EC204

Digital System Design Lab

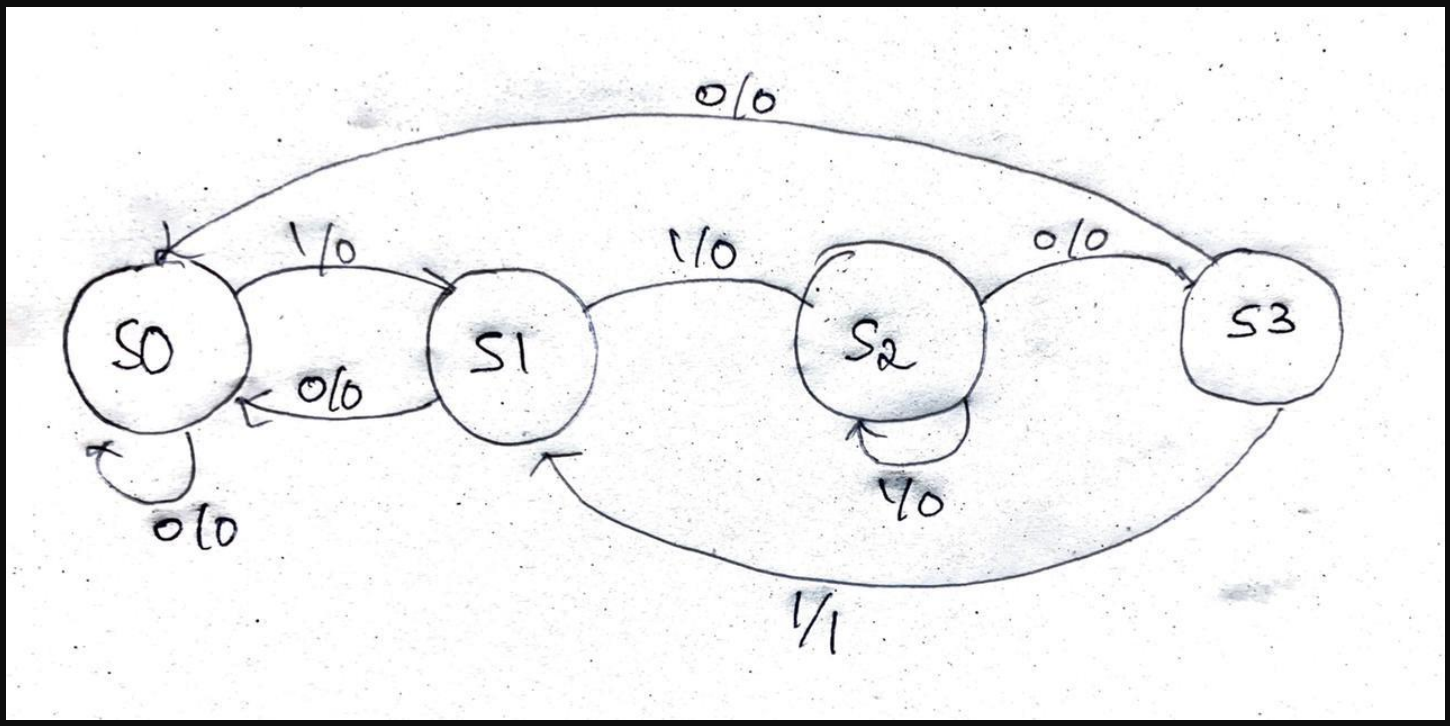
Lab – 8

Utkarsh R Mahajan

201EC164

1] To design and simulate a circuit in Logisim that checks for the sequence 1101 continuously in a data sequence

-> State Diagram considering Mealy fsm(since output depends upon input):



We get 4 states S0, S1, S2 and S3. We will consider S for state, with values for each state as 00, 01, 10, 11 respectively and considering x for input and y for output, We will build a state table for the following.

a) using DFF

-> State Table:

S_1	S_0	X	S_1'	S_0'	Y	D_{s1}	D_{s0}
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0
1	0	1	1	0	0	1	0
1	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0
1	1	1	0	1	1	0	1

Where D_{s1} and D_{s0} are inputs of the D flip flops.

Taking S_1 S_0 and X as A, B, C.

For D_{s1}:

Map

	$\overline{C}$	C
$\overline{A}\overline{B}$	0	0
$\overline{A}B$	0	1
$A\overline{B}$	0	0
AB	1	1

$D_{s1} = AB' + A'BC = S_1S_0' + S_1'S_0X$

For D_{s0}:

Map

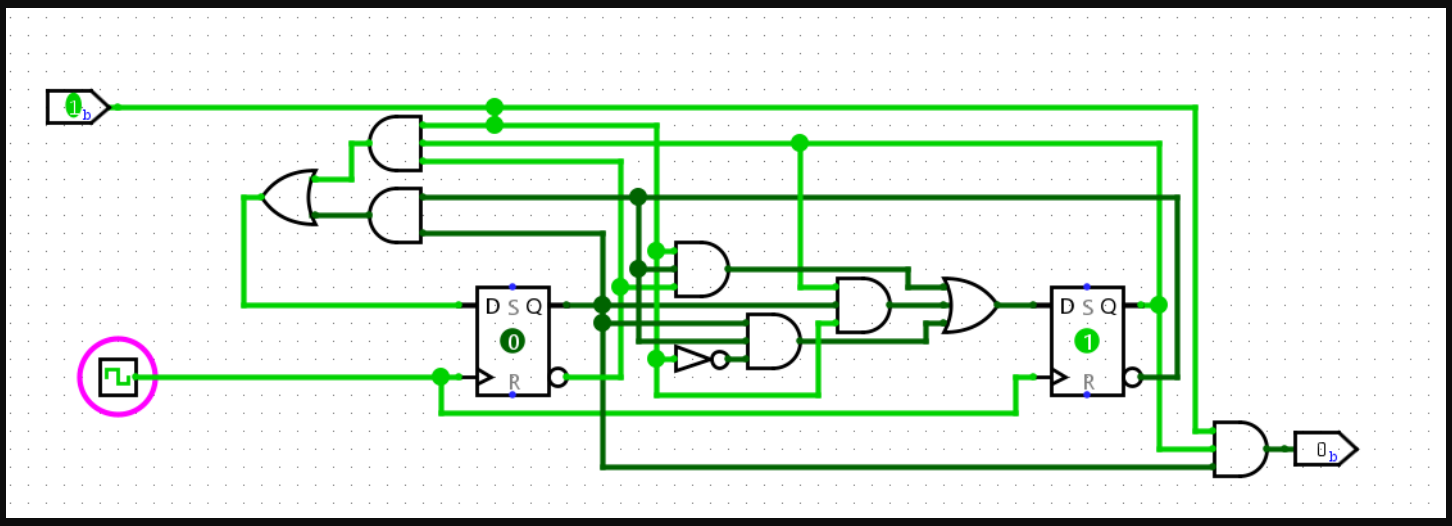
	$\overline{C}$	C
$\overline{A}\overline{B}$	0	1
$\overline{A}B$	0	0
AB	0	1
$A\overline{B}$	1	0

D_{s0} $= A'B'C + AB'C' + ABC = S_1'S_0'X + S_1S_0'X' + S_1S_0X$

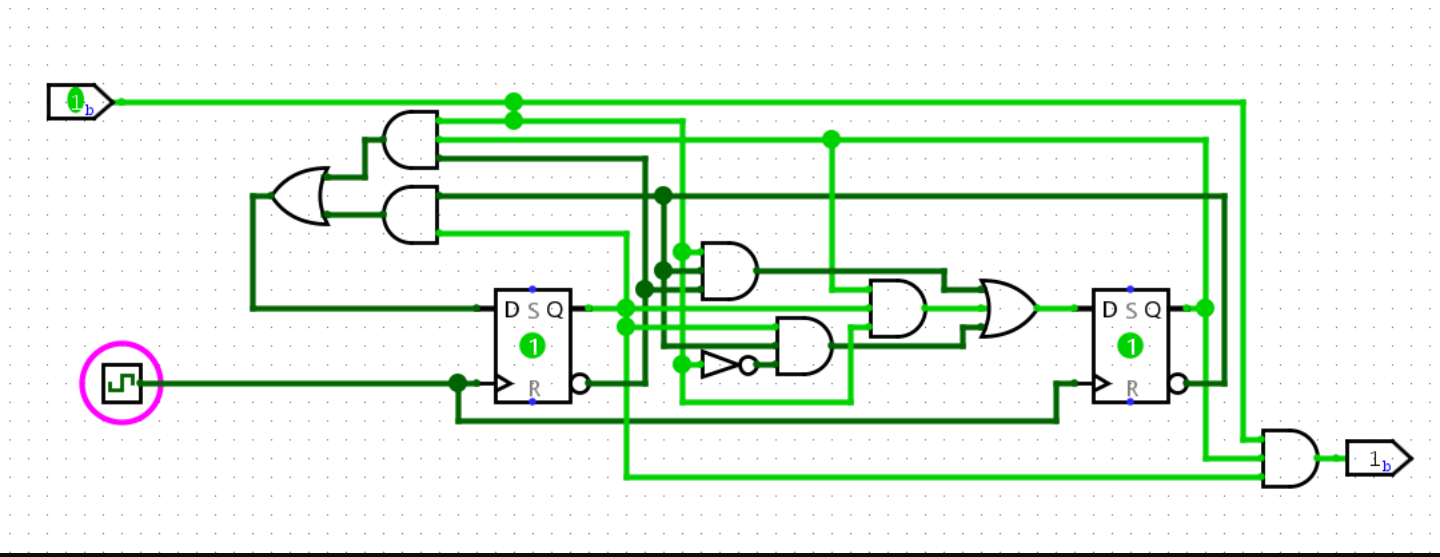
For Y: We can see from the above table that

