Minimal OpenRISC System on Chip

FAQ

Table of Contents

MinSoC

Adaptation

How to adapt the firmware to my implementation?

Answer:

1. edit minsoc/sw/support/orp.ld line 14 LENGTH = 0x00006E00 to
   a) your memory amount in Bytes $4 \cdot 2^{MEMORYADRWIDTH}$, where MEMORYADRWIDTH is defined in `define MEMORY_ADR_WIDTH in "minsoc/rtl/verilog/minsoc_defines.v"

      $4 \cdot 2^{MEMORYADRWIDTH}$ minus ORIGIN = 0x00001200

      (e.g. 4*2^13 = 32,768 Bytes = 0x8000 | LENGTH = 0x8000 – 0x1200 = 0x6E00)
2. select your STACK size on "minsoc/sw/support/board.h" line 16 #define STACK_SIZE 0x01000
   a) change your IN_CLK if not using 25000000 (25MHz)

How to configure the simulation

Answer:

1. configure your system: "minsoc/rtl/verilog/minsoc_defines.v"
   a) You can uncomment ETHERNET on "minsoc_defines.v" to input data to the SoC's Ethernet interface and read data from it.
2. configure "minsoc/bench/verilog/minsoc_bench_defines.v"
   a) Your testbench will use a memory model, not actually the same memory controller the implementation uses. This enables the option "`define INITIALIZE_MEMORY_MODEL", where the firmware is loaded to the memory before testbench start.
   b) You may use the actual implementation memory:
      - comment "`define INITIALIZE_MEMORY_MODEL"
      - edit "minsoc/sim/run/generate_bench"
        - substitute "../bin/minsoc_model.txt" for "../bin/minsoc_memory.txt"
      - You might want to uncomment "`define START_UP", it loads the firmware to a SPI memory. At start of testbench the system reads this memory and loads the firmware to main memory. Takes +-3 min. This is possible to be used for a real system, all you have to do is uncomment "`define START_UP" from "minsoc/rtl/verilog/minsoc_defines.v".
3. Modify testbench as you please.

   Please refer to "minsoc.pdf" chapter 4 Simulation for more information.

Is it possible to debug the simulation as I debug the firmware running on my board?

Answer: Yes

Open 3 terminals:

1. terminal 1: from minsoc/sim/run/
   a) ./generate_bench
   b) ./run_bench <your_firmware.hex>
      - ./run_bench ../../sw/uart/uart-nocache-twobyte-sizefirst.hex
2. terminal 2: from minsoc/sim/run
   a) ./start_server
3. terminal 3: at minsoc/sw/uart
   a) or32-elf-gdb uart-nocache.or32
   b) target remote :9999
   c) load
      - if you have INITIALIZE_MEMORY_MODEL enabled you don't have to do this
      - if you have START_UP and waited for the message: “Memory start-up completed...” you also don't need this
   d) set $pc=0x100
   e) c

My device is full, can I reduce the used logic of the SoC?

Answer: yes

1. configure minsoc/rtl/verilog/or1200/rtl/verilog/or1200_defines.v (recommended values for different devices under synthesis_examples.pdf)
   a) Target FPGA memories (OR1200_XILINX_RAMB16 for Xilinx, Spartan 3 and above, OR1200_ALTERA_LPM for all Altera)

      (if you do this, check: I have generate bench errors, what happened?)

   b) Type of register file RAM (OR1200_RFRAM_GENERIC, OR1200_RFRAM_TWOPORT or OR1200_RFRAM_DUALPORT) (dual port is supported by Xilinx BRAM and Altera)

      (select only one of the three)

      - if Altera: include `define OR1200_ALTERA_LPM_XXX (right under `define OR1200_ALTERA_LPM if you wish)

   c) comment `define OR1200_PM_IMPLEMENTED

d) If not using Linux you can:

- uncomment `define OR1200_NO_DC
- uncomment `define OR1200_NO_IC
- uncomment `define OR1200_NO_DMMU
- uncomment `define OR1200_NO_IMMU
- comment out `define OR1200_CFGR_IMPLEMENTED

e) If you don't need multiplication, mac operations or divisions

- comment out `define OR1200_MULT_IMPLEMENTED
- comment out `define OR1200_MAC_IMPLEMENTED
- comment out `define OR1200_DIV_IMPLEMENTED

(If you do this, change sw/support/Makefile.inc line 7: GCC_OPT=-mhard-mul -g -nostdlib to GCC_OPT=-msoft-mul -msoft-div -g -nostdlib)

Problems

I have generate bench errors, what happened?

