ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	5
EXPERIMENT DATE	:	16.04.2021
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G17

GROUP MEMBERS:

150180024	:	ŞULE BEYZA KARADAĞ
150190710	:	SENİHA SERRA BOZKURT
150190024	:	AHMET FURKAN KAVRAZ

SPRING 2021

Contents

1 INTRODUCTION [10 points]

In the following experiment, we are going to implement sequential logic circuits by using the Verilog. We refreshed our knowledge on following topics to implement this experiment easier;

1. How to analyze a given sequential circuit,
2. How to design circuits with Mealy and Moore models.

We used 7th and 8th slides of BLG 231E-Digital Circuits course for this experiment. You can see the corresponding references in the Ref. section below.

2 MATERIALS AND METHODS [40 points]

2.1 Preliminary

2.1.1 Experiment: Part-1

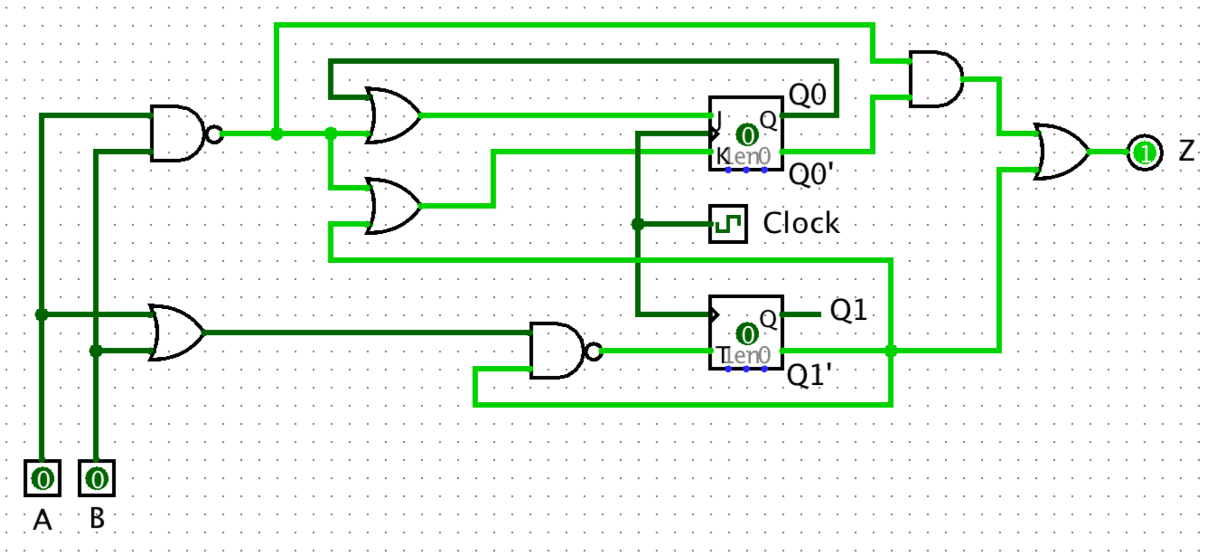


Figure 1: Sequential circuit with flip flops

We analyzed the circuit given in Figure 1. As you can see below, we obtained the equations that represents the next states of the flip flops and output Z with using the characteristic equations of the flip flops in order to create the state transition table and the state chart.

J-K Flip Flop: $\mathbf{J} = Q_0 + (A.B)' = Q_0 + A' + B'$

$\mathbf{K} = Q_1' + (A.B)' = Q_1' + A' + B'$

J-K Flip Flop Characteristic Equation: $Q_0^+ = (J.Q_0') + (K'.Q_0)$

Replace all J and K variables with above values:

$$\begin{aligned}
Q_0^+ &= (Q_0 + A' + B').Q_0' + (Q_1' + A' + B')'.Q_0 \\
&= (Q_0.Q_0') + (A'.Q_0') + (B'.Q_0') + (Q_1.A.B).Q_0 \\
&= 0 + (A'.Q_0') + (B'.Q_0') + Q_1.A.B.Q_0 \\
&= (A' + B').Q_0' + Q_1.A.B.Q_0
\end{aligned}$$

T Flip Flop: $\mathbf{T} = (A + B).Q_1'$

J-K Flip Flop Characteristic Equation: $Q_1^+ = T.Q_1' + T'.Q_1$

Replace T variable with above value:

$$\begin{aligned}
Q_1^+ &= ((A + B).Q_1').Q_1' + ((A + B).Q_1')'.Q_1 \\
&= (A + B).Q_1' + (A.Q_1' + B.Q_1')'.Q_1 \\
&= A.Q_1' + B.Q_1' + (A.Q_1')'.(B.Q_1')'.Q_1 \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).(B' + Q_1).Q_1 \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).(B'.Q_1 + Q_1.Q_1) \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).(B'.Q_1 + Q_1) \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).(B'.Q_1 + 1.Q_1) \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).(B' + 1).Q_1 \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).1.Q_1 \\
&= A.Q_1' + B.Q_1' + (A' + Q_1).Q_1 \\
&= A.Q_1' + B.Q_1' + (A'.Q_1 + Q_1.Q_1) \\
&= A.Q_1' + B.Q_1' + (A'.Q_1 + Q_1) \\
&= A.Q_1' + B.Q_1' + (A'.Q_1 + 1.Q_1) \\
&= A.Q_1' + B.Q_1' + (A' + 1).Q_1 \\
&= A.Q_1' + B.Q_1' + Q_1
\end{aligned}$$

Z Output: $\mathrm{Z} = Q_1' + Q_0'.(A.B)'$

$$\begin{aligned}
&= Q_1' + Q_0'.(A' + B') \\
&= Q_1' + Q_0'.A' + Q_0'.B'
\end{aligned}$$

2.2 Experiment

2.2.1 Part 1

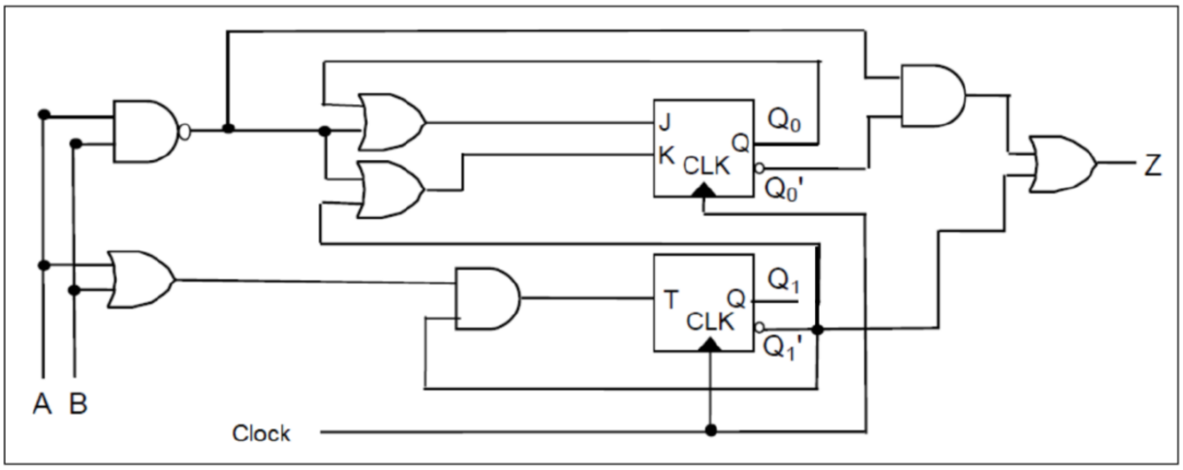


Figure 2: Sequential circuit with flip flops

In this part, we implemented the circuits in Figure 2 given to us in the pdf by using Verilog. We used different type flip flop modules, logic gates and clock signal to synchronize the circuit.

As you can see below, we created the state transition table for this circuit and state diagram graphically according to equations that we obtained in **Preliminary-1** section.

