Minimal OpenRISC System on Chip

How To

1 Download IP cores

1. download minsoc
2. download further necessary IP cores
   a) cd minsoc/rtl/verilog
   b) svn co http://opencores.org/ocsvn/adv_debug_sys/adv_debug_sys/trunk adv_debug_sys
   c) svn co http://opencores.org/ocsvn/ethmac/ethmac/trunk ethmac
   d) svn co http://opencores.org/ocsvn/openrisc/openrisc/trunk/or1200 or1200
   e) svn co http://opencores.org/ocsvn/uart16550/uart16550/trunk uart16550

2 Install GNU toolchain and adv_jtag_bridge

1. Follow: http://www.opencores.org/openrisc,gnu_toolchain (to install binutils, gcc, gdb)
2. To debug and load the firmware you have to use the new advanced_debug_system. This project is included in the minsoc files inside of minsoc/rtl/verilog/adv_debug_sys. There you can find the software in Software and the documentation, which shall help you to go under Doc.
   a) change the Makefile in minsoc/rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge and compile the software using make.
      - change Makefile line 34, "INCLUDE_JSP_SERVER=true" to "INCLUDE_JSP_SERVER=false"
      - make
      - sudo make install
   b) Copy the description file of your FPGA to your home directory "cp /opt/Xilinx/10.1/ISE/spartan3e/data/xc3s500e_fg320.bsd ~/"
3. With the adv_jtag_bridge you can also debug your simulation. To do so, the simulation has to include a vpi module. This has to be compiled by your system. The sources are found under "minsoc/rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge/sim_lib/icarus".
   a) cd minsoc/rtl/verilog/adv_debug_sys/Software/adv_jtag_bridge/sim_lib/icarus
   b) make
   c) cp jp-io-vpi.vpi minsoc/bench/verilog/vpi
4. The adv_jtag_bridge connect the debug system to gdb, the GNU debugger. But the actual version of gdb has some issues, which have to be corrected before use. To do so, the adv_jtag_bridge software includes a patch for gdb. Save the patch to the gdb source code directory installed by the toolchain installation script and patch it:

a) 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

b) cd toolchain_build_directory/gdb-6.8

c) patch -p1 < gdb-6.8-bz436037-reg-no-longer-active.patch

d) make

e) sudo make install

3 Compile Software

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

4 Simulation

1. Install Icarus Verilog
   a) You will need at least version 0.9.1 (ftp://ftp.icarus.com/pub/eda/verilog/v0.9/)
2. configure your system: minsoc/rtl/verilog/minsoc_defines.v **(not necessary)**
   a) **Note:** (**since revision 17)** Ethernet functionality of testbench does not base on eth_phy.v anymore. This means that you can uncomment ETHERNET on minsoc_defines.v and simulate it without simulation speed decrease and without changing any file.
   b) **Previous to revision 17:** if you uncomment "`define ETHERNET" you have to:
      - edit "minsoc/sim/run/generate_bench":
        - substitute "../bin/minsoc_model_fast.txt" for "../bin/minsoc_model_complete.txt"
      - THIS WILL SLOW DOWN YOUR SIMULATION BY FACTOR 300
3. configure minsoc/bench/verilog/minsoc_bench_defines.v **(not necessary)**
   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.

4. Modify testbench **(not necessary)**
5. 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”
6. Change or1200_defines.v (adaption to or1200_r3)
   a) `define OR1200_BOOT_ADR *32'hf0000100* to **32'h00000100**
7. 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

Debuging the testbench (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

5 Implementation

1. configure minsoc/rtl/verilog/minsoc_defines.v (recommended values for different devices under 7. Examples)
   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)
   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.
   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/or1200/rtl/verilog/or1200_defines.v (optional -> reduce logic usage) (recommended values for different devices under 7. Examples)
   a) Target FPGA memories (OR1200_XILINX_RAMB16 for Xilinx, Spartan 3 and above, OR1200_ALTERA_LPM for all Altera)
   b) Type of register file RAM (generic, twoport or dual port) (dual port is supported by Xilinx BRAM and Altera)**(select only one of the three)**
   c) If not using Linux you can comment out OR1200_QMEM_IMPLEMENTED.
3. 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"
4. Change or1200_defines.v (adaption to or1200_r3)
   a) `define OR1200_BOOT_ADR *32'hf0000100* to **32'h00000100**
5. define user constrains for system pinout (edit minsoc/backend/yourboard.ucf file)
6. create project in project manager (ISE, Quartus), include files
7. synthesize, P&R and upload bitfile
8. connect the cable to the selected JTAG TAP

