Project Checkpoint 1

Simple ALU Checkpoint 1

Logistics

This is the first Project Checkpoint for our processor. We will post clarifications, updates, etc. on Ed so please monitor the threads there.

- Due: **Friday, September 23, 2022,** by **11:59 PM** **(Duke time)**
    - Late policy can be found on the course webpage/syllabus

Introduction

You will design and simulate an ALU using Verilog. You must support:

- a **non-RCA** adder with support for addition & subtraction

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`

Control Signals (In)

- `ctrl_shiftamt`
    - Shift amount for SLL and SRA operations
    - Only needs to be used in SLL and SRA operations (not required for this checkpoint)

Information Signals (Out)

- `isNotEqual` (not required for this checkpoint)
  - Asserts true **iff** `data_operandA` and `data_operandB` are not equal
  - Only needs to be correct after a SUBTRACT operation
- `isLessThan` (not required for this checkpoint)
  - 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

Test	Percentage	Scoring Method
Addition	40	Proportional (-0.5 pts / test case fail until zero)
Subtraction	40	Proportional (-0.5 pts / test case fail until zero)
Overflow	20	All or Nothing

Submitted designs will be tested for a grade using a test bench with numerous test cases, including corner cases.

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 recitation 1).

Submission Instructions

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

Writing Code

- Keep all of your source files in the top-level directory (don't create sub folders for your source code; generated files are fine)
- Make sure you structure your code so that alu.v is the top-level entity and it contains the provided alu interface
- **Make sure the name of all testbench file ends with '_tb.v'**, or it will be involved in the style check and negatively affect your submission grade.
- Change how your repo is configured at your own risk
- You can choose to use the GitHub repository to manage your codebase if you are familiar with that.
    - 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.
    - Be sure to only put files into version control that are source files (*.v)

Submission Requirements

- One group should only upload one file
- Use Gradescope to submit your code and a README.md file. You should submit **one .zip file or a link** to the GitHub repository containing **all necessary modules** for your project. Including *.qpf, *.qsf, *.qws files are fine. You don't need to submit the folders or the bank up files.
- A README.md (written in markdown, Github flavor) should includes
    - You and your partner's name and netID
    - 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

Resources

We have provided you base code in the attachment on Sakai including the "alu.v" file which you can start coding with and the testbench "alu_tb.v" file may help you test your code primarily. This should also help you test your alu and 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.**