LOGIC PROJECT

Vending Machine

DR. Howida Abd Ailatif

Digital Logic Design, CS221 - Assignment

ASSIGNMENT Title:

Vending Machine

No.	Name & Email	Academic Number	Task Completed
1.2	Mahmoud Galal Ramadan El-Gendy	20812021101328	Code, Simulation, Schematic diagram using Active HDL
	mahmoudgalal173.95@gmail.com		
2.1	Hassan Mohamed Hassan Ali	20812021101068	
	hassanalzyat.contact@gmail.com		
3.2	Karim Salah Eldin Aboumesalam	20812021101453	
	kimos20139@gmail.com		
4.2	Menna Allah Essam Ahmed Salem	20812021200937	20TK (Truth Tables - K-Maps Boolean Functions)
	me762927@gmail.com		
5.2	Mohamed Gehad Hussien Metwally	20812021100025	Document, State Diagram
	engmgehad@gmail.com		
6.2	Nancy Ayman Nabil Mohamed	20812021200506	10TK – 15TK (Truth Tables K-Maps Boolean Functions)
	na8594761@gmail.com		
7.1	Reham Hamdy Mohamed Ibrahim	20812021201109	
	rehamhamdi53@gmail.com		
8.1	Shahd AlSayed Ahmed Ali	20812021200543	
	shahdelsayed3303@gmail.com		
9.1	Shahd AbdelNabi Mahmoud AbdelNabi	20812021200898	Design using Proteus
	shahdelshamy964@gmail.com		
10.1	Shehab Shukri AbdelNabi Ibrahim	20812021101006	
	sh.s.ghazal2003@gmail.com		

“The assignment Aim”

A vending machine is a self-service device that allows customers to purchase products or services without the need for human intervention. It is a common sight in various public spaces such as schools, offices, airports, train stations, and shopping centers.

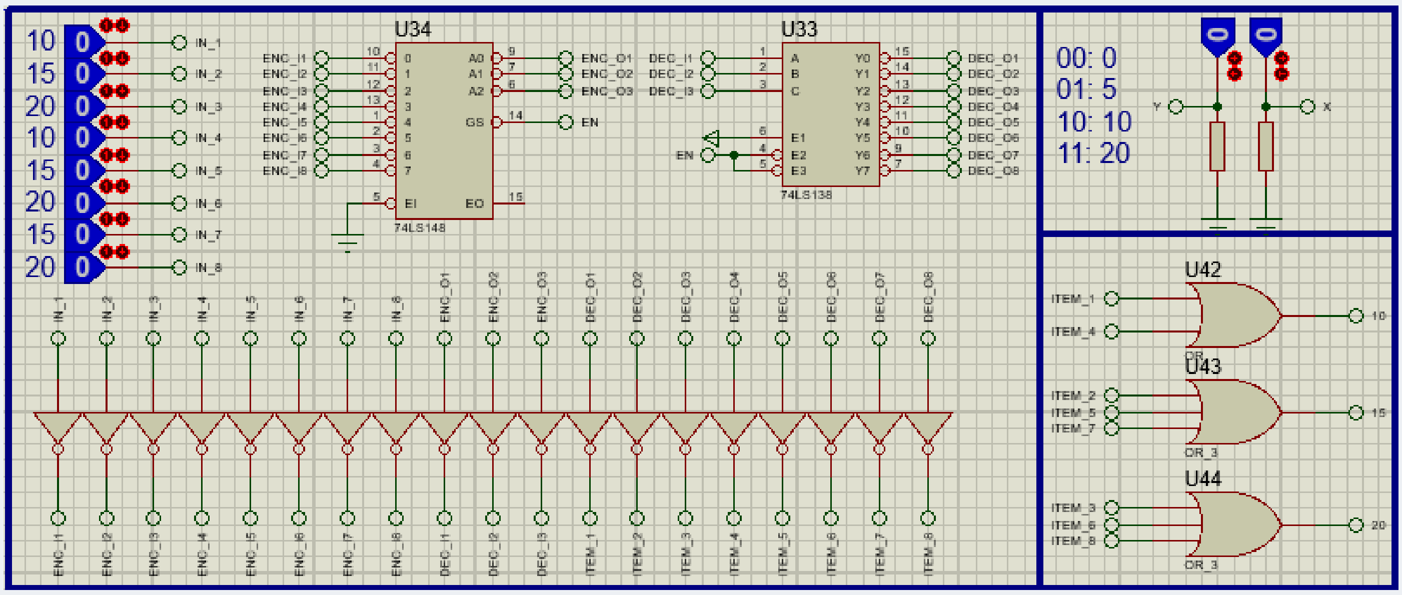
The primary purpose of a vending machine is to provide convenience and accessibility to customers. It typically consists of a cabinet or enclosure that houses the products, a selection mechanism, a payment system, and a dispensing mechanism.

The content of our vending machine:

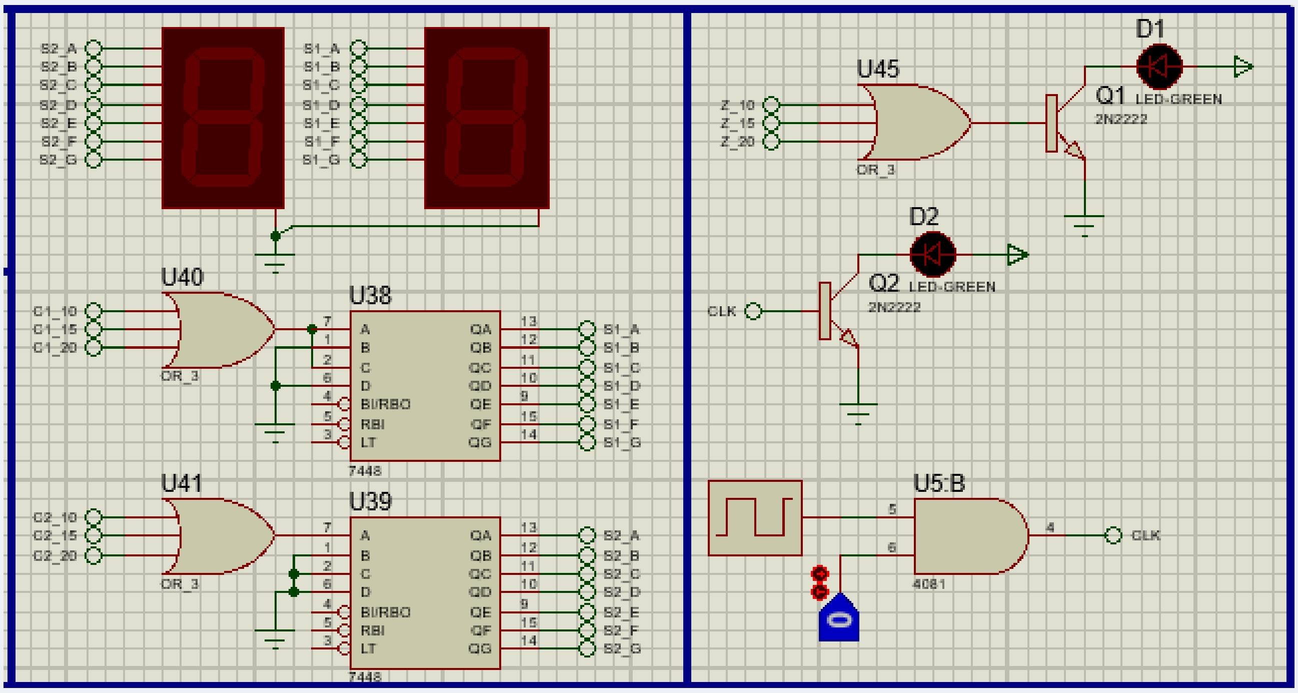
contains 3 products. You can enter prices of 10, 15, and 20 If you enter price that bigger than the product price you will get the product and the exchange of your many. If you enter price less than product price the Machine will wait to the rest for specific time if you don't the process will be canceled, and machine will get out your money.

The Problem Solution

Inputs:



Outputs:

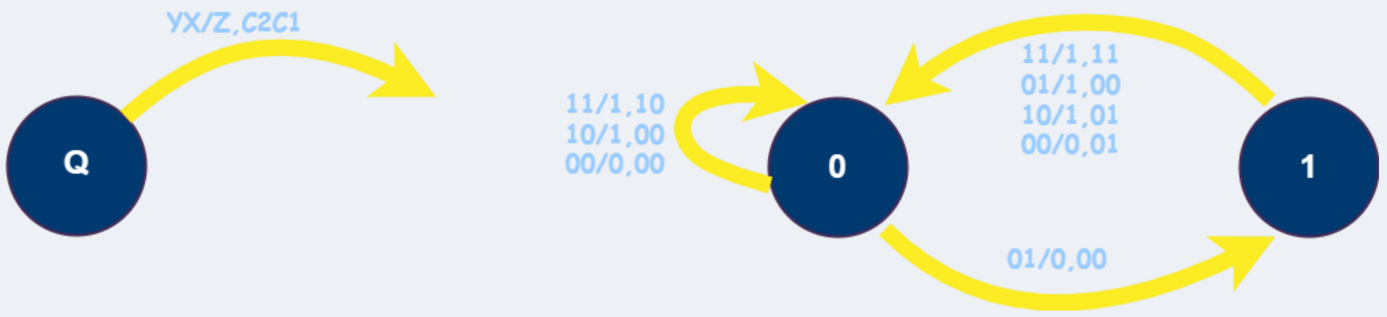


For 10TK

Truth Tables:

Q	Y	X	Q*	Z	C2	C1
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	1	1	0	1	1	0
1	0	0	0	0	0	1
1	0	1	0	1	0	0
1	1	0	0	1	0	1
1	1	1	0	1	1	1

State Diagram:



K-Maps & Boolean Functions:

Q \ YX	00	01	11	10
0	0	1	0	0
1	0	0	0	0

$Q^* = Q'Y'X$

Q \ YX	00	01	11	10
0	0	0	1	1
1	0	1	1	1

$Z = Y + QX$

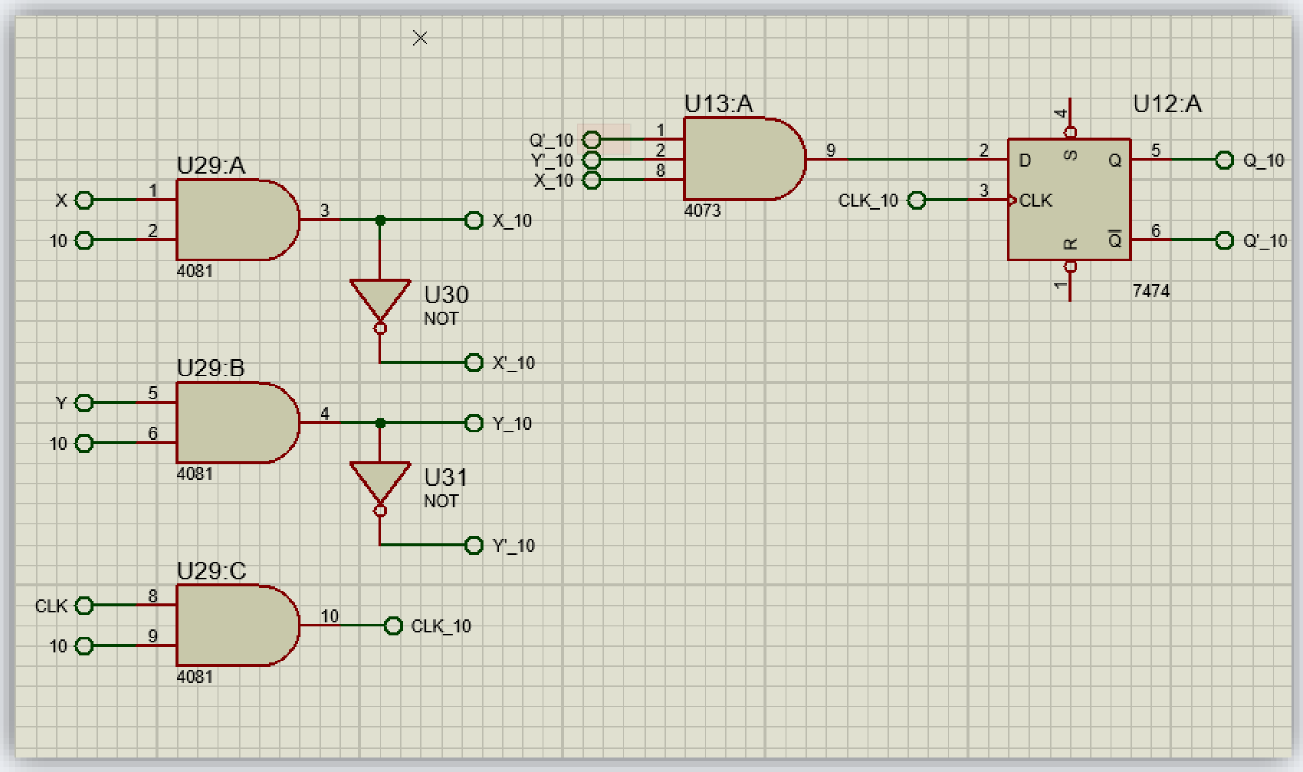
Q \ YX	00	01	11	10
0	0	0	1	0
1	0	0	1	0

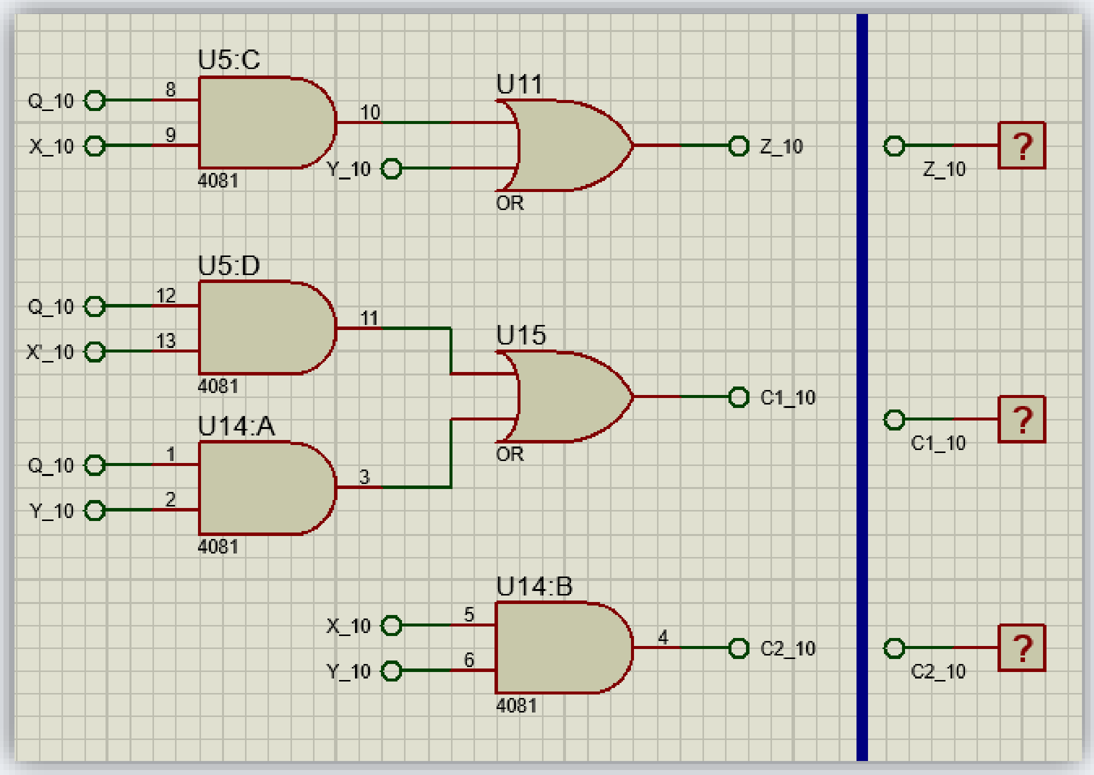
$C_2 = XY$

Q \ YX	00	01	11	10
0	0	0	0	0
1	1	0	1	1

$C_1 = QY + QX'$

Logic Circuits Design:



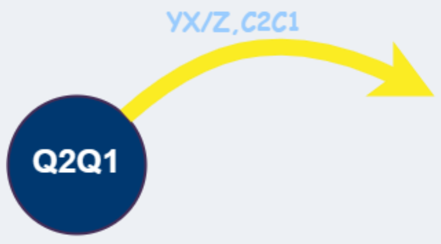


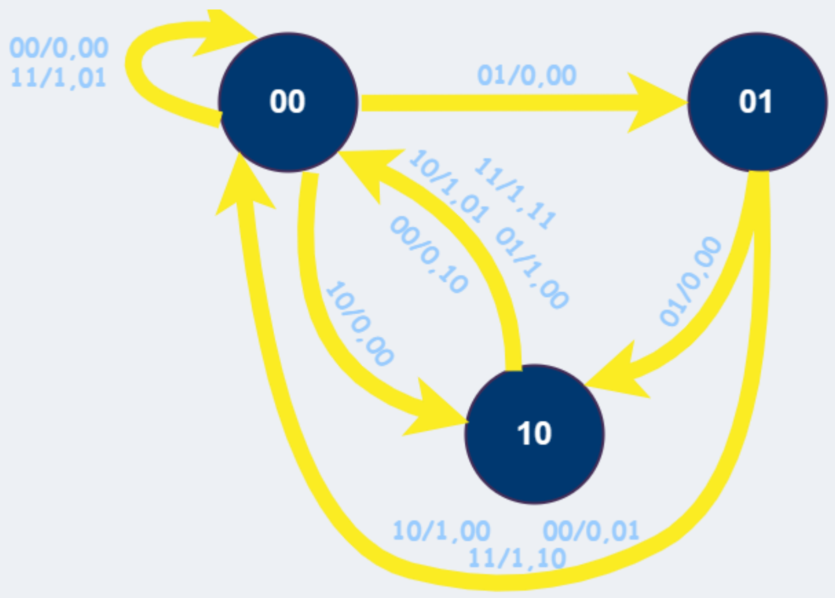
For 15TK

Truth Tables:

Q2	Q1	Y	X	Q2*	Q1*	Z	C2	C1
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	0	1	0	1
0	1	0	0	0	0	0	0	1
0	1	0	1	1	0	0	0	0
0	1	1	0	0	0	1	0	0
0	1	1	1	0	0	1	1	0
1	0	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0	0
1	0	1	0	0	0	1	0	1
1	0	1	1	0	0	1	1	1
1	1	0	0	X	X	X	X	X
1	1	0	1	X	X	X	X	X
1	1	1	0	X	X	X	X	X
1	1	1	1	X	X	X	X	X

State Diagram:





K-Maps & Boolean Functions:

YX / Q2 Q1	00	01	11	10
00	0	0	0	1
01	0	1	0	0
11	X	X	X	X
10	0	0	0	0

$Q_2^* = Q_2'Q_1'YX' + Q_1Y'X$

YX / Q2 Q1	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	X	X	X	X
10	0	0	0	0

$Q_1^* = Q_2'Q_1'Y'X$

YX / Q2 Q1	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	X	X	X	X
10	0	1	1	1

$Z = YX + Q_1Y + Q_2X + Q_2Y$

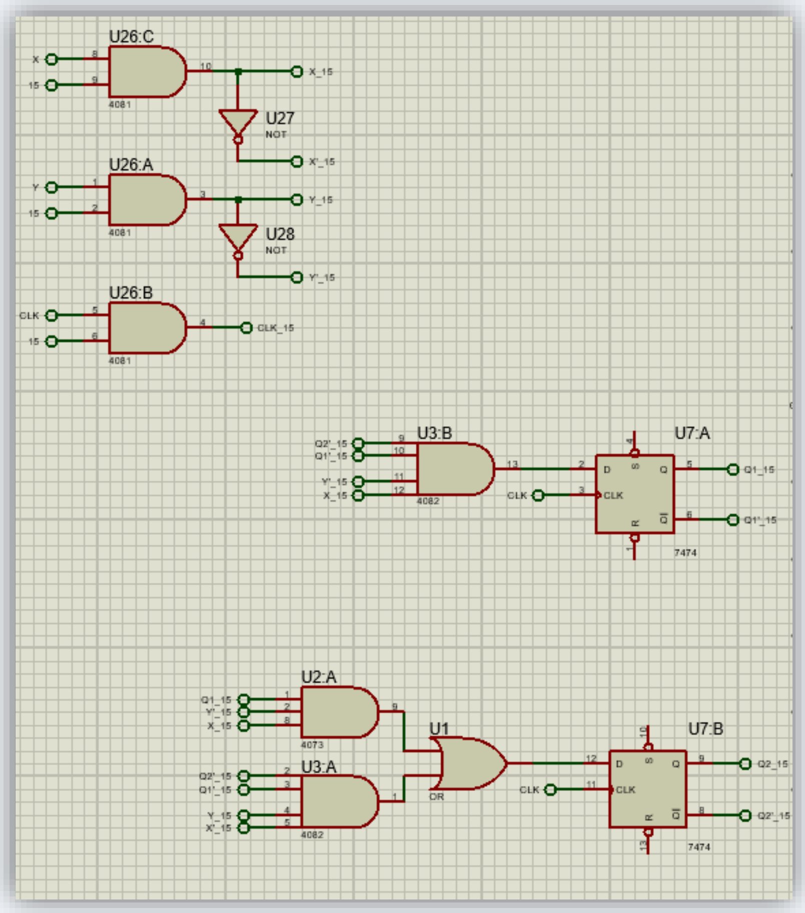
YX / Q2 Q1	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	1	0	1	0

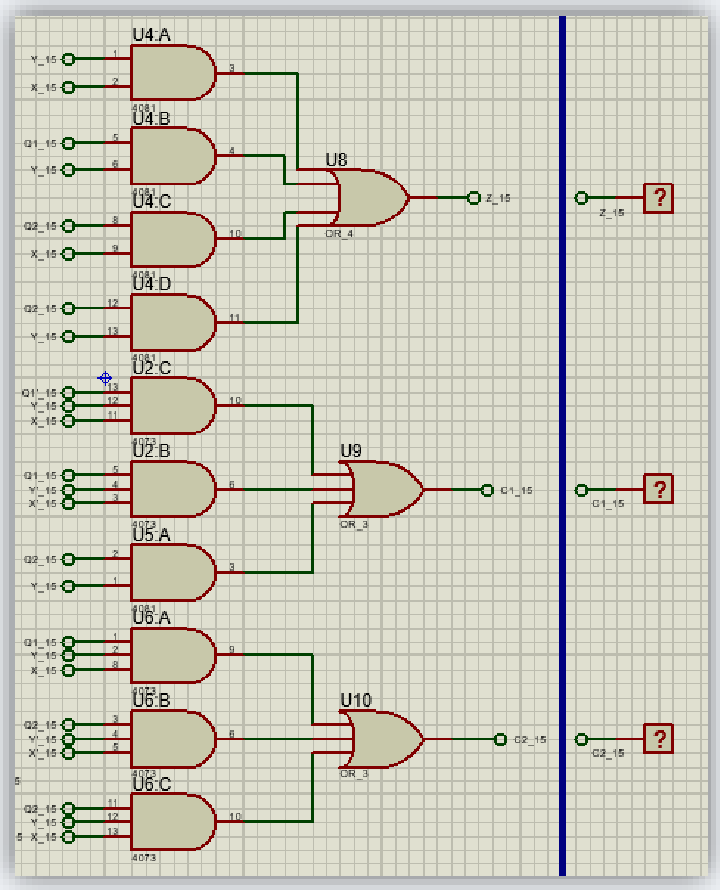
$C_2 = Q_2Y'X' + Q_1YX + Q_2YX$

YX / Q2 Q1	00	01	11	10
00	0	0	1	0
01	1	0	0	0
11	X	X	X	X
10	0	0	1	1

$C_1 = Q_2Y + Q_1'YX + Q_1Y'X'$

Logic Circuits Design:



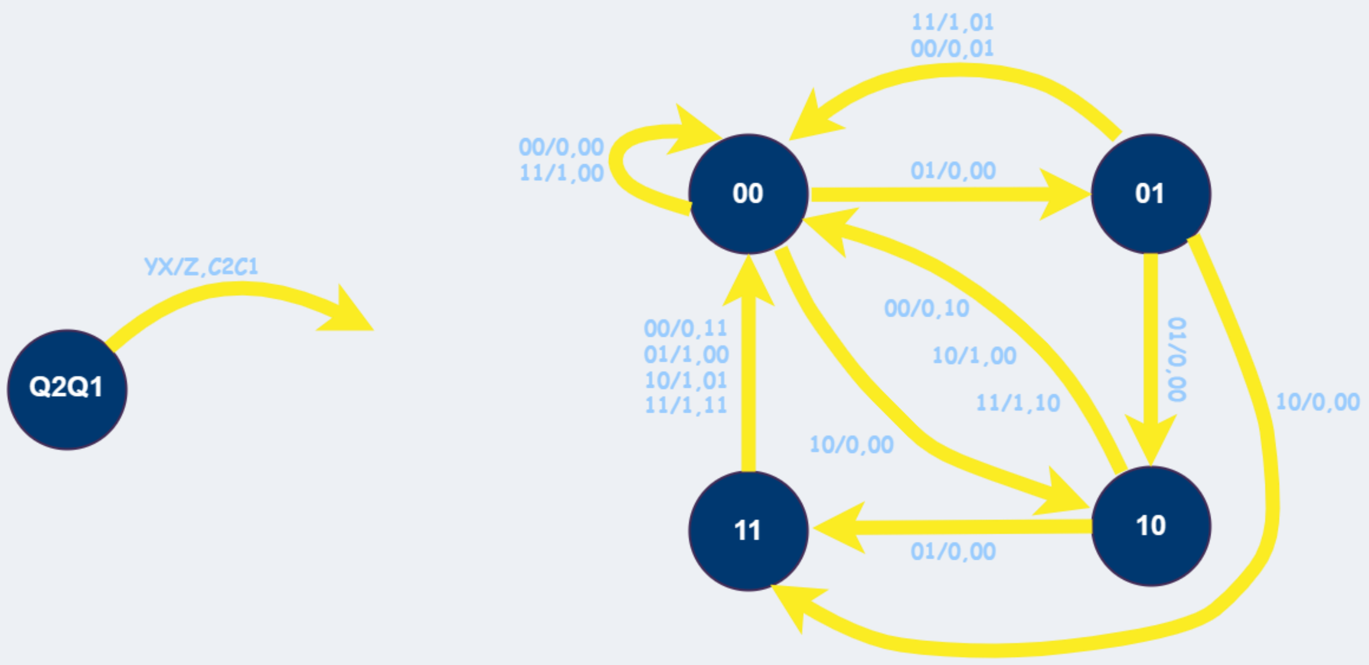


For 20TK

Truth Tables:

Q2	Q1	Y	X	Q2*	Q1*	Z	C2	C1
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	0	1	0	0
0	1	0	0	0	0	0	0	1
0	1	0	1	1	0	0	0	0
0	1	1	0	1	1	0	0	0
0	1	1	1	0	0	1	0	1
1	0	0	0	0	0	0	1	0
1	0	0	1	1	1	0	0	0
1	0	1	0	0	0	1	0	0
1	0	1	1	0	0	1	1	0
1	1	0	0	0	0	0	1	1
1	1	0	1	0	0	1	0	0
1	1	1	0	0	0	1	0	1
1	1	1	1	0	0	1	1	1

State Diagram:



K-Maps & Boolean Functions:

YX / Q_2Q_1	00	01	11	10
00	0	0	0	1
01	0	1	0	1
11	0	0	0	0
10	0	1	0	0

$Q_2^* = Q_2'YX' + Q_2'Q_1Y'X + Q_2Q_1'Y'X$

YX / Q_2Q_1	00	01	11	10
00	0	1	0	0
01	0	0	0	1
11	0	0	0	0
10	0	1	0	0

$Q_1^* = Q_1'Y'X + Q_2'Q_1YX'$

YX / Q_2Q_1	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	1

$Z = YX + Q_2Y + Q_2Q_1X$

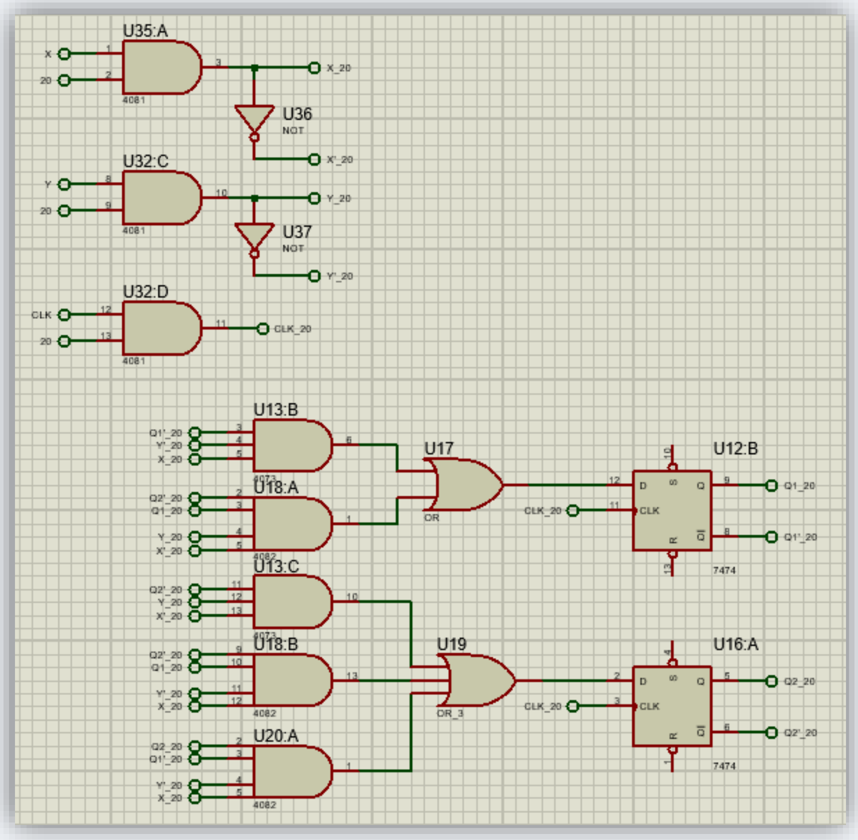
YX / Q_2Q_1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	1	0
10	1	0	1	0

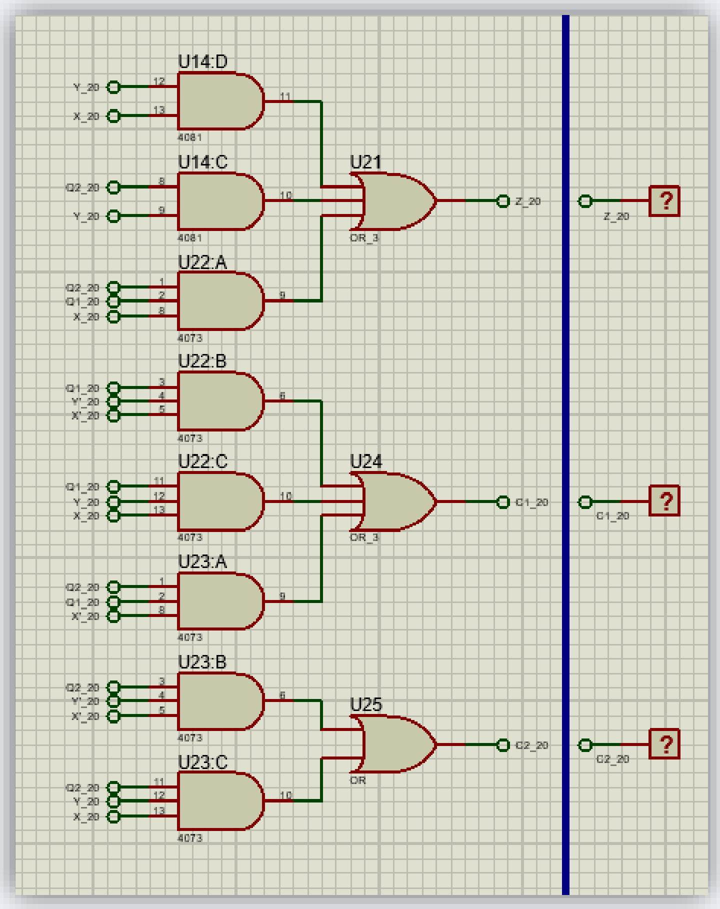
$C_2 = Q_2Y'X' + Q_2YX$

YX / Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	1	0	1	1
10	0	0	0	0

$C_1 = Q_2Q_1X' + Q_1YX + Q_1Y'X'$

Logic Circuits Design:

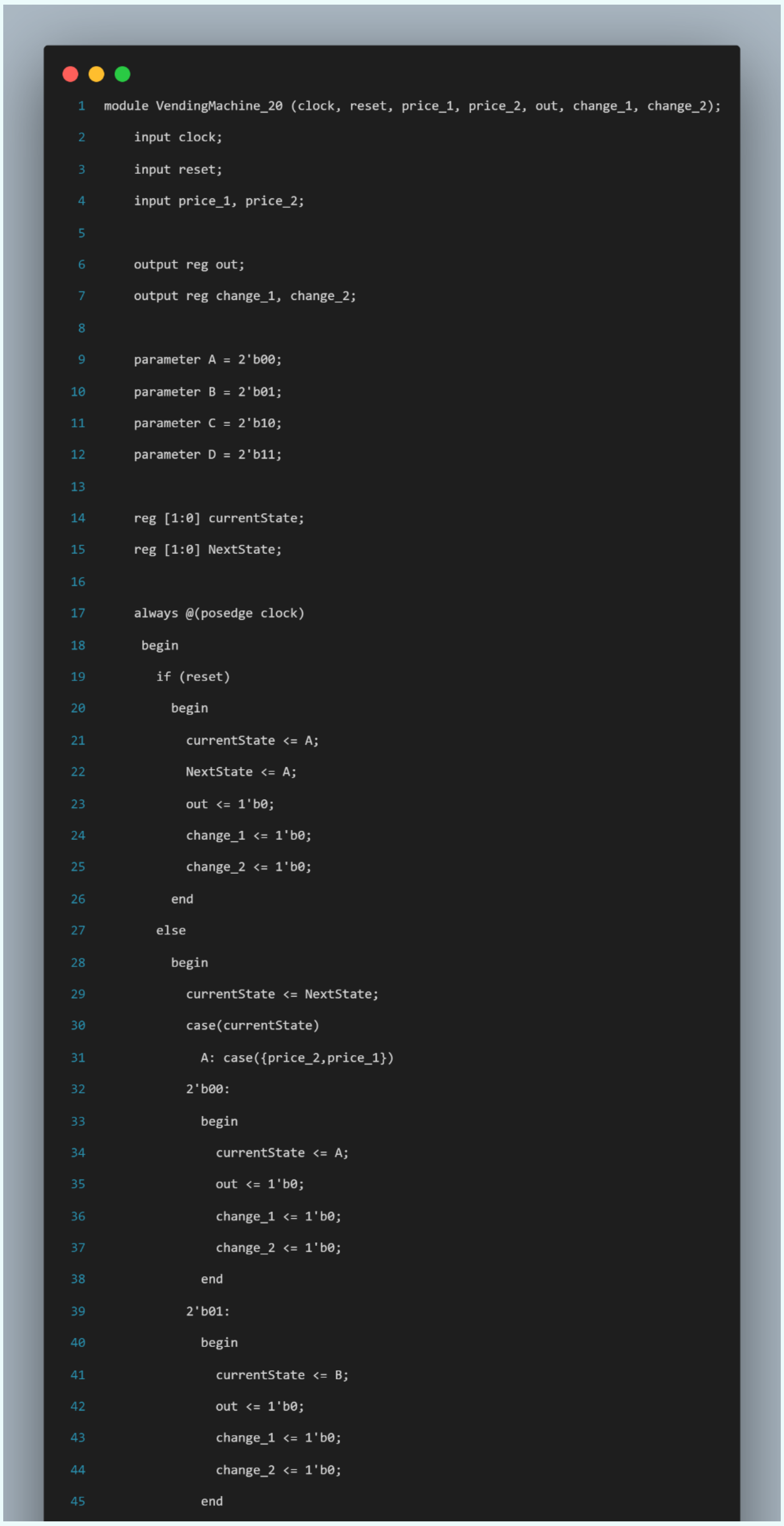




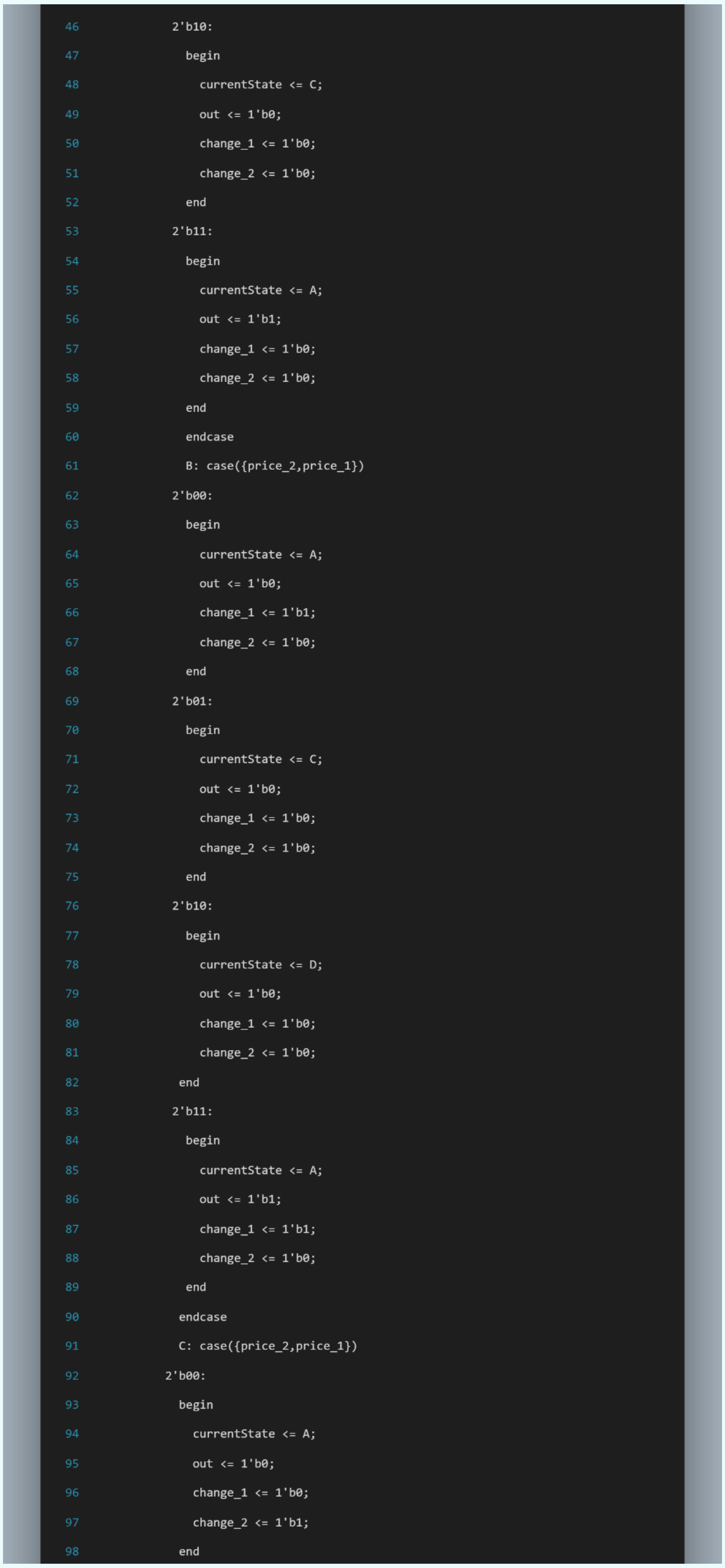
Implementation

Implementation the system using Verilog HDL:

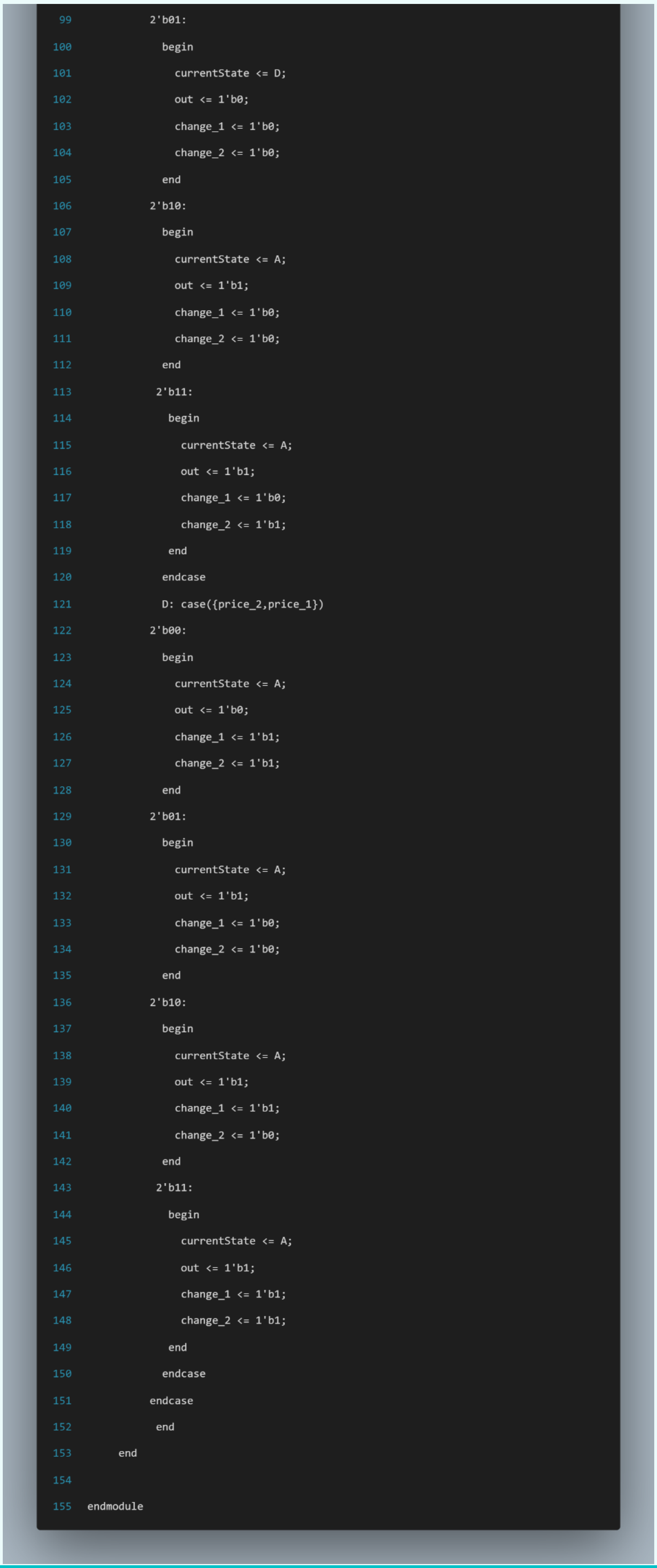
20:



```
module VendingMachine_20 (clock, reset, price_1, price_2, out, change_1, change_2);
    input clock;
    input reset;
    input price_1, price_2;

    output reg out;
    output reg change_1, change_2;

    parameter A = 2'b00;
    parameter B = 2'b01;
    parameter C = 2'b10;
    parameter D = 2'b11;

    reg [1:0] currentState;
    reg [1:0] NextState;

    always @(posedge clock)
     begin
       if (reset)
         begin
           currentState <= A;
           NextState <= A;
           out <= 1'b0;
           change_1 <= 1'b0;
           change_2 <= 1'b0;
         end
       else
         begin
           currentState <= NextState;
           case(currentState)
             A: case({price_2,price_1})
           2'b00:
             begin
               currentState <= A;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b01:
             begin
               currentState <= B;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
```

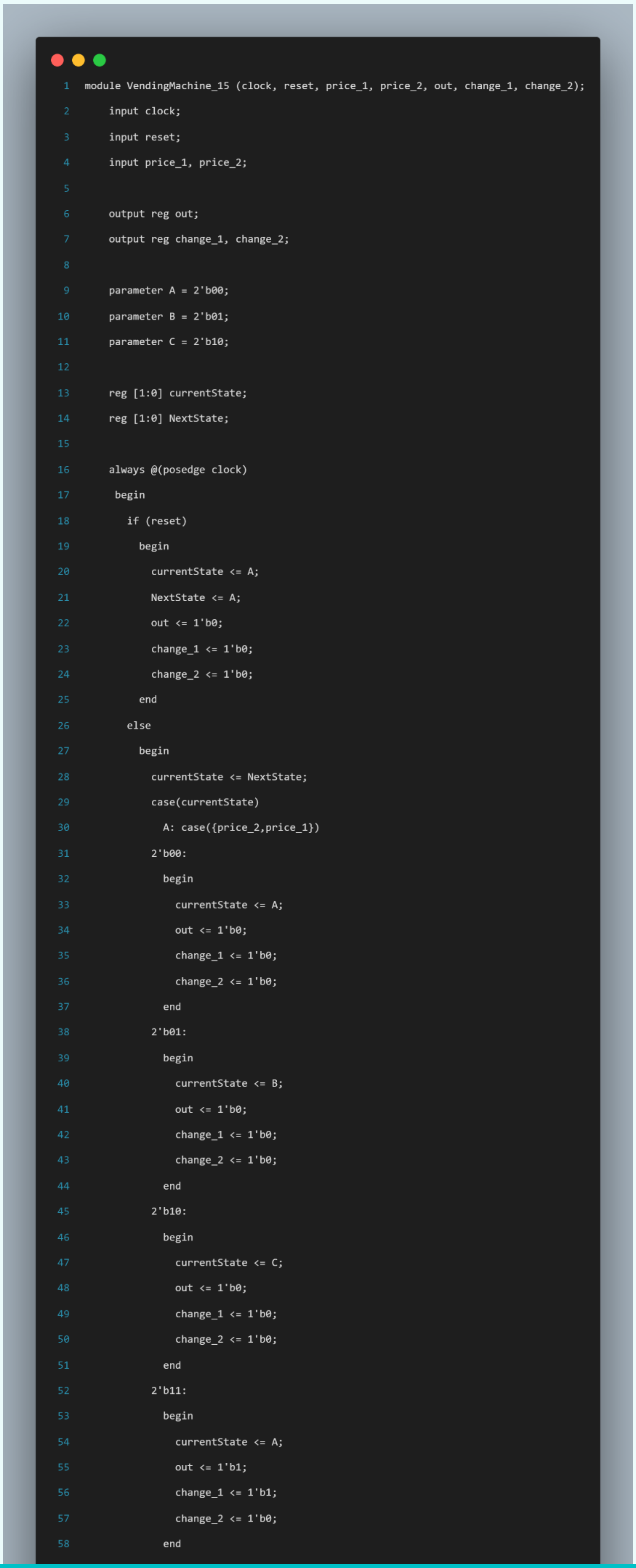


```
          2'b10:
            begin
              currentState <= C;
              out <= 1'b0;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
          2'b11:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
            endcase
            B: case({price_2,price_1})
          2'b00:
            begin
              currentState <= A;
              out <= 1'b0;
              change_1 <= 1'b1;
              change_2 <= 1'b0;
            end
          2'b01:
            begin
              currentState <= C;
              out <= 1'b0;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
          2'b10:
            begin
              currentState <= D;
              out <= 1'b0;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
           end
          2'b11:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b1;
              change_2 <= 1'b0;
            end
           endcase
           C: case({price_2,price_1})
         2'b00:
           begin
             currentState <= A;
             out <= 1'b0;
             change_1 <= 1'b0;
             change_2 <= 1'b1;
           end
```

