Lab 8 - Register File

What to hand in

1. Archive of Register File project (.qar) Named "Team#_lab8.qar"
2. .do file that test at least 3 different registers
3. .bmp for the .do file you create

Pre-lab

1. Draw gate diagram for a D-flip flop with a clock.
2. Using D-FFs (as a symbol), draw a 4-bit register with input D[3..0] and output Q[3..0]. Remember, each D-FF will handle only one bit. This is like the multiplexor from lab 6.
3. With the 4-bit register made in question 2, create a 4x4-bit register file (4 registers in total, each one can store 4 bits). You can create a symbol for the 4-bit register file you created above. You should only have input signals regS[1..0], regD[1..0], dataD[4..0], and clock to control the register file. There should be only one output, dataS[4..0]. regS will specify which register to output. regD will tell the register file which register to write the data to. Finally, dataD is the new data that will be stored to the register that regD specifies. Note: You should have the following devices in your register file: 2-4 decoder, and a 4-1 4-bit multiplexor. Look at the last image in this lab for help in placing the inputs and outputs.

Part 1

This weeks lab will deal with creating a register file. A register file is a location on a processor where the registers are stored/created. This is where registers R0-R15 of our processor will be stored. We implement this register file using D Flip Flops(D-FF's). As we know from class, DFF's are used to store values. This means that they are a quick, albeit small, type of memory. A single D-FF can hold 1 bit of memory.

Multiple bit D-FF's

As you know from the pre-lab, if we want a multi bit value to be stored using D-FF's, we simply create a D-FF for each bit of that value. This means for a 16 bit register, you will have 16 D-FF's to store its value. For a picture example, please refer to figure 2.

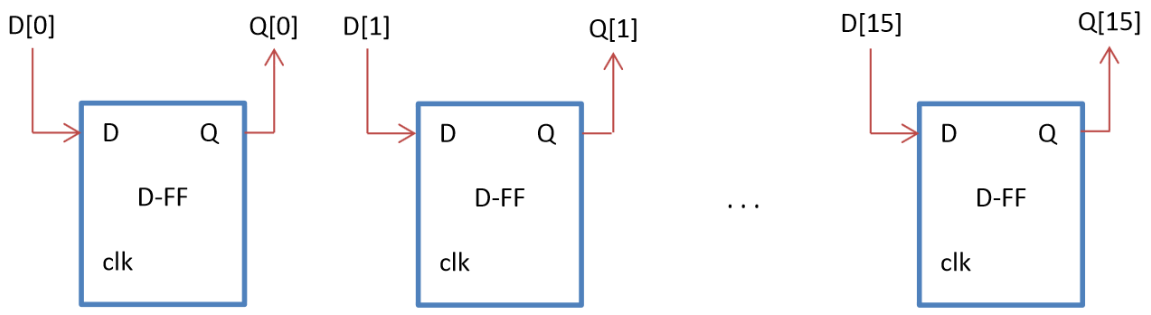

Figure 1: Multi bit D-FF

Now that we understand how multiple bit values are stored using D-FF's, we can begin to create our Register file

Components that the Register file requires

For this lab, you will need 16-bit registers (registers are made of D-FF's), a 4-16 decoder, and two 16-1 multiplexor's. In order for this lab to be reasonable, we have given you the VHDL code for each of these components. Please go to Piazza to download the VHDL files.

Using multiple components in VHDL

Since you have all the components needed to create the register file, all you need know is to know how to use multiple components in a single VHDL file. This is quite simple. All you need to do is declare the component one after another like you normally did. Please see the below code if there is any confusion (HINT: That is the code you need to include the components for your register file).

```
COMPONENT mux16
  PORT (
      d0,d1,d2,d3,d4,d5,d6,d7    :IN std_logic_vector(15 downto 0);
      d8,d9,dA,dB,dC,dD,dE,dF    :IN std_logic_vector(15 downto 0);
      sel                :IN std_logic_vector(3 downto 0);
      f                  :OUT std_logic_vector(15 downto 0)
  );
END COMPONENT;
COMPONENT reg16
  PORT(
    data                    :IN std_logic_vector(15 downto 0);
    enable,  reset,  Clock          :IN std_logic;
    output                  :OUT std_logic_vector(15 downto 0)
  );
END COMPONENT;
COMPONENT decoder16
  PORT(
    Sel    :IN std_logic_vector(3 downto 0);
    Output :OUT std_logic_vector(15 downto 0)
  );
END COMPONENT;
```

Inputs and Outputs of the register file

Now you need to know what kind of inputs and outputs this register file. Simply put, you will need:

1. Three 4-bit inputs to choose what register to look at/ write to
2. A 16 bit input data value that would be the value to update a register with
3. Three 1-bit inputs called clock, reset, and enable, that will determine the use of a register
4. And two 16-bit outputs that will contain the values of the register you chose

One more time, with a table:

INPUTS	OUTPUTS
Reset	DataS[15..0]
Enable	DataT[15..0]
Clock	
RegD[3..0]	
RegT[3..0]	
RegS[3..0]	
DataD[15..0]	

To further explain these inputs and output, let's look at each one separately. Reset will be an input control flag that will reset the values in the registers to 0 if Reset is 1. Enable will be an input control flag that turns on a register for writing. If enable is 0, then the value of the register will not change. Clock is an input control flag that is necessary because D-FF's can only be controlled by the rising edge of a clock, otherwise they would never change. RegD will be the input that goes to the decoder. This value will get broken into 16 different values, and these 16 different values will act as another enable to the register. **NOTE: You will want to AND each of these outputs with the enable input and that determined value will be the one that goes to the register**. RegS and RegT will be the selector inputs to the 16-1 multiplexor's. The other inputs to the 16-1 multiplexor's will be each Register. The outputs DataS and DataT will be the outputs from the 16-1 multiplexor's.