Answer:

```
foo@ubuntu:~/minsoc/sim/run$ ./generate_bench
../../bench/verilog/minsoc_bench.v:590: error: Could not find variable
``minsoc_top_0.or1200_top.or1200_cpu.or1200_rf.rf_a.ramb16_s36_s36.mem" in
``minsoc_bench.init_fpga_memory"
../../bench/verilog/minsoc_bench.v:591: error: Could not find variable
``minsoc_top_0.or1200_top.or1200_cpu.or1200_rf.rf_b.ramb16_s36_s36.mem" in
``minsoc_bench.init_fpga_memory"
2 error(s) during elaboration.
foo@ubuntu:~/minsoc/sim/run$
```

You tried to use the Xilinx RAM or your specific memory, by uncommenting the 'define OR1200_XILINX_RAMB16 or others in the minsoc/rtl/verilog/or1200/rtl/verilog/or1200_defines.v file.

On or1200_r3 the register file, or1200_rf.v, always instantiates a generic memory for DUAL PORT RAM. Previously it instantiated a target specific or generic memory depending on your sets of or1200_defines.v.
Since the CPU does not work if the registers aren't set to zero previous to simulation start, my testbench especifically set the memory content of the registers to zero, before simulation start. I didn't try to find out why this is like that, I only noticed it was that way.
First, I commented out the initialization for dual port RAM to test if the new memory would work. The simulation failed as it did before.
I could include a new initialization for the new memory. Because the new memory is generic and I believe target specific memory should be used whenever possible, I'd recommend you to switch it back to the way it was before and not to touch the memory initializations:

On or1200_rf.v: lines 304 and 280 edit this way:

line 304:

```
or1200_dpram_32x32
rf_b
(
.rst_a(rst),
.rst_b(rst),
.oe_a(1'b1),
// Port A
.clk_a(clk),
.ce_a(rf_enb),
...
```

line 280:

```
/* or1200_dpram #
(
.aw(5),
.dw(32)
)*/
or1200_dpram_32x32
rf_a
(
.rst_a(rst),
.rst_b(rst),
.oe_a(1'b1),
// Port A
.clk_a(clk),
.ce_a(rf_ena),
...
```

This might be a typo or maybe a work around for something else. That's something the OpenRISC developers have to tell us. It is remarkable that both, OR1200_RFRAM_GENERIC and OR1200_TWOPORT are exactly the way there were before. However, the OR1200_RFRAM_DUALPORT uses now a new module, which is basically an adaptation of the OR1200_RFRAM_GENERIC one, but with DUALPORT, of course.

MinSoC firmwares won't compile.

Answer: there are two versions of GCC right now. The older version adds a leading underscore to C functions in its assembly counterpart and the other does not. MinSoC firmware includes assembly files which access C functions using the leading underscore being compatible with the older version. If you have the new version GCC, remove the underscores of C functions being accessed by the assembly. The error output of GCC will tell you which are the functions whose underscore you have to remove.

Tweaks

I want my design to automatically initialize my firmware on power-up, how do I do that?

Answer: Nowadays there are 3 options to do that:

1) MinSoC comprehends the standard of them which is based on the project OR1k Start-up from OpenCores. At system power-up the CPU's program counter is jumped to the OR1k Start-up module by a Start-up Starter module (inside of minsoc_top.v). The OR1k Start-up contains a hardwired code which reads the data from a SPI memory and copy it to main memory. The amount of data copied is decided upon reading of the first 32 bits of the SPI memory, which must contain the firmware's size. After the code has been completely copied, it restarts the CPU.

   To use this, all you have to do is uncomment "`define START_UP" from "minsoc_defines.v". Don't forget to load the hex file created by the project's Makefiles into the SPI memory, so actually your firmware is copied to the main memory. The hex file is created from the or32 executable by the project's existing Makefiles.

   Everytime the firmware is updated it has to be uploaded to the SPI memory.

   You can read about it in more detail in the minsoc.pdf file chapter 2.3 Start-Up Modules.

