DIGITAL SYSTEM DESIGN APPLICATION – EHB 436E

Experiment VI

Yiğit Bektaş GÜRSOY

040180063

Class Lecturer: Sıddıka Berna Örs Yalçın

Class Assistant:
Serdar Duran
Yasin Fırat Kula
Mehmet Onur Demirtürk

1. STATE ENCODING

- State encoding is the technique that names all states that will be used in the future. One of the encoding techniques is the **binary encoding.** In binary encoding, all states have binary names from counting zero. There will be at least log2(#state) flip flop in this method.
- Following figure shows the algorithm.

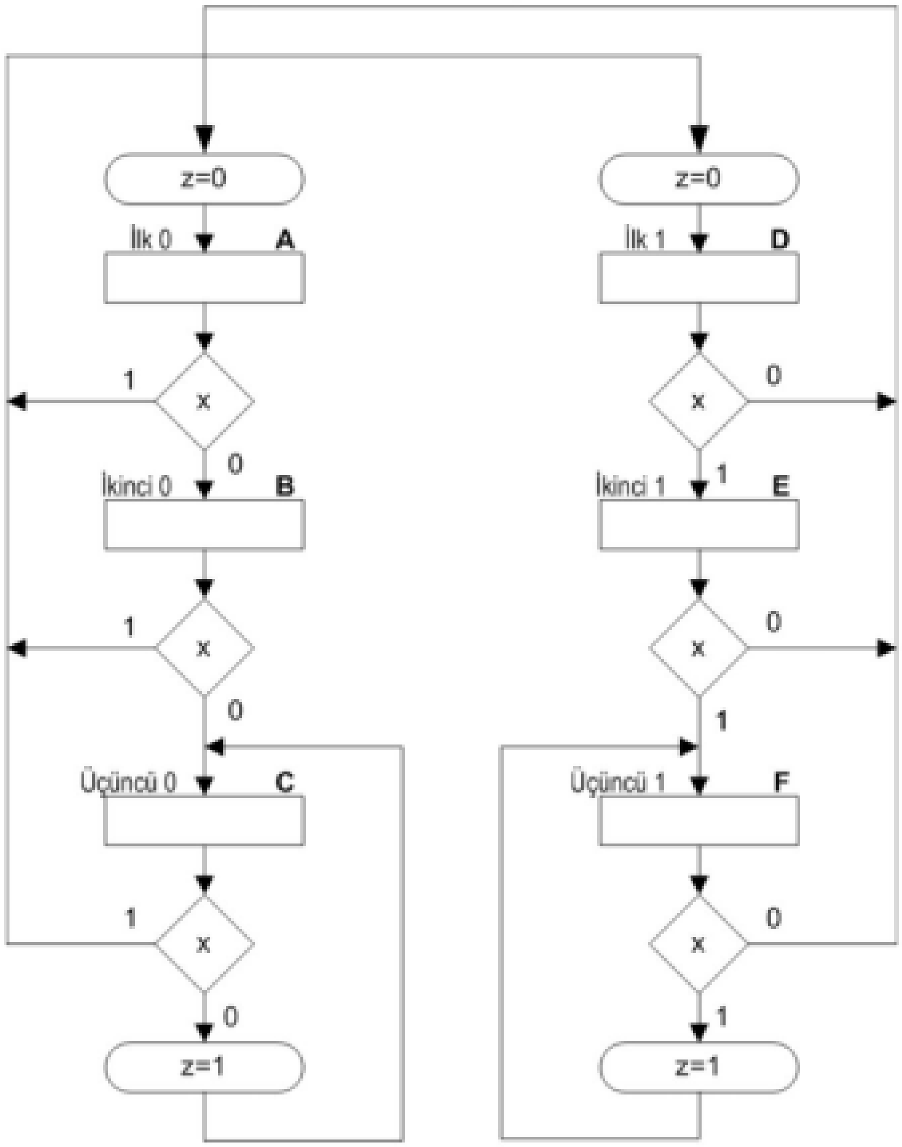


Figure 1: Algorithmic State Diagram

- Following figures shows binary encoding for states. I would have done same encoding for states. Therefore, I used the same figures in the experiment paper.
- States A…F have binary codes from 000 to 101. Current situation says we are going to use 3 flip flops in the circuit. (without reduction)

State	next state, output	
	x = 0	x = 1
A	B,0	D,0
B	C,0	D,0
C	C,1	D,0
D	A,0	E,0
E	A,0	F,0
F	A,0	F,1

Figure 2: State Table without any reduction

state	binary code
A	000
B	001
C	010
D	011
E	100
F	101

Figure 3: State Encoding Example

2. STATE REDUCTION

- State reduction is the technique that removes the states that have the same relation between previous and next state. These states can be reduced to one state.

State	next state, output	
	x = 0	x = 1
A	B,0	D,0
B	C,0	D,0
C	C,1	D,0
D	A,0	E,0
E	A,0	F,0
F	A,0	F,1

Figure 2: State Table without any reduction

- As you can see in the figure above, every state has a unique "next state and output". If some of the states have the same output and next state, we would make reduction. However, state reduction is not possible in this algorithm design.

Yiğit Bektaş GÜRSOY
040180063

3. REDUCTION OF COMBINATORIAL PART

- K-maps of next state functions

Q_2

Final Expression = xq1q0 + xq2

x,q2\q1,q0	00	01	11	10
00	0_{0}	0_{1}	0_{3}	0_{2}
01	0_{4}	0_{5}	$-_{7}$	$-_{6}$
11	1_{12}	1_{13}	$-_{15}$	$-_{14}$
10	0_{8}	0_{9}	1_{11}	0_{10}

Q_1

Final Expression = xq2'q1' + q2'q1'q0 + q1q0'

x,q2\q1,q0	00	01	11	10
00	0_{0}	1_{1}	0_{3}	1_{2}
01	0_{4}	0_{5}	$-_{7}$	$-_{6}$
11	0_{12}	0_{13}	$-_{15}$	$-_{14}$
10	1_{8}	1_{9}	0_{11}	1_{10}

Q_0

Final Expression = q2'q1'q0' + xq0' + xq1'

x,q2\q1,q0	00	01	11	10
00	1_{0}	0_{1}	0_{3}	0_{2}
01	0_{4}	0_{5}	$-_{7}$	$-_{6}$
11	1_{12}	1_{13}	$-_{15}$	$-_{14}$
10	1_{8}	1_{9}	0_{11}	1_{10}

