ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
LOGIC CIRCUITS LABORATORY
LATCHES AND FLIP-FLOPS

HOMEWORK NO : 2
GROUP NO : G08

GROUP MEMBERS:

150200081 : KAAN KARATAŞ
150210719 : NACİ TOYGUN GÖRMÜŞ

SPRING 2023

Contents

1. INTRODUCTION

In this homework, flip-flops are thought over in preliminary section and truth tables needed are constructed. After that in experiment section parts, requested circuits are designed.

2. Preliminary

2.1. Part 1

Flip-Flop. *Circuit element that can store data.*

A flip-flop is a key component of digital circuits and a sequential logic device capable of storing binary data that has two stable states.

2.2. Part 2

The main distinction between latches and flip flops is that flip flops are edge-triggered and only allow output changes on rising or falling edges of the clock signal, latches are level-sensitive and can modify their outputs whenever the input changes.

2.3. Part 3

SR-latch is a digital circuit with two stable states which is able to store a bit of data. Two input signals, S and R, determine the state of the latch. S changes the output to 1, and R resets it to 0.

2.4. Part 4

S	R	Q	Q^+	Function
0	0	0	0	HOLD
0	0	1	1	HOLD
0	1	0	0	RESET
0	1	1	0	RESET
1	0	0	1	SET
1	0	1	1	SET
1	1	0	X	INVALID
1	1	1	X	INVALID

Truth table of S-R latch without Enable input

2.5. Part 5

E	S	R	Q	Q^+	Function
0	0	0	0	0	HOLD
0	0	0	1	1	HOLD
0	0	1	0	0	HOLD
0	0	1	1	1	HOLD
0	1	0	0	0	HOLD
0	1	0	1	1	HOLD
0	1	1	0	0	HOLD
0	1	1	1	1	HOLD
1	0	0	0	0	HOLD
1	0	0	1	1	HOLD
1	0	1	0	0	RESET
1	0	1	1	0	RESET
1	1	0	0	1	SET
1	1	0	1	1	SET
1	1	1	0	X	INVALID
1	1	1	1	X	INVALID

Truth table of S-R latch with Enable input

2.6. Part 6

CLK	D	Q^+	Q_N^+
0	X	Q	Q_N
1	X	Q	Q_N
↑	0	0	1
↑	1	1	0

Truth table of D flip-flop

2.7. Part 7

CLK	J	K	Q^+	Q_N^+
0	X	X	Q	Q_N
1	X	X	Q	Q_N
↑	0	0	Q	Q_N
↑	0	1	0	0
↑	1	0	1	0
↑	1	1	Q_N	Q

Truth table of JK flip-flop

3. Experiments

3.1. Part 1

```
module SR_latch(input S, input R, output Q, output Q_neg);
    wire NS, NR;
    assign NS = ~S;
    assign NR = ~R;
    nand_gate2 ng3(NS, Q_neg, Q);
    nand_gate2 ng4(NR, Q, Q_neg);
endmodule
```

