LAB #4 Sequential Logic, Latches, Flip-Flops, Shift Registers, and Counters

LAB OBJECTIVES

1. Introduction to latches and the D type flip-flop
2. Use of actual flip-flops to help you understand sequential logic
3. Become more familiar with simulation
4. Understand the function of a "clock"
5. Understand flip-flop clock inputs using rising edge or falling edge

LAB PROCEDURE

PART 1: NAND gate version of the RS latch

Write a **Data Flow model description** in Verilog HDL for a NAND gate version of the RS latch as shown in Figure 4-1. Follow the **Introduction to Simulation of Verilog Designs Using ModelSim Graphical Waveform Editor** document on moodle, as well as the **How to create and run testbenches in Quartus II version 13.0** video, also found on moodle. Bring the file to Lab class and compile and simulate your design. IF satisfied with the results, download your Verilog solution to the CPLD-FPGA board. Test the latch and fill in the RS latch truth table. Be sure to show your simulation waveform in your lab report.

With a text editor, design the following:
Create the following file, save as rs_latch.v (bring file to lab).

```
module rs_latch (R,S,Q, NQ);
        input R, S;
        output Q, NQ;
        assign Q = ~( S & NQ); // S (SET) is active low
        assign NQ = ~(R & Q); // R (RESET) is active low
endmodule
```

S	R	Q_{old}	Q_{new}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Table 1. RS Latch NAND Truth Table

***Off-Line -** The S and R inputs for the NAND version latch in Part I are active low. This is the opposite logic for the S and R inputs of the NOR version latch (active high). Often inputs, such as switches, can be "true" when they are closed and shorted to ground giving a logic "0" as shown in the diagram below.*

QUESTION#1: What is the purpose of the two resistors in the schematic below?
{Be sure to put all the answers to the Questions in your conclusion.
Please number each question}

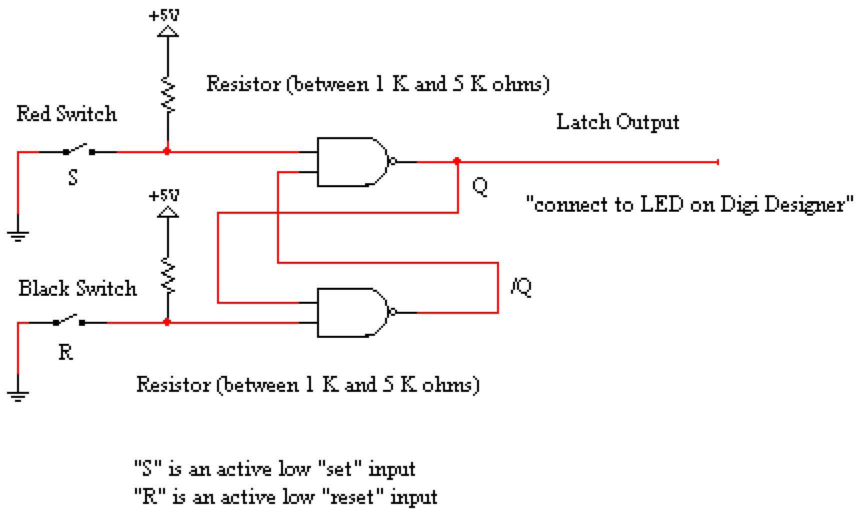

+5V
Resistor (between 1 K and 5 K ohms)
Red Switch
Latch Output
S
Q
+5V
"connect to LED on Digi Designer"
Black Switch
/Q
R
Resistor (between 1 K and 5 K ohms)
"S" is an active low "set" input
"R" is an active low "reset" input

Figure 4-1: NAND gate version of the RS latch

PART 2: NOR gate version of the RS latch

Write a Data Flow model description in Verilog HDL for a NOR gate version of the RS latch. Again, you should write the Verilog before lab class using a TEXT EDITOR. Bring the file to Lab class and compile and simulate your design. IF satisfied with the results, download your Verilog solution to the CPLD-FPGA board. Test the latch and create an RS latch function table like the one in lecture. BE SURE to show the simulation waveform in your lab report.