$Q_1^+Q_0^+$,Z	AB			
Q_1Q_0	00	01	10	11
00	01,1	11,1	11,1	10,1
01	00,1	10,1	10,1	10,1
10	11,1	11,1	11,1	10,0
11	10,0	10,0	10,0	11,0

Table 1: State transition table

S^+,Z	AB			
S	00	01	10	11
X	Y,1	W,1	W,1	Z,1
Y	X,1	Z,1	Z,1	Z,1
Z	W,1	W,1	W,1	Z,0
W	Z,0	Z,0	Z,0	W,0

Table 2: State chart

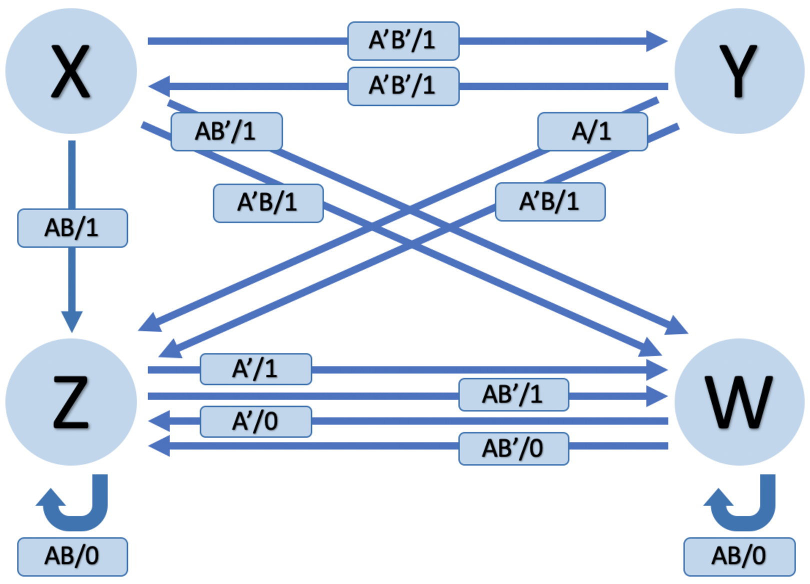
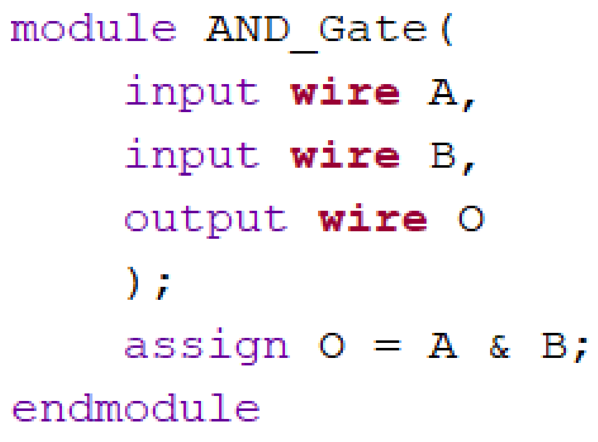


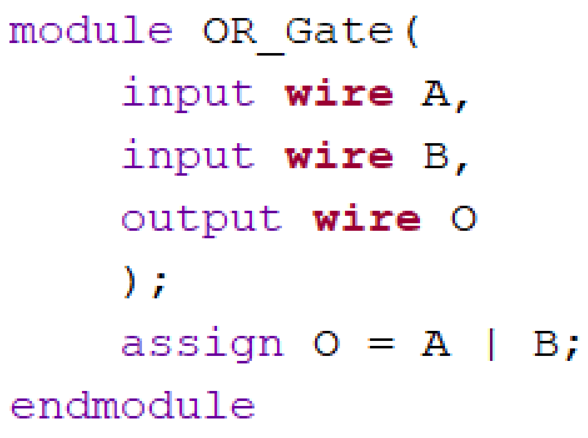
Figure 3: State diagram

RELATED GATES & FLIP-FLOPS & LATCHES WE USED IN IMPLEMENTATIONS:



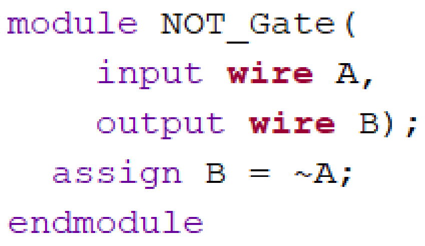
```
module AND_Gate(
    input wire A,
    input wire B,
    output wire O
    );
    assign O = A & B;
endmodule
```

(a) AND Gate



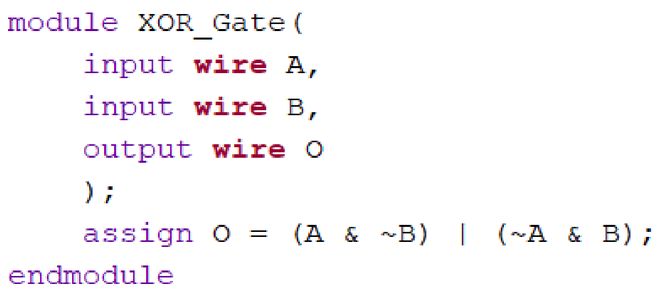
```
module OR_Gate(
    input wire A,
    input wire B,
    output wire O
    );
    assign O = A | B;
endmodule
```

(b) OR Gate



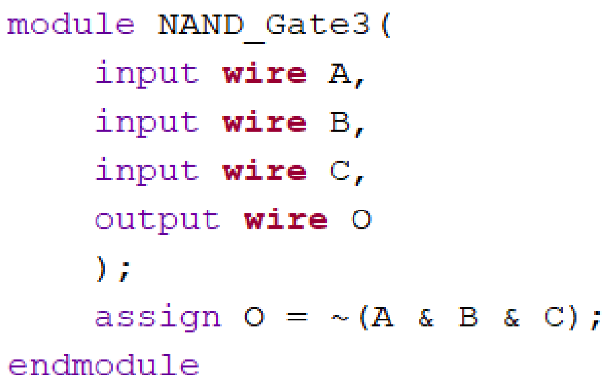
```verilog
module NOT_Gate(
     input wire A,
     output wire B);
   assign B = ~A;
endmodule
```

(a) NOT Gate



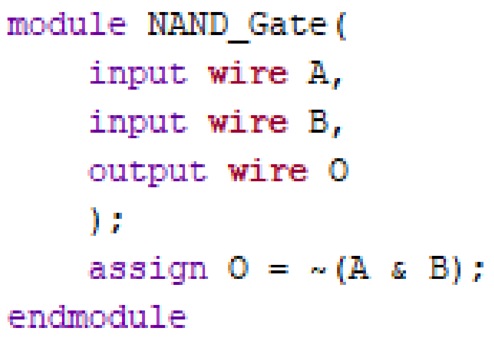
```verilog
module XOR_Gate(
    input wire A,
    input wire B,
    output wire O
    );
    assign O = (A & ~B) | (~A & B);
endmodule
```

(b) XOR Gate



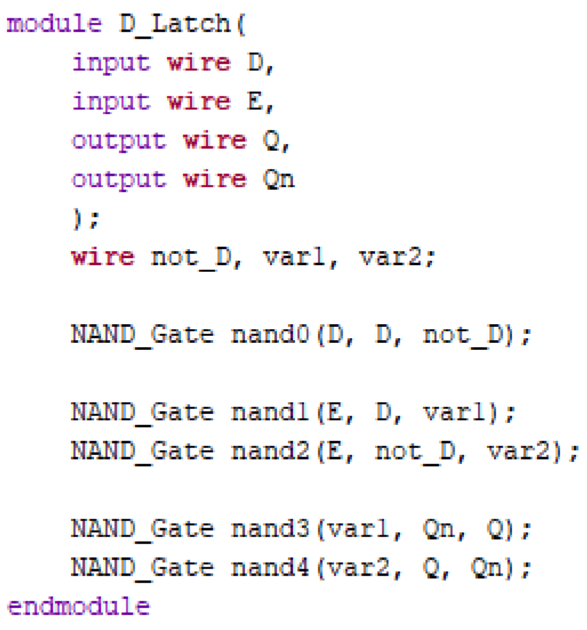
```verilog
module NAND_Gate3(
     input wire A,
     input wire B,
     input wire C,
     output wire O
     );
     assign O = ~(A & B & C);
endmodule
```

(a) NAND Gate W/ 3 inputs



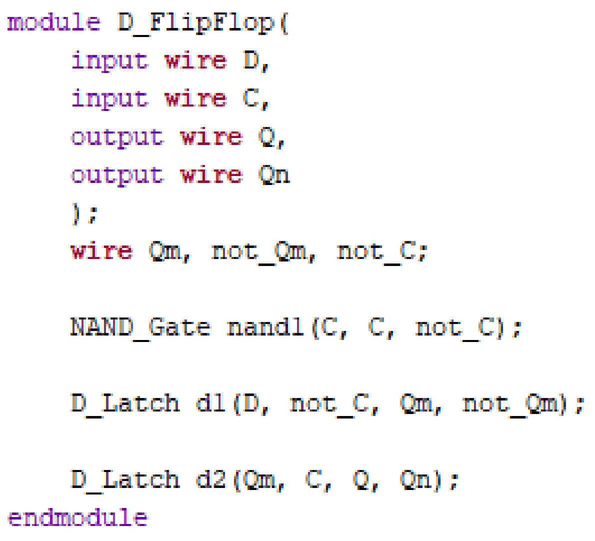
```verilog
module NAND_Gate(
     input wire A,
     input wire B,
     output wire O
     );
     assign O = ~(A & B);
endmodule
```