   Advantages: works with any memory type. The speed will be the speed needed by the memory programmer to program the external SPI memory. After that you press the reset button and the firmware will be loaded from the SPI memory by the system. It will then be run.
   Drawback: no debugging, somewhat bigger SoC, required onboard SPI flash.

2) A Xilinx approach has been implemented by Ravi Kumar. A similar Altera approach has been implemented by "sotusotu". This approach creates a memory initialization file off the firmware executable file, which is then linked to "minsoc_onchip_ram_top.v".

   Everytime the firmware is changed the synthesizer and P&R must be run for the SoC.

   Link Xilinx: http://opencores.org/forum,OpenRISC,0,3628,0 post 9.

   Link Altera: http://opencores.org/forum,OpenRISC,0,3749 post 5.

   Advantages: once you have a target firmware, it will be on RAM right after FPGA configuration.
   Drawbacks: you have to re-synthesize, re-place&Route to create the bitfile with the firmware, no debugging, only works with onchip-RAM.

3) The XSOC project has yet another approach to pre-initialize the memory with a firmware. They have created a Perl script, that update the SoC resulting bitfile substituting the memory initialization with the target firmware.

   The difference to approach 2 is that approach 2 requires new run of the synthesizer & P&R in order to exchange the firmware, while this directly updates the bitfile result of that very process. After that only a FPGA configuration is necessary to have the new firmware.

   Link: http://pm.stu.cn.ua/wiki/10/Getting_started "Updating bitstream with new software"

   Advantages: high speed,on RAM right after FPGA configuration.
   Drawbacks: no debugging, only works with onchip-RAM.

I want to compile my firmware independent of MinSoC library but it does not work, what am I doing wrong?

Short example: user “ag1986” says: “test.c

```
#include stdio.h
int main()
{int a,b,c;
a=5;b=10;
c=a+b;
}
```

binary generation: “or32-elf-gcc test.c -o test.or32”

simulating using “or1k-sim: or32-elf-sim -f sim.cfg --enable-profile test.or32” does not work.”

Answer: You need a "bootloader" in order to make your software work.

I will try to clearly explain its task in the following lines by comparing it to you running a program on your own operating system.

When you type in the "linux shell" "ls" or "dir" in "windows command prompt", the operating system creates a new process for this program; set a runnable environment for it, with stack, registers and memory; and then at some point set the program counter to the start of your program. (http://en.wikipedia.org/wiki/Process_fork).

The key points here are, you are creating a binary which:
-does not have any information about position in memory
-you are not initializing a stack
-you are not initializing registers
-you are not saying how much memory is available

And without these, your software will simply not work. For instance, if your binary will be located starting at address 0x00, your software will not be ever executed from the beginning by the OpenRISC since it resets to 0x100 and go then onwards until memory finishes.

So, how do I solve this? Generally you cannot directly set registers in C, so you actually need sort of a C assembly mix and then link them together. Furthermore a linker script is used to create memory spaces so you can actually organize the memory dividing data from instruction, exceptions from code and so on.

In the link below you find a small Makefile project with a bootloader and explanation to it in the 5th post.

Link: http://opencores.org/forum,OpenRISC,0,3598

Note: you find more about this in “minsoc.pdf” chapter 3.1 Bootstrapping: a tiny bootloader.

How to include a 8 bit module to the SoC?

Answer: Modules are connected to the Wishbone 32 bit interconnect, to which all SoC modules are connected and can thus communicate. If you want to connect an 8 bit module to the 32 bit interconnect

you need a bridge, which resolves the non-aligned memory accesses to the right address and forwards the right Byte data to the 8 bits on the other end. To do so, you need to connect the module “minsoc_wb_32_8_bridge” to the interconnect and the 8 bit module on the other end. The module “minsoc_wb_32_8_bridge” can be found under “minsoc/utils/”.
Once the hardware is ready, you have to take special care while programing your firmware not to issue 32 bit accesses to an 8 bit module. That can be easily done by using the right macros to do so:

```
#define REG8(add) *((volatile unsigned char *)(add))
#define REG16(add) *((volatile unsigned short *)(add))
#define REG32(add) *((volatile unsigned long *)(add))
```

These macros are found in the provided support library (“minsoc/sw/support/support.h”)

To access 8 bit modules, use only **REG8**. That assures that the instruction l.sb is used and not l.sw. The former stores a byte, while the latter a word.

