# Project Checkpoint 2

Regfile

## Logistics

This is the second Project Checkpoint for our processor.

- Due: **Tuesday, September 24, 2019** by **7:59:59 PM** exactly by Duke's timezone. One second late is considered late. However, all assignments have an automatic four hour "grace period," during which you can still submit your assignment without a penalty (11:59:59 PM). However, your assignment is still considered late.
- Late Policy: no submissions accepted without an official extension granted by the Head TA, the Professor, or the Professor in conjunction with your Dean

## 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)

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 cannot receive a score.*

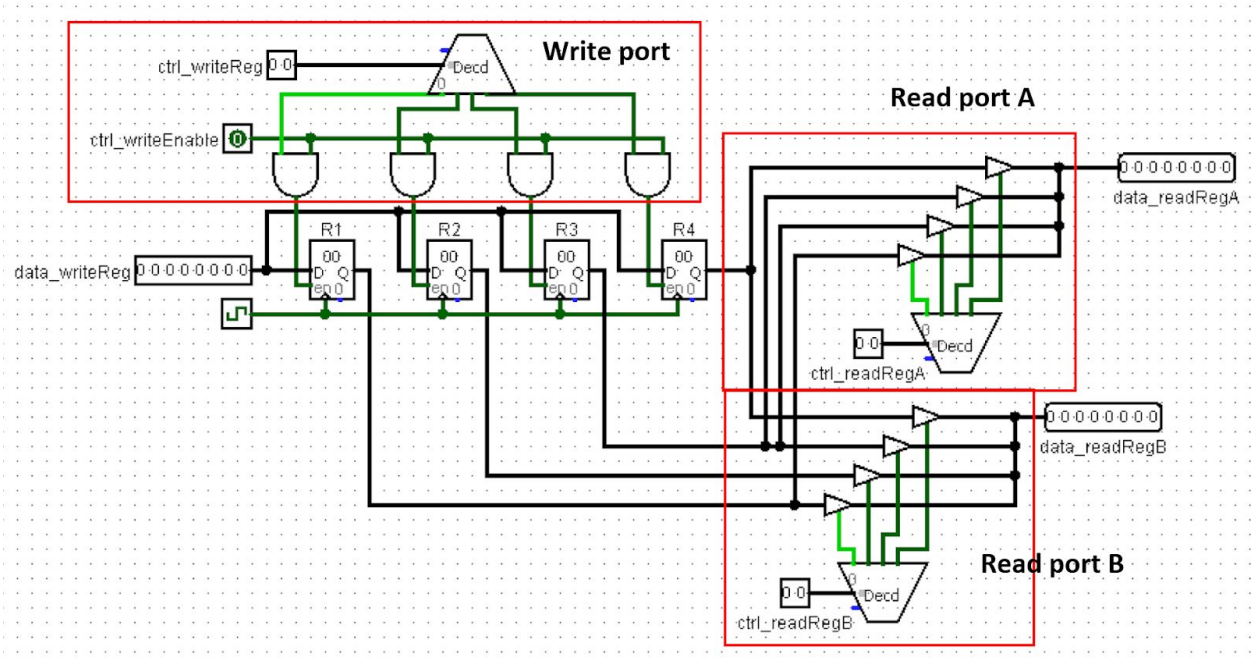
Your module must use the following interface (n.b. it is template provided to you in regfile.v):

```
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.

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




**Note**: the read ports above contain elements called **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 (see **Banned Verilog**).

## 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;` (Note: `&`, `|`, `*`, `/`, `^`, etc. are banned)
- `==, >=, =<, 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

## 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)
- Module names and file names must be the same. Otherwise, the autograder will not compile the submission

## Submission Instructions

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

### Setup

- Create a Github account
- Associate your netid@duke.edu email address with your Github account
- Sign into http://duke.ag350project.com to access your project repository
- Download git
    - Windows: find Github Desktop or use git bash

    - Mac: you'll have to install Xcode; you can also use Github Desktop
    - Linux: you should have git installed already, but you can google around to find the correct installation instructions for your operating system

### Writing Code

- Use git to manage your codebase
- Branch off of master and merge changes back in when you're sure everything works
- 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; each module must be in a filename that matches, e.g. module_foo must be in module_foo.v
    - The autograder will add a tests directory, so please don't put any files in this directory (the autograder will delete them)
    - 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 regfile.v is the top-level entity and it contains the provided regfile 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...")
    - A description of how fast you estimate your register file can be clocked
    - If there are bugs or issues, descriptions of what they are and what you think caused them

### Submitting for Grading

- Push to Github in your repository

## Grading

Complete, working designs will be tested for a grade using a test bench in a Quartus 16.0/Modelsim environment. **Caution**: other tools/environments like iverilog behave differently. If your code doesn't work in our environment, you may receive a 0.

## Resources

If you followed the setup instructions correctly, your repository has a sample testbench titled regfile_tb.v. This should help you test your regfile 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 any points.