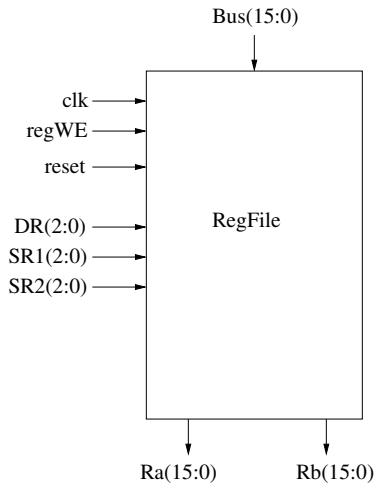
Lab 6 – Register File

1 Objective

Use D Flip-Flop as the basis for a 1-bit memory cell. Use hierarchical design to construct a 16-bit by 8-word 3-port register file. Simulation and testing will be required to verify circuit operation. Learn how to address registers, to write to them and to read from them.

2 Introduction

In this lab you will design a 16-bit by 8-word register file from the ground up. Note that a register file is just a multi-ported memory. Remember from class that you have learned that a memory is something that can be addressed for writing to and reading from. A memory with n address lines will contain 2^n locations. For example, a memory with 4 address lines will contain 16 locations.



Independent from this is the consideration of what is stored in each location. A memory could be designed to hold just 1 bit per location. Thus, a 16x1 memory would have 4 address lines, a single data-in line, and a single data-out line. On the other hand, a memory could be designed to hold 32 bits per location. Such a memory would have a 32-bit data-in port and a 32-bit data-out port.

Further, the memory you are to make will be dual-ported. That is, it contains two different sets of address lines. One set controls which location is written to using the data-in signals and one set controls which location is read from onto the data-out signals.

Thus, there will be three sections to your design. The first section will accomplish writing into the register file. The register file itself will just be a collection of D-type flip flops. Finally, there is the third section which will do the reading from the register file.

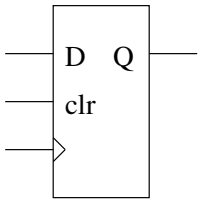
The key point is that the writing and reading are independent. Thus, on a given clock cycle you can, for example, write to register 2 but be reading from register 0.

In this lab, you will design a 8x16 register file. It will have 3 sets of address lines of 3-bits each (one set is the 'write' address and the other two sets are the 'read' addresses), 16

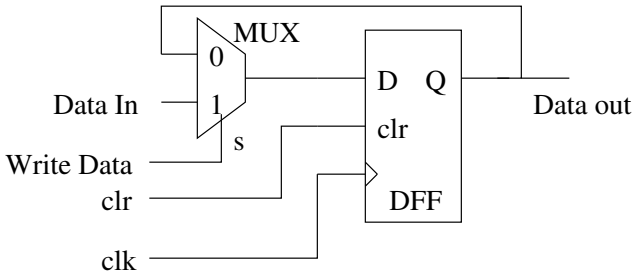
data-in lines, and two sets of 16 bit data-out lines. Finally, it will need a write enable signal to tell the selected register when to write new data.

We will begin D flip-flop with just a clear from lab # 5 (e.g., DFF_C.VHD).

The symbol is as follows:



We need to be able to control when the flip flops load a new value, so we will add a 2:1 multiplexer as follows:

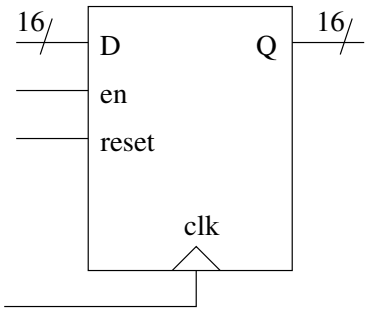


NOTE: The signal which selects the Data In is called the Write signal because when it is asserted, the Data In is written to the flip flop on the falling clock edge.

2.1 Preparation

1. Design your own DFF with an enable (en). Use the approach described above. It should have single-bit inputs d, clk, reset and en. The output will be q.

2. Use this new single bit enabled DFF to design a 16bit wide register. It should have single bit inputs clk, reset and en. It will have a 16bit input d(15 downto 0) and a 16bit output q(15 downto 0). It should look like:



3. Next Design a 3 to 8 decoder. You can do this by either using your 2 to 4 decoder as we did in the homework or build one from scratch.

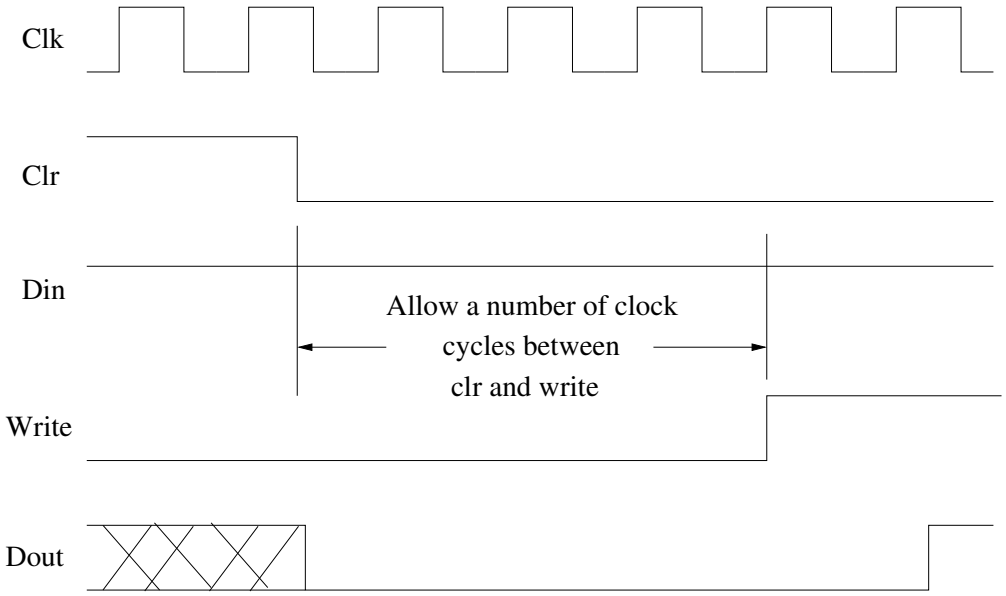
4. Design a 8 to 1 MUX. This can be done with some of the other MUXes we have already designed (4to1, 2to1, etc.)

