Laboratory Exercise 4 – ECE241 Fall 2014

Latches, Flip-flops, and Registers

The purpose of this exercise is to investigate latches, flip-flops, and registers.

Preparation

You are required to do step 1 of Part I. You are also required to write and simulate Verilog code for Parts II and III. For marking by the teaching assistants, you will need to bring with you (pasted into your lab book), your schematic for Part I, your schematic for Part II, and your Verilog code and simulation output for Parts II and III. Please comment your simulation output by pointing out (on the timing diagram) what is being tested as the simulation proceeds.

In-lab Work

You are required to implement and test all of Parts I to III of the lab and demonstrate them to the teaching assistant.

Part I

Figure 1 shows the circuit for a gated D latch. In this part, you will build the gated D latch using the 7400 chips (as in Lab 1) and the protoboard (breadboard). Refer back to the Lab 1 handout for the specifications of the 7400 chips.

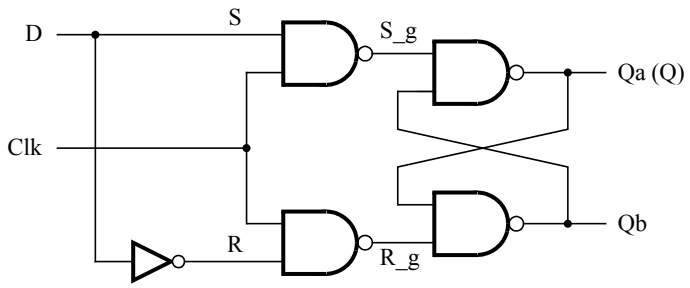

Figure 1: Circuit for a gated D latch.

Perform the following steps:

1. In your lab book, draw a schematic of the gated D latch using interconnected 7400-series chips. Don't forget to hook up the power and ground!
2. Build the gated D latch using the chips and protoboard. Use switches to control the clock and D input. Use lights to make Qa and Qb visible.
3. Study the behaviour of the latch for different D and clock settings.
4. Demonstrate your latch implementation to the TA.

Part II

Figure 2 shows a positive-edge-triggered flip-flop with several multiplexers. In this part of the lab, you will use eight instances of the circuit in Figure 2 to design a left/right 8-bit rotating register with parallel load shown in Figure 3. The *LoadLeft* input of all eight instances of the circuit in Figure 2 should be tied to a single rotating register input *RotateRight* because when you want to rotate the bits right, you have to load the bit to the left. The *loadn* input of all eight instances should be tied to a single rotating register input *ParallelLoadn*. The *clock* input of all eight instances should be tied to a single rotating register input *clock*. Create an 8-bit-wide rotating register input *DATA_IN*, whose individual wires *DATA_IN[7]* to *DATA_IN[0]* are tied to the *D* input of each instance of the circuit in Figure 2. Likewise, create an 8-bit-wide rotating register output *Q*, whose individual wires *Q[7]* to *Q[0]* are tied to the *Q* output of each instance of the circuit in Figure 2.

The remaining connections between the eight instances of the circuit in Figure 2 should realize the following behaviour:

1. When *ParallelLoadn = 0*, the value on *DATA_IN* is stored in the flip-flops on the next positive clock edge (i.e., parallel load behaviour).

2. When *ParallelLoadn = 1*, *RotateRight = 1* and *ASRight = 0* the bits of the register rotate to the right on each positive clock edge (notice the bits rotate to the right with wrap around):

   $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$
   $Q_0Q_7Q_6Q_5Q_4Q_3Q_2Q_1$
   $Q_1Q_0Q_7Q_6Q_5Q_4Q_3Q_2$
   ...

3. When *ParallelLoadn = 1*, *RotateRight = 1* and *ASRight = 1* the bits of the register rotate to the right on each positive clock edge but the most significant bit is replicated. This is called an *Arithmetic shift right*:

   $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$
   $Q_7Q_7Q_6Q_5Q_4Q_3Q_2Q_1$
   $Q_7Q_7Q_7Q_6Q_5Q_4Q_3Q_2$
   ...

4. When *ParallelLoadn = 1* and *RotateRight = 0*, the bits of the register rotate to the left on each positive clock edge. *ASRight* is ignored:

   $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$
   $Q_6Q_5Q_4Q_3Q_2Q_1Q_0Q_7$
   $Q_5Q_4Q_3Q_2Q_1Q_0Q_7Q_6$
   ...

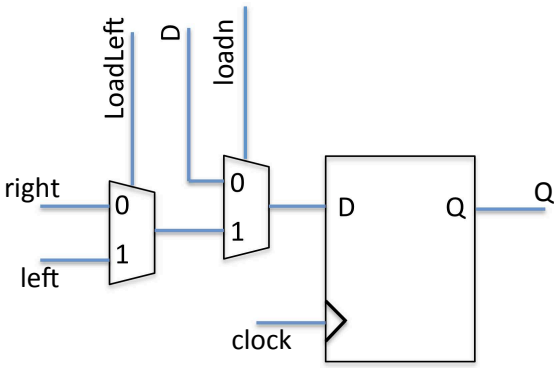

Figure 2: Sub-circuit for Part II.