Question#2: What is the difference between NOR gate version and NAND gate version when using the S and R inputs?

S	**R**	**Q_{old}**	**Q_{new}**
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Table 2. RS Latch NOR Truth Table

PART 3: NOR gate version of the D Flip-Flop

Write a **Data Flow model description** in Verilog HDL for a NOR gate version of the D Flip-Flop. The logic diagram for the NOR gate version is on Figure 4-2. Draw a diagram for your report with the signal names you used in your Hardware Description. Again, you should write the Verilog before class using a TEXT EDITOR. Bring the file to lab class and compile and download your Verilog solution to the CPLD-FPGA board.

> ***Hint:*** *you will have 4 equations, two of them are:*
> *Q = ~ (R | QN) QN = ~ (S | Q)*
> *You will also need equations for R and S.*

Test the D Flip-Flop and record your results in a function table. You should begin to understand the true meaning of some being “edge-triggered”. Watch the outputs of the gates changing as the clock changes. The Q output will change to the D input during the clock transition.

Question#3: Which edge of the clock can cause the Q output of the D Flip-Flop (NOR version) to change, the positive / rising edge of the clock (from 0 to 1 transition) or the negative / falling edge transition) of the clock?

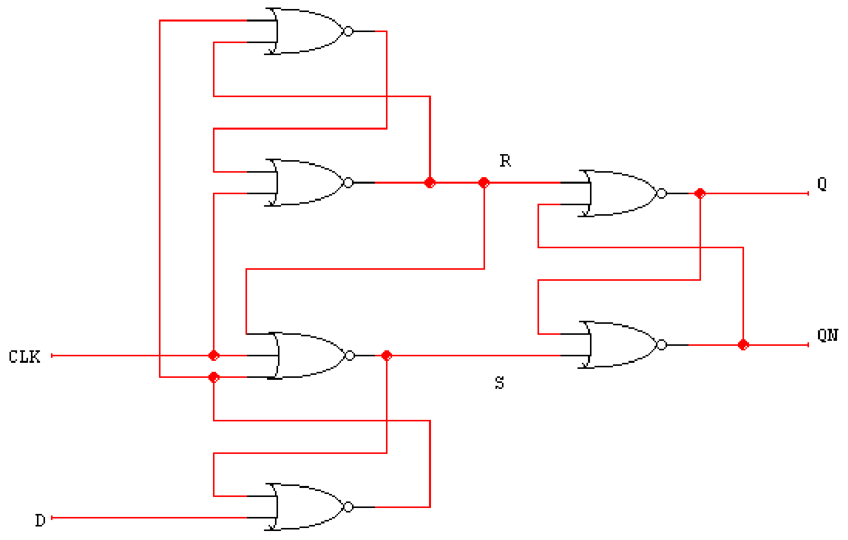
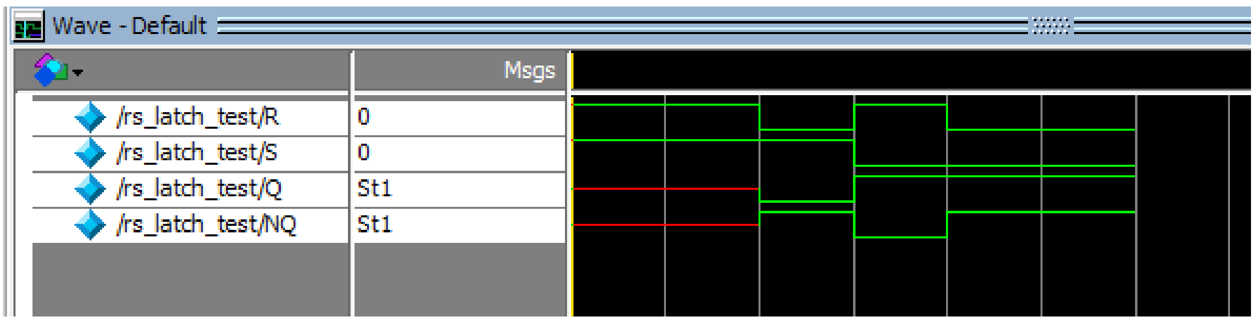


