such complex operation (program loading, setting breakpoints, etc.) processor is stalled. Smaller operations like register or memory read can be made during normal processor operation[1].

4.3 Hardware Supported Breakpoints and Watchpoints

Since debug unit has limited number of matchpoint resources, they should be used wisely. gdb default operation is first to set HW breakpoints and then SW ones. Hardware breakpoint can be set explicitly on e.g. some ROM location, using `hbreak` command. DVRx and DCRx pairs are programmed to set proper matchpoints. Normal breakpoints use only one matchpoint, while watchpoints at least two (e.g. data access watchpoint is set on memory address range, thus yielding conditional: addr >= 0x1000 && addr <= 0x1003). For each watchpoint chaining is set in DMR1 register to properly connect matchpoint conditionals. We always tend to use lower indexes first and sometimes mathcpoints must be reordered to find optimal fit.

4.4 Ending Communication

It is not necessary for gdb to do anything when ending remote session. However, sometimes processor is connected to viable equipment. If continuing program or unpredicted state is entered, damage can occur, thus processor stall is attempted[2].

[1] gdb user must be aware that he is using asynchronous operation.
[2] Note that it is not always possible for gdb to properly end communication, e.g. cable to the target is disconnected.

5 Supported Features

Warning: breakpoints, watchpoints and catchpoints have different definition in or1k than in GDB.

or1k feature	Description	Function in GDB	Comment
processor stop	immediately stops or1k	`^C`	fully supported
full register and memory access		`set`	fully supported
l.brk	software breakpoint	`breakpoint`	Fully supported. At the same time software conditional breakpoints are supported by GDB. First HW breakpoints are set.

matches, watchpoints, breakpoints	hardware breakpoint	`breakpoint,`
		`hbreak,`
		`watch`
			A lot of work to determine whether expression satisfies break criteria.
			Also HW breakpoints are set to positions where SW cannot be placed (e.g. flash).
			Limited conditional watchpoints.
			(like e.g. mips)

trace	trace	`trace`	We chose to declare or1k specific `trace` instruction: `hw_trace`, which matches or1k hardware trace. `trace` would be reserved for SW trace.
catchpoints	special events, that cause breakpoint		supported by GDB, but custom names has to be defined somehow

Appendix A

Parallel Port

Unavailable yet. Search for "parallel port programming" and "parallel port pins" on the Internet.