Minimal OpenRISC System on Chip

Status Progress

1 Implement a instantiable standard memory verilog file (90%)

1. compatible for simulation, xilinx, altera, asics (100%)
2. resizable (100%)
3. **Able to read in 1 clock cycle (30%)**
   a) **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**

   b) **Change back onchip_ram.v to generic implementation to posedge because it correspond to the reality of internal FPGA memories. (100%)**

   c) **phasing the clock connection from onchip_ram_top to the onchip_rams by 180° it works both in bench and on board. onchip_ram(.clk(~wb_clk_i));**
      - Though, self test of or32 by adv_jtag_bridge fails, has to be debugged
        - maybe the adv_jtag_bridge or the debug unit is stalling the cpu before it can complete something. It stores a program in the memory from 0x0 to 0x28.
          - memory 2 clks: runs till 0x20, stalls for 1 instruction, runs (0x28,0x10,0x14)
          - memory 1 clk: runs till 0x24, stalls for 1 instruction, runs (0x10)
        - difference: signal aborted_r from or1200_iwb_biu is supressed for 1 clk memory while not for 2 clk at instruction 0x20 (reading instruction from addr 0x20)
      - It is not standard to use the negative clock edge to solve this speed issue (check wishbone specification). Generally this speed decrease would get better with burst accesses, if the acknowledges after the first come instantaneously.
        - But for that, OR1200_WB_B3 has to be uncommented in or1200_defines.v and minsoc_onchip_ram_top.v has to be changed accordingly to use the signals: wb_bte_i and wb_cti_i
        - This is also only relevant if cache is implemented, otherwise all accesses from instruction wishbone interface are single accesses. (I guess)
4. Use a tc_top.v from older orpsoc, which manages the system memory and enables connection of peripherals (100%)

2 Implement a standard clock divider, which is automatically configured by the system definition file (75%)

1. Standard (100%)
2. Xilinx (100%)
3. **Altera (0%)**
    a) **For now Altera clock divider is implemented as the Standard**
4. implement in a separated file (100%)

3 Implement a standard an unique system definition file, where one can select: (100%)

1. how much memory to instantiate (100%)
2. FPGA manufacturer and type (100%)
3. which JTAG Tap to use (Generic of FPGA) (100%)
    a) which fpga, overwrite _internal_jtag_options.v(100%)
4. system clock, clock divider (100%)
5. Which interfaces to be connected to the system (UART and ETH) (100%)

4 Have standard software which can be directly compiled with make to be uploaded to the system (40%)

1. Have it (100%)
2. **direct set amount of memory and stack size for software based on system definition file (0%)**
3. **direct adopt system address space as configured in definition file to the device drivers written in sw/support directory (0%)**

5 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%)

1. **read and write for most interfaces (ETH, UART, CAN, I2C, SPI) (70%)**
2. regular testbench for the SoC (100%)
    a) runnable testbench (100%)
    b) debug interface (100%)
    c) uart output (100%)
    d) 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%)

e) 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%)

f) select firmware on command line (100%)

g) Including tb_eth_defines.v , eth_phy_defines.v , eth_phy.v reduces simulation speed by factor 300 more or less, solve this (100%)

- 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
- eth_phy.v removed from project, instead eth_rx_send task has been ported to the main testbench and is working. It does not handle collisions, but collisions only happen with more communicating nodes, which are not standard here. Speed is back to normal and task is functional.

6 Have different constraint files for different boards → inside of backend directory

1. Spartan 3A DSP 1800 (100%)
2. Spartan 3E Starter Kit (100%)
   a) minsoc_defines.v
      - comment `define SPARTAN3A
      - uncomment `define SPARTAN3E
      - change CLOCK_DIVISOR from 5 to 2
      - comment `define ETHERNET
   b) or1200_defines.v
      - uncomment `define OR1200_XILINX_RAMB16

7 To Do v. 2:

1. Add memory interfaces for external memory
   a) SDRAM, DDR, DDR2
2. Look for a way to allow automatically insertion of new modules to minsoc_top:
   a) memory address input
   b) automatic wishbone connection for minsoc_top
   c) automatic connection to minsoc_tc_top
   d) Full switch function, according to amount of masters, i.e.:
      - 2 masters, 2 buses, 2 router: arbiter has to assign correct bus to the calling master
   e) Switch issues, “minsoc_tc_top.v”:
      - modules instantiated by generate cannot be accessed through variable later on
      - maybe it can be done with parameters, macro and loop only
      - possibility of perlilog use for that
        - hardens testbench creation and raises compatibility issues (I suppose)