Minimal Openrisc System on Chip Implementation

The Minimal System on Chip is a System-on-Chip (SoC) implementation with standard IP Cores available at OpenCores. This implementation is composed by a standard project, comprehending the standard IP Cores necessary for a SoC embedding the OpenRISC implementation or1200.

This project idea is to offer a SoC, which can be uploaded to every FPGA and be compatible with every FPGA board, without the requirement of changing its code. In order to deliver such a project, the project has been based on a standard memory implementation and the Advanced Debug System, which allows system debug with the same cables used for FPGA configuration.

The adaptation of the project to a target board is made in 2 steps maximum. First the minsoc_defines.v file has to be adjusted, generally one has to only uncomment his FPGA manufacturer and FPGA model definitions. After that a constraint file for your specific pinout has to be created. There are constraint files for standard boards also, in the backend directory of the project.

Furthermore the project offers for this same SoC a working testbench and firmwares. The actual testbench can be run out of the box using Icarus Verilog v. 9.1. The firmwares are nearly the same of those of orpsoc_v2. The differences are for now, that the known uart "hello world" example now runs with interrupts and a new Ethernet example has been added to it.

To complete, the size of the standard memory of the impementation can be adapted to your needs/possibilities by defining its address width inside of the same minsoc_defines.v file.

How To

1. install tools (binutils, gdb, adv_jtag_bridge)
   a) Follow: http://www.opencores.org/openrisc,gnu_toolchain (to install binutils, gdb, gcc, gdb)
   b) 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.
      - change the Makefile in Software/adv_jtag_bridge and compile the software using make.
      - sudo make install
      - Copy the description file of your FPGA to your home directory "cp /opt/Xilinx/10.1/ISE/spartan3e/data/xc3s500e_fg320.bsd ~/"
2. download minsoc
   a) download further necessary IP Cores
      - cd minsoc/rtl/verilog
      - svn co http://opencores.org/ocsvn/adv_debug_sys/adv_debug_sys/trunk adv_debug_sys
      - svn co http://opencores.org/ocsvn/ethmac/ethmac/trunk ethmac

- svn co http://opencores.org/ocsvn/openrisc/openrisc/trunk/or1200 or1200
- svn co http://opencores.org/ocsvn/uart16550/uart16550/trunk uart16550

3. Simulation
   a) Install Icarus Verilog
      - You will need at least version 0.9.1 (ftp://ftp.icarus.com/pub/eda/verilog/v0.9/)
   b) configure your system: minsoc_defines.v (if necessary)
      - For now, if you use "`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
   c) configure minsoc_bench_defines.v (if necessary)
      - 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.
      - You may use the actual implementation memory:
        - comment "`define INITIALIZE_MEMORY_MODEL"
        - edit minsoc/sim/run/generate_bench
          - substitute "../bin/minsoc_model_fast.txt" for "../bin/minsoc_memory_fast.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.
   d) Modify testbench (if necessary)
   e) command to start testbench and select firmware
      - from minsoc/sim/run/
        - ./generate_bench
        - ./run_bench <your_firmware.hex>
          - ./run_bench ../../sw/uart/uart-nocache-twobyte-sizefirst.hex
   f) Debuging the testbench
      - from minsoc/sim/run/
        - ./generate_bench
        - ./run_bench <your_firmware.hex>

        - ./run_bench ../../sw/uart/uart-nocache-twobyte-sizefirst.hex
    - ./start_server
  - from another terminal at minsoc/sw/uart
    - or32-elf-gdb uart-nocache.or32
    - target remote :9999
    - 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
    - set $pc=0x100
    - c

4. Implementation
   a) configure minsoc_defines.v
      - `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)
   b) configure or1200_defines.v (optional -> reduce logic usage)
      - Target FPGA memories (OR1200_XILINX_RAMB16 for Xilinx, Spartan 3 and above)
      - Type of register file RAM (generic, twoport or dual port) (dual port is supported by Xilinx BRAM)**(select only one)**
   c) define user constrains for system pinout (edit backend/yourboard.ucf file)
   d) create project in project manager (ISE, Quartus), include files
   e) synthesize, P&R and upload bitfile
   f) connect the cable to the selected JTAG TAP

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

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

        (e.g. 4*2^13 = 32,768 Bytes = 0x8000 | LENGTH = 0x8000 – 0x1200 = 0x6E00)
   b) select your STACK size on board.h line 16 #define STACK_SIZE 0x01000
      - change your IN_CLK if not using 25000000 (25MHz)
   c) inside of sw/support make clean, make all
   d) inside of the target software (e.g. sw/uart) make clean, make all

6. use GDB to upload software and debug for simulation and implementation
   a) start adv_jtag_bridge
      - cd ~/
      - sudo adv_jtag_bridge xpc3 (xess, usbblaster, xpc_usb, ft2232)
      - Let the program running and open another terminal
   b) Open a terminal program (e.g. gtkterm)
      - configure port to a serial port connected to your board
      - configure bitrate to 115200
   c) start gdb, load firmware (example)
      - cd minsoc/sw/uart
      - or32-elf-gdb uart-nocache.or32
      - target remote :9999
      - load
      - set $pc=0x100
      - c
   d) 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: different constraint files for different boards → inside of backend directory
   a) Spartan 3A DSP 1800 (100%)
      - minsoc_defines.v
        - no definitions change, ready to go
      - or1200_defines.v (optional, reduce logic use)
        - uncomment `define OR1200_XILINX_RAMB16
        - uncomment `define OR1200_RFRAM_DUALPORT
        - comment `define OR1200_RFRAM_GENERIC
   b) Spartan 3E Starter Kit (100%) **(not tested)**
      - minsoc_defines.v
        - comment `define SPARTAN3A
        - uncomment `define SPARTAN3E
        - change CLOCK_DIVISOR from 5 to 2
        - comment `define ETHERNET
      - or1200_defines.v
        - uncomment `define OR1200_XILINX_RAMB16