Z

Final Expression = x'q1q0' + xq2q0

x,q2\q1,q0	00	01	11	10
00	0_{0}	0_{1}	0_{3}	1_{2}
01	0_{4}	0_{5}	$-_{7}$	$-_{6}$
11	0_{12}	1_{13}	$-_{15}$	$-_{14}$
10	0_{8}	0_{9}	0_{11}	0_{10}

4. FSM 1

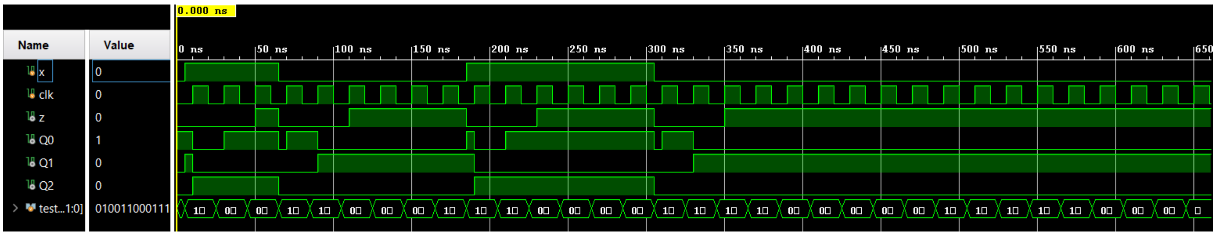
- Verilog Codes

```
module FSM1(

    input x,clk,
    //output reg z //moore
    output z //mealy
    );

    wire Q0,Q1,Q2,zTEMP;
    reg q0,q1,q2;

    //starting state
    initial begin
    // first state:000
   q0=0;
   q1=0;
   q2=0;
    end

    assign
Q0=(x&~q1)|(~q0&~q1&~q2)|(x&~q0);
    assign
Q1=(q1&~q0)|(q0&~q1&~q2)|(x&~q2&~q1);
    assign Q2=(x&q2)|(x&q0&q1);
    //assign
zTEMP=(x&q0&q2)|(~x&q1&~q0); //moore
    assign z=(x&q0&q2)|(~x&q1&~q0);
//mealy

    always @(posedge clk) begin
    q2 <= Q2;
    q1 <= Q1;
    q0 <= Q0;
    //z = zTEMP; //moore machine
    end
endmodule
```

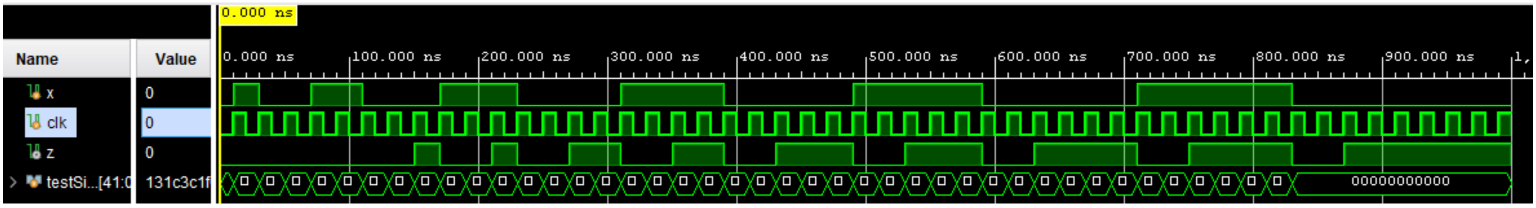
- Testbench Code

```
module FSM1(

    input x,clk,
    //output reg z //moore
    output z //mealy
    );

    wire Q0,Q1,Q2,zTEMP;
    reg q0,q1,q2;

    //starting state
    initial begin
    // first state:000
   q0=0;
   q1=0;
   q2=0;
    end

    assign
Q0=(x&~q1)|(~q0&~q1&~q2)|(x&~q0);
    assign
Q1=(q1&~q0)|(q0&~q1&~q2)|(x&~q2&~q1);
    assign Q2=(x&q2)|(x&q0&q1);
    //assign
zTEMP=(x&q0&q2)|(~x&q1&~q0); //moore
    assign z=(x&q0&q2)|(~x&q1&~q0);
//mealy

    always @(posedge clk) begin
    q2 = Q2;
    q1 = Q1;
    q0 = Q0;
    //z = zTEMP; //moore machine
    end
endmodule
```

- Behavioral Simulation
  - ➢ Behavioral simulation is shown in the figures below. Behavioral simulation results are not true**.** It will be explained why in next steps.

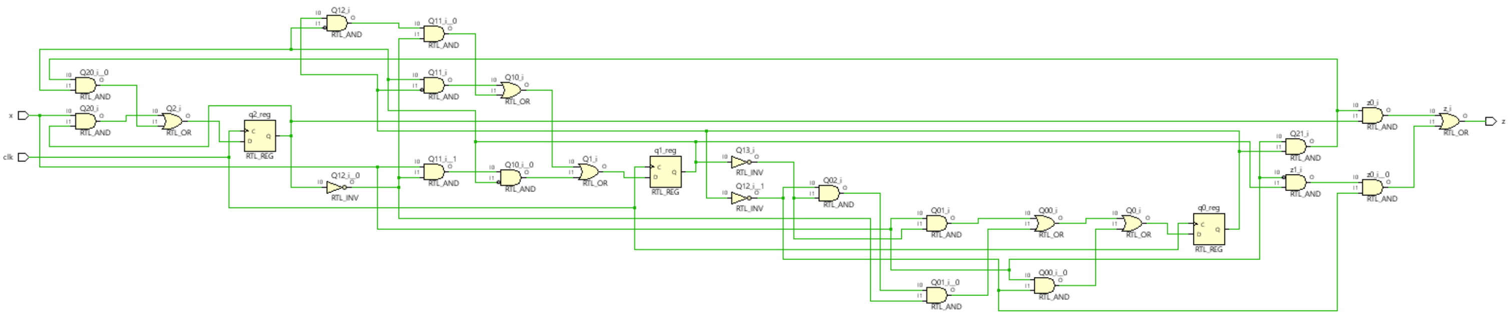


- Post Implement Functional Simulation
  - ➢ Post Implement Functional Simulation **has wrong outputs.** It will be explained why in Behavioral Simulation section.

