Duke ECE496 – Spring 2013 – Project Part 5: Unpipelined CPU

200 Points. Due electronically by 11:59pm on Tuesday, March 19

This is NOT an easy assignment. Start early! That way you have time to get help.

In this part of the project, you will build a complete but unpipelined single-cycle processor using the components that you have been building over the course of the semester. As part of this project, you will build your processor in Quartus (VHDL recommended), download your design onto an FPGA prototyping board, and demonstrate that it works by running a test program provided to you. Recall the specification for the Duke152-S13-32 Instruction Set Architecture in the Resources section in Sakai.

Project Part 5a: Single-Stage Single-Issue 32-bit Integer Processor

The processor has three inputs: a 1-bit input called "clock", a 1-bit input called "reset", and a 32-bit input called "keyboard_in". The processor has three outputs: a 1-bit output called "keyboard_ack", a 1-bit output called "lcd_write", and a 32-bit output called "lcd_data".

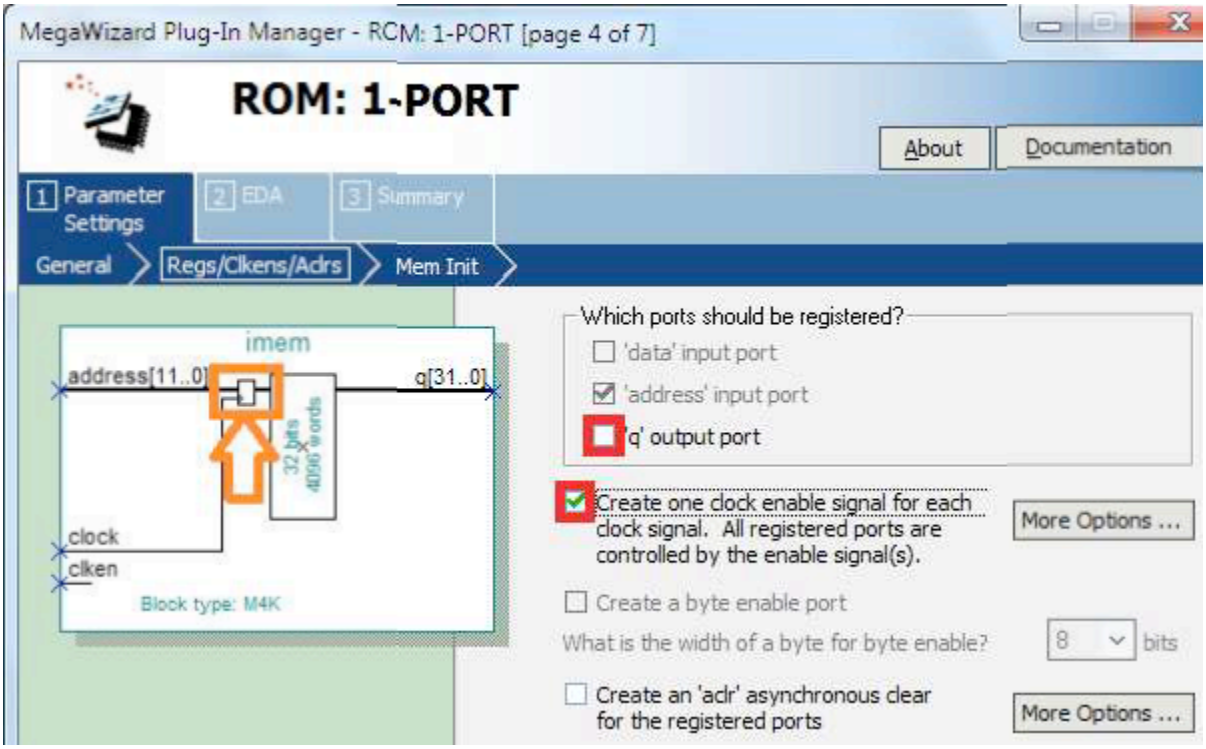
Until "reset" is asserted high, the state of the processor is undefined (can be anything). After "reset" is asserted, the processor begins execution from instruction memory address zero with an all-zeroes register file and progresses from there.

