Gabriel Emerson
ELEC 4200 – Lab 8
10/21/21

# Lab 8: Modelling Registers and Counters

## Goals:

The goal for this lab is to become more familiar with the design of registers and counters. There are several tasks that ask for different properties of the circuit that would change the functionality of the circuit altogether. These registers and counters are very important because they are frequently used as an underlying subsystem for more complex programs. Counters are used often in sequential circuits. This means the understanding of these two concepts are crucial to understanding the beginnings of more complex systems.

## Design Process:

Task 1 and 2 are both modeling a 4 bit register with reset and load signals. The only difference between the two is task 2 requires to also use a set signal input in the design. This should not change the simulation output however. We are not asked to simulate task 2, but if we were to simulate both tasks, their output should be the same. The expected result for task 1 and 2 is shown in Figure 1.

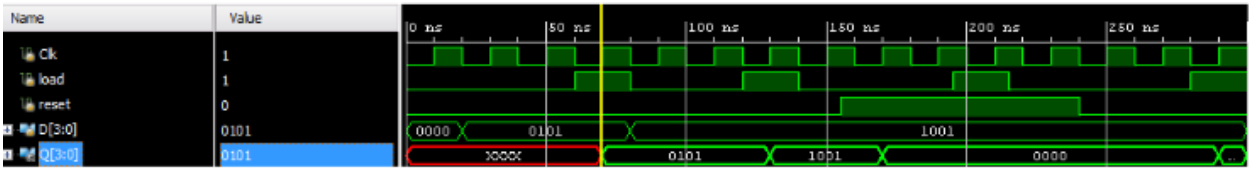


Figure 1

Task 3 begins a new idea of shifting bits into an output register. On this task we will use code provided in the lab writeup to create a 1 bit delay line shift register. The expected result from the simulation is shown in Figure 2.

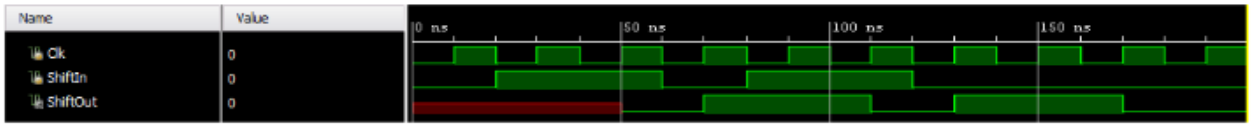


Figure 2

Task 4 also uses the shift register idea but now wants a 4-bit parallel in left shift register. This code is also provided above in the lab writeup. The expected result from this task is shown in Figure 3.

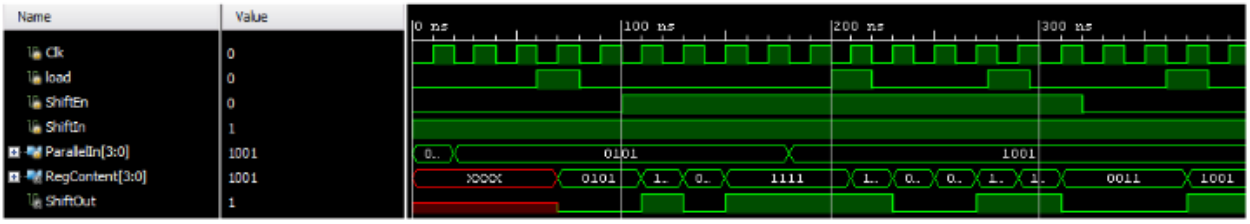


Figure 3

Task 5 asks us to create our own 4-bit parallel out shifter by using our knowledge from the previous two tasks. By viewing the code given to us in the last two tasks, we can determine how to create this shifter. Then we will simulate the design and the result from this should match the figure shown in Figure 4.

