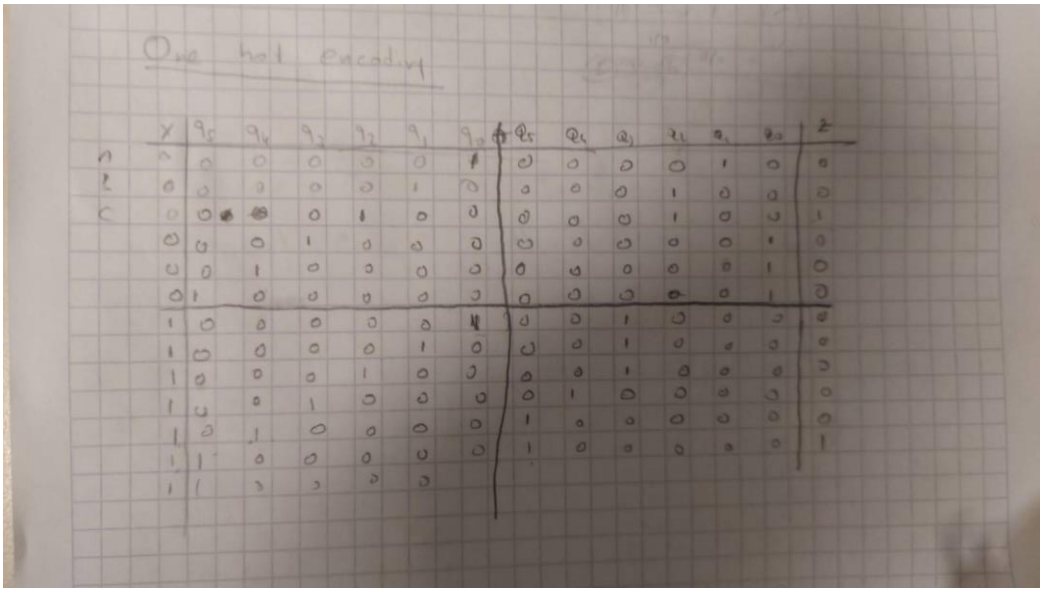
Ömer Faruk SERT

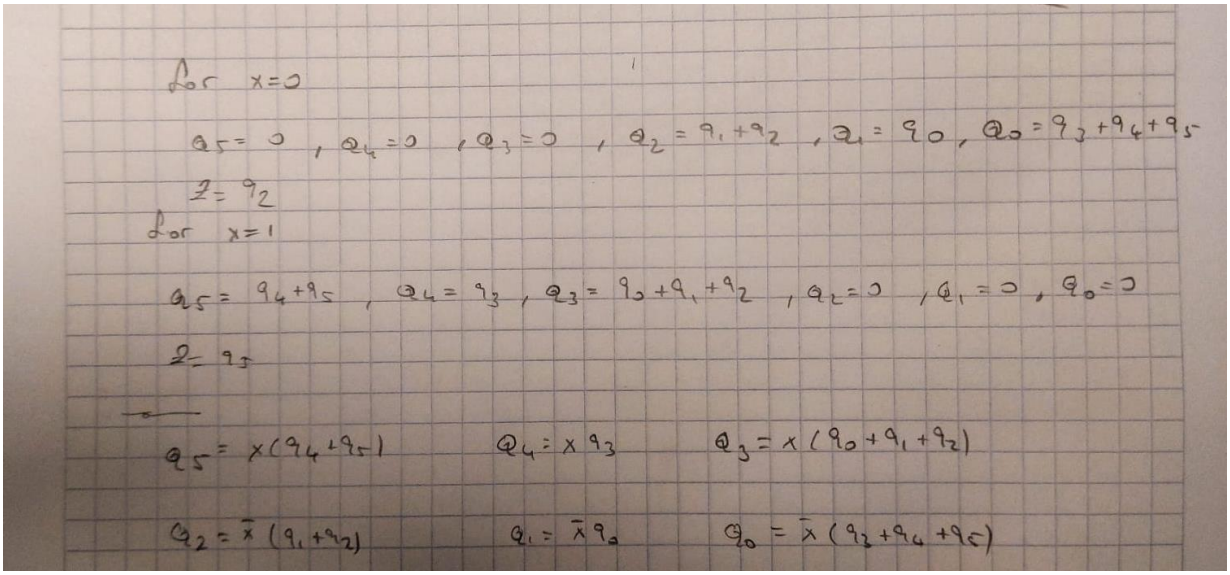
040170244

State Reduction and Encoding

Binary encoding uses less flip flops but more gates. Gray encoding uses more gates but same amount of flip flops, switching activity less in this encoding that's why it uses less power. One hot encoding uses much more flip flops but less gates, in FPGAs has more flip flops then gates that's why it is suitable for it.