$Y = S_1S_0X$

We get the following circuit in Logisim:



By following the sequence 1101 as input, We get the required result:



b) using JKFF

-> Similarly, for JK flip flop considering J=K=T for the input we can build the following table:

S_1	S_0	X	S_1'	S_0'	Y	$T_{s1(J=K=T)}$	$T_{s0(J=K=T)}$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	0	0	1
0	1	1	1	0	0	1	1
1	0	1	1	0	0	0	0
1	0	0	1	1	0	0	1
1	1	0	0	0	0	1	1
1	1	1	0	1	1	1	0

Where T_{s1} and T_{s0} are inputs of the JK flip flops.

For Y: We can see from the above table that

$Y = S_1 S_0 X$

Taking S_1 S_0 and X as A, B, C.

Map

	$\overline{C}$	C
$\overline{A}.\overline{B}$	0	0
$\overline{A}.B$	0	1
A.B	1	1
$A.\overline{B}$	0	0

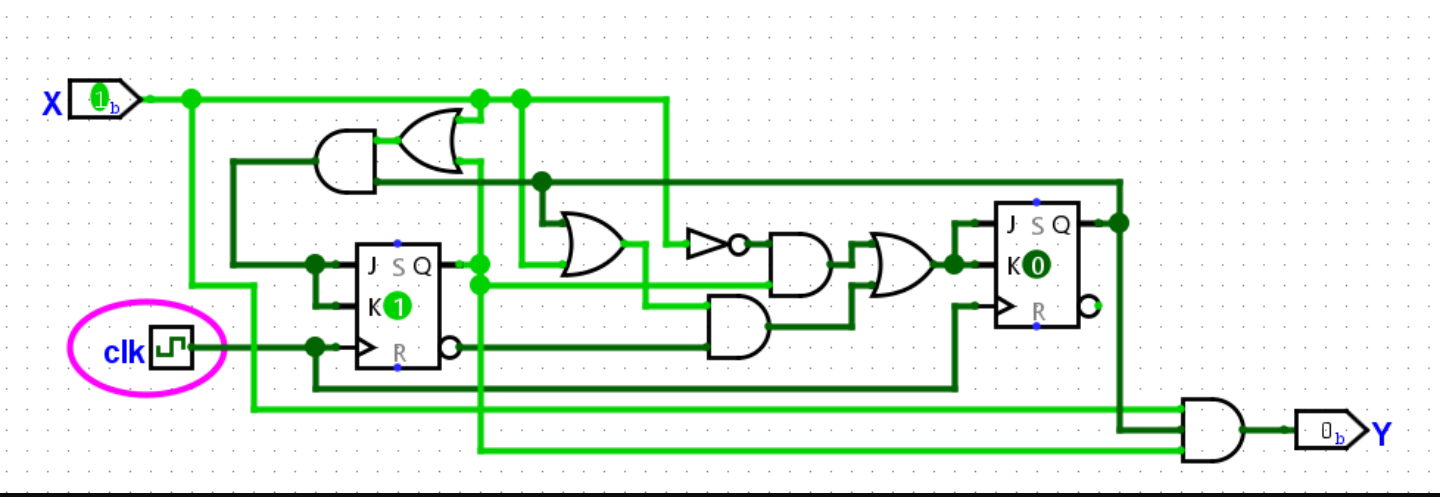
$T_{s1} = BC + AB = S_0X + S_1S_0$

Map

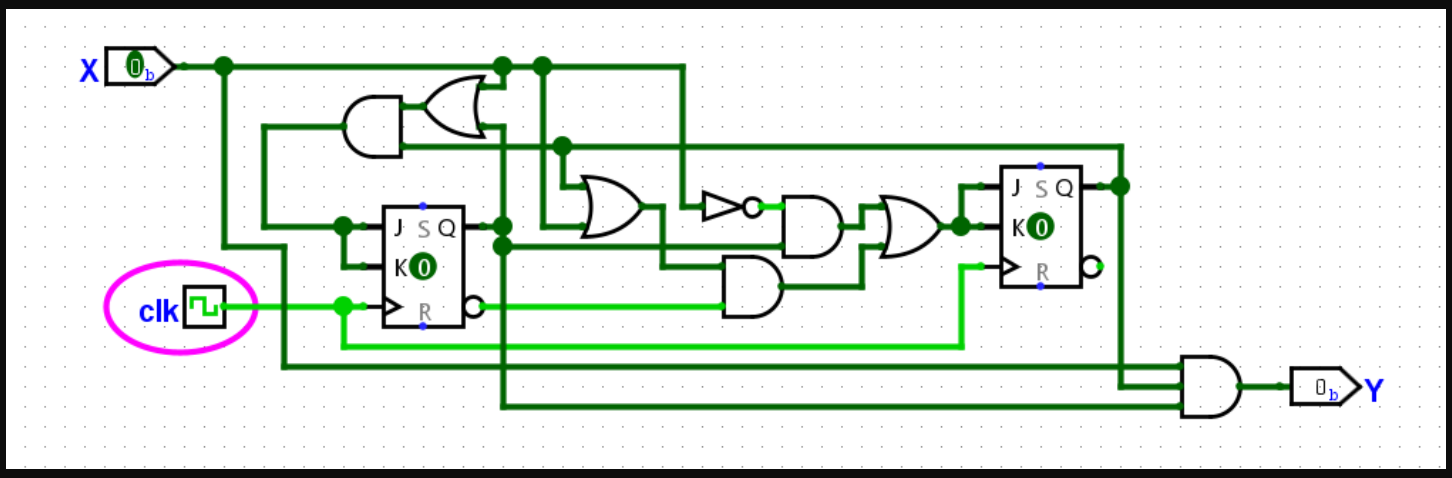
	$\overline{C}$	C
$\overline{A}.\overline{B}$	0	1
$\overline{A}.B$	1	1
A.B	1	0
$A.\overline{B}$	1	0

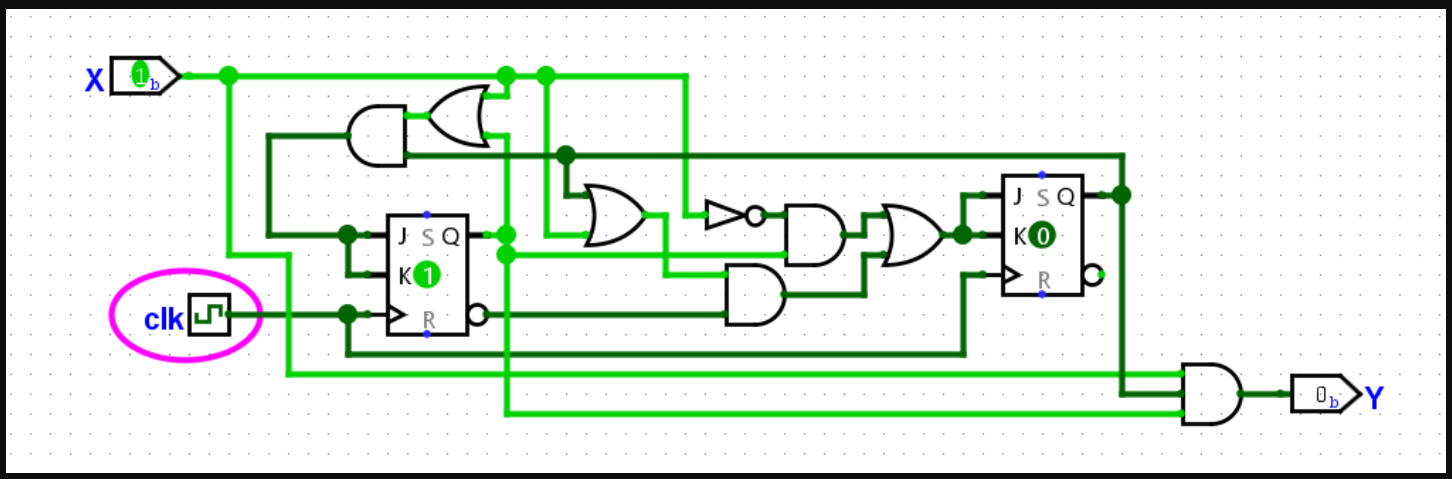
$T_{s0} = A'C + A'B + AC' = S_1'X + S_1'S_0 + S_1X'$