For more information about how to include new modules to the system, please refer to “doc/minsoc.pdf” “2.2.2 Attaching Modules to the System”.

Related discussion: http://opencores.org/forum,OpenRISC,0,3552

Advanced Debug System

Problems

I have problems compiling the adv_jtag_bridge, what is going on?

Answer: If you set in Makefile SUPPORT_PARALLEL_CABLES to true, you will need libioperm under Cygwin, for Linux this option is standard.

Under Cygwin: run Cygwin setup.exe and select “libioperm”

If you set in Makefile SUPPORT_USB_CABLES and SUPPORT_FTDI_CABLES to true, you need libusb and libftdi:

Under Linux: install libusb (http://www.libusb.org/) and libftdi

(http://www.intra2net.com/en/developer/libftdi/)

Under Debian based distributions: sudo apt-get install libusb-dev libftdi-dev

Under Cygwin: run Cygwin setup.exe and select “libusb-win32”

-install libftdi from: http://www.intra2net.com/en/developer/libftdi/

Adv_jtag_bridge does not connect to my cable, why?

Example 1: 'Failed to find USB-Blaster'

Answer: Nathan Yawn: “It should be possible. The error you are seeing basically means that libUSB enumerated all USB devices, but did not find one with the manufacturer / device ID combo it was looking for (09FB:6001). What are using for your development system?

If it's windows/cygwin, make sure you have a libUSB filter installed for the USBBlaster, then run the

libUSB test program and see what devices it shows. If it shows a USBBlaster with a different ID, then you can change the ID in cable_usbblaster.c to match. If it shows nothing, then you need to set up libUSB correctly. If you don't know what it shows, send me the output and I'll take a look.

If you're using Linux, then I think "lsusb" will show all the USB devices in the system. Again, look for a USBBlaster and see if it's ID matches, and if you can't tell what's there, send me the output."

Related discussion: http://opencores.org/forum,OpenRISC,0,3954

Example 2: sudo adv_jtag_bridge xpc_usb

```
Enumerating JTAG chain...
WARNING: maximum supported devices on JTAG chain (1024) exceeded.

Devices on JTAG chain:
Index Name ID Code IR Length
-----------------------------------------------------------------
0: (unknown) 0xFFFFFFFF -1
1: (unknown) 0xFFFFFFFF -1
2: (unknown) 0xFFFFFFFF -1
3: (unknown) 0xFFFFFFFF -1
.

1022: (unknown) 0xFFFFFFFF -1
1023: (unknown) 0xFFFFFFFF -1

Target device 0, JTAG ID = 0xffffffff
ERROR! Unable to autoprobe IR length for device index 0;  Must set IR size on command line. Aborting.
```

Answer a: (xpc_usb/Xilinx DLC9 cable): Nathan Yawn: **"**The Xilinx Parallel Cable DLC9 is not publicly documented. Downloading a bitstream to the FPGA puts the DLC9 into a mode from which we cannot recover. The cable must be unplugged from the workstation an re-attached before it can be used with adv_jtag_bridge. (In some cases, unlocking the cable in Impact has been reported to work.)"

Related discussion: http://opencores.org/forum,OpenRISC,0,3624

Difficulty: Many current boards are being delivered with the DLC9 cable already on-board. That means that only a regular USB cable is connected between computer and board, without the old-style DLC9 cable-box in between. However, that makes it impossible to reset the cable without powering off the entire board, losing the FPGA configuration on the process. If unlocking the cable in Impact does not work, there is no direct way for adv_jtag_bridge to use the connection.

To circumvent that, you have two options:

Option 1: You can write the bitfile to an available on-board memory, from which the FPGA can self-configure on power-on. This way the FPGA is already configured on power-on and the cable is unlocked, enabling the use of the adv_jtag_bridge.

Note: Be certain that the FPGA is configured with MinSoC. You can use the adv_jtag_bridge self-test for that. "adv_jtag_bridge -t <cable>". The first test checks memory accesses and the second the CPU. However, the 3rd release of OpenRISC is failing the CPU test. So you have to skip the test, when really using the adv_jtag_bridge.

Option 2: For this solution, a real JTAG cable is necessary. Furthermore, the generic JTAG tap has to be used instead of the one available on the FPGA. That is done by uncommenting "`define GENERIC_TAP" and commenting out "`define FPGA_TAP". That also requires free FPGA pins available as board pins, that will be connected to the JTAG cable. The FPGA pins have to be bound to

the SoC by the definition of a constraint file (e.g. ucf file under Xilinx).