Your implementation of the "input" instruction should have the following semantics: on the same cycle that you read the data provided on "keyboard_in", you must assert "keyboard_ack" high for that clock cycle and only that clock cycle. Failing to assert it high will make you always read the same input; asserting it high for more than the one cycle will destroy unread data in the keyboard buffer.

Your implementation of the "output" instruction should have the following semantics: on the same cycle that you write data to "lcd_data", you must assert "lcd_write" high for that clock cycle and that clock cycle only. Failing to assert it high will cause no output to be displayed; asserting it high for more than the one cycle will display additional gibberish.

You may implement your instruction decode combinational using Behavioral VHDL. This is the *only* exception to the rule about not using Behavioral VHDL. You learned how to minimize logic in ECE 52; in this class we let the CAD (Computer Aided Design) tools do it for us as that is not the focus of ECE 152. When-else statements are recommended to generate simple combinational logic; process blocks are not.

Recall from Project Part 2 – Memory that the inputs to your instruction memory (imem) are registered (have flip-flops), as highlighted in orange in the diagram below. This input address register is a duplicate of your "Current PC" register. This means three things. First, it should be on the same clock edge as the "Current PC" register. Second, it should be reset to zero whenever reset is asserted. Third, whenever reset is not asserted, it should receive the same input as the "Current PC" register.



You will implement your processor in Structural VHDL using the Quartus II software. You should create one VHDL (processor.vhd) or Block-Diagram (processor.bdf) file that has exactly the same format as the diagram in Figure 1. This top-level processor.vhd or processor.bdf file is likely to refer to other lower-level files (e.g., imem.vhd, dmem.vhd, alu.vhd, regfile.vhd, etc.). The top-level file processor.vhd or processor.bdf file is what you will then insert into the skeleton framework for downloading to the FPGA. Figure 1 is a screenshot of the processor component in Quartus, and it shows the signal names that you MUST use in your design to facilitate testing, grading, and interfacing with the skeleton framework.

Project Part 5b: Testing Your Unpipelined Processor

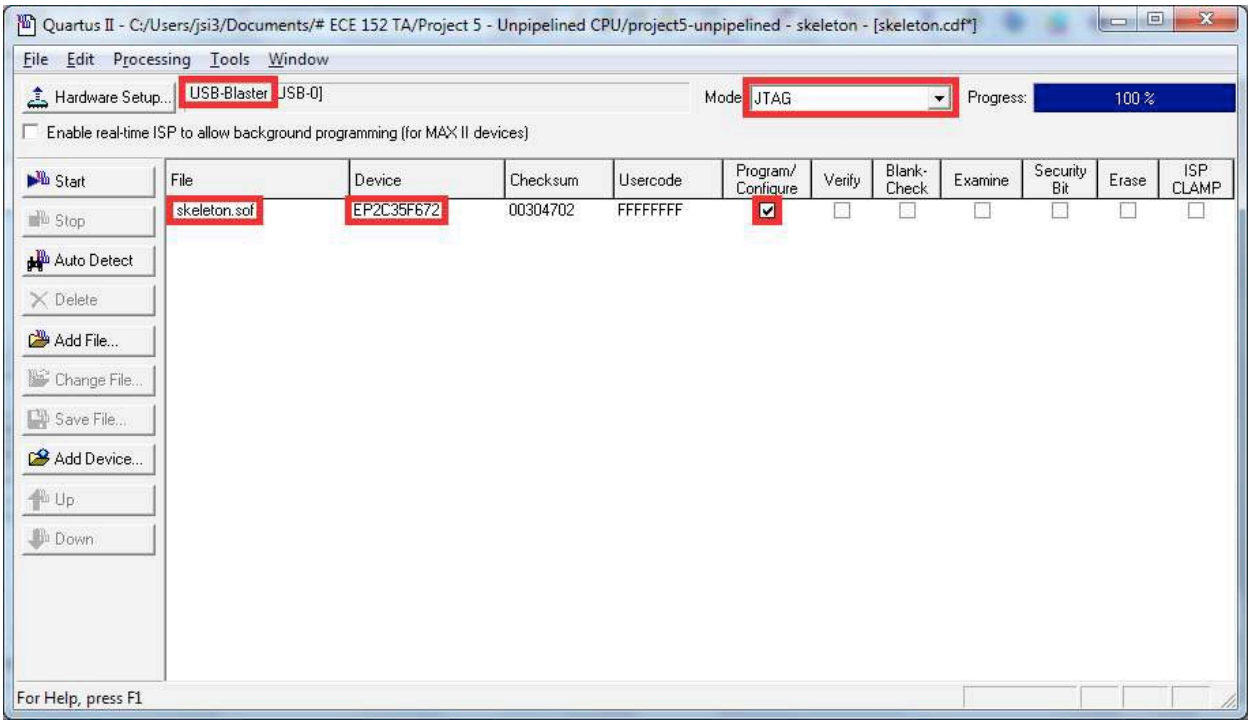
After designing your processor, you should use Quartus to test it thoroughly before attempting to download it to the FPGA. In engineering, it is assumed that a complicated design does not work until proven otherwise through testing. One good way to test your processor is to write short simple programs to exercise specific instructions, run them, and make sure that the expected results are produced. It is recommended that you create additional outputs for debugging purposes to observe selected signals in your processor (e.g., PC, data_writeReg, ALU inputs, etc.). To assemble and simulate programs for your processor, an assembler/simulator is provided in Sakai under the assignment page. Running the assembler produces the Instruction and Data Memory Initialization Files. The simulator is a functional architectural simulator only, i.e. no microarchitecture-specific timing. Test programs testGiveMeN.asm and testFibonacci.asm are provided in the zip file. testGiveMeN simply inputs a number from the keyboard, stores the number, and then prints the number to the LCD. testFibonacci inputs a number N from the keyboard, calculates the Nth Fibonacci number, and then prints the result to the LCD. Note that these are old 16-bit programs and thus are not examples of optimized code and may not work with