Figure 1: SR Latch Code

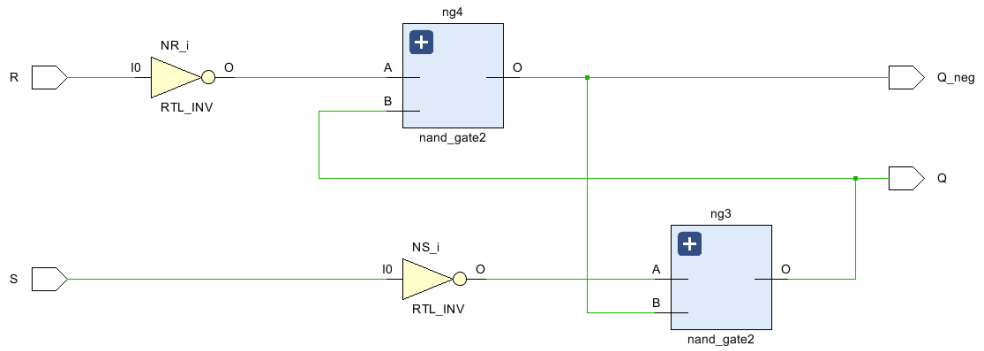

Figure 2: SR Latch Schematic

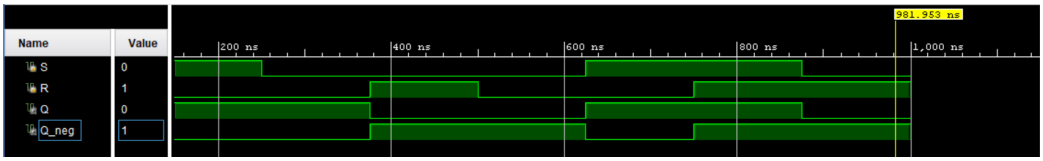

Figure 3: SR Latch Simulation

Q^+	RS 00	01	11	10
0	0	1	-	0
1	1	1	-	0

$$Q^+ = S + R'Q$$

When S and R are both 0, the output Q does not change its state and holds previous value. The output Q is 0 when S is 0 and R is 1. Q is 1 when S is 1 and R is 0. Forbidden input (1, 1) results in an unpredictable output.

3.2. Part 2

```
module SR_latch_E(input S, input R, input E, output Q, output Q_neg);
     wire SE, RE;
     nand_gate2 ng1(S, E, SE);
     nand_gate2 ng2(R, E, RE);
     nand_gate2 ng3(SE, Q_neg, Q);
     nand_gate2 ng4(RE, Q, Q_neg);
endmodule
```

Figure 4: SR Latch With Enable Code

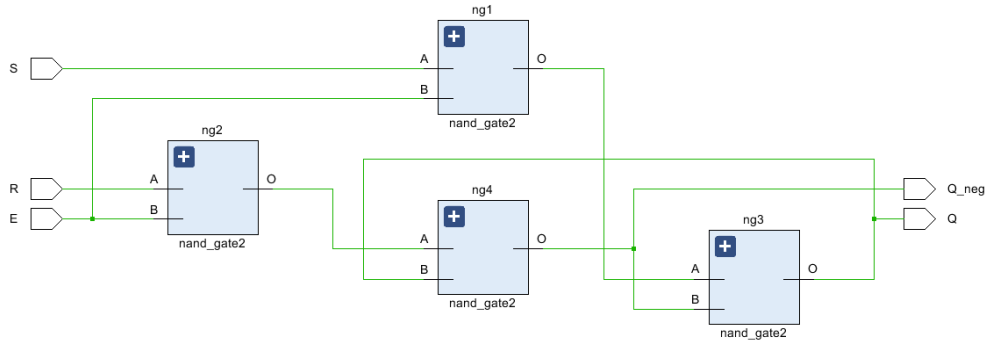

Figure 5: SR Latch With Enable Schematic

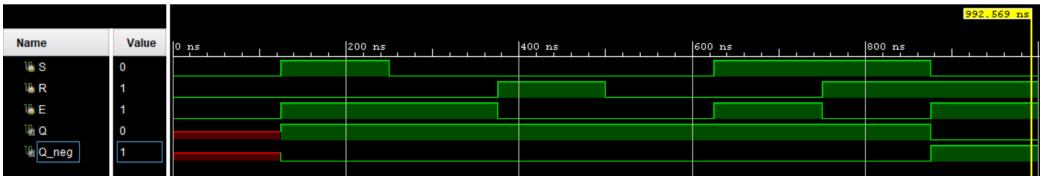

Figure 6: SR Latch With Enable Simulation

Q^+E \ RS	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	1	-	-
10	0	1	0	0

$$Q^+ = E'Q + R'Q + ES$$

The equation for Q^+ is found using the Karnaugh Map above. When S and R are both 1 (forbidden input), both Q and Q_n becomes 1. If the latch is disabled in this state, the output will be indeterminate.