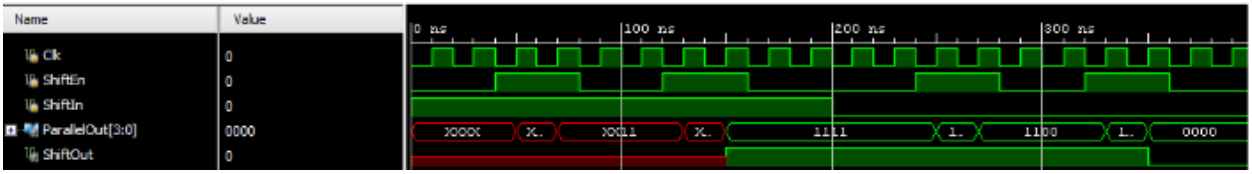


Figure 4

Task 6 and 7 begin a new topic of using counters and designing them using different flip-flops. First in task 6 we are asked to design an 8-bit counter using T flip-flops. This will be done by using behavioral modeling. Then in task 7 we will design the same, but instead use D flip-flops. This can be done because T flip-flops can be created by using D flip-flops. This means the result of both tasks should be the same. The expected output for both should be similar to what is given in the writeup which is shown in Figure 5.

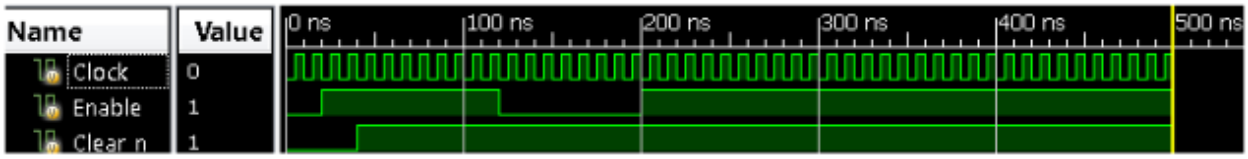


Figure 5

Task 8 is the last task of this lab and asks to design a 4 bit down counter which is shown in the writeup. Then we will develop a testbench and simulate the circuit. The expected result from this is shown in Figure 6.

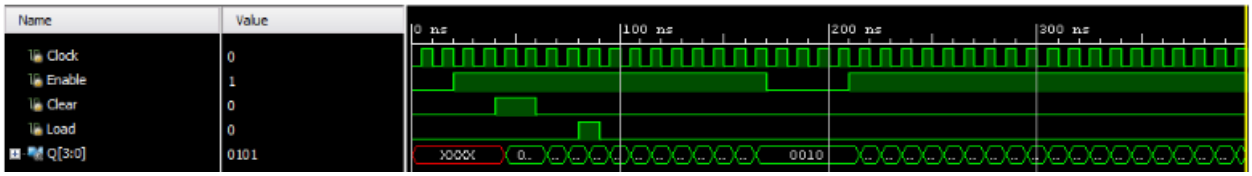


Figure 6

**Detailed Design:**

Task 1 and 2 are very similar in the design except the use of the set signal used in task 2. This means that the code for both will be very similar except for one input signal. We did not have to simulate the design for task 2, however, I found it useful to run the simulation using the same testbench as task 1. The code for Task 1 is shown in Figure 7, the code for Task 2 is shown in Figure 8, and the testbench used in shown in Figure 9.

```
module Register_with_synch_reset_load_behavior(input [3:0] D, input Clk, input reset, input load, output reg [3:0] Q);
    always @(posedge Clk)
    if (reset)
         begin
             Q <= 4'b0;
    end
    else if (load)
        begin
            Q <= D;
        end
endmodule
```

Figure 7

```
module Register_with_synch_reset_set_load_ehavior(input [3:0]D, input Clk, input reset, input set, input load, output reg [3:0] Q);

    always @(posedge Clk)
    if (reset)
        begin
            Q <= 4'b0;
        end
    else if (set)
        begin
            Q <= 4'b1;
        end
    else if (load)
        begin
            Q <= D;
        end
endmodule
```