We get the following circuit in Logisim:

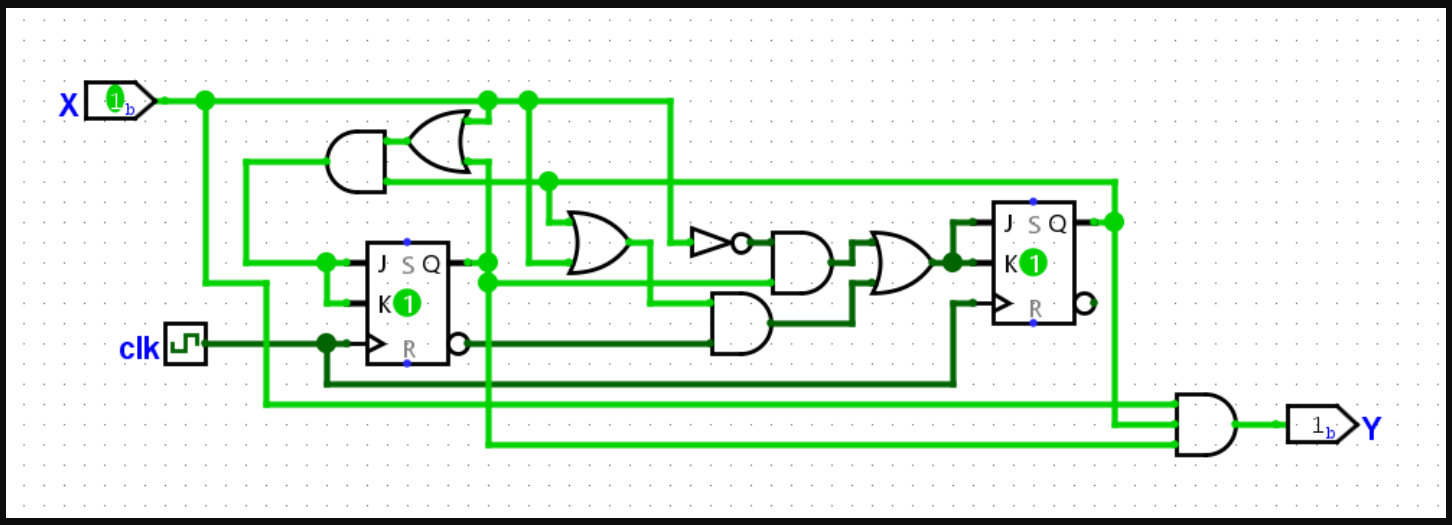


Some outputs:





After following the sequence, we get:



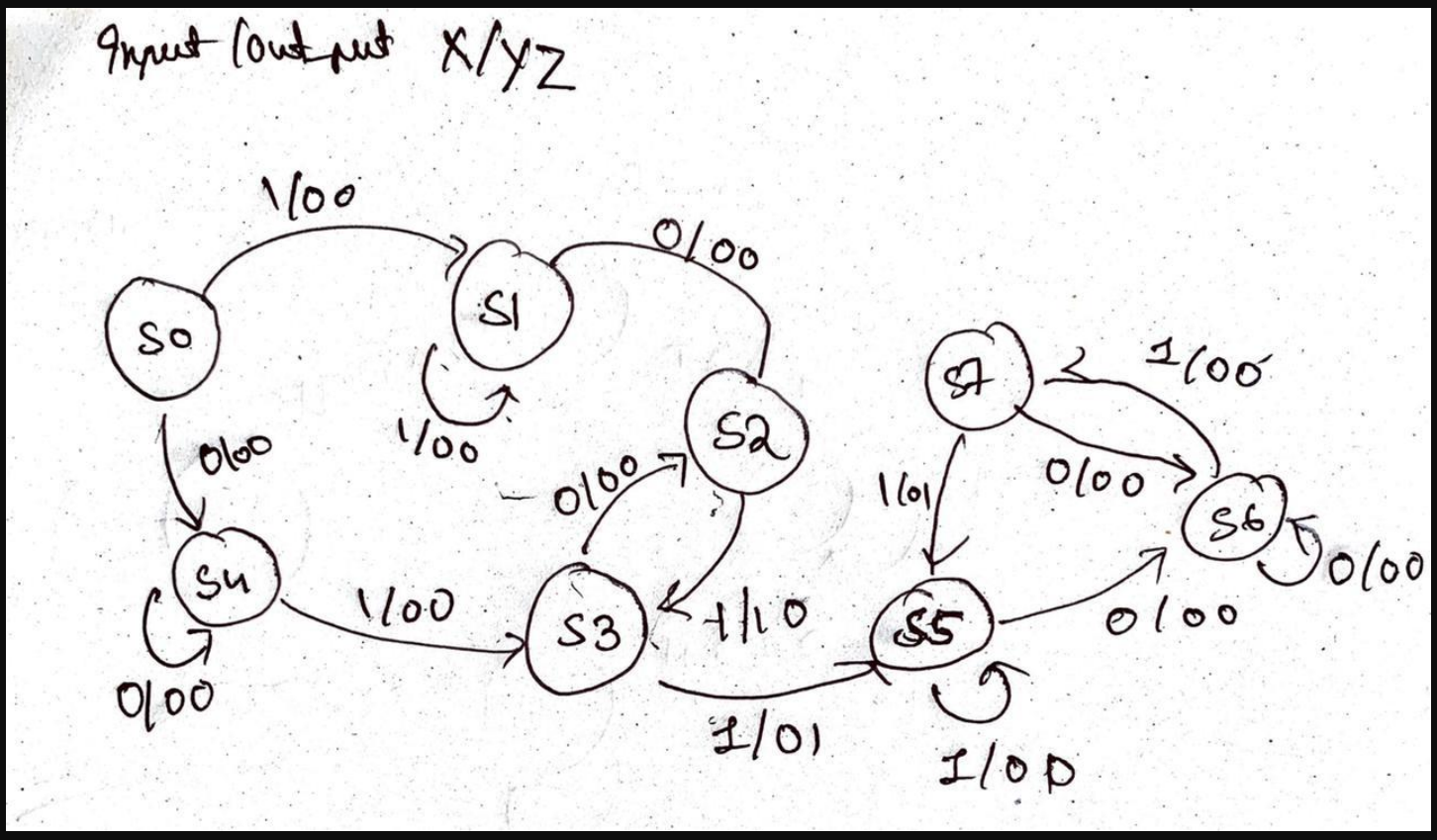
2] A digital system has one input X and two outputs Y and Z. The output Y=1 occurs every time the input sequence 101 is completed provided that the sequence 011 has never occurred. The output Z=1 occurs each time the output 011 is completed, i.e. once Z=1 has occurred, Y=1 can never occur but not vice versa. Design the Mealy state machine and implement it using Verilog.

->

Considering all the inputs and outputs as given in the question.

We can see that it's a mealy machine as the output at states also depends upon the inputs.

Building a mealy fsm based state diagram for the following, we get:



From the following we can denote States S0, S1 … S7 with binary values as given below in the Verilog code.

