CprE 381: Computer Organization and Assembly Level Programming

Project Part 1

*[Note: this is a TEAM assignment and should be actively collaborated on and submitted as a single project team. Although you are still required to document your design process via a lab report, your focus should be more on creating a smart design that you will use for the remainder of the semester and **testing** it thoroughly than on a lengthy writeup. The TAs and I are still available to help during the lab sessions, office hours, and on Canvas, but there is now much more freedom in how to implement components and you are expected to come up with major aspects of the design as a project team. Fortunately, you have **two** weeks to complete this assignment.]*

0) Prelab. With your project group members, create a **list of best practices** / tips for designing, compiling, and testing VHDL modules based on your experiences so far with these labs, both working individually and as a group. *[Make sure to include the limited cases when process statements may be used as discussed in lecture.]*

1) The MIPS ISA also contains several shift instructions. For this part we will consider the **barrel shifter** design discussed in class.

(a) Describe the difference between logical (`srl`) and arithmetic (`sra`) shifts. Why does MIPS not have a `sla` instruction? *[This may require you to look up these instructions in an ISA specification such as P&H textbook A.10.]*

(b) Unfortunately, barrel shifters are not covered explicitly in the P&H textbook, but the following Java applet shows how an 8-bit barrel shifter can be created using cascaded 2:1 muxes (https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/60-barrel/shifter8.html). Implement a 32-bit right shifter (both arithmetic and logical) using structural VHDL. In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations. *[There is a strong pattern to how the muxes are mapped in the conventional barrel shifter design. If you make sense of this pattern your code will be considerably simplified.]*

(c) In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations. Integrate this functionality into your existing design. *[Hint: it can be done by trivially modifying the input and output. An additional shifting component is not necessary.]*

(d) Use Modelsim to test your shifter designs thoroughly to make sure they are working as expected. Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.

2) You will update your ALU to meet the full requirements of the MIPS ISA. Your **MIPS ALU** will take three inputs (operand A, operand B, and a minimum-width control input called ALUOP), and four outputs (result F, Carryout, Overflow, and Zero). It must support following operations: `add/sub` (both signed and unsigned), `slt`, `and`, `or`, `xor`, `nor`, `sll`, `srl`, and `sra`.

(a) Draw a simplified schematic for this 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is `slt` implemented? *[The answer to these questions can be found in P&H Section B.5 or in lecture slides. Verify with your TA before proceeding. Your design may require you to update the implementation of your adder/subtractor.]*

(b) Implement this 32-bit ALU using structural VHDL. *[Leverage as many of the components you and your partner have already designed and tested as possible!]*

(c) Use Modelsim to test your 32-bit ALU thoroughly to make sure it is working as expected. Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

3) You should now be able to demonstrate using a **test program** a much wider variety of behavior than your Lab 4 datapath.

(a) Integrate your new ALU into your MIPS datapath from Lab 4.

(b) Augment your previous test program(s) with ones that test the new functionality from this lab (e.g., overflow, slt, or, and, nor, xor, sll, srl, sra, etc.). Don't simply test that they work for one value, test them for multiple values including edge cases and justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.

4) [OPTIONAL] Add a byte-sized SIMD functional unit that can support .

(a) Design a separate ALU to support the MIPS DSP instructions: `absq_s.qb`, `addu.qb`, `addu_s.qb`, `raddu.w.qb`, `repl.qb`, `replv.qb`, and `subu.qb`. *[Note that you only need lw and sw to load and store quad bytes, so no additional memory functions are required.]*

(b) Integrate your SIMD ALU into your MIPS datapath from part 3.

(c) Augment your previous test program(s) with ones that test the new functionality from this lab. Don't simply test that they work for one value, test them for multiple values including edge cases and justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.

5) [OPTIONAL – *Starting with your first full processor you will be required to synthesize the design, so you may want to ensure that your initial ALU can be synthesized. However, only attempt this if you have a fully-functional design based on your Modelsim simulations!***]**As we discussed in lecture, the design of the components within the datapath can impact the worst-case length of time it takes input values to propagate to output values (i.e., a design's **critical path**). Since we don't know at design time when or if these worst-case inputs will arise, we have to guarantee that the worst-case inputs will have the time to affect the outputs by setting the clock period to be long enough to according to the worst-case propagation time. *[This is a primary motivation for the performance optimization called pipelining that we will be learning about in several weeks. Being able to understand where the critical paths arise will help us in our pipelining decisions.]*

(a) Open Quartus and create a new project and include your VHDL files.
Click: 'File' → 'New Project Wizard'

Click ‘Next >’
Enter the directory for your project files and a fitting name for your project.
Click ‘Next >’
Click ‘Next >’
Click ‘…’ button next to the ‘File Name’ blank.
Select all of your files to add to this project.
Click ‘Next >’
Make sure the ‘Cyclone IV E’ Family is selected under the ‘Device’ tab.
Selected the EP4CE115F29C7 device is selected under ‘Available devices’.
Click ‘Next >’
Click ‘Next >’
Click ‘Finish’

(b) Set your MIPS datapath VHDL file as the top level of your project.
Select ‘Files’ in ‘Project Navigator’
Right click on your datapath VHDL and click on ‘Set as Top-Level Entity’.

(c) Run the compile and synthesis for your project. This step will take some time, mainly due to the size of the memory model. *[Up through your single-cycle processor design we will be using a model that won’t synthesize to SRAM blocks, but rather each bit synthesizes to an individual flip-flop element. Quartus is treating these as 256k different logic elements during optimizations, hence the long synthesis time. After the single-cycle design we will start using a memory model which directly maps to SRAM blocks on the FPGA and synthesis should improve in performance.]*
In the ‘Tasks’ pane, expand the ‘TimeQuest Timing Analysis’ item.
Double click on ‘Edit Settings’.
Click on the checkbox ‘Report worst-case paths during compilation’.
Double click on ‘Compile Design’.

(d) Set your clk as a constrained clock in Quartus and rerun the timing analysis. Does your design meet the timing constraints that were set? Report the critical path from register to register using TimeQuest. What is the fastest clock period that your MIPS datapath can be safely run at?
Under the ‘Compilation Report’, expand ‘TimeQuest Timing Analyzer’.
Click on ‘Clocks’.
Right click on ‘CLK’ and select ‘Edit Clock Constraint…’.
Set ‘Period’ and ‘Falling’ to make a 50MHz clock.
Click ‘Run’.
Click on ‘Report Top Failing Paths’ Histograms’ under ‘Macros’ in the ‘Tasks’ frame.
Right click on the 1 in for the first row. Click on ‘Report Worst-Case Path’. In the ‘Extra Fitter Information’ tab you can now see the trace of the path through your design starting with the driving register and ending with the reading register.
For fun, double click on ‘Create All Clock Histograms’ under ‘Macros’ in the ‘Tasks’ frame. You will be able to see the distribution of slack relative to your clock’s period (0ns of slack). Slack is how much extra time a particular path has between its worst-case propagation delay and the clock period. A negative slack means that the path violates timing. This should give you an idea of how close your design is to meeting the timing requirements overall.

Credit: Parts of this project description were originally created by Dr. Joe Zambreno.