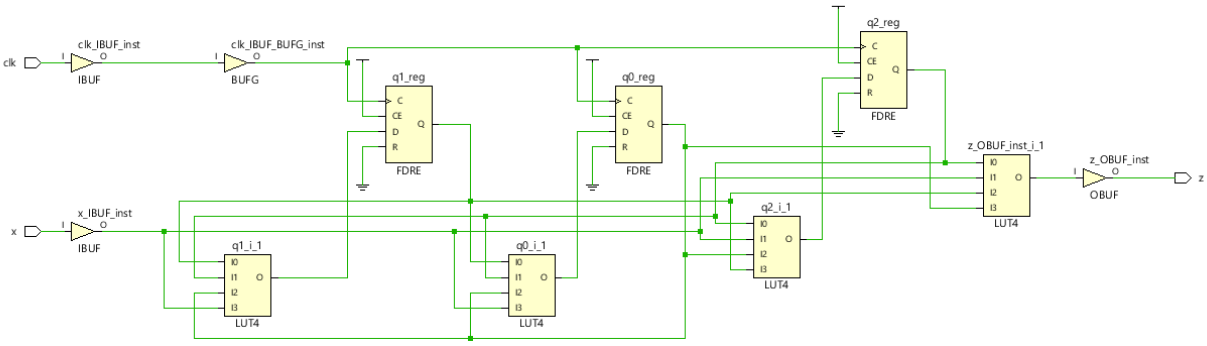


  - ➢ The simulation obtained from post implantation function simulation has wrong results. Characteristic of the faulty outputs are the same. Every 3 same inputs, circuit generates high output instead it should generate for 4 same inputs. The reason why circuit generates these wrong results is about mealy machine. In the mealy machine, inputs and states have effect on the output. Therefore, output is not waiting the next clock signal. To avoid such situations we can use moore machine.

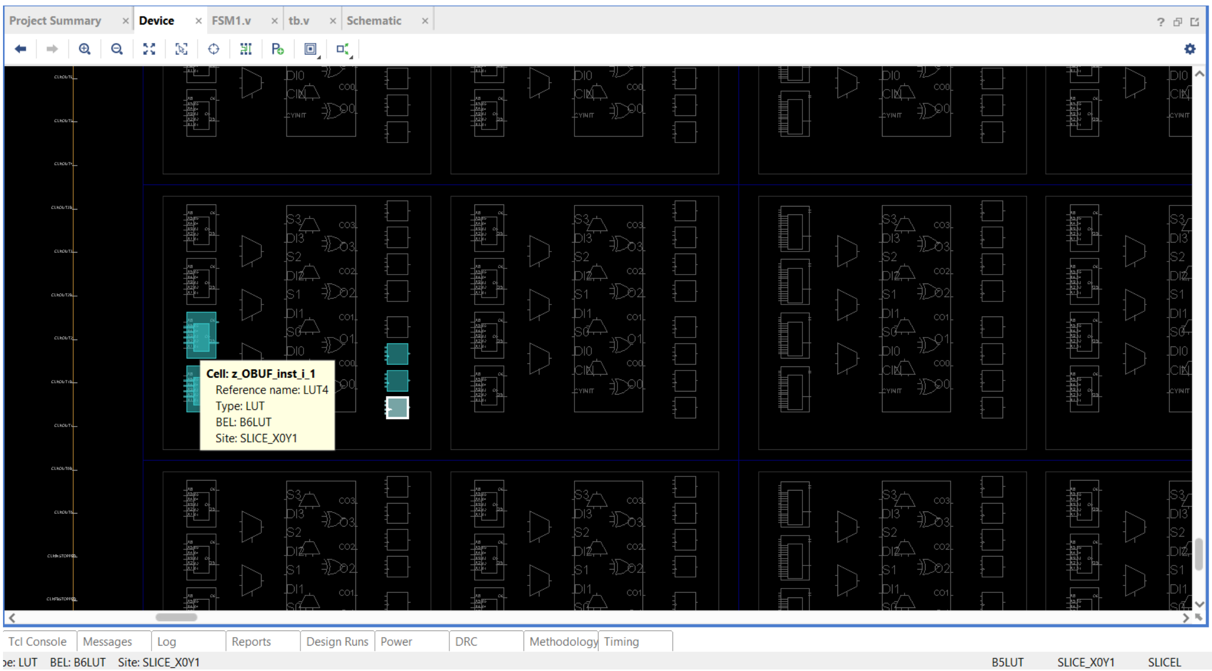
- RTL Schematic



- Technology Schematic



- Device Layout



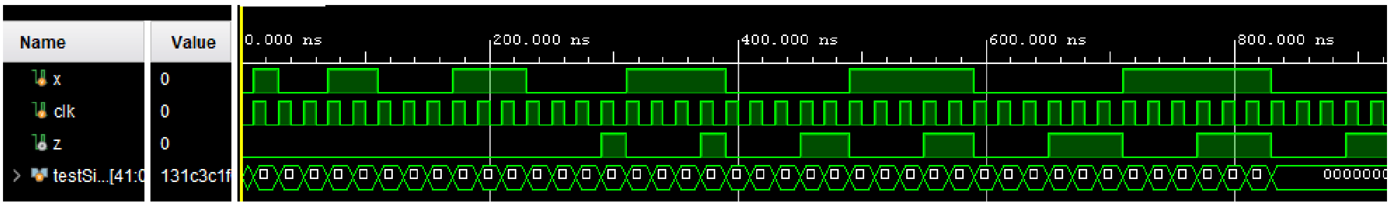
- Changing Machine Type
  - ➢ We can add D flip flop to previously designed circuit to prevent malfunctions. New circuit became moore machine instead of mealy machine.

- FSM 1 designed as Moore Machine
  - ➢ We are going to add D flip flop to the end of the previously designed circuit.

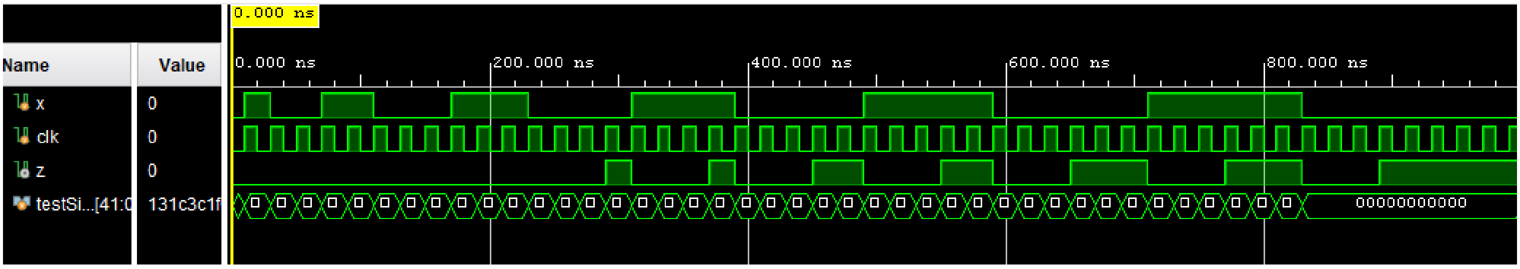
- Verilog Codes

```
timescale 1ns / 1ps
module FSM1(
    input x,clk,
    output reg z //moore
    //output z //mealy
    );

    wire Q0,Q1,Q2,zTEMP;
    reg q0,q1,q2;

    //starting state
    initial begin
    // first state:000
   q0=0;
   q1=0;
   q2=0;
    end

    assign Q0=(x&~q1)|(~q0&~q1&~q2)|(x&~q0);
    assign Q1=(q1&~q0)|(q0&~q1&~q2)|(x&~q2&~q1);
    assign Q2=(x&q2)|(x&q0&q1);
    assign zTEMP=(x&q0&q2)|(~x&q1&~q0); //moore
    //assign z=(x&q0&q2)|(~x&q1&~q0); //mealy

    always @(posedge clk) begin
    q2 = Q2;
    q1 = Q1;
    q0 = Q0;
    z = zTEMP; //moore machine
    end
endmodule
```