large numbers. A sample waveform for your unpipelined processor running the testGiveMeN program is provided at
http://people.ee.duke.edu/~sorin/ece152/project/testGiveMeN_unpipelined.vwf

Project Part 5c: Hardware Demonstration

After designing and testing your processor in Quartus, you will download it to one of the Altera DE2 FPGA prototyping boards and demonstrate that it works using the "testFibonacci" program. Do not wait until demo day to complete this final step; just because your design works in Quartus does not mean that there will not be hiccups getting it onto the board. A skeleton framework with all of the pin mappings and keyboard and LCD controllers is provided at http://people.ee.duke.edu/~sorin/ece152/project/skeleton.qar. Restore from the archive (Project Restore Archived Project) and add your processor into this skeleton project.

To download your processor onto the FPGA, perform a full compilation of the skeleton framework project with your processor included (Processing Start Compilation). After compilation finishes, select Tools Programmer. If "No Hardware" is listed in the top-left corner of the Programmer window, plug in the board via USB, click "Hardware Setup" and select "USB-Blaster" from the menu, then hit Close. Select "JTAG" mode, ensure that the file is "skeleton.sof" on device "EP2C35F672", and check "Program/Configure".



Set the toggle switch to the left of the LCD to "RUN", then hit the red button to turn on the device. In the Quartus Programmer, press Start. Programming is fairly quick, and the program will start immediately once programming is complete. The "Reset" button is the bottom-right blue button labeled "KEY0". You will have to reprogram the device every time you power it on.

Submitting This Assignment

To submit this assignment, create a Quartus Archive (Project Archive Project) named project5.qar of all the files needed to implement your design. Make sure that your processor file is named processor.vhd or processor.bdf. Names of lower-level files are unrestricted, but be sure to include them along with your top-level design entity in the Quartus Archive file. Submit your Quartus archive to the assignments section in Sakai. You will also demonstrate your working processor on an FPGA board in lab on a day to be determined by the instructor at a later date.

Figure 1: processor VHDL and BDF screenshots

```
ENTITY processor IS
    PORT (  clock, reset    : IN STD_LOGIC;
            keyboard_in : IN STD_LOGIC_VECTOR(31 downto 0);
            keyboard_ack, lcd_write : OUT STD_LOGIC;
            lcd_data    : OUT STD_LOGIC_VECTOR(31 downto 0) );
END processor;
```