Experiment - 6: Mealy and Moore FSM Simulation Using Quartus Prime

NAME: Jayakrishnan Menon

REG NO: 22BEC1205

DATE: 06/01/2025

Aim:
Write a Verilog RTL code for sequence detection using Mealy and Moore Finite State Machines. Also, perform the simulation of the Full Adder using Quartus Prime and Model Sim. Finally, implement these circuits on the FPGA kit, "5CSXFC6D6F31C6N"

Software Required: Quartus Prime**,** ModelSim

Haraware Required: Altera Cyclone V 5CSXFC6D6F31C6N

Procedure:

- Open Quartus Prime 21.1.
- Go to File -> New Project wizard -> select Source folder & type file name -> Next.
- Select Empty Project in Project type -> Next.
- Click on Next in Add Files dialog box.
- Select "Cyclone V SX Extended Features" in Device Family.
- In available devices, select the one ending with "31C6" -> Next.
- Select the tool name as "ModelSim" and Format as "VerilogHDL" in simulation.
- Click on Finish. The project is now created.
- In the Task window, select RTL simulation and run, this would open the ModelSim window.
- Simulate the full adder as you would do using ModelSim by forcing the input values.
- In Compilation -> select compile design.
- Go to Assignments tab -> Pin planner -> Give the location for each i/o pin.
- Go to Hardware Setup -> Select USB.
- Change the file to Fulladder.v in the program/configure option and select Start.

Theory

Finite State Machines:

The input combinational circuit in synchronous sequential circuits is made up of a series of logic gates, with flip flops serving as memory elements. The synchronous sequence machine is described by the finite state machine (FSM), which is an abstract model. In a sequential circuit, the output is determined by the current input as well as previous history, necessitating an endless storage capacity.

Finite state machines are utilized because machines with infinite storage capacity are impossible to implement. Finite state machines are sequential circuits with a finite number of ways in which their past history might affect their future behaviour.

Machines with a finite number of states are known as finite state machines. There are a limited number of memory devices in any finite-state system. We can construct a periodic sequence of fewer than or equal to n-states using an n-state machine.

There are two types of finite state machines (FSM) and the major difference between the two is in the generation of output. These types are:

1. Mealy Machine
2. Moore Machine

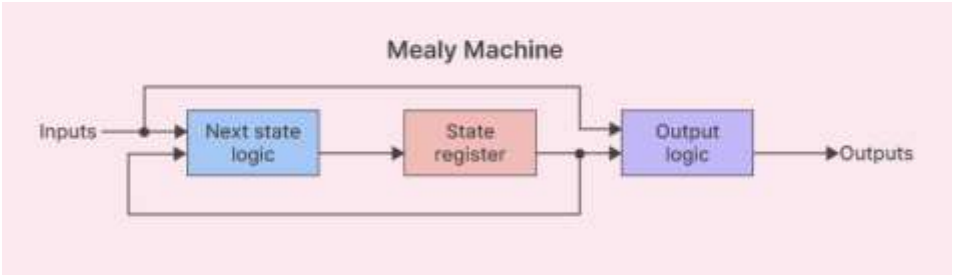
Mealy FSM:

A Mealy machine is a machine in the theory of computation in which the output symbol is linked to the transition (state and input) of a finite state machine. It creates an equivalent output string for each input letter/string.

If the length of the input string is n, the length of the Moore machine's output string is also n. The value of the output function becomes a function of transitions and changes when the input logic is completed in the current state.

On the current clock, the asynchronous output generation changes its state to synchronous. The Mealy circuit's output values are influenced by both the current state output of memory elements and external inputs.

In this type of circuit, Mealy machines respond to inputs more quickly. They all seem to react in the same clock cycle. It takes care of all string transitions across a specified alphabet. A Mealy machine is not a counter.



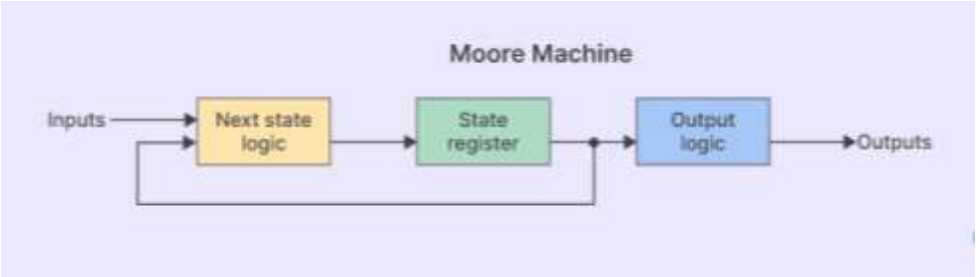
Moore FSM:

When each state of a finite state machine has an output symbol, the automata is known as a Moore machine. It generates an equivalent string for each input string.