For Fig1, we cannot make state reduction because it does not have a representation that we can encode directly.





z=((~x)&q2)|(x&q5)

LUT4

If I want to use only LUT4, then I would want to use least possible gates and it is possible with one hot encoding.

Verilog Code

Module Code

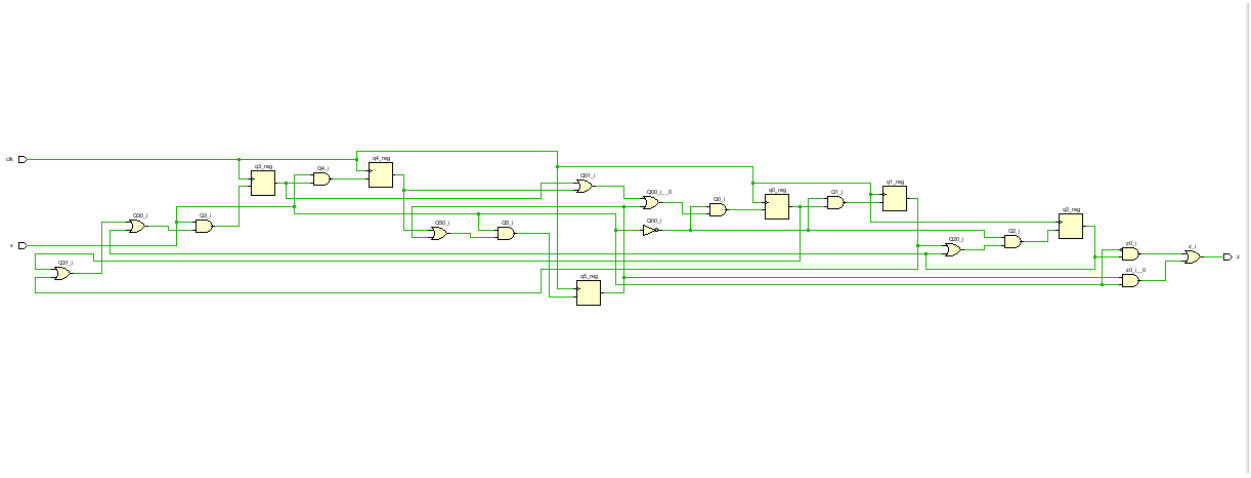
```
`timescale 1ns / 1ps


module FSM1(
        input x,
        input clk,
        output z
    );
    reg q0=1'b1, q1=1'b1, q2=1'b0 , q3=1'b0,  q4=1'b0, q5=1'b0;
    wire Q0,Q1,Q2,Q3,Q4,Q5;
    always@(posedge clk)
    begin
        q0 <= Q0;
        q1 <= Q1;
        q2 <= Q2;
        q3 <= Q3;
        q4 <= Q4;
        q5 <= Q5;
        //z <= ((~x)&q2)|(q5&x);
    end

    assign Q5 = x&(q4|q5);
    assign Q4 = x&q3;
    assign Q3 = x&(q0|q1|q2);
    assign Q2 = (~x)&(q1|q2);
    assign Q1 = (~x)&q0;
    assign Q0 = (~x)&(q3|q4|q5);
    assign z = ((~x)&q2)|(q5&x);

endmodule
```

