Cpr E 281 LAB 12
ELECTRICAL AND COMPUTER ENGINEERING
IOWA STATE UNIVERSITY

Registers and Register Files

PRELAB!

Read the entire lab and **complete** the prelab questions (Q1-Q4) on the report template and submit your completed questions on Canvas **before** your lab time. You will submit this report again once you have completed the lab.

1.0 Objectives

In this lab you will create a register file to store data. Once the data has been stored, further data processing may be used to conduct operations based on the stored data. Re-read the lecture slides on registers and register files and complete the prelab before you come to the lab.

2.0 Setup

Begin by extracting the lab file to **U:\CPRE281\Lab12**. This folder includes skeleton code to get you started.

A register file is a series of interconnected parallel-access registers. Here, we will use a Block Design File to create the one-bit parallel-access register, then use Verilog to create larger registers to suit our needs.

While it is entirely possible to create a register file using only Schematic Captures or with only Verilog, we shall employ these methods together to demonstrate how Block Diagram Files may be used in a Verilog file and vice versa.

3.0 Parallel-Access 4-bit Register

3.1 1-bit register

Create a new project with a new schematic capture. Give this project the name **register**. Design the one-bit parallel access register with inputs **LOAD**, **IN**, **CLRN**, and **CLOCK**, and with output **OUT**. A sample diagram is shown below in Figure 1, but **you should also include the CLRN connection to your schematic and you should set the PRN to high (a.k.a VCC)**. Use a D flip-flop as memory and use the included mux2to1 Verilog file to choose the data source for the D flip-flop. Save this file as **register.bdf**.

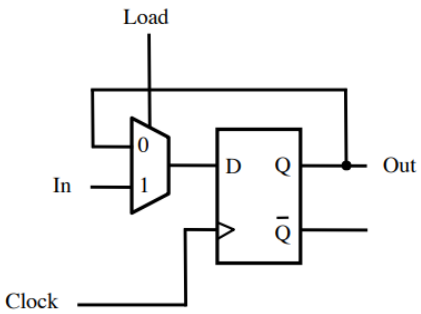

Figure 1: One-bit parallel access register.

When you have finished recreating the register in schematic capture, simulate this register in ModelSim by forcing the inputs. Observe the operation of the register as follows:
To change the output of the register, you must first force **CLRN** to 0 for a period of time. Then, the **CLOCK** signal must produce a positive edge and **LOAD** must be 1. Once these requirements have been met, the output **OUT** will change to match the input **IN**. Fill in the characteristic table in your report. Note: there is no DO file for this 1-bit register.

After you have verified the proper operation of your one-bit register, create a symbol for your register. You will use multiple copies of this one-bit register to make much more elaborate register circuitry.

3.2 4-bit register
Make a new .bdf file, name it **reg4b.bdf**, and then verify that the files **register.bdf** and **register.bsf** are in this project folder. Make this new **.bdf** file the top-level entity for your project. This can be done by either pressing Ctrl-Shift-J on the reg4b file or by selecting **Project -> Set as Top-Level Entity** from the Quartus menu.

Next, you will create a 4-bit register. Place four copies of the 1-bit **register** in your circuit schematic file. Connect all of the CLOCK inputs to the same input pin **CLK**. Similarly, connect all of the LOAD inputs to the same input pin **LD** and all of the CLRN inputs to the same input pin **CLRN**.

The input and output connections will be made as a bus. Add one input pin for the data input and name it **IN[3..0]**. The same method will be used for the output; name the output pin **OUT[3..0]**.

Connect the bus wires individually to each **register** as shown below, in Figure 2. Notice that each wire connecting to the IN bus and OUT bus connection is given the same name with a corresponding index (IN[3], OUT[3], IN[2], OUT[2], etc.). Make sure that the indexing you use is consistent; i.e., if a register receives **IN[3]**, it should also produce **OUT[3]**.

Once this circuit is complete, you will have four input pins (**CLK**, **LD**, **CLRN**, and **IN**[3..0]) and one output pin (**OUT**[3..0]). This will be the register in the register file that you will create in the next step.

Use ModelSim and the **reg4b.do** DO file to verify that your circuit works. Include a screenshot of your reg4b.bdf file and your waveform.

