EHB436E

DIGITAL SYSTEM DESIGN APPLICATIONS

2022 FALL

Muhammed Erkmen

040170049

EXPERIMENT-6 REPORT

State Reduction in sequential circuits:

State reduction is the method of recuding number of flip-flops in a sequential circuits.

While recuding, we look for that, is there any states giving exact same output and same next states in all inputs. If there is not, we can't reduce the states.

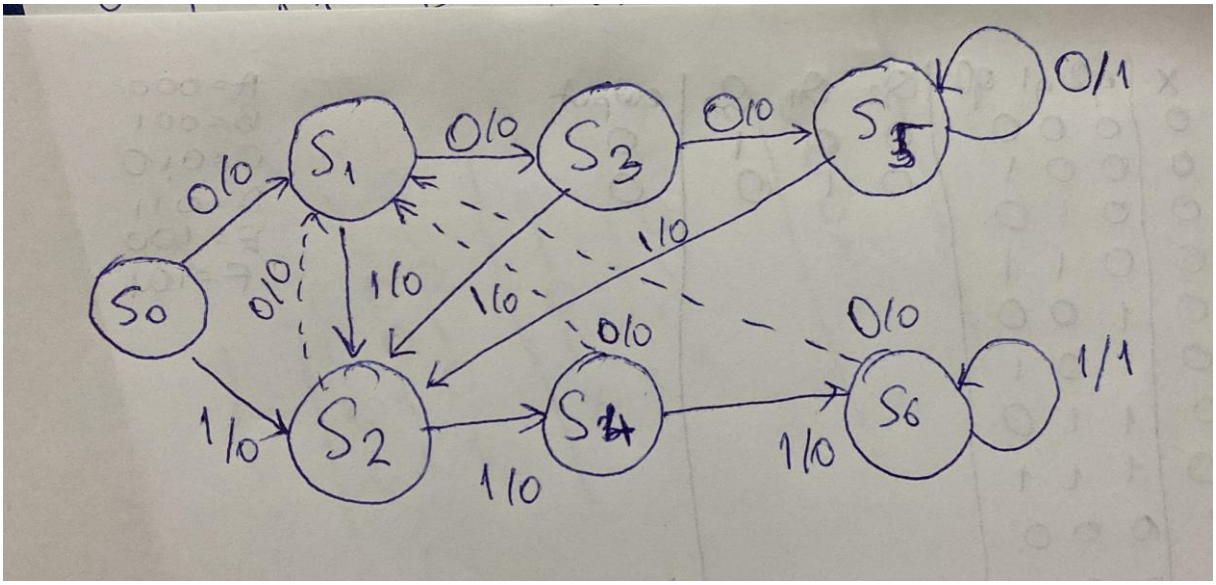
State reduction is not possible in Fig-1 because there is no states that fits these requirements. Every state goes a different output / state in this system as we can see in my state diagram.

State Encoding:

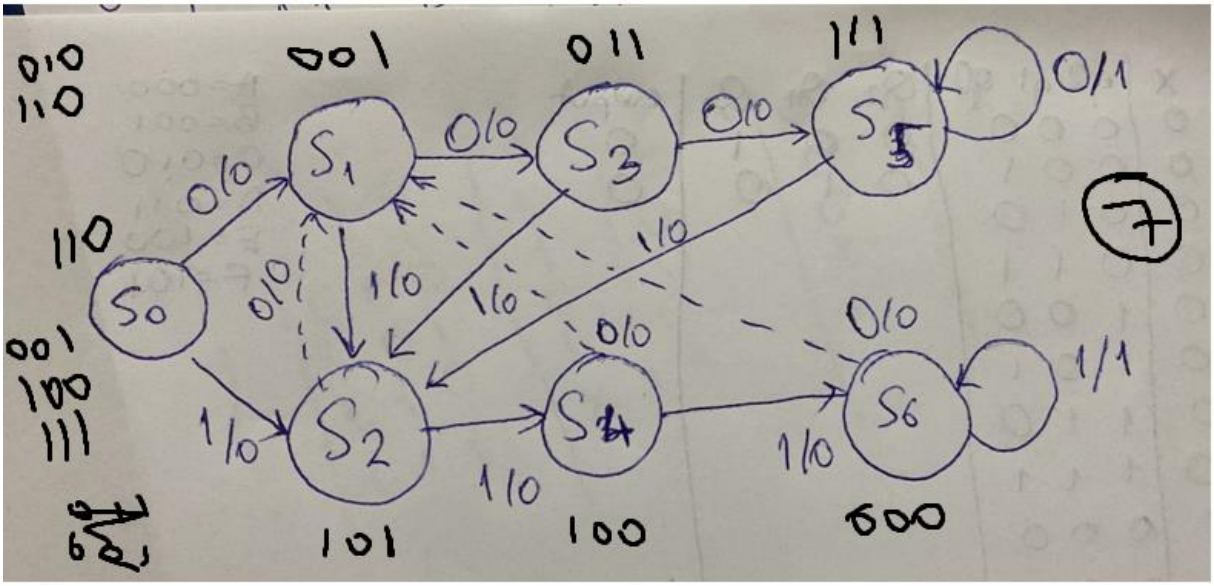
State encoding is the method of symbolising the circuits' states with binary encoded numbers. There are 3 types of state encoding: Binary, Gray and One-Hot Encoding. One-Hot encoding is the fastest method but needs more area. Gray encoding is the optimal, binary encoding is not efficient.

For 8 states here is how they done:

Binary Encoding	Gray Encoding	One-Hot Encoding
SB	Gray	Value
000	000	00000001
001	001	00000010
010	011	00000100
011	010	00001000
100	110	00010000
101	111	00100000
110	101	01000000
111	100	10000000



Here is my state diagram for the circuit. S0 is the starting state and system never comes back this starting state. To make state diagram efficiency good, we should change bits between states as possible as we can.



In here, there is 7 link is changing by just 1 bit. S0 is starting state so it is not that important because never going to come that state again

Reduction of combinatorial parts:
Next State Truth Table:

x	q2	q1	q0	Q2	Q1	Q0	z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	0
0	0	1	0	x	x	x	x
0	0	1	1	1	1	1	0
0	1	0	0	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	0
0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	x	x	x	x
1	0	1	1	1	0	1	0
1	1	0	0	0	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	1	0
1	1	1	1	1	0	1	0

My State Encoding:

