CSCE 230L – Lab 8: Register File

March, 2013

Objectives

- Build your own 16x16bit register file that you can use in the project.

Useful References and Resources

Appendix A in the textbook

Lab8.zip file provides 4-16 Decoder and 16 bit DFF

What to Turn In

- Pre-lab questions (due at the beginning of lab; in paper)
  - All questions total to 20 points
- Files and questions from lab (submit on handin)
  - Project archive file (.qar)
  - Simulation results for each component

Answer all the questions in a single text file. For the project files, create a Quartus archive file (.qar).

Name (1 point):

Pre-Lab (19 points)

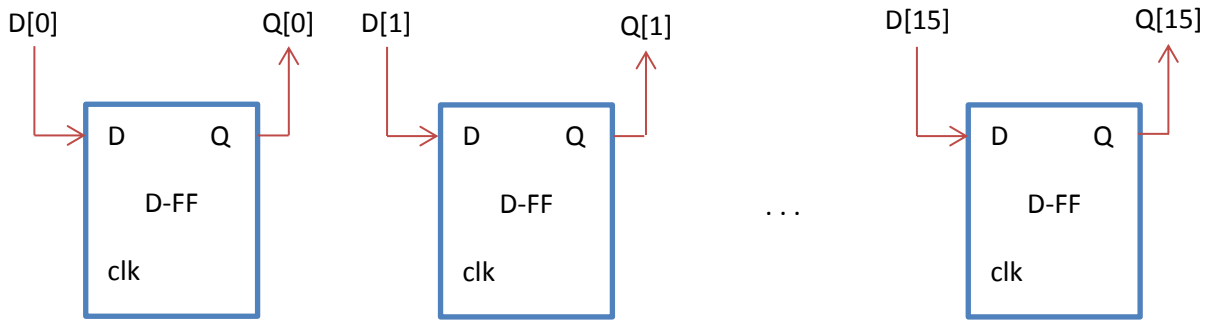
1. Draw gate diagram for a D-flip flop with a clock.

2. Using D-FFs (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.

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). Don't worry about write-enable or reset for this question. Have input signals regS[1..0], regD[1..0], dataD[4..0], and clock to control the register file. regS will tell the register file which register the output data will come from; regD will tell the register file which register the input data will store in; dataD is the new input data that the register file will store in one of its register. (you can take the last figure as reference)

Lab 1

We know a D flip flop can be used as a 1 bit register, so to accommodate the use of multiple bit systems, we need a register to be able to store more than 1 bit. To implement a multiple bit register using single bit flip-flops, we can use a flip flop for each bit and connect them as shown. Think of the end result as a 16-bit wide D flip-flop.



As you may have noticed/experienced last lab, the outputs don't always change exactly when the inputs change. This is because of propagation delays of the signals. A computer system overcomes this issue by using a clock: most components wait for the clock to be 1 or for the transition from 0 to 1 operate (synchronous). There are components and devices that operate independent of the clock (asynchronous). Since the project will use data words of 16 bits, we want to create a 16x16bit register file. To save us some time, we can use Quartus to create a 16-bit register, a 16-bit 16-to-1 multiplexer, and a 4-to-16 decoder.

16-bit register (16-bit wide D-FF) (also available on Blackboard for those using Quartus v11)

1. Go to Tools->"MegaWizard PlugIn Manager..."
2. Select "Create a new custom megafunction variation" then click next
3. Under Storage, select "LPM_FF" and specify a name for the file in your project folder and click next
4. Select 16 for the number of flip flops then click next
5. Under asynchronous inputs, check the "Clear" box. Then click next twice
6. In the summary, check the box for "Quartus II Symbol File" then click finish

16-bit wide 16-to-1 MUX

1. Go to Tools->"MegaWizard PlugIn Manager..."
2. Select "Create a new custom megafunction variation" then click next
3. Under Gates, select "LPM_MUX" and specify a name for the file in your project folder and click next
4. We want 16 data inputs and they should be 16 bits wide then click next twice
5. In the summary, check the box for "Quartus II Symbol File" then click finish

4-to-16 decoder

1. Go to Tools->"MegaWizard PlugIn Manager..."
2. Select "Create a new custom megafunction variation" then click next
3. Under Gates, select "LPM_DECODE" and specify a name for the file in your project folder and click next
4. Select the data width to be 4 then click next.
5. Add all of the “eq” outputs then click next.
6. Make sure “no” is selected for pipelining then click next twice.
7. In the summary, check the box for "Quartus II Symbol File" then click finish

Now we can add these just like any other symbol. They will be under the projects folder.