Figure 4-2: NOR gate version of the D Flip-Flop



	Msgs
/rs_latch_test/R	0
/rs_latch_test/S	0
/rs_latch_test/Q	St1
/rs_latch_test/NQ	St1

Figure 4-3: Simulation ‘timing diagram’ of a D Flip-Flop

***FAQ -** I find all this latch and flip-flop stuff confusing. Why?*
Answer: with combinational logic, when you change an input you automatically expect an output to change! With latches and flip-flops it is not this simple. Sometimes when inputs change, the output remains unchanged - and this can be correct. It takes time to fully understand sequential logic. Be patient and concentrate.

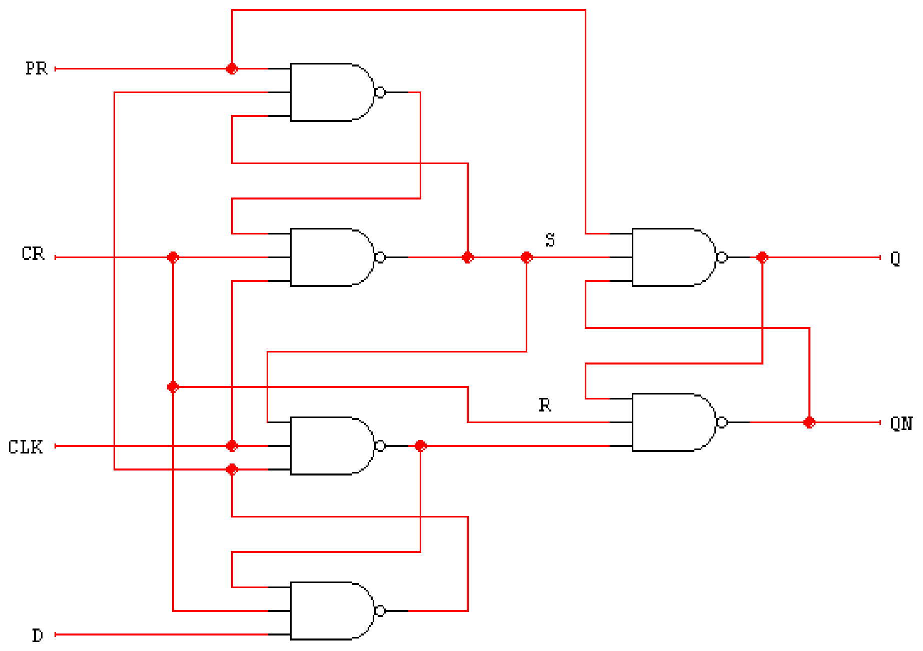


Figure 4-4: NAND gate version of the D Flip-Flop with Preset (PR) and Clear (CR)

PART 4: NAND gate version of the D Flip-Flop with Preset (PR) and Clear (CR)
Write a Data Flow model description for a NAND gate version of the D Flip-Flop with Preset (PR) and Clear (CR). The logic diagram for the NAND gate version is on Figure 4-4. Draw a Block diagram for your report with the signal names you used in your Hardware Description. Again, you should write the Verilog before lab class. Bring the file to lab class, compile and simulate. Download your Verilog solution to the CPLD-FPGA board. Test the D Flip-Flop and record your results in a function table like the one we did in lecture. Your function table should have the PR and CR signals. Show the waveform display (**Simulation**) in your lab report.