Example:

“minsoc/rtl/verilog/minsoc_defines.v”:

```
//
// TAP selection
//
`define GENERIC_TAP
//`define FPGA_TAP
```

“minsoc/backend/spartan3a_dsp_kit.ucf”:

```
############################
##
## JTAG
##
#net "jtag_tms" loc = "aa23";                                  #SAM D0
#net "jtag_tdi" loc = "u20";                                   #SAM D2
#net "jtag_tdo" loc = "aa25";                                  #SAM D4
#net "jtag_tck" loc = "u18" | CLOCK_DEDICATED_ROUTE = FALSE;   #SAM D6
#net "jtag_gnd" loc = "y23";                                   #SAM D8
#net "jtag_vref" loc = "t20";                                  #SAM D10
############################
```

Answer b: (all other cables) This implies that you are using the wrong cable or wrong cable support, either you are selecting the wrong cable (xpc3, xess, usbblaster, xpc_usb, ft2232), or you are using the wrong implementation for your cable. Wrong selection of implementation occurs especially for Altera USBBlasters. They have two different implementations, a libftdi based and a libusb based.

You can solve the wrong cable selection by selecting the right cable.

The cable with wrong implementation can be solved by changing adv_jtag_bridge Makefile:

SUPPORT_FTDI_CABLES=true

USE_ALT_FTDI_USBBLASTER_DRIVER=true

Recompile and then try again.

Note: Further information about cables can be found under “rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge/doc/adv_jtag_bridge.pdf”

Adv_jtag_bridge does not enumerate my device, why?

Example: adv_jtag_bridge xpc_usb
Found Xilinx Platform Cable USB (DLC9)
Found Xilinx Platform Cable USB (DLC9)
firmware version = 0x0404 (1028)
cable CPLD version = 0x0012 (18)
Enumerating JTAG chain...

```
Devices on JTAG chain:
Index   Name          ID Code        IR Length
-----------------------------------------------------------------
0:     (unknown)      0x06E5E093     -1
1:     (unknown)      0xF5046093     -1
2:     XC3S500E_FG320 0x41C22093     6

Target device 0, JTAG ID = 0x06e5e093

ERROR! Unable to autoprobe IR length for device index 0;  Must set IR
size on command line. Aborting.
```

Answer: What occurs here is that the adv_jtag_bridge does not know which protocol the target chip uses. You have two options:

1. That is generally solved by copying the bsdl files into your home directory. In this special case the FPGA chip is the third in the chain, in those cases the bsdl file for the two previous chips are also required for the automatic enumeration. So copying the respective bsdl files for the two remaining chips solves the issue.

   Note: Please, use the -b <directory> parameter of adv_jtag_bridge to indicate the directory where you put the bsdl files. The home directory should be automatically recognized, but I have seen cases where it was not.

2. You input 3 configuration parameters besides of the cable type instead of relying on the bsdl files, for instance “adv_jtag_bridge -x2 -l 2:6 -c 0x02 xpc_usb”. The parameter 'x' informs the device index in the chain, 2 in this example. Then you need to find out (through bsdl file for instance) what is the USER1 command of your chip, xc3s500e_fg320.bsd:612:

   “USER1” (000010),”

   The parameter 'l' informs first again the chip index '2' and then the bit width of the chain commands, in this case '6'. Finally, -c informs the USER1 command we just picked out from the bsdl file, it is given in binary format so “2'b10” corresponds to 2 or 0x02 for 6 bit width.

Adv_jtag_bridge self test fails?

