ENGR xD52: MP b010

Due November 8th 8PM EST

This lab assignment stresses reuse: Reuse of design components and reuse of structural elements. Additionally, it will help you understand the timing constraints of more modern architectures.

Work in groups of 3 or 4. This may require some team reshuffling. One representative from each team must submit all deliverables electronically to comparch13@gmail.com as a single compressed archive. Please format the email's Header as "[CA] [MP1] Name1 Name2 Name3"

The Situation

You have successfully been hired by National Instruments (you lucky dog you) in to their ASIC design team. They are creating a tiny semi-custom MIPS-like CPU core to be included in their next chip. Given the specific algorithms this core will be computing, they have elected to remove unnecessary instructions in favor of die-space. You are responsible for the ALU.

The Work Plan

This subdeliverable is due within 36 hours as an email to comparch13@gmail.com with the subject line formatted as [CA][MP2 Work Plan] Name1 Name2 Name3". Any questions regarding this section should be directed to Molly Farison or Eric VanWyk.

Create a work plan for this lab. Break down the lab in to small portions, and for each portion predict how long it will take (in hours) and when it will be done by (date). You will be comparing your predictions to reality later. Use what you learned in MP1!

The Tests

This subdeliverable is due November 1st 8PM EST.

Construct a Test Bench for your ALU in behavioral verilog. This is testing for design correctness, it does not need to test for manufacturing faults. Be intelligent in your selection of your test cases: Making use of the hierarchy of your design will allow you to avoid exhaustive testing.

The portion of the writeup about your testbenches are also due with this portion of the submission. They may be updated for the final submission as appropriate.

Note: Doing this section well will require you to have *started* the rest of the Lab. You should probably have all the module headers written, but you don't need any of the module bodies yet.

The ALU

Construct a reduced-instruction ALU for a 32-bit MIPS CPU. This ALU must implement ADD, SUB, XOR, and SLT. Additionally, it must implement at least one of the following: SLLV, SRAV, SRLV, or MUL.

The Writeup

Create a semi-formal write-up of your ALU design. Assume that the primary intended audience is your project manager. Convince her that your design is ready to be included in the tape-out of the upcoming prototype run. This should include the following:

Correctness

Use your Test Bench to provide evidence that your design is functionally correct. You may use knowledge of the structure of your design in your analysis – you do not have to treat it as a black box as you did in the previous lab. Be sure to explain why you chose the tests you did, and more importantly why you didn't include the tests you chose to omit.

Timing Analysis

Give expected worst-case propagation delays for each of the implemented ALU instructions.

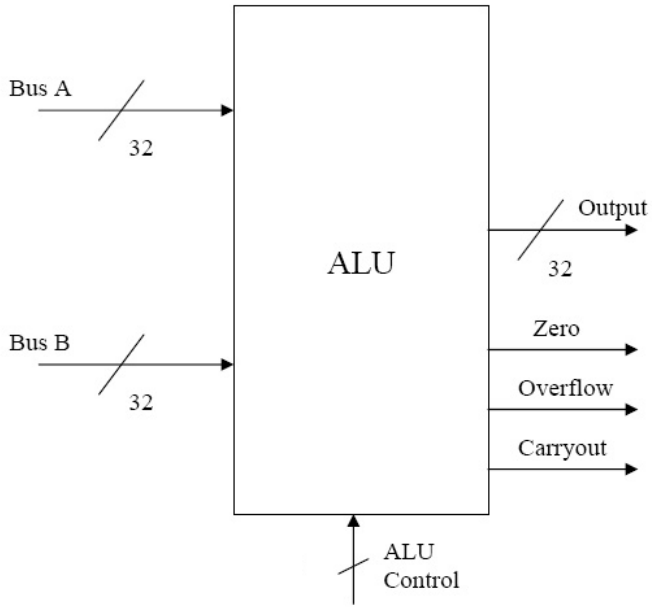
Give at least one example timing diagram. Label which components are "active" (causing delay) during the time example.

Design Justification

Provide reasoning for your design decisions and design trade-offs. For example, you will have to make several area-vs-speed tradeoffs in this design; document them.

Implementation

The MIPS ALU has 7 ports: the two input ports A and B, the output port, ALU control, zero detect output, overflow detect output, and the carryout output. The ALU control line assignments are given below. Please use these inputs to select the ALU function.



Control Lines	Function
000	ADD
001	SUB
010	XOR
011	SLT
100	MUL
101	SLLV
110	SRAV
111	SRLV

Hints / Notes

Timing

In order to be slightly more accurate, we will model gate delay as proportional to the number of inputs in the gate. Standardize on 10 units per input. Therefore, an inverter has delay 10, a 32-input AND gate has delay 320. The basic gates are NAND, NOR, NOT. AND, OR, etc all have 'hidden' inverters that must be accounted for in your propagation delays.

This timing formula is still a horrible horrible lie that Brad will have to fix when you take VLSI. Sorry Brad

Design Reuse

You may freely reuse code created for previous labs. **Be sure to redo the timing!**

Behavioral vs Structural

Only use behavioral Verilog where you are already confident in the structural equivalent. This is only to save some typing. When you do this, include a brief description of the structural equivalent as a comment, and use this information to create an appropriate propagation delay.

For example:

```
// This instantiates a 2 input mux, which uses a 2-input OR, 2 2-
input ANDs, and an inverter.  The propagation delay is ____.
```

I explicitly do not want to see complex or compound behavioral statements. There are ways to instantiate multipliers or shifters in a single line of behavioral Verilog. Do not use these.

Each behavioral element must be independently wrapped in its own structural box, as per the in class examples.

Preprocessors

Use preprocessors or code generators as appropriate. I would appreciate it if teams shared their preprocessors/code generators from the previous lab. If you use someone else's generator, credit them in your writeup.

Documentation

Code legibility and commenting can be worth up to a third of your grade in this lab. The key is to make sure that your code is "inheritable" – If you leave the company, your replacement should be able to understand what you were up to without re-inventing it. Block comments at the top of your modules, occasional intramodule comments describing interesting details.