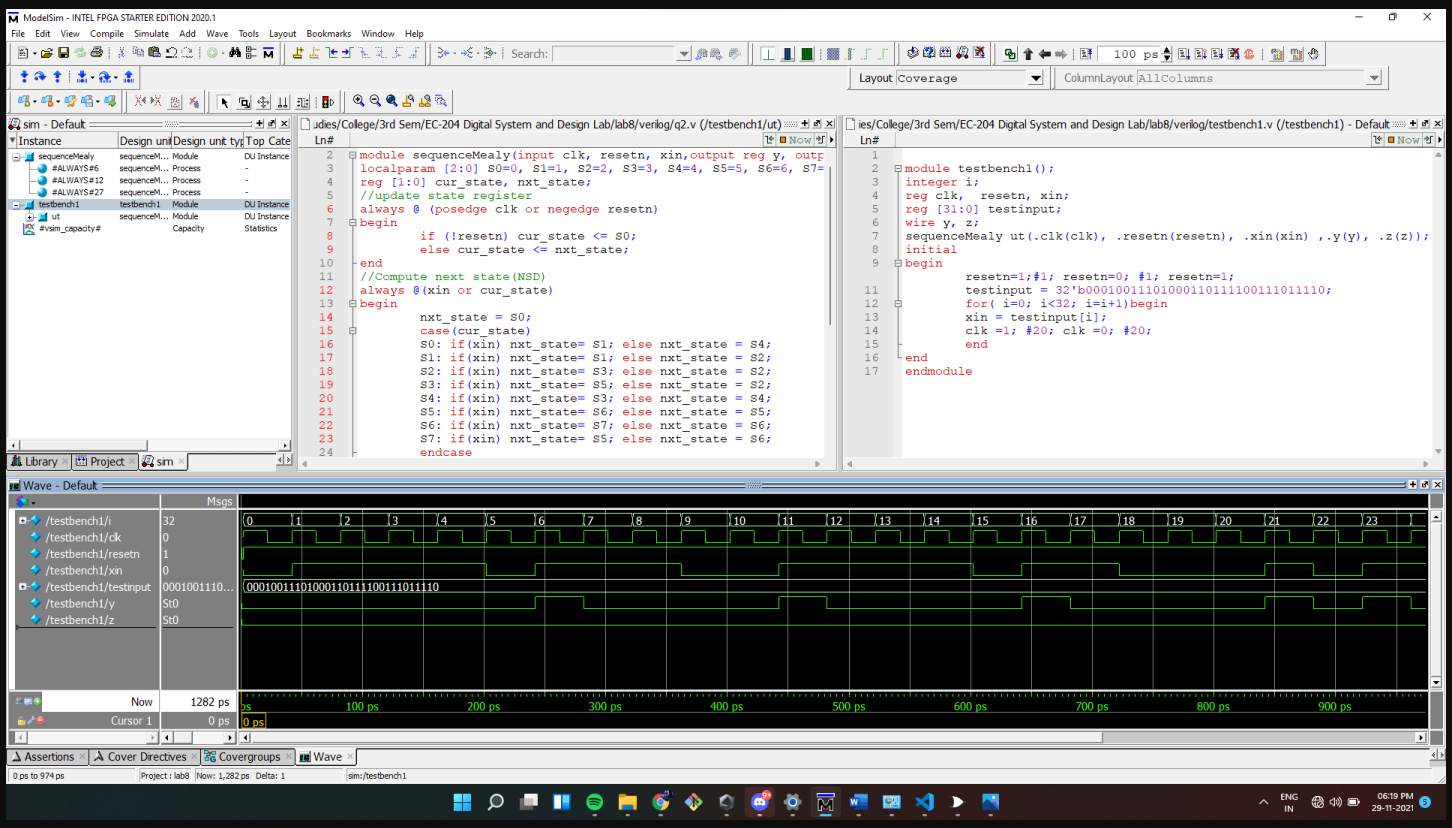
```
module sequenceMealy(input clk, resetn, xin,output reg y, output reg z);
localparam [2:0] S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6, S7=7;
reg [1:0] cur_state, nxt_state;
//update state register
always @ (posedge clk or negedge resetn)
begin
    if (!resetn) cur_state <= S0;
    else cur_state <= nxt_state;
end
//Compute next state(NSD)
always @(xin or cur_state)
begin
    nxt_state = S0;
    case(cur_state)
    S0: if(xin) nxt_state= S1; else nxt_state = S4;
    S1: if(xin) nxt_state= S1; else nxt_state = S2;
    S2: if(xin) nxt_state= S3; else nxt_state = S2;
    S3: if(xin) nxt_state= S5; else nxt_state = S2;
    S4: if(xin) nxt_state= S3; else nxt_state = S4;
    S5: if(xin) nxt_state= S6; else nxt_state = S5;
    S6: if(xin) nxt_state= S7; else nxt_state = S6;
    S7: if(xin) nxt_state= S5; else nxt_state = S6;
    endcase
end
//compute output (OD)
always @ (xin or cur_state)
begin
    y=1'b0; z=1'b0;
    case(cur_state)
    S0: begin y=1'b0; z=1'b0; end
    S1: begin y=1'b0; z=1'b0; end
    S2: begin if(xin) begin y=1'b1; z=1'b0; end else y=1'b0; z=1'b0; end
    S3: begin if(xin) begin y=1'b0; z=1'b1; end else y=1'b0; z=1'b0; end
    S4: begin y=1'b0; z=1'b0; end
    S5: begin y=1'b0; z=1'b0; end
    S6: begin y=1'b0; z=1'b0; end
    S7: begin if(xin) begin y=1'b0; z=1'b1; end else y=1'b0; z=1'b0; end
    endcase
end
endmodule
```

Verilog code for the testbench:

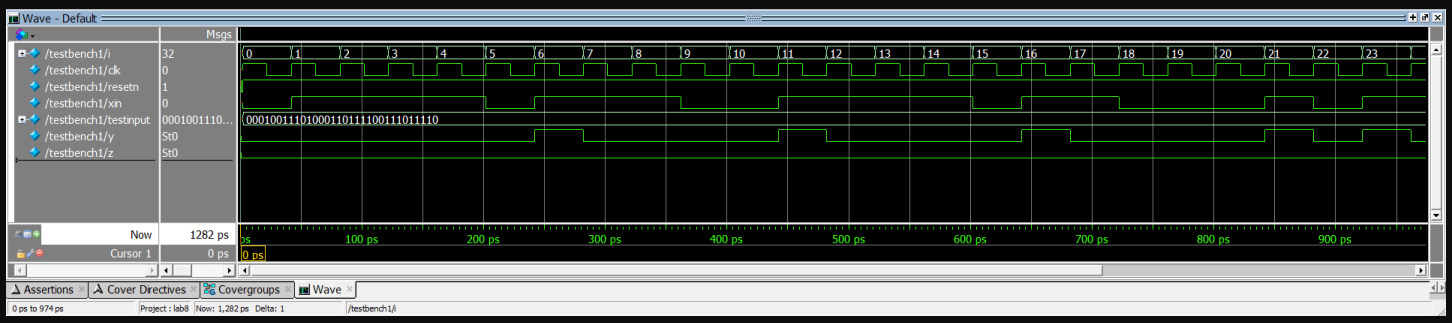
```
module testbench1();
integer i;
reg clk,  resetn, xin;
reg [31:0] testinput;
wire y, z;
sequenceMealy ut(.clk(clk), .resetn(resetn), .xin(xin) ,.y(y), .z(z));
```