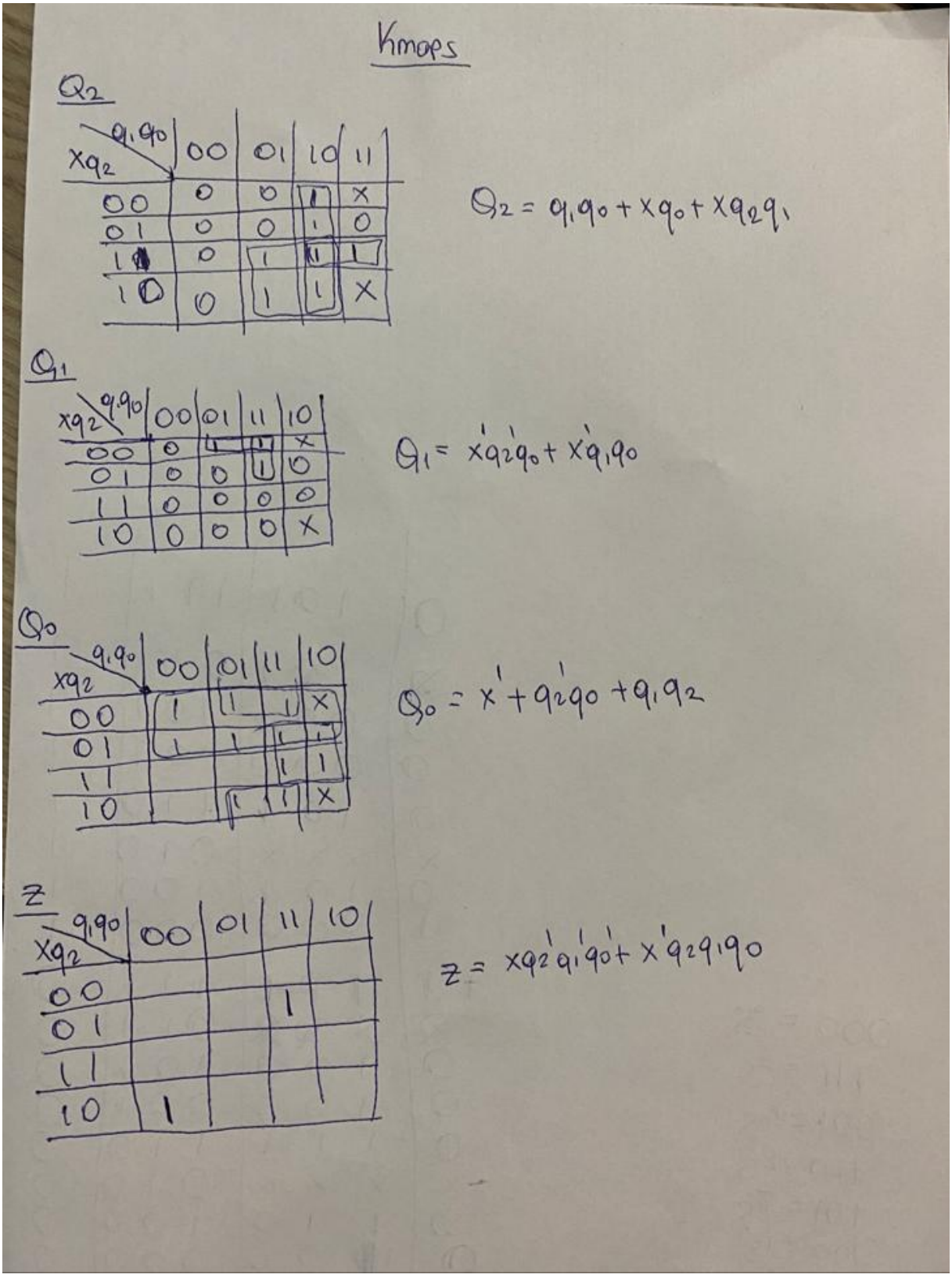
A= 110
B= 001
C = 011
D = 111
E = 101
F = 100
G= 000

My State Table:

Current State	X=0	X=1
A	B,0	E,0
B	C,0	E,0
C	D,0	E,0
D	D,1	E,0
E	B,0	F,0
F	B,0	G,0
G	B,0	G,1

As we can see, we can’t reduce this states. Because of the 6 states is shown by at least 3 bits, so i added the starting state.

Kmaps:


Kmaps
Q2
Q2 = q1q0 + xq0 + xq2q1
Q1
Q1 = x'q2'q0 + x'q1q0
Q0
Q0 = x' + q2'q0 + q1q2
Z
Z = xq2'q1'q0' + x'q2q1q0

We can make the less **LUT4 cells with less states**. If we want to use less LUT cells, we should minimize the states in binary. If we can implement an FSM with 2 binary, it will use less LUT cells then a FSM with 3 binary.

Source code of Mealy machine:

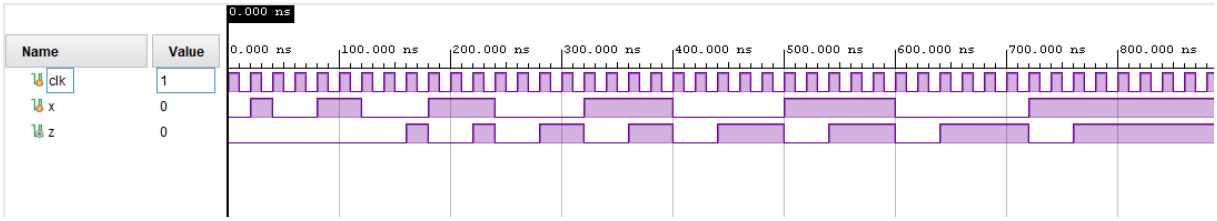
```
`timescale 1ns / 1ps

module FSM1(
   input x,
   input clk,
   output z

   );


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

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

In this machine, i calculate the next states with q0,q1 and q2 wires, than in a posedge of clock, these values goes to Q0,Q1,Q2 registers.

Testbench code of both machines:

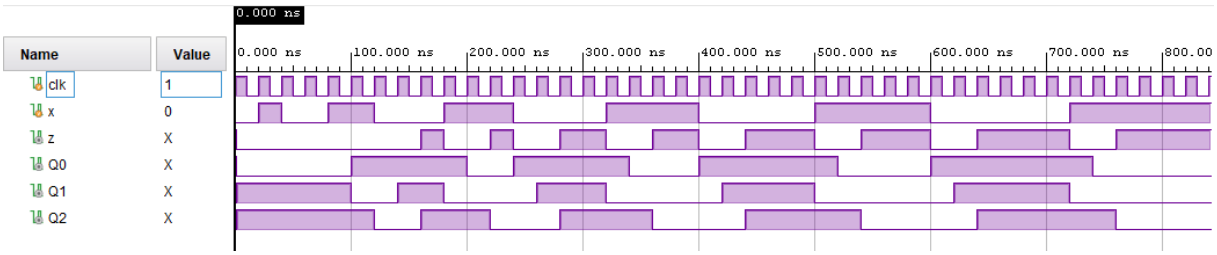
```
`timescale 1ns / 1ps
module FSM1_tb;

reg clk=0;
reg x;
wire z;

FSM1 fatihsultanmehmet1 (.clk(clk),.x(x),.z(z));

always begin
clk = ~clk;
#10;
end

initial begin
x=0;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;