- Test bench codes are the same with mealy machine.
- Behavioral Simulation
  - ➢ After adding flip flop to previously designed mealy machine circuit, the new circuit generates **true results.**



- Post Implementation Functional Simulation



- Arbitrary First State
  - Previous designs need to start with first state that is encoded with 000. In this section, I changed first state to an arbitrary state that is encoded as 011. Note that in this section, mealy machine are going to be used.

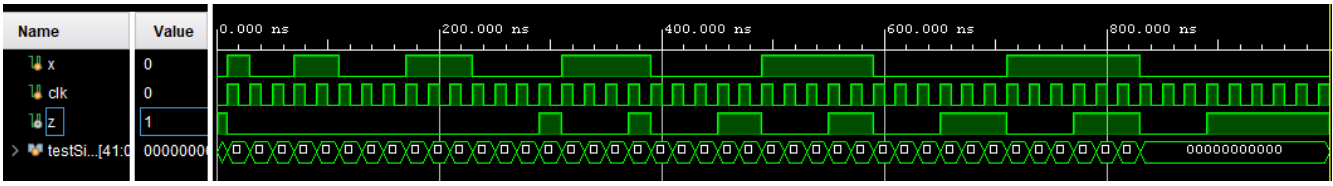
- Verilog Codes

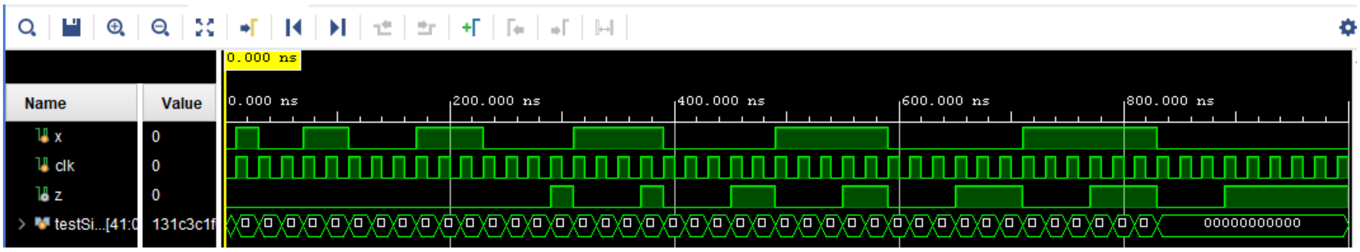
```
`timescale 1ns / 1ps
module FSM1(
    input x,clk,
    //output reg z //moore
    output z //mealy
    );

    wire Q0,Q1,Q2,zTEMP;
    reg q0,q1,q2;

    //starting state
    initial begin
    // first state:000
//      q0=0;
//      q1=0;
//      q2=0;
    // first state:110, arbitrary
        q0=0;
        q1=1;
        q2=1;
    end

    assign Q0=(x&~q1)|(~q0&~q1&~q2)|(x&~q0);
    assign Q1=(q1&~q0)|(q0&~q1&~q2)|(x&~q2&~q1);
    assign Q2=(x&q2)|(x&q0&q1);
    //assign zTEMP=(x&q0&q2)|(~x&q1&~q0); //moore
    assign z=(x&q0&q2)|(~x&q1&~q0); //mealy

    always @(posedge clk) begin
    q2 = Q2;
    q1 = Q1;
    q0 = Q0;
    //z = zTEMP; //moore machine
    end
endmodule
```

- Testbench is the same with previously used testbenchs.

- Behavioral Simulation for arbitrary state (110)
  - ➢ Behavioral simulation of the circuit is true. First result of the circuit is 1 instead 0. This is the only wrong result, after that circuit escapes arbitrary states and generates true results.

- Behavioral Simulation for other arbitrary state (111)





  - ➢ **I will not add verilog codes** for this because almost everything is the same. I set first state as 111 instead 000.

- Arbitrary first state comment
  - ➢ According to behavioral simulation of the circuit has first state is one of the arbitrary state, the circuit can escape from arbitrary states to encoded states. Note that they can generate false results at the beginning of the simulation.

5. ALTERNATIVE DESIGN FSM 2

- Showing Fig. 1 and Fig 5 in the experiment are the same
  - New circuit (Fig 5) has the two different algorithms to realize the algorithm in the Fig 1.

