Experiment - 7: FSM Based Vending Machine Simulation Using Quartus Prime

NAME: Jayakrishnan Menon

REG NO: 22BEC1205

DATE: 17/02/2025

Aim:
Write a Verilog RTL code for Simulation and Implementation of n FSM based Vending Machine. 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

The operation of an FSM based vending machine can be described using a mathematical model with a finite number of states, where the machine transitions between these states based on input signals like coin insertions or button presses, ultimately dispensing a product when the correct amount is received and a selection is made; essentially, it's a way to represent the vending machine's behaviour as a series of logical steps with clear defined states like "idle," "accepting coins," "selecting product," and "dispensing item.".

Key points about vending machine FSMs:

States:
Each state represents a specific point in the vending machine's operation, like "waiting for input," "collecting coins," "calculating total amount," "ready to dispense".

Transitions:
Movements between states are triggered by input signals like coin insertion, button presses, or sensors detecting the presence of a product.

Inputs:
These are the signals that cause the machine to change states, usually including coin values or product selection buttons.

Outputs:
Actions taken based on the current state, such as displaying the current amount collected or activating the dispensing mechanism.

Benefits of using FSM for vending machines:

Clear design process:
Helps visualize the complex logic of a vending machine by breaking it down into simple states and transitions.

Easy implementation:
Can be readily implemented in hardware using logic circuits or software to control the machine's behaviour.

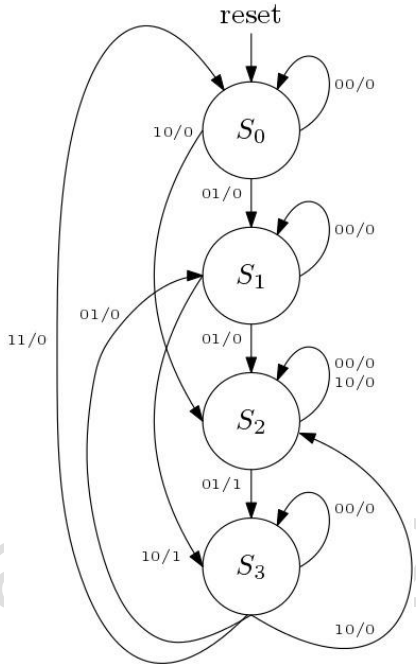
Error detection:
By analysing state transitions, potential issues with the machine's logic can be identified.

Lab Task:

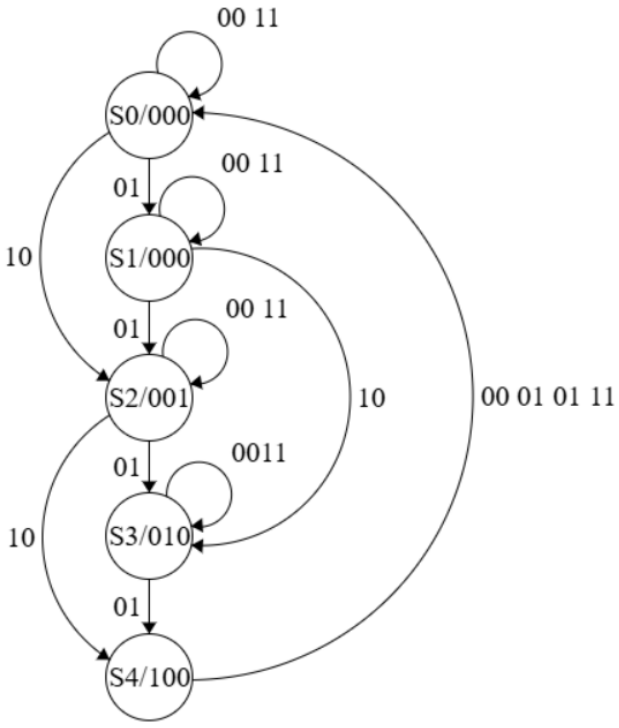
To create a vending machine that dispenses a single product when the amount of money inserted reaches no more and no less than the set threshold.

To create a vending machine that dispenses multiple products, depending on the amount of money inserted.

FSM for 1st Task:



FSM for 2nd Task:

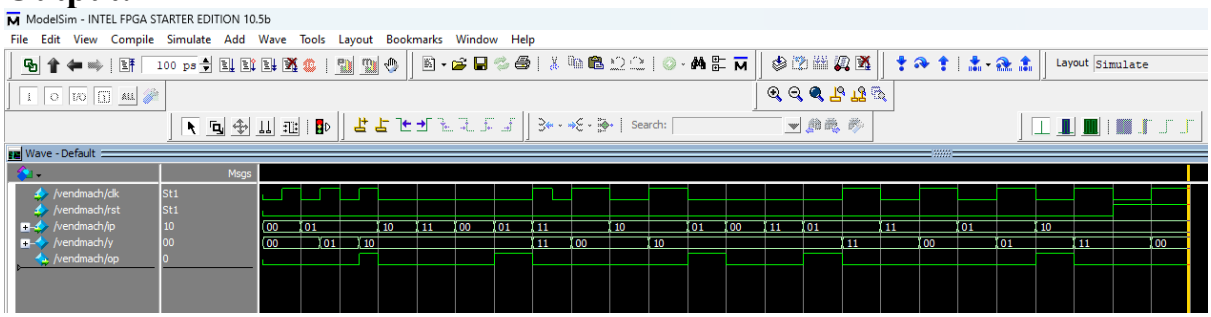


Source code and Outputs:

FSM for 1st Task:

```
module vendmach(input clk, rst, input [1:0]ip, output reg op);
reg [1:0]y,Y,A,B,C,D;

initial begin
A=2'b00; B=2'b01; C=2'b10; D=2'b11;
y=A; Y=2'bxx; 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==2'b01)        begin Y=B; op=1'b0; end
   else if (ip==2'b10)  begin Y=C; op=1'b0; end
   else if (ip==2'b00)  begin Y=A; op=1'b0; end
B: if(ip==2'b01)        begin Y=C; op=1'b0; end
   else if (ip==2'b10)  begin Y=D; op=1'b1; end
   else if (ip==2'b00)  begin Y=B; op=1'b0; end
C: if(ip==2'b01)        begin Y=D; op=1'b1; end
   else if (ip==2'b10)  begin Y=C; op=1'b0; end
   else if (ip==2'b00)  begin Y=C; op=1'b0; end
D: if(ip==2'b01)        begin Y=B; op=1'b0; end
   else if (ip==2'b10)  begin Y=C; op=1'b0; end
   else if (ip==2'b00)  begin Y=D; op=1'b0; end
   else if (ip==2'b11)  begin Y=A; op=1'b0; end //jump to start after transaction
default:    begin Y=2'bxx;  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:

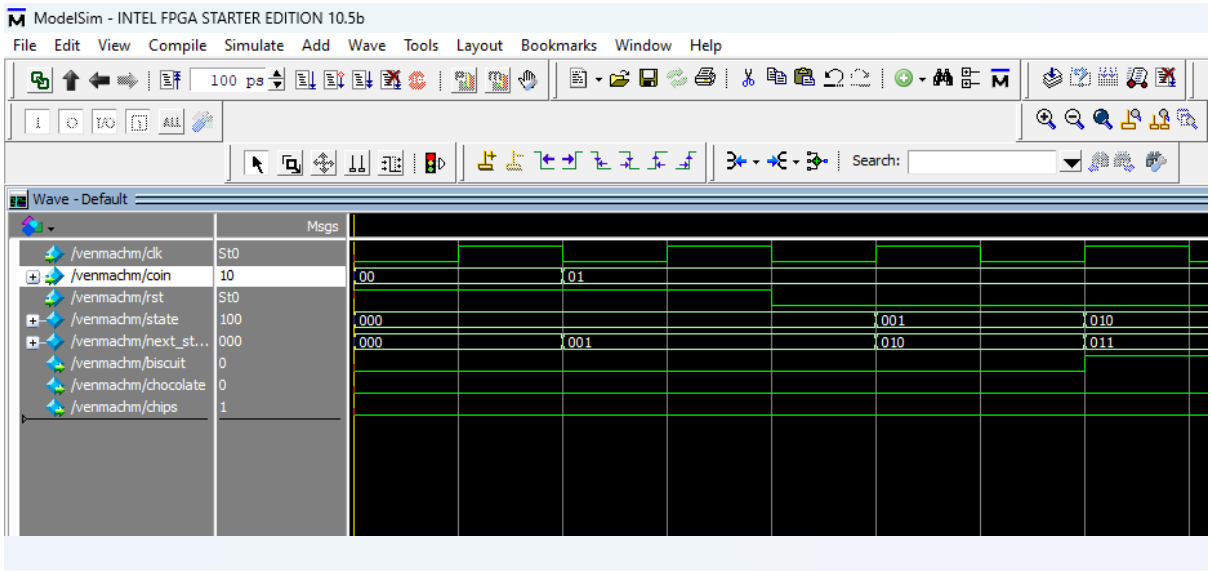


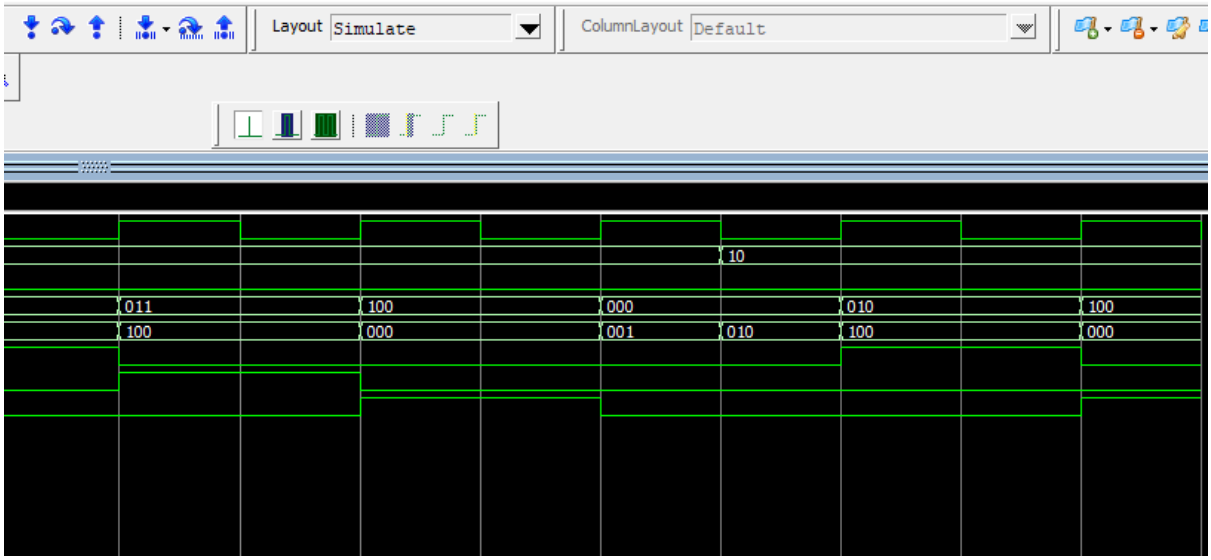
FSM for 2nd Task:

```
module venmachm(
    input clk,
    input rst,
    input [1:0] coin,
    output reg chocolate,
output reg chips,
output reg biscuit
);

parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011 ,S4= 3'b100;
reg[2:0]  state,next_state;

    always @(posedge clk or posedge rst) begin
        if (rst)
            state <= S0;
        else
            state <= next_state;
    end

    always @(*) begin
        next_state = state;
        chocolate = 0;
 chips=0;
 biscuit=0;
        case (state)
            S0: begin
                if (coin == 2'b01)
                    next_state = S1;
                else if (coin == 2'b10)
                    next_state = S2;
            end
            S1: begin
```

```
            if (coin == 2'b01)
                next_state = S2;
            else if (coin == 2'b10)
                next_state = S3;
        end
        S2: begin
biscuit =1;
            if (coin == 2'b01)
                next_state = S3;
 else if(coin == 2'b10)
 next_state = S4;
        end
        S3: begin
chocolate =1;
            if(coin==2'b01)
next_state=S4;

        end
S4: begin
chips =1;
next_state=S0;
        end
      endcase
   end

endmodule
module vending_machine_tb;
   reg clk;
   reg rst;
   reg [1:0] coin;
   wire chocolate,chips,biscuit;
```

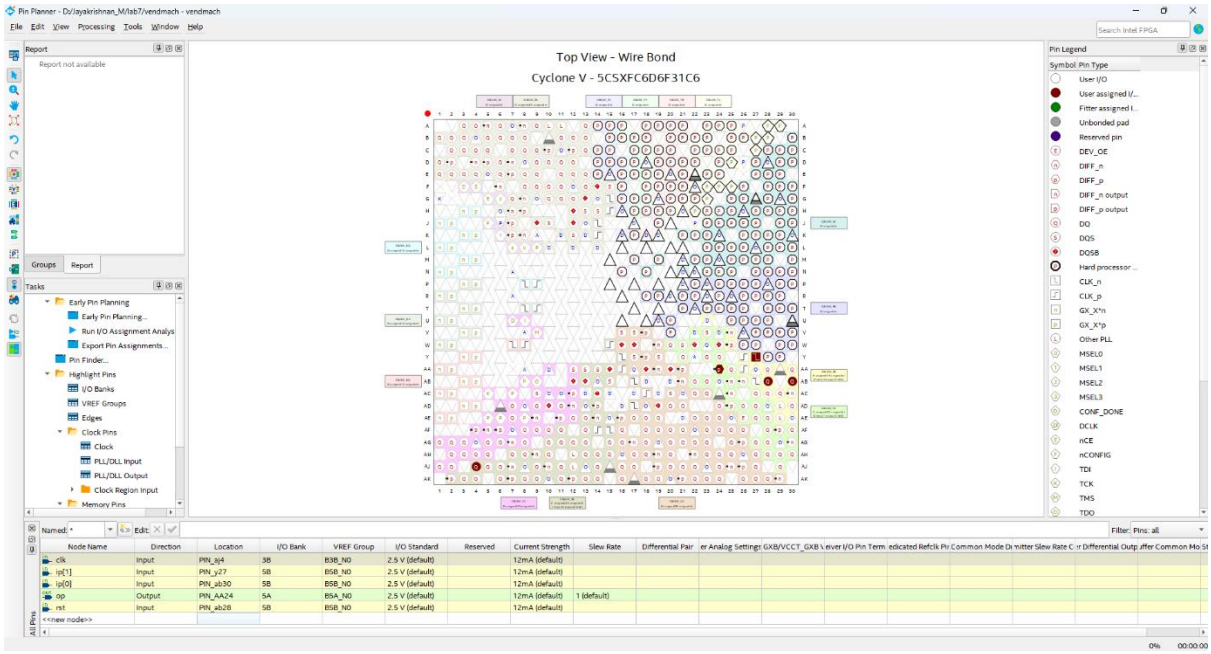
Output:


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Search:
Wave - Default
Msgs
/venmachm/clk
/venmachm/coin
/venmachm/rst
/venmachm/state
/venmachm/next_st...
/venmachm/biscuit
/venmachm/chocolate
/venmachm/chips
St0
10
St0
100
000
0
0
1
00
01
000
001
000
001
010
010
011

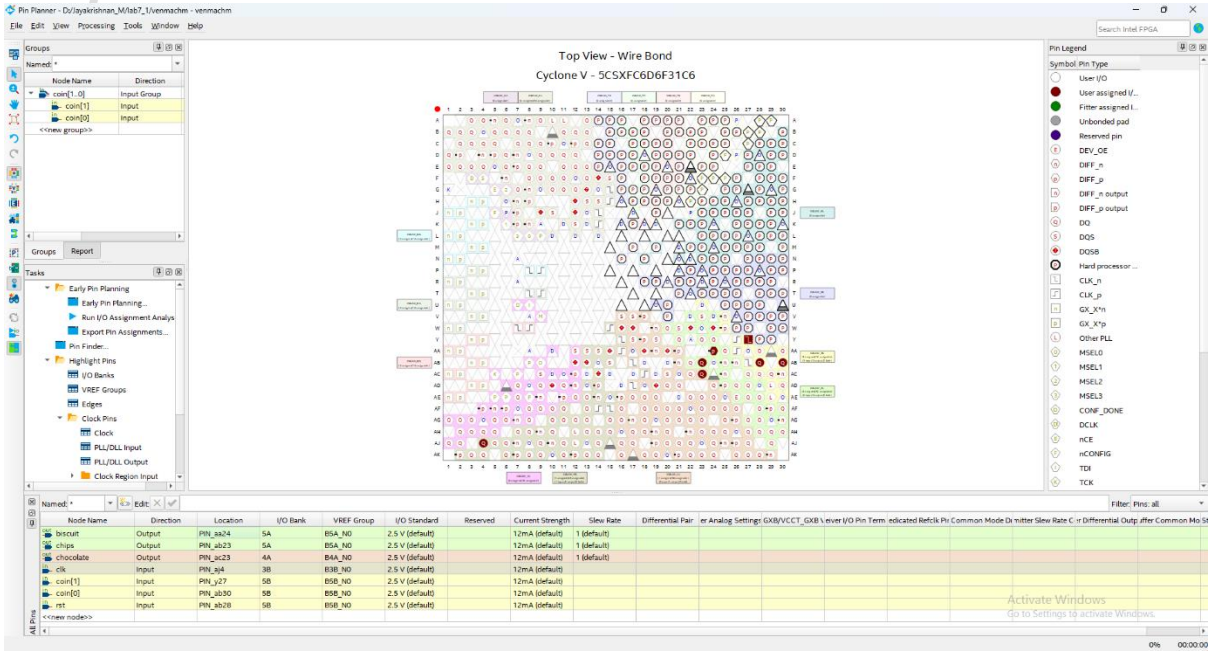

Layout Simulate
ColumnLayout Default
10
011
100
100
000
000
001
010
010
100
100
000

Pin Planning:

For 1st Task:

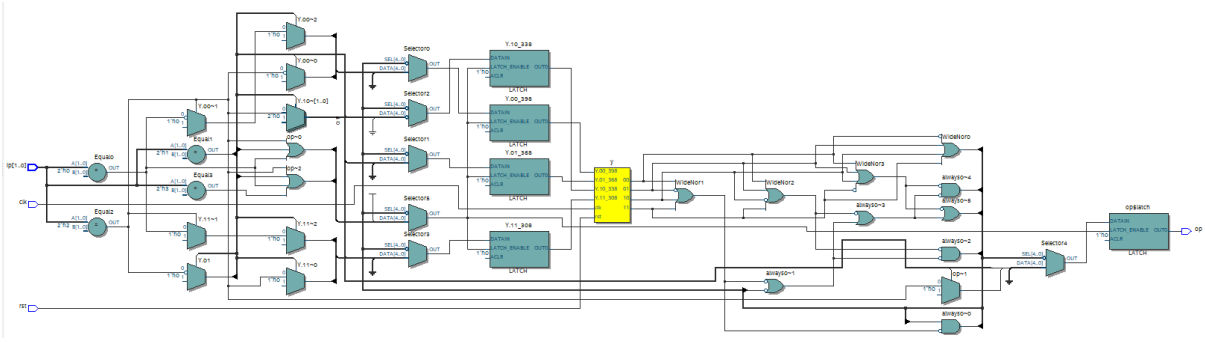


For 2nd Task:

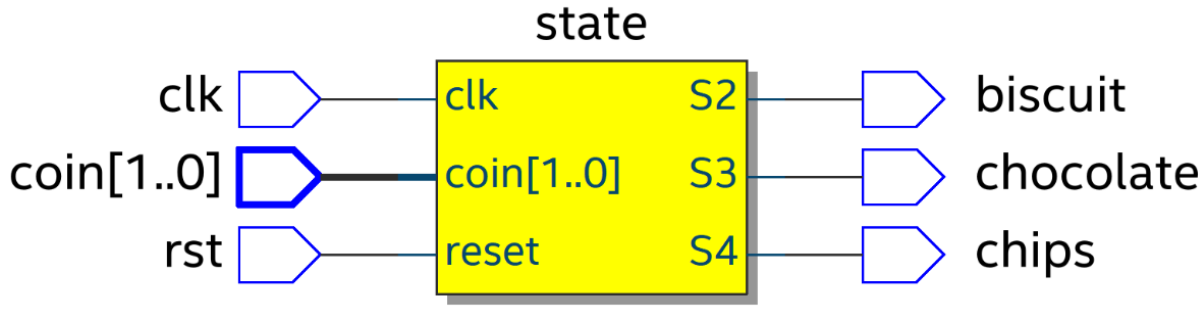