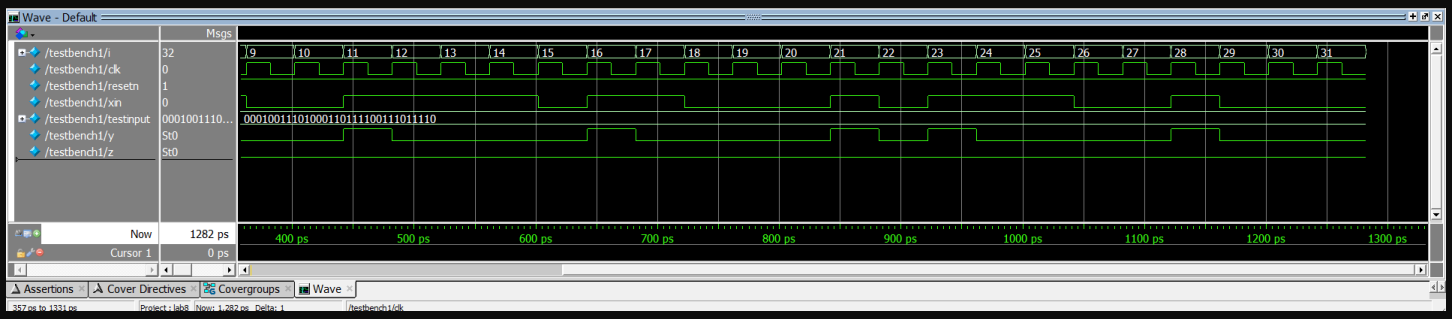
```
initial
begin
    resetn=1;#1; resetn=0; #1; resetn=1;

    testinput = 32'b00010011101000110111100111011110;
    for( i=0; i<32; i=i+1)begin
    xin = testinput[i];
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

Full Screenshot:



Output wave:





3] **Vending Machine** : A sequential network is to be used to control the operation of a vending machine which dispenses 200ml cup of water costing Rs 3. The network has 3 inputs FIVE, TWO, ONE and 2 outputs WATER and CHANGE. The coin detector mechanism in the vending machine is synchronized with the same clock as the sequential network you are to design. The coin detector inputs a signal 1 to the FIVE, TWO, ONE for every Five rupees, Two rupees or One rupee coin, respectively, that the customer inserts. Only one input will 1 at a time. When the customer has inserted the cost of the cup of water in any combination of Five, Two or Ones, the vending machine must give change and dispense the water. The coin return mechanism gives change by returning One Rupee coins to the customer. For every 1 output on CHANGE, the coin return mechanism will return One rupee to the customer. The water will be dispensed when the network outputs a signal 1 on output WATER. The network should reset after dispensing the product.

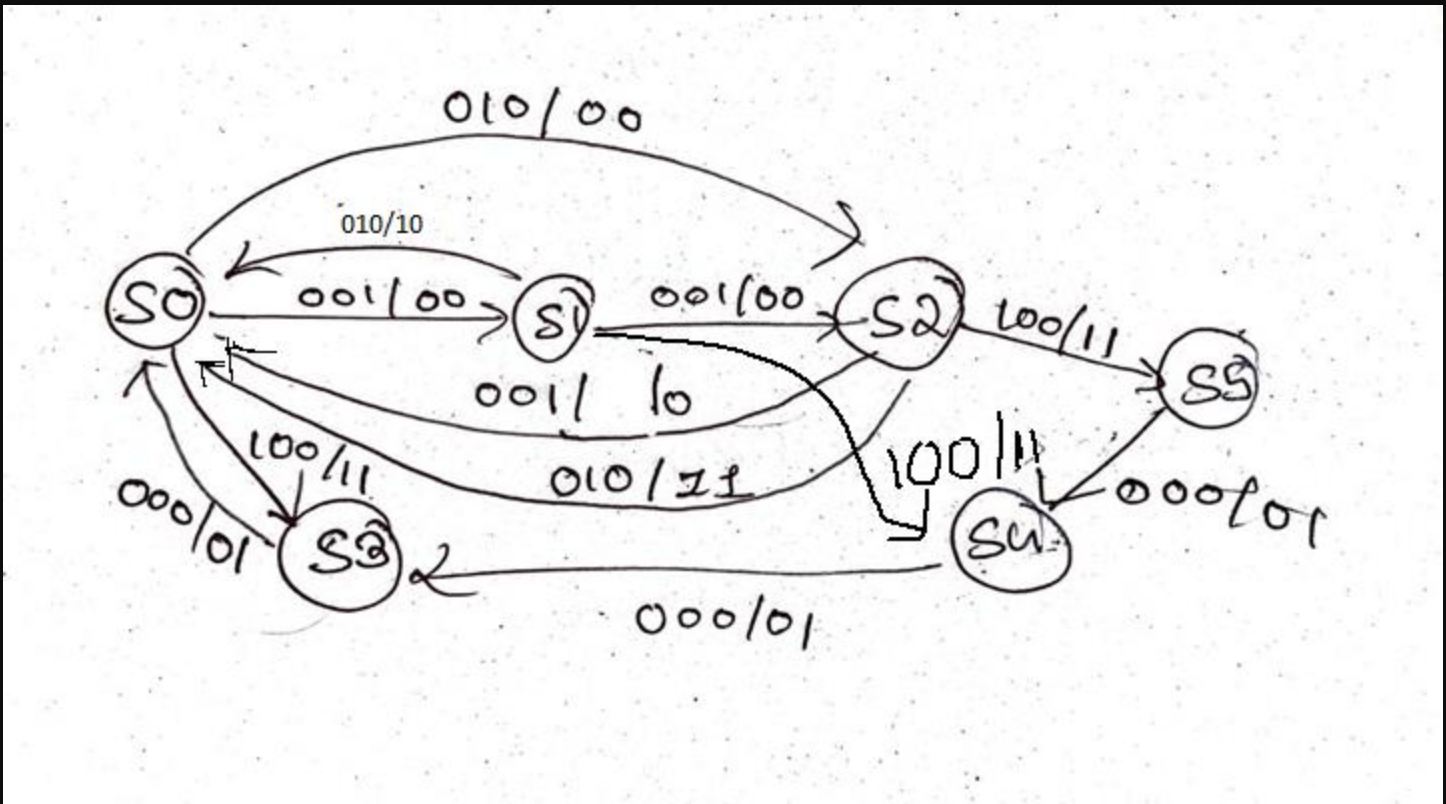
Draw the minimal state diagram for the system and model in Verilog.

->

For drawing the state diagram we will consider the input outputs on arrows denoted as (5rs)(2rs)(1rs)/(water)(change)

For example, for input 2rs and gives water and no change then transition on state will be 010/10

We can reduce the number of states by excluding the states where inputs other than 0 cannot be taken.



Verilog Code:

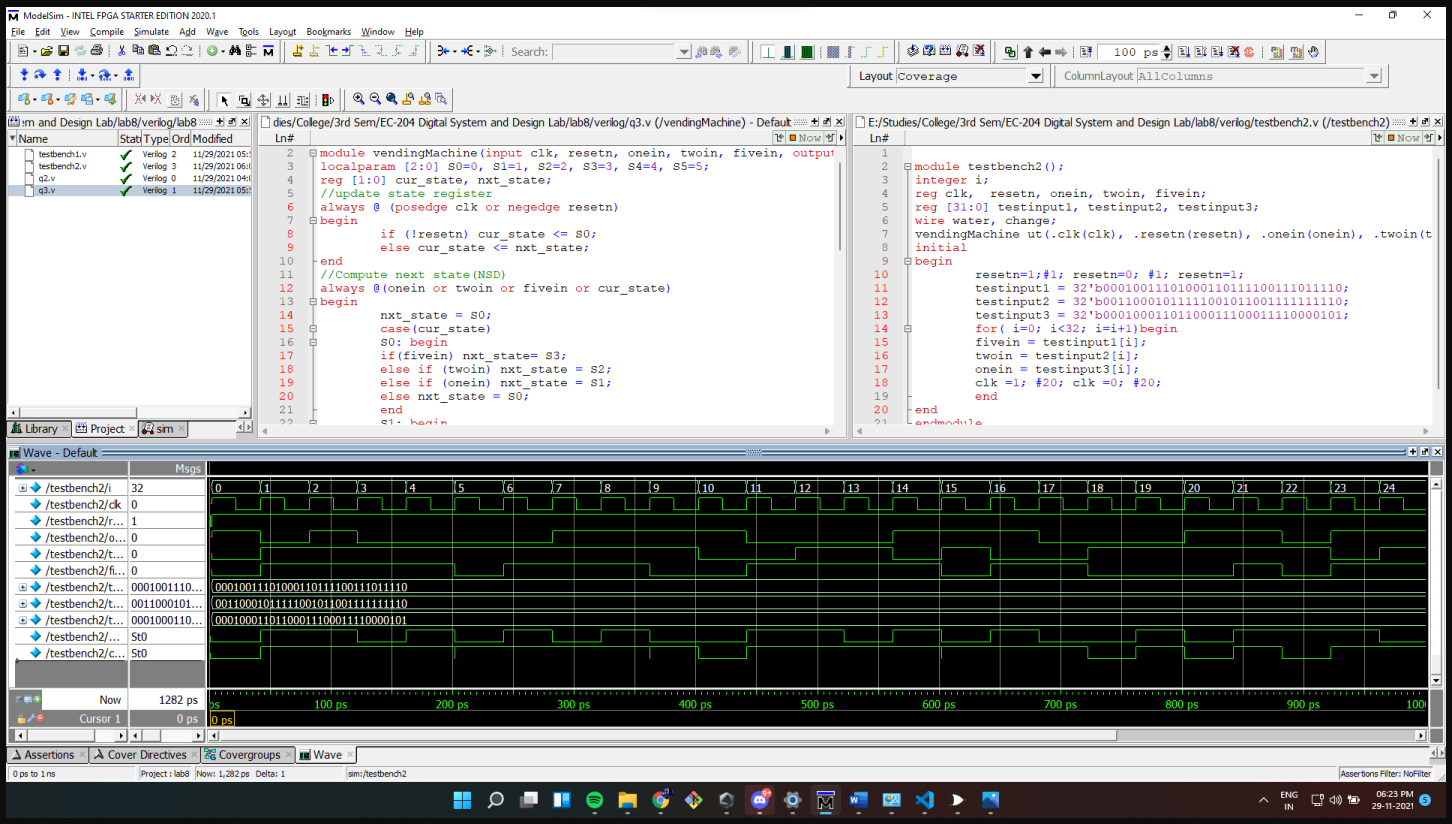
```
module vendingMachine(input clk, resetn, onein, twoin, fivein, output reg water, output reg
change);
localparam [2:0] S0=0, S1=1, S2=2, S3=3, S4=4, S5=5;
reg [1:0] cur_state, nxt_state;
//update state register
always @ (posedge clk or negedge resetn)
begin
    if (!resetn) cur_state <= S0;
    else cur_state <= nxt_state;
end
//Compute next state(NSD)
always @(onein or twoin or fivein or cur_state)
begin
    nxt_state = S0;
    case(cur_state)
    S0: begin
    if(fivein) nxt_state= S3;
    else if (twoin) nxt_state = S2;
    else if (onein) nxt_state = S1;
    else nxt_state = S0;
    end
    S1: begin
```