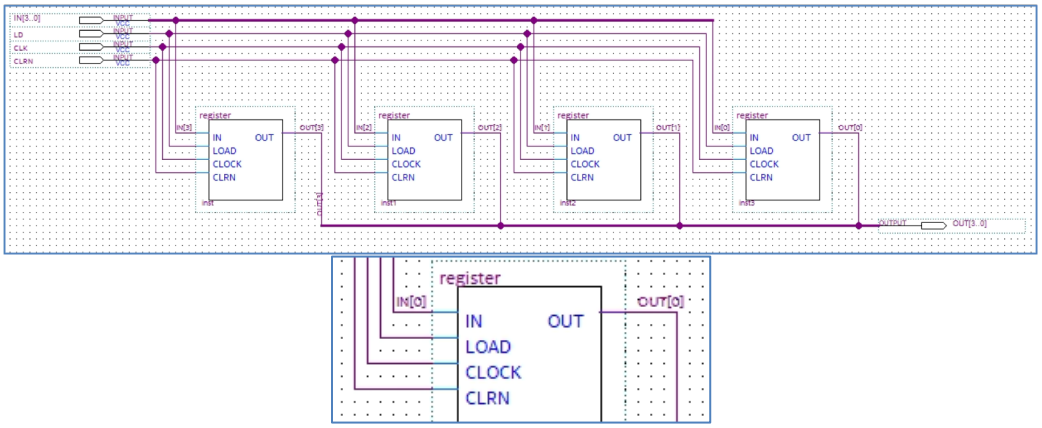

Figure 2: Four-bit parallel access register.

4.0 Register File

Now, it is time to create a register file, which is the next component in the hierarchy that you will create for this lab. The register file will have one write port, two read ports, and data will be stored in eight 4-bit registers. Therefore, you need to start by making eight copies of the four-bit register that you created in Part 3.0.

In addition to the eight registers, the register file also includes the controlling components that specify which register will be read and which register will be written. In order to read from the registers, you will use a multiplexer that will select which register output will become the output of the entire register file. To write to the registers, you will use a decoder. Since there are eight registers, you will need an 8-to-1 multiplexer and a 3-to-8 decoder.

The ports of the register file are as follows:
DATAP: First output of 4-bit data from the register file.
DATAQ: Second output of 4-bit data from the register file.
RP: 3-bit Read address that specifies which register will send its output through DATAP.
RQ: 3-bit Read address that specifies which register will send its output through DATAQ.
WA: 3-bit Write address that indicates which register will update its data.
LD_DATA: 4-bit data that will be written into the register specified by the address in WA.
WR: When this is zero, no register will update its data. When this is one, the register specified by WA will update its value with the value in LD_DATA.
CLK: A typical clock connection.

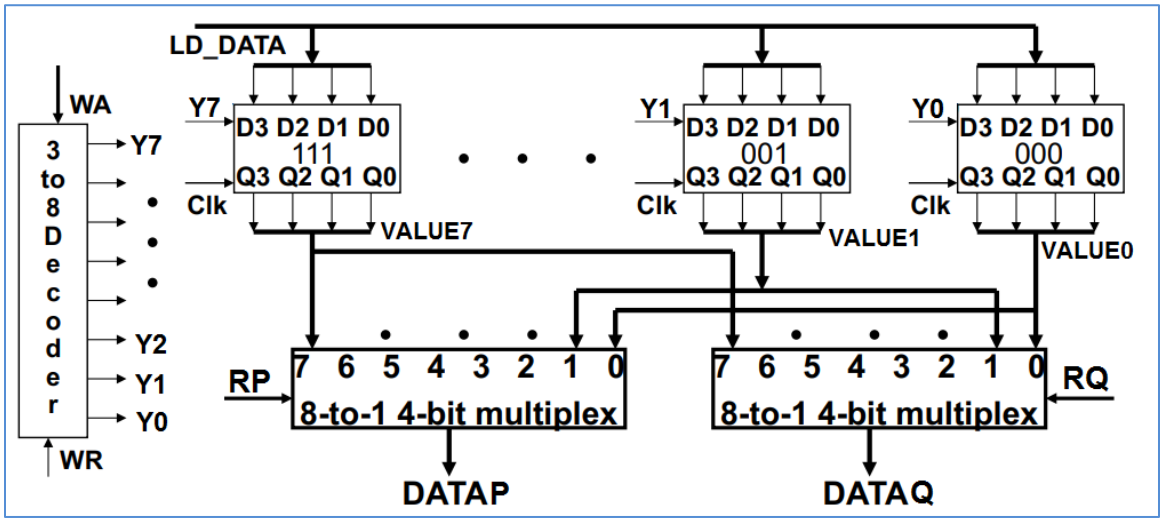

Figure 3: Rough schematic for the register file.