Task

For the task for this lab, you need to make a 16x16-bit register file. This means you will create a register file that has 16 registers and that each of these registers are 16-bits.

Actually creating the register in VHDL will not be too difficult. If you follow the upcoming step by step process, you should be able to get this done quickly.

1. First you want to decode your RegD.
2. Then you will want determine your enable signal. You will have a write enable and the decoded RegD bits that will determine whether or not to enable a register.
3. Next you will want to PORT MAP to your registers. The outputs of these registers should be 16-bit signals. You should have 15 different registers that are PORT MAPPED to (reg0 will always be zero, so there is no reason the PORT MAP an extra register). To make reg0(which should be a signal like the rest of them) always zero, place the following line of code somewhere in your VHDL file.

```
reg0 <= (OTHERS => '0'); --This assumes the register signals
    are name reg0 - reg15 and are 16-bits wide.
```

4. Finally, you will want to PORT MAP each register, r0-r15, to two Multiplexor's. The only difference between the two multiplexor's will be the selector input and the output. You should use RegS and RegD here.

Testing

As for testing, you should simply test to make sure you can read and write to each register. That means you should test reg1 first, and make sure that if you give it a value, that you can read that value in later clock cycle. Test each register for this kind of operation. The value you write to the registers does not matter, as long as it is not zero.

Picture of what the Register File looks like as a Block Diagram

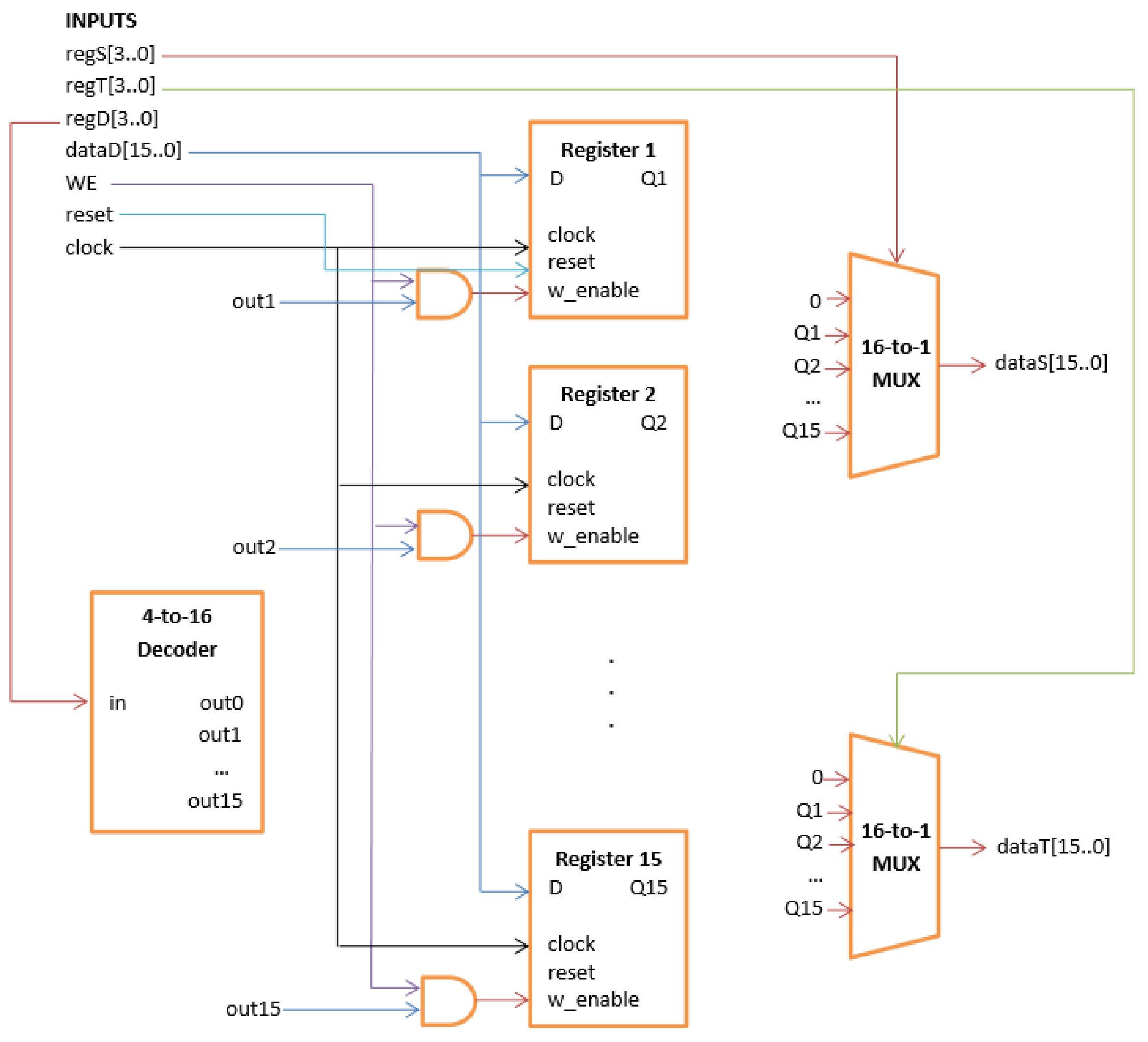

Figure 2: Multi bit D-FF