It includes all string transitions for the given alphabet. The current state and current input symbol determine the next state. After the entire input sequence has been applied, the output is evaluated.

The output and state of the device change is synchronous with the clock edge. Moore machine generates the output associated with the initial state by reading no input symbol.

The output of a Moore circuit is solely determined by the current state of the memory elements. If the length of the input string is n, the length of the string produced by this machine will be (n+1).



Distinctive Features of Mealy and Moore FSM:

Mealy Machine	Moore Machine
1. The output of the circuit may be affected by changes in the information.	1. The output of the circuit is unaffected by changes in the input.
2. It requires fewer number of states for implementing the same function.	2. For implementing the same function, it requires more number of states
3. The output is a function of the current state as well as the input.	3. The output is a function of the current state only.
4. The counter cannot be referred to as a Mealy Machine.	4. The counter is referred to as a Moore Machine.
5. The design of the Mealy model is complex.	5. The design of the Moore model is easy.
6. Mealy machines react to changes more quickly. They all seem to react in the same way.	6. Decoding the output necessitates more logic, resulting in longer circuit delays. They usually react after one clock cycle
7. If the external inputs vary, the output can alter between clock edges.	7. Only when the active clock edge will the output alter.
8. The output is set on the transition.	8. The output is set to state.
9. It's easier to design with less hardware.	9. To design, more hardware is necessary

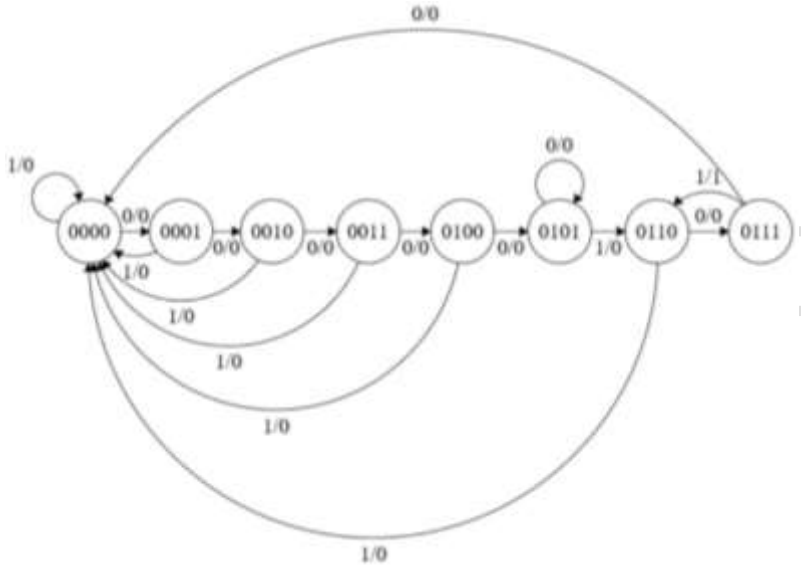
Lab Task:

To create an overlapping (if possible) sequence detecting Mealy and Moore FSM considering the BCD equivalent of the last 2 digits of your registration number.

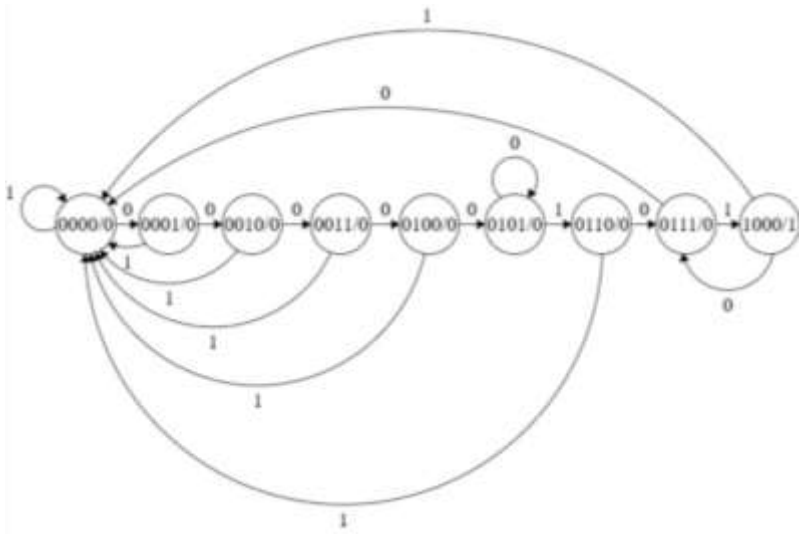
For 1205, last 2 digits are 05 and their BCD equivalent is 00000101.