To Do:

1. **Implement a instantiable standard memory verilog file (90%)**
   a) compatible for simulation, xilinx, altera, asics (100%)
   b) resizable (100%)
   c) Able to read in 1 clock cycle (0%)
      - **minsoc_onchip_ram_top needs 2 cycles to complete a read operation, because the read acknowledge is triggered on the rising edge of wb_clk after wb_cyc has been set**
         - **Attempt to change it to negedge led to non running system for XILINX_RAMB16: neither CPU self test of adv_jtag_bridge nor firmware running did work**
2. Use a tc_top.v from older orpsoc, which manages the system memory and enables connection of peripherals (100%)
3. **Implement a standard clock divider, which is automatically configured by the system definition file (75%)**
   a) Standard (100%)
   b) Xilinx (100%)
   c) Altera (0%)
      - **For now Altera clock divider is implemented as the Standard**
   d) implement in a separated file (100%)
4. Implement a standard an unique system definition file, where one can select: (100%)
   a) how much memory to instantiate (100%)
   b) FPGA manufacturer and type (100%)
   c) which JTAG Tap to use (Generic of FPGA) (100%)
      - which fpga, overwrite _internal_jtag_options.v(100%)
   d) system clock, clock divider (100%)
   e) Which interfaces to be connected to the system (UART and ETH) (100%)
5. **Have standard software which can be directly compiled with make to be uploaded to the system (40%)**
   a) Have it (100%)
   b) direct set amount of memory and stack size for software based on system definition file (0%)
   c) direct adopt system address space as configured in definition file to the device drivers written in sw/support directory (0%)
6. **Have a standard testbench, with which one can simulate through iverilog easily and which can be easily redefined to simulate the environment (interfaces read and write functions)**

(90%)

a) **read and write for most interfaces (ETH, UART, CAN, I2C, SPI) (70%)**

b) regular testbench for the SoC (100%)

- runnable testbench (100%)
- debug interface (100%)
- uart output (100%)
- spi start-up (100%)
  - start-up rom memory (or1k-startup project) (100%)
  - create a SPI model to load the firmare to it at begin (100%)
    - create a memory, which can be written by $readmemh and read from through spi interface to the or1k-startup project (100%)
    - Add some glue logic to the SoC switch to assign the first commands to the openrisc to jump to the address of the or1k_startup 0x40000000. (100%)
      - instead of setting the address as in orpsoc 1, assert the instruction through wb_rim_dat_i . (100%)
- Fill memory with a $readmem code for fast firmware upload on simulation. (100%)
  - It is not allowed to access instances created by generate through a variable, so it is not possible to access each memory block from the memory to load the firmware before testbench execution (now working 100%)
    - Create a memory model, which changes the address width of the memory blocks allowing any memory depth with only 4 instances. Substitute the memory controller used by the implementation with it for the testbench. This way both testbench and implementation work. (100%)
      - This is only necessary for the `define INITIALIZE_MEMORY_MODEL, where the firmware is uploaded to the memory even before testbench execution.
      - To use the real memory from the implementation instead, comment this and change minsoc/sim/run/generate_bench script to use “minsoc/sim/bin/minsoc_memory_fast.txt” instead of “minsoc/sim/bin/minsoc_model_fast.txt”.
  - Filled memory does not run the complete program: (“o World.”) (now working 100%)
    - After reset by gdb (set $pc=0x100), continue leads to SIGBUS error. (now working 100%)
      - this seems to happen from the debug side, debug has asserted du_stall, why?
      - Reloading the program by gdb works, if we split the memory into 4 blocks (100%)
    - SIGBUS error and (“o World.”) output happened, because after the program size

bytes the following program goes into the memory starting from address 0x4 not 0x0. (now working 100%)

- select firmware on command line (100%)
- **Including tb_eth_defines.v , eth_phy_defines.v , eth_phy.v reduces simulation speed by factor 300 more or less, solve this (10%)**
  - **minsoc/sim/bin/minsoc_model_fast.txt removes them from bench initialization**
    - **This requires you to comment `define ETHERNET from minsoc/rtl/verilog/minsoc_defines.v**
    - **Then edit generate_bench to use minsoc_model_fast.txt**
  - **minsoc_model_fast runs hello world in 13 seconds**
    - **minsoc_model_complete would run it in 65 minutes**

7. Have different constraint files for different boards → inside of backend directory
   a) Spartan 3A DSP 1800 (100%)
   b) Spartan 3E Starter Kit (100%) **(not tested)**
   - minsoc_defines.v
     - comment `define SPARTAN3A
     - uncomment `define SPARTAN3E
     - change CLOCK_DIVISOR from 5 to 2
     - comment `define ETHERNET
   - or1200_defines.v
     - uncomment `define OR1200_XILINX_RAMB16