(b) NAND Gate W/ 2 inputs



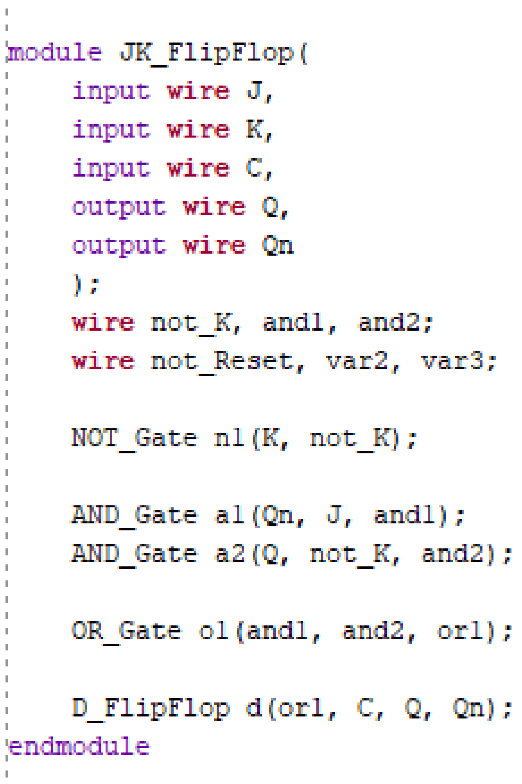
```verilog
module D_Latch(
    input wire D,
    input wire E,
    output wire Q,
    output wire Qn
    );
    wire not_D, var1, var2;

    NAND_Gate nand0(D, D, not_D);

    NAND_Gate nand1(E, D, var1);
    NAND_Gate nand2(E, not_D, var2);

    NAND_Gate nand3(var1, Qn, Q);
    NAND_Gate nand4(var2, Q, Qn);
endmodule
```

(a) D latch



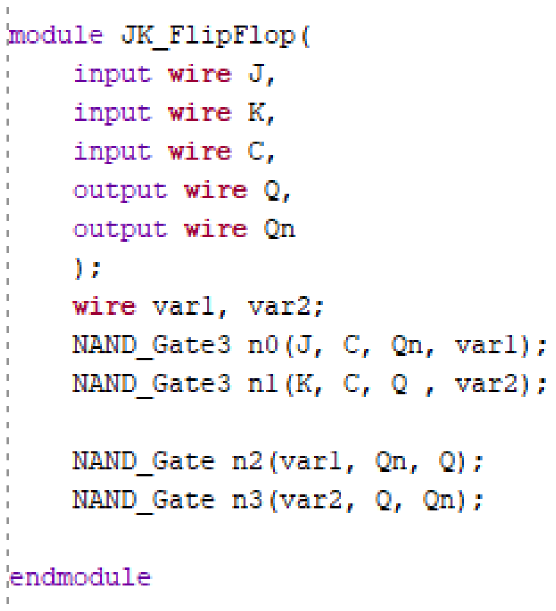
```verilog
module D_FlipFlop(
    input wire D,
    input wire C,
    output wire Q,
    output wire Qn
    );
    wire Qm, not_Qm, not_C;

    NAND_Gate nand1(C, C, not_C);

    D_Latch d1(D, not_C, Qm, not_Qm);

    D_Latch d2(Qm, C, Q, Qn);
endmodule
```

(b) D Flip-flop using D latches



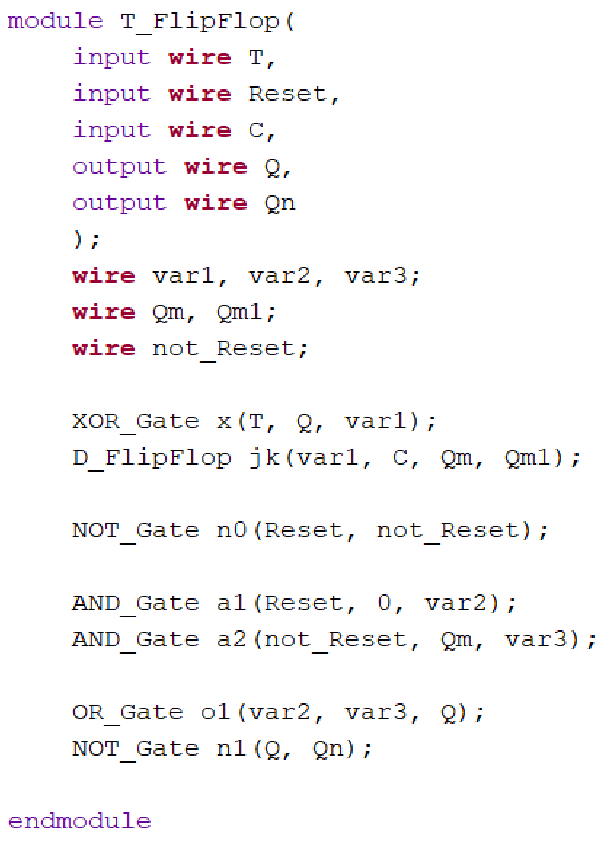
```
module JK_FlipFlop(
    input wire J,
    input wire K,
    input wire C,
    output wire Q,
    output wire Qn
    );
    wire not_K, and1, and2;
    wire not_Reset, var2, var3;

    NOT_Gate n1(K, not_K);

    AND_Gate a1(Qn, J, and1);
    AND_Gate a2(Q, not_K, and2);

    OR_Gate o1(and1, and2, or1);

    D_FlipFlop d(or1, C, Q, Qn);
endmodule
```

(a) JK1 Flip-flop



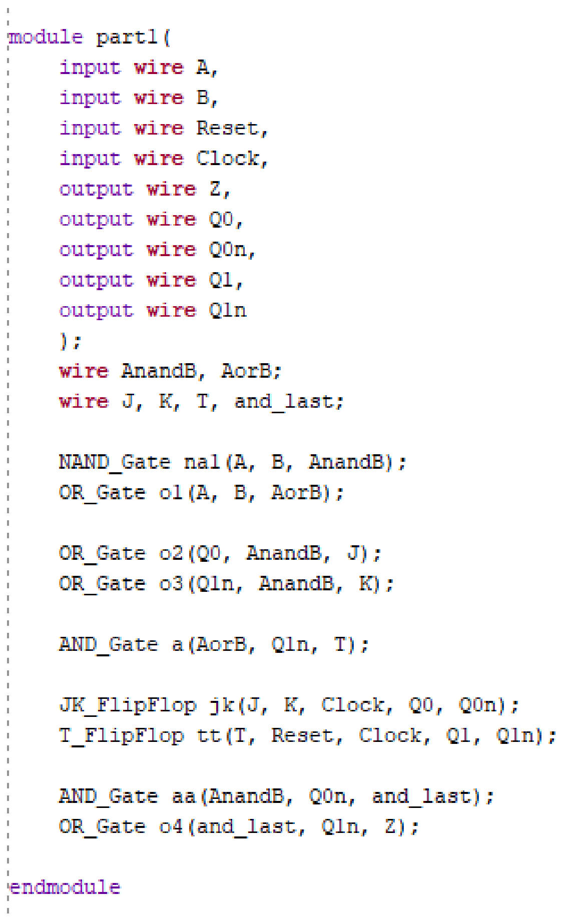
```
module JK_FlipFlop(
    input wire J,
    input wire K,
    input wire C,
    output wire Q,
    output wire Qn
    );
    wire var1, var2;
    NAND_Gate3 n0(J, C, Qn, var1);
    NAND_Gate3 n1(K, C, Q , var2);

    NAND_Gate n2(var1, Qn, Q);
    NAND_Gate n3(var2, Q, Qn);

endmodule
```

