Project Checkpoint 1

Simple ALU

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: **Tuesday, September 17, 2024,** by **11:59 PM** **(Duke time)**
  - Late policy can be found on the course webpage/syllabus
- You can submit an unlimited number of times before the deadline, and each submission will show whether the compiling requirement is passed. Your final grade will be for your last submission before the deadline.
- To give you a chance to improve your results, we will run the auto-grader 2 days before the deadline so that you can see your temporary grade based on your last submission before then.

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`

Both data_operandA and data_operandB are **signed**, therefore, we need to confirm whether an overflow happens or not. Please utilize 2's complement for the operations.

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.

You can use

- Ternary assign: `assign out = cond ? high : low;` (cond, high, low must be wire(s) (or input/output ports), you should not write an expression in cond)
  - For example, assign data_result = (ctrl_ALUopcode == 2'b00000) ? Add_result : Sub_result; You cannot use '==' here, because you should not write an expression in cond.
- Primitive instantiation: `and (out, in1, in2)`
- Generate Blocks: generate if, generate for, genvar (This is a tutorial if you do not know them: https://fpgatutorial.com/verilog-generate/)
- RCAs to construct your 32-bit adder, as long as the 32-bit adder is not RCA.
- Parameters: `parameter a=0; localparam b = a*2;`
- Any expression you like **inside the range specifier**: `a[i*15+36/2-13%2]`

You cannot use

- Behavioral Description Structures: `if ... else ... for...` (This is a loop, not the generate for, which is allowed.) `case ...`
- Megafunctions outside the range, generate control or parameter expressions: `+, -, *, /, %, **, ==, >=, <=, &&, ||, !, <<, <<<, >>, >>>`
- SystemVerilog

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

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.

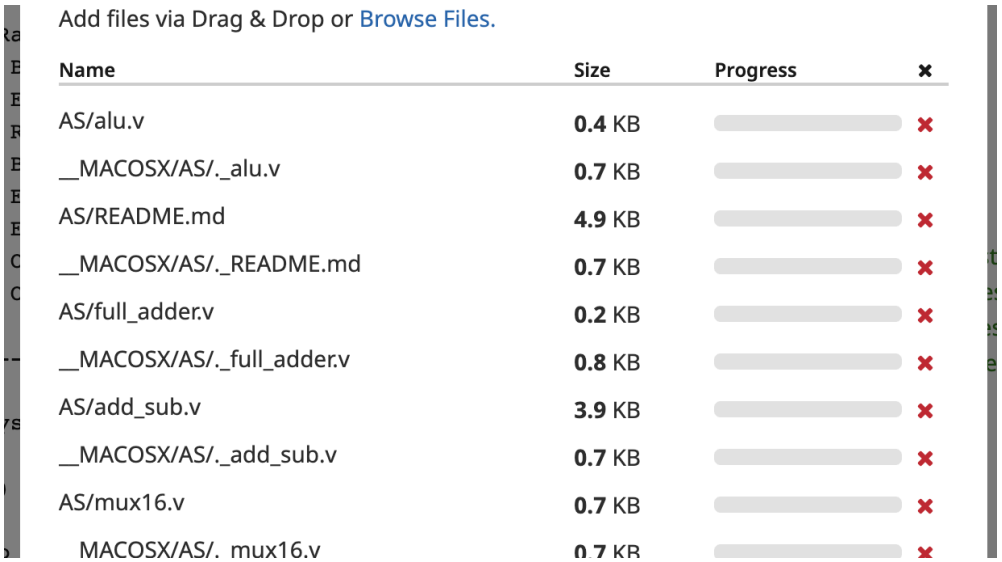
Writing Code

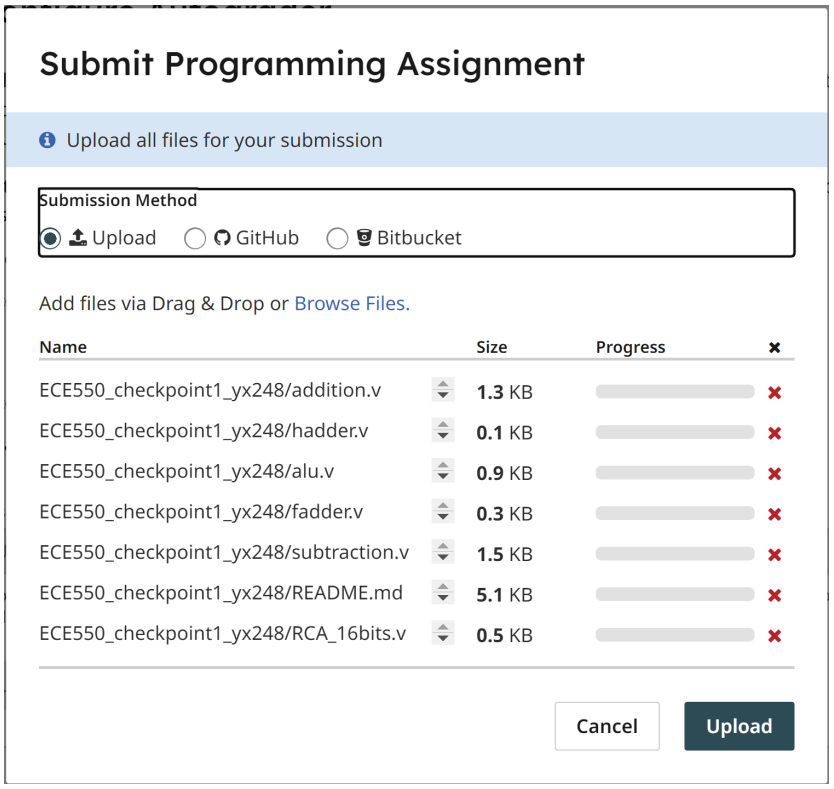
- Keep all of your source files in the top-level directory (don't create subfolders for your source code; generated files are fine)
- Make sure you structure your code so that alu.v is the top-level entity and it contains the provided alu interface
- **Please do not include testbench in your submission.**
- 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
    - Projects 1,2, and 3 are individual assignments
    - In Projects 4 and 5, all students should work in groups of two.
        - You will be working with the same team members for two projects unless there are major issues.
- Use Gradescope to submit your code and a README.md file. You should submit **one .zip file** containing **all necessary modules** for your project. Including *.qpf, *.qsf, *.qws files are fine. Please do not include more subfolders in your .zip file, or you may not get grades.

- The interface when submitting files may look like the attached pictures. (The first picture is for Mac. The second picture is for Windows.)





- A README.md (written in markdown, Github flavor) should includes
  - Your 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
- Make sure you get 'SUCCESS' for the design compilation check, address any issue prompted, and resubmit. Or you may not get the grade.

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.**