```
    if(fivein) nxt_state= S4;
    else if (twoin) nxt_state = S0;
    else if (onein) nxt_state = S2;
    else nxt_state = S1;
    end
    S2: begin
    if(fivein) nxt_state= S5;
    else if (twoin) nxt_state = S0;
    else if (onein) nxt_state = S0;
    else nxt_state = S1;
    end
    S3: nxt_state = S0;
    S4: nxt_state = S3;
    S5: nxt_state = S4;
    endcase
end
//compute output (OD)
always @ (onein or twoin or fivein or cur_state)
begin
    water=1'b0; change=1'b0;
    case(cur_state)
    S0: begin
    if(fivein) begin water=1'b1; change=1'b1; end
    else begin water=1'b0; change=1'b0; end
    end
    S1: begin
    if(fivein) begin water=1'b1; change=1'b1; end
    else if(twoin) begin water=1'b1; change=1'b0; end
    else begin water=1'b0; change=1'b0; end
    end
    S2: begin
    if(fivein) begin water=1'b1; change=1'b1; end
    else if(twoin) begin water=1'b1; change=1'b1; end
    else if(onein) begin water=1'b1; change=1'b0; end
    else begin water=1'b0; change=1'b0; end
    end
    S3: begin water=1'b0; change=1'b1; end
    S4: begin water=1'b0; change=1'b1; end
    S5: begin water=1'b0; change=1'b1; end
    endcase
end
endmodule
```

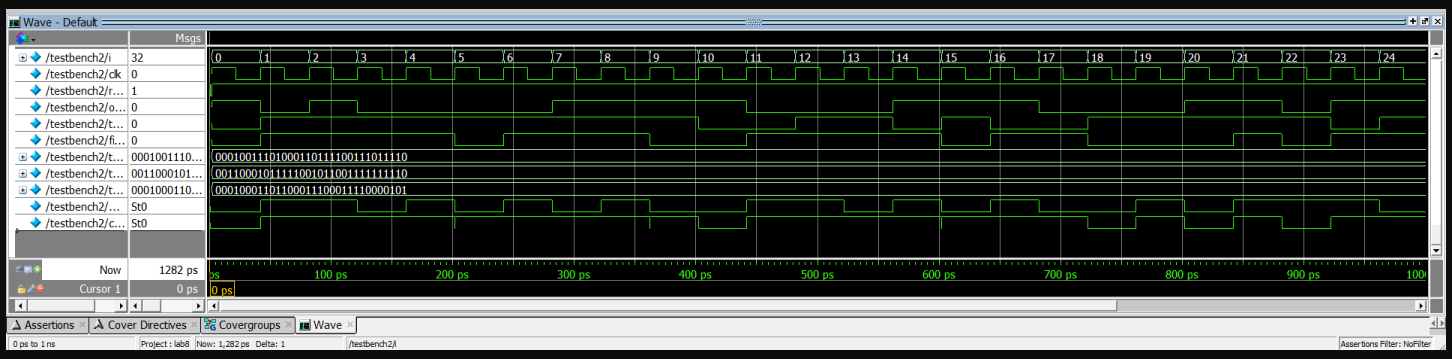
Verilog code for the testbench:

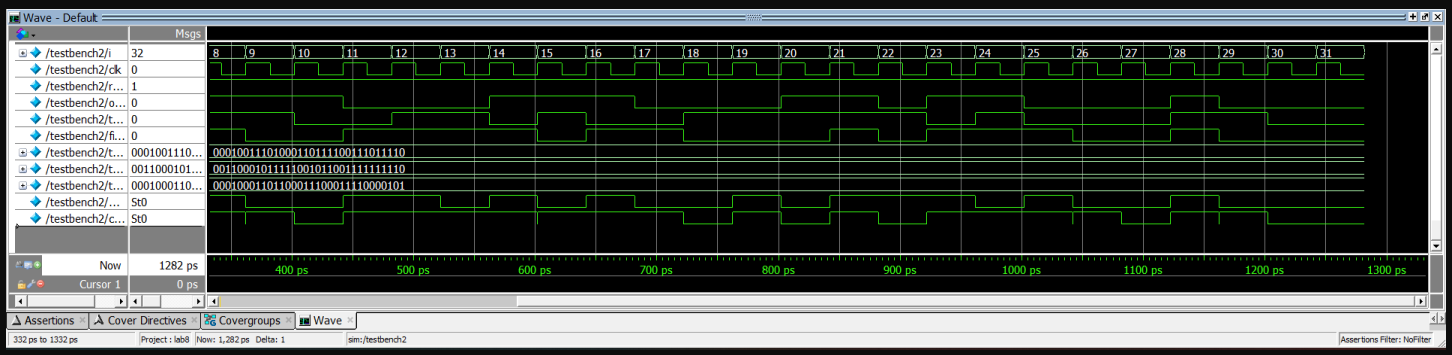
```
module testbench2();
integer i;
reg clk,  resetn, onein, twoin, fivein;
reg [31:0] testinput1, testinput2, testinput3;
wire water, change;
vendingMachine ut(.clk(clk), .resetn(resetn), .onein(onein), .twoin(twoin), .fivein(fivein),
.water(water), .change(change));
initial
begin
    resetn=1;#1; resetn=0; #1; resetn=1;

    testinput1 = 32'b00010011101000110111100111011110;
    testinput2 = 32'b00110001011111001011001111111110;
    testinput3 = 32'b00010001101100011100011110000101;
    for( i=0; i<32; i=i+1)begin
    fivein = testinput1[i];
    twoin = testinput2[i];
    onein = testinput3[i];
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

Full screenshot:



Output wave:





4] Traffic light controller for NITK Design a traffic light controller for NITK, implement using DFF and simulate using Logisim. The National highway NH66 is intersected by the NITK road as shown in Figure. Detectors are placed along the NITK road to raise the signal C as long as a vehicle is waiting to cross the highway. The traffic light controller should operate as follows: as long as no vehicle is detected on the NITK road the lights should remain green in the highway direction. If a vehicle is detected on the NITK road, the highway lights should change from green to yellow to red, allowing the NITK road lights to become green. The NITK road lights stay green only as long as a vehicle is detected on the NITK road and never longer than a set interval to allow the traffic to flow on the highway. If these conditions are met the NITK road lights change from green to yellow to red, allowing the highway lights to return to green. Even if vehicles are waiting to cross the highway, the highway lights should remain green for a set interval. You may assume that there is an external timer that, once set via the control signal ST (set timer) will assert the signal TS after a short time interval has expired (used for timing yellow lights), TLH and TLF after a longer time interval (for green

lights, highway and NITK road respectively). The timer is automatically reset when ST is asserted.

-> We will consider 4 following states with inputs considering it as Mealy machine as it output depends on the input C (NITK road traffic)

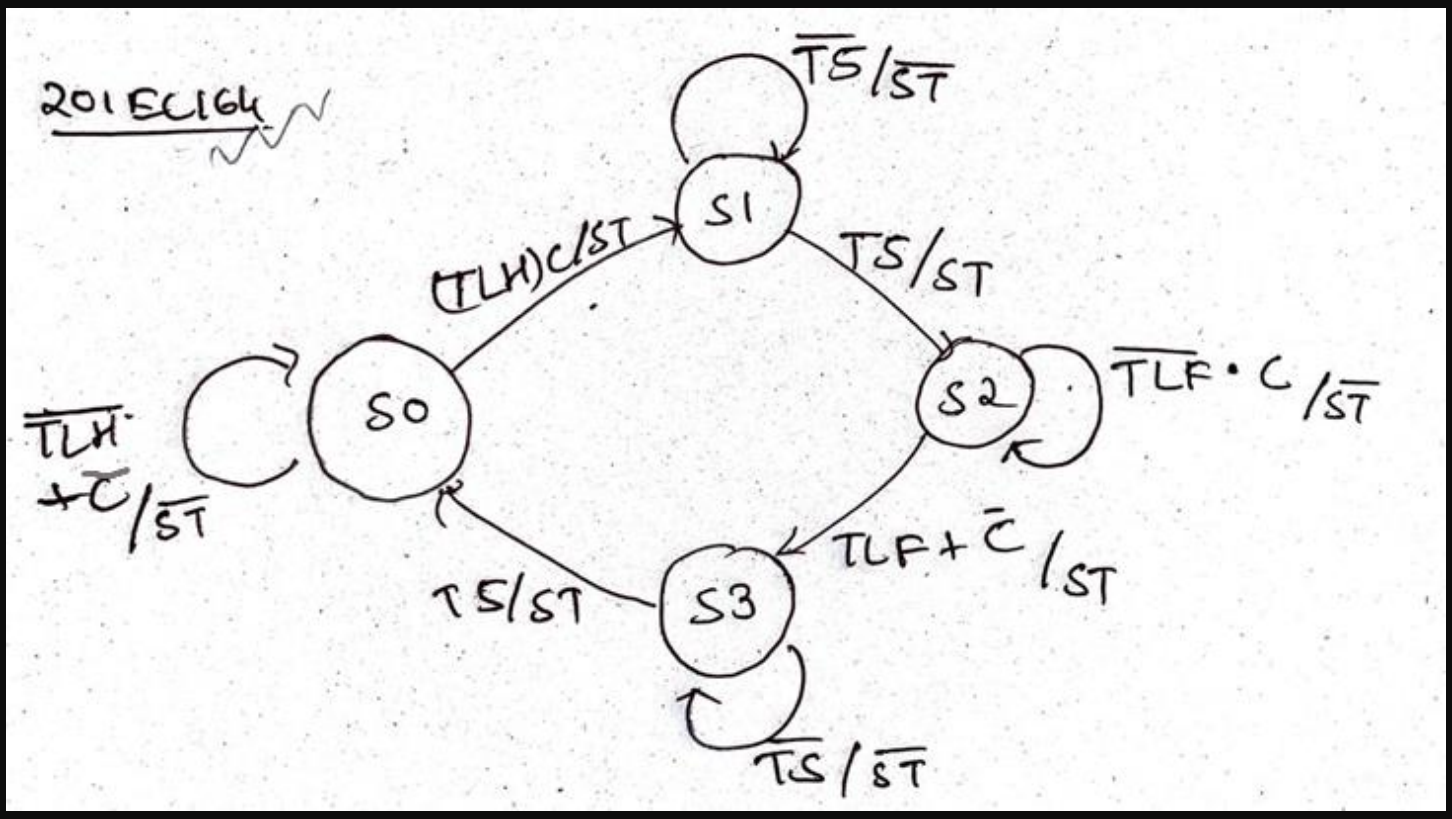
S0: S=00: Highway Light Green && NITK road Red

S1: S=01: Highway Light Yellow && NITK road Red

S2: S=10: Highway Light Red && NITK road Green

S3: S=11: Highway Light Red && NITK road Yellow

Taking, inputs as given, we can build the following state diagram for the required FSM.