Example 1: The jtag chain enumerates correctly, the SRAM test completes successfully, but the CPU test fails with the following results:

```
Testing CPU0 (or1k) - writing instructions
Setting up CPU0
Starting CPU0!
Read npc = 00000010 ppc = 00000028 r1 = 00000005
Expected npc = 00000010 ppc = 00000028 r1 = 00000005
Read npc = 00000028 ppc = 00000028 r1 = 00000008
Expected npc = 00000010 ppc = 00000028 r1 = 00000008
Read npc = 00000024 ppc = 00000020 r1 = 0000000b
Expected npc = 00000028 ppc = 00000024 r1 = 0000000b
Read npc = 00000020 ppc = 00000020 r1 = 00000018
Expected npc = 00000024 ppc = 00000020 r1 = 00000018
Read npc = 0000001c ppc = 00000018 r1 = 00000031
Expected npc = 00000020 ppc = 0000001c r1 = 00000031
Read npc = 00000020 ppc = 0000001c r1 = 00000032
```

```
Expected npc = 00000024 ppc = 00000020 r1 = 00000032
Read npc = 00000010 ppc = 00000028 r1 = 00000063
Expected npc = 00000014 ppc = 00000010 r1 = 00000063
Read npc = 00000010 ppc = 00000028 r1 = 00000063
Expected npc = 00000028 ppc = 00000024 r1 = 00000065
Read npc = 00000010 ppc = 00000028 r1 = 00000094
Expected npc = 00000010 ppc = 00000028 r1 = 000000c9
result = deaddef0
Self-test FAILED *** Bailing out!
```

Answer: I have the same problem here, do not start adv_jtag_bridge with the "-t" option.

Nathan Yawn: "Yes, there appears to be some incompatibility with the OR1200v3 and the self-test that the Advanced Debug System uses. Since the self-test hasn't changed any, there are two possibilities:

1. The self-test, which was written many years ago for jp2/OR1200v1, depends on some sort of non-spec behavior of the OR1200, which has been 'fixed' in OR1200v3.

2. The OR1200v3 is broken.

I haven't taken the time to figure out which it is, and at this point I'm not sure I care to. I'll try again once the new OR1200 has stabilized.

Warning: even if you disable the self-test, you'll still have problems with ADS and OR1200v3. When I've tried it, the system (repeatably) stops responding the third time I do a stepi. YMMV."

Related discussion: http://opencores.org/forum,OpenRISC,0,3861

Example 2:
```
[root@localhost run]# ./start_server
Enumerating JTAG chain...

Devices on JTAG chain:
Index   Name            ID Code         IR Length
-----------------------------------------------------------------
0:      (unknown)       0x149511C3      -1

Target device 0, JTAG ID = 0x149511c3
Using command-line debug command 0x8
*** Doing self-test ***
Stall or1k - CPU(s) stalled.
SRAM test:
expected 11112222, read 3211112222
expected 33334444, read 3233334444
expected 55556666, read 3255556666
expected 77778888, read 3277778888
expected 9999aaaa, read 329999aaaa
expected bbbbcccc, read 32bbbbcccc
expected ddddeeee, read 32ddddeeee
expected ffff0000, read 32ffff0000
expected dedababa, read 32dedababa
SRAM test failed!!!
Testing CPU0 (or1k) - writing instructions
Setting up CPU0
Starting CPU0!
Read            npc = 3200000010 ppc = 00000028 r1 = 00000005
Expected  npc = 00000010 ppc = 00000028 r1 = 00000005
```

```
Read          npc = 3200000010 ppc = 00000028 r1 = 00000008
Expected  npc = 00000010 ppc = 00000028 r1 = 00000008
Read          npc = 3200000028 ppc = 00000024 r1 = 0000000b
Expected  npc = 00000028 ppc = 00000024 r1 = 0000000b
Read          npc = 3200000024 ppc = 00000020 r1 = 00000018
Expected  npc = 00000024 ppc = 00000020 r1 = 00000018
Read          npc = 3200000020 ppc = 0000001c r1 = 00000031
Expected  npc = 00000020 ppc = 0000001c r1 = 00000031
Read          npc = 3200000024 ppc = 00000020 r1 = 00000032
Expected  npc = 00000024 ppc = 00000020 r1 = 00000032
Read          npc = 3200000014 ppc = 00000010 r1 = 00000063
Expected  npc = 00000014 ppc = 00000010 r1 = 00000063
Read          npc = 3200000028 ppc = 00000024 r1 = 00000065
Expected  npc = 00000028 ppc = 00000024 r1 = 00000065
Read          npc = 3200000010 ppc = 00000028 r1 = 000000c9
Expected  npc = 00000010 ppc = 00000028 r1 = 000000c9
result = 1c2deaddead
Self-test FAILED *** Bailing out!
```

Answer: Edit the start_server file, remove the -t option. This will skip the tests, hopefully everything else will work out fine. If you take a look the error is about having longer results than expected: expected 11112222, read 3211112222

I suppose your system is 64 bits. That's an issue with the adv_jtag_bridge as far as I can tell. I already reported it to Nathan Yawn. That's the creator of the Advanced Debug System (adv_jtag_bridge). For now it hasn't been corrected.