For this sequence of 8 bits, an overlapping detector is impossible, because the sequence itself lacks the properties of symmetry and repetition. However, by detecting the first 4 bits, only once we can create FSMs which will detect the repetition of the last 2 digits, and hence only the lower order nibble (0101).

Mealy FSM for 00000101 detection, with overlap detection for the last 2 bits:



Moore FSM for 00000101 detection, with overlap detection for the last 2 bits:

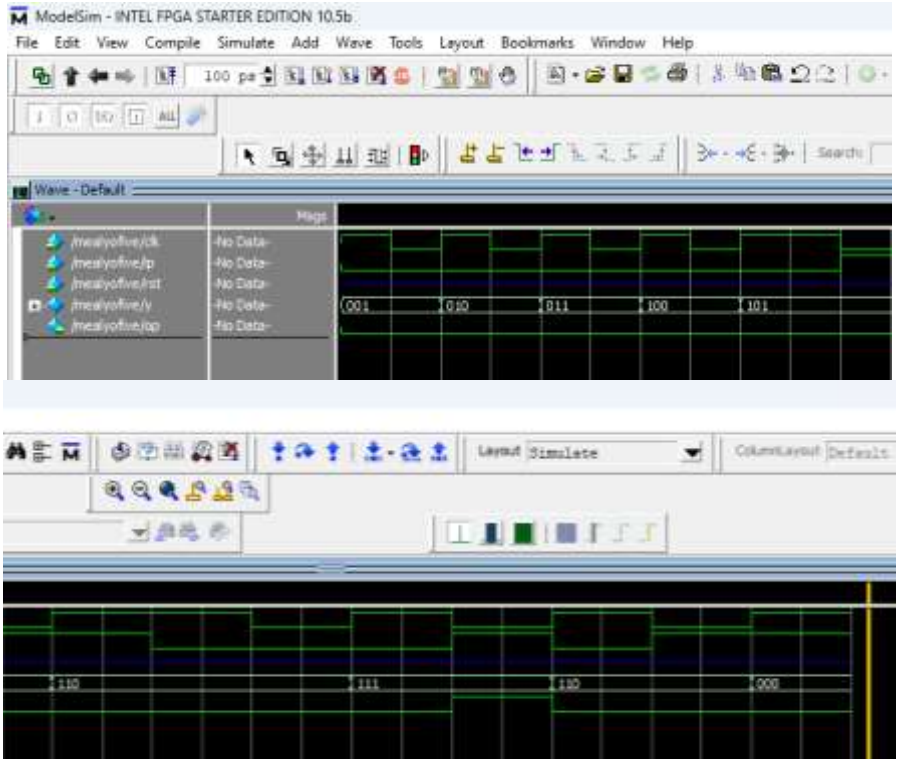


Source code and Outputs:

Mealy Sequence Detector:

```
module mealyofive(input clk, rst, ip, output reg op);
reg [2:0]y,Y,A,B,C,D,E,F,G,H;

initial begin
A=4'b000; B=4'b001; C=4'b010; D=4'b011;
E=4'b100; F=4'b101; G=4'b110; H=4'b111;
y=A; Y=3'bxxx; op=1'b0;
//y=currentstate Y=nextstate op=currentoutput  HEX1=op HEX0=st
end

always@(y,ip)begin //COMBINATIONAL LOGIC
case(y)
A: if(ip==0) begin Y=B; op=1'b0; end
else    begin Y=A; op=1'b0; end
B: if(ip==0) begin Y=C; op=1'b0; end
else    begin Y=A; op=1'b0; end
C: if(ip==0) begin Y=D; op=1'b0; end
else    begin Y=A; op=1'b0; end
D: if(ip==0) begin Y=E; op=1'b0; end
else    begin Y=A; op=1'b0; end
E: if(ip==0) begin Y=F; op=1'b0; end
else    begin Y=A; op=1'b0; end
F: if(ip==1) begin Y=G; op=1'b0; end
else    begin Y=F; op=1'b0; end
G: if(ip==0) begin Y=H; op=1'b0; end
else    begin Y=A; op=1'b0; end
H: if(ip==1) begin Y=G; op=1'b1; end
else    begin Y=A; op=1'b0; end
default:    begin Y=3'bxxx;  op=1'bx; end
endcase
end

always@(posedge clk)begin //SEQUENTIAL PART
if(rst==1) begin y<=A;          end
else     begin y<=Y;          end

end
endmodule
```

Output:

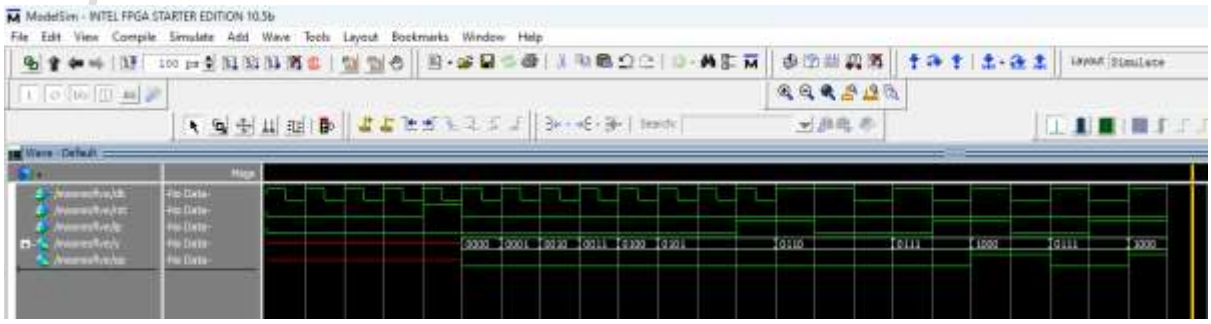


2. Moore Sequence Detector:

```
module mooreofive(input clk, rst, ip, output reg [3:0]y, output reg op);
reg [3:0]Y,A,B,C,D,E,F,G,H,I;
reg ph;

initial begin
A=4'b0000; B=4'b0001; C=4'b0010; D=4'b0011;
E=4'b0100; F=4'b0101; G=4'b0110; H=4'b0111;
I=4'b1000;
y=A; Y=4'bxxxx; op=1'bx;
//y=currentstate Y=nextstate op=currentoutput ph=placeholder HEX1=op HEX0=st
end

always@(y,ip)begin //COMBINATIONAL LOGIC
case(y)
A: if(ip==0) begin Y=B; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
B: if(ip==0) begin Y=C; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
C: if(ip==0) begin Y=D; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
D: if(ip==0) begin Y=E; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
```

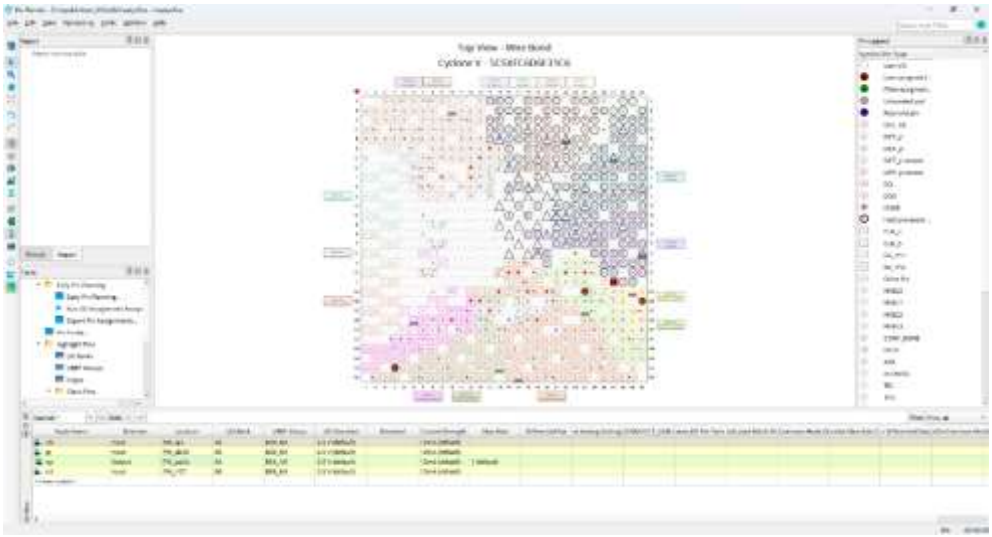
```
E: if(ip==0) begin Y=F; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
F: if(ip==1) begin Y=G; ph=1'b0; end
else    begin Y=F; ph=1'b0; end
G: if(ip==0) begin Y=H; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
H: if(ip==1) begin Y=I; ph=1'b1; end
else    begin Y=A; ph=1'b0; end
I: if(ip==0) begin Y=H; ph=1'b0; end
else    begin Y=A; ph=1'b0; end
default:    begin Y=4'bxxxx;  ph=1'bx; end
endcase
end

always@(posedge clk)begin //SEQUENTIAL PART
if(rst==1) begin y<=A; op=1'b0; end
else     begin y<=Y; op<=ph;  end

end

endmodule
```


Output:



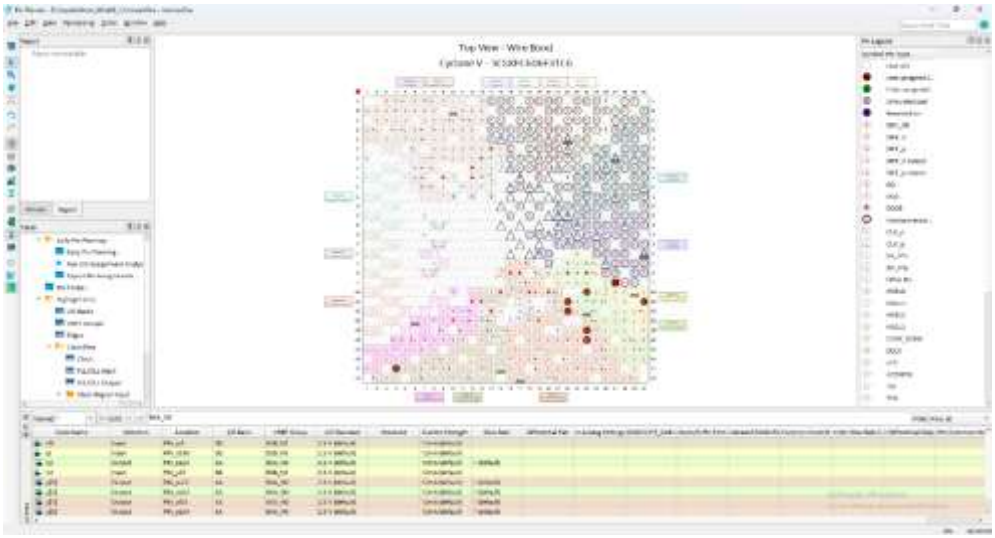
Pin Planning:

Mealy:



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate
clk	Input	PIN_aj4	3B	B3B_N0	2.5 V (default)		12mA (default)	
ip	Input	PIN_ab30	5B	B5B_N0	2.5 V (default)		12mA (default)	
op	Output	PIN_aa24	5A	B5A_N0	2.5 V (default)		12mA (default)	1 (default)
rst	Input	PIN_Y27	5B	B5B_N0	2.5 V (default)		12mA (default)	
<<new node>>								

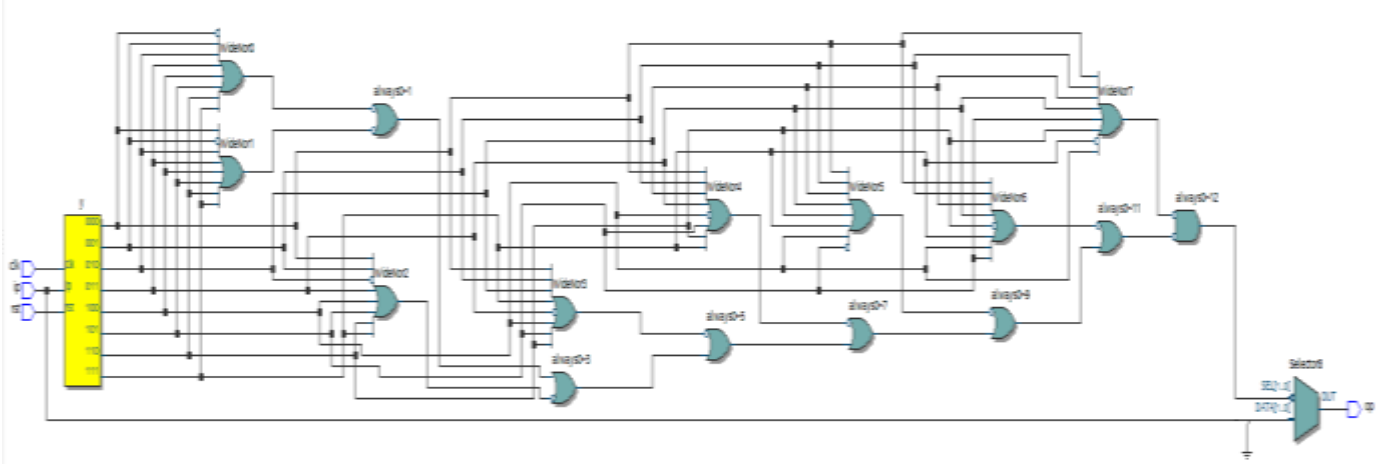
Moore:



