Gabriel Emerson
ELEC 4200 – Lab 4
09/16/21

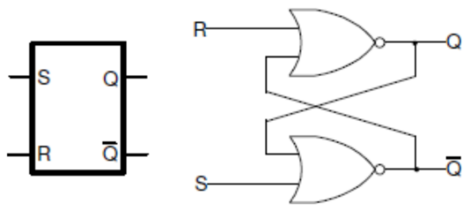
Lab 4: Modeling Latches and Flip-Flops

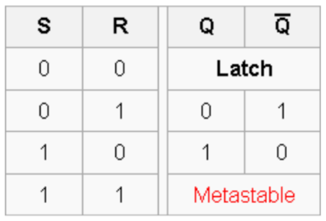
Goals:

The goal for this lab is to become familiar with the different modeling structures of circuits with latches or flip-flops. The newest addition on this lab is that we now write our own testbenches for simulation testing in in Verilog. The code is mostly given for this lab and we now use what we have learned to write a testing file in order to verify the functionality of the program.

Design Process:

Getting started for task one is essentially just viewing what was given in the writeup for the lab. After essentially copy and pasting the code from the writeup, we then just write a testing file for simulation purposes. This simulation should just test the different states of the SR latch and what the outputs are. The schematic and truth table are shown below in Figure 1 and the output should be similar to the output shown in the writeup which is shown in Figure 2.





S	R	Q	Q̄
0	0	Latch	
0	1	0	1
1	0	1	0
1	1	Metastable	

Figure 1

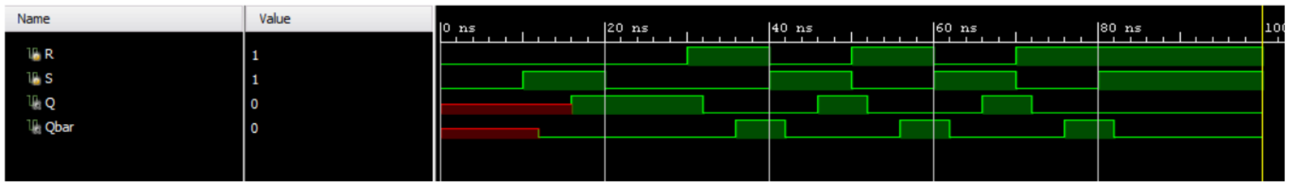
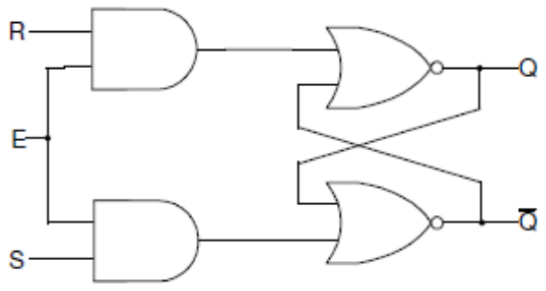
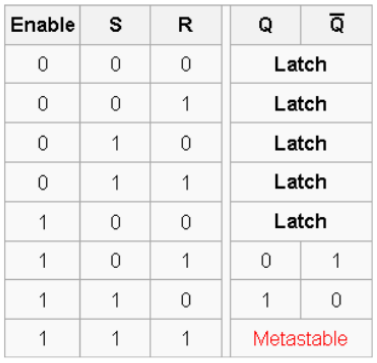


Figure 2

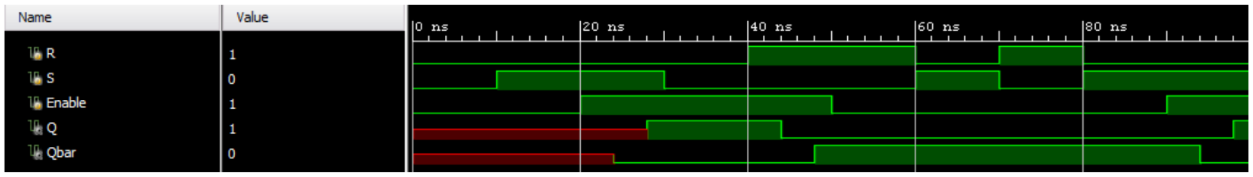
Task 2 was similar to task 1, except instead of using the code of the latch shown in the writeup, we write our own using dataflow modeling. We then also write our own test file, which should be similar to the one used in task 2. Figure 3 shows the schematic and truth table for the given SR latch, and Figure 4 is the what the output should match, which was given in the lab writeup.





Enable	S	R	Q	$\overline{Q}$
0	0	0	Latch	
0	0	1	Latch	
0	1	0	Latch	
0	1	1	Latch	
1	0	0	Latch	
1	0	1	0	1
1	1	0	1	0
1	1	1	Metastable	

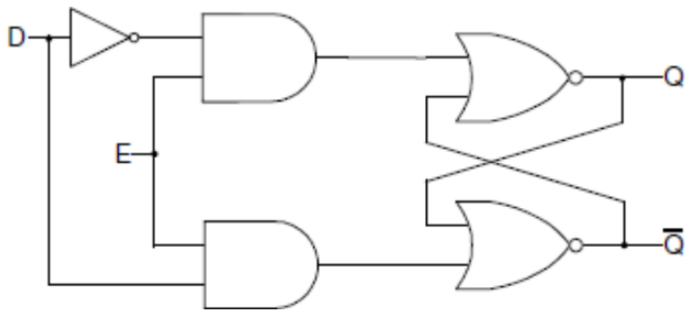
Figure 3

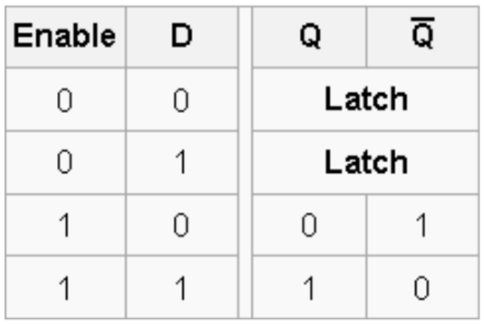


Name	Value
R	1
S	0
Enable	1
Q	1
Qbar	0

Figure 4

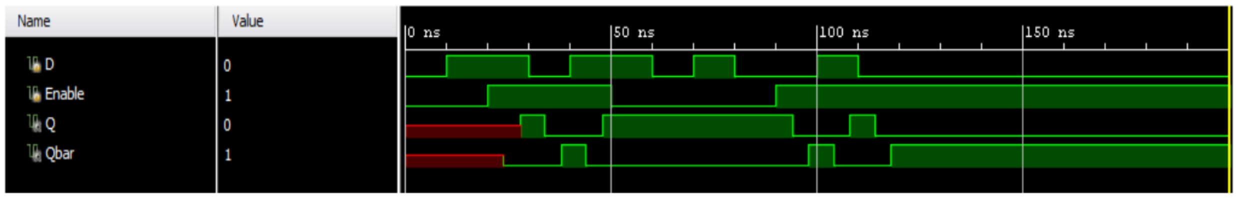
Task 3 had the same concept of the previous tasks, but this time used a D latch instead of the previous SR latches. This has the same concept of using latches, but there is only one latch option with an Enable input as well. There is also a task of writing a usable testbench in order to simulate the files to verify functionality. The schematic and truth table for a D latch is shown in Figure 5 and the example output given in the writeup is shown in Figure 6.