Start a new project and name it **regfile**. Open a new **Verilog** file and name it **regfile.v**. This file will contain the required components to implement a register file. See prelab question Q4 for skeleton code. The following steps outline the components needed to provide functionality to the register file. These components will include copies of the 4-bit register that you created in the previous step, so be sure to bring the **register**, **reg4b,** and **mux2to1** files into this new folder.

4.1 Multiplexer

The multiplexer that will read from the register file will receive all of the outputs from each **reg4b** that exists within the regfile. Each of these outputs is a four-bit value and the multiplexer must then select one of these four-bit values to output. What you will create for this register file is a 4-bit 8-to-1 multiplexer. You will create this 4-bit 8-to-1 multiplexer in Verilog similarly to how you created the 1-bit 4-to-1 multiplexer in Lab 08. Make this multiplexer in a separate Verilog file named **Mux8_4b**. See prelab question Q2 for skeleton code. You will bring this file together with the other components in section 4.4.

Use ModelSim to verify your circuit and include your circuit and waveform on your report.

Note 1: Intermediate multiplexer values are four-bit connections and should be declared as such. Although it is certainly possible to create the multiplexer without any intermediate expressions, it may be beneficial to include them. A four-bit intermediate connection can be declared as **wire [3:0] X**; this 4-bit wire X can be used to connect an output from a 2-to-1 multiplexer to another 2-to-1 multiplexer's input. There are other declarations that we need to do in this lab; see section 4.5 for more details.

Note 2: Notice that the output of each multiplexer is a single 4-bit value. The individual bits from the output bus can be referenced in Verilog using brackets. For instance, the individual bits from an output bus **DATAP** can be referenced as **DATAP[0], DATAP[1], DATAP[2],** and **DATAP[3]**. This will be used when we bring our components together in 4.4.

4.2 Decoder

The decoder will assert one of the LOAD lines for the eight registers, depending on the value of the write address. These LOAD lines will specify which register's value will be updated on the next clock edge. Also, add an ENABLE line to this decoder to allow for the opportunity to maintain all register values unchanged. To accomplish this, create a 3-to-8 decoder with ENABLE in Verilog. This, again, will be in a separate file named **Decoder3to8**. See the prelab for skeleton code.

4.3 Register Collection

You can create duplicate objects from the code files that you've already used. Similar to adding a symbol to a schematic file and wiring it.

```
Symbol_name symbol_inst(.Symbol_input(in_wiring), .Symbol_output(out_wiring))
reg4b my_reg4b(.IN(LD_DATA), .LD(Y[0]), .CLK(CLK), .OUT(R0), .CLRN(CLRN))
```

This will create one of the eight 4-bit registers from Part 3.0 that you will need to make the register file. You will have to specify the remaining connections to the other **reg4b** entities.

4.4 Bringing Everything Together

Now, you are ready to create the register file in Verilog. Start by declaring the module inputs and outputs as follows:

```
module regfile(DATAP,DATAQ,RP,RQ,WA,LD_DATA,WR,CLRN,CLK);
       // address and control port
       input [2:0] RP, RQ, WA;
       input WR, CLRN, CLK;
       // input data port
       input [3:0] LD_DATA;
       // output data ports
       output [3:0] DATAP,DATAQ;
       wire [3:0] VALUE0, VALUE1, VALUE2, VALUE3, VALUE4, VALUE5, VALUE6,
VALUE7;
       wire [7:0] Y; //decoder ouput

       <<< insert code here >>>

endmodule
```

The bus width is specified with its highest and lowest index as part of the declaration **input [3:0]**. This declares an input connection that has a width of four bits. Each bit can be accessed individually using the indices **[0], [1], [2],** and **[3]**. This will also be shown below.

The decoder can be added in the same way that you added the registers in part 4.3. A separate decoder file can be integrated with the register file module like this:

```
Decoder3to8 my_decoder(.EN(WR), .W(WA), .Y(Y));
```

The inputs to the decoder are also inputs to the register file. They determine which register, if any, will update its value. The outputs Y0 to Y7 are intermediate wires in Verilog that connect to the register **LOAD** connections as follows:

```
reg4b my_reg0(.IN(LD_DATA), .LD(Y[0]), .CLK(CLK), .OUT(VALUE0), .CLRN(CLRN))
```

This connects the decoder to register 0. The register's load connection is connected directly to the **Y0** output of the decoder. The output of the register is placed on the connection labeled **VALUE0**; this output will also become the output of the register file if address 0 is specified as either of the two read addresses. The data, clock, and clear inputs are inputs to the entire register file and will serve the same purpose for all registers. The remaining seven registers will be connected in the same way, except they will receive the corresponding Y output from the decoder and will output to the appropriate VALUE connection.