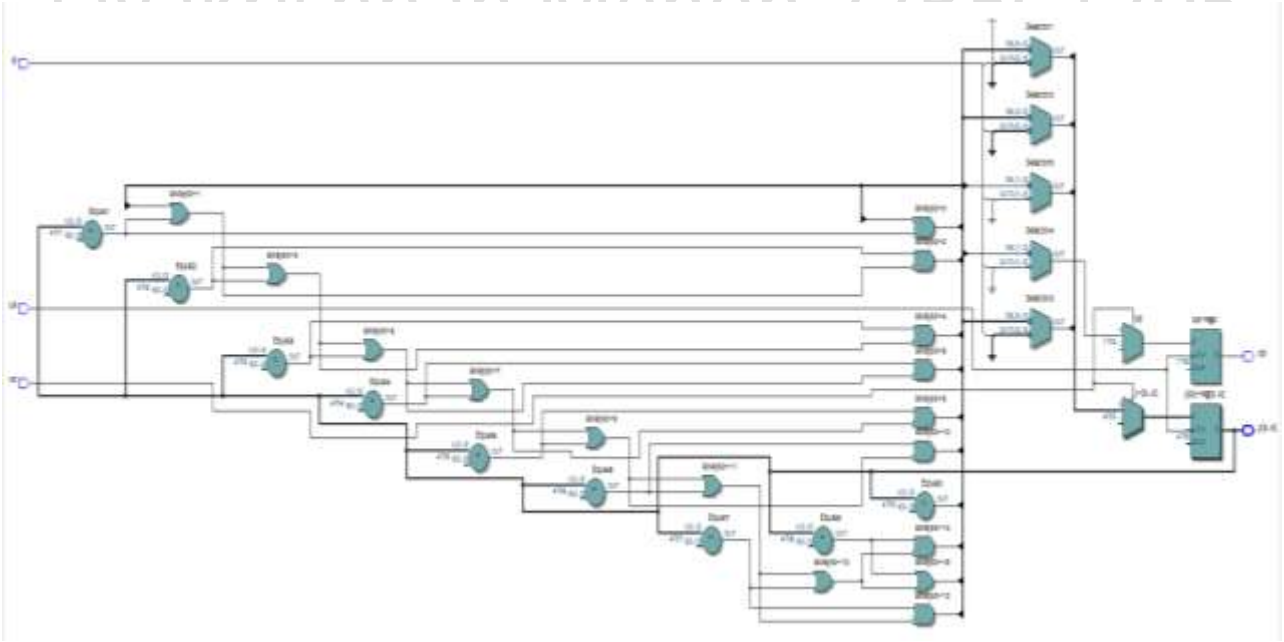
Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate
clk	Input	PIN_aj4	3B	B3B_N0	2.5 V (default)		12mA (default)	
ip	Input	PIN_ab30	5B	B5B_N0	2.5 V (default)		12mA (default)	
op	Output	PIN_aa24	5A	B5A_N0	2.5 V (default)		12mA (default)	1 (default)
rst	Input	PIN_y27	5B	B5B_N0	2.5 V (default)		12mA (default)	
y[3]	Output	PIN_ac22	4A	B4A_N0	2.5 V (default)		12mA (default)	1 (default)
y[2]	Output	PIN_ab22	5A	B5A_N0	2.5 V (default)		12mA (default)	1 (default)
y[1]	Output	PIN_af24	4A	B4A_N0	2.5 V (default)		12mA (default)	1 (default)
y[0]	Output	PIN_ae24	4A	B4A_N0	2.5 V (default)		12mA (default)	1 (default)

RTL Viewer:

Mealy:

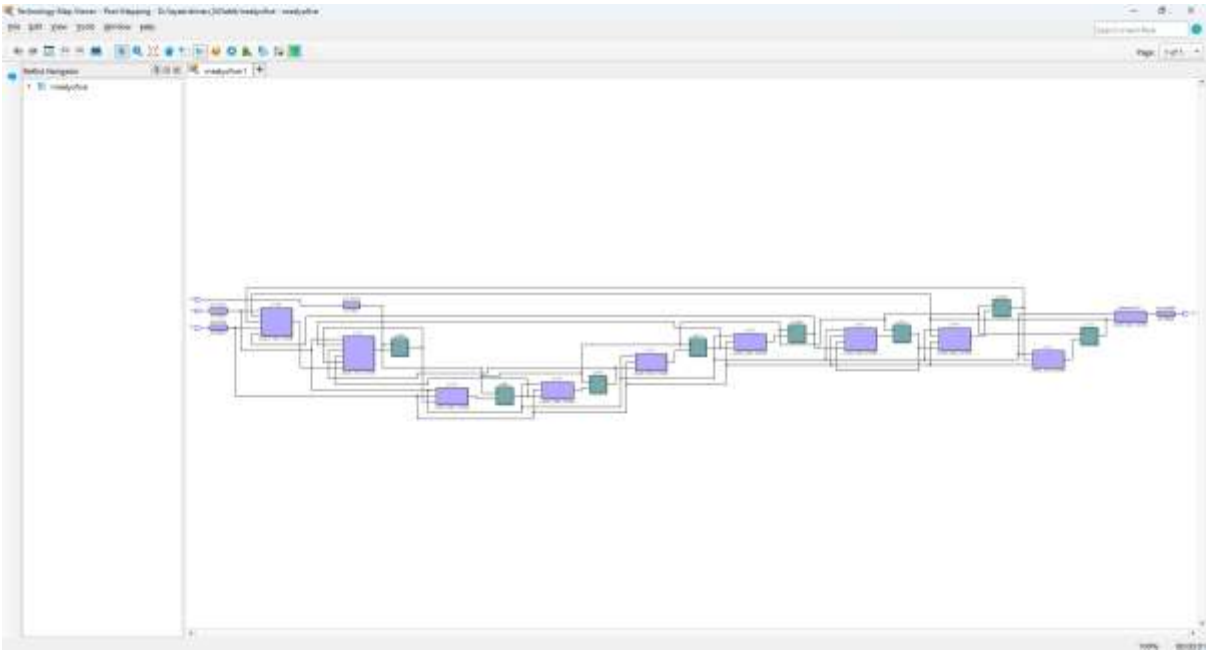


Moore:

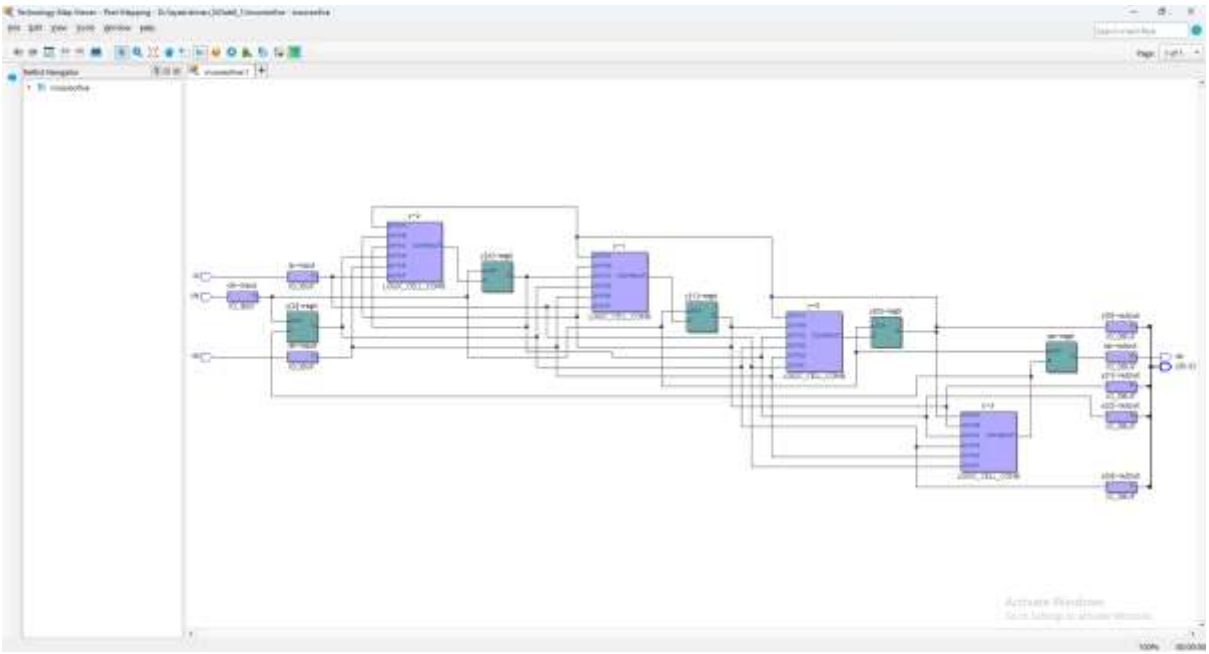


Technology Map Viewer:

Mealy:

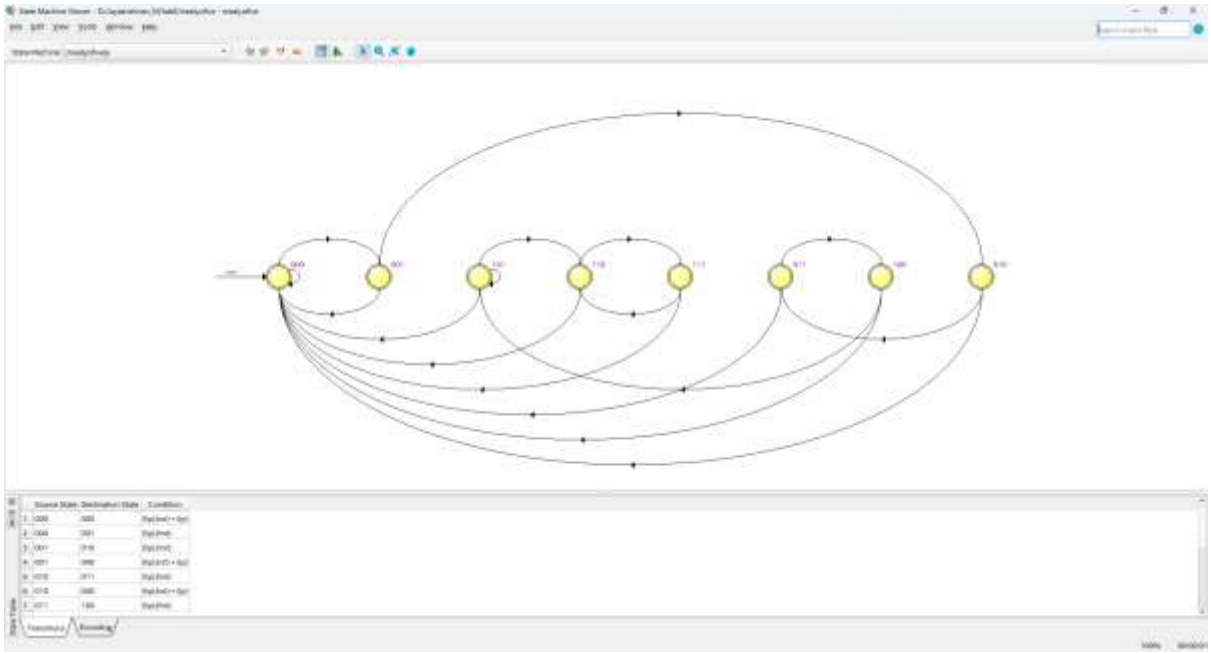


Moore:

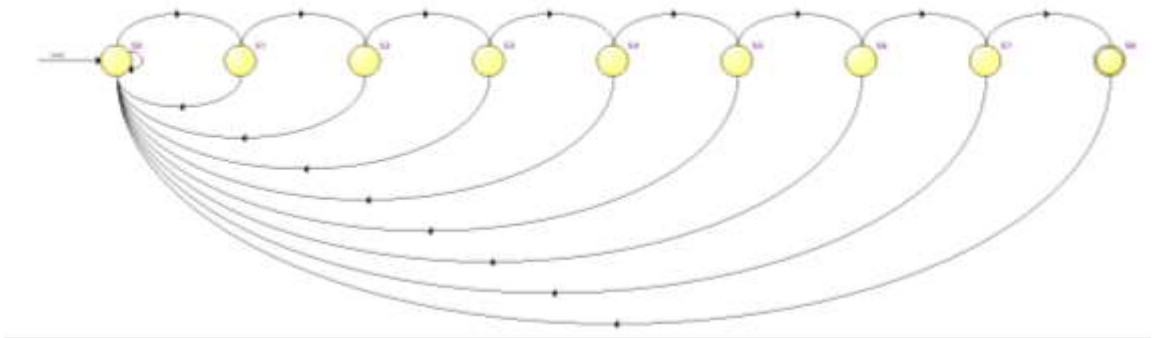


State Machine Viewer:

Mealy:



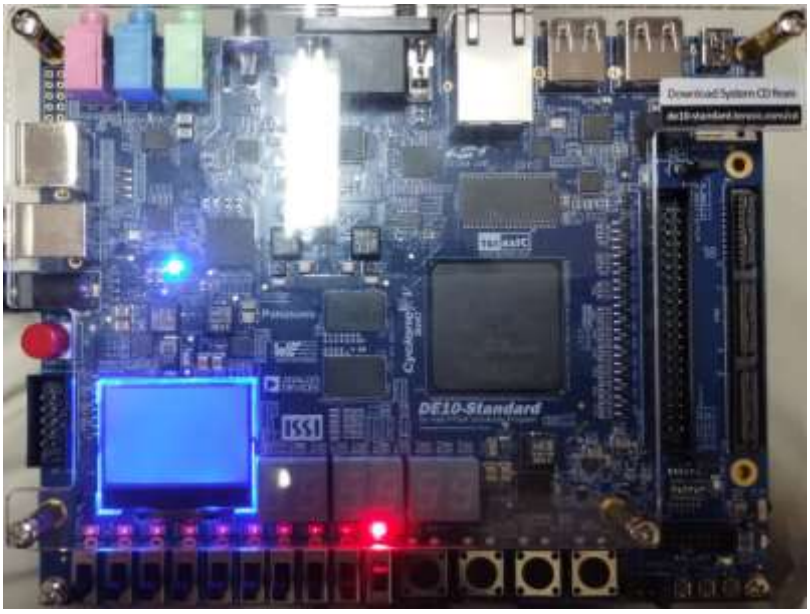
Moore:



Hardware Implementation:

Mealy:

State: 111, Input: 1, Output:1



Moore:

State:1000, Input: 1, Output:1

Result:

Hence, Verilog RTL codes for sequence detection using Mealy and Moore Finite State Machines., were successfully verified and implemented using Quartus Prime. The given task was completed successfully