end
endmodule
```

Behavioral simulation results of Mealy:



There are faulty outputs that when 3 0 or 1 comes, z goes 1. But we wanted that result when 4 0 or 1 comes.

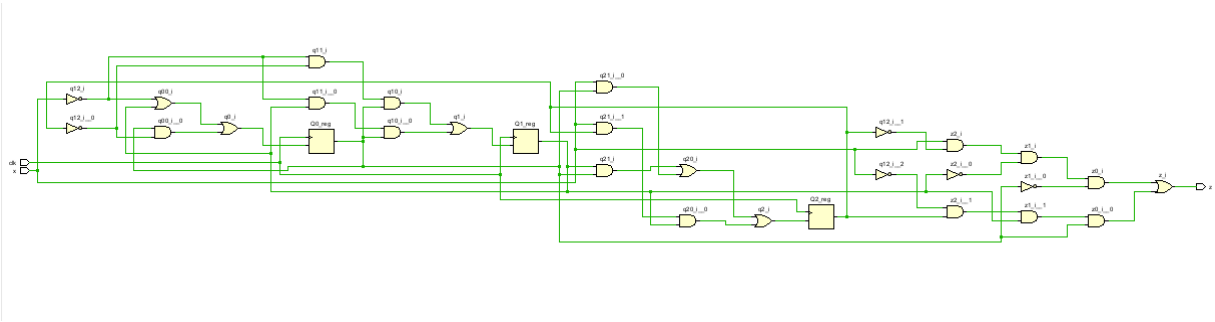
Post implementation functional simulation results:



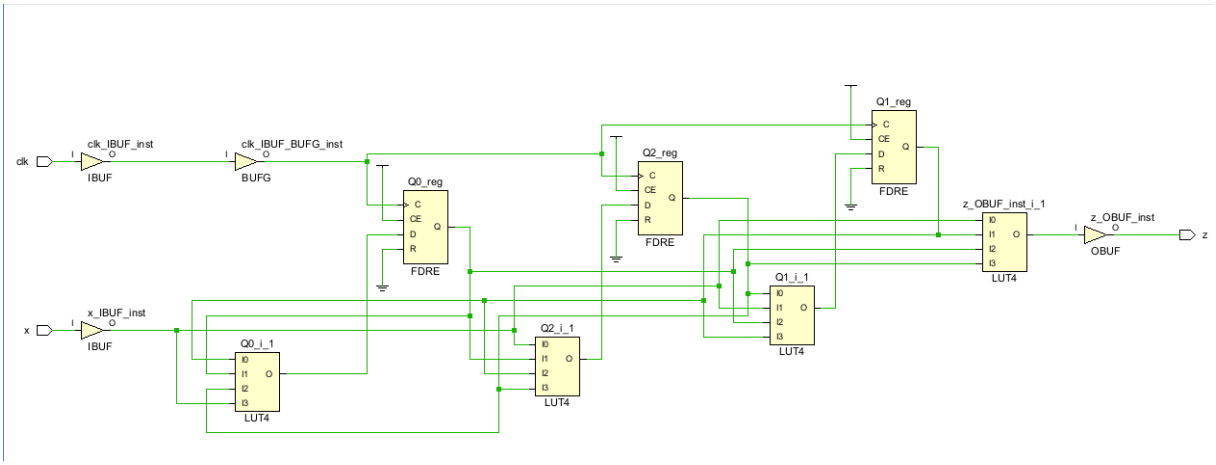
There are still faulty outputs in post implementation functional simulation result. But there is no spike or false output in my circuit because my circuit has an initial state and given clock is suitable for the circuit.

Following constraint added for implementation:
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]

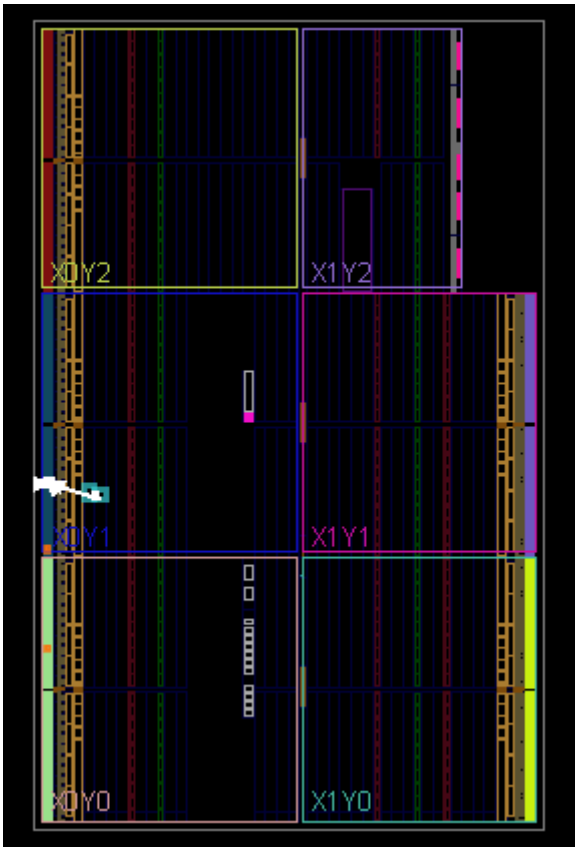
RTL Schematic of FSM1:



Technology Schematic of FSM1:



Device Layout:



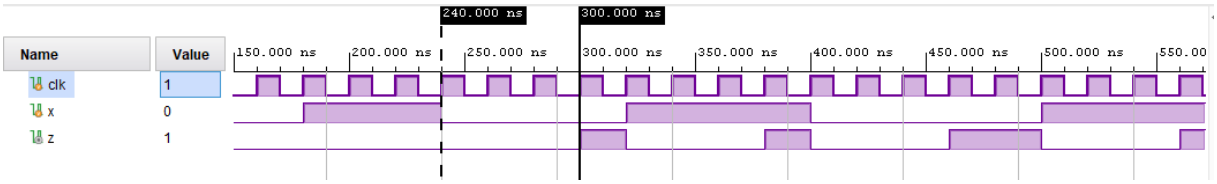
Machine change & prevention methods:

We can make another FF and update this in posedges so there will be no faulty output. In the experiment sheet, we'll do that by changing machine style with 1 clock cycle delay using another FF. Mealy machines give output when the input comes, but moore machines give the output when next state comes. And this is excatly what a FF does, delays output 1 clock cycle.

Source code of Moore machine:

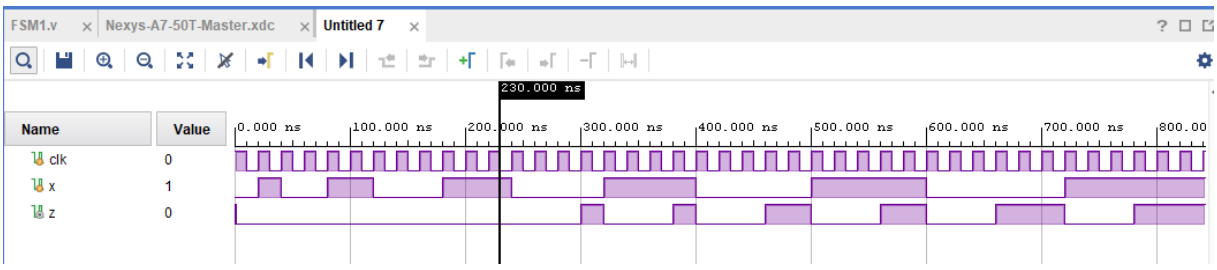
```
`timescale 1ns / 1ps

module FSM1(
   input x,
   input clk,
   output reg z

   );