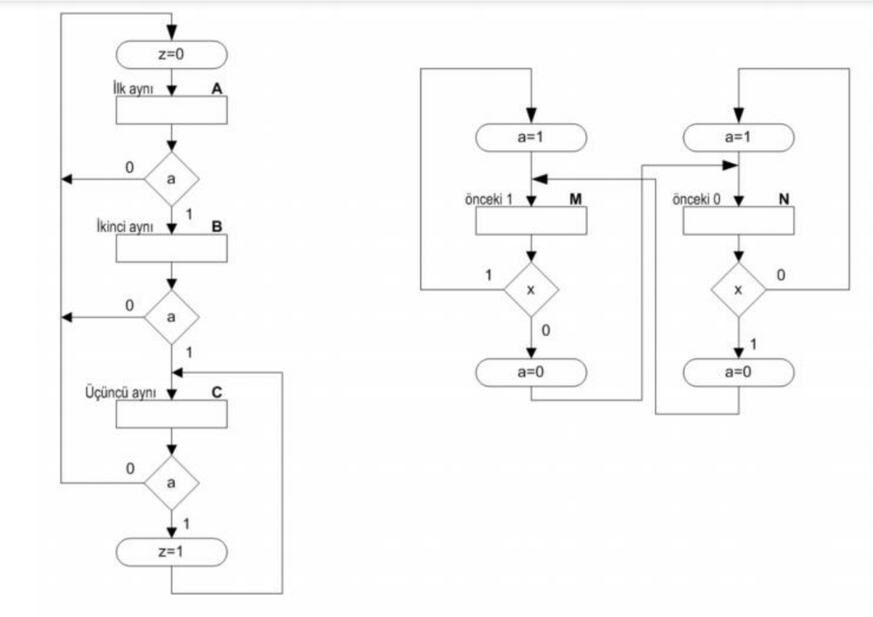
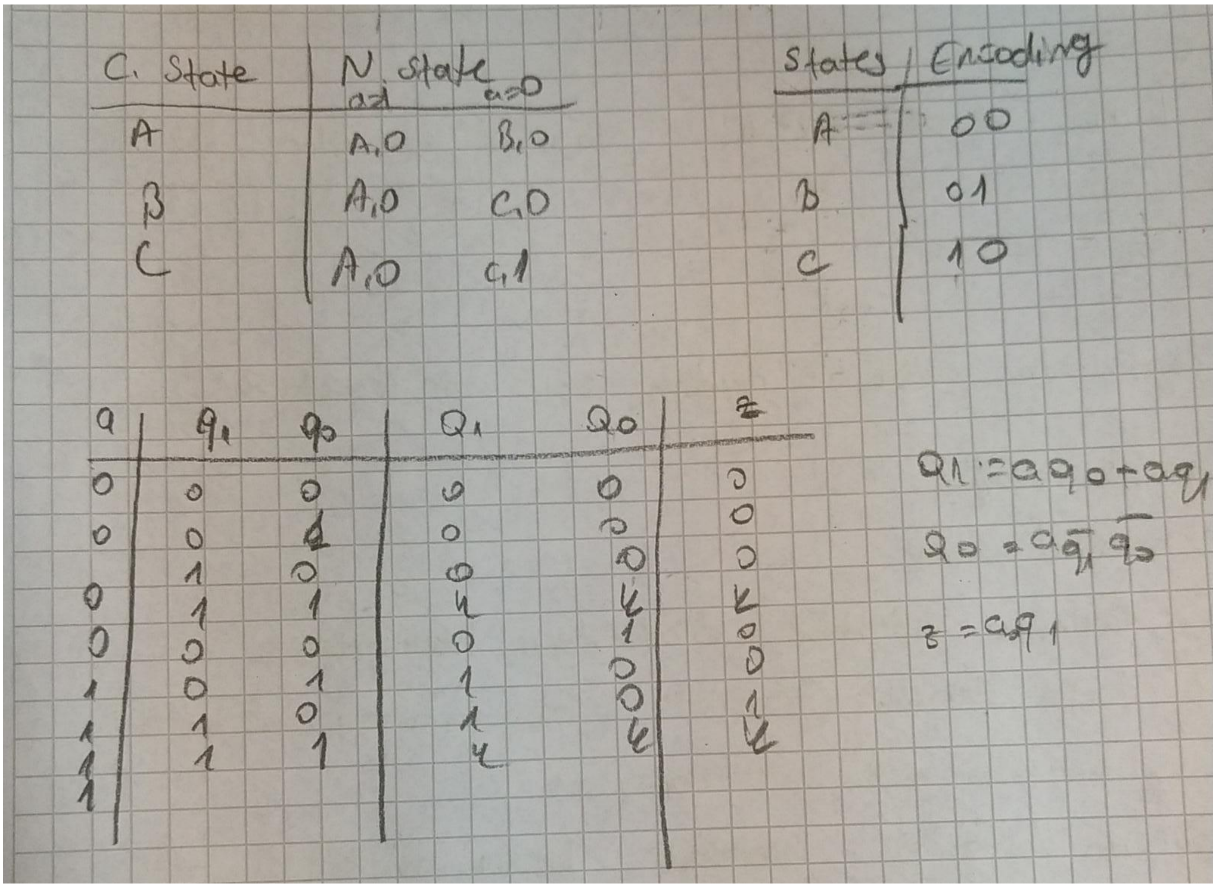


Figure 5: Divided Algorithmic State Table

  - Left algorithm is to detect 3 consecutive input 1. Right algorithm makes to generalize for both 0 and 1 consecutive inputs.
  - Finally if we realize both algorithms, we would make the circuit that detects 3 consecutive 1 or 0.

- States and state encoding for FSM2
  - There are next state table, state encoding table, and next state functions. K-maps are not used because functions can easily be found.



C. State	N. state a=1	N. state a=0
A	A,0	B,0
B	A,0	C,0
C	A,0	C,1

States	Encoding
A	00
B	01
C	10

a	q_1	q_0	Q_1	Q_0	z
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	X	X	X
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	X	X	X

$Q_1 = aq_0 + aq_1$

$Q_0 = a\overline{q_1}\,\overline{q_0}$

$z = aq_1$

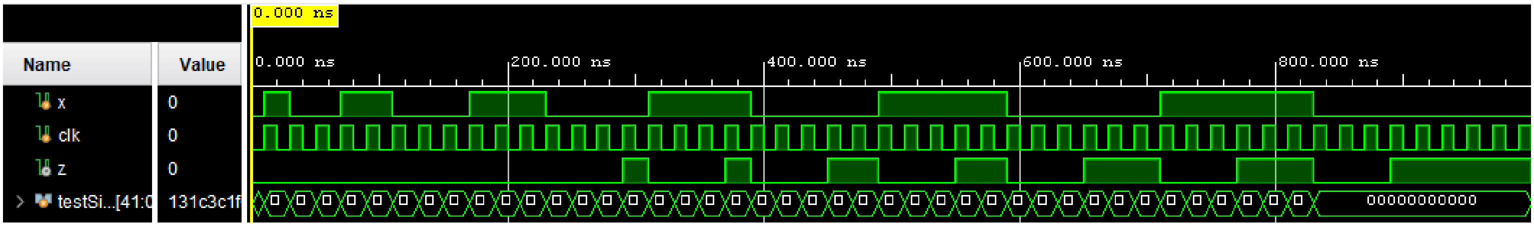
- Verilog Codes
  - These module code includes both mealy and moore codes.

```
`timescale 1ns / 1ps
module FSM2(
    input x,clk,
    //output reg z //moore
    output z //mealy
    );

    //moore
//    wire Q0,Q1,Q2,zTEMP,aTEMP;
//    reg q0,q1,q2,a;
    //mealy
    wire Q0,Q1,Q2,aTEMP;
    reg q0,q1,q2;

    //starting state
    initial begin
    // first state:000
   q0=0;
   q1=0;
   q2=0;
    //a=0; // moore
    end

    assign Q0=aTEMP&~q0&~q1;
    assign Q1=aTEMP&(q0|q1);
    assign Q2=~x;
    assign z=aTEMP&q1; //mealy
    assign aTEMP = x^q2;  //mealy
//    assign zTEMP=aTEMP&q1; //moore
//    assign aTEMP = x^q2;  //moore

    always @(posedge clk) begin
    q2 = Q2;
    q1 = Q1;
    q0 = Q0;
//    z = zTEMP; // moore
//    a = aTEMP; //moore
    end
endmodule
```