NOTICE: The multiplexer output is a 4-bit intermediate wired connection and must be declared as such. See section 4.5 for more information.

Once all of the registers have been placed in the file, we can use the multiplexer to select from one of the eight register outputs (**VALUE0, VALUE1, VALUE2, …**). Since our register file will have two read ports (**DATAP** and **DATAQ**), we will use two multiplexers to decide the output on each port. For instance, the output DATAP will be decided by a multiplexer that processes the values based on the read address in RP. In other words,

```
Mux8_4b my_muxP(.S(RP), .W0(VALUE0),
    .W1(VALUE1), .W2(VALUE2), .W3(VALUE3), .W4(VALUE4), .W5(VALUE5),
    .W6(VALUE6), .W7(VALUE7), .F(DATAP))
```

A similar result will be done for read address in RQ.

4.5 Wire Declarations

Wire declarations are also necessary. Previously, we did not have to declare intermediate wired connections in Verilog, since all intermediate values were just one bit in width. In this case, the output of the register file is a 4-bit value. If this output were left undeclared, then the compiler would assume this wire to be only one bit wide, as done before, which will create errors when these wires are used in place of the required four-bit connections. To avoid this problem, we will declare all of our intermediate buses as four-bit wires, as shown below.

```
// wire declarations
wire [3:0] VALUE0, VALUE1, VALUE2, VALUE3, VALUE4, VALUE5, VALUE6, VALUE7;
wire [7:0] Y;
```

These values are the outputs of the register file and the outputs of the multiplexer. They are all four-bit values that are neither a module input nor a module output. These declarations should be placed in the register file module with the declarations for the input and output ports.

Your circuit should now be able to store eight 4-bit numbers. Using ModelSim and the regfile.do DO file, test your circuit and once you understand how it works, demonstrate your result to the TA. Make sure include a screenshot of your code and waveform for the register file.

NOTICE: Make sure to add and compile all of the previous lab12 components in ModelSim. Otherwise, you may get compile errors.

5.0 Interaction of the Register File with Other Components

Finally, you will interface the newly created register file with other components that you have created in previous labs. Before closing the register file project, make a symbol for it. Now, create a new project and name this project **lab12_final**. Create a new schematic file and save it. Bring all of the previous lab12 components into the same folder as lab12_final: **add_sub, adder_4bit, busmux, Decoder3to8, FA, mux8_4b, register, reg4b, and mux2to1.**

Once these files have been successfully copied into the new folder, connect the components as shown in the figure below.

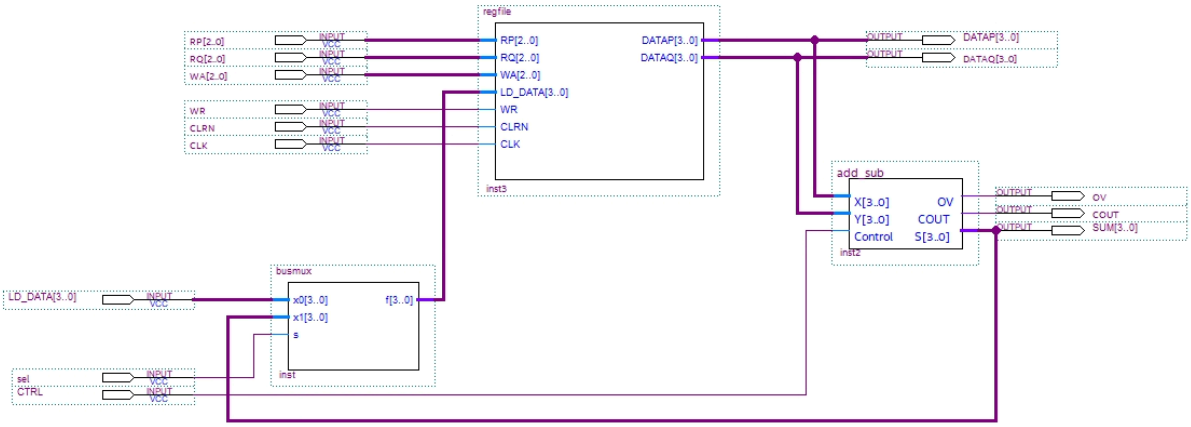

Figure 5: Register file with 4-bit adder connected.

Once your circuit has been connected, verify it in ModelSim using the lab12_final.do DO file. Include a screenshot of your BDF file and waveform in your report.

6.0 Complete

Congrats! You are done with labs for the semester. Please double check your report and submit it to Canvas. Good luck on finals and enjoy your break.