wire q2,q1,q0;
reg Q2;
reg Q1;
reg Q0;
wire zout;
assign q0 = !x | (Q1) | (Q0&!Q2);
assign q1 = (!x & !Q2 & Q0) | (!x & Q1 & Q0);
assign q2 = (Q1 & Q0) | (x&Q0) | (x & Q2 & Q1);
assign zout = ((x & !Q2&!Q1&!Q0) | (!x&Q2&Q1&Q0));
always @(posedge clk) begin
Q2<=q2;
Q1<=q1;
Q0<=q0;
z <=zout;
end
endmodule
```

Behavioral Simulation results of Moore:



The machine is now moore so our faulties became true outputs. That means we get the output after we did operation.

Post implementation simulation results of Moore:



Now i deleted D flip flop of output and started with my arbitrary state. I just have 2 arbitrary state because i had a starting state. **There is no functional change if i don't start from my starting state, it is just for make everything clear because there is already 3 bit is in use.**

My formulas:

Q0 = !x | (q1) | (q0&!q2);
Q1 = (!x & !q2 & q0) | (!x & q1 & q0);
Q2 = (q1 & q0) | (x&q0) | (x & q2 & q1);

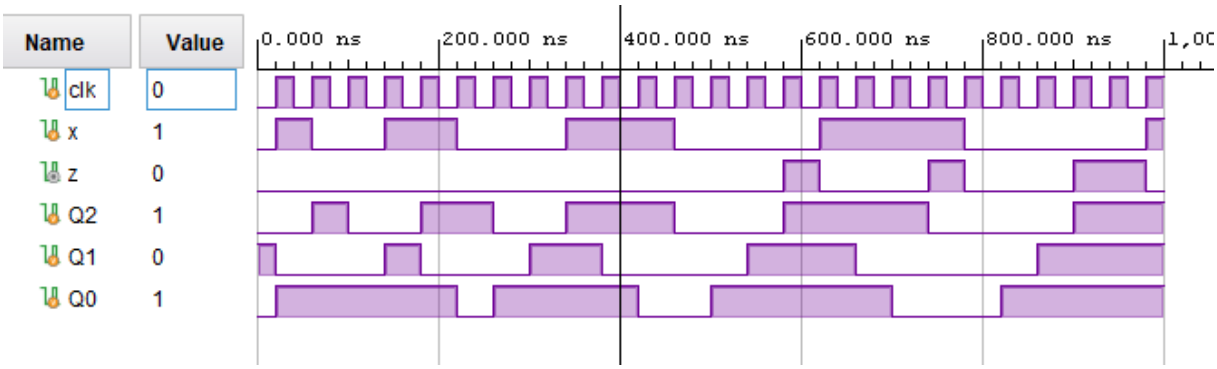
My arbitrary state is 010.

When x=0 in 010:
Q0 = 1 | 1 | 1 = 1,
Q1 = 0 | 0 = 0