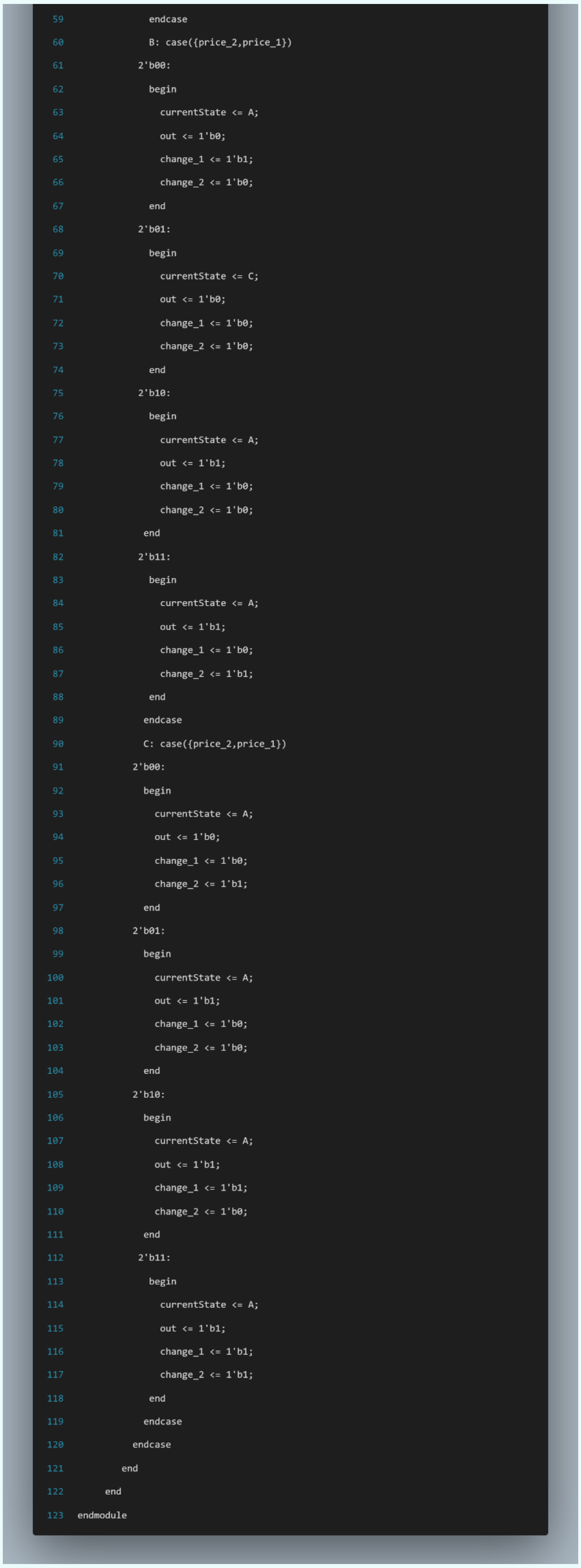


```
          2'b01:
            begin
              currentState <= D;
              out <= 1'b0;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
          2'b10:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b0;
               change_2 <= 1'b1;
             end
            endcase
            D: case({price_2,price_1})
          2'b00:
            begin
              currentState <= A;
              out <= 1'b0;
              change_1 <= 1'b1;
              change_2 <= 1'b1;
            end
          2'b01:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
          2'b10:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b1;
              change_2 <= 1'b0;
            end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b1;
               change_2 <= 1'b1;
             end
            endcase
          endcase
           end
     end

endmodule
```

15:

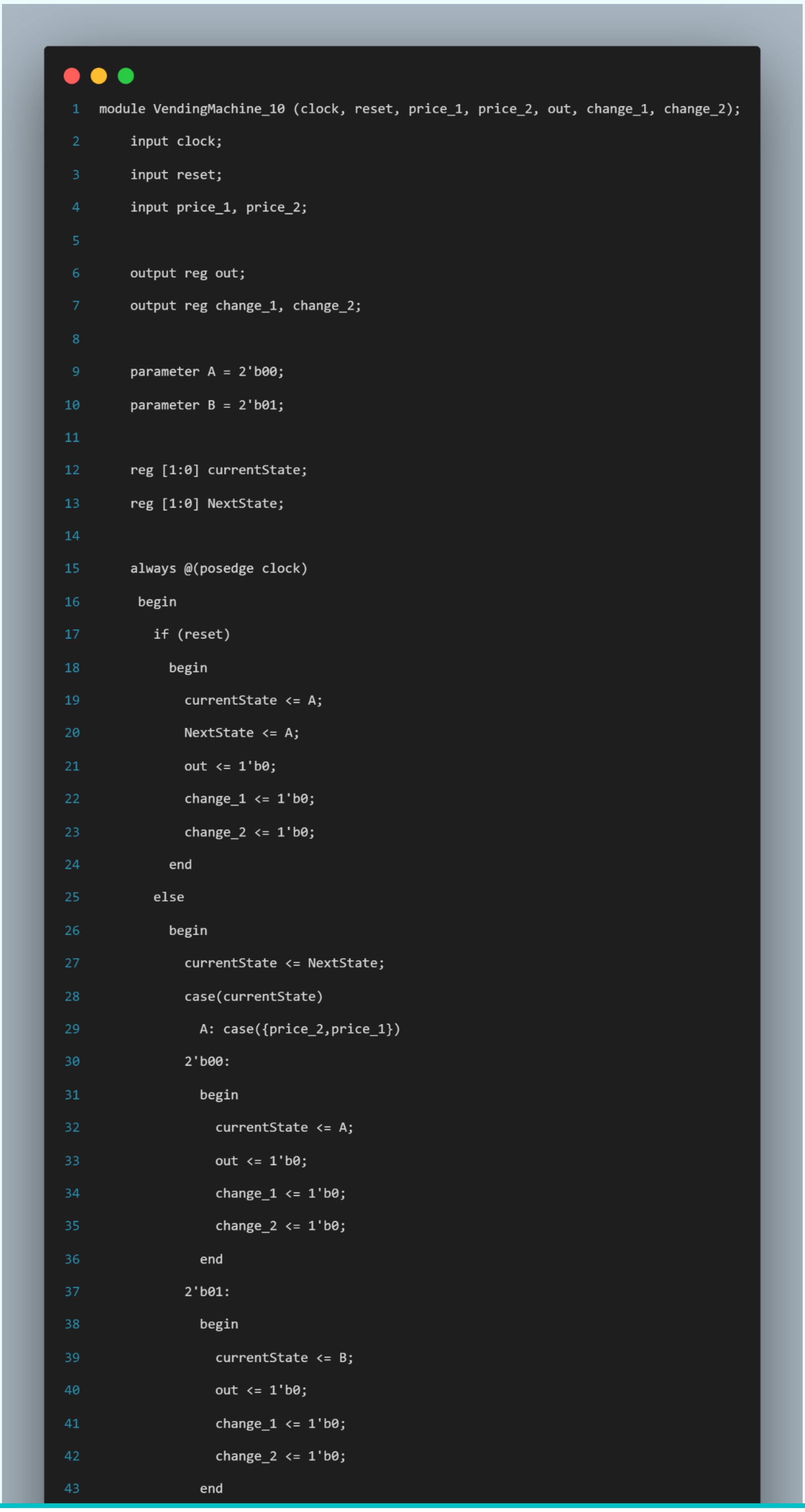


```
module VendingMachine_15 (clock, reset, price_1, price_2, out, change_1, change_2);
    input clock;
    input reset;
    input price_1, price_2;

    output reg out;
    output reg change_1, change_2;

    parameter A = 2'b00;
    parameter B = 2'b01;
    parameter C = 2'b10;

    reg [1:0] currentState;
    reg [1:0] NextState;

    always @(posedge clock)
     begin
       if (reset)
         begin
           currentState <= A;
           NextState <= A;
           out <= 1'b0;
           change_1 <= 1'b0;
           change_2 <= 1'b0;
         end
       else
         begin
           currentState <= NextState;
           case(currentState)
             A: case({price_2,price_1})
           2'b00:
             begin
               currentState <= A;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b01:
             begin
               currentState <= B;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b10:
             begin
               currentState <= C;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b1;
               change_2 <= 1'b0;
             end
```

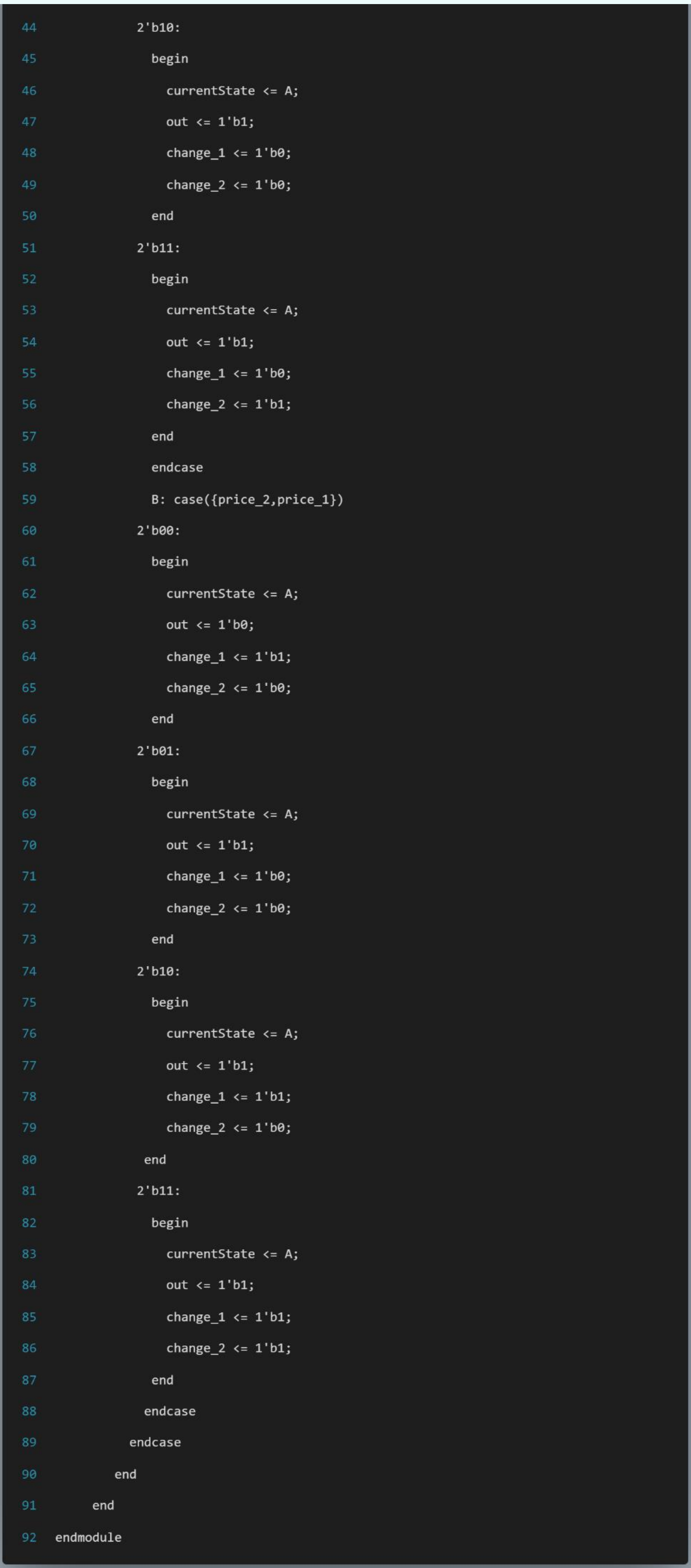


```
             endcase
             B: case({price_2,price_1})
           2'b00:
             begin
               currentState <= A;
               out <= 1'b0;
               change_1 <= 1'b1;
               change_2 <= 1'b0;
             end
           2'b01:
             begin
               currentState <= C;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b10:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
            end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b0;
               change_2 <= 1'b1;
             end
            endcase
            C: case({price_2,price_1})
          2'b00:
            begin
              currentState <= A;
              out <= 1'b0;
              change_1 <= 1'b0;
              change_2 <= 1'b1;
            end
          2'b01:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b0;
              change_2 <= 1'b0;
            end
          2'b10:
            begin
              currentState <= A;
              out <= 1'b1;
              change_1 <= 1'b1;
              change_2 <= 1'b0;
            end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b1;
               change_2 <= 1'b1;
             end
            endcase
          endcase
        end
     end
endmodule
```

10:

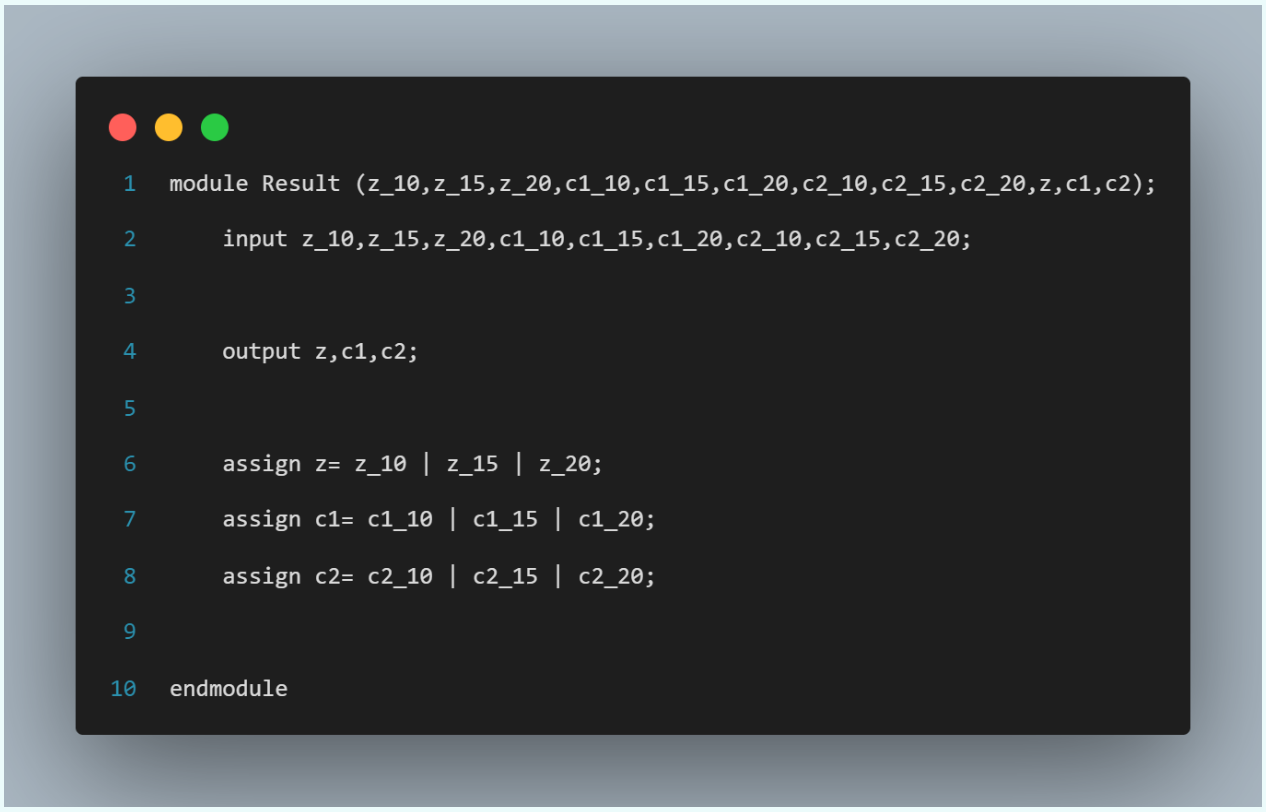


```
module VendingMachine_10 (clock, reset, price_1, price_2, out, change_1, change_2);
    input clock;
    input reset;
    input price_1, price_2;

    output reg out;
    output reg change_1, change_2;

    parameter A = 2'b00;
    parameter B = 2'b01;

    reg [1:0] currentState;
    reg [1:0] NextState;

    always @(posedge clock)
     begin
       if (reset)
         begin
           currentState <= A;
           NextState <= A;
           out <= 1'b0;
           change_1 <= 1'b0;
           change_2 <= 1'b0;
         end
       else
         begin
           currentState <= NextState;
           case(currentState)
             A: case({price_2,price_1})
           2'b00:
             begin
               currentState <= A;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b01:
             begin
               currentState <= B;
               out <= 1'b0;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
```



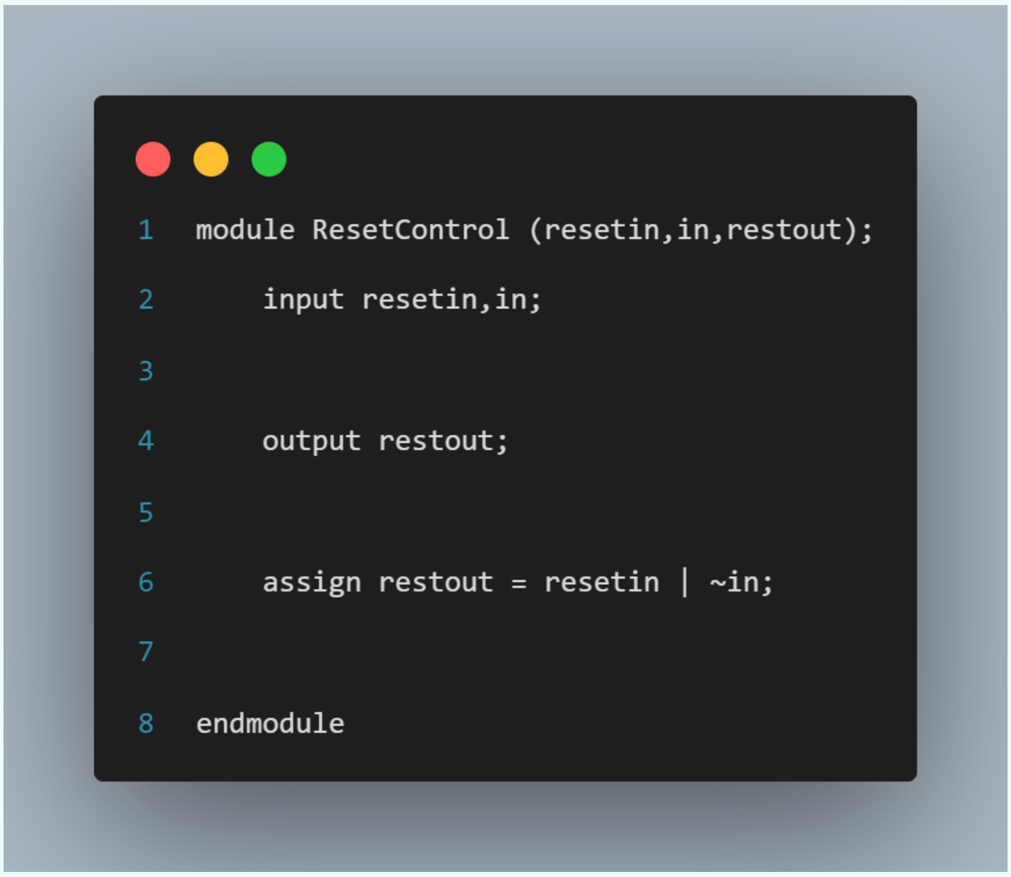
```
           2'b10:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b0;
               change_2 <= 1'b1;
             end
             endcase
             B: case({price_2,price_1})
           2'b00:
             begin
               currentState <= A;
               out <= 1'b0;
               change_1 <= 1'b1;
               change_2 <= 1'b0;
             end
           2'b01:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b0;
               change_2 <= 1'b0;
             end
           2'b10:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b1;
               change_2 <= 1'b0;
            end
           2'b11:
             begin
               currentState <= A;
               out <= 1'b1;
               change_1 <= 1'b1;
               change_2 <= 1'b1;
             end
            endcase
          endcase
        end
     end
endmodule
```

Result:



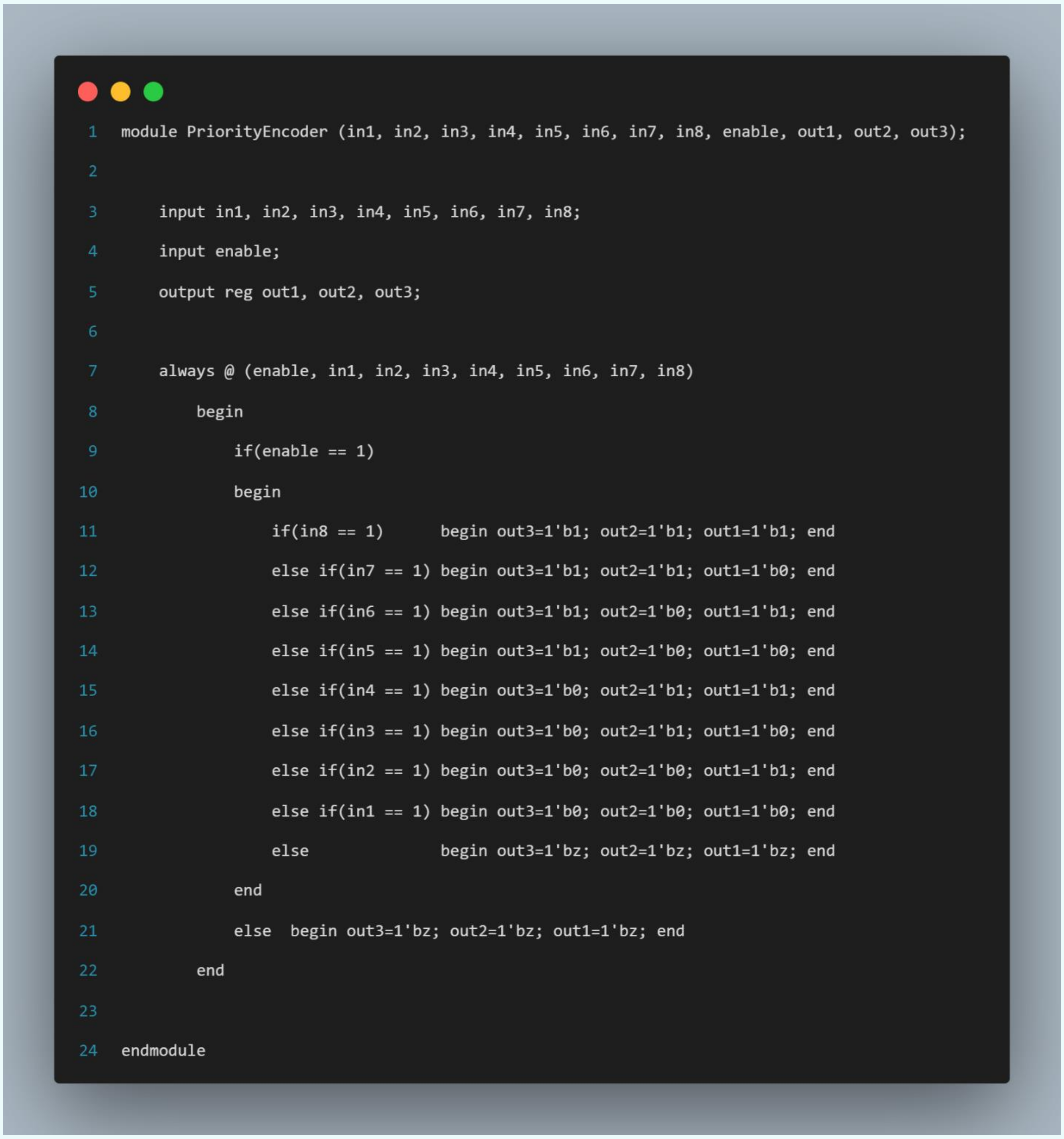
```
module Result (z_10,z_15,z_20,c1_10,c1_15,c1_20,c2_10,c2_15,c2_20,z,c1,c2);
    input z_10,z_15,z_20,c1_10,c1_15,c1_20,c2_10,c2_15,c2_20;

    output z,c1,c2;

    assign z= z_10 | z_15 | z_20;
    assign c1= c1_10 | c1_15 | c1_20;
    assign c2= c2_10 | c2_15 | c2_20;

endmodule
```

Reset Control:



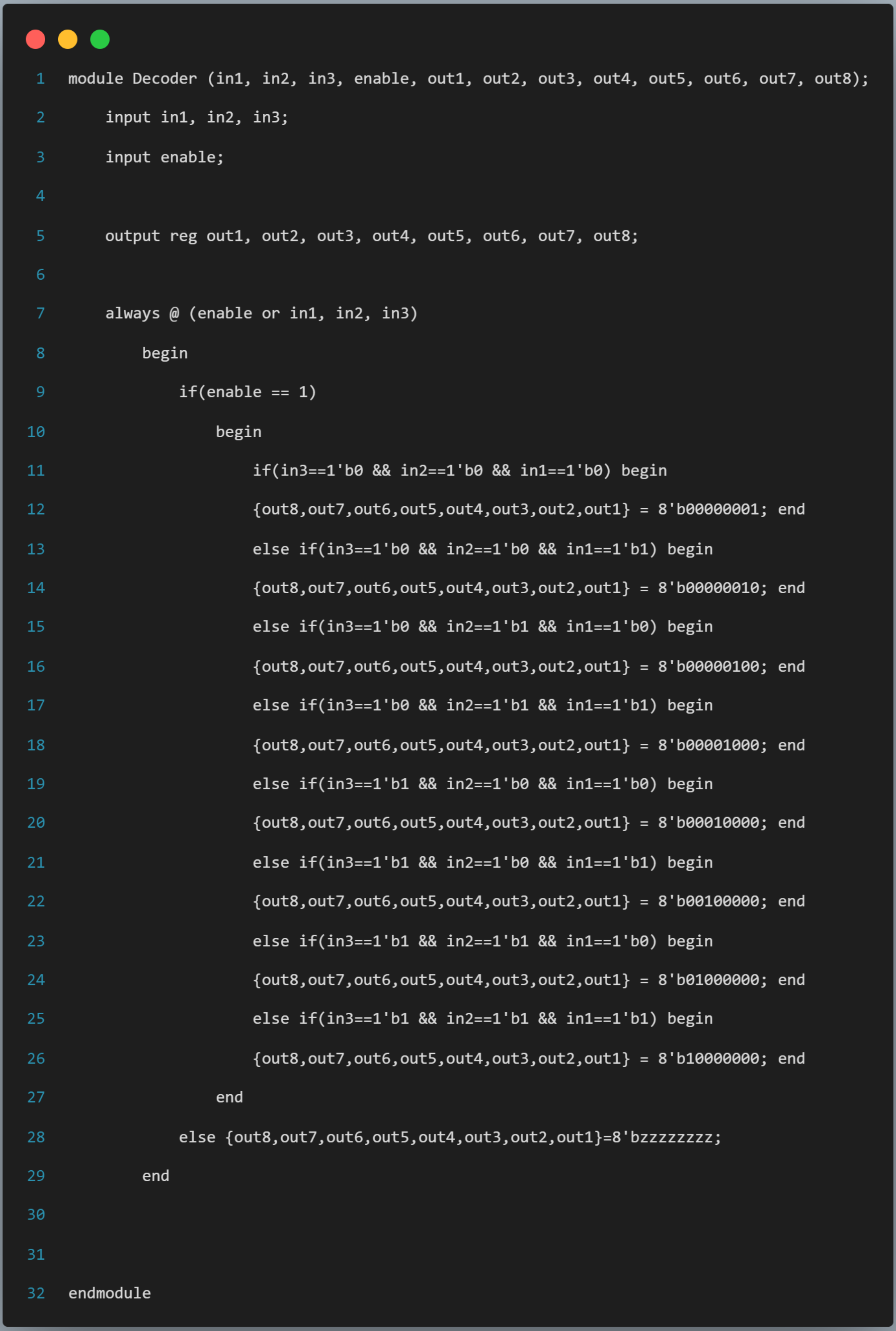
```
module ResetControl (resetin,in,restout);
    input resetin,in;

    output restout;

    assign restout = resetin | ~in;

endmodule
```

Priority Encoder:



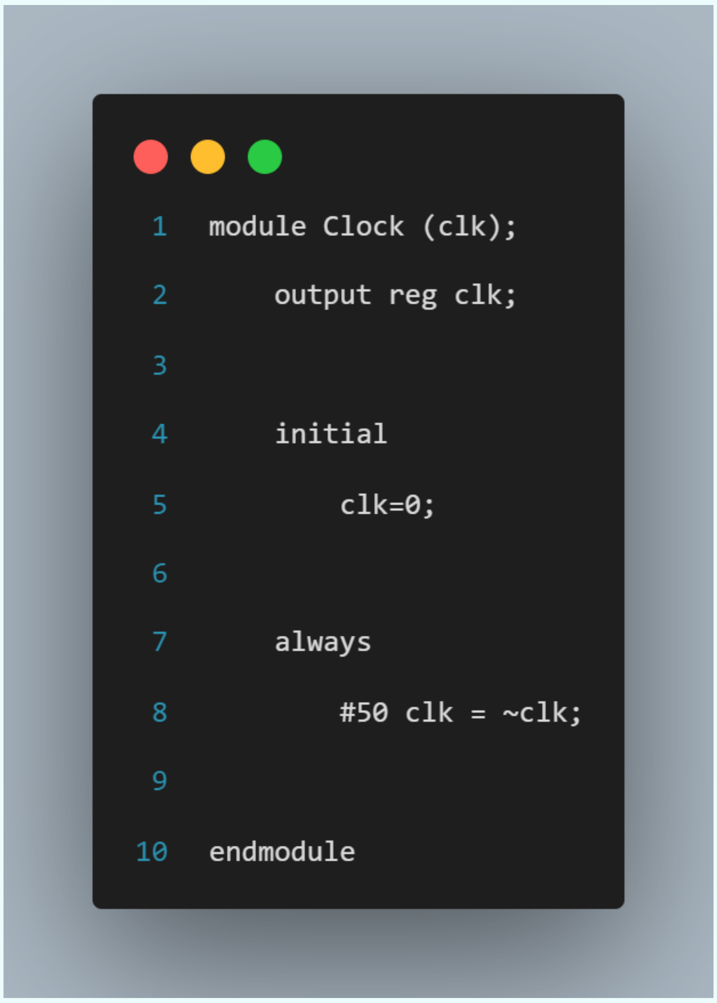
```
module PriorityEncoder (in1, in2, in3, in4, in5, in6, in7, in8, enable, out1, out2, out3);

    input in1, in2, in3, in4, in5, in6, in7, in8;
    input enable;
    output reg out1, out2, out3;

    always @ (enable, in1, in2, in3, in4, in5, in6, in7, in8)
        begin
            if(enable == 1)
            begin
                if(in8 == 1)      begin out3=1'b1; out2=1'b1; out1=1'b1; end
                else if(in7 == 1) begin out3=1'b1; out2=1'b1; out1=1'b0; end
                else if(in6 == 1) begin out3=1'b1; out2=1'b0; out1=1'b1; end
                else if(in5 == 1) begin out3=1'b1; out2=1'b0; out1=1'b0; end
                else if(in4 == 1) begin out3=1'b0; out2=1'b1; out1=1'b1; end
                else if(in3 == 1) begin out3=1'b0; out2=1'b1; out1=1'b0; end
                else if(in2 == 1) begin out3=1'b0; out2=1'b0; out1=1'b1; end
                else if(in1 == 1) begin out3=1'b0; out2=1'b0; out1=1'b0; end
                else              begin out3=1'bz; out2=1'bz; out1=1'bz; end
            end
            else  begin out3=1'bz; out2=1'bz; out1=1'bz; end
        end

endmodule
```

Decoder:



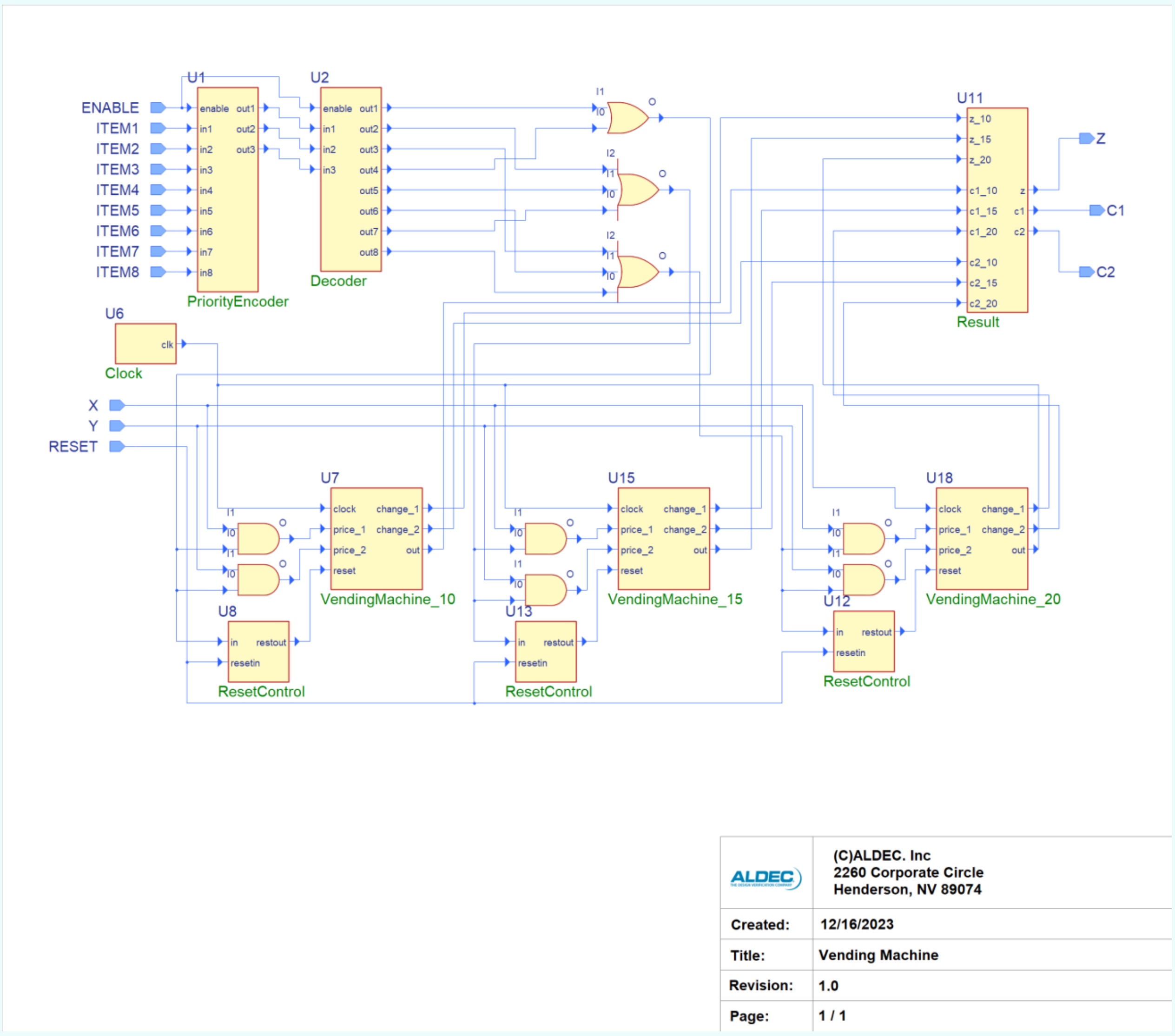
```
module Decoder (in1, in2, in3, enable, out1, out2, out3, out4, out5, out6, out7, out8);
    input in1, in2, in3;
    input enable;

    output reg out1, out2, out3, out4, out5, out6, out7, out8;

    always @ (enable or in1, in2, in3)
        begin
            if(enable == 1)
                begin
                    if(in3==1'b0 && in2==1'b0 && in1==1'b0) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b00000001; end
                    else if(in3==1'b0 && in2==1'b0 && in1==1'b1) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b00000010; end
                    else if(in3==1'b0 && in2==1'b1 && in1==1'b0) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b00000100; end
                    else if(in3==1'b0 && in2==1'b1 && in1==1'b1) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b00001000; end
                    else if(in3==1'b1 && in2==1'b0 && in1==1'b0) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b00010000; end
                    else if(in3==1'b1 && in2==1'b0 && in1==1'b1) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b00100000; end
                    else if(in3==1'b1 && in2==1'b1 && in1==1'b0) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b01000000; end
                    else if(in3==1'b1 && in2==1'b1 && in1==1'b1) begin
                    {out8,out7,out6,out5,out4,out3,out2,out1} = 8'b10000000; end
                end
            else {out8,out7,out6,out5,out4,out3,out2,out1}=8'bzzzzzzzz;
        end


endmodule
```

Clock:



```verilog
module Clock (clk);
    output reg clk;

    initial
        clk=0;

    always
        #50 clk = ~clk;

endmodule
```