(b) JK2 Flip-flop



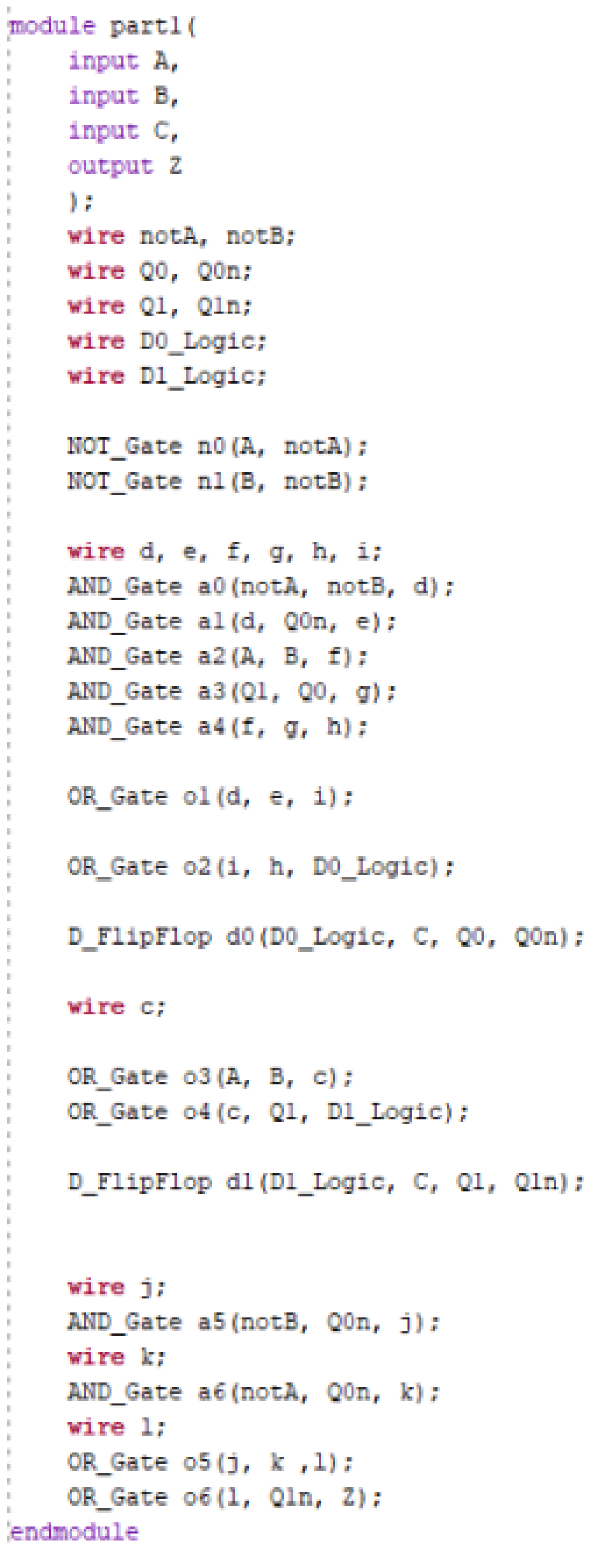
```
module T_FlipFlop(
    input wire T,
    input wire Reset,
    input wire C,
    output wire Q,
    output wire Qn
    );
    wire var1, var2, var3;
    wire Qm, Qm1;
    wire not_Reset;

    XOR_Gate x(T, Q, var1);
    D_FlipFlop jk(var1, C, Qm, Qm1);

    NOT_Gate n0(Reset, not_Reset);

    AND_Gate a1(Reset, 0, var2);
    AND_Gate a2(not_Reset, Qm, var3);

    OR_Gate o1(var2, var3, Q);
    NOT_Gate n1(Q, Qn);

endmodule
```

(a) T Flip-flop



```
module part1(
    input wire A,
    input wire B,
    input wire Reset,
    input wire Clock,
    output wire Z,
    output wire Q0,
    output wire Q0n,
    output wire Q1,
    output wire Q1n
    );
    wire AandB, AorB;
    wire J, K, T, and_last;

    NAND_Gate na1(A, B, AandB);
    OR_Gate o1(A, B, AorB);

    OR_Gate o2(Q0, AandB, J);
    OR_Gate o3(Q1n, AandB, K);

    AND_Gate a(AorB, Q1n, T);

    JK_FlipFlop jk(J, K, Clock, Q0, Q0n);
    T_FlipFlop tt(T, Reset, Clock, Q1, Q1n);

    AND_Gate aa(AandB, Q0n, and_last);
    OR_Gate o4(and_last, Q1n, Z);

endmodule
```

(b) First Code of Part 1



```
module part1(
    input A,
    input B,
    input C,
    output Z
    );
    wire notA, notB;
    wire Q0, Q0n;
    wire Q1, Q1n;
    wire D0_Logic;
    wire D1_Logic;

    NOT_Gate n0(A, notA);
    NOT_Gate n1(B, notB);

    wire d, e, f, g, h, i;
    AND_Gate a0(notA, notB, d);
    AND_Gate a1(d, Q0n, e);
    AND_Gate a2(A, B, f);
    AND_Gate a3(Q1, Q0, g);
    AND_Gate a4(f, g, h);

    OR_Gate o1(d, e, i);

    OR_Gate o2(i, h, D0_Logic);

    D_FlipFlop d0(D0_Logic, C, Q0, Q0n);

    wire c;

    OR_Gate o3(A, B, c);
    OR_Gate o4(c, Q1, D1_Logic);

    D_FlipFlop d1(D1_Logic, C, Q1, Q1n);


    wire j;
    AND_Gate a5(notB, Q0n, j);
    wire k;
    AND_Gate a6(notA, Q0n, k);
    wire l;
    OR_Gate o5(j, k ,l);
    OR_Gate o6(l, Q1n, Z);
endmodule
```

Figure 10: Second Code of Part 1

In this part, we tried to implement the circuit that is given to us by using Verilog in **3 different ways** by using 2 different JK Flip-Flops. We couldn't manage to succeed, but we would really appreciate a debugging or at least an explanation about this simulation results.

The gates, latches and flip flops are nearly same with our past experiments. We just copy and get them. We implement JK and T Flip Flops in this experiment. We implement 2 JK Flip Flops, however both of them does not run properly. So, our part 1 code does not run properly even it is correct.

We use different types of JK Flip Flop implementations and different implementations of Part1. But the both ways, as you can see in the Figure 12, results of JK Flip Flop is in High Empedans. But when we simulate JK Flip Flop normally, it gives no error in simulation as you can see in Figure 11. We can not find the problem that "Why JK Flip Flop is not run properly in Part1?", but our code is running properly except the JK Flip Flop problem. If we have a JK Flip Flop which is running properly, our simulation results will be as in the state diagram, (see **here**).

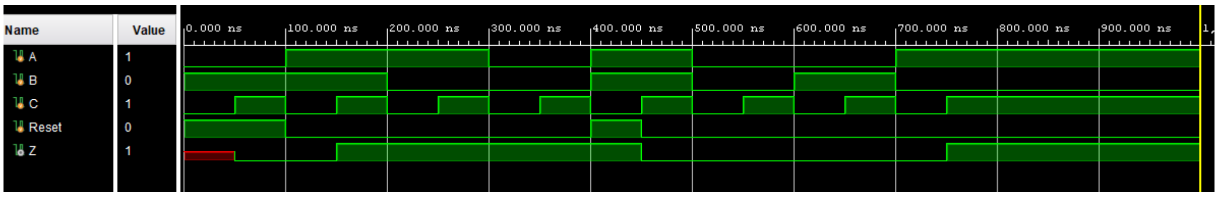


Figure 11: Simulation result of the JK Flip Flop

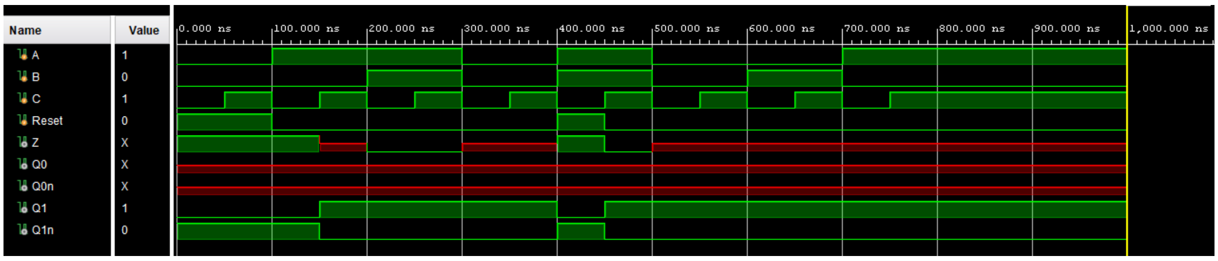


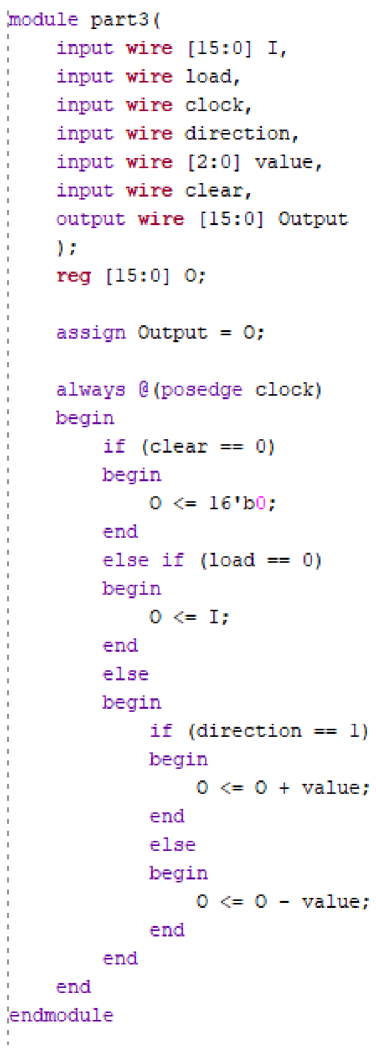
Figure 12: Simulation result of the circuit given in Part 1

2.2.2 Part 3

In this part, we implemented a 16-bit up-down counter that have 16-bit input data, load, clock, direction, 3-bit increment/decrement value and clear inputs using **reg** type parameter and **always** block. The output of the module is 16-bit current value of the counter.