Figure 8

```
module Lab7_1_tb();

    reg [3:0] D;
    reg Clk, reset, load;
    wire [3:0] Q;
    Register_with_synch_reset_load_behavior UUT(.D(D), .Clk(Clk), .reset(reset), .load(load), .Q(Q));

    initial
        begin
            Clk = 0;
            forever
            begin
            #10 Clk = ~Clk;
            end

    end
    initial begin
        D = 0; load = 0; reset = 0;
        #20 D = 4'b0101;
        #40 load = 1'b1;
        #20 D = 4'b1001; load = 0;
        #40 load = 1'b1;
        #20 load = 1'b0;
        #15 reset = 1'b1;
        #40 load = 1'b1;
        #20 load = 1'b0;
        #25 reset = 1'b0;
        #40 load = 1'b1;
        #40 $finish;

    end
endmodule
```

Figure 9

Task 3 to design 1 bit delay line shift register was simply using the code given in the writeup to become more familiar with the actions of a shift register. We were also given to write our own testbench for this task in order to check the system. The code used is shown below in Figure 10, and the testbench developed is shown in Figure 11.

```
module delay_line3_behavior(input Clk, input ShiftIn, output ShiftOut);
    reg [2:0] shift_reg;
    always @ (posedge Clk)
        shift_reg <= {shift_reg [1:0], ShiftIn};
        assign ShiftOut = shift_reg[2];
endmodule
```

Figure 10

```
module Lab7_3_tb;
    reg Clk, ShiftIn;
    wire ShiftOut;

    delay_line3_behavior UUT(.Clk(Clk), .ShiftIn(ShiftIn), .ShiftOut(ShiftOut));

    initial
         begin
             Clk = 0;
             forever
                begin
                #10 Clk=~Clk;
                end

             end

    initial begin
        ShiftIn = 0;

            #20 ShiftIn = 1'b1;
            #40 ShiftIn = 1'b0;
            #20 ShiftIn = 1'b1;
            #40 ShiftIn = 1'b0;
            #90 $finish;
    end
endmodule
```

Figure 11

Task 4 asked to use code given in the writeup to once again test and view the functionality of the shift register. This is used since in the next task we are asked to design our own which is done easily by viewing the design of this task and the previous one as well. We also developed our own testbench to verify the circuit. The code and testbench are shown below in Figures 12 and 13.

```
module Parallel_in_serial_out_load_enable_behavior(input Clk, input ShiftIn, input [3:0] ParallelIn, input load, input ShiftEn, output ShiftOut,output [3:0] RegContent);
    reg [3:0] shift_reg;
    always @(posedge Clk)
        if(load)
            shift_reg <= ParallelIn;
        else if (ShiftEn)
            shift_reg <= {shift_reg[2:0], ShiftIn};
        assign ShiftOut = shift_reg[3];
        assign RegContent = shift_reg;

endmodule
```

Figure 12

```
module Lab7_4_tb;
    reg Clk, load, ShiftEn,ShiftIn;
    reg [3:0] ParallelIn;
    wire [3:0] RegContent;
    wire ShiftOut;
    Parallel_in_serial_out_load_enable_behavior DUT(.Clk(Clk),.load(load),.ShiftEn(ShiftEn),.ShiftIn(ShiftIn),.ParallelIn(ParallelIn),.RegContent(RegContent),.ShiftOut(ShiftOut));

    initial
        begin
            Clk = 0;
            forever
                begin
                    #10 Clk=~Clk;
                end
        end
    initial begin
        load = 0;
        ShiftIn = 0;
        ShiftEn = 0;

            #0 ShiftIn=1'b1; ParallelIn = 4'b0011;
            #20 ParallelIn = 4'b0101;
            #40 load = 1'b1;
            #20 load = 1'b0;
            #20 ShiftEn = 1'b1;
            #80 ParallelIn = 4'b1001;
            #20 load = 1'b1;
            #20 load = 1'b0;
            #55 load = 1'b1;
            #20 load = 1'b0;
            #25 ShiftEn = 1'b0;
            #40 load = 1'b1;
            #20 load = 1'b0;
            #30 $finish;
    end
endmodule
```