Enable	D	Q	$\overline{Q}$
0	0	Latch	
0	1	Latch	
1	0	0	1
1	1	1	0

Figure 5



Name	Value
D	0
Enable	1
Q	0
Qbar	1

Figure 6

Task 4 is very similar in setup to that of task 3. Both tasks use the same D flip flop, however, this task requires us to use behavioral modelling. Besides this small change, the testbench used in task 3, and most everything else should be roughly the same. After developing this, we can verify functionality by checking the schematic in Figure 5 above, and then checking the output example given, shown below in Figure 7.

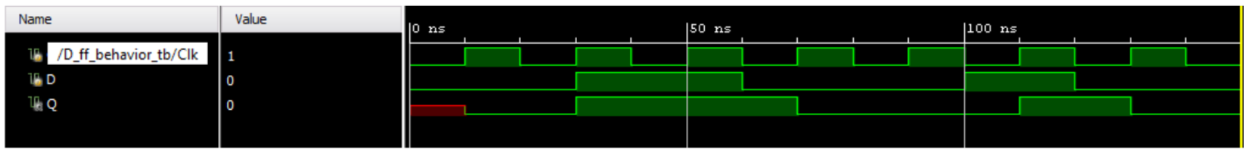


Figure 7

Task 5 is setting up 3 different circuits to test the difference between all 3 of them. The 3 circuits are the D latch and both D flip-flops (Rising edge and Falling edge). After setting up all 3 of these we test them with our own testbench to compare to the image given for this task. This image is shown in Figure 8.

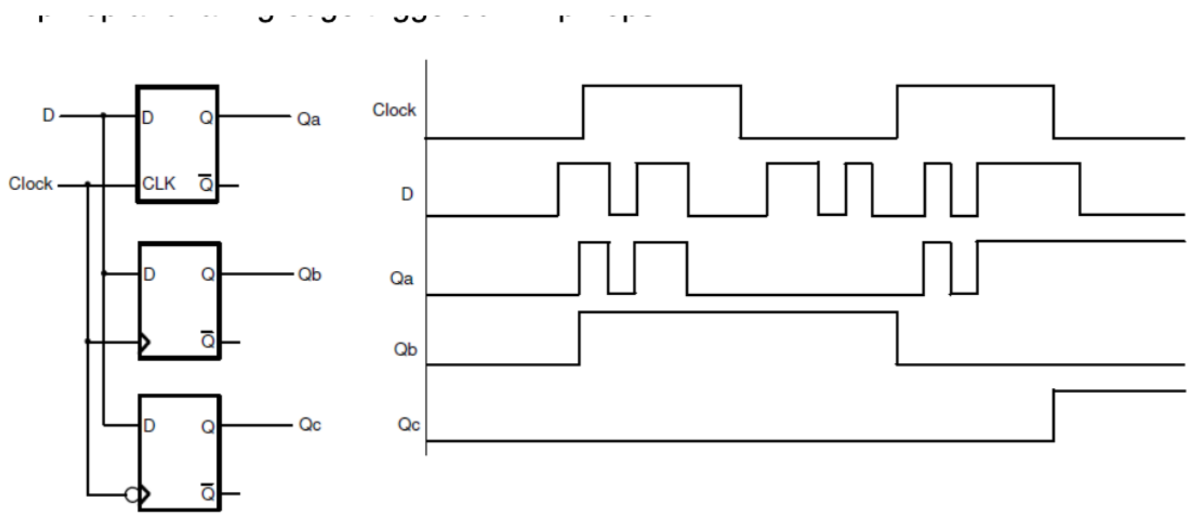


Figure 8

Task 6 begins studying the different circuits given a synch, reset, or clock enable all possibly acting on the circuit. For this task we only have the synchronous reset. We also must create a testbench to verify the program in simulation. After simulating the task, we compare to the output example shown in Figure 9.

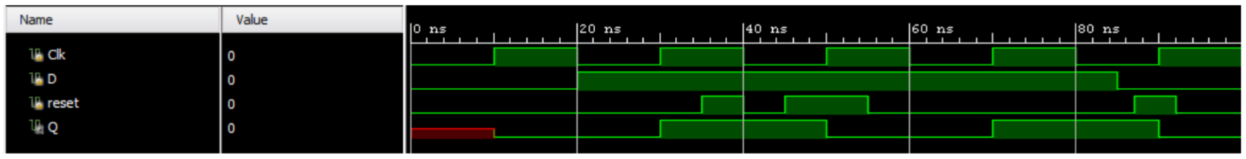


Figure 9

Task 7 models the same type of circuit in Task 6, but this time we add a clock enable pin to only act for output at certain points of the enable pin. Like the previous tasks, we then create a testbench to verify if the circuit works as intended. We can compare the results with the results given in the example output for this task, shown in Figure 10.

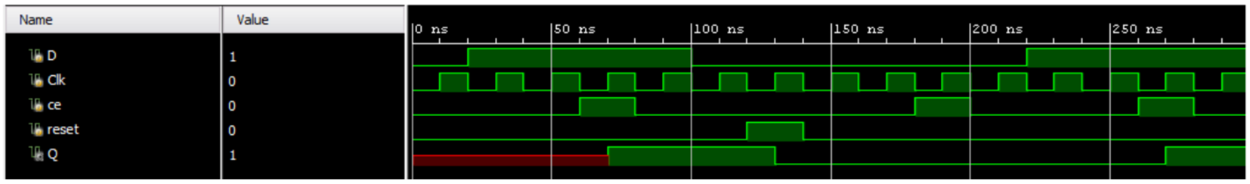
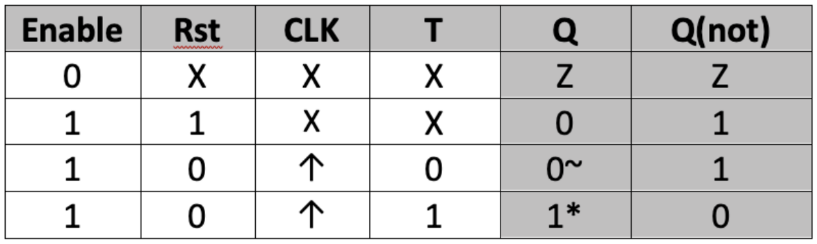


Figure 10

Task 8 is the final task for this lab, and Is the first creation of the T flip-flop which looks a lot like the other flip-flops. This task did not need a testbench, but simply needed to verify that it has been done. We can test to get the right program by comparing or logic to a normal toggle flip-flop, shown in Figure 11 below.