Q2 = 0 | 0 | 0 = 0

So the state after x=0 is 0 0 1. This state is my FIRST 0 state B and it is not stuck, also works *fine*.

When x=1 in 010:
Q0 = 0 | 1 | 1 = 1,
Q1 = (0&1&0) | (0&1&0) = 0
Q2 = 0 | 0 | 0 = 0

In this state, it is not stuck but not works right. It goes to FIRST 0 state B.

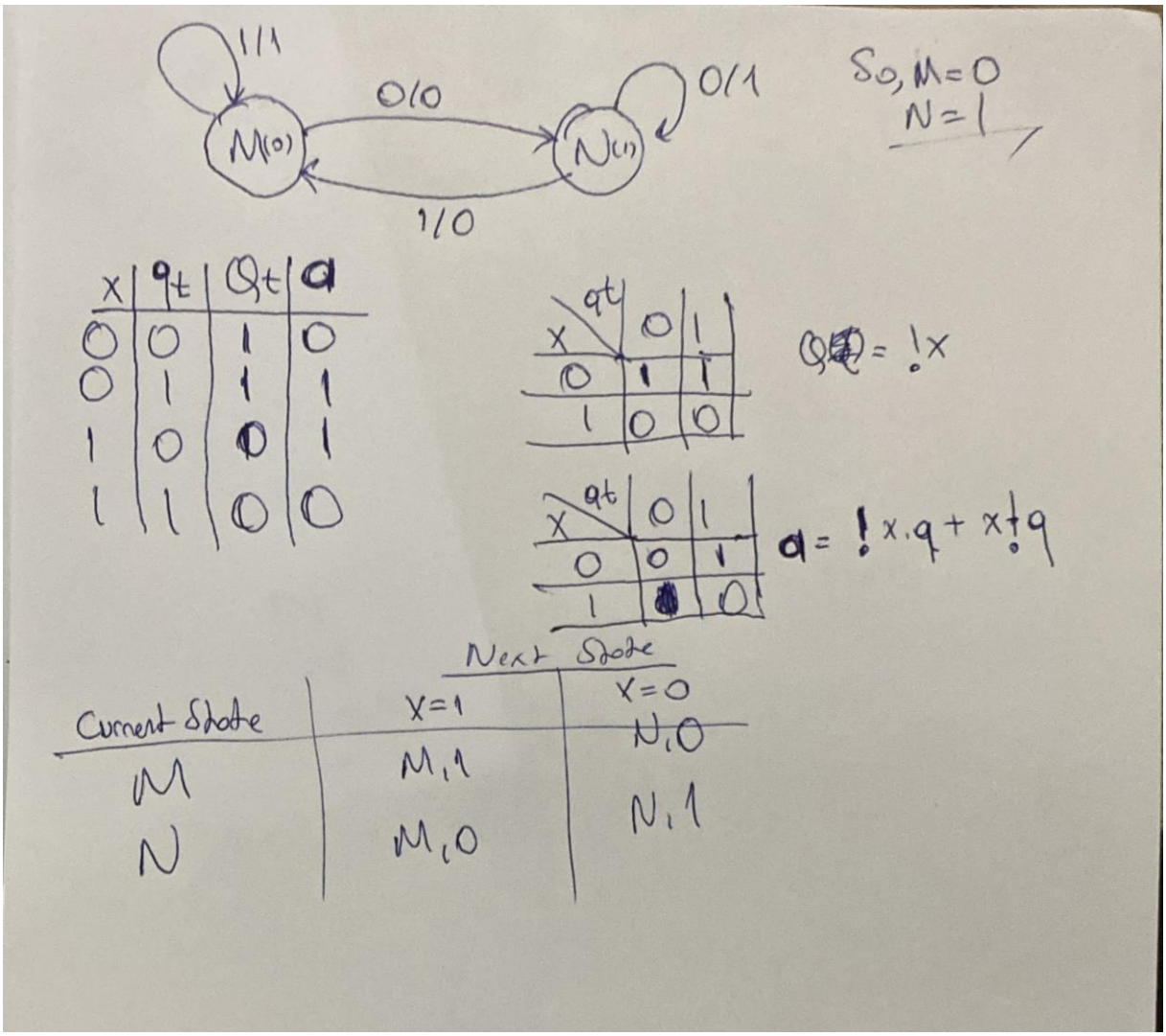


In our input series, i started my system from 010 and it is not stuck.

FSM2:

I made the calculations of Part2 first. In this part x in input and a is output. There is 2 states and not possible to reduce.

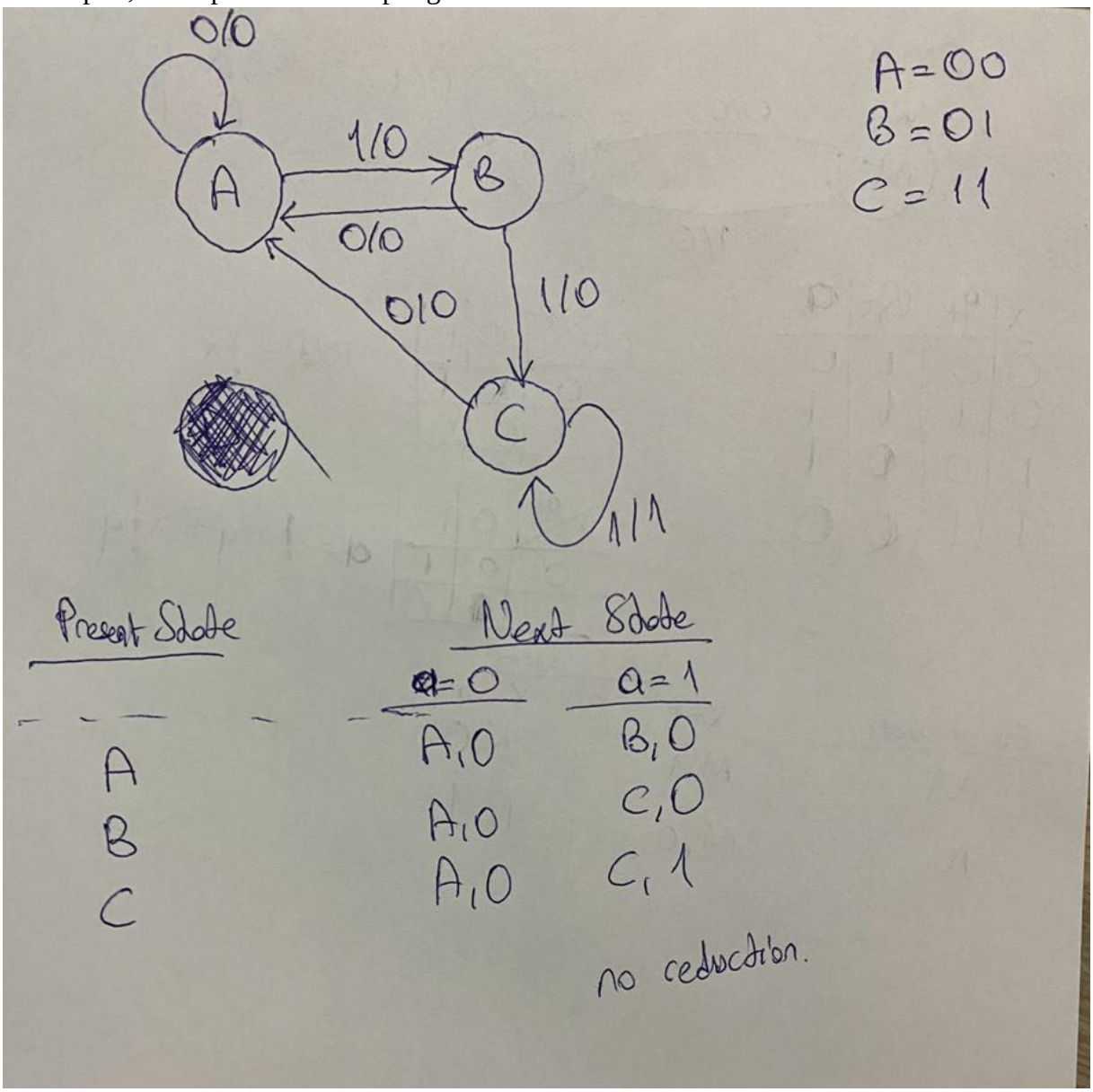
Figure2:



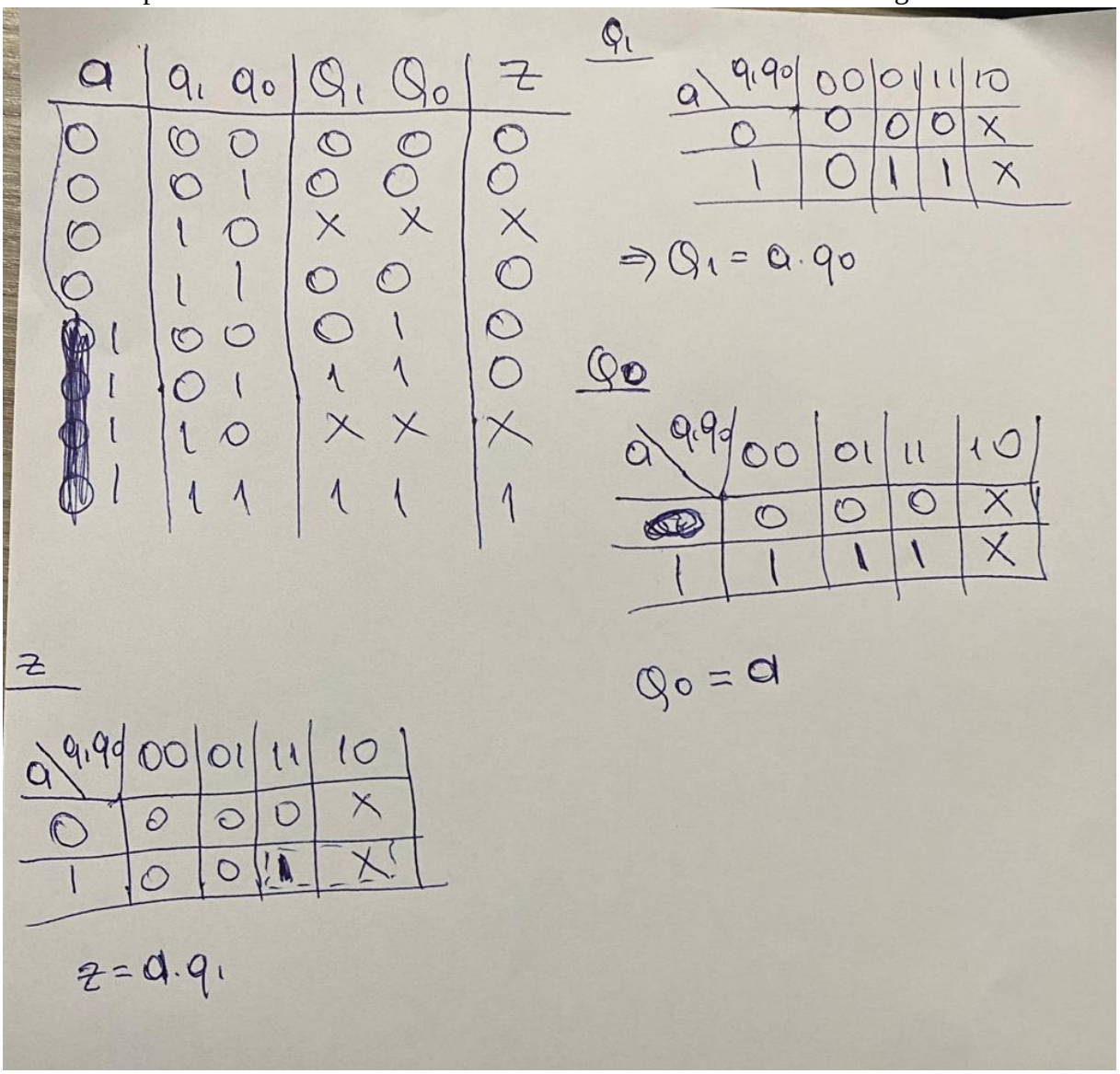
State diagram, State encoding, Current state – Next state table, Kmaps and Q0 formula are written in the given above.

Figure 1:

In this part, a is input and the output goes out z.



There is no possible reduction here too. The truth table and other tables are given below.



So in Figure 1:

Q1 = a & q0;
Q0 = a;
z = a&q1 formulas are here.

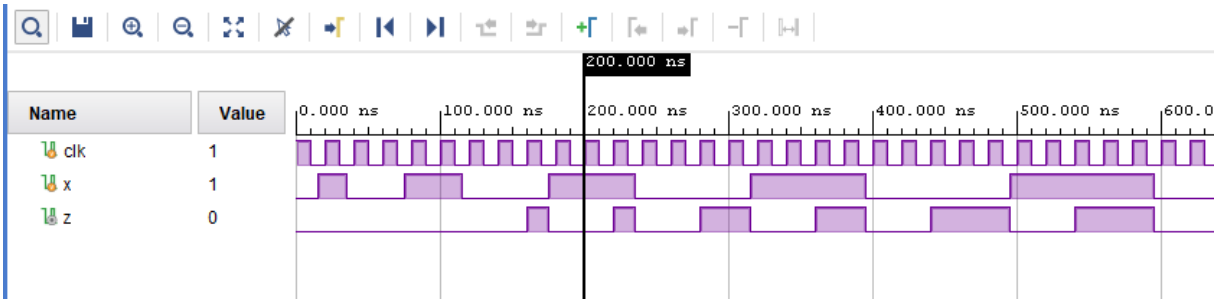
Source code of Mealy machine :

