Project Checkpoint 3

Regfile

Logistics

This is the third Project Checkpoint for our processor. We will post clarifications, updates, etc. on Sakai and Ed.

- Due: **Friday, October 14, 2022,** by **11:59 PM** **(Duke Time)**
  - Late policy can be found on the course webpage/syllabus

Introduction

Design and simulate a **register file** using Verilog. You must support:

- 2 read ports
- 1 write port
- 32 registers (registers are 32-bits wide)

Module Interface

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

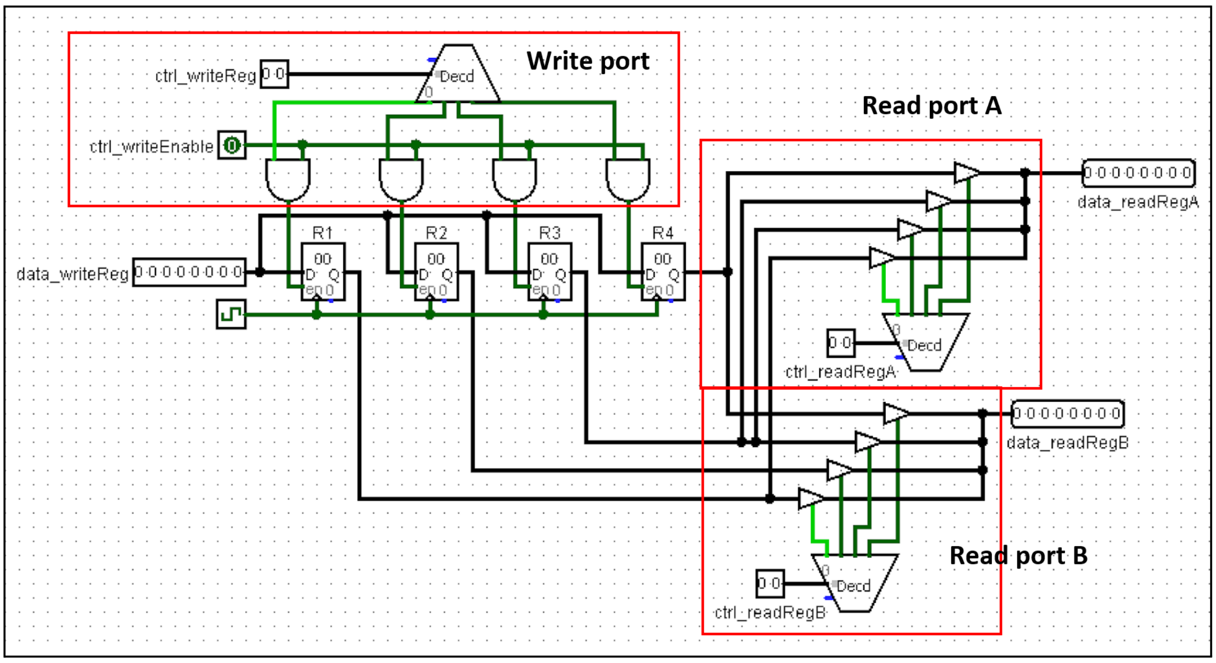
Your module must use the following interface (n.b. it is a template provided in regfile.v):

```
module regfile(
      clock, ctrl_writeEnable, ctrl_reset, ctrl_writeReg,
      ctrl_readRegA, ctrl_readRegB, data_writeReg, data_readRegA,
      data_readRegB
);

   input clock, ctrl_writeEnable, ctrl_reset;
   input [4:0] ctrl_writeReg, ctrl_readRegA, ctrl_readRegB;
   input [31:0] data_writeReg;

   output [31:0] data_readRegA, data_readRegB;
endmodule
```

Background

A register file is a series of individual registers containing key information in a CPU. The register file allows for two essential actions: reading register values and writing values to registers. This is accomplished by **ports**. A **read port** takes in data from all of the registers in the register file and outputs only the data (in this case, `data_readRegA` or `data_readRegB`) from the desired register, as designated by control bits (`ctrl_readRegA, ctrl_readRegB`). A **write port** uses similar control bits (`ctrl_writeReg`) to determine which register to write data (`data_writeReg`) to.

Below is an example of a register file laid out in Logisim. Keep in mind that this example only contains 4 8-bit registers, while your module must contain 32 32-bit registers.



Note: the read ports above contain **tristate buffers**. These are common in read ports and can act as a faster mux (see the tristate buffer sections of this article for more information). The Verilog equivalent of such an element is:

`assign buffer_output = buffer_select ? output if true : 1’bz;`

This is a form of the ternary operator and **is allowed** in this project.

Permitted and Banned Verilog

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

No "megafunctions."

- *Tip: think about whether your codes can specify only one design!*

Use structural Verilog like:

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

Not allowed to use SystemVerilog or syntactic sugar like:

- `+, -, *, /, %, **, ==, >=, &&, ||, !, <<, <<<, etc`
- ***`if`***`,` ***`else`***`, and` ***`case`***` statements,` ***`for`***` loop, etc`

except in constructing a DFFE (i.e., you can use whatever you need to construct a DFFE). Please name the DFFE module file 'dffe.v' to allow the style checker to bypass your DFFE implementation.

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

- **`Bitwise not(~)`**
- **`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
- **`generate if, generate for,`** and/or **`genvar`**
    - It could reduce the repeated lines but maintain the structural design
    - Any expression to specify the range, e.g., a[(i+24)%7]

Other Specifications

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

- Your design must function with no longer than a 20ns clock period (i.e., it must be able to be clocked as fast as 50 MHz)
- Register 0 must always read as 0 (no matter what is written to it, it will output 0)

Grading

Submitted designs will be tested by a grading testbench. The total score is 100 points, and 0.2 points will be deducted per failed test case. Please submit your regrading request on Gradescope within one week after the grade is published.

Submission Instructions

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

Writing Code

- Keep all of your source files in the top-level directory.
- Make sure you structure your code so that regfile.v is the top-level entity and contains the provided interface.
- 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 main to implement your projects and merge changes back into main 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).
    - Modify .gitignore at your own risk.

Submission Requirements

- When using **Gradescope** to submit your design, please submit **one .zip file** and the file should include your code and a README.md file. For **Github** submission, click 'connect GitHub', link your account, and select the correct repo and branch you want to submit.
- The submitted codes should contain **all necessary *.v modules** to execute your regfile. The autograder will read and examine all .v files in the .zip file; therefore, you may be able to include subfolders but you should be aware that if you submit unnecessary .v files it could cause compile errors.
- **Make sure the name of all testbench files ends with '_tb.v'**; otherwise, it will be involved in the style check and negatively affect your submission grade.
- A README.md (written in markdown, Github flavor) should include
    - Your name and netID,
    - A text description of your design implementation (e.g., "I used X,Y,Z to ..."),
    - If there are bugs or issues, descriptions of what they are and what you think caused them.

Resources

We have provided you with the regfile.v you can start working on, a sample testbench titled regfile_tb.v, and a dffe.v for you to refer to. They should help you construct and test your regfile and also write testbenches in the future. However, the testbench used for grading will be more extensive than the one presented here. Passing the included testbench does not ensure any points.