- When the clear input is zero, current value of the counter becomes zero.
- When the load input is zero, input data is loaded to the the current value register.
- Otherwise, the current value of the counter is increased or decreased according to direction input and increment/decrement value on each positive edge of the clock input.

To illustrate this design clearly with an example: when the direction input is 1 and increment/decrement value is 2 (and clear=1, load=1), the module increases the counter with number 2. When the direction input is zero and decrement number is 3, the module decreases the counter with 3 (see Figure 13).



```
module part3(
    input wire [15:0] I,
    input wire load,
    input wire clock,
    input wire direction,
    input wire [2:0] value,
    input wire clear,
    output wire [15:0] Output
    );
    reg [15:0] O;

    assign Output = O;

    always @(posedge clock)
    begin
        if (clear == 0)
        begin
            O <= 16'b0;
        end
        else if (load == 0)
        begin
            O <= I;
        end
        else
        begin
            if (direction == 1)
            begin
                O <= O + value;
            end
            else
            begin
                O <= O - value;
            end
        end
    end
endmodule
```

Figure 13: Code of the circuit given in Part 3

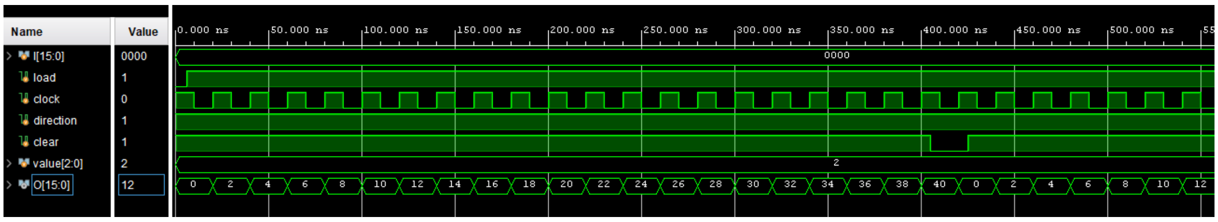
2.2.3 Part 4

In this part, we implemented below counter modules using 16-bit up-down counter module that is implemented in Part 3;
- A counter that counts up from 0 to 40 in circular way with increment value 2.
- A counter that counts up from 350 to 371 in circular way with increment value 3.
- A counter that counts down from 93 to 5 in circular way with decrement value 4.
- A counter that counts up from 22525 to 22535 in circular way with increment value 1.

Each module has clock input, initiate input, and 16-bit current value output. When the initiate input is zero, our counter is set to any number which is in the range of the counter. For example, for the 0 to 20 counter, it can be set to current value as 0 when the initiate input is zero. The clock input is used to update the current value.

```
reg [15:0] I;
reg load, clock, direction, clear;
reg [2:0] value;
wire [15:0] O;

part3 uut(I, load, clock, direction, value, clear, O);

/*   0 to 40 in circular way with increment value 2 */
initial begin
    I = 16'd0; load = 0;  direction = 1;   value = 3'd2; clock = 0; clear = 1; #6;
    load = 1;
end
always begin

    clock = ~clock; #5;
    if (O >= 16'd40)
    begin
        clear = 0; #5;
    end
    else
    begin
        clear = 1; #5;
    end

end
```

Figure 14: A counter that counts up from 0 to 40 in circular way with increment value 2



Name	Value
I[15:0]	0000
load	1
clock	0
direction	1
clear	1
value[2:0]	2
O[15:0]	12

Figure 15: Simulation results of counter that counts up from 0 to 40 in circular way with increment value 2

```verilog
reg [15:0] I;
reg load, clock, direction, clear;
reg [2:0] value;
wire [15:0] O;

part3 uut(I, load, clock, direction, value, clear, O);

 /*   350 to 371 in circular way with increment value 3 */
initial begin
    I = 16'd350; load = 0;  direction = 1;   value = 3'd3; clock = 0; clear = 1; #6;
    load = 1;
end
always begin

    clock = ~clock; #5;

    if (O >= 16'd371)
    begin
        load = 0;
        I = 16'd350; #5;
    end
    else
    begin
        load = 1; #5;
    end
```

Figure 16: A counter that counts up from 350 to 371 in circular way with increment value 3

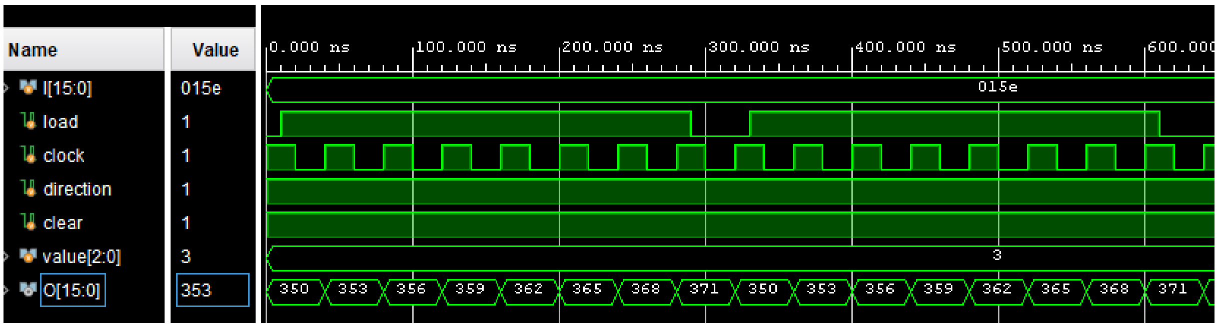
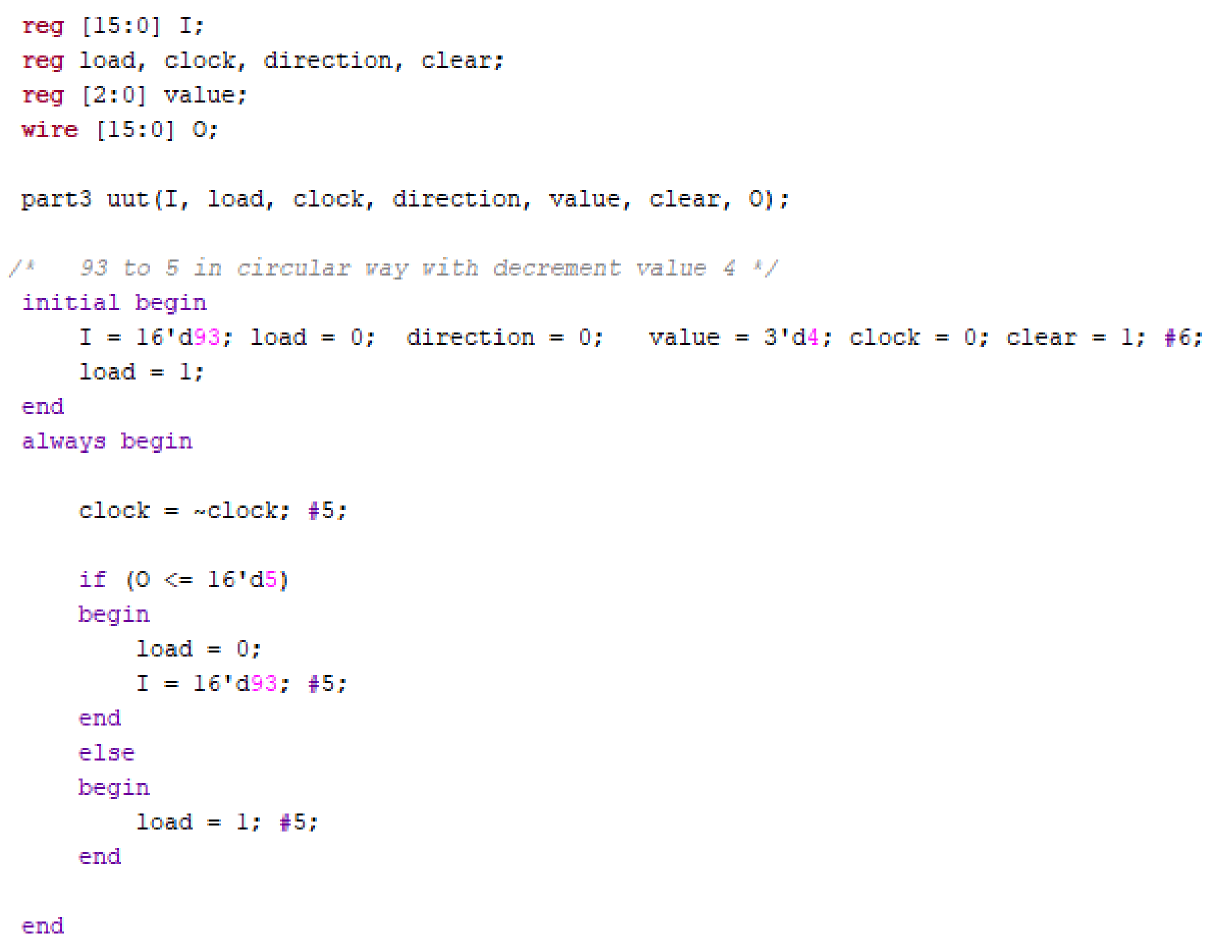