We get the following State Table,

S_1	S_0	S_1'	S_0'	C	TS	TLH	TLF	ST
0	0	0	0	0	d	0	d	0
0	0	0	0	1	d	0	d	0
0	0	0	0	0	d	1	d	0
0	0	0	1	1	d	1	d	1
0	1	0	1	d	0	d	d	0
0	1	1	0	d	1	d	d	1
1	0	1	0	1	0	d	0	0
1	0	1	1	0	d	d	1	1
1	0	1	1	1	d	d	1	1
1	0	1	1	0	d	d	0	1
1	1	1	1	d	0	d	d	0
1	1	0	0	d	1	d	d	1

For 4 states we will need 2 D flip flops to switch between states. We will name them as D1 and D0 for state inputs S1 and S0

Considering D flip flop table,

C	D	Q(n+1)
0	X	Q(n)
1	0	0
1	1	1

We can get the following combinational logic for D inputs of both the flip flops

$D_0 = S_0' = (S1'S0'TIhC) + (S1'S0TS') + (S1S0'(TIf+C')) + (S1S0TS')$

$D_1 = S_1' = (S1'S0TS) + (S1S0'TIf'C) + (S1S0'(TIf+C')) + (S1S0TS')$

For the timer we will build 2 count down counters using D flip flops and also a button to set the timer for it. So that when the counters counts to 0 it will turn on the respective signals TLH, TLF or TS.

Q2	Q1	Q0	Q2’	Q1’	Q0’
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

We get the following combination,

Map

	$\overline{C}$	C
$\overline{A}.\overline{B}$	1	0
$\overline{A}.B$	1	0
A.B	1	0
A.$\overline{B}$	1	0

D0 = Q0’ = (Q0)’

Map

	$\overline{C}$	C
$\overline{A}.\overline{B}$	0	1
$\overline{A}.B$	1	0
A.B	1	0
A.$\overline{B}$	0	1

D1 = Q1’ = (Q1)'Q0 + Q1(Q0)'

Map

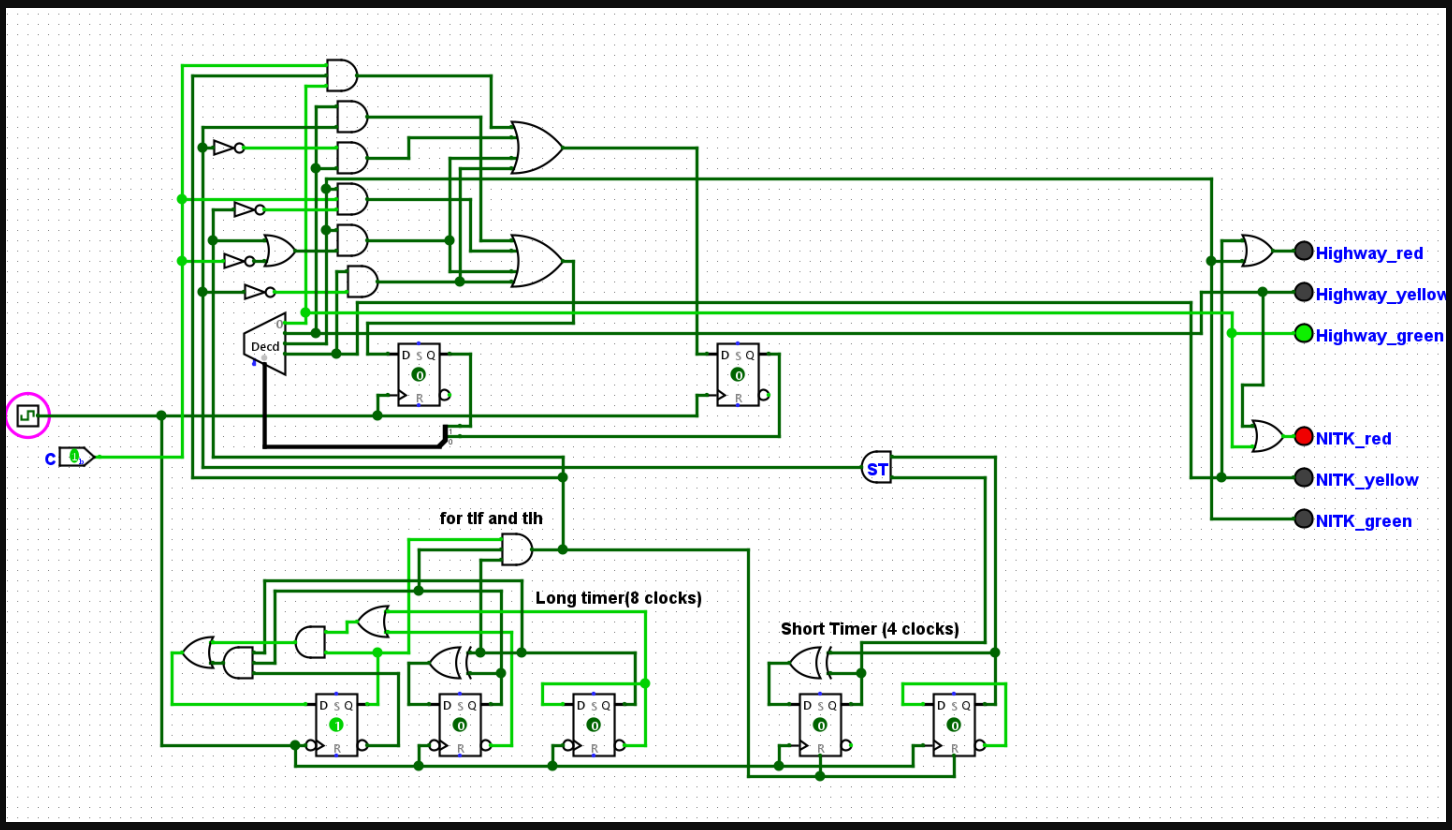
	$\overline{C}$	C
$\overline{A}\overline{B}$	0	0
$\overline{A}B$	0	1
$A\overline{B}$	1	0
AB	1	1

D2 = Q2’ = Q2(Q1)' + Q2(Q0)' + (Q2)'Q1Q0

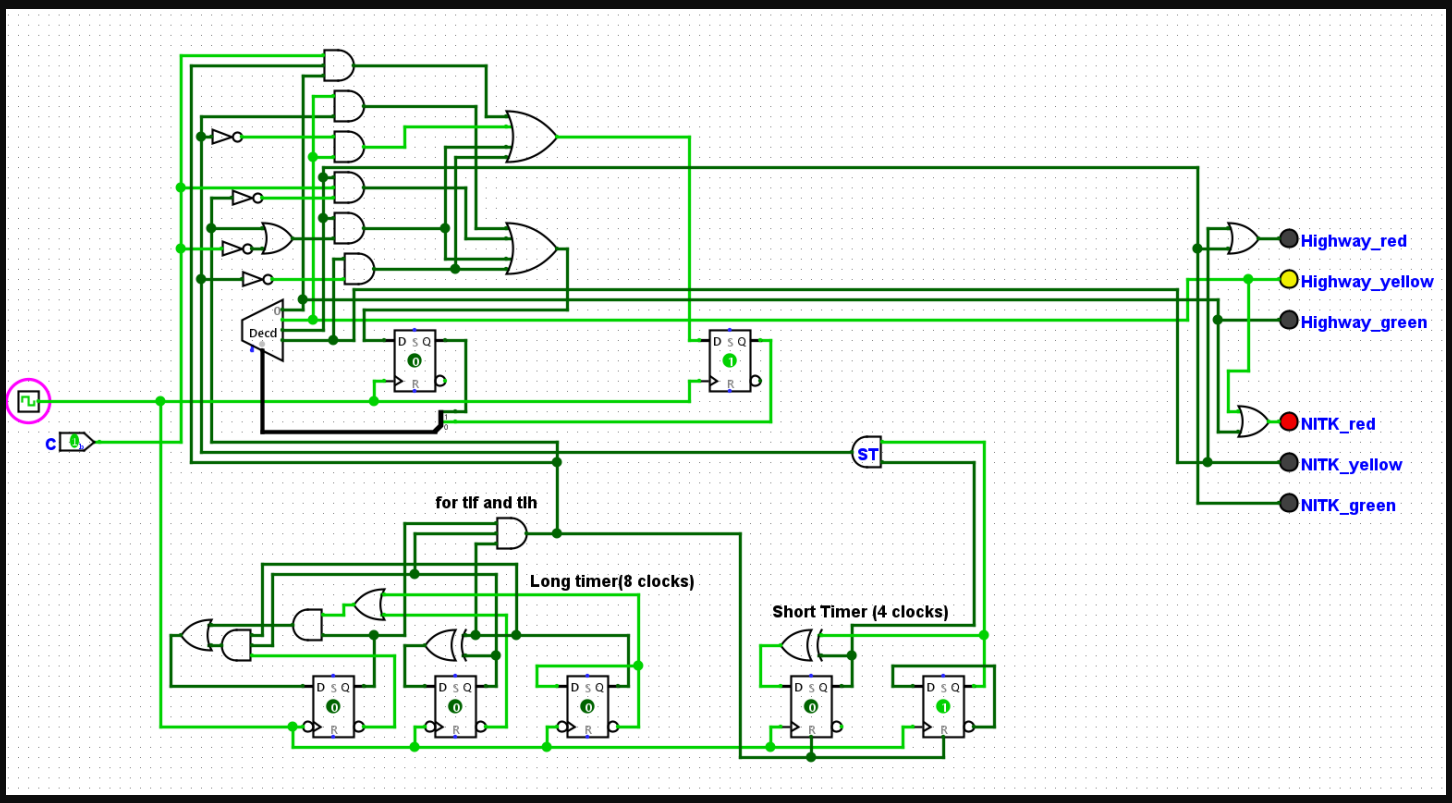
Building counter with following logic we can build the timers required, we will use a single timer for Tlf and Tlh but a smaller one for TS.