Question#4: Which edge of the clock can change the Q output of our D Flip-Flop (NAND version), the positive / rising edge (from 0 to 1 transition), or the negative / falling edge transition)?

Question#5: What are the logic levels of Q and QN when PR and CR are BOTH active at the same time? (NAND version)

Question#6: What is the difference between the NAND version and NOR version of the D Flip- Flop when using the Preset (PR) and Clear (CR)?

PART 5: Circuit using four D-type flip-flops
The diagram in Figure 4-5 shows a circuit using four D-type flip-flops. Write the Verilog equation for each of the four "D" inputs. For example, in FF_0 the equation for it's D input is

```
D0 = ~Q0;
```

In a similar manner, in FF_1 the equation for it's D input is:

```
D1 = (~Q1 & Q0) | (~Q0 & Q1);
```

Compile and then simulate. Notice that the only input is the "clock" for the flip-flops. Describe in your report the behavior of this design. Use the simulator and create a timing waveform with at least 20 clock pulses; observe the flip-flop outputs in the bit order Q3, Q2, Q1, Q0.

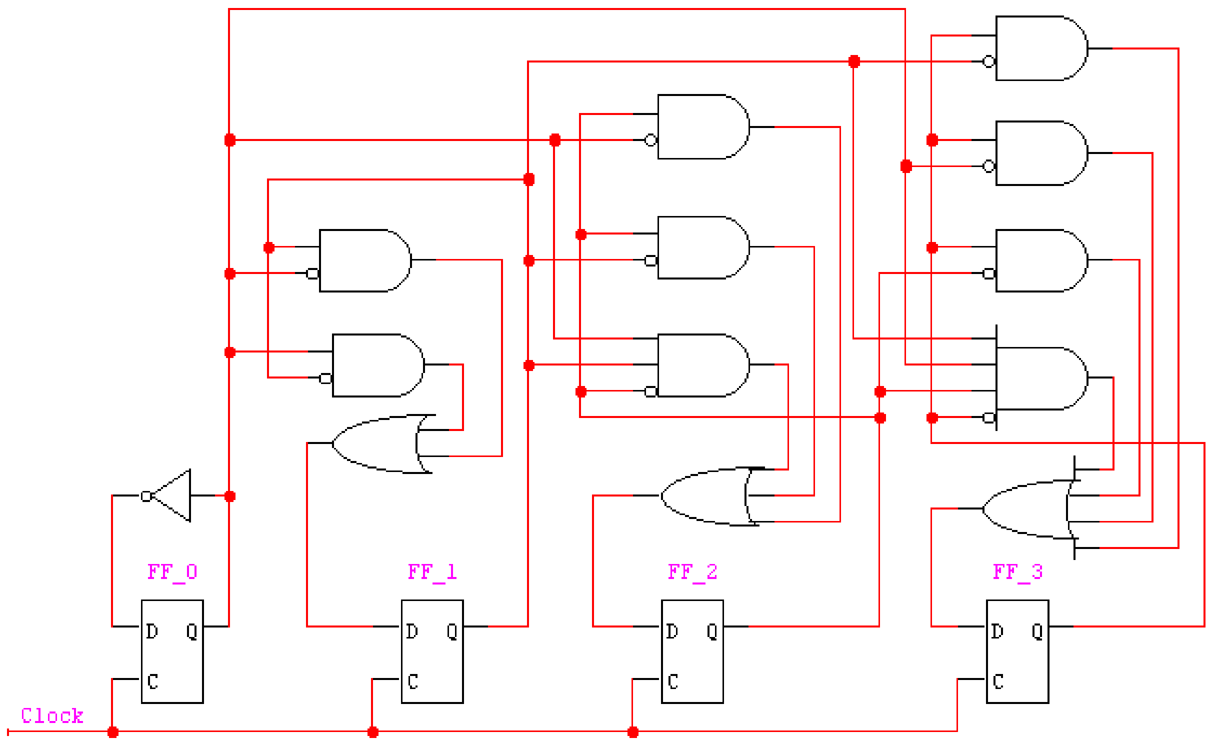


Figure 4-5: Circuit using four D-type flip-flops

Assign the Q outputs of the flip-flops to pins that output to light emitting diodes. Be sure to assign the clock pin of your design to a clock pin on your CPLD-FPGA; the momentary contact switches on the CPLD-FPGA board are connected to clock pins on the CPLD-FPGA. You will need to **debounce** the switches, see the manual clock section.