5. Using your 8 to 1 MUX design a 128 to 8 MUX. You definitely will need to use vectors to do this easily and you might want to review the lecture on the generate statement (lab4 lecture).

2.2 Procedure

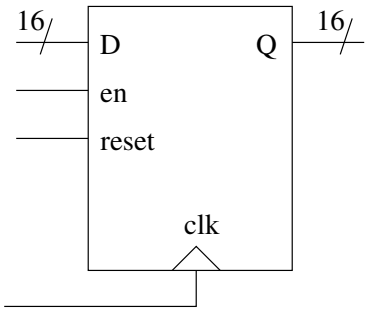
1. Build the 1-bit register from two circuit elements: a 2 to 1 mux and a D-type flip flop with a rising-edge triggered clock and a synchronous reset. Start with the DFF_C.VHD from lab 5 and reuse your 2 to 1 MUX.

   Build your 1-bit register from the VHDL files and simulate it to verify that it is correct. Don't forget to clear the register at the beginning of the simulation. The output waveform for this testing the 1-bit register should resemble the following:



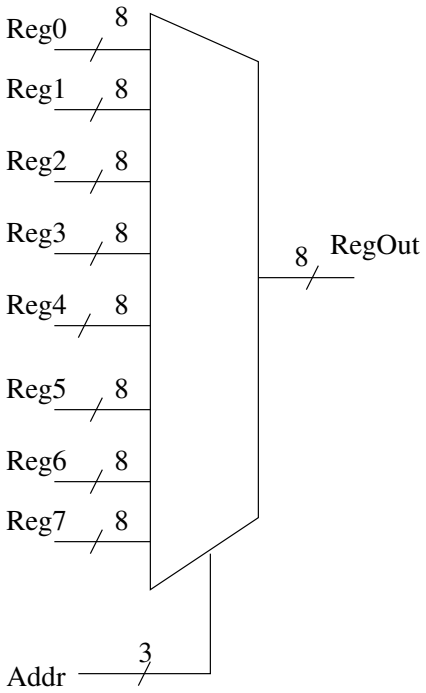
2. Using the circuit for the one bit register and referring to the above preparation, construct a register word consisting of 16 bits. Simulate it to verify that it is correct. Using the generate statement (again lab4 lecture), will make this much easier to do.

   During simulation, you should force your 16 data inputs to a "1" and not change them. Start with the Write signal off and hold it there for a number of clock cycles. Then assert the Write signal and watch the input data pattern get written to the 16 data outputs. You also want to check that your bits are not getting swapped in the register. Verify this by forcing your 16 data inputs to 0x"A5A5". Again, watch the outputs of the register to ensure that the proper bits are coming out of your register. A symbol for the 16-bit register would look like the following:



3. Build the 128 to 16 multiplexer that you designed above. It is used to select the appropriate register output according to the read address select signals. Again, the generate statement will help aid in this (see lab4 lecture).

   The symbol for a 128 to 16 multiplexer would look like the following:



   Simulate it to verify that it is correct. For this simulation, you should force your 16 data inputs from each register to a distinct constant hex value. Each register should have a different hex value. Begin varying your read address bits. For each read address, you should see the hex value for that register come out.

4. Complete the 8 word by sixteen bit register file.

   Using the module for the 128 to 16 multiplexer and your 8 word register and referring to the lecture on register files (16), construct a complete register file. Make sure that it is a 3-port design (i.e. the two read and write addresses should be separate and independent).

5. Test Bench (a testing circuit)

   Using your new 16 bit by 8 word register file, construct a test bench to test it on the Xilinx Spartan 3 board. Your test bench should have:

4 data input bits:	Switches 7-4	– Make sure your bits are in the correct order! (SW 7 is the most significant bit)
4 data input bit indicators:	LEDs 7-4	– LED 7 is the most significant bit
2 write address input bits:	Switches 3-2	– SW 3 is the MSB
2 read address input bits:	Switches 1-0	– SW 1 is the MSB
4 register file output bits:	LEDs 3-0	– LED 3 is the MSB
1 write control signal:	Button 3	
1 clear control signal:	Button 0	
clock signal	System clock	

   The board-level test bench will not be able to see the whole word or input all of the data bits unless we get real creative. So, we will be just testing one of the read ports (Port A) and the write port and only the bottom (lsb) four bits of those registers. Be sure to test your design in simulation with a full testbench to ensure that all of the other locations, ports and bits are working correctly.

6. Implement the Design - **download** to the board for testing.

   Following the same steps as you did in the previous labs, Implement the design. Test the results of your design. Have a TA verify the operation of your working circuit on the Spartan3 board. (You can use the top-level files from lab5 to do this. Be sure to use and modify s3demo.vhd. You also need to use s3demo.ucf to map the inputs and outputs of the chip.) You will need to connect the switch and LEDs as described above.

   Pass-off after your test bench is working correctly.