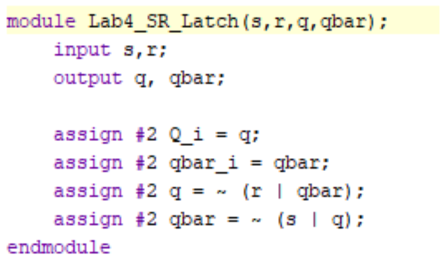


Enable	Rst	CLK	T	Q	Q(not)
0	X	X	X	Z	Z
1	1	X	X	0	1
1	0	↑	0	0~	1
1	0	↑	1	1*	0

Figure 11

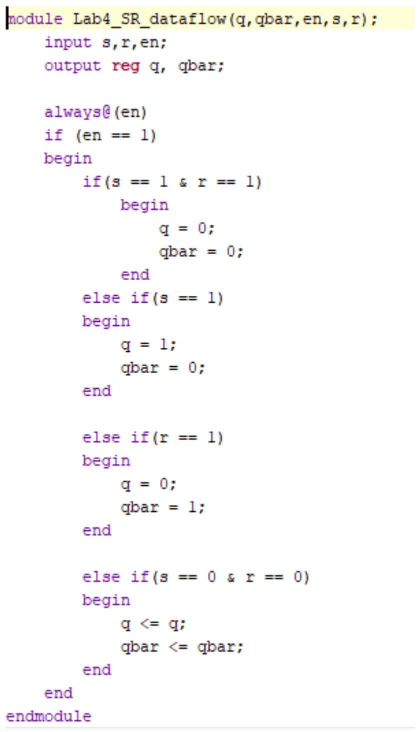
Detailed Design:

Task 1 and Task 2 were both designing and testing of SR latch circuits, except task 1 the code was given and all that must be done is designing of the testbench, and testing in simulation. The testbench is easy to implement for these two tasks, by simply setting cycles and setting the inputs to be certain values, and view the outputs. Task 1 (given code) is shown in Figure 12, task 2 code is shown in Figure 13, and task 1 & 2 testbenches are shown in Figures 14 & 15.



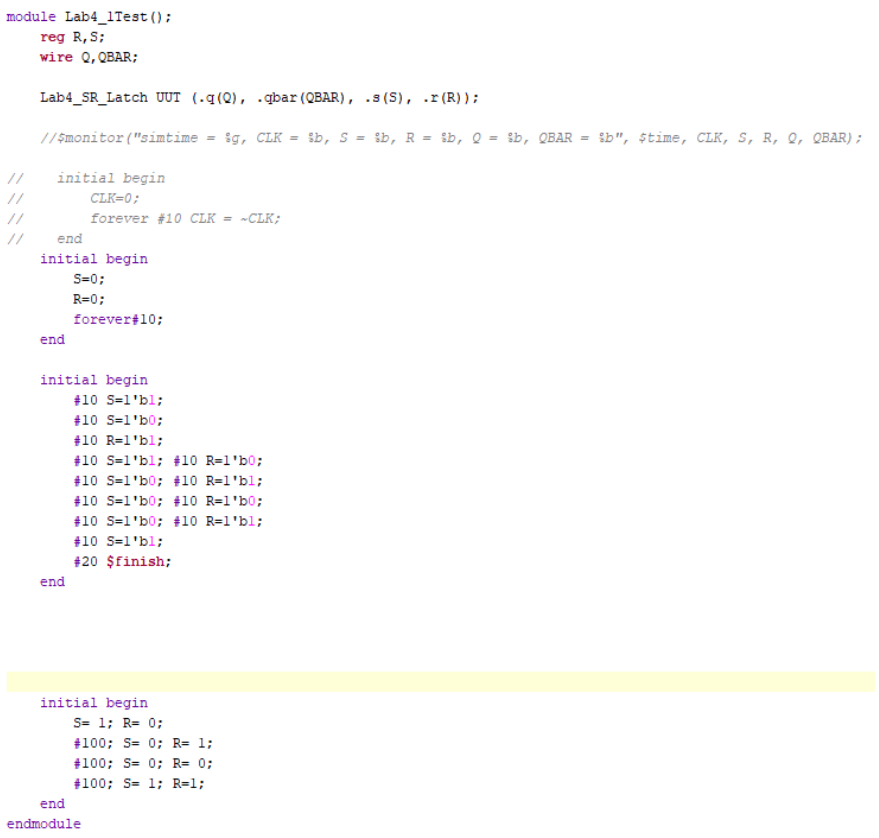
```
module Lab4_SR_Latch(s,r,q,qbar);
    input s,r;
    output q, qbar;

    assign #2 Q_i = q;
    assign #2 qbar_i = qbar;
    assign #2 q = ~ (r | qbar);
    assign #2 qbar = ~ (s | q);
endmodule
```

Figure 12



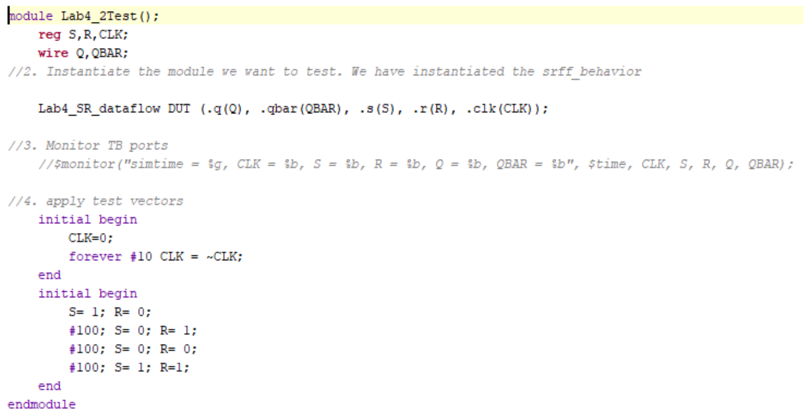
```
module Lab4_SR_dataflow(q,qbar,en,s,r);
    input s,r,en;
    output reg q, qbar;

    always@(en)
    if (en == 1)
    begin
        if(s == 1 & r == 1)
            begin
                q = 0;
                qbar = 0;
            end
        else if(s == 1)
        begin
            q = 1;
            qbar = 0;
        end

        else if(r == 1)
        begin
            q = 0;
            qbar = 1;
        end

        else if(s == 0 & r == 0)
        begin
            q <= q;
            qbar <= qbar;
        end
    end
endmodule
```

Figure 13



```
module Lab4_1Test();
    reg R,S;
    wire Q,QBAR;

    Lab4_SR_Latch UUT (.q(Q), .qbar(QBAR), .s(S), .r(R));

    //$monitor("simtime = %g, CLK = %b, S = %b, R = %b, Q = %b, QBAR = %b", $time, CLK, S, R, Q, QBAR);

//    initial begin
//        CLK=0;
//        forever #10 CLK = ~CLK;
//    end
    initial begin
        S=0;
        R=0;
        forever#10;
    end

    initial begin
        #10 S=1'b1;
        #10 S=1'b0;
        #10 R=1'b1;
        #10 S=1'b1; #10 R=1'b0;
        #10 S=1'b0; #10 R=1'b1;
        #10 S=1'b0; #10 R=1'b0;
        #10 S=1'b0; #10 R=1'b1;
        #10 S=1'b1;
        #20 $finish;
    end



    initial begin
        S= 1; R= 0;
        #100; S= 0; R= 1;
        #100; S= 0; R= 0;
        #100; S= 1; R=1;
    end
endmodule
```