3.3. Part 3

```
module D_latch_E(input D, input E, output Q, output Q_neg);
    wire ND, NDE, DE;
    assign ND = ~D;
    nand_gate2 ng1(D, E, DE);
    nand_gate2 ng2(ND, E, NDE);
    nand_gate2 ng3(DE, Q_neg, Q);
    nand_gate2 ng4(NDE, Q, Q_neg);
endmodule

 module NE_D_flipflop(input D, input CLK, output Q, output Q_neg);
    wire NCLK, firstD, empty;
    assign NCLK = ~CLK;
    D_latch_E l1(D, CLK, firstD, empty);
    D_latch_E l2(firstD, NCLK, Q, Q_neg);
 endmodule
```

Figure 7: Negative Edge Triggered D Flip Flop Code

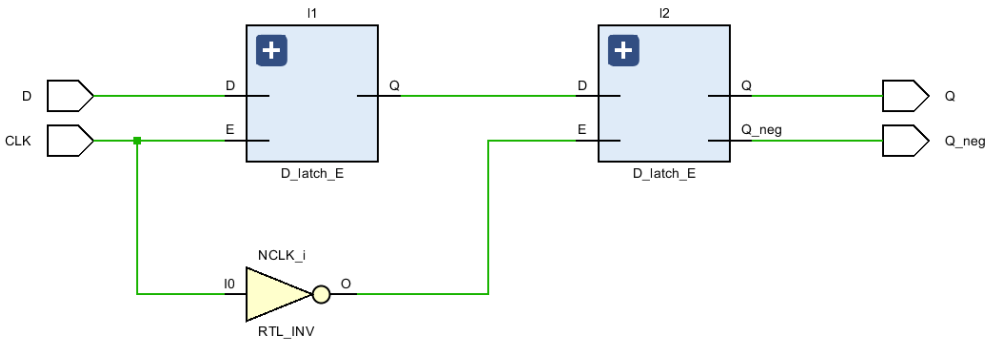

Figure 8: Negative Edge Triggered D Flip Flop Schematic

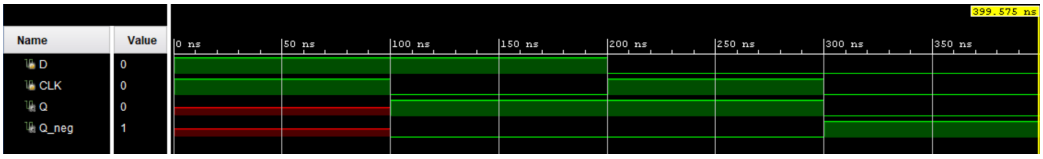

Figure 9: Negative Edge Triggered D Flip Flop Simulation

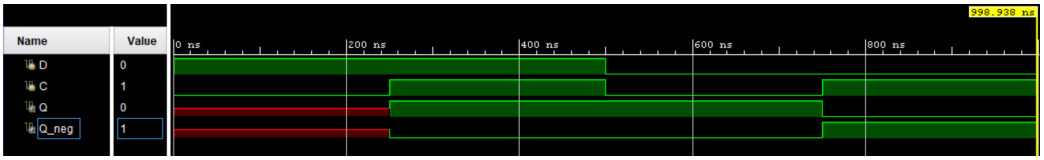

Figure 10: D Latch Simulation

$$Q^+ = D$$

It is clear from simulation that D input determines the subsequent state. The D latch equation is therefore Q(t+1) = D.

3.4. Part 4

```
module PE_JK_flipflop(input J, input K, input CLK, output Q, output Q_neg);
    wire JQ, KQ, NK, JKQ;
    nand_gate2 ng1(J, Q_neg, JQ);

    nand_gate2 ng2(K, K, NK);
    nand_gate2 ng3(NK, Q, KQ);

    nand_gate2 ng4 (JQ, KQ, JKQ);
    NE_D_flipflop dff(JKQ, CLK, Q, Q_neg);
endmodule
```