Figure 17: Simulation results of the counter that counts up from 350 to 371 in circular way with increment value 3



```
reg [15:0] I;
reg load, clock, direction, clear;
reg [2:0] value;
wire [15:0] O;

part3 uut(I, load, clock, direction, value, clear, O);

/*   93 to 5 in circular way with decrement value 4 */
initial begin
    I = 16'd93; load = 0;  direction = 0;   value = 3'd4; clock = 0; clear = 1; #6;
    load = 1;
end
always begin

    clock = ~clock; #5;

    if (O <= 16'd5)
    begin
        load = 0;
        I = 16'd93; #5;
    end
    else
    begin
        load = 1; #5;
    end

end
```

Figure 18: A counter that counts down from 93 to 5 in circular way with decrement value 4

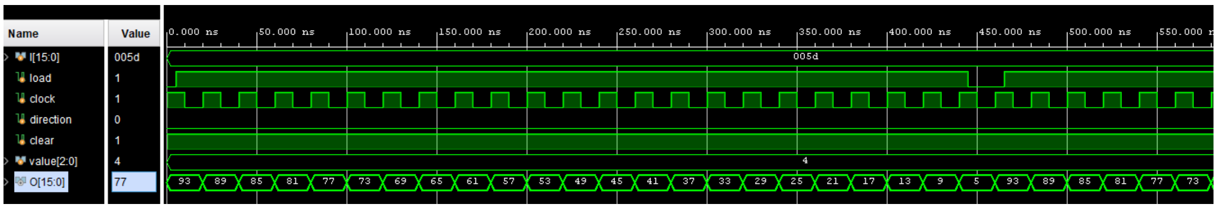


Figure 19: Simulation results of the counter that counts down from 93 to 5 in circular way with decrement value 4

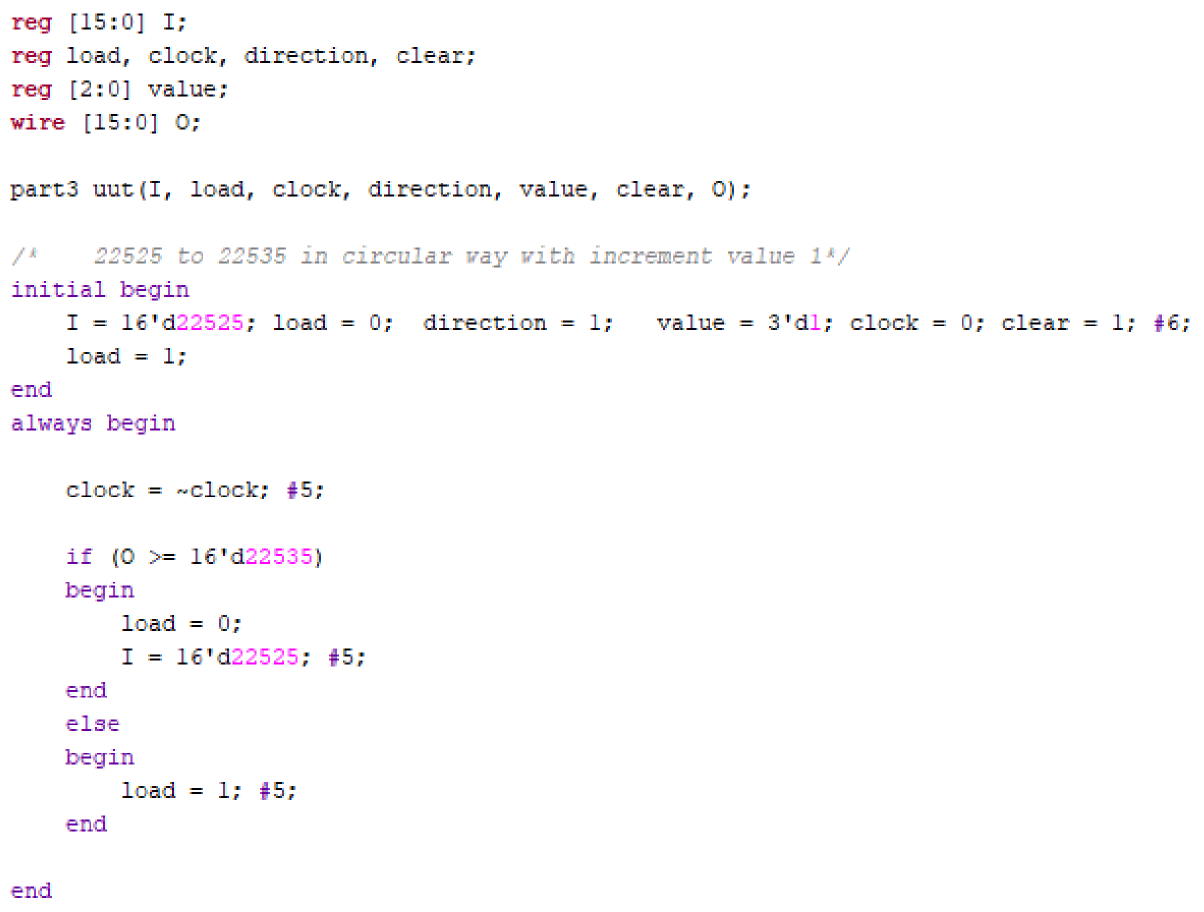

```
reg [15:0] I;
reg load, clock, direction, clear;
reg [2:0] value;
wire [15:0] O;

part3 uut(I, load, clock, direction, value, clear, O);

/*    22525 to 22535 in circular way with increment value 1*/
initial begin
    I = 16'd22525; load = 0;  direction = 1;   value = 3'd1; clock = 0; clear = 1; #6;
    load = 1;
end
always begin

    clock = ~clock; #5;

    if (O >= 16'd22535)
    begin
        load = 0;
        I = 16'd22525; #5;
    end
    else
    begin
        load = 1; #5;
    end

end
```

Figure 20: A counter that counts up from 22525 to 22535 in circular way with increment value 1

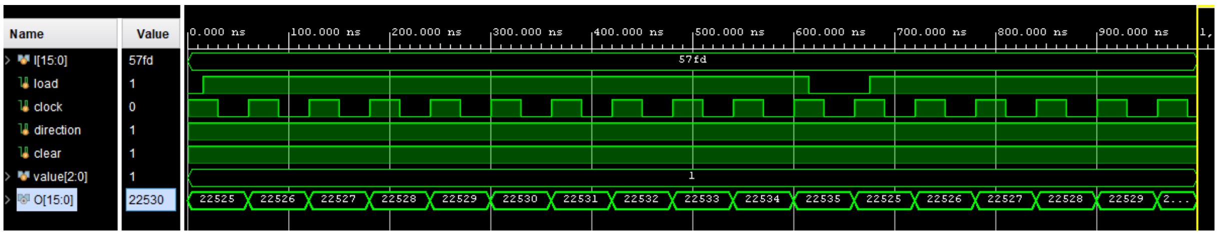
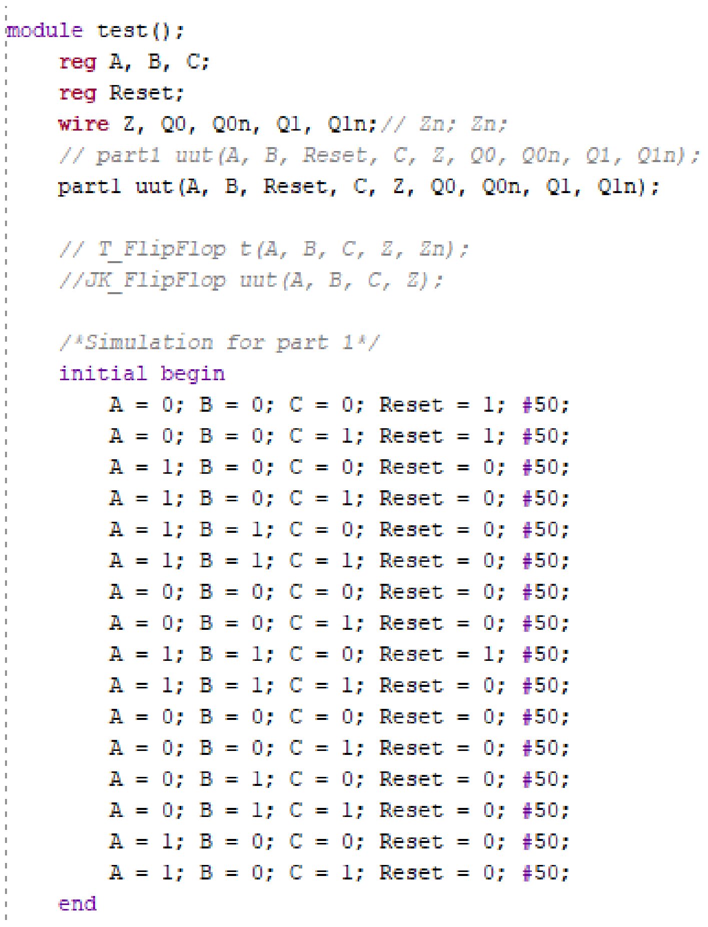

```
Name          Value
I[15:0]       57fd
load          1
clock         0
direction     1
clear         1
value[2:0]    1
O[15:0]       22530
```

Figure 21: Simulation results of the counter that counts up from 22525 to 22535 in circular way with increment value 1

3 RESULTS [15 points]

You can see the simulation codes we used for Experiments Part1 and its simulation results. To carry out each different implementation of this part of the experiment, we arranged inputs, clock and reset of each simulation according to part or input/reset it belongs. The simulation results we obtained after completing each step of the experiments resulted NOT IN THE WAY we expected. This concludes the fact that our experiment part-1 is not coherent.