Testbench Code

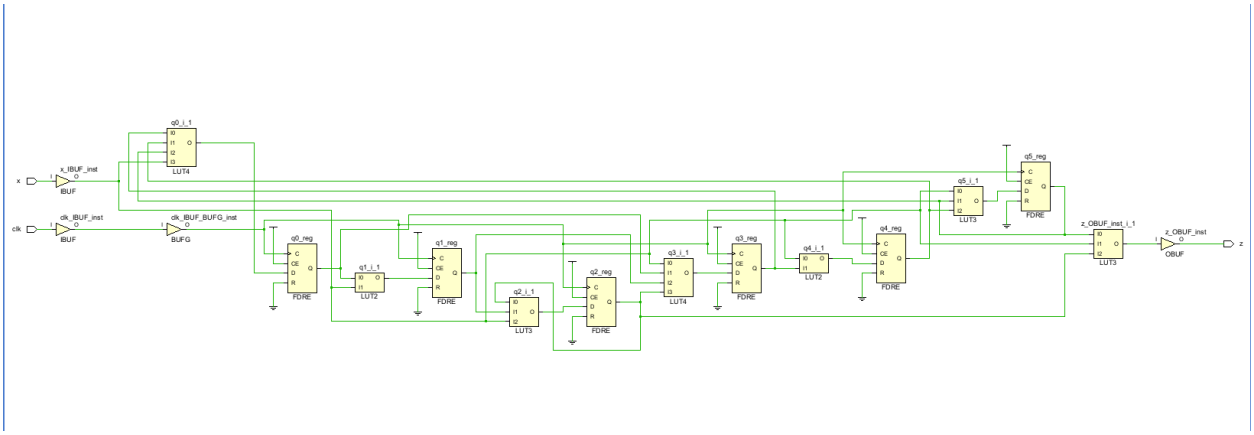
```
`timescale 1ns / 1ps



module experiment6tb;

    reg clk;
    wire z;
    reg x;
    reg [31:0]values;
    reg [9:0]values2;


    FSM1 uut(.x(x),.clk(clk),.z(z));
    integer i;
    initial
    begin

        clk=1'b0;
        values = 32'b01001100011100001111000001111100;
        values2 = 10'b0000111111;
        i=31;
        while(i>=0)
        begin
            clk = ~clk;
            x=values[i];
            #5
            clk <=~clk;
            #5
            i = i-1;
        end
            clk = ~clk;
            x=values2[9];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[8];
            #5
            clk <=~clk;
            #5
```

```
            clk = ~clk;
            x=values2[7];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[6];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[5];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[4];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[3];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[2];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[1];
            #5
            clk <=~clk;
            #5
            clk = ~clk;
            x=values2[0];
            #5
            clk <=~clk;
            #5
    $finish;
    end
endmodule
```

RTL Schematic Mealy Machine



Technology Schematic and Layout Mealy Machine

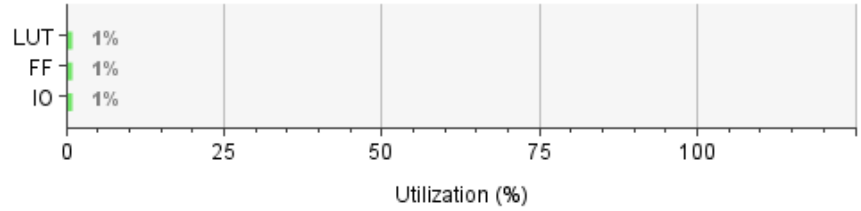



X0Y3
X1Y3
X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0

Utilization Summary Mealy Machine

Summary

Resource	Utilization	Available	Utilization %
LUT	4	63400	0.01
FF	6	126800	0.00
IO	3	210	1.43