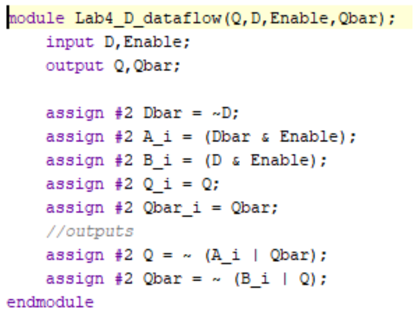
Figure 14



```
module Lab4_2Test();
    reg S,R,CLK;
    wire Q,QBAR;
//2. Instantiate the module we want to test. We have instantiated the srff_behavior

    Lab4_SR_dataflow DUT (.q(Q), .qbar(QBAR), .s(S), .r(R), .clk(CLK));

//3. Monitor TB ports
    //$monitor("simtime = %g, CLK = %b, S = %b, R = %b, Q = %b, QBAR = %b", $time, CLK, S, R, Q, QBAR);

//4. apply test vectors
    initial begin
        CLK=0;
        forever #10 CLK = ~CLK;
    end
    initial begin
        S= 1; R= 0;
        #100; S= 0; R= 1;
        #100; S= 0; R= 0;
        #100; S= 1; R=1;
    end
endmodule
```

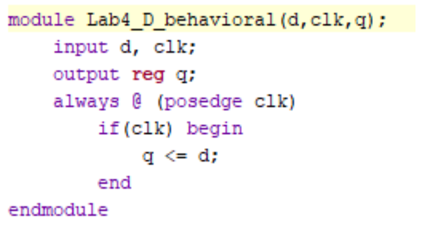
Figure 15

Task 3 and 4 were similar to the previous tasks, except these tasks used a D latch instead of the previous SR latch. Task 3 asked to use dataflow modeling for the development of this circuit, and task 4 used behavioral modeling. Both of these tasks should use similar testbenches since both of them are testing the functionality of a D latch circuit. The code for task 3 & 4 are shown in Figures 16 & 17, and the code for their respective testbenches are shown in Figures 18 & 19.



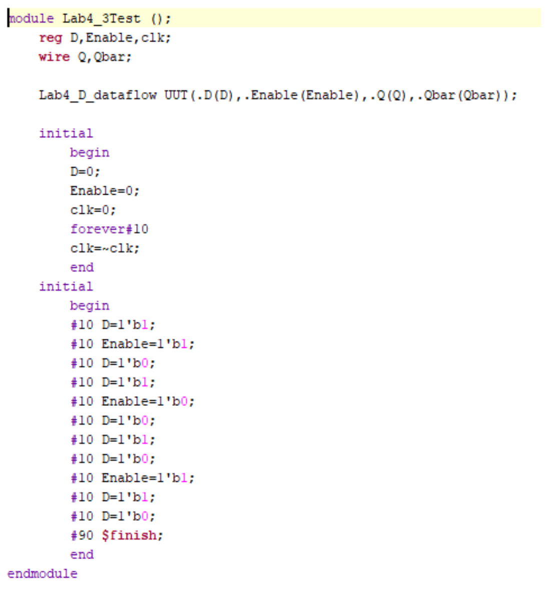
```
module Lab4_D_dataflow(Q,D,Enable,Qbar);
    input D,Enable;
    output Q,Qbar;

    assign #2 Dbar = ~D;
    assign #2 A_i = (Dbar & Enable);
    assign #2 B_i = (D & Enable);
    assign #2 Q_i = Q;
    assign #2 Qbar_i = Qbar;
    //outputs
    assign #2 Q = ~ (A_i | Qbar);
    assign #2 Qbar = ~ (B_i | Q);
endmodule
```

Figure 16



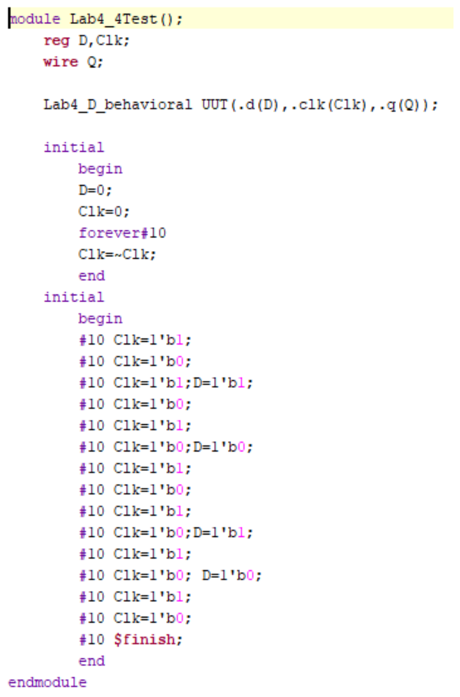
```
module Lab4_D_behavioral(d,clk,q);
    input d, clk;
    output reg q;
    always @ (posedge clk)
        if(clk) begin
            q <= d;
        end
endmodule
```

Figure 17



```
module Lab4_3Test ();
    reg D,Enable,clk;
    wire Q,Qbar;

    Lab4_D_dataflow UUT(.D(D),.Enable(Enable),.Q(Q),.Qbar(Qbar));

    initial
        begin
        D=0;
        Enable=0;
        clk=0;
        forever#10
        clk=~clk;
        end
    initial
        begin
        #10 D=1'b1;
        #10 Enable=1'b1;
        #10 D=1'b0;
        #10 D=1'b1;
        #10 Enable=1'b0;
        #10 D=1'b0;
        #10 D=1'b1;
        #10 D=1'b0;
        #10 Enable=1'b1;
        #10 D=1'b1;
        #10 D=1'b0;
        #90 $finish;
        end
endmodule
```

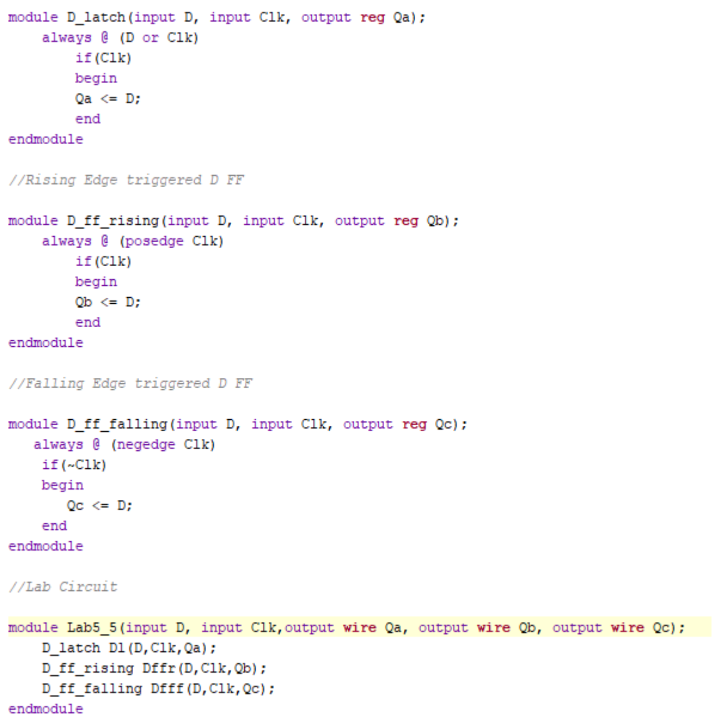
Figure 18



```
module Lab4_4Test();
    reg D,Clk;
    wire Q;

    Lab4_D_behavioral UUT(.d(D),.clk(Clk),.q(Q));

    initial
        begin
        D=0;
        Clk=0;
        forever#10
        Clk=~Clk;
        end
    initial
        begin
        #10 Clk=1'b1;
        #10 Clk=1'b0;
        #10 Clk=1'b1;D=1'b1;
        #10 Clk=1'b0;
        #10 Clk=1'b1;
        #10 Clk=1'b0;D=1'b0;
        #10 Clk=1'b1;
        #10 Clk=1'b0;
        #10 Clk=1'b1;
        #10 Clk=1'b0;D=1'b1;
        #10 Clk=1'b1;
        #10 Clk=1'b0; D=1'b0;
        #10 Clk=1'b1;
        #10 Clk=1'b0;
        #10 $finish;
        end
 endmodule
```