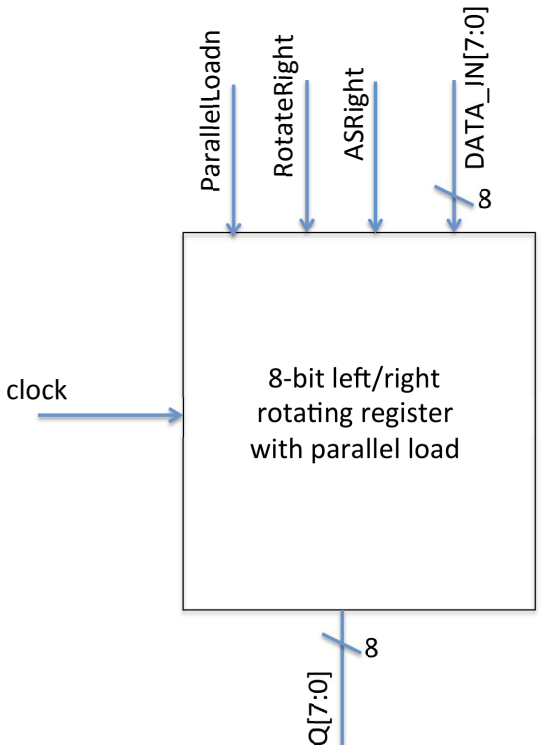

Figure 3: Top-level circuit for Part II.

Figure 3 shows the inputs and outputs of the top-level left/right rotating register circuit with parallel load, which will contain eight instances of the circuit in Figure 2.

Do the following steps:

1. Draw a schematic for the 8-bit rotating register with parallel load. Your schematic should contain eight instances of the circuit in Figure 2. Paste the schematic into your lab book. Label the signals on your schematic.
2. Create a new Quartus II project.
3. Write a Verilog module for the circuit in Figure 2.
4. Write a Verilog module for the rotating register with parallel load that instantiates eight instances of your Verilog module for Figure 2. This Verilog module should match with the schematic in your lab book. Use *SW[7:0]* as the inputs *DATA_IN[7:0]*. Use *SW[8]* as the *RotateRight* input, *SW[9]* as the *ASRight* input and *SW[10]* as the *ParallelLoadn* input. Use *KEY[0]* as the clock, but **read the important note below about switch bouncing.** The outputs *Q[7:0]* should be displayed on the red LEDs (*LEDR[7:0]*).
5. Include the Verilog code in your project.
6. Compile your Verilog code and simulate the design with QSim. In your simulation, you should use the parallel load to initialize the rotating register to 0xBE (hexadecimal) at the start of the simulation. Then, clock the register for several cycles to demonstrate rotation in the left and right directions.
7. Download your circuit into the Cyclone II FPGA on the DE2 board.
8. Test the functionality of your rotating register.

Note: If you run into bounce problems with *KEY_0* for your clock you are welcome to try using any of the keys. All mechanical switches, such as a push/toggle button, will often make contact several times due the electrical contacts bouncing. This happens quickly in human time, but not in electrical time. With a bouncing switch you can observe multiple high-frequency toggles making it difficult to create single clock edges. Although the DE2 keys/switches are supposed to be debounced it doesn't seem to work.

Part III

You will use the ALU you designed in Part IV of Lab 3 to build the circuit shown in Figure 4. The circuit contains an 8-bit register that drives the B input of the ALU. Design your register with an active-low synchronous reset. Observe that at each positive clock edge, the data on the ALU output is stored in the register, and as such, it becomes an operand in the next computation. This circuit can do a variety of computations, based on the "instruction" appearing on the 3-bit-wide *OPCODE* input, and where the result of a computation is stored in memory (the 8-bit register).

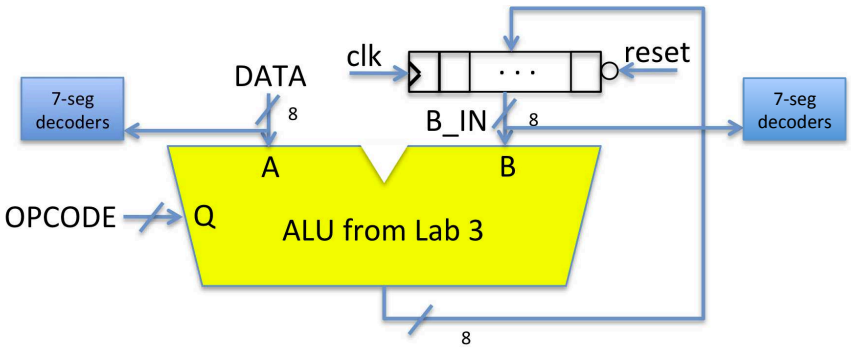

Figure 4: Circuit for Part III.

We wish to display the hexadecimal value of the 8-bit number *B_IN* on the two 7-segment displays, *HEX[1:0]*. Likewise, we wish to display the hexadecimal value of the 8-bit number *DATA* on the two 7-segment displays, *HEX[3:2]*. You will need to design a 7-segment decoder that displays the correct hexadecimal digit for a 4-bit binary input. Your design will instantiate four instances of your decoder.

Use *KEY[0]* for the clock input. Use *SW[11]* for the reset input. Use *SW[7:0]* for the *DATA* input. Use *SW[10:8]* for the *OPCODE* input.

1. Create a new Quartus II project which will be used to implement the desired circuit on the Altera DE2 board.
2. Write a Verilog module that provides the necessary functionality.
3. Include the Verilog file in your project and compile the circuit.
4. Simulate the circuit with QSim for a few cycles to ensure your circuit is working properly. You should use the reset input to clear the register to `0x00` (hexadecimal) at the beginning of the simulation.
5. Assign the pins on the FPGA to connect to the switches and 7-segment displays, as indicated in the User Manual for the DE2 board.
6. Recompile the circuit and download it into the FPGA chip.
7. Test the functionality of your design by toggling the switches, keys, and observing the output displays.
8. For those with time to spare, you can try this. This will not be marked. Increase the functionality of your ALU by adding shifting capability. Replace the register in Part III with the shifter from Part II. You will need to extend the *OPCODE* to add the shifter functions. Think about how the *reset* can be done.