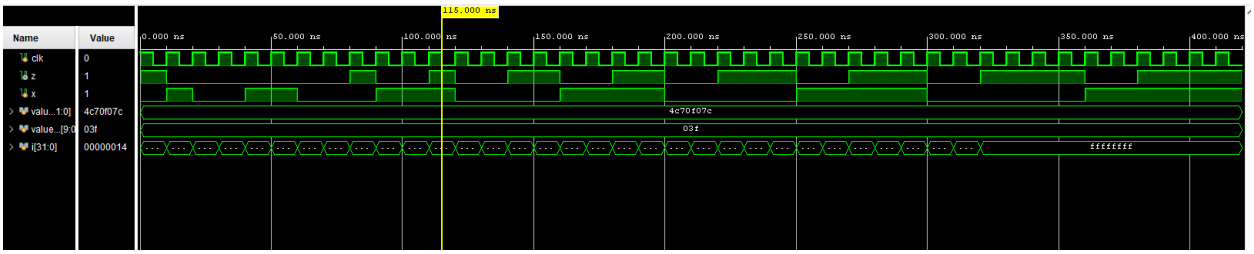
Slic
unu
ice

Combinational Delay Mealy Machine

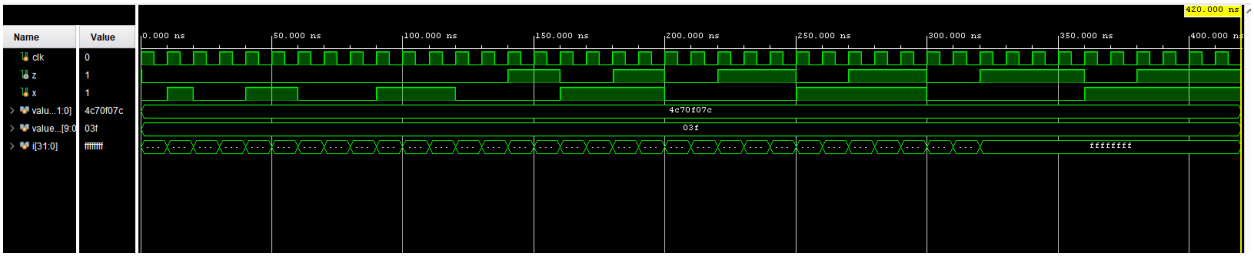
Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
x	z	8.613	SLOW	2.578	FAST

Behavioral Simulation Wave Form Mealy Machine



Post Implementation Functional Simulation Mealy Machine



Analyzing my results

As it can be seen in the simulation results, this circuit is producing faulty 0s and 1s because it is a mealy machine, after 3 successive inputs the next input checked immediately without waiting the next rising edge of the clock. To prevent this, we can add a register to the output so it will gain its value at the rising edge of the clock, this way it won't respond immediately to the input.

Machine Type Changing

Machine is now Moore machine, because output determined by only the state, not with input and state.

Verilog Code Moore Machine

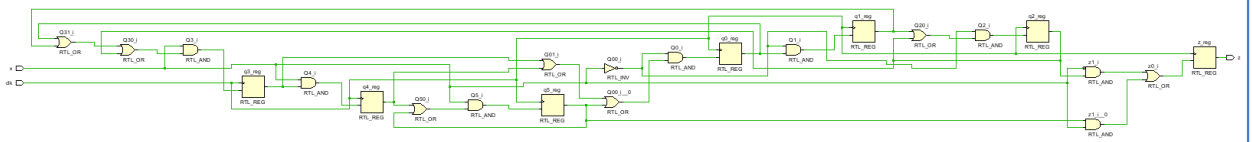
Module Code

```
`timescale 1ns / 1ps


module FSM1(
        input x,
        input clk,
        output reg z
    );
    reg q0=1'b1, q1=1'b0, q2=1'b0 , q3=1'b0,  q4=1'b0, q5=1'b0;
    wire Q0,Q1,Q2,Q3,Q4,Q5;
    always@(posedge clk)
    begin
        q0 <= Q0;
        q1 <= Q1;
        q2 <= Q2;
```

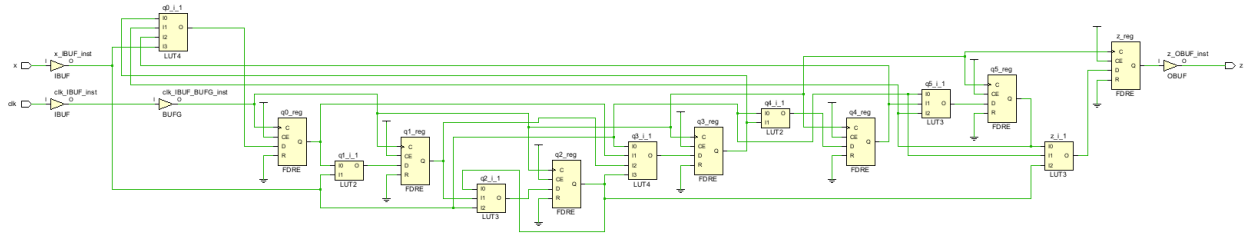
```
        q3 <= Q3;
        q4 <= Q4;
        q5 <= Q5;
        z <= ((~x)&q2)|(q5&x);
    end

    assign Q5 = x&(q4|q5);
    assign Q4 = x&q3;
    assign Q3 = x&(q0|q1|q2);
    assign Q2 = (~x)&(q1|q2);
    assign Q1 = (~x)&q0;
    assign Q0 = (~x)&(q3|q4|q5);
    //assign z = ((~x)&q2)|(q5&x);

endmodule
```

