Project Checkpoint #1

ALU

Logistics

This is the second Project Checkpoint for our processor.

- Due: **Tuesday, September 17, 2019** by **11:59:59PM (Duke time)**
- Late policy can be found on the course webpage/syllabus

Introduction

Design and simulate an ALU using Verilog. You must support:

- a **non-RCA** adder with support for addition & subtraction
- bitwise AND, OR **without** the built in &, &&, |, and || operators
- 32-bit barrel shifter with SLL and SRA **without** the <<, <<<, >>, and >>> operators

We will post clarifications, updates, etc. on Piazza so please monitor the threads there.

Module Interface

Designs which do not adhere to the following specification will incur significant penalties.

Your module must use the following interface (n.b. It is the template provided to you in alu.v):

```
module alu(data_operandA, data_operandB, ctrl_ALUopcode,
ctrl_shiftamt, data_result, isNotEqual, isLessThan, overflow);

   input [31:0] data_operandA, data_operandB;
   input [4:0] ctrl_ALUopcode, ctrl_shiftamt;

   output [31:0] data_result;
   output isNotEqual, isLessThan, overflow;

endmodule
```

Each operation should be associated with the following ALU opcodes:

Operation	ALU Opcode	Description
ADD	00000	Performs `data_operandA + data_operandB`
SUBTRACT	00001	Performs `data_operandA - data_operandB`
AND	00010	Performs (bitwise) `data_operandA & data_operandB`
OR	00011	Performs (bitwise) `data_operandA | data_operandB`
SLL	00100	Logical left-shift on `data_operandA`
SRA	00101	Arithmetic right-shift on `data_operandA`

Control Signals (In)

- `ctrl_shiftamt`
  - Shift amount for SLL and SRA operations
  - Only needs to be used in SLL and SRA operations

Information Signals (Out)

- `isNotEqual`
  - Asserts true **iff** `data_operandA` and `data_operandB` are not equal
  - Only needs to be correct after a SUBTRACT operation
- `isLessThan`
  - Asserts true **iff** `data_operandA` is **strictly** less than `data_operandB`
  - Only needs to be correct after a SUBTRACT operation
- `overflow`
  - Asserts true **iff** there is an overflow in ADD or SUBTRACT
  - Only needs to be correct after an ADD or SUBTRACT operation

Permitted and Banned Verilog

Designs which do not adhere to the following specifications cannot receive a score.

No "megafunctions."

Use **only** structural Verilog like:

- `and and_gate(output_1, input_1, input_2 ... );`

And not syntactic sugar like:

- `assign output_1 = input_1 & input_2;`
- `==, >=, =<, etc.`

except in constructing your DFFE (i.e. you can use whatever verilog you need to construct a DFFE).

However, feel free to use the following syntactic sugar and primitives:

- `assign ternary_output = cond ? High : Low;`
    - The ternary operator is a simple construction that passes on the “High” wire if the cond wire is asserted and “Low” wire if the cond wire is not asserted

Grading Breakdown

Test	Percentage	Scoring Method
Less Than	5	All or Nothing
Or	5	All or Nothing
Addition	25	Proportional
And	5	All or Nothing
Not Equal	5	All or Nothing
Logical Left Shift	10	Proportional
Arithmetic Right Shift	10	Proportional
Subtraction	30	Proportional
Overflow	5	All or Nothing

Other Specifications

Designs which do not adhere to the following specifications will incur significant penalties.

Your design must operate correctly with a 50 MHz clock. Also, please remember that we are ultimately deploying these modules on our FPGAs. Therefore, when setting up your project in Quartus, be sure to pick the correct device (check the “Quartus Environment Setup” PDF found under Resources > Labs on Sakai).

Submission Instructions

Designs which do not adhere to the following specifications will incur significant penalties.

Writing Code

- Use duke.ag350project.com and the corresponding GitHub repo to manage your codebase
- Branch off of master to implement your projects and merge changes back into master when you've completed a feature or you want to test
- Keep all of your source files in the top-level directory (don't create sub folders for your source code; generated files is fine)
- Be sure to only put files into version control that are source files (*.v)
    - Modify .gitignore at your own risk
    - The Autograder will compile and run your *.v files, so it needs to be able to find all of them
    - The Autograder will add a _master_ag.tb.v file, so please don't name any of your files this
    - If you change how your repo is configured, you may run the risk of breaking the Autograder scripts
- Make sure you structure your code so that alu.v is the top-level entity and it contains the provided alu interface

Submission Requirements

- A repository containing your code
- A README.md (written in markdown, Github flavor) that includes
    - A text description of your design implementation (e.g., "I used X,Y,Z to ...", or "My hierarchical decoder tree was...")
    - If there are bugs or issues, descriptions of what they are and what you think caused them

Grading

Complete, working designs will be tested for a grade using a test bench.

Resources

If you followed the setup instructions correctly, your repository has a sample testbench titled alu_tb.v. This should help you test your alu and also write test benches in the future. However, the test bench used for grading will be more extensive than the one presented here. **Passing the included test bench does not ensure that you will pass the grading test bench.**