Figure 11: JK Flip Flop Code

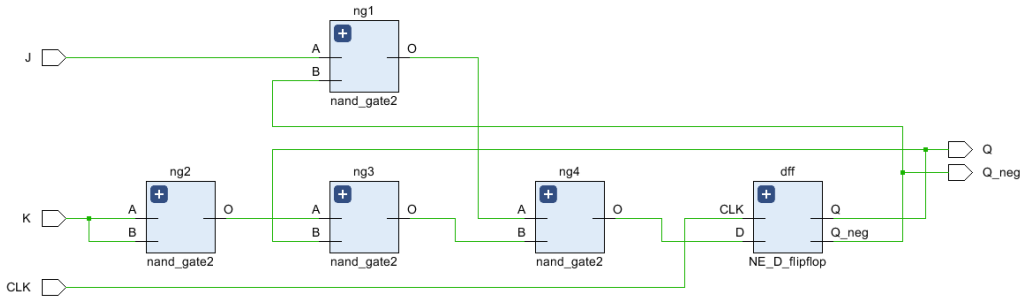

Figure 12: JK Flip Flop Schematic

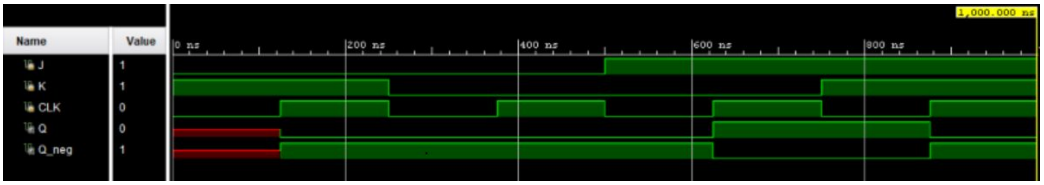

Figure 13: JK Flip Flop Simulation

3.5. Part 5

```
module async_up_counter(input J,input K,input CLK,output[3:0] Q, output[3:0]Q_neg );
    wire in1, in2, in3, in4, w1, w2, w3, w4, w5, w6, w7;
    wire nreset, in;

    nand_gate2 ng1(J, K, nreset);
    nand_gate2 ng2(nreset, nreset,in);
    PE_JK_flipflop ff1(J, K, CLK, Q[0], Q_neg[0]);

    nand_gate2 ng3(nreset, CLK, w1);
    nand_gate2 ng4(in, Q_neg[0], w2);
    nand_gate2 ng5(w1, w2, in1);
    PE_JK_flipflop ff2(J, K, in1, Q[1],Q_neg[1]);

    nand_gate2 ng6(nreset, CLK, w3);
    nand_gate2 ng7(in, Q_neg[1], w4);
    nand_gate2 ng8(w3, w4, in2);
    PE_JK_flipflop ff3(J, K, in2, Q[2], Q_neg[2]);

    nand_gate2 ng9(nreset, CLK, w5);
    nand_gate2 ng10(in, Q_neg[2], w6);
    nand_gate2 ng11(w5, w6, in3);
    PE_JK_flipflop ff4(J, K, in3, Q[3], Q_neg[3]);
endmodule
```