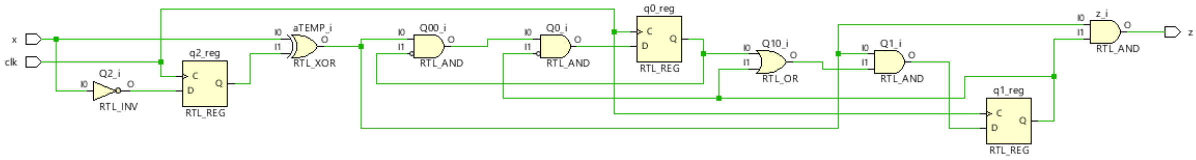
- Testbench is almost same.

```
`timescale 1ns / 1ps
module tb;
    reg x;
    reg clk;
    wire z;

//    FSM1 uut(x,clk,z);
    FSM2 uut(x,clk,z);

    reg[41:0] testSignal =
42'b010011000111000011110000011111000000111111;

    initial begin
        x = 0;
        clk = 0;
    end

    always #10 clk = !clk;

    always @(posedge clk) begin
        testSignal = testSignal << 1;
        x = testSignal[41];
    end
endmodule
```

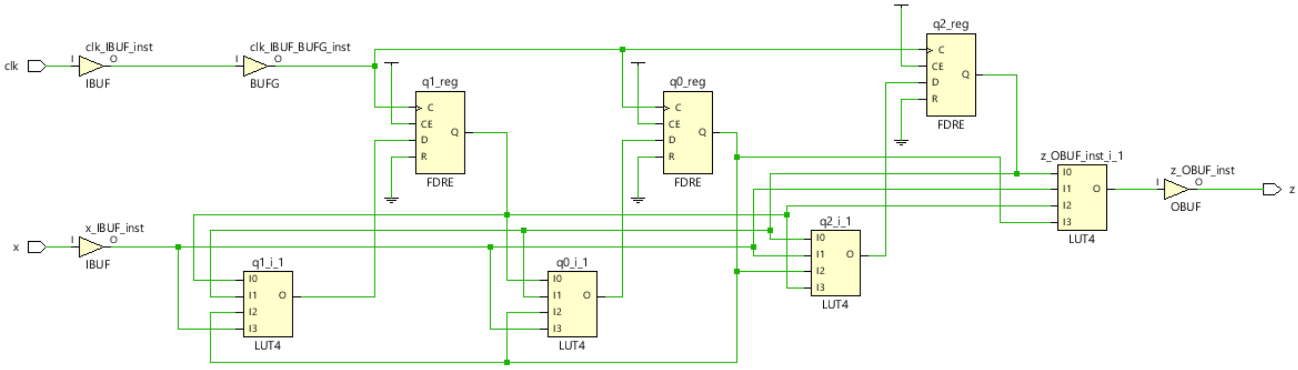
- Behavioral Simulation of FSM2 (mealy)
  - Mealy implementation of FSM2 was obtained. It has same mealy problem. It will be solved with moore design.



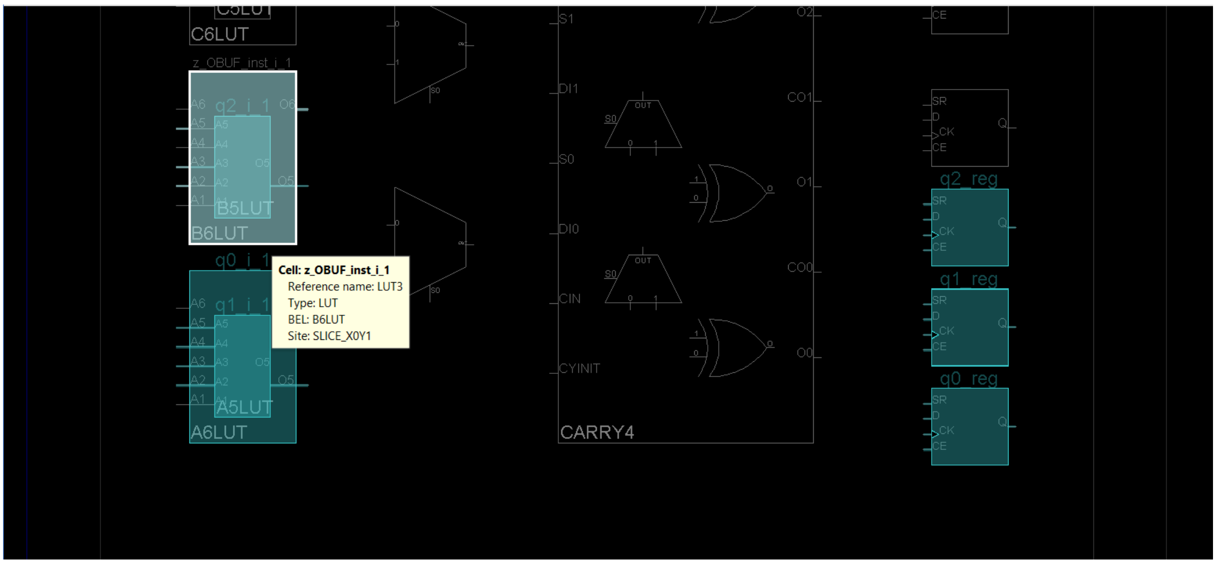
- RTL Schematic of FSM2 (mealy)