Otherwise, if you use a 32 bit system, it will work flawlessly.

GDB reports "Value being assigned to is no longer active.", what happened?

Example: (gdb) set $pc=0x100

Value being assigned to is no longer active.

Answer: The adv_jtag_bridge connect the debug system to gdb, the GNU debugger. But the version 6.8 of gdb has some issues, which have to be corrected before use. To do so, the adv_jtag_bridge software includes a patch for it. Save the patch to the gdb source code directory installed by the toolchain installation script and patch it.

If you installed a binary version of the OR1000 GNU toolchain and see this, you will have to install it using the installation script. The installation script downloads the sources and compiles the software. After the script installation is done, you have to patch the gdb source and recompile it.

- cp minsoc/rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge/gdb-6.8-bz436037-reg-no-longer-active.patch toolchain_build_directory/gdb-6.8
- cd toolchain_build_directory/gdb-6.8
- patch -p1 < gdb-6.8-bz436037-reg-no-longer-active.patch
- make
- sudo make install

Nathan Yawn: **"**GDB 6.8 has a bug which prevents it from working when no stack frame is present

(such as at start-up on a bare-metal debugger, such as this one). A simple patch applied to GDB will work around the problem (a general solution is not yet available). This patch can be found in the Patches/GDB6.8/ directory."

Adv_jtag_bridge "Ignoring packet error, continuing... " problem:

Answer: Nathan Yawn: "Ignoring packet error, continuing" means that GDB is timing out waiting for a response from a packet. In adv_jtag_bridge, rsp-server.c, set GDB_BUF_MAX back to it's original value of ((NUM_REGS) * 8 + 1). Smaller packets, done faster, GDB won't have to wait as long. Then, be patient. Things happen very slowly in simulation. Minutes, not seconds."

Note: same is true for xpc_usb and ftdi usbblaster, it seems.

Static solution:

Nathan Yawn: "In adv_jtag_bridge, open rsp-server.c in a text editor and change the GDB_BUF_MAX back to its earlier value of ((NUM_REGS) * 8 + 1)."

Furthermore, ignore the "Ignoring packet error, continuing...", it might take time to load depending on the cable.

Dynamic solution:

AhmedHMSoliman: "Use the following command before any other command in gdb to set the remote timeout at the first place.

"set remotetimeout 10"

and you can make sure time out was set to 10 seconds by using the following gdb command:

"show remotetimeout"

after that connect to remote target in the ordinary fashion:

"target remote :9999"'

Furthermore, ignore the "Ignoring packet error, continuing...", it might take time to load depending on the cable.

Related discussions:

-Static: http://opencores.org/forum,OpenRISC,0,3718

-Dynamic: http://opencores.org/forum,OpenRISC,0,3787

Cannot step through instructions after breakpoints, what to do?

Answer: Nathan Yawn: "This is a bug in the current versions of the OR1200 CPU (v3). The current version (SVN rev. 388) has a bug related to the hardware single-step function. The maintainers have improved the behavior some, but it's still a problem.

I recommend you checkout an older tag of the OR1200, and use that. Unless there's something in the OR1200v3 you really need, the OR1200v1 will work just fine, and it doesn't have the single-step bug."

Related discussion: http://opencores.org/forum,OpenRISC,0,3948

Tweaks

I'm running adv_jtag_bridge under Linux. How do I use adv_jtag_bridge with xpc3 or xess cables in non-privileged mode?

Answer: The adv_jtag_bridge accesses the parallel port physical memory directly. The advantage of that code is that it is portable under Linux and Cygwin equally. However, the code cannot be run by a non-privileged user on Linux. In order to avoid this privilege requirement, you have to replace the "cable_parallel.c" file from adv_jtag_bridge with the one provided by MinSoC under "minsoc/utils".

- cp minsoc/utils/cable_parallel.c minsoc/rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge/cable_parallel.c
- cd minsod/rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge
- make clean
- make
- sudo make install

Now you can run adv_jtag_bridge for xpc3 and xess without issuing the sudo command or being root.