About part-1, see **Part 1** section above.



```
module test();
    reg A, B, C;
    reg Reset;
    wire Z, Q0, Q0n, Q1, Q1n;// Zn; Zn;
    // part1 uut(A, B, Reset, C, Z, Q0, Q0n, Q1, Q1n);
    part1 uut(A, B, Reset, C, Z, Q0, Q0n, Q1, Q1n);

    // T_FlipFlop t(A, B, C, Z, Zn);
    //JK_FlipFlop uut(A, B, C, Z);

    /*Simulation for part 1*/
    initial begin
        A = 0; B = 0; C = 0; Reset = 1; #50;
        A = 0; B = 0; C = 1; Reset = 1; #50;
        A = 1; B = 0; C = 0; Reset = 0; #50;
        A = 1; B = 0; C = 1; Reset = 0; #50;
        A = 1; B = 1; C = 0; Reset = 0; #50;
        A = 1; B = 1; C = 1; Reset = 0; #50;
        A = 0; B = 0; C = 0; Reset = 0; #50;
        A = 0; B = 0; C = 1; Reset = 0; #50;
        A = 1; B = 1; C = 0; Reset = 1; #50;
        A = 1; B = 1; C = 1; Reset = 0; #50;
        A = 0; B = 0; C = 0; Reset = 0; #50;
        A = 0; B = 0; C = 1; Reset = 0; #50;
        A = 0; B = 1; C = 0; Reset = 0; #50;
        A = 0; B = 1; C = 1; Reset = 0; #50;
        A = 1; B = 0; C = 0; Reset = 0; #50;
        A = 1; B = 0; C = 1; Reset = 0; #50;
    end
```

Figure 22: Simulation result of the circuit given in Part 1

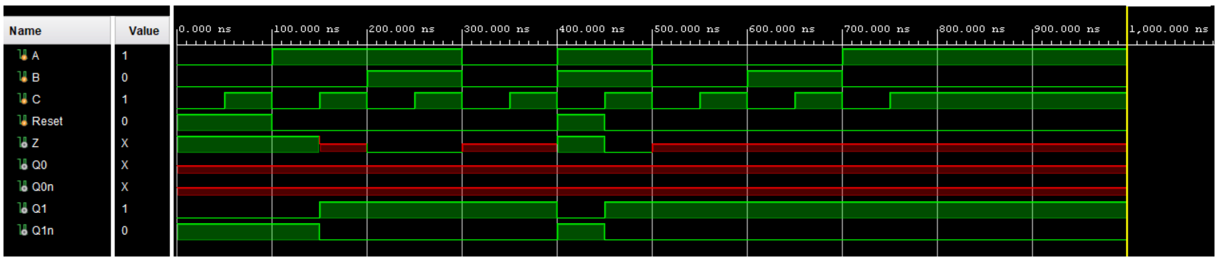


Figure 23: Simulation result of the circuit given in Part 1

You can see the simulation codes we used for Part3 and Part4 Experiments' parts and their simulation results, all separately ran one after another, while the other are in the commented version to prevent the program from crush. To carry out each part of the experiment, we arranged inputs and the clocks of each simulation according to part or input/reset it belongs. The simulation results we obtained after completing each step of the experiments resulted as we expected. This concludes the fact that our experiment part-3 and so part-4 ended up being coherent.

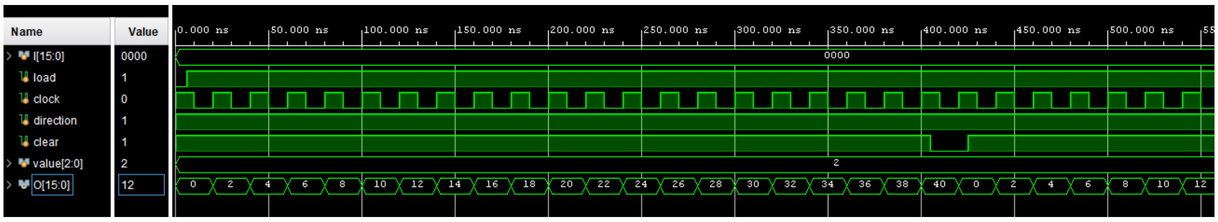


Figure 24: Simulation results of counter that counts up from 0 to 40 in circular way with increment value 2

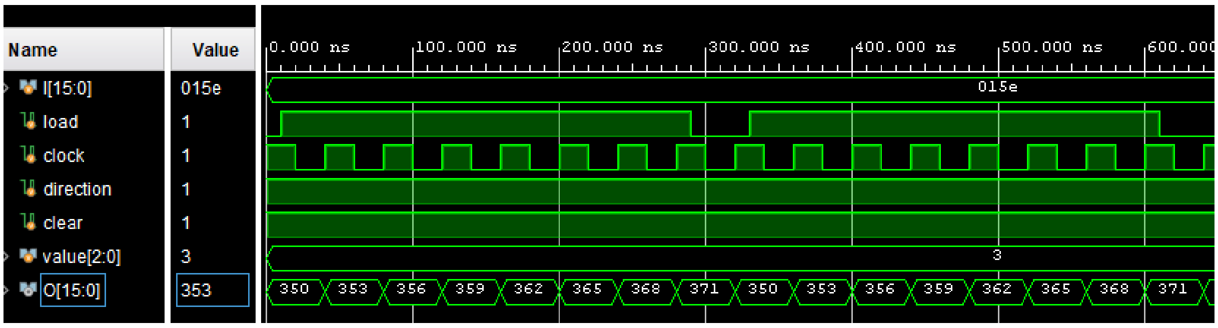


Figure 25: Simulation results of the counter that counts up from 350 to 371 in circular way with increment value 3

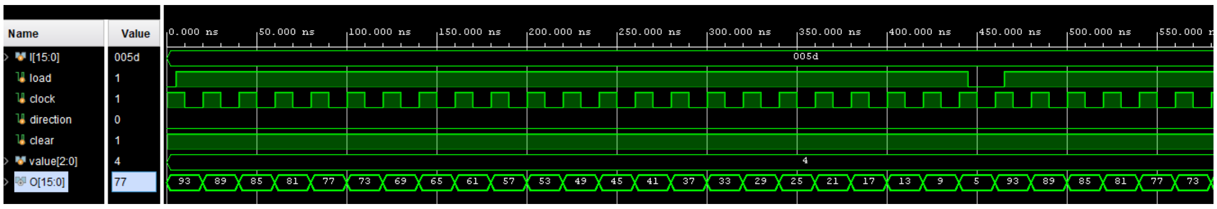


Figure 26: Simulation results of the counter that counts down from 93 to 5 in circular way with decrement value 4

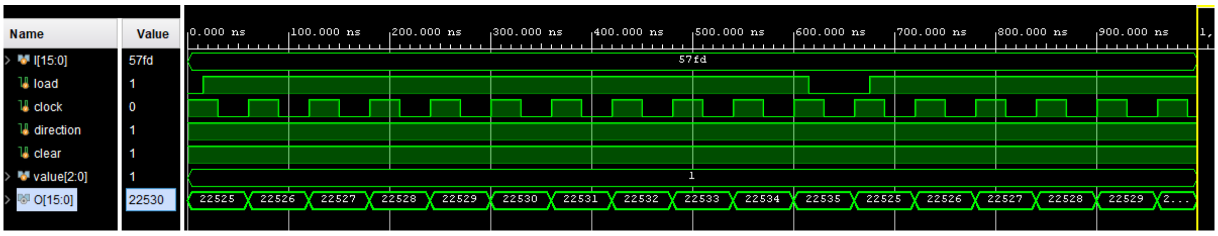


Figure 27: Simulation results of the counter that counts up from 22525 to 22535 in circular way with increment value 1

4 DISCUSSION [25 points]

In Part - 1, we tried to implement the circuit that is given to us in **In Part - 1** by using Verilog in **3 different ways** by using 2 different JK Flip-Flops. We couldn't manage to succeed, but we would really appreciate a debugging or at least an explanation about this simulation results.