RTL and Technology Schematics Moore Machine

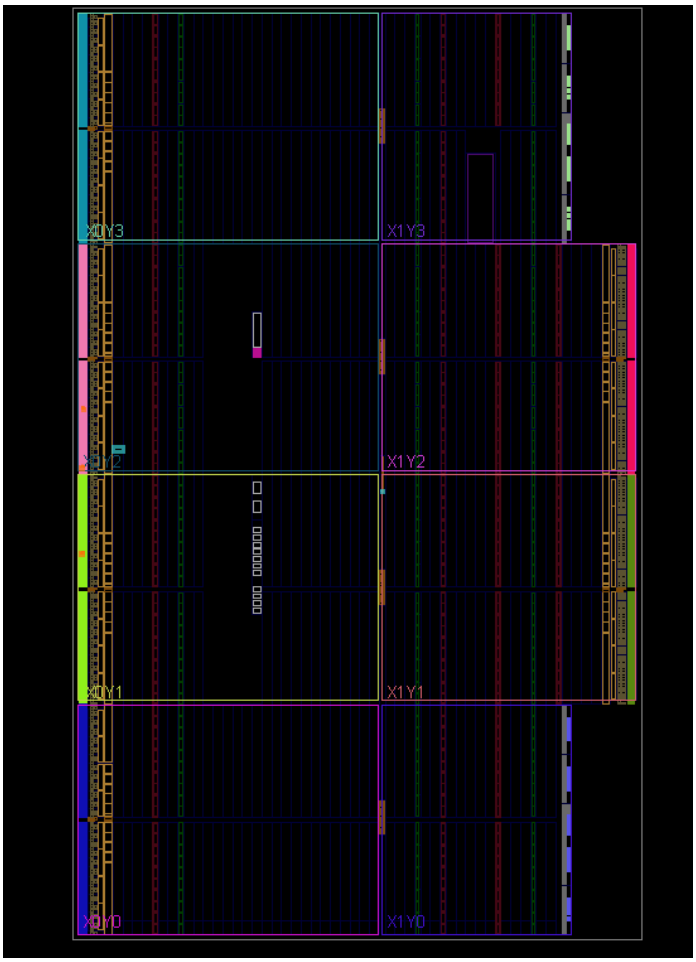
RTL Schematic



Technology Schematic

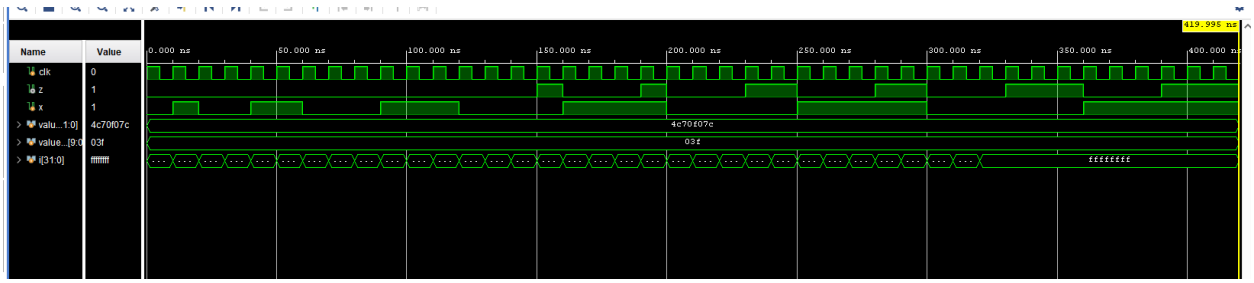


Layout

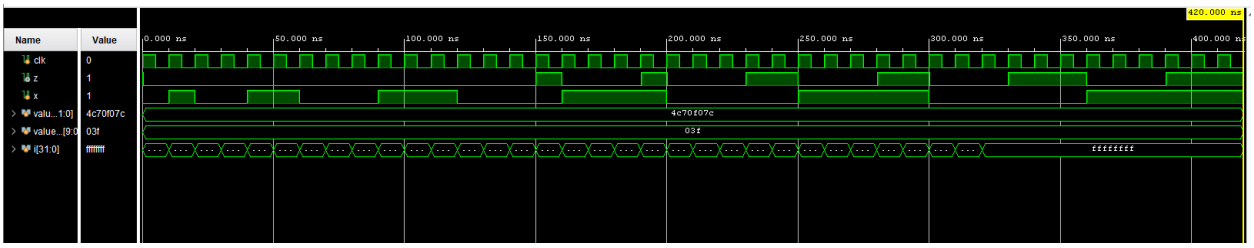


Post Implementation Functional and Behavioral Simulation Moore Machine

Behavioral Simulation Wave Form



Post Implementation Functional Simulation



Analyzing Results

Machine is now Moore machine, so it won't determine the output with input, instead it determines the output with states.