```
`timescale 1ns / 1ps

module FSM2(

    input clk,
    input x,
    output z

    );

// Figure 1 next state things
wire Q1_f1,Q0_f1;
reg q1_f1=0;
reg q0_f1=0;
// Figure 2 next state things
wire Q_f2,a;
reg q_f2=0;

// Figure 1
assign Q1_f1 = a & q0_f1;
assign Q0_f1= a;
assign z = a&q1_f1;
// Figure 2
assign a = (!x & q_f2) | (x & !q_f2);
assign Q_f2 = !x;

always @(posedge clk) begin
q_f2 <= Q_f2;
q1_f1 <= Q1_f1;
q0_f1 <= Q0_f1;
end
endmodule
endmodule
```

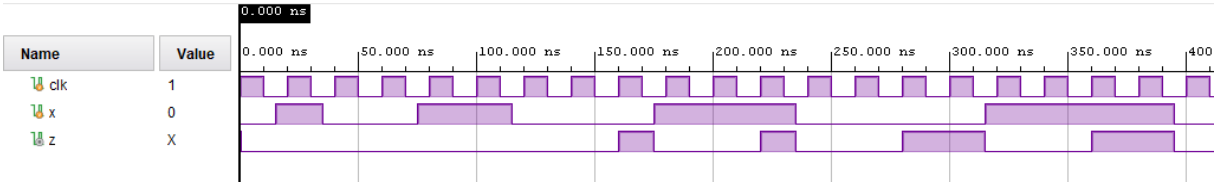
Testbench code of both Mealy and Moore:

```
`timescale 1ns / 1ps
module FSM1_tb;

reg clk=0;
reg x;
wire z;

FSM1 fatihsultanmehmet1 (.clk(clk),.x(x),.z(z));

always begin
clk = ~clk;
#10;
end

initial begin
x=0;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=0;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;
x=1;#20;

end
endmodule
```

Behavioral Simulation:



When 3 0s or 1s came, z immediately goes 1 as expected. There are faulty outputs in this behavioral simulation too. I did the implementation and analyze the post-implementation functional simulation:



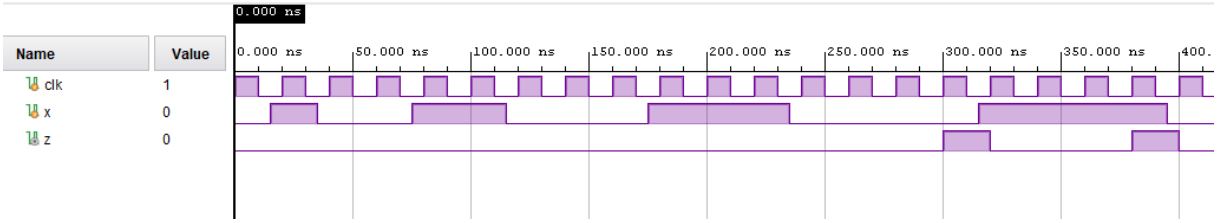
There are faulty outputs in this too.

Machine change:

Source code of Moore:

```
`timescale 1ns / 1ps

module FSM2(

    input clk,
    input x,
    output reg z

    );

// Figure 1 next state things
wire Q1_f1,Q0_f1;
reg q1_f1=0;
reg q0_f1=0;
// Figure 2 next state things
wire Q_f2,a;
reg q_f2=0;

// Figure 1
assign Q1_f1 = a & q0_f1;
assign Q0_f1= a;
assign zn = a&q1_f1;
// Figure 2
assign a = (!x & q_f2) | (x & !q_f2);
assign Q_f2 = !x;

always @(posedge clk) begin
q_f2 <= Q_f2;
q1_f1 <= Q1_f1;
q0_f1 <= Q0_f1;
z <= zn;
end
endmodule
```

Behavioral Simulation of Moore:



As we can see, there is one clock cycle delay here too. There are no faulty outputs.

Starting from arbitrary states:

My arbitrary state is 10 on fig1.

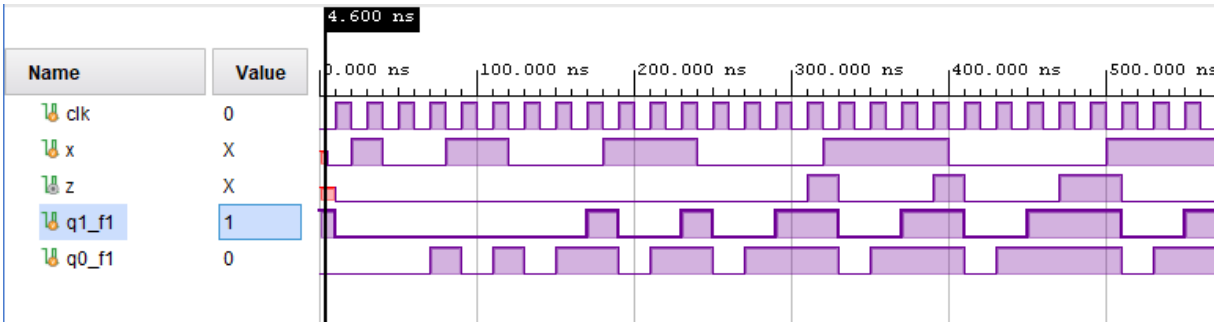
Formulas:

Q1 = a & q0;
Q0 = a

When a =0

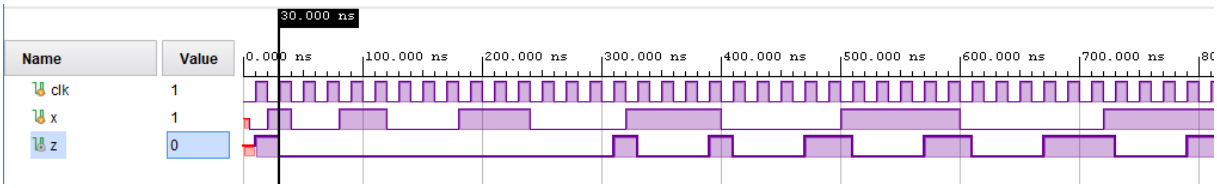
Q1 = 0 & 0 = 0;
Q0 = 0 ; so the machine is not stucking also works right. Z is also goes 0.

As we can see below it is not stuck in arbitrary state 10.



When a = 1:

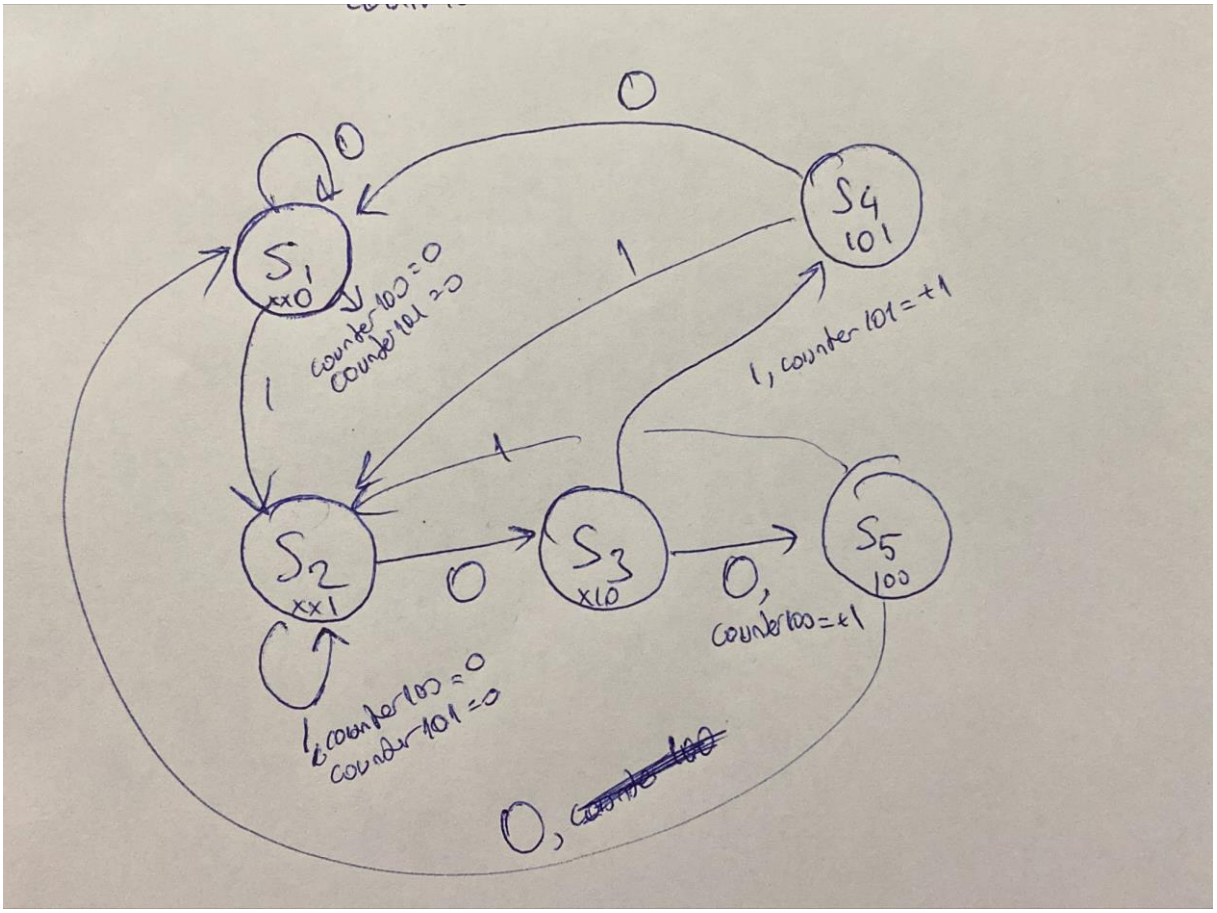
Q1 = 1 & 0 = 0
Q0 = 1 = 1 so machine is not stucking again but works wrong because Z goes 1. As we can see below.



FSM3

I designed this FSM with always and case structures.
This FSM created like
S1=> xx0
S2=> xx1.
S3=> x10.
S4=> 101.
S5=> 100.

They are encoded by using one hot encoding.



State coding:

S1 = 00001; // xx0
S2 = 00010; // xx1
S3 = 00100; // x10
S4 = 01000; // 101
S5 = 10000; // 100.

State Table

Present State	Next State (x=0)	Next State (x=1)
S1	S1	S2
S2	S3	S2
S3	S5	S4
S4	S1	S2
S5	S1	S2

.

Source code:

```
`timescale 1ns / 1ps
module FSM3(
    input x,
    input clk,
    output reg z
    );
localparam S1 = 5'b00001;      // xx0
localparam S2 = 5'b00010;        // xx1
localparam S3 = 5'b00100;        // x10
localparam S4 = 5'b01000;      // 101
localparam S5 = 5'b10000;        // 100

reg [3:0] state=S1;
reg [1:0] counter101=0;
reg [1:0] counter100=0;
always @(posedge clk) begin
case(state)

    S1:              //xx0
    begin
    counter101<=0;
    counter100<=0;
    case(x)
    0:
        begin
        state <= S1;
        end
    1:
        begin
        state <= S2;
        end
    endcase
    end

    S2:              //xx1
    case(x)
    0:
        begin
        state <= S3;
        end
    1:
        begin
```

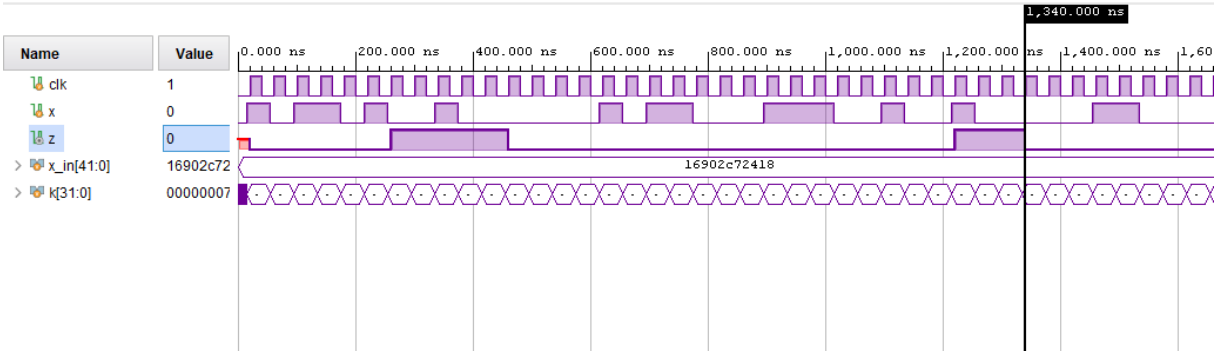
```
        state <= S2;
        counter101<=0;
        counter100<=0;
        end
    endcase

    S3:             //x10
    begin
    case(x)
    0:
        begin
        state <= S5;
        counter101<=0;
        counter100<=counter100+1;
        end
    1:
        begin
        state <= S4;
        counter100<=0;
        counter101<=counter101+1;
        end
    endcase
    end

    S4:             //100
    begin
    case(x)
    0:
        begin
        state <= S1;
        end
    1:
        begin
        state <= S2;
        end
    endcase
    end

    S5:             //101
    begin
    case(x)
    0:
        begin
        state <= S1;
        end
    1:
        begin
        state <= S2;
        end
    endcase
    end
endcase

if(counter101==2 || counter100==2)
z<=1;
else
z<=0;


end
endmodule
```

Testbench code:

```
`timescale 1ns / 1ps

module fsm3_tb;

reg clk=0;
reg x;
wire z;