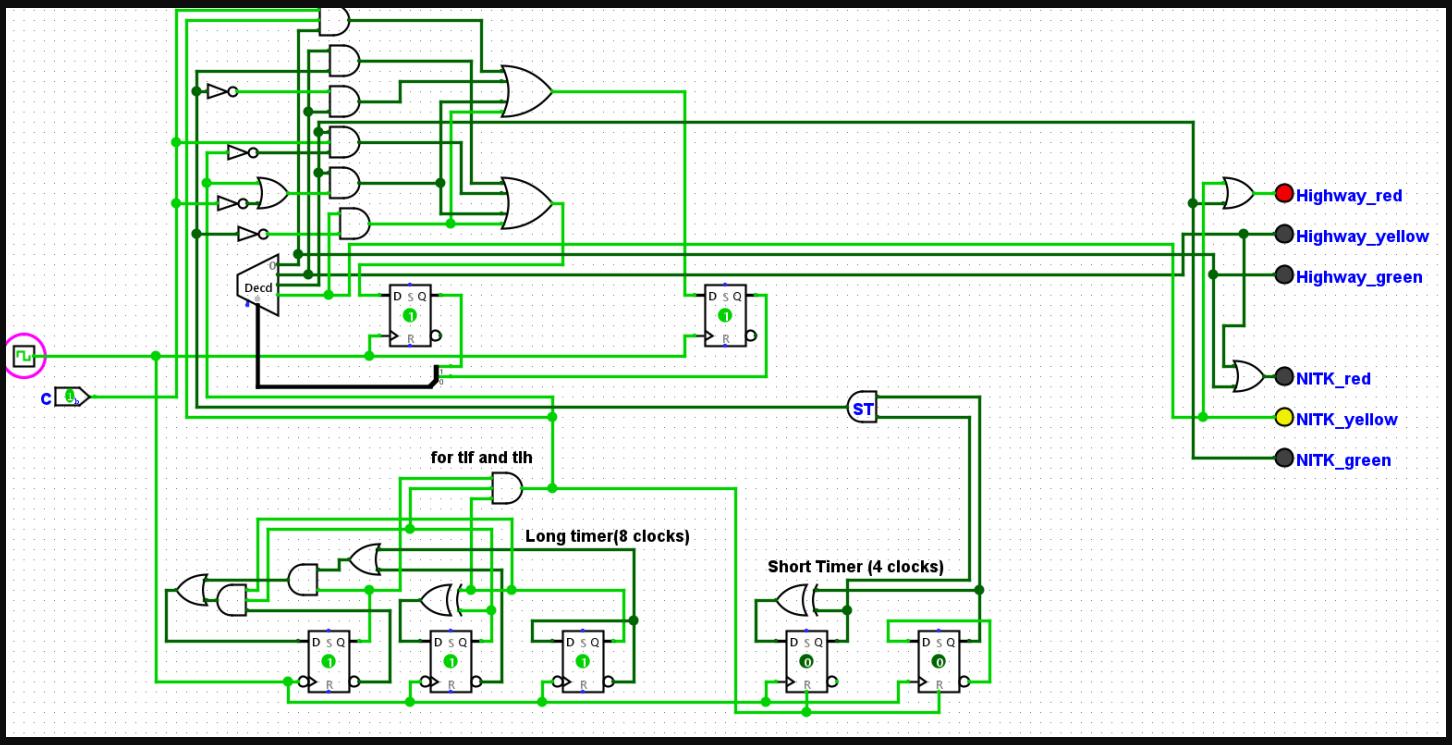
Instead of connecting ST output to reset timer counters, we will include to auto reset in the timer fsm itself.

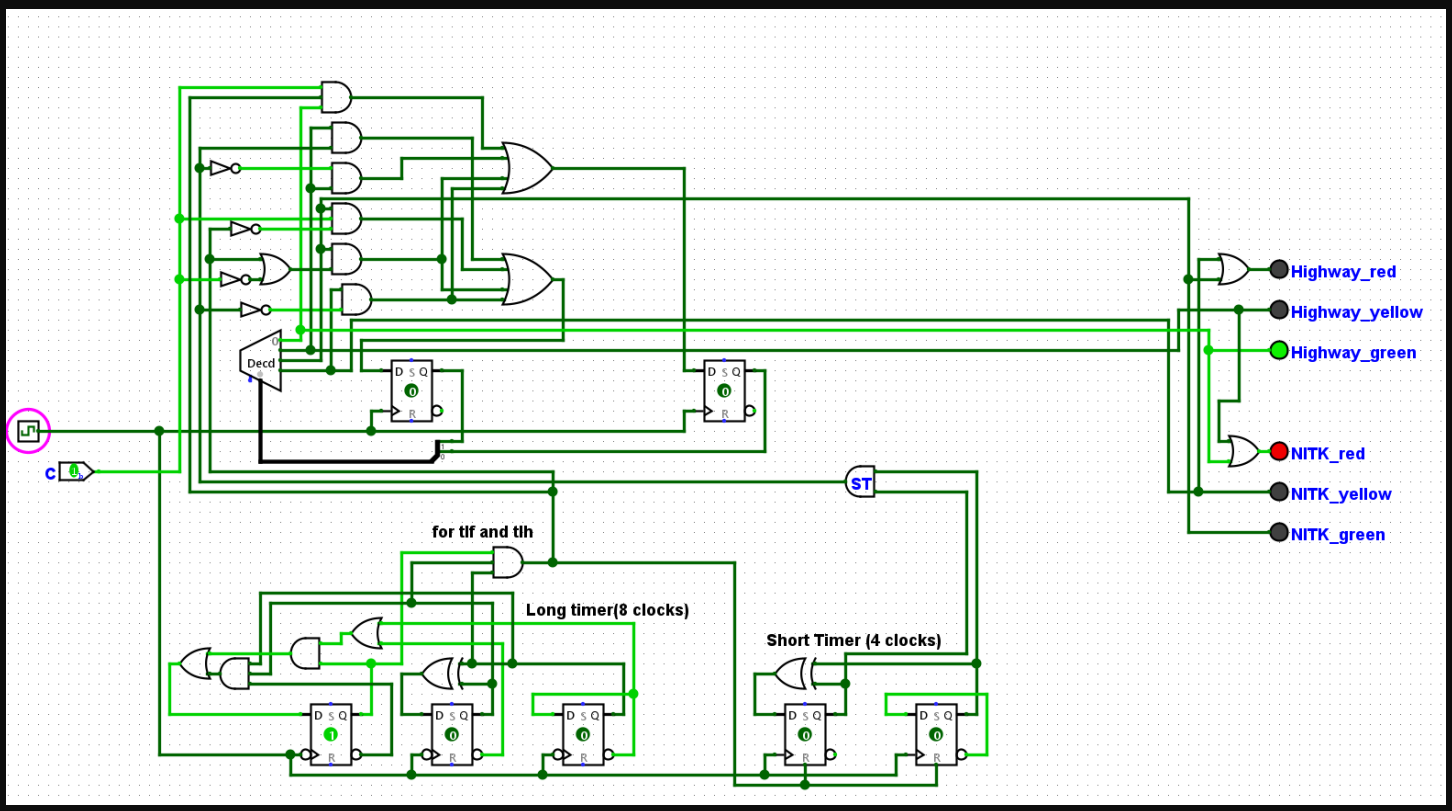
We get the following circuit:



Some outputs:






Highway_red
Highway_yellow
Highway_green
NITK_red
NITK_yellow
NITK_green
Decd
C
ST
for tlf and tlh
Long timer(8 clocks)
Short Timer (4 clocks)