Figure 14: Asynchronous Up Counter Code

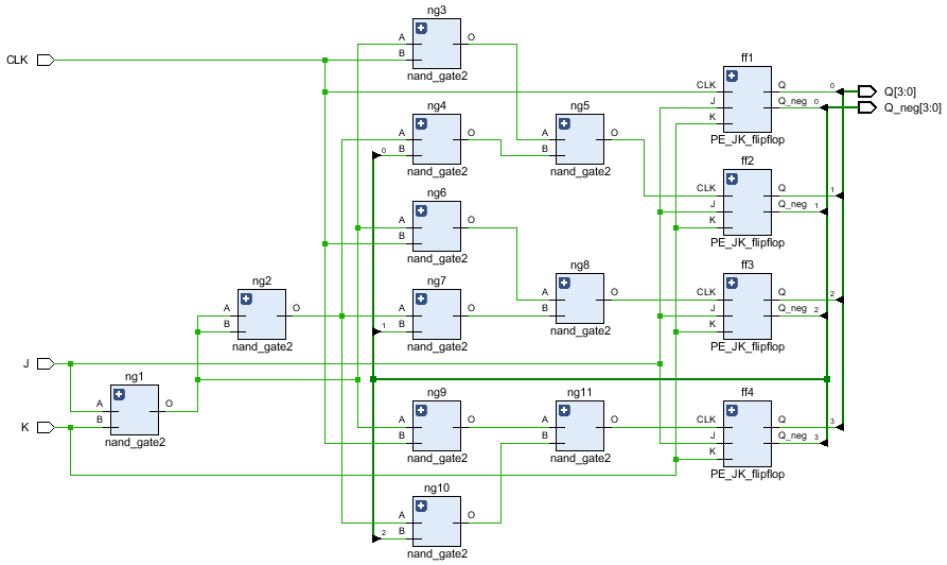

Figure 15: Asynchronous Up Counter Schematic

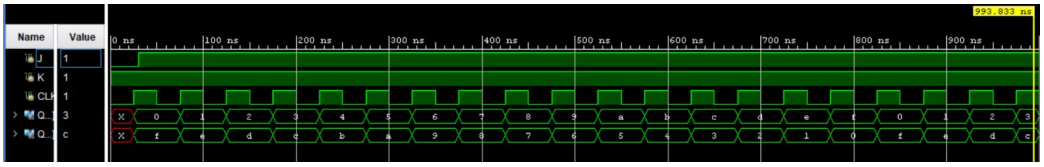

Figure 16: Asynchronous Up Counter Simulation

Count	Q_3(MSB)	Q_2	Q_1	Q_0(LSB)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

Truth Table of Asynchronous Up Counter

3.6. Part 6

```
module sync_up_counter(input J, input K, input CLK, output[3:0]Q, output[3:0]Q_neg);
    wire w1, w2;
    PE_JK_flipflop ff1(J, K, CLK, Q[0], Q_neg[0]);

    PE_JK_flipflop ff2(Q[0], Q[0], CLK, Q[1] ,Q_neg[1]);
    assign w1 = Q[0] & Q[1];

    PE_JK_flipflop ff3(w1, w1, CLK, Q[2] ,Q_neg[2]);
    assign w2 = w1 & Q[2];

    PE_JK_flipflop ff4(w2, w2, CLK, Q[3], Q_neg[3]);
endmodule
```