RTL Viewer:

For 1st Task:

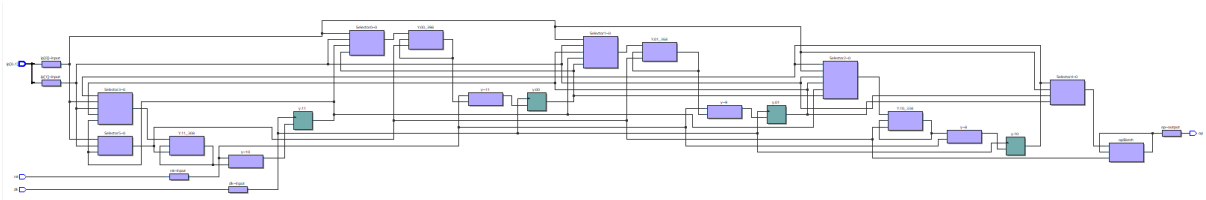


For 2nd Task:

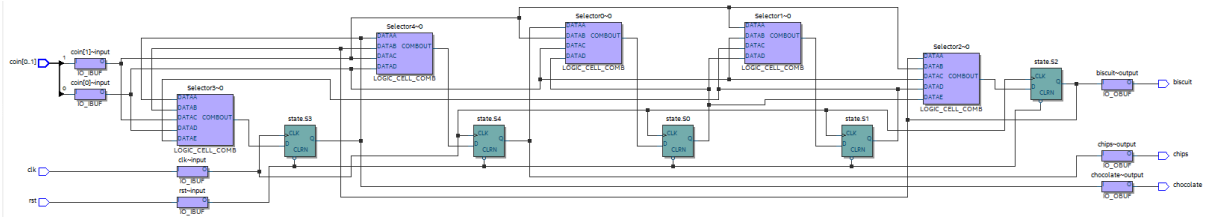


Technology Map Viewer:

For 1st Task:

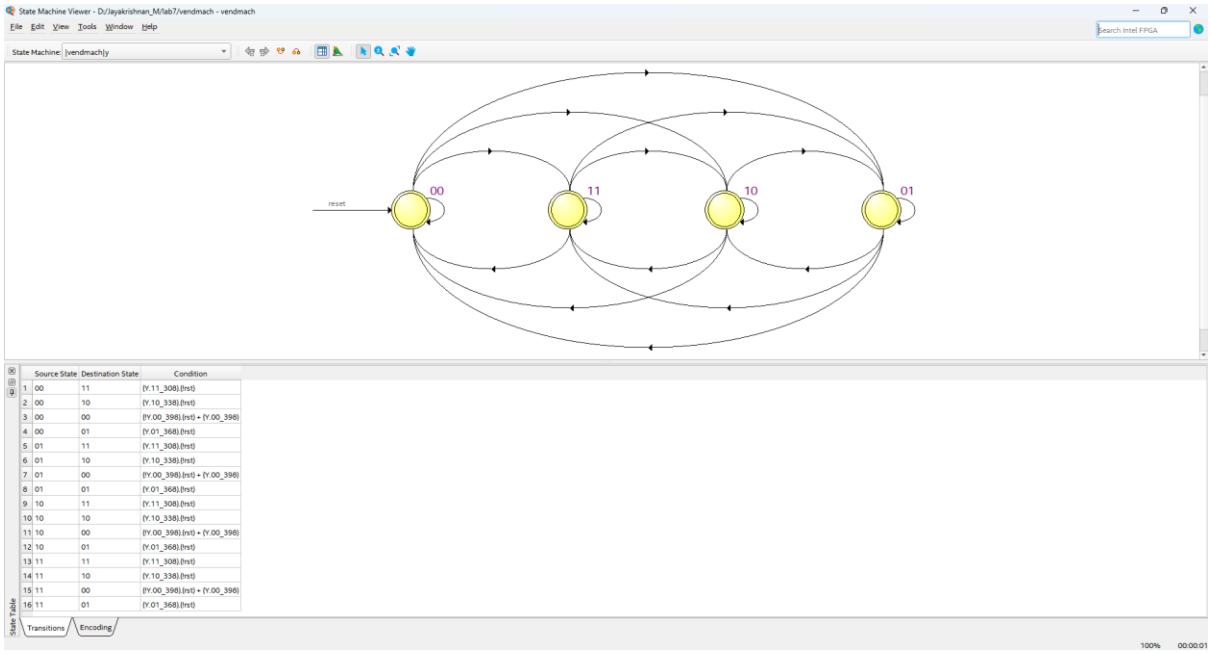


For 2nd Task:

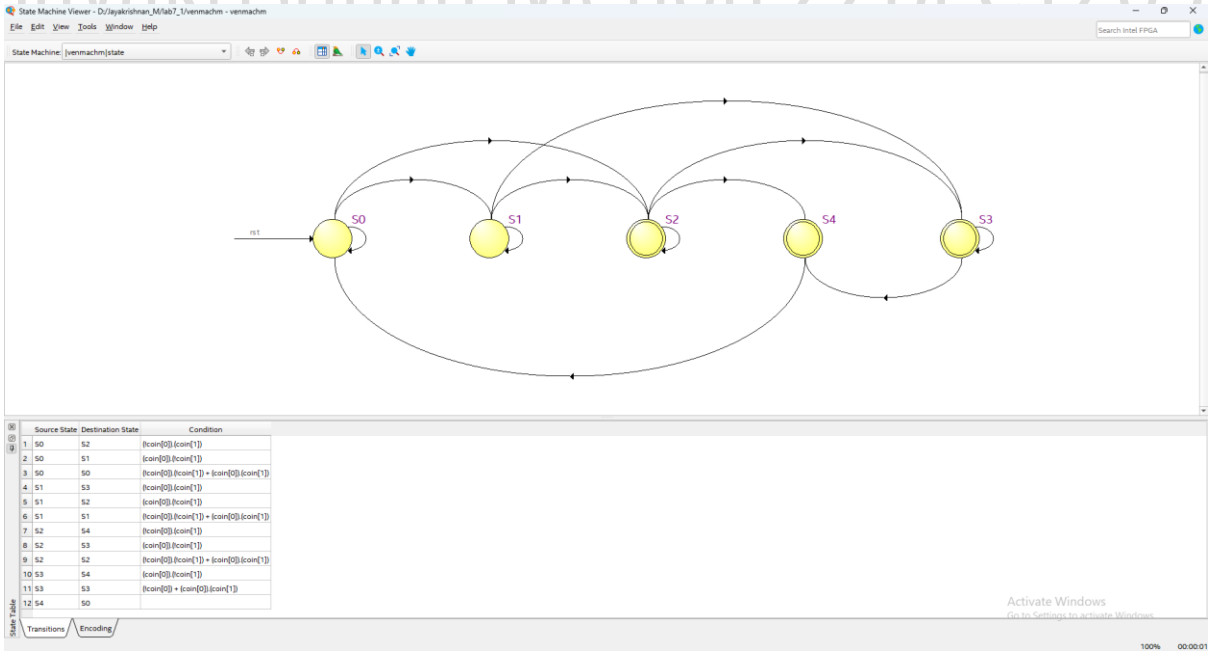


State Machine Viewer:

For 1st Task:



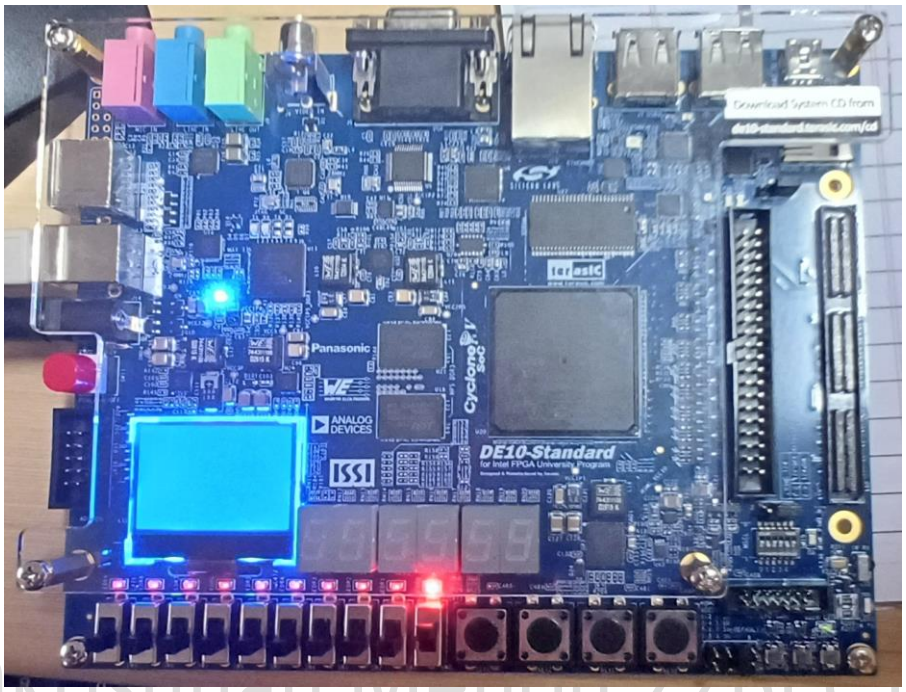
For 2nd Task:



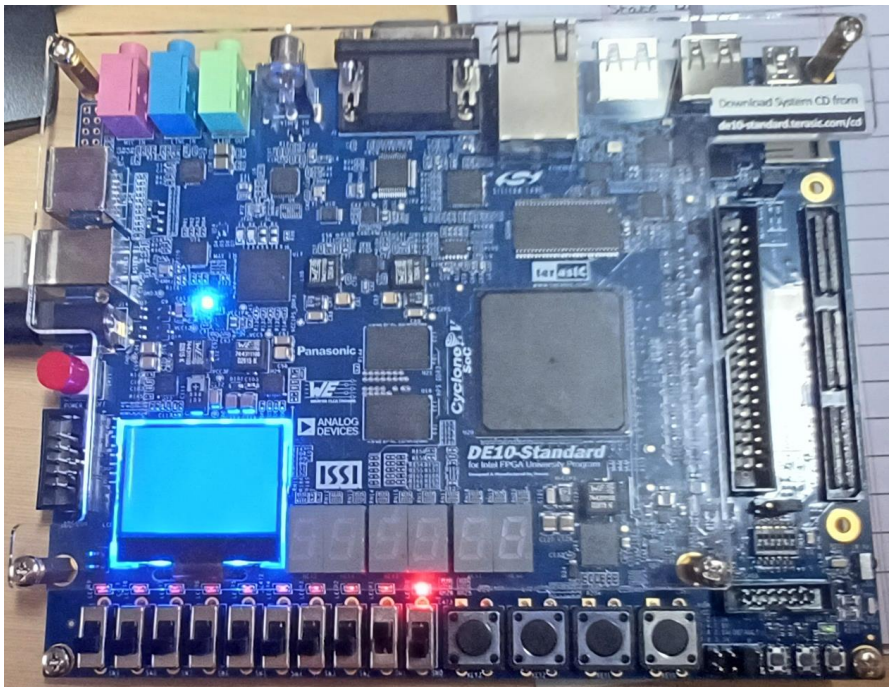
Hardware Implementation:

For 1st Task:

State: S2, Input: 01, Output:1

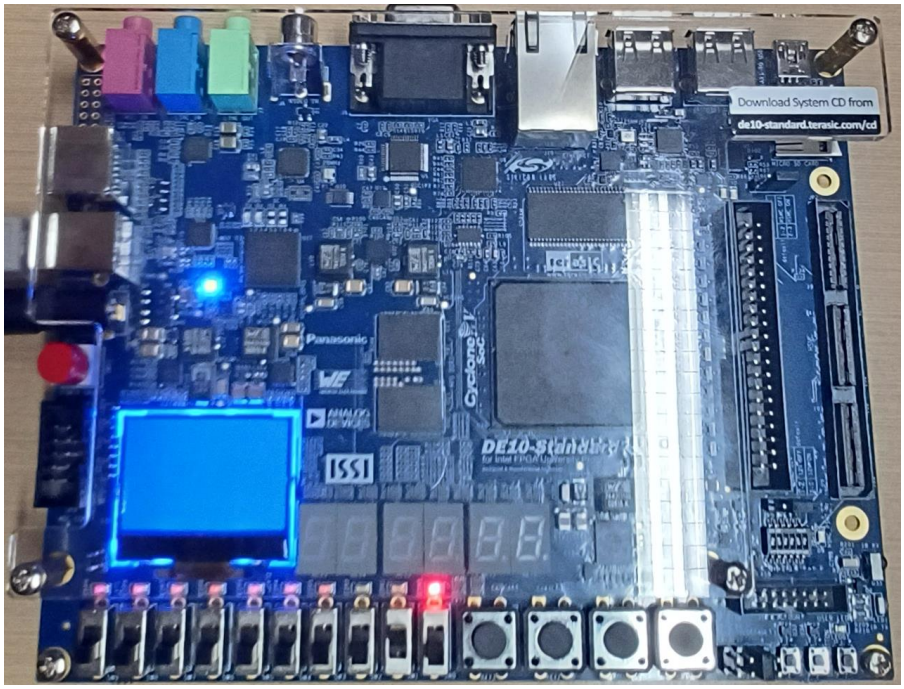


State: S1, Input: 10, Output:1

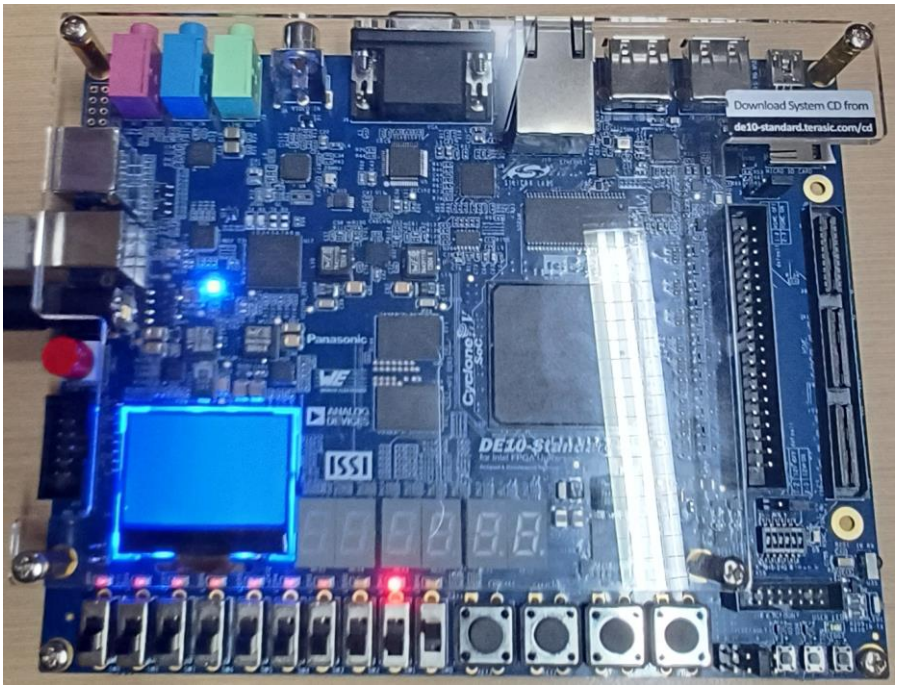


For 2nd Task:

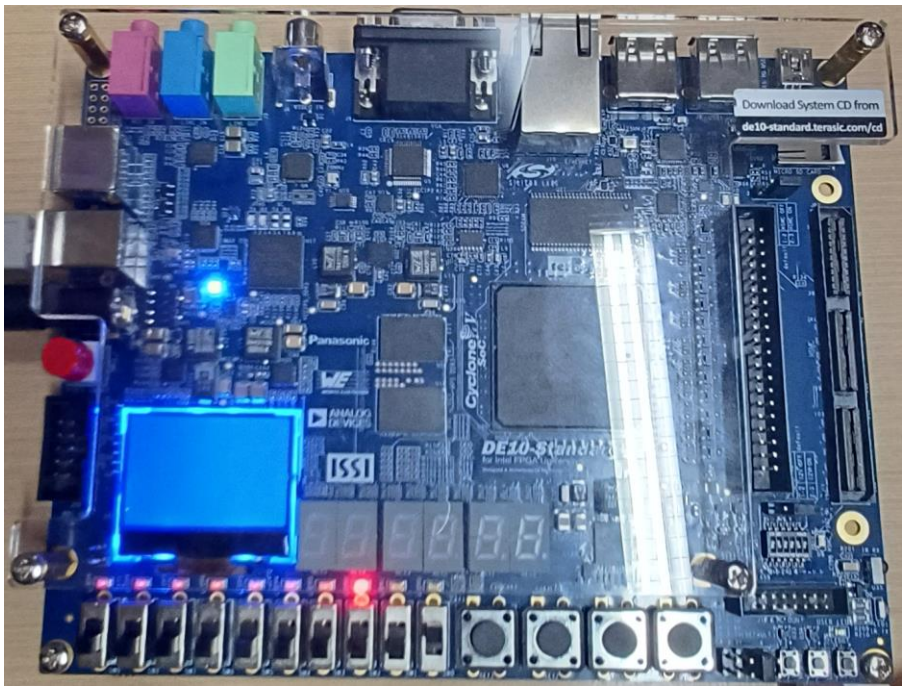
State: S2, Input: 01, Output: 1



State: S3, Input: 01, Output: 1



State: S4, Input: 01, Output: 1



Result:

Hence, Verilog RTL codes for both Vending Machines were successfully verified and implemented using Quartus Prime. The given task was completed successfully