Download to the CPLD-FPGA board. Does your hardware match the waveform simulation? Describe the circuit's function in your report. **Use at least 20 clock pulses with your simulation waveform display**.

See appendix B for further assistance.

PART 6: Three-Stage Shift Register
The logic diagram in Figure 4-6 illustrates a three-stage shift register. Using this idea, expand the circuit into a eight stage shift register. Design Verilog code to model your design. Assign outputs to LEDs, compile, simulate, and download and test. Record your results for you lab report with your **simulation** waveform.

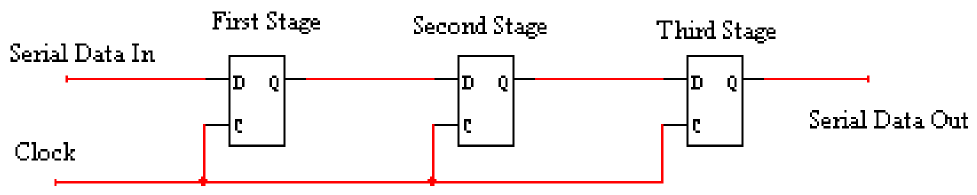


Figure 4-6: Three-Stage Shift Register

PART 7: Eight Bit Shift Register Design
Design an eight bit shift register that can be synchronously pre-loaded via a control line called "PL" (Parallel Load). You need to write "**equations**" for each of the D F/F inputs as a function of the previous flip-flop output, the "PL" signal and the parallel load data signals (call them P7, P6, P5, P4, P3, P2, P1, P0). The D F/F signals (D7 – D0) should be displayed on the LEDs next to the Q outputs (Q7 – Q0). For this part you need to use eight assign statements for D7 – D0. Record your results and show the **simulation** waveform in your report.

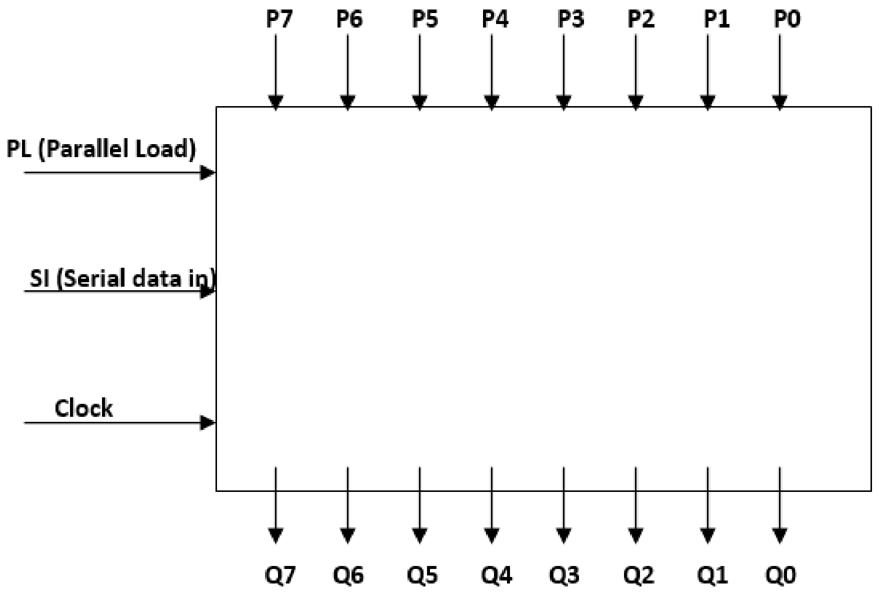


Figure 4-7: Eight Bit Shift Register

Which way is your circuit "Shifting"? Is your circuit Shifting Left or Shifting, Right? Write a Verilog Description (any style) so your circuit does BOTH. You will need an additional control line called SHL or SHR, meaning Shift Left or Shift Right respectively.