Figure 17: Synchronous Up Counter Code

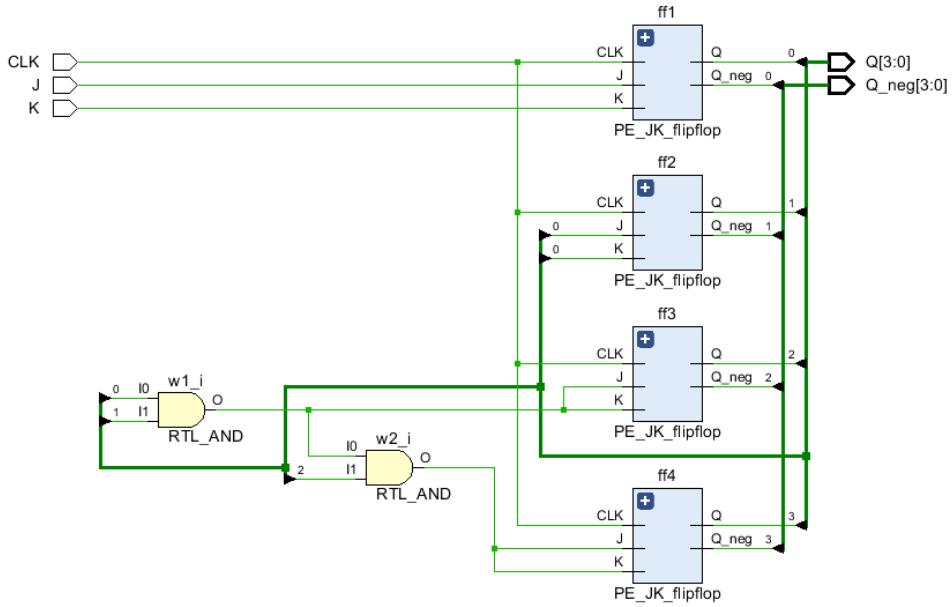

Figure 18: Synchronous Up Counter Schematic

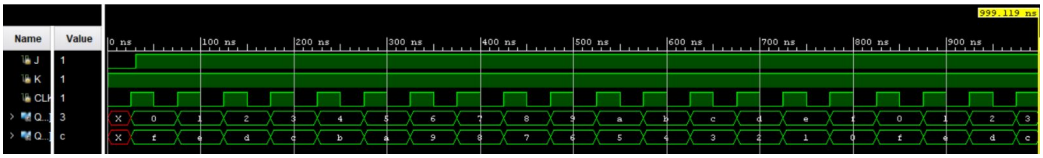

Figure 19: Synchronous Up Counter Simulation

Count	Q_3(MSB)	Q_2	Q_1	Q_0(LSB)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

Truth Table of Synchronous Up Counter

3.7. Part 7

```
module pulse_generator(input [15:0] in_data, input load, input CLK, output reg Q);
        reg [15:0] shift_register;

        always @(posedge CLK) begin
            if(load) begin
                Q = 1'b0;
                shift_register <= in_data;
            end
            else begin
                shift_register <= {shift_register[14:0], shift_register[15]};
            end
                assign Q =  shift_register[15];
        end
endmodule
```

Figure 20: Pulse Generator Code

```
module pulse_generator_test();
    reg [15:0]in_data;
    reg load, CLK;
    wire Q;
    pulse_generator uut(in_data, load, CLK, Q);
    always #2.5 CLK = ~CLK;
    initial begin
        CLK = 0;
        in_data = 16'b1010101010101010;
    end

    initial begin
        load = 1;
        in_data = 16'b1010101010101010; #6;
        load = 0; #119;
        load = 1;
        in_data = 16'b1100110011001100; #6;
        load = 0; #119;
        load = 1;
        in_data = 16'b1111000011110000; #6;
        load = 0; #119;
        load = 1;
        in_data = 16'b1000000010000000; #6;
        load = 0; #119;
        load = 1;
        in_data = 16'b1110000000000000; #6;
        load = 0; #119;
        load = 1;
        in_data = 16'b1111111111000000; #6;
        load = 0; #119;
        $finish;
    end
endmodule
```


Figure 21: Pulse Generator Simulation Code

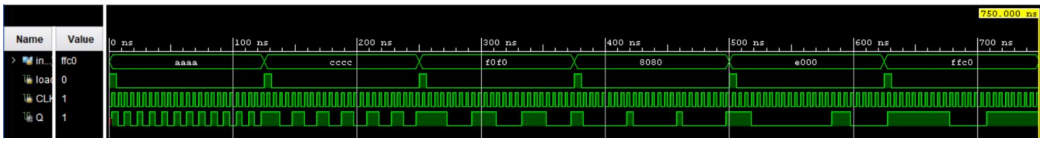

Figure 22: Pulse Generator Simulation

The provided 16-bit input value is loaded when the load input is 1, along with the clock signal. The output is the most significant bit of the loaded value. And circular shift operation is completed when the load input is 1, along with the clock signal.

4. RESULTS

Every circuit in the homework assignment is implemented using Verilog. The test results supported what we had predicted based on our calculations on paper. All test results are given on the relevant section.

5. DISCUSSION

With the help of the verilog programming language, we learned how to implement and use various latches and flip flops through this homework. Using the clock signal, we also learned how to simulate these circuits. Additionally, we learned how to implement frequency design and create synchronous and asynchronous counters. Our outcomes were fully in line with what we had anticipated.

6. CONCLUSION

With the help of the verilog programming language, we were able to simulate flip-flops, latches and counters and learned how to implement them through this homework. To sum up, this was a rather lengthy and challenging homework assignment for us, but we were able to successfully complete all of the parts.