6 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, usbblaster, 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)

7 Examples

Note: different constraint files for different boards → inside of backend directory

1. Spartan 3A DSP 1800
   a) minsoc/rtl/verilog/minsoc_defines.v
      - no definitions change, ready to go
   b) minsoc/rtl/verilog/or1200/rtl/verilog/or1200_defines.v (optional, reduce logic use)
      - uncomment `define OR1200_XILINX_RAMB16
      - uncomment `define OR1200_RFRAM_DUALPORT
      - comment `define OR1200_RFRAM_GENERIC
2. Spartan 3E Starter Kit **no Ethernet**
   a) minsoc/rtl/verilog/minsoc_defines.v
      - comment `define SPARTAN3A
      - uncomment `define SPARTAN3E
      - change CLOCK_DIVISOR from 5 to 2
      - comment `define ETHERNET
   b) minsoc/rtl/verilog/or1200/rtl/verilog/or1200_defines.v
      - uncomment `define OR1200_XILINX_RAMB16
3. Spartan 3E Starter Kit **with Ethernet**
   a) Synthesis properties:
      - Optimization Goal: Area
      - Optimization Effort: High
   b) minsoc/rtl/verilog/minsoc_defines.v

- comment `define SPARTAN3A
- uncomment `define SPARTAN3E
- let CLOCK_DIVISOR at 5
- change MEMORY_ADR_WIDTH from 13 to 12
- uncomment `define ETHERNET
- comment `define UART
  - this is not necessary, though you will get 99% device usage if not commenting, 89% otherwise.

c) minsoc/rtl/verilog/or1200/rtl/verilog/or1200_defines.v
- uncomment `define OR1200_XILINX_RAMB16
- comment `define OR1200_MULT_IMPLEMENTED
- comment `define OR1200_MAC_IMPLEMENTED
- uncomment `define OR1200_RFRAM_DUALPORT
- comment `define OR1200_RFRAM_GENERIC
- comment `define OR1200_PM_IMPLEMENTED
- comment `define OR1200_QMEM_IMPLEMENTED
- comment `define OR1200_CFGR_IMPLEMENTED

d) minsoc/rtl/verilog/ethmac/rtl/verilog/eth_defines.v
- uncomment `define ETH_FIFO_XILINX
- uncomment `define ETH_XILINX_RAMB4

e) Collateral effects:
- from sw/support/Makefile.inc line 7:
  - GCC_OPT=-mhard-mul -g to GCC_OPT=-msoft-mul -g
- change sw/support/orp.ld:
  - ram: LENGTH = from 0x00006E00 to 0x00002E00
  - this is not much memory, I recommend the inclusion of the wb_ddr project to minsoc to use your DDR SRAM memory
- change sw/support/board.h
  - IN_CLK to 10000000 //(10MHz) this will make the simulation have problems with the uart output but will work on implementation
  - STACK_SIZE to 0x00180
  - UART_BAUD_RATE to 9600 //baudrate 115200 leads to a high baudrate skew due to a truncation. PC cannot recognize the output
- reduce sw/eth.c

- remove lines 230-231
- remove lines 215-220
- remove line 206
- remove line 202
- change uart_print_long to uart_print_short, line 162
- change lines 98 and 99 to char tx_data[64] and char rx_data[64]
- remove lines 53-70 void uart_print_long(unsigned int ul) {}
- remove lines 31-42 void uart_interrupt(){}

f) Further area optimization possibilities: **(not necessary, DON'T DO)**

- Turn off: pic, tick timer or debug unit

4. Altera Devices

a) minsoc/rtl/verilog/minsoc_defines.v

- uncoment `define ALTERA_FPGA
- comment `define XILINX_FPGA
- comment `define SPARTAN3A
- select your memory amount "`define MEMORY_ADR_WIDTH 13"
- choose a clock division for your global clock related to your design max speed by changing the definition: "`define CLOCK_DIVISOR 5". Since you have an Altera device please use only even numbers for the division, odd numbers are going to be rounded down.
- Define your RESET polarity uncommenting "`define POSITIVE_RESET" or "`define NEGATIVE_RESET" and commenting the other.

b) minsoc/rtl/verilog/or1200/rtl/verilog/or1200_defines.v

- uncomment `define OR1200_ALTERA_LPM
- uncomment `define OR1200_RFRAM_DUALPORT
- comment `define OR1200_RFRAM_GENERIC
- comment `define OR1200_QMEM_IMPLEMENTED