Write a Verilog Description (any style) so your circuit does the SHIFT Left and SHIFT Right function and also the ROTATE Left and ROTATE Right function. You will need additional control lines called SHL or SHR, (meaning Shift Left or Shift Right respectively) and ROL or ROR, meaning Rotate Left or Rotate Right respectively.

Lab 4 – Sequential Logic Demo List

Part Number	**Requirements**	**Completed**
Part 1 – NAND gate version of the RS latch		
	Multisim	
	Verilog: Use “data flow” modeling	
	Waveform	
Part 2 – NOR gate version of the RS latch		
	Multisim	
	Verilog: Use “data flow” modeling	
	Waveform	
Part 3 – NOR gate version of the D Flip-Flop		
	Multisim	
	Verilog: Use “data flow” modeling	
	Waveform	
Part 4 – NAND gate version of the D Flip-Flop with Preset(PR) and Clear (CR)		
	Verilog: Use “data flow” modeling	
	Waveform	
Part 5 – Circuit using four D-type flip-flops (Ensure clock signal is debounced)		
	Verilog	
	Waveform	
Part 6 – Three stage shift register (Ensure clock signal is debounced)		
	Verilog	
	Waveform	
Part 7 – Eight Bit Shift Register (Ensure clock signal is debounced)		
	Verilog	
	Waveform	

Note 1: “Verilog” is referring to testing the code on your FPGA (Hardware)

Note 2: “Waveform” is referring to creating a test bench then producing a waveform.

Appendix A – Debouncing a "manual" clock

If you are using a DE0-nano the push buttons are already debounced and can be used for your "manual" clock signal.

What does it take to make a "manual" clock? The circuit below is a latching circuit. Instead of two switches, this circuit uses what is called a single pole, double throw switch. Pick resistors between 1K and 4.7K.

