

Minimal OpenRISC System on Chip

How To

1 Compile Software

1. inside of sw/support make clean, make all
2. inside of sw/utills make clean, make all
3. 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 line 67, “`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 $2^{MEMORY_ADR_WIDTH}$, 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 line 67, “define DBG_JSP_SUPPORTED”
3. define user constraints for system pinout (edit minsoc/backend/yourboard.ucf file)
4. create project in project manager (ISE, Quartus), include files
5. synthesize, P&R and upload bitfile
6. connect the cable to the selected JTAG TAP

4 Software Upload and Debugging

Upload software and debug for simulation and implementation using GDB

1. start adv_jtag_bridge
 - a) cd ~/
 - b) sudo adv_jtag_bridge xpc3 (xess, usbbaster, xpc_usb, ft2232)
 - c) Let the program running and open another terminal
2. Open a terminal program (e.g. gtkterm)
 - a) configure port to a serial port connected to your board
 - b) configure bitrate to 115200
3. 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
4. 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)