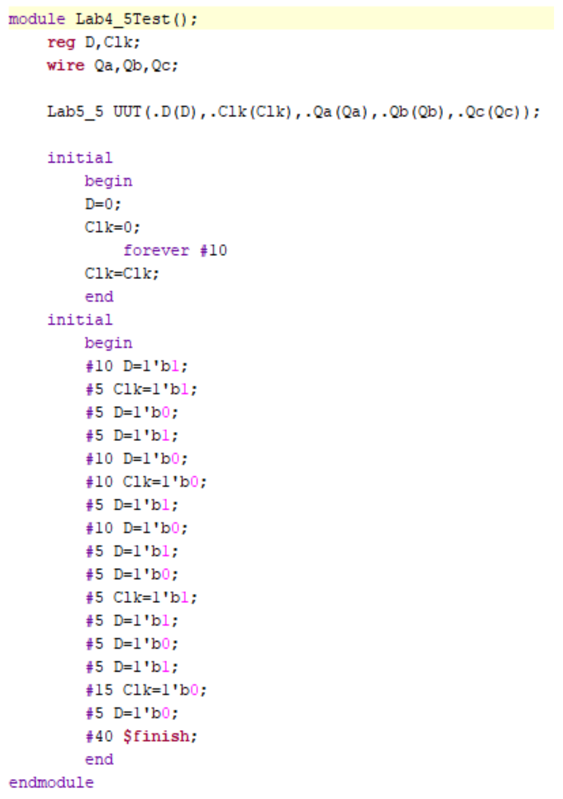
Figure 19

Task 5 was a circuit to test the differences in D latch, and D flip-flops. This task contains 3 different circuits all contained into one task, and then we view the output in the testbench made. The code for task 5 is shown in Figure 20, and the testbench is shown in Figure 21.



```
module D_latch(input D, input Clk, output reg Qa);
    always @ (D or Clk)
        if(Clk)
        begin
        Qa <= D;
        end
endmodule

//Rising Edge triggered D FF

module D_ff_rising(input D, input Clk, output reg Qb);
    always @ (posedge Clk)
        if(Clk)
        begin
        Qb <= D;
        end
endmodule

//Falling Edge triggered D FF

module D_ff_falling(input D, input Clk, output reg Qc);
   always @ (negedge Clk)
    if(~Clk)
    begin
       Qc <= D;
    end
endmodule

//Lab Circuit

module Lab5_5(input D, input Clk,output wire Qa, output wire Qb, output wire Qc);
    D_latch D1(D,Clk,Qa);
    D_ff_rising Dffr(D,Clk,Qb);
    D_ff_falling Dfff(D,Clk,Qc);
endmodule
```

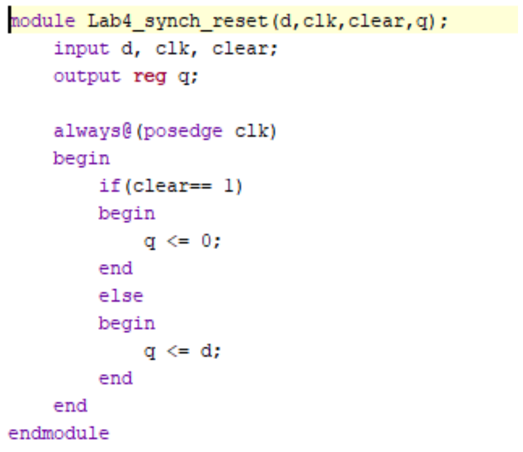
Figure 20



```
module Lab4_5Test();
    reg D,Clk;
    wire Qa,Qb,Qc;

    Lab5_5 UUT(.D(D),.Clk(Clk),.Qa(Qa),.Qb(Qb),.Qc(Qc));

    initial
        begin
        D=0;
        Clk=0;
            forever #10
        Clk=Clk;
        end
    initial
        begin
        #10 D=1'b1;
        #5 Clk=1'b1;
        #5 D=1'b0;
        #5 D=1'b1;
        #10 D=1'b0;
        #10 Clk=1'b0;
        #5 D=1'b1;
        #10 D=1'b0;
        #5 D=1'b1;
        #5 D=1'b0;
        #5 Clk=1'b1;
        #5 D=1'b1;
        #5 D=1'b0;
        #5 D=1'b1;
        #15 Clk=1'b0;
        #5 D=1'b0;
        #40 $finish;
        end
endmodule
```

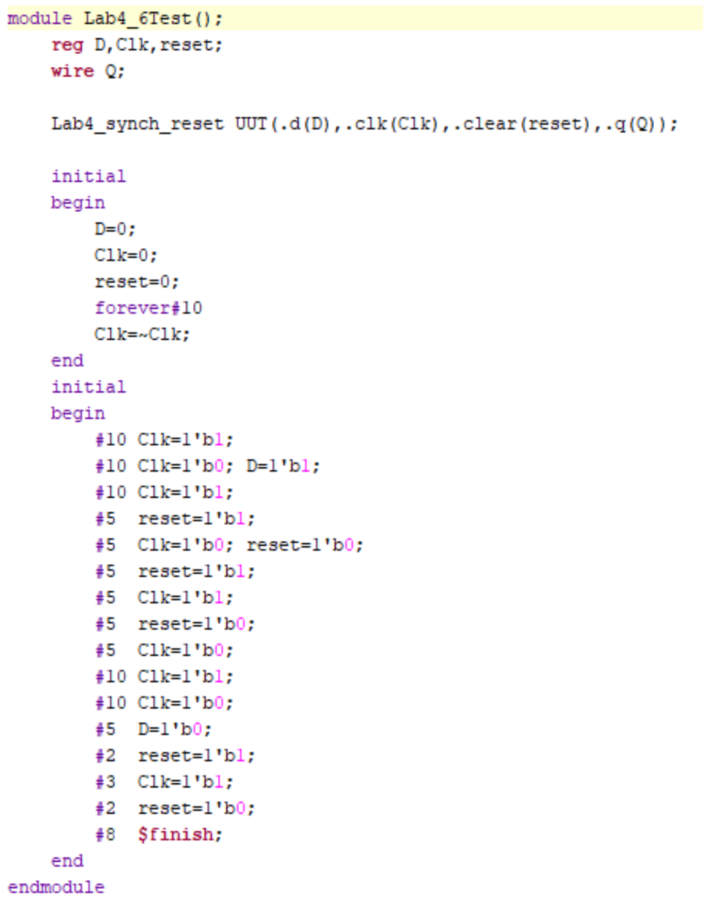
Figure 21

Task 6 begins modeling of different D flip-flops. Task 6 asked to create a flip-flop with synchronous reset using behavioral modeling. Then we develop a testbench to test the functionality of the circuit. This is similar to the circuit developed in task 5, but with the added synchronous reset. The code for task 6 is shown in Figure 22 and the testbench is shown in Figure 23.



```
module Lab4_synch_reset(d,clk,clear,q);
    input d, clk, clear;
    output reg q;

    always@(posedge clk)
    begin
        if(clear== 1)
        begin
            q <= 0;
        end
        else
        begin
            q <= d;
        end
    end
endmodule
```