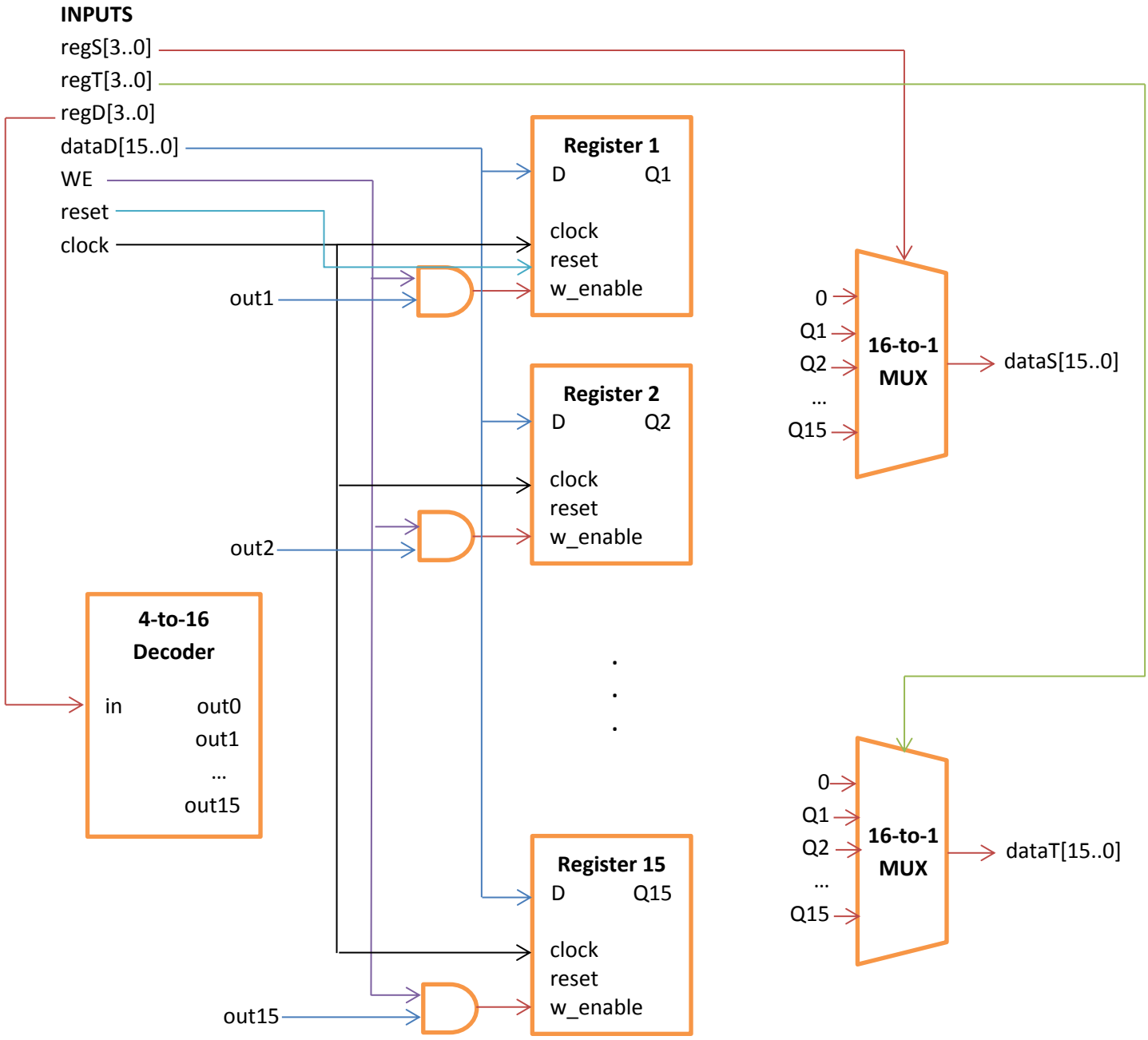
We have a number of inputs for the register file. Think of the instruction "add rd, rs, rt" where rd is the destination and rs and rt are the source registers. To do this instruction, the register file needs to be able to read two values at a time. So we have inputs:

regD[3..0] - address of destination register
regS[3..0] - address of source register
regT[3..0] - address of source register
dataD[15..0] - data to store in regD
clock – the system clock
WE - write enable for the registers (1: write to regD, 0: don’t write to regD)
reset - reset bit (reset all registers to zero when this is 1)

Most of the inputs should be what is expected. We need a write enable bit so that regD isn't written to when we don't want it to be. For example, when executing the instruction "ldw rs, (rt)", we can see that regD will always have some value and in this case we don't want regD to be written over, so WE should be 0. Since the 16-bit register won’t have a WE input, how can we prevent registers from being overwritten? Remember, the registers only operate when the clock is 1, so we can AND the WE signal with the clock. Now when the WE is 0, the clock will be 0, meaning the register will keep its value.

To make sure that register 0 always has the value of 0, create a constant 0 with the mega function wizard. Make sure you specify the constant value to be 16 bits wide.

Using the given picture as an example, create the 16x16-bit register file. Then create a vector waveform file to show that it works. You don’t need to test every possible input value. Check that you can write to and read from each register.


INPUTS
regS[3..0]
regT[3..0]
regD[3..0]
dataD[15..0]
WE
reset
clock
Register 1
D
Q1
clock
reset
w_enable
out1
Register 2
D
Q2
clock
reset
w_enable
out2
4-to-16
Decoder
in
out0
out1
...
out15
Register 15
D
Q15
clock
reset
w_enable
out15
0
Q1
Q2
...
Q15
16-to-1
MUX
dataS[15..0]
0
Q1
Q2
...
Q15
16-to-1
MUX
dataT[15..0]