Figure 13

Task 5 is writing our own design for a shift register that is parallel out. We can use the previous two tasks to get ideas on how this is done. We then also are tasked with writing a testbench file to simulate our design to verify it works before uploading it to the board. The code used in this task is shown in Figure 14 and the testbench is shown in Figure 15.

```
module Lab7_5(input Clk, input ShiftEn, input ShiftIn, output [3:0] ParallelOut, output ShiftOut);
    reg [3:0] shift_reg;

    always @(posedge Clk )
        if(ShiftEn)
            shift_reg <= {shift_reg[3:0],ShiftIn};
            assign ParallelOut = shift_reg;
            assign ShiftOut = shift_reg[3];
endmodule
```

Figure 14

```
module lab7_5_tb;
    reg Clk, ShiftEn, ShiftIn;
    wire [3:0] ParallelOut;
    wire ShiftOut;

    Lab7_5 DUT(.Clk(Clk),.ShiftEn(ShiftEn),.ShiftIn(ShiftIn),.ParallelOut(ParallelOut),.ShiftOut(ShiftOut));

    initial begin
        Clk = 0;
        forever begin
            #10 Clk=~Clk;
        end
    end
    initial begin
        ShiftEn = 0;
        ShiftIn = 0;

        #0 ShiftIn = 1'b1;
        #40 ShiftEn = 1'b1;
        #40 ShiftEn = 1'b0;
        #40 ShiftEn = 1'b1;
        #40 ShiftEn = 1'b0;
        #40 ShiftIn = 1'b0;
        #40 ShiftEn = 1'b1;
        #40 ShiftEn = 1'b0;
        #40 ShiftEn = 1'b1;
        #40 ShiftEn = 1'b0;
        #50 $finish;
    end
endmodule
```

Figure 15

Tasks 6 and 7 both are creating an 8 bit counter using different flip-flops. In task 6 we will use a T flip-flop and task 7 uses a D flip-flop. This can be done very similarly since you can create a T flip-flop using D flip-flops. Then we will model a testbench to check both tasks. Since they are both output counters, the results should be exactly the same. The code used in Task 6 is shown in Figure 16, the code of Task 7 is shown in Figure 17, and the testbench for both is shown in Figure 18.

```
// Define T_FF
module T_ff (input Clk, input reset, input T, output reg Q);
    always @(posedge Clk or negedge reset)
     begin
         if(~reset)
            Q <= 1'b0;
        else
            Q <= Q ^ T;
    end
endmodule
//Define 8 bit Counter

module counter_8bit(input Clk, input Clear_n, input Enable, output [7:0] Q);
    wire w0, w1, w2, w3, w4, w5, w6, w7;
    //ff1
    T_ff ff0(.Clk(Clk),.reset(Clear_n),.T(Enable),.Q(Q[0]));
    and (w0, Q[0], Enable);
    //ff2
    T_ff ff1(.Clk(Clk),.reset(Clear_n),.T(w0),.Q(Q[1]));
    and (w1,Q[1],w0);
    //ff3
    T_ff ff2(.Clk(Clk),.reset(Clear_n),.T(w1),.Q(Q[2]));
    and (w2,Q[2],w1);
    //ff4
    T_ff ff3(.Clk(Clk),.reset(Clear_n),.T(w2),.Q(Q[3]));
    and (w3,Q[3],w2);
    //ff5
    T_ff ff4(.Clk(Clk),.reset(Clear_n),.T(w3),.Q(Q[4]));
    and (w4,Q[4],w3);
    //ff6
    T_ff ff5(.Clk(Clk),.reset(Clear_n),.T(w4),.Q(Q[5]));
    and (w5,Q[5],w4);
    //ff7
    T_ff ff6(.Clk(Clk),.reset(Clear_n),.T(w5),.Q(Q[6]));
    and (w6,Q[6],w5);
    //ff8
    T_ff ff7(.Clk(Clk),.reset(Clear_n),.T(w6),.Q(Q[7]));
endmodule
```