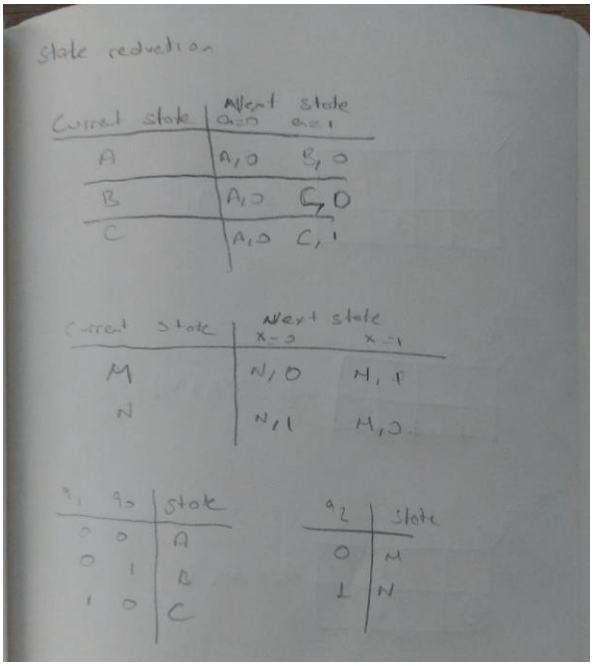
Arbitrary State

When make the initial state 111111, it works opposite till it sees 4 consecutive 1s or 0s, but because of the machine type which is Mealy machine, it responds at the 3rd bit and corrects itself. But if we make the initial state 000000 then the circuit always makes the output zero.

FSM2

Fig5 has the same functionality as Fig1 because in Fig5 it fist detects x is same as the previous x then makes a=1 and on the left hand side of the Fig5 when a is equal to it proceeds counting and after the fourth same bit makes z=1.