FSM3 fatihsultanmehmet (.clk(clk),.x(x),.z(z));
reg [41:0] x_in =42'b00_0101_101_0010_0000_010110_0011_1000_0101_1000_11000;
always begin
   #20;
   clk=~clk;
end
integer k;
initial begin
   x<=x_in[41];
   #15;
   for(k=40;k>= 0;k=k-1) begin
      x<=x_in[k];
      #40;
   end
   #20;
   $finish;
end
endmodule
```

Inside the code, i got 2 counters which counts 101 and 100. It goes to 0 in required links as i drew in my state diagram. Z will not go to 0 if the 2 101 came and the flow goes through another 101. But when the flow is broken, z goes 0 again too.

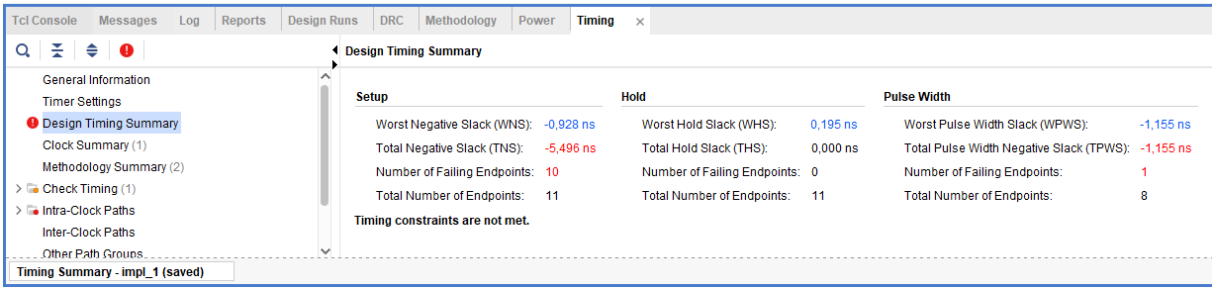
Behavioral Simulation Results:



So here is 101101 given, with the last 1, 100100 given and works right. After that 101100 given and circuit is working right. Everything looks fine.

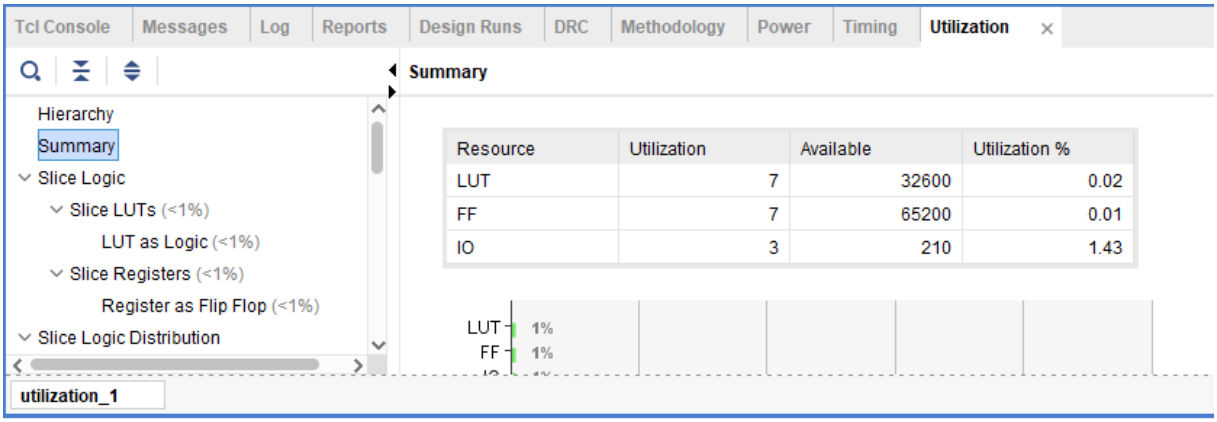
```
## Clock signal
##set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 1.00 -waveform {0 0.5} [get_ports {clk}];

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
##Switches
set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports { x }]
```

So i implemented in T=1ns 1GHz clock. WNS came as -0.928 nanosecond

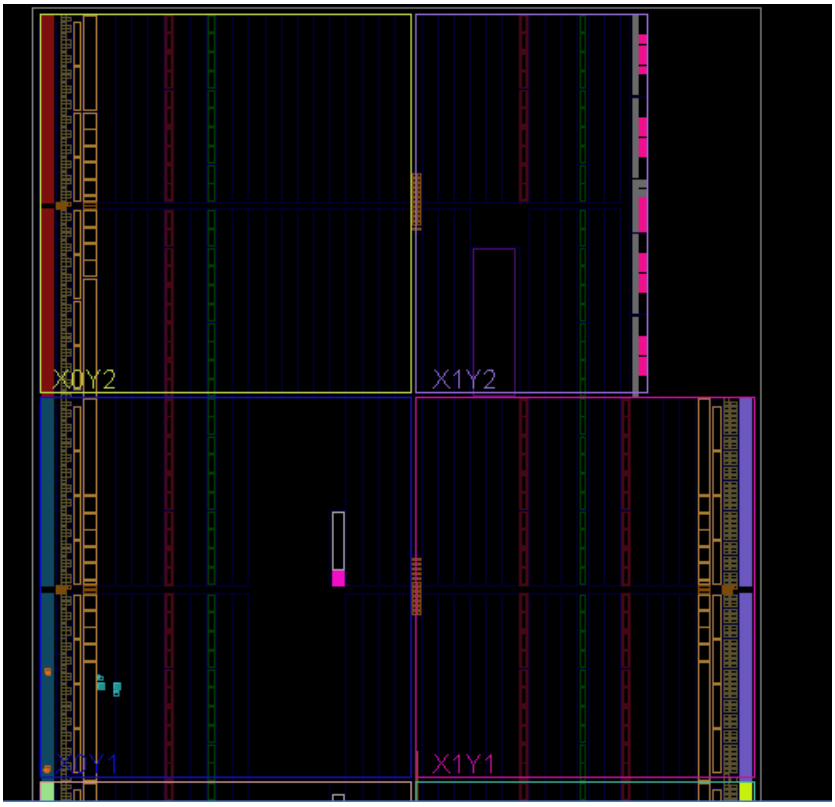


So my maximum frequency is 1/(1+0,928)nS and it is equal to 503 MHz

Utilization Report:



Device:



In this one, i used JUST CASE statements but counters.

I designed the same circuit by using case and if statements together:

I used again 1ns period clock (1GHz)

And the timing result is:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-0,359 ns	Worst Hold Slack (WHS):	0,177 ns	Worst Pulse Width Slack (WPWS):	-1,155 ns
Total Negative Slack (TNS):	-1,155 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	-1,155 ns
Number of Failing Endpoints:	6	Number of Failing Endpoints:	0	Number of Failing Endpoints:	1
Total Number of Endpoints:	6	Total Number of Endpoints:	6	Total Number of Endpoints:	7

Timing constraints are not met.

So that means 735,83 MHz.

And utilization report of 735.83 MHz is:

Summary

Resource	Utilization	Available	Utilization %
LUT	6	32600	0.02
FF	6	65200	0.01
IO	3	210	1.43

Source code of 735.83 MHz module:

```
`timescale 1ns / 1ps
module FSM3(
    input x,
    input clk,
    output reg z
    );
localparam S1 = 5'b00001;      // xx0
localparam S2 = 5'b00010;        // xx1
localparam S3 = 5'b00100;        // x10
localparam S4 = 5'b01000;      // 101
localparam S5 = 5'b10000;        // 100

reg [4:0] state=S1;
reg [1:0] counter101=0;
reg [1:0] counter100=0;
always @(posedge clk) begin
case(state)

    S1:             //xx0
    begin
        if(!x)
        state <= S1;
        else
        state <= S2;
    end

    S2:             //xx1
    if(!x)
        begin
        state <= S3;
        end
    else
        begin
        state <= S2;
        counter101<=0;
        counter100<=0;
        end

    S3:             //x10
    begin
    if(!x)
```

```
        begin
        state <= S5;
        counter101<=0;
        counter100<=counter100+1;
        end
    else
        begin
        state <= S4;
        counter100<=0;
        counter101<=counter101+1;
        end
    end

    S4:             //100
    begin
    if(!x)
        begin
        state <= S1;
        end
    else
        begin
        state <= S2;
        end
    end

    S5:             //101
    begin
    if(!x)
        begin
        state <= S1;
        end
    else
        begin
        state <= S2;
        end
    end
endcase

if(counter101==2)
    begin
    z<=1;
    end
else if(counter100==2)
    begin
    z<=1;
    end
else
z<=0;




end
endmodule
```

Testbenches are the same.