Figure 16

```
//D Flip-flop
module D_ff_behavioral(D,Clk,reset,Q);
    input D,Clk,reset;
    output reg Q;
    always @(posedge Clk or negedge reset)
        if(~reset)
            Q <= 1'b0;
        else
            Q <= D;
endmodule
//T Flip-Flop
module T_ff(T,Clk,reset,Q);
    input T, Clk, reset;
    output Q;

    wire Q_in, w1;

    assign w1 = (T^Q);
    assign Q_in = Q;
    D_ff_behavioral ff1(.D(w1),.Clk(Clk),.reset(reset),.Q(Q));
endmodule
// 8-bit Counter
module dff_counter_8bit(input Clk, input Clear_n, input Enable, output [7:0] Q);
    wire w0, w1, w2, w3, w4, w5, w6, w7;
    //ff1
    T_ff ff0(.Clk(Clk),.reset(Clear_n),.T(Enable),.Q(Q[0]));
    and (w0, Q[0], Enable);
    //ff2
    T_ff ff1(.Clk(Clk),.reset(Clear_n),.T(w0),.Q(Q[1]));
    and (w1,Q[1],w0);
    //ff3
    T_ff ff2(.Clk(Clk),.reset(Clear_n),.T(w1),.Q(Q[2]));
    and (w2,Q[2],w1);
    //ff4
    T_ff ff3(.Clk(Clk),.reset(Clear_n),.T(w2),.Q(Q[3]));
    and (w3,Q[3],w2);
    //ff5
    T_ff ff4(.Clk(Clk),.reset(Clear_n),.T(w3),.Q(Q[4]));
    and (w4,Q[4],w3);
    //ff6
    T_ff ff5(.Clk(Clk),.reset(Clear_n),.T(w4),.Q(Q[5]));
    and (w5,Q[5],w4);
    //ff7
    T_ff ff6(.Clk(Clk),.reset(Clear_n),.T(w5),.Q(Q[6]));
    and (w6,Q[6],w5);
    //ff8
    T_ff ff7(.Clk(Clk),.reset(Clear_n),.T(w6),.Q(Q[7]));
endmodule
```

Figure 17

```
module counter_8bit_tb;
    reg Clk,Clear_n,Enable;
    wire [7:0] Q;

    counter_8bit UUT(.Clk(Clk),.Clear_n(Clear_n),.Enable(Enable),.Q(Q));

    initial begin
        Clk = 0;
        forever
        #5 Clk = ~Clk;
        end
    initial begin
        Enable = 0;
        Clear_n = 0;

        #20 Enable = 1'b1;
        #20 Clear_n = 1'b1;
        #80 Enable = 1'b0;
        #80 Enable = 1'b1;
        #300 $finish;
    end
endmodule
```

Figure 18

Task 8 uses code given in the writeup to design a circuit that is 4 bits and counts down. This is done by simply viewing the code given to use in the lab writeup. Then we will develop our own testbench and test the circuit. The code is shown below in Figure 19, and the testbench in Figure 20.

```
module downcounter_4bit(input Clock, input Enable,input Clear, input Load,output [3:0] Q);
    reg [3:0] count;
    wire cnt_done;

    assign cnt_done = ~| count;
    assign Q = count;

    always @(posedge Clock)
        if (Clear)
            count <= 0;
        else if (Enable)
        if (Load| cnt_done)
            count <= 4'b1010;
        else
        count <= count - 1;
endmodule
```

Figure 19

```
module Lab7_8_tb;
reg Clock,Enable,Clear,Load;
wire [3:0] Q;

downcounter_4bit UUT(.Clock(Clock),.Enable(Enable),.Clear(Clear),.Load(Load),.Q(Q));

initial begin
    Clock = 0;
    forever
        begin
        #5 Clock = ~Clock;
        end
end
initial begin
    Clock=0; Enable=0;Clear=0; Load=0;

    #20 Enable = 1'b1;
    #20 Clear = 1'b1;
    #20 Clear = 1'b0;
    #20 Load = 1'b1;
    #10 Load = 1'b0;
    #80 Enable = 1'b0;
    #40 Enable = 1'b1;
    #220 $finish;
end
endmodule
```