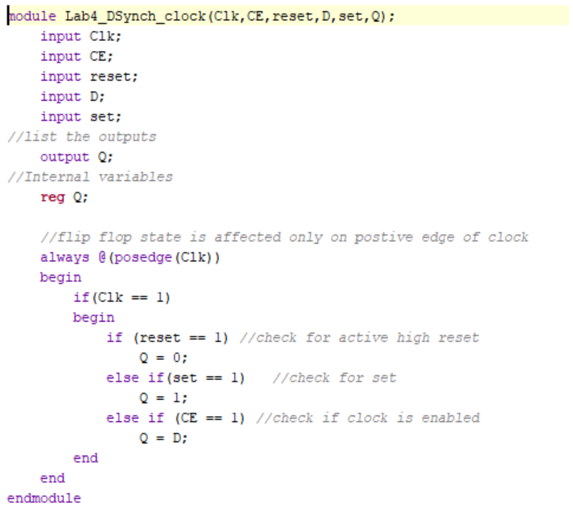
Figure 22



```
module Lab4_6Test();
    reg D,Clk,reset;
    wire Q;

    Lab4_synch_reset UUT(.d(D),.clk(Clk),.clear(reset),.q(Q));

    initial
    begin
        D=0;
        Clk=0;
        reset=0;
        forever#10
        Clk=~Clk;
    end
    initial
    begin
        #10 Clk=1'b1;
        #10 Clk=1'b0; D=1'b1;
        #10 Clk=1'b1;
        #5  reset=1'b1;
        #5  Clk=1'b0; reset=1'b0;
        #5  reset=1'b1;
        #5  Clk=1'b1;
        #5  reset=1'b0;
        #5  Clk=1'b0;
        #10 Clk=1'b1;
        #10 Clk=1'b0;
        #5  D=1'b0;
        #2  reset=1'b1;
        #3  Clk=1'b1;
        #2  reset=1'b0;
        #8  $finish;
    end
endmodule
```

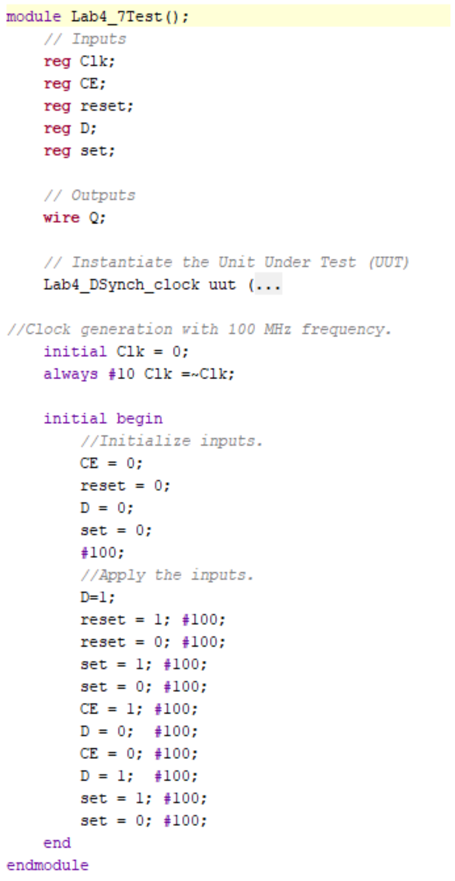
Figure 23

Task 7 used a similar circuit to task 6, except the task also used a clock enable, as well as the previous tasks synchronous reset, using behavioral modeling. This means we add another input signal to a circuit used before in task 6. The code is shown in Figure 24, and testbench is shown in Figure 25.



```
module Lab4_DSynch_clock(Clk,CE,reset,D,set,Q);
    input Clk;
    input CE;
    input reset;
    input D;
    input set;
//list the outputs
    output Q;
//Internal variables
    reg Q;

    //flip flop state is affected only on postive edge of clock
    always @(posedge(Clk))
    begin
        if(Clk == 1)
        begin
            if (reset == 1) //check for active high reset
                Q = 0;
            else if(set == 1)   //check for set
                Q = 1;
            else if (CE == 1) //check if clock is enabled
                Q = D;
        end
    end
endmodule
```

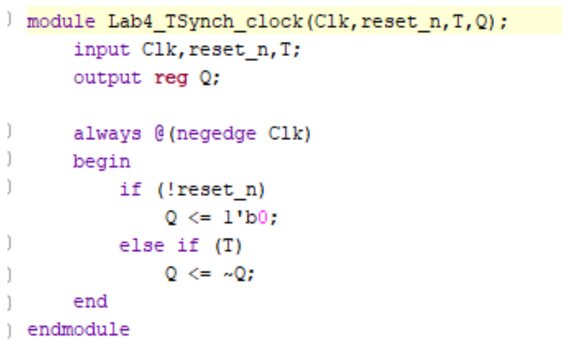
Figure 24



```
module Lab4_7Test();
    // Inputs
    reg Clk;
    reg CE;
    reg reset;
    reg D;
    reg set;

    // Outputs
    wire Q;

    // Instantiate the Unit Under Test (UUT)
    Lab4_DSynch_clock uut (...

//Clock generation with 100 MHz frequency.
    initial Clk = 0;
    always #10 Clk =~Clk;

    initial begin
        //Initialize inputs.
        CE = 0;
        reset = 0;
        D = 0;
        set = 0;
        #100;
        //Apply the inputs.
        D=1;
        reset = 1; #100;
        reset = 0; #100;
        set = 1; #100;
        set = 0; #100;
        CE = 1; #100;
        D = 0;  #100;
        CE = 0; #100;
        D = 1;  #100;
        set = 1; #100;
        set = 0; #100;
    end
endmodule
```

Figure 25

Task 8 was very different from other tasks. This task asked for a modeling of a T or Toggle flip-flop with synchronous negative-logic reset and clock enable. This code was shown in the writeup and was simply a copy and paste task. There was no simulation to verify this task, but was tested visually after uploading to the board. The code for task 8 is shown in Figure 26.



```verilog
module Lab4_TSynch_clock(Clk,reset_n,T,Q);
    input Clk,reset_n,T;
    output reg Q;

    always @(negedge Clk)
    begin
        if (!reset_n)
            Q <= 1'b0;
        else if (T)
            Q <= ~Q;
    end
endmodule
```

Figure 26

Verification:

Each task(except for task 8) is verified using a simulation testbench to verify outputs given each input. Then reading the truth tables for each of these tasks and comparing to their simulation outputs, we can verify the results. The simulations for tasks 1-7 are shown below in Figures 27-33.