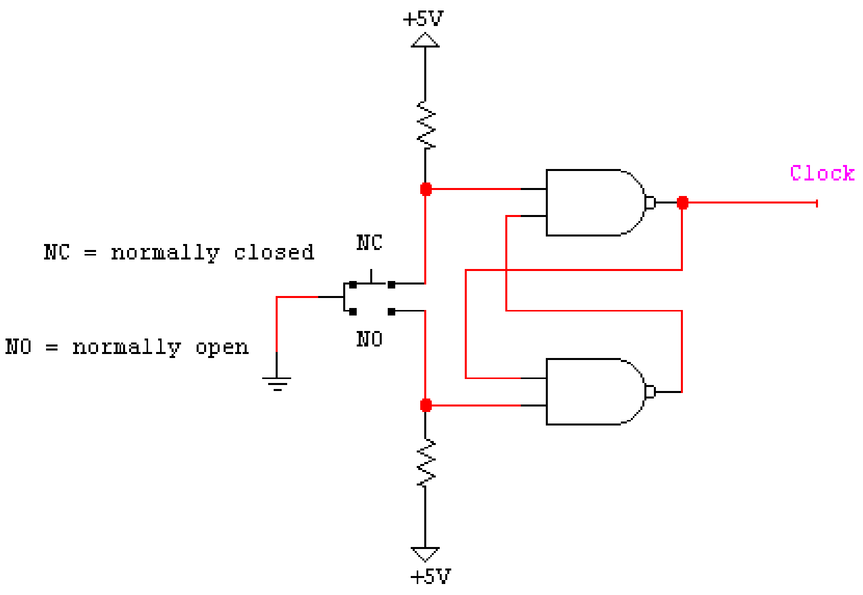


Figure 4-8: "Manual" Clock with Momentary Contact Switches

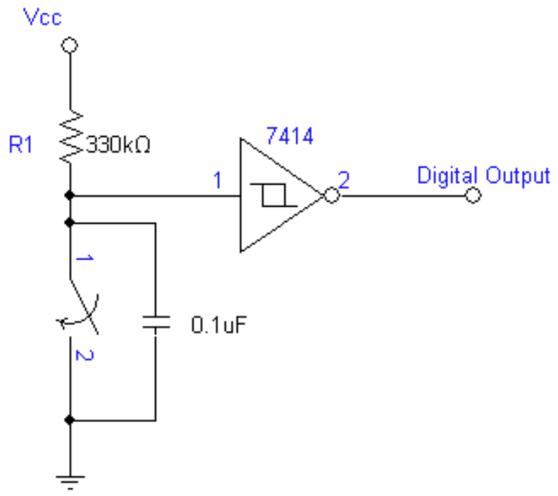
Remember in part 5 you used the momentary switches as a manual clock for the first time to increment your 4-bit counter. You found that there was a difference between the right momentary switch and the left. Do you remember the difference? Do Part 5 again using the left momentary switch as your clock input. Your 4-bit counter must be able to count 0-Fh without skipping any counts using the *left momentary switch* as your clock input.

If you are using a DE0-nano the push buttons are already debounced and can be used for your "manual" clock signal.

Reference information on debouncing switches STSP and STDP:

Digital Electronics: Debouncing a Slider Switch (SPDT)

Digital Electronics: Debouncing a Push Button Switch (SPST)



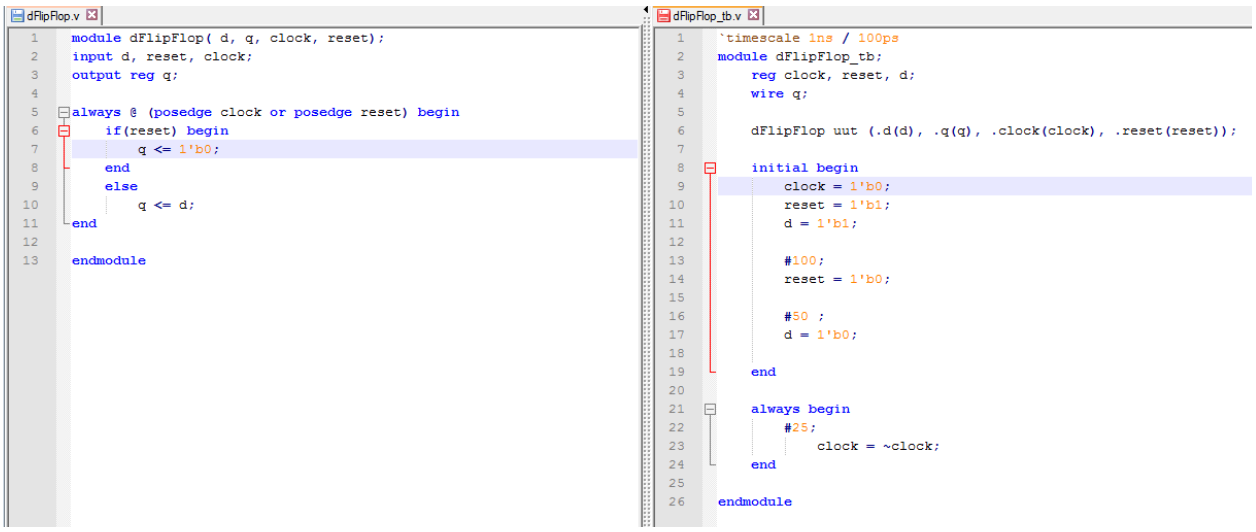
```
module posedgedff(D,clock,Q);
input wire clock,D;
output reg Q;

always @ (posedge clock)
Q<=D;
endmodule
```

```
module debounce(clk, switch, q);
input clk;      //  onboard clock *pin 12
input switch;  // "switch" is the glitched, asynchronous, active low push-button signal
output reg q;   // 1 while the push-button is active (down)

reg [15:0] cnt;
wire mcount = &cnt;          // true when all bits of cnt are 1's

        always @(posedge clk)
                if(q!=switch) cnt <= 0;        // nothing's going on
                else
                begin
                  cnt <= cnt + 1;              // something happened, start counting
                  if(mcount) q <= ~q;          // if the counter is maxed, switch changed!
                end
endmodule
```