State Reduction



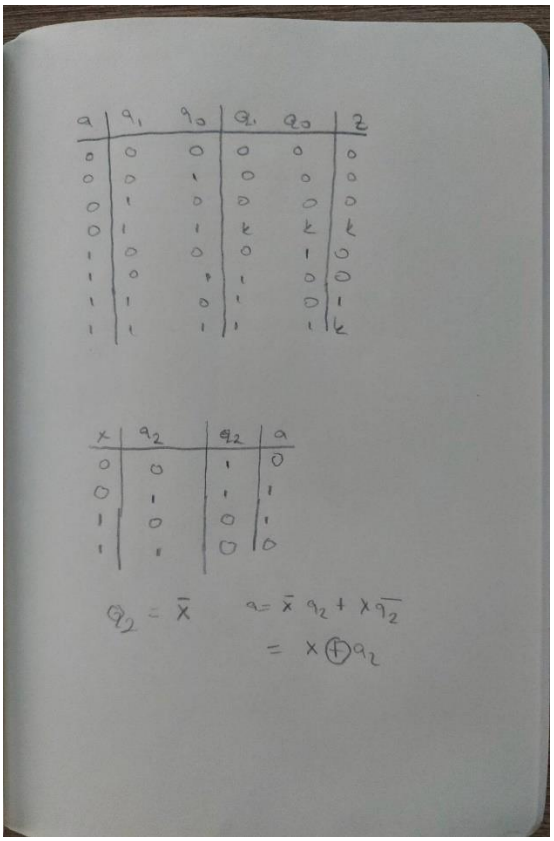
State reduction

Current state	Next state a=0	a=1
A	A, 0	B, 0
B	A, 0	C, 0
C	A, 0	C, 1

Current state	Next state x=0	x=1
M	N, 0	M, 0
N	N, 1	M, 0

q_1	q_0	state
0	0	A
0	1	B
1	0	C

q_2	state
0	M
1	N

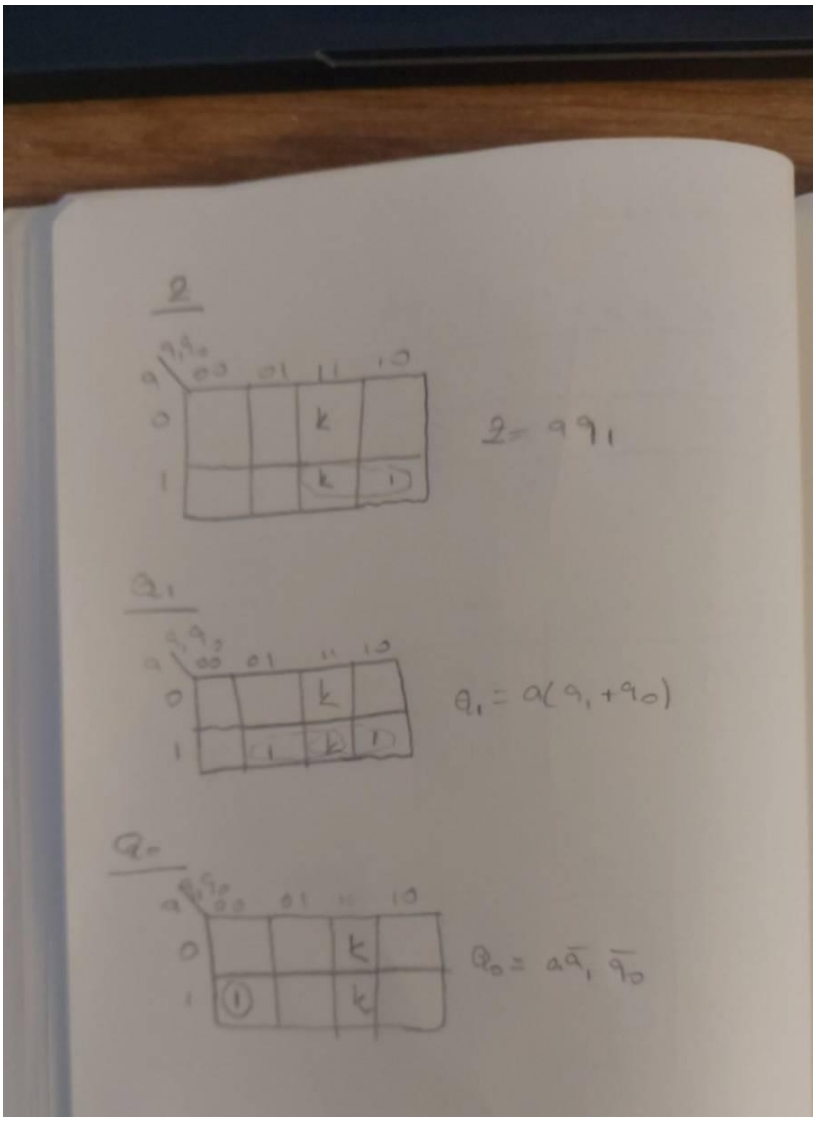


a	q_1	q_0	Q_1	Q_0	Z
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	k	k	k
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	k

x	q_2	q_2	a
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	0

$Q_2 = \bar{x}$ $\quad a = \bar{x}\, q_2 + x\bar{q_2}$

$= x \oplus q_2$

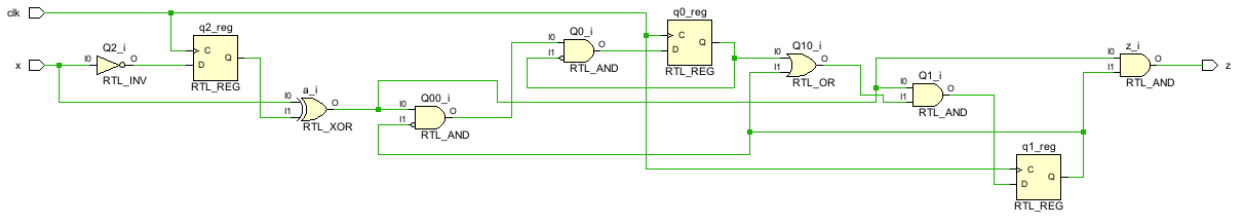


Verilog Code

```
`timescale 1ns / 1ps


module FSM2(
        input x,
        input clk,
        output z
    );
    reg q0=1'b0, q1=1'b0, q2=1'b0 ;
```

```
    wire Q0,Q1,Q2;
    always@(posedge clk)
    begin
        q0 <= Q0;
        q1 <= Q1;
        q2 <= Q2;
    end

    assign a= (x^q2);
    assign Q2 = (~x);
    assign Q1 = (a&(q0|q1));
    assign Q0 = (a&(~q1)&(~q0));
    assign z=(a&q1);

endmodule
```

RTL Schematic



Technology Schematic