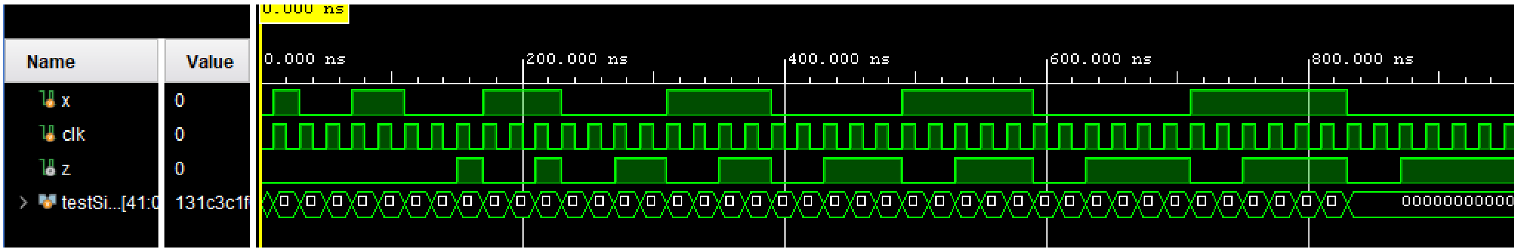
- Technology Schematic of FSM2 (mealy)



- Device Layout

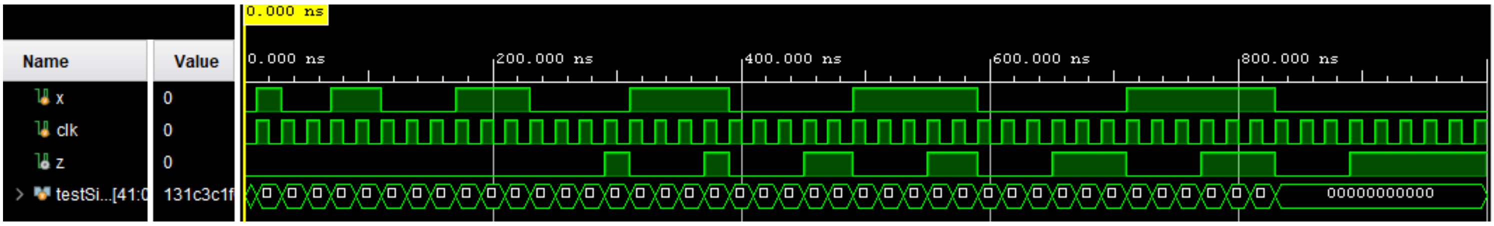


- Post Implementation Functional Simulation fo FSM2 (mealy)
  - It has the same **faulty results.** Like in the previous chapter about FSM1, the mealy circuit generates same wrong results. Every 3 same inputs, circuit generates high output instead it should generate for 4 same inputs. The reason why circuit generates these wrong results is about mealy machine. In the mealy machine, inputs and states have effect on the output. Therefore, output is not waiting the next clock signal. To avoid such situations we can use moore machine.



- Moore Machine design for FSM2
  - I add d flip flop to previous mealy FSM2 design. (It has both mealy and moore machine codes.). Testbench is the same.

- Post Implementation Functional Simulation fo FSM2 (moore)
  - The moore machine generates **true results** for post implement functional simulation.

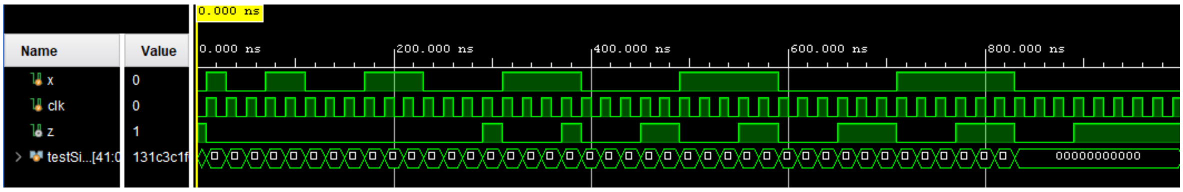


6. ARBITRARY STATE FOR FSM2

- Verilog codes for arbitrary states:

```
`timescale 1ns / 1ps
module FSM2(
    input x,clk,
    //output reg z //moore
    output z //mealy
    );

    //moore
//    wire Q0,Q1,Q2,zTEMP,aTEMP;
//    reg q0,q1,q2,a;
    //mealy
    wire Q0,Q1,Q2,aTEMP;
    reg q0,q1,q2;

    //starting state
    initial begin
    // first state:000
//    q0=0;
//    q1=0;
//    q2=0;
    //a=0; // moore
    // first state:111, arbitrary
        q0=0;
        q1=1;
        q2=1;
     //   a=0;// moore
    end

    assign Q0=aTEMP&~q0&~q1;
    assign Q1=aTEMP&(q0|q1);
    assign Q2=~x;
    assign z=aTEMP&q1; //mealy
    assign aTEMP = x^q2;  //mealy
//    assign zTEMP=aTEMP&q1; //moore
//    assign aTEMP = x^q2;  //moore

    always @(posedge clk) begin
    q2 = Q2;
    q1 = Q1;
    q0 = Q0;
//    z = zTEMP; // moore
//    a = aTEMP; //moore
    end
endmodule
```

- Behavioral Simulation
  - Behavioral simulation of mealy FSM2 with arbitrary state 110 is **quite true** for all the values except for first one.



- Arbitrary first state comment
  - According to behavioral simulation of the circuit has first state is one of the arbitrary state, the circuit can escape from arbitrary states to encoded states. Note that they can generate false results at the beginning of the simulation.

- Comparing FSM1 and FSM2 about LUT usage.
  - FSM 1 uses 4 LUTs that are 4 LUT4.
  - FSM 2 uses 3 LUTs that are 2 LUT4 and 1 LUT3.
  - It is easy to decide who is winner. **FSM2 design is far better about LUT usage.**

7. DETECTOR

- The code written below is a circuit that outputs 1 when it catches numbers with a pattern of 101101 or 100100.
- If z is 0 it means, there is no pattern.
- If z is 1 it means, we are in the one of the pattern.