Task 1:

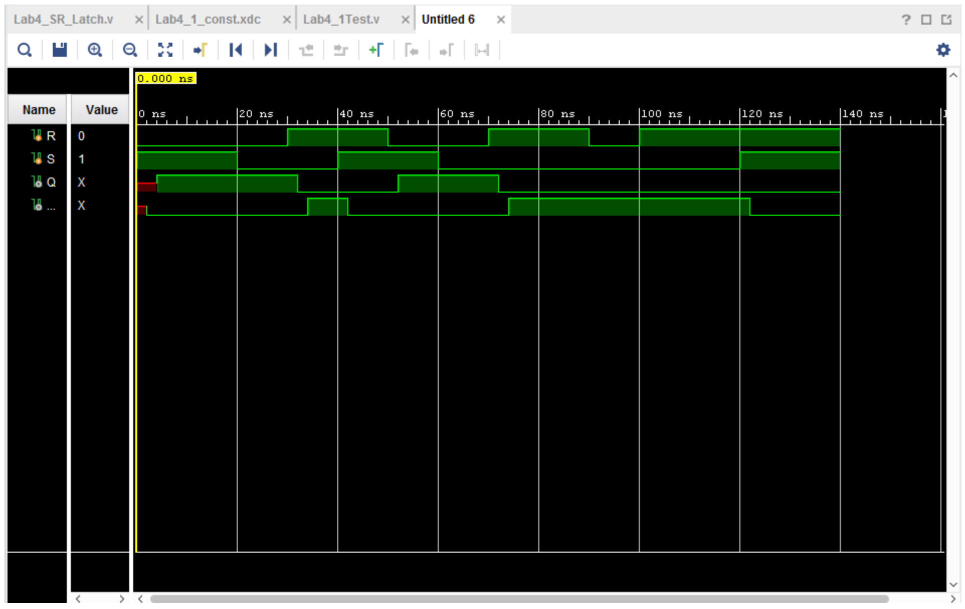


Figure 27

Task 2:

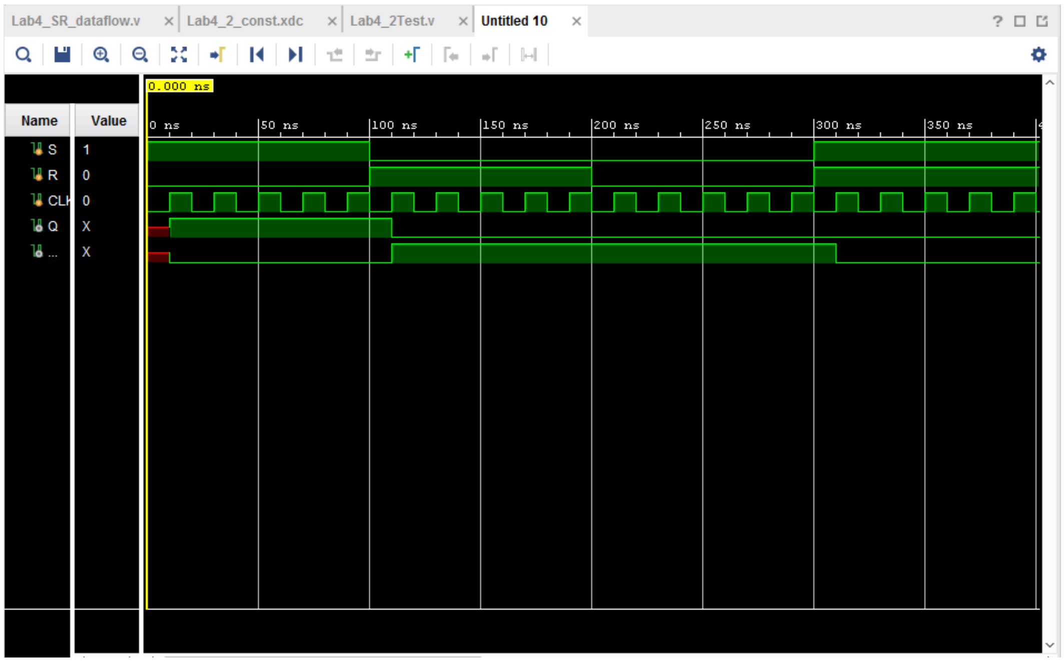


Figure 28

Task 3:

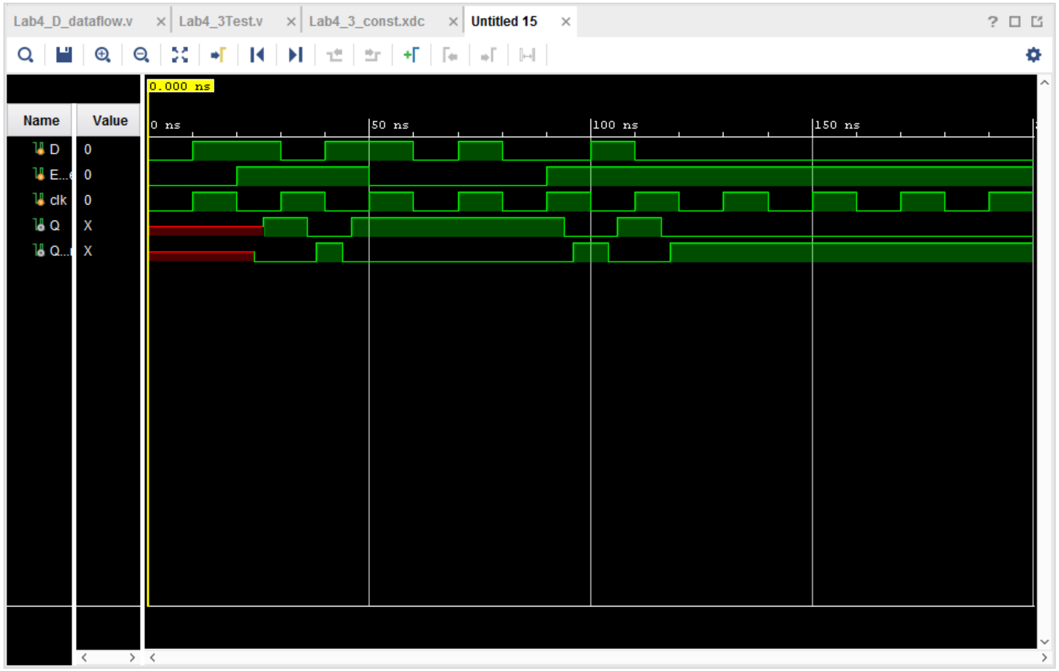


Figure 29

Task 4:

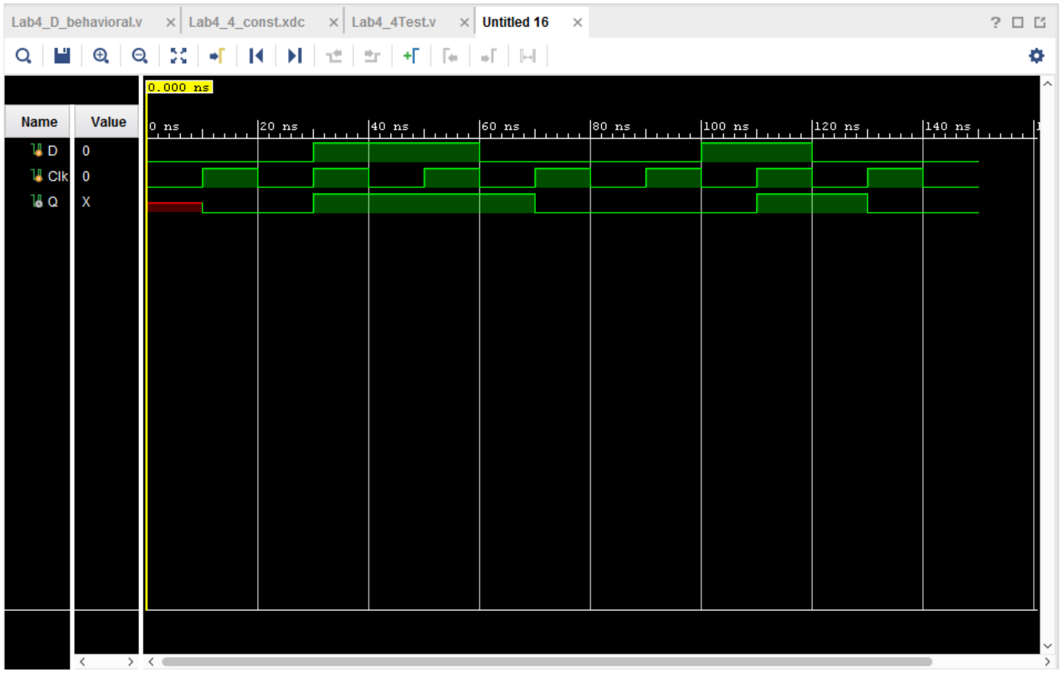


Figure 30

Task 5:

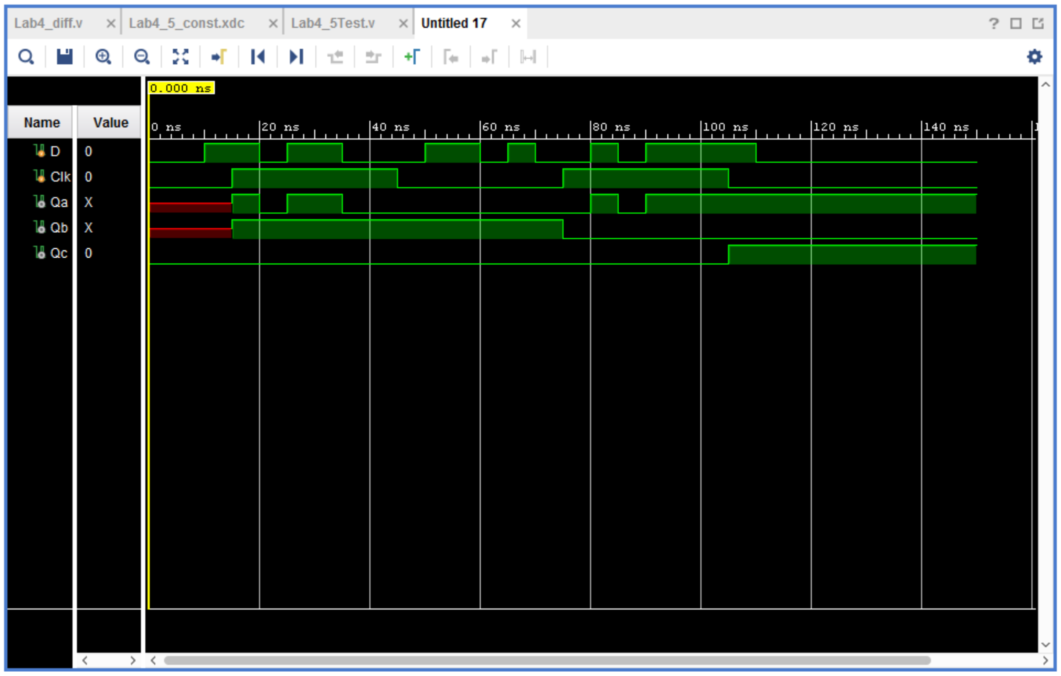


Figure 31

Task 6:

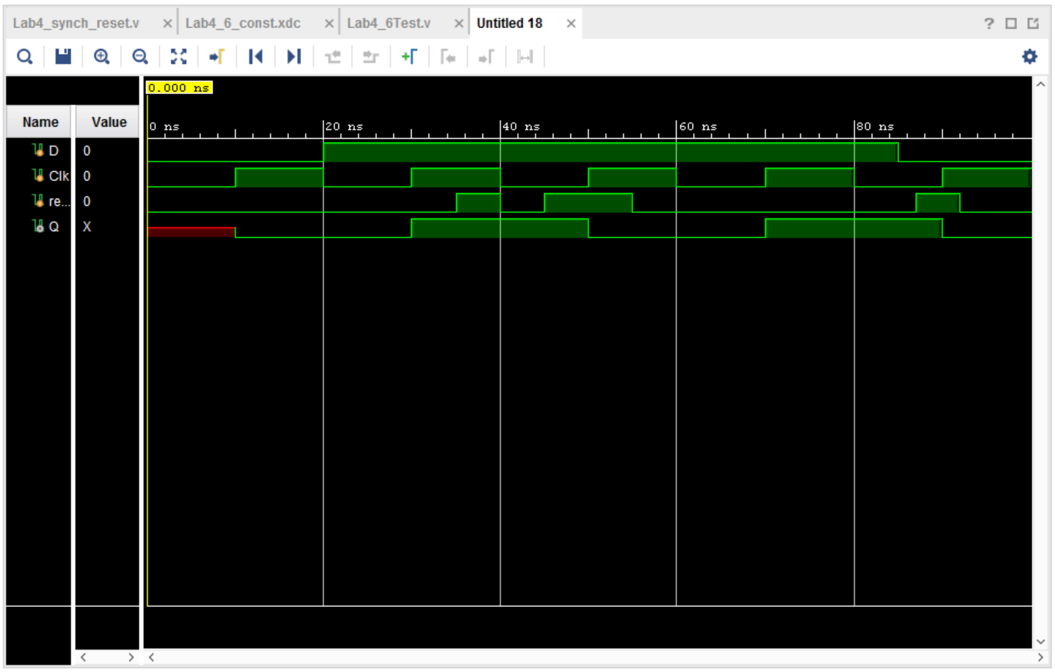


Figure 32

Task 7:

The simulation for task 7 (Figure 33) was not found due to not being able to finish this lab.

After most of these tasks seeing the simulation, tasks 1-3 & 6-8 are asked to verify by uploading to the board and visually test the output of the circuit. Then we take the outputs and compare to the corresponding truth tables shown for each task. Then by comparing and getting the same result as the truth tables, we verify our circuit physically works.

Conclusion:

What I learned in this lab is the functionalities of several different latches and flip-flops and how each one could be used differently then the others. This includes SR latch, D latch, D flip-flop (negative and positive edge triggered), then adding synchronous reset, adding clock enable, and finally a T flip-flop. We also learned a lot about writing our own testbenches and how the testbench is related with the main program file. I also struggled with finishing this lab and only made it to task 6 and did not get to verify tasks 7 and 8. This is due to struggling to get previous tasks working 100% correct, which took too much time to finish a very long lab such as this one. This lab also taught me if the program is taking too long to get correct, I could just move on and try the next one since the code I have could possibly work already. I need to find the perfect way to manage time better.