Figure 20

**Verification:**

All of these tasks can be verified using a testbench and simulation to view whether the output of the simulation looks like the expected simulation or not. Task 2 does not explicitly say to do this, however, the result should look very similar to that of task 1. The simulations for all tasks (task 1 and 2 are the same and shown in Figure 21) are shown in the Figures 21-27 respectively. Besides testing their simulation, we also can upload the programs to the board to visually verify their functionality. By doing this we can see first hand how the program works in real time.

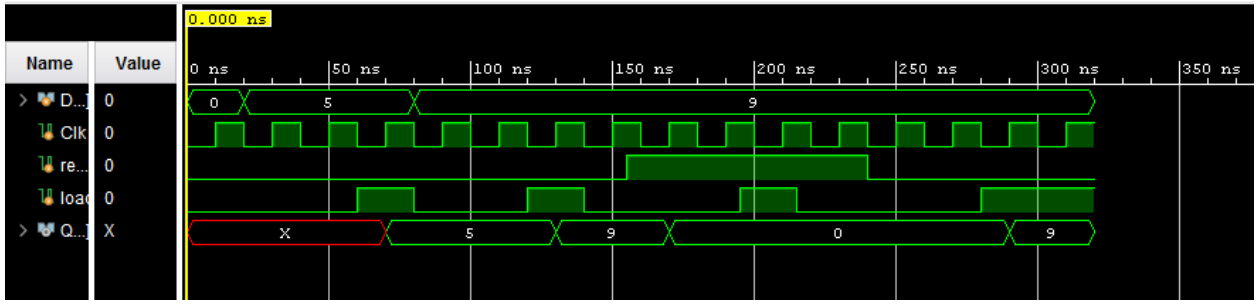


Figure 21

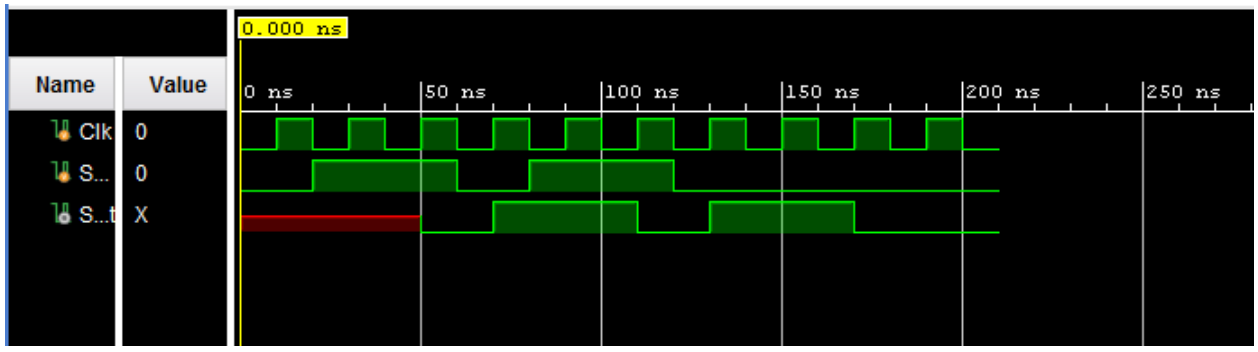


Figure 22

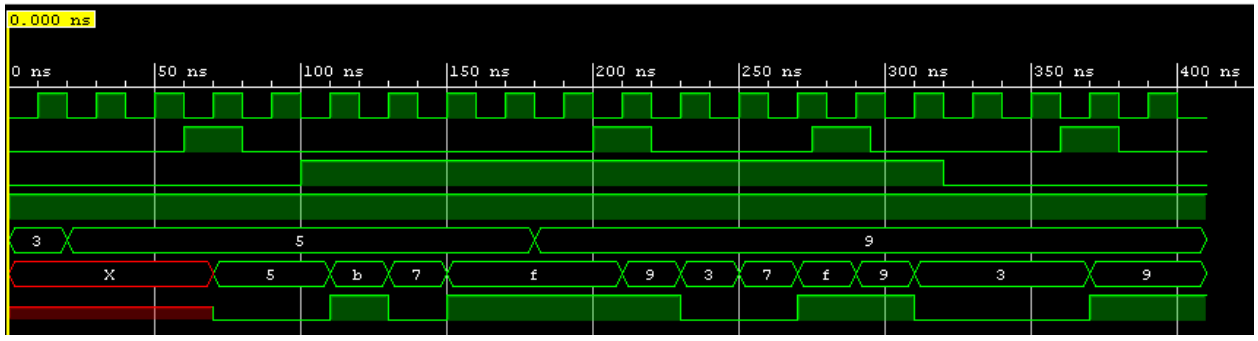


Figure 23

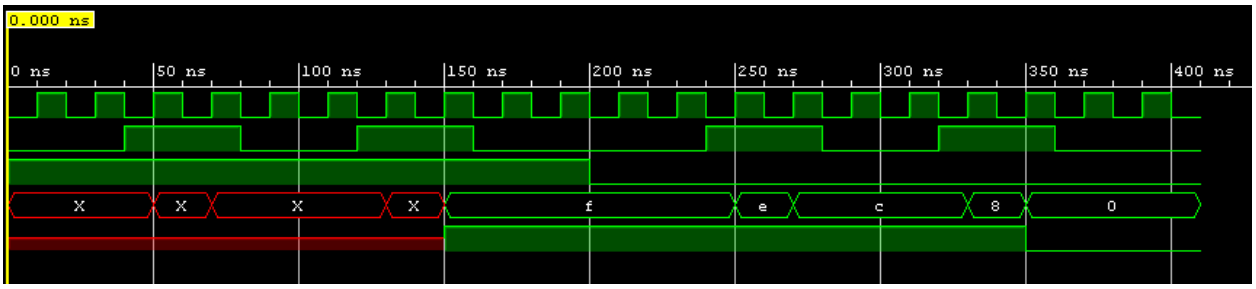


Figure 24

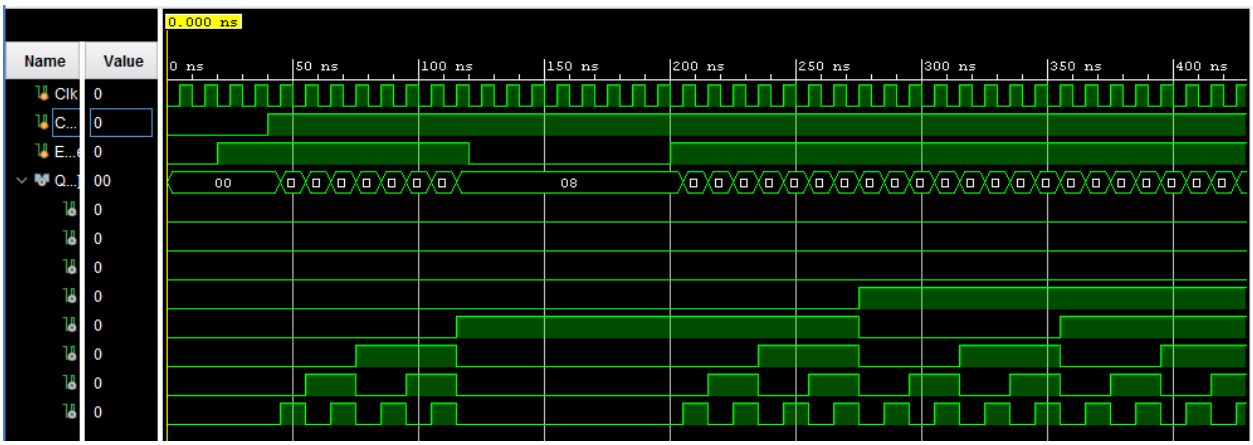


Figure 25

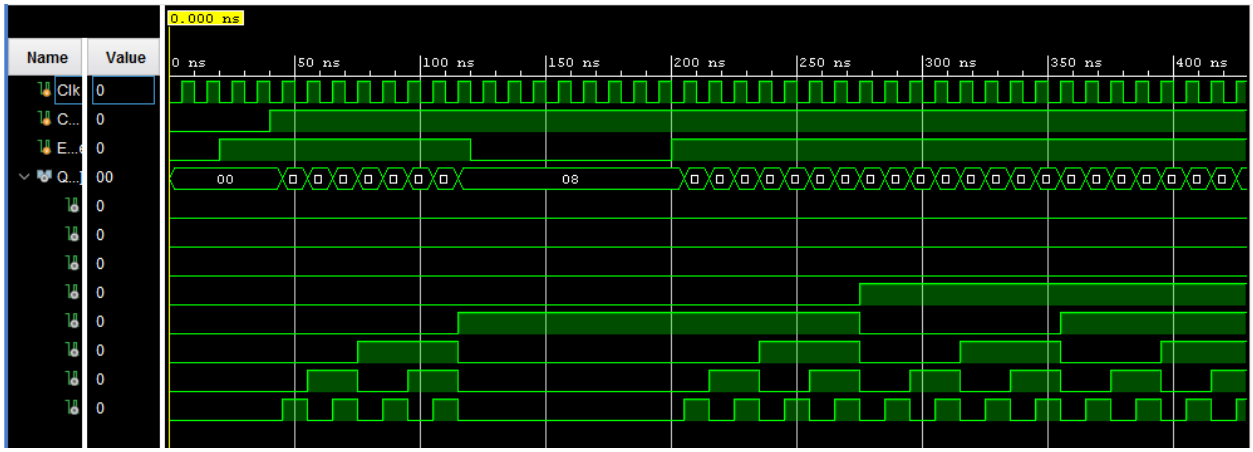


Figure 26

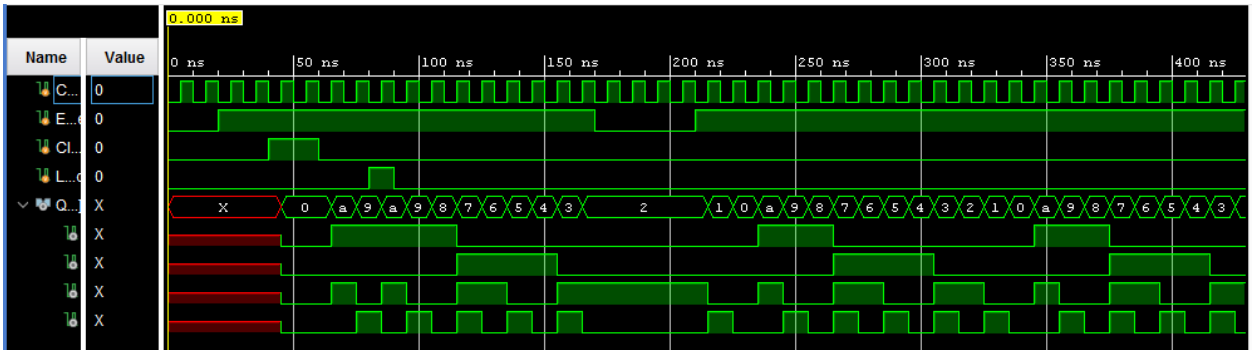


Figure 27

**Conclusion:**

What I learned in this lab is how to design many different registers and counters with varying properties. This includes systems such as a simple register, shift registers, and counters going up and down. We also got some experience with using code given to us and modifying it to make something else that is similar but does other things. Since we developed our own testbenches for all tasks we also got more experience writing a testbench and debugging our code functionality using the simulation. This was a good lab since systems of registers and counters are frequently used as subsystems to much more complex circuits.