Appendix B – Asynchronous Reset and Clock Signal in Testbench

For those who are working on lab 4 and are curious on how to create a clock signal on the test bench I attached a photo of a test bench and a module of a d-flip flop below. If you look in the dFlipFlop_tb.v file there is an always block below the initial block. This always block will invert the clock signal every 25 time units, this way we can simulate a clock signal and depending on the time units we make it we can change the signal frequency to be faster or slower. One thing to note though is that since the we invert the signal every 25 time units we need to initially set the clock signal to be low (0) or high (1).

Looking at the d-FlipFlop module, you can see that in the sensitivity list on line 5 there is a reset signal. And inside the always block there is a if statement that if reset is high the output q will be 0. This is referred to as an asynchronous reset because it will set the output signal q to be 0 regardless of the clock signal. This type of signal will be useful in future lab parts like lab 4 part 5 where the d input is tied only to the other d-flipflop output q signals. The asynchronous reset can set your counter back to zero.



```
1   module dFlipFlop( d, q, clock, reset);
2   input d, reset, clock;
3   output reg q;
4
5   always @ (posedge clock or posedge reset) begin
6       if(reset) begin
7           q <= 1'b0;
8       end
9       else
10          q <= d;
11  end
12
13  endmodule
```

```
1   `timescale 1ns / 100ps
2   module dFlipFlop_tb;
3       reg clock, reset, d;
4       wire q;
5
6       dFlipFlop uut (.d(d), .q(q), .clock(clock), .reset(reset));
7
8       initial begin
9           clock = 1'b0;
10          reset = 1'b1;
11          d = 1'b1;
12
13          #100;
14          reset = 1'b0;
15
16          #50 ;
17          d = 1'b0;
18
19      end
20
21      always begin
22          #25;
23              clock = ~clock;
24      end
25
26  endmodule
```

Appendix C - Useful EXAMPLES of Verilog HDL for Lab 4

```
switchin(s,r,q,module i1,i2,led1,led2);
// Design for 6 sets of latched switches
// leds are simply connected to inputs i1 and i2
input i1,i2;
input [5:0] s,r;
output [5:0] q;
output led1,led2;

reg [5:0] q;
reg [5:0] qq;
wire led1, led2;

assign led1 = i1;
assign led2 = i2;

always@(s or r or qq)
begin
    qq <= ~s | (r & qq);
    q <= qq;
end

endmodule
```

```
module D_ff (D,clock,Q,Qbar);

input    D,clock;
output  Q,Qbar;

reg       Q,Qbar;

always@(posedge clock)  // will build a complete
D-type flip-flop
begin
     Q <= D;
     Qbar<= ~D;
end

endmodule
```

```
module counter (reset,clock,y0,y1,y2,y3);

input      reset,clock;
output   y0,y1,y2,y3;
reg    y0,y1,y2,y3;
always@(posedge clock or posedge reset)
    begin
       if (reset)
         {y3,y2,y1,y0} <= 4'b0000;
    else
         {y3,y2,y1,y0} <= {y3,y2,y1,y0} + 1; // "+" is the Verilog addition operator
end

endmodule
```