

Project: Part #5 for ECE 152

An Unpipelined Processor

200 points

Must be submitted electronically by 10:00AM on Wednesday, March 26

IMPORTANT: This is NOT an easy assignment. Get started early!! That way, when problems arise (which they will!), you will have time to ask me and/or the TAs for help.

In this part of the project, you will build a complete, but unpipelined, Duke 152/16 processor using the components you have been building during the course of the semester. As part of this project, you will:

- Demonstrate that your design works correctly in Quartus and show how fast it can execute a test program that I have provided for you at: <http://www.ee.duke.edu/~sorin/ece152/project/testprogram1.sim>.
- Download your design into an FPGA prototyping board, demonstrate that it works, and show how fast it can execute my test program.

Recall that the specification for the Duke 152/16 architecture can be found at: <http://www.ee.duke.edu/~sorin/ece152/project/duke152-arch.pdf>

1 Requirements

The processor has three inputs:

- a 1-bit input called `clock`
- a 1-bit input called `reset`
- a 16-bit input from the keyboard called `keyboard_in`

The processor has three outputs:

- a 1-bit output called `keyboard_ack`
- a 16-bit output called `lcd_out`
- a 1-bit output called `lcd_write`

See Section 3 for information on how to hook up these signals to the correct locations on the FPGA prototyping board. In Quartus, these inputs and outputs are pins.

Until `reset` is asserted with a 1, the state of the processor is undefined (can be anything). After `reset` is asserted, the processor begins executing the instruction at memory location zero and continues from there.

Your implementation of the `input` instruction must have the following semantics: on the same cycle that you read the data provided on `keyboard_in`, you must assert `keyboard_ack` (with a 1) for that clock cycle (and that clock cycle only). Failing to assert it will make you always read the same input; asserting it multiple times will destroy other data in the keyboard buffer.

Your implementation of the `output` instruction must have the following semantics: on the same cycle that you write data to `led_out`, you must assert `led_write` (with a 1) for that clock cycle (and that clock cycle only). Failing to assert it will cause no output to be rendered. Asserting it multiple times will cause the same character to be written multiple times.

Your microarchitecture must use multi-cycle control that is implemented with a finite state machine. I would permit micro-coded control, but the boards we have in the lab do not easily allow this. I will permit the use of VHDL or Verilog for designing the control.

2 Testing Your Processor

You must test your processor to make sure it is correct as well as to see how fast it is.

2.1 Correctness

After designing your processor, you must use Quartus to test it *thoroughly* to demonstrate that it works correctly. It is a well-known engineering fact that anything that hasn't been tested doesn't work. Do not underestimate the importance of testing.

One good way to test your processor is to write simple programs for it, run them, and then make sure they produce the expected results. To help you in this process, we have provided an assembler. The assembler infrastructure is in a tarball file at <http://www.ee.duke.edu/~sorin/ece152/project/duke152asm.tar.gz>. Copy this file to your directory and unpack it (i.e., with `gunzip` and `tar -xvf`). Build the assembler with `make asm`. We have

also provided a simple code example, called `simple.s`, in the tarball. Running the assembler on this assembly program (`asm simple.s`) produces three files: `simple.dmem.mif`, `simple.imem.mif`, and `simple.sim`. The first two files are the data memory and instruction memory images, respectively. The third file is for use with the simulator (also in the tarball).

2.2 The Need for Speed

Using the Quartus simulator, you must determine how fast your processor can execute instructions. You must consider the critical path, since it is not correct to clock something at a given frequency if that frequency is sometimes too fast. At the end of the semester, I will give bonus points to the fastest processors, so keep speed in mind for the future.

3 Hardware Demos

After designing and testing the processor in Quartus, you will download it into one of the Altera DE2 FPGA prototyping boards. You will test that it works on the board.

We are providing you with a Quartus “skeleton project” that already contains all of the pin mappings for the LCD, keyboard, clock, and reset. It is at: <http://www.ee.duke.edu/~sorin/ece152/project/skeleton.zip>. Copy this file to your directory and unzip it. You will get a directory called `skeleton/`. You will add your processor components into this skeleton project.

To download your processor onto the FPGA. After performing a full compilation of your project, select Tools->Programmer from the menu. If "No Hardware" is listed in the top-left of the programmer, click "Hardware Setup" and select USB-Blaster from the menu and hit close. Indicate that you want to Program/Configure the file that corresponds to EPCS16.

Make sure the DE2 board is connected to power and connected to the computer via USB. Set the toggle switch on the left side of the board to "PROG", then hit the red button to turn on the device. Back in Quartus, click the Start button, and the programming process should begin (it takes nearly a minute). When it completes, turn the board off, set the toggle switch back to "RUN", then power on the board.

To load your program into the instruction memory. Open the MegaWizard Plugin Manager. Select edit an existing custom megafunction, and select imem.vhd (which you created in the last part of the project). On page 10, you are asked if you want to specify the initial content of the memory; say yes, and indicate the location of the imem memory initialization file. Loading the data memory works the same way.

Demonstrations. We will have graded demos of the FPGA implementations that will be held in the ECE 154 lab (Hudson 202A) just after the submission deadline (exact times to be announced later). You will want to test your designs on the hardware long before then, to make sure they work correctly (even if they worked without problem in Quartus). To provide you with access to the lab with the prototyping boards (Hudson 202A), the TAs will hold office hours in the lab during the last week before the project is due.

4 Submission

Submit this assignment in the same way in which you submitted previous project parts. You must name your high-level .bdf file `processor.bdf`. There are no other file naming restrictions.

You may re-submit as often as you like, but a re-submission will overwrite whatever you've previously submitted for this assignment. I will grade whatever has been submitted before 10:00AM on Wednesday, March 26.