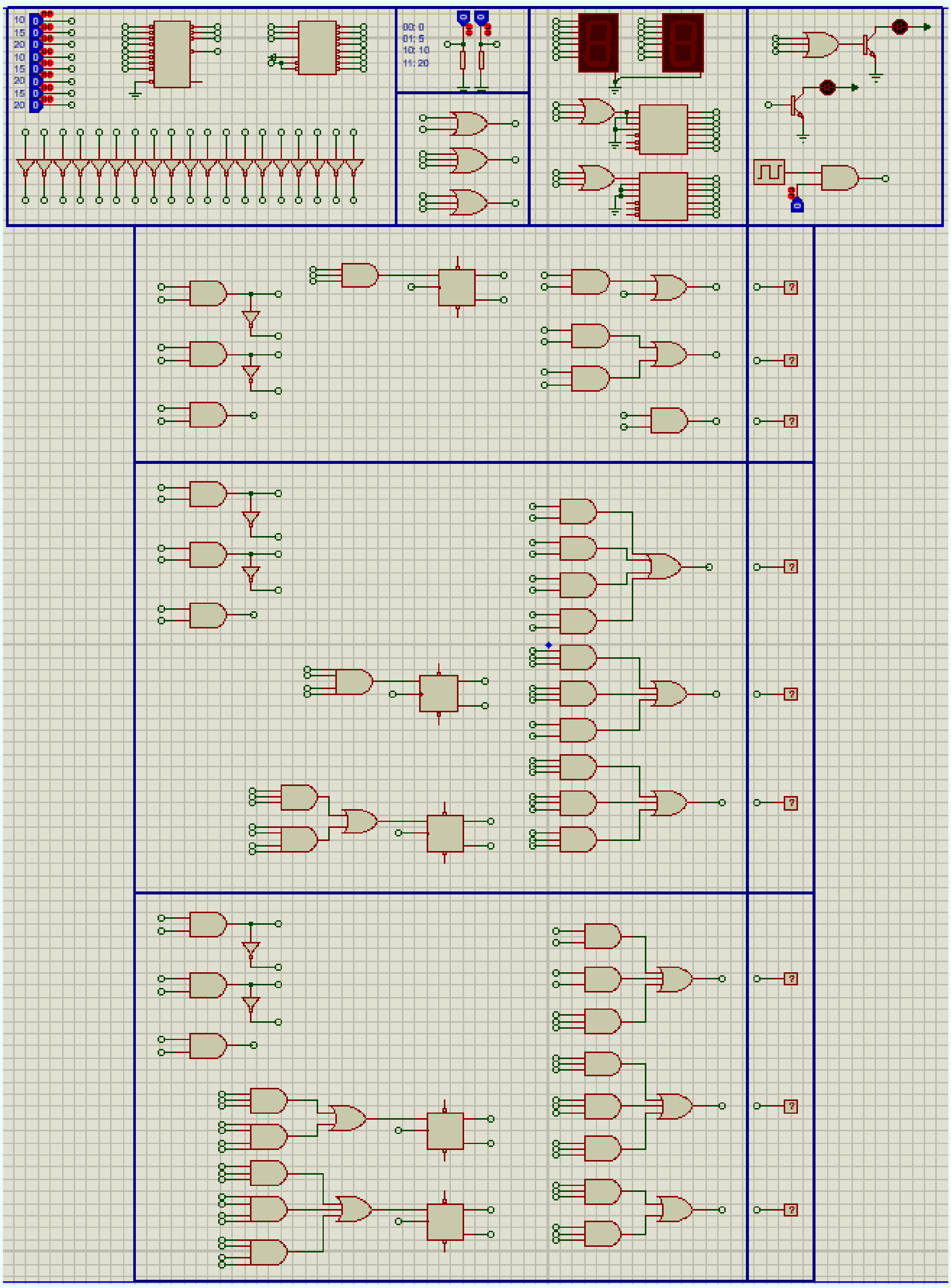
Schematic diagram using Active HDL



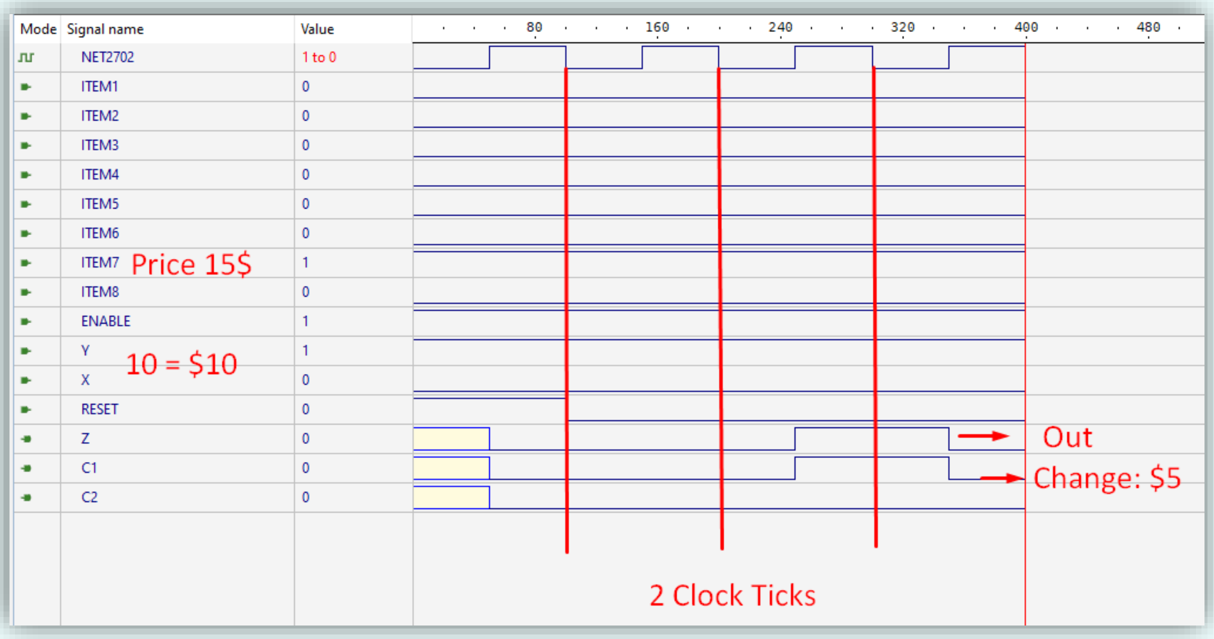
(C)ALDEC. Inc
2260 Corporate Circle
Henderson, NV 89074

Created:	12/16/2023
Title:	Vending Machine
Revision:	1.0
Page:	1 / 1

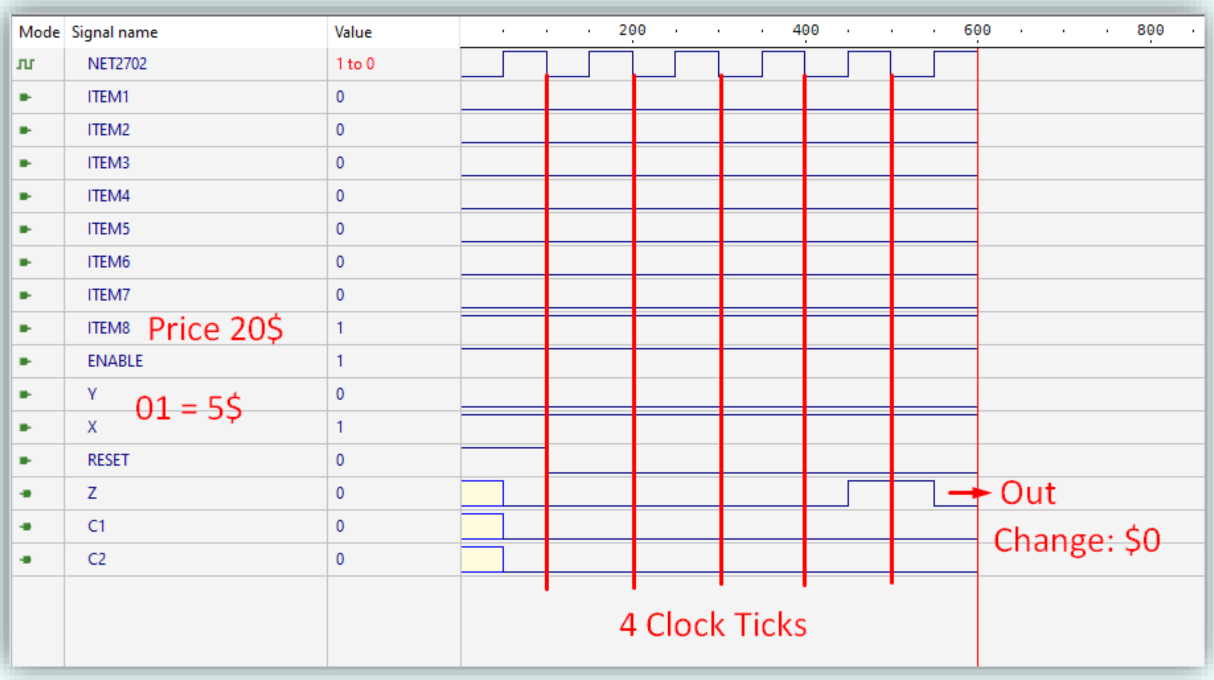
Schematic diagram using Proteus



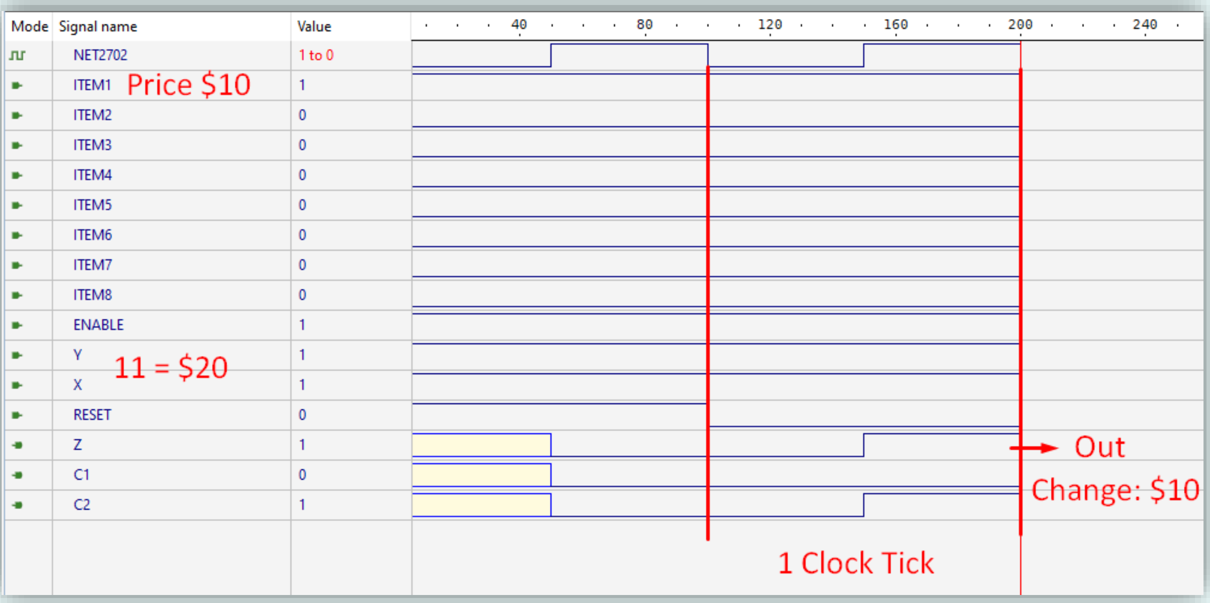
The design simulation



Mode	Signal name	Value
	NET2702	1 to 0
	ITEM1	0
	ITEM2	0
	ITEM3	0
	ITEM4	0
	ITEM5	0
	ITEM6	0
	ITEM7	1
	ITEM8	0
	ENABLE	1
	Y	1
	X	0
	RESET	0
	Z	0
	C1	0
	C2	0



Mode	Signal name	Value
	NET2702	1 to 0
	ITEM1	0
	ITEM2	0
	ITEM3	0
	ITEM4	0
	ITEM5	0
	ITEM6	0
	ITEM7	0
	ITEM8	1
	ENABLE	1
	Y	0
	X	1
	RESET	0
	Z	0
	C1	0
	C2	0



Mode	Signal name	Value
	NET2702	1 to 0
	ITEM1	1
	ITEM2	0
	ITEM3	0
	ITEM4	0
	ITEM5	0
	ITEM6	0
	ITEM7	0
	ITEM8	0
	ENABLE	1
	Y	1
	X	1
	RESET	0
	Z	1
	C1	0
	C2	1

THANK YOU