We used different types of flip flop modules:

- JK Flip Flop,
- T Flip Flop,
- D Flip Flop (within JK & T Flip Flops)

We used different types of logic gates:

- AND Gates
- OR Gates
- NOT Gates
- NAND Gates

And clock signal to synchronize the circuit. We drew the state transition table for this circuit and draw state diagram graphically (see **Figure 3**).

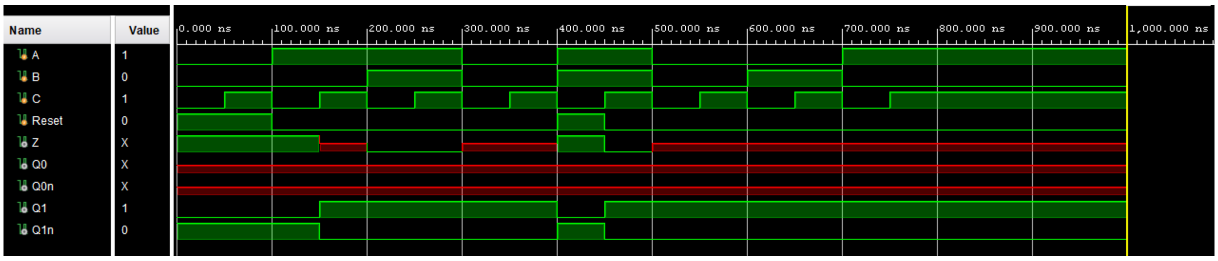


Figure 28: Simulation result of the circuit given in Part 1

Also, we showed our simulation results (timing diagram) by testing each different combination (implementation) of our circuits. But we can not get a proper result. We added reset input to set initial values. We didn't use always block.

In Part-2, we read the question, watched the videos in the hint-1 and read hint-2, but... but there's no way we can do it. Being sorry for this :(

In Part-3, we implemented a 16-bit up-down counter that have 16-bit input data, load, clock, direction, 3-bit increment/decrement value and clear inputs using **reg** type parameter and **always** block. The output of the module is 16-bit current value of the counter.
- When the clear input is zero, current value of the counter becomes zero.
- When the load input is zero, input data is loaded to the the current value register.
- Otherwise, the current value of the counter is increased or decreased according to direction input and increment/decrement value on each positive edge of the clock input.

To illustrate this design clearly with an example: when the direction input is 1 and increment/decrement value is 2 (and clear=1, load=1), the module increases the counter with number 2. When the direction input is zero and decrement number is 3, the module decreases the counter with 3.

In Part-4, we implemented below counter modules using 16-bit up-down counter module that is implemented in Part 3;
- A counter that counts up from 0 to 40 in circular way with increment value 2.
- A counter that counts up from 350 to 371 in circular way with increment value 3.
- A counter that counts down from 93 to 5 in circular way with decrement value 4.
- A counter that counts up from 22525 to 22535 in circular way with increment value 1.

Each module has clock input, initiate input, and 16-bit current value output. When the initiate input is zero, our counter is set to any number which is in the range of the counter. For example, for the 0 to 20 counter, it can be set to current value as 0 when the initiate input is zero. The clock input is used to update the current value.

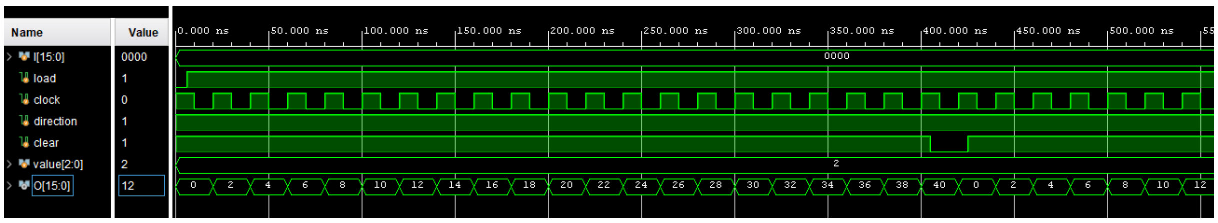


Figure 29: Simulation results of counter that counts up from 0 to 40 in circular way with increment value 2

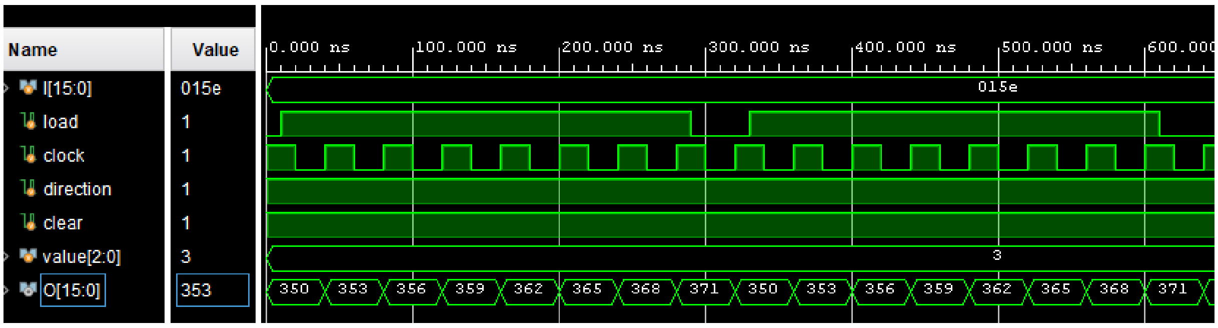


Figure 30: Simulation results of the counter that counts up from 350 to 371 in circular way with increment value 3

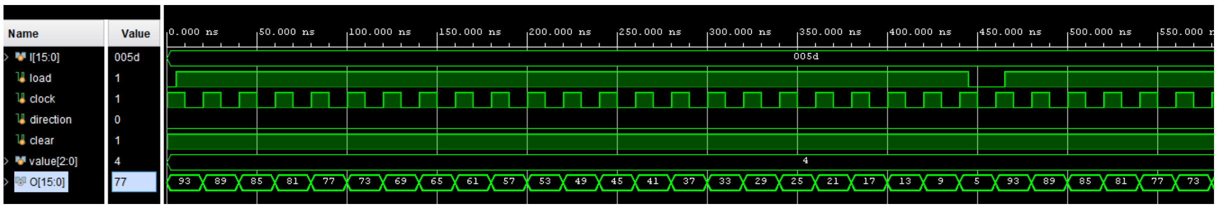


Figure 31: Simulation results of the counter that counts down from 93 to 5 in circular way with decrement value 4

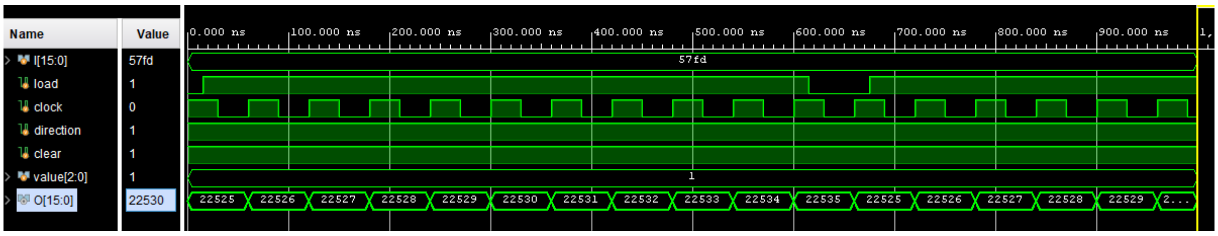


Figure 32: Simulation results of the counter that counts up from 22525 to 22535 in circular way with increment value 1

5 CONCLUSION [10 points]

We learned how to designed JK and T Flip Flops different ways, however, we can not implement JK flip flops. Sequential circuit, which is a type of Mealy model, is implemented without using always block. Firstly, we obtained the equations of our circuit elements by making appropriate calculations. By using them, we constructed the transition tables and drew our state transition diagram. Then we try to implement our circuit on Verilog. But, since JK Flip Flop does not run the correct way in Part - 1 code, our result are invalid. We try to fix this problem by coding different designs but we can not find the correct results and failed. At the end, we have not a proper solution for Part - 1.

Also, we learn how to implement 16 bit up-down counters with load, reset and increment value. By using that, we implement circular counters and simulate them. Coding that is very easy since we can use always and if-else structures. First, we load the inputs and then we increment the values sequentially as shown in the simulation results. For circular case, we write an if-else structure and change them when the result will be out of range.

In conclusion, we had difficulties on analyzing and coding circuits, especially in Part - 1 and Part - 2. But at least we tried to solve our problems by applying different methods.