Laboratory Exercise 4

Latches, Flip-flops, and Registers

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

Preparation Before the Lab

You are required to complete Parts I to III of the lab by writing and testing Verilog code and compiling it with Quartus II. Show your Verilog, simulations and schematics for Parts I to III to the teaching assistants. You must simulate your circuit with ModelSim (using reasonable test vectors using the format shown in the previous lab).

In-lab Work

You are required to implement and test all of Parts I to III of the lab. You need to demonstrate all parts to the teaching assistants.

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.

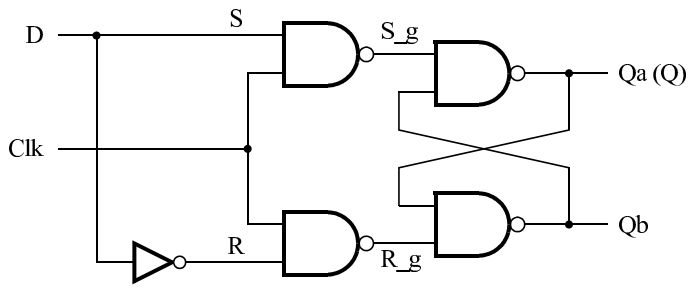

Figure 1: Circuit for a gated D latch.

As seen in class, ~~2~~ latches placed together form a flip flop. In Verilog, a ~~posedge~~ flip flop with an active low synchronous reset (meaning reset only happens when reset is 0 ~~and~~ on rising clock edge) is shown below:

```
always @(posedge clock)      // triggered every time clock rises
begin
   if (reset == 1'b0)        // when reset is 0 (note this is on every time clock rises)
      q <= 0;                // q is set to 0
   else                      // when reset is not 0
      q <= d;                // value of d passes through to output q
);
```

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

Using Lab 3 Part III, remove case 5 and add 3 more cases: a shift, multiply and hold current value. In this section, you will need to also implement a register at the output of the 8 to 1 multiplexer shown below. The output of this register will connect back to input B from your previous lab. Input A will now use SW_{3-0}.

```
always @(*)          // declare always block
begin
   case (select)     // start case statement
   begin
      0: Addition using Lab 3 Part II of this Lab
      1: Addition using '+' operator
      2: Sign extend value B to 8 bits
      3: Find if at least 1 of the 8 bits is 1 using a single OR operation
      4: Find it all of the 8 bits are 1 using a single AND operation
      5: Left shift B by A
      6: Multiply B by A using '*' operator
      7: Hold current values
      default: ...   // default case
   end
end
```

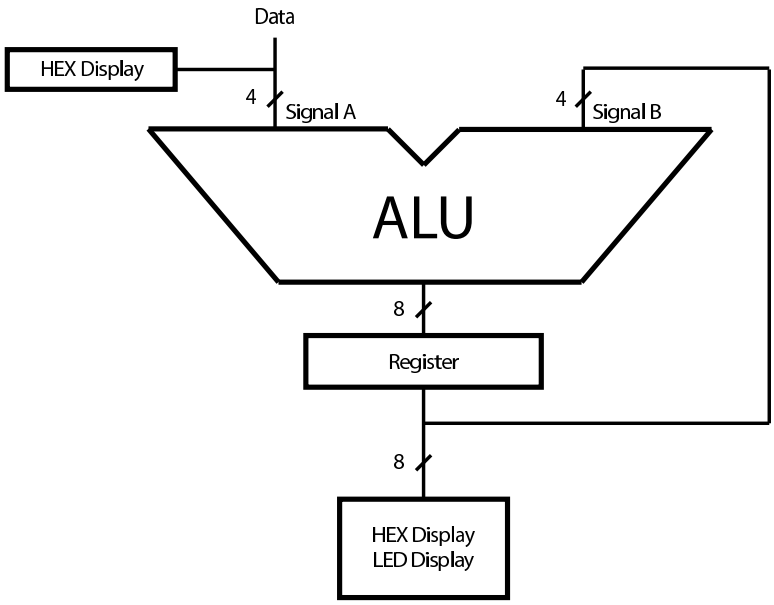

Figure 2: ALU Circuit for Part II.

Perform the following steps.

1. Create a new Quartus II project for your circuit.

2. Create a Verilog module for the simple ALU. Connect the $data$ input to switches SW_{3-0}. Connect KEY_0 to clock, SW_9 to reset and KEY_{3-1} for select signals. Display the outputs on $LEDR_{7-0}$; have *HEX0* display output of $data$ and set *HEX1*, *HEX2* and *HEX3* to 0. *HEX4* and *HEX5* should display the sum and carry out respectively.

3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must show this to the TA as part of your prelab.

4. Draw a schematic outlining the hierarchies you used and explain them to the TA as another part of your prelab.

5. Compile the project.

6. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit.

Part III

Figure 3 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 3 to design a left/right 8-bit rotating register with parallel load shown in Figure 4. The *LoadLeft* input of all eight instances of the circuit in Figure 3 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 3. 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 3.