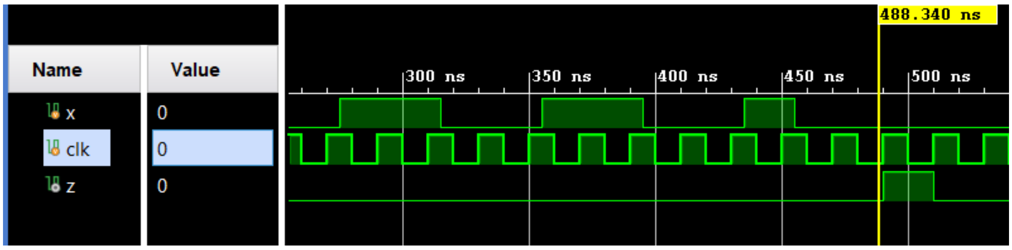
- Verilog Code

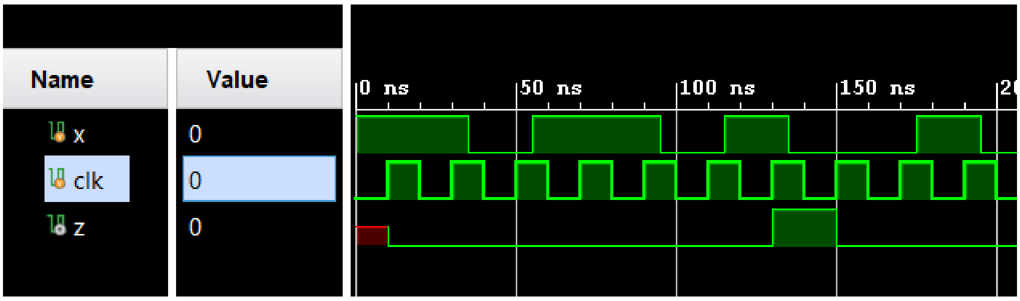
```
`timescale 1ns / 1ps

module detector(
    input x, clk,
    output reg z
    );
    wire Q0 , Q1,Q2, Z;
    reg q0 = 1'b0;
    reg q1 = 1'b0;
    reg q2 = 1'b0;

    assign Q2 = (x & q1 & q0) | (~x & q2 & ~q0);
    assign Q1 = (q1 & ~q0) | ( ~x & ~q2 & ~q1 & q0);
    assign Q0 = (q1 & ~q0) | (x & ~q1) | (q2 & ~q0);
//    assign z = q2 & q0; //mealy
    assign Z = q2 & q0;
    always @ (posedge clk)
    begin
        q2 <= Q2;
        q1 <= Q1;
        q0 <= Q0;
        z <= Z;  //moore
    end
endmodule
```

- Testbench Code

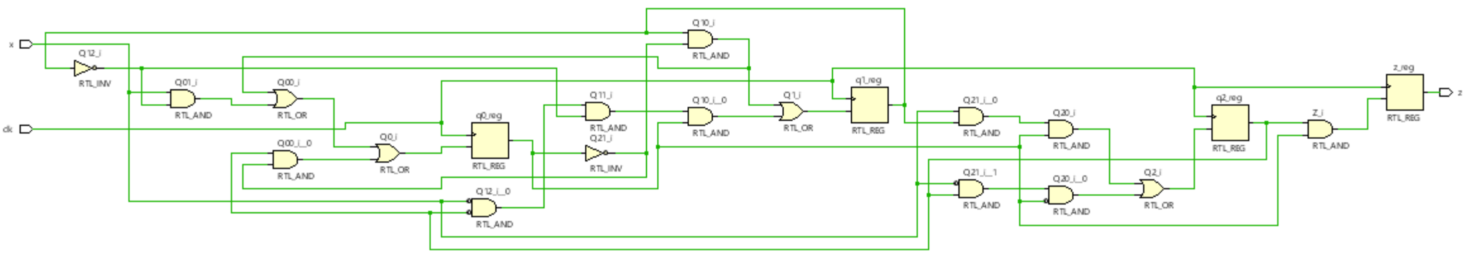
```
`timescale 1ns / 1ps

module detector_tb( );
   reg x;
   reg clk;
   wire z;

   detector  DUT (
       .x(x),
       .z(z),
       .clk(clk)
       );

   always
   begin
       clk = ~clk; #10;
   end

   initial
   begin
       clk = 1'b0;
       x = 1'b1; #15;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b1; #20;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b1; #20;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b1; #20;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b1; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
       x = 1'b0; #20;
   end
endmodule
```

- Behavioral Simulations
  - ➢ Detector detects after 100100 in 7th clock cycle. Detector is working successfully.
  - ➢ Detector detects after 101101 in 7th clock cycle. Detector is working successfully.

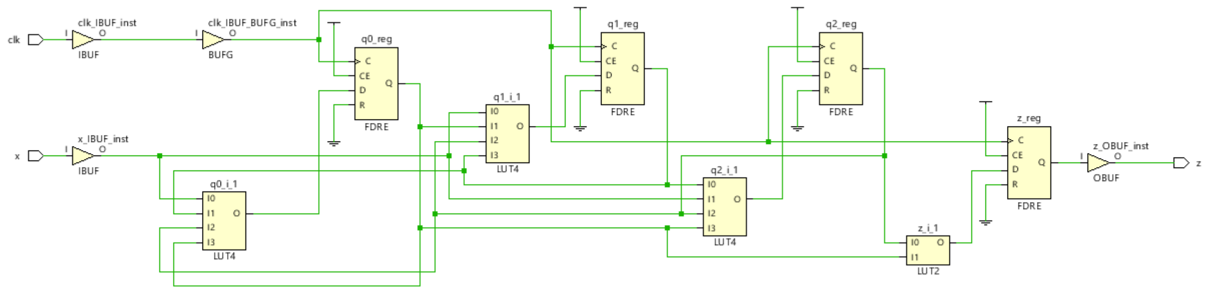




- RTL Schematic



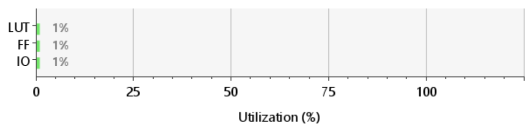
- Technology Schematic



- Utilization Summary

Summary

Resource	Utilization	Available	Utilization %
LUT	2	32600	0.01
FF	4	65200	0.01
IO	3	210	1.43



- Timing summary

Unconstrained Paths - NONE - NONE - Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	De
Path 1	∞	2	1	1	z_reg/C	z	4.891	3.226	1.664	∞		
Path 2	∞	2	1	3	x	q1_reg/D	2.085	1.090	0.995	∞	input port clock	
Path 3	∞	2	1	3	x	q0_reg/D	2.081	1.090	0.991	∞	input port clock	
Path 4	∞	2	1	3	x	q2_reg/D	2.080	1.085	0.995	∞	input port clock	
Path 5	∞	2	1	4	q2_reg/C	z_reg/D	1.440	0.743	0.697	∞		

Unconstrained Paths - NONE - NONE - Hold

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Soı
Path 6	∞	2	1	4	q0_reg/C	q2_reg/D	0.357	0.184	0.173	-∞	
Path 7	∞	2	1	4	q0_reg/C	q1_reg/D	0.359	0.186	0.173	-∞	
Path 8	∞	2	1	4	q0_reg/C	z_reg/D	0.367	0.183	0.184	-∞	
Path 9	∞	2	1	4	q0_reg/C	q0_reg/D	0.370	0.186	0.184	-∞	
Path 10	∞	2	1	1	z_reg/C	z	1.663	1.331	0.331	-∞	