Minimal OpenRISC System on Chip

How To

1 Compile Software

1. inside of sw/utils make clean, make all
2. inside of sw/support make clean, make all
3. inside of sw/drivers make clean, make all
4. inside of the target software (e.g. sw/uart) make clean, make all

2 Simulation

1. configure minsoc/rtl/verilog/adv_debug_sys/Hardware/adv_dbg_if/rtl/verilog/adbg_defines.v
   a) comment out, "`define DBG_JSP_SUPPORTED"
2. command to start testbench and select firmware
   a) from minsoc/sim/run/
      - ./generate_bench
      - ./run_bench <your_firmware.hex>
        - ./run_bench ../../sw/uart/uart-nocache-twobyte-sizefirst.hex

3 Synthesis

1. configure minsoc/rtl/verilog/minsoc_defines.v (recommended values for different devices under synthesis_examples.pdf)
   a) Select your FPGA device by uncommenting its manufacturer and commenting all other manufacturers. Select then your FPGA model by uncommenting it in case you have a Xilinx FPGA, for Altera comment all out.
   b) `define MEMORY_ADR_WIDTH 13 defines the amount of memory you get. The depth is defined by $2^{MEMORYADRWIDTH}$, since its data width is 32 bits, the amount in Bytes is 4 times its depth. (this is not allowed to be less than 12, 11 is the memory block address width)

   (if you change from 13, check FAQ->How to adapt the firmware to my implementation?)

   c) choose a clock division for your global clock related to your design max speed by changing the definition: "`define CLOCK_DIVISOR 5". If you have an Altera device please use only even numbers for the division, odd numbers are going to be rounded down.

   (if your resulting clock is not 25MHz, check FAQ->How to adapt the firmware to my implementation?)

   d) Define your RESET polarity: uncomment "`define POSITIVE_RESET" for an active high reset or "`define NEGATIVE_RESET" for an active low reset and comment the other.

2. configure minsoc/rtl/verilog/adv_debug_sys/Hardware/adv_dbg_if/rtl/verilog/adbg_defines.v
   a) comment out, "`define DBG_JSP_SUPPORTED"
3. define user constrains for system pinout (edit/create minsoc/backend/yourboard.ucf file)
4. create project in project manager (ISE, Quartus), include all Verilog files under "minsoc/rtl/verilog" and subdirectories
5. include the user constraint for system pinout (created on step 3) to your project on the project manager
6. synthesize, P&R and upload bitfile

4 Software Upload and Debugging

Upload software and debug for simulation and implementation using GDB

1. connect the cable to the selected JTAG TAP
2. start adv_jtag_bridge
   a) cd ~/
   b) adv_jtag_bridge <xpc3 | xess | usbblaster | xpc_usb | ft2232>
      - Note: if you run it under Linux and use cables xpc3 or xess, you have to run as superuser. You can bypass that, check out how on FAQ under, "I'm running adv_jtag_bridge under Linux. How do I use adv_jtag_bridge with xpc3 or xess cables in non-privileged mode?"
   c) Let the program running and open another terminal
3. Open a terminal program (e.g. gtkterm)
   a) configure port to a serial port connected to your board
   b) configure bitrate to 115200
4. start gdb, load firmware (example)
   a) cd minsoc/sw/uart
   b) or32-elf-gdb uart-nocache.or32
   c) target remote :9999
   d) load
   e) set $pc=0x100
   f) c
5. Inside of gtkterm "Hello World." should have appeared, if you press any key inside of gtkterm the processor will return the next alphabetical letter (press a, it returns b)