The remaining connections between the eight instances of the circuit in Figure 3 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:

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

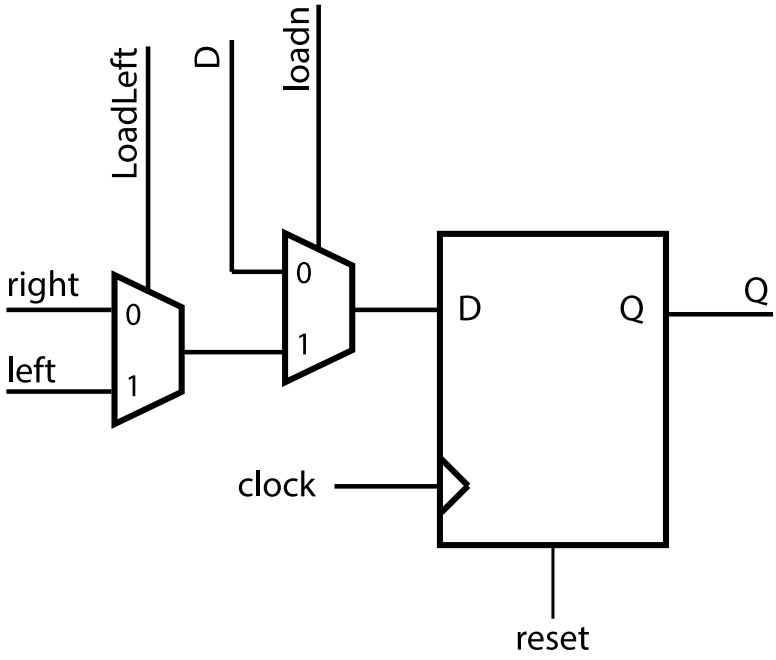

Figure 3: Sub-circuit for Part III.

Using code for a flop flop we saw in Part I and the *mux2to1* module from Lab 2, we can build the circuit shown above. Below, is a sample ~~hierarchy~~ code showing the D flip flop with one 2 to 1 multiplexer connected to it. Add the proper connections and the remaining multiplexer to the Verilog below:

```
mux2to1 M1(                    //instantiates 2nd multiplexer
    .x(rotatedata)             //output from left most multiplexer
    .y(data_D)                 //data D coming in
    .s(parallel_loadn)         //selects input D or rotate
    .m(datato_dff)             //outputs to flip flop
);

flipflop F0(                   //instantiates flip flop
    .d(datato_dff)             //input to flip flop
    .q(out_Q)                  //output from flip flop
    .clock(clock)              //clock signal
    .reset(reset)              //synchronous active high reset
);
```

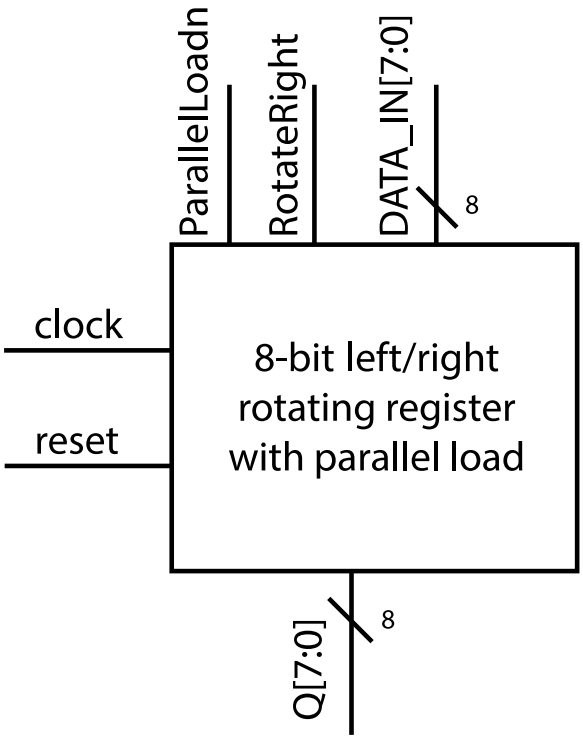

Figure 4: Top-level circuit for Part III.

Figure 4 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 3.

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 3. Label the signals on your schematic.
2. Create a new Quartus II project.
3. Write a Verilog module for the circuit in Figure 3.
4. Write a Verilog module for the rotating register with parallel load that instantiates eight instances of your Verilog module for Figure 3. This Verilog module should match with the schematic in your lab book. Use SW_{7-0} as the inputs $DATA\_IN_{7-0}$ and SW_9 as a synchronous active high reset. Use KEY_1 as the *ParallelLoadn* input, KEY_2 as the *RotateRight* input and KEY_3 as the *ASRight* input. Use KEY_0 as the clock and SW_0 as a synchronous active high reset, 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 ModelSim. In your simulation, you should perform the reset operation first. Then, clock the register for several cycles to demonstrate rotation in the left and right directions. **(NOTE: If you do not perform a reset first, your simulation will not work! Try simulating without doing reset first and see what happens. Can you explain the results?)**
7. Download your circuit on the DE1-SoC board.
8. Test the functionality of your rotating register.