




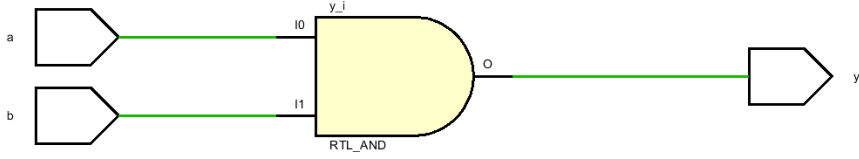
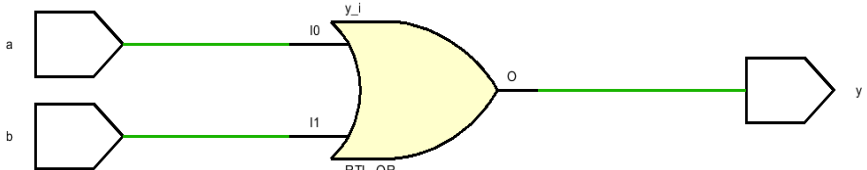


## MSPA-01 Evaluation

Name : <b>SHIVAM NITIN SALVE</b>	Roll No. <b>56</b>
Branch : <b>VLSI</b>	Sem <b>5</b>
Batch: <b>A4</b>	Reg no: <b>23070029</b>

### Classroom Examples ( 5 Marks )

Code	<pre> 22  //SHIVAM SALVE 23  module and_gate(y,a,b); 24     output wire y; 25     input wire a,b; 26      assign y=a&amp;b; 27  endmodule 28  </pre>
RTL View	

Code	<pre> //SHIVAM SALVE module or_gate(y,a,b); output y; input a,b; assign y=a b; endmodule </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module ex_or_gate(Y,a,b); 22     input a,b; 23     output Y; 24     wire t1,t2,t3,t4; 25     assign t1=~a; 26     assign t2=~b; 27     assign t3=a&amp;t2; 28     assign t4=b&amp;t1; 29     assign Y=t3 t4; 30 endmodule 31 </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module ckt_1(y,a,b,c); 22     output y; 23     input a,b,c; 24     wire t1,t2; 25     assign t1=a&amp;b; 26     assign t2=b c; 27     assign y=t1&amp;t2; 28 endmodule </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module half_adder(sum,carry,a,b); 22     output sum,carry; 23     input a,b; 24     assign sum=a^b; 25     assign carry=a&amp;b; 26 endmodule 27 </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module fullAdder(sum,cout,a,b,cin); 22     input a,b,cin; 23     output sum,cout; 24     wire t1,t2,t3; 25     assign t1=a^b; 26     assign t2=a&amp;b; 27     assign sum=t1^cin; 28     assign t3=t1&amp;cin; 29     assign cout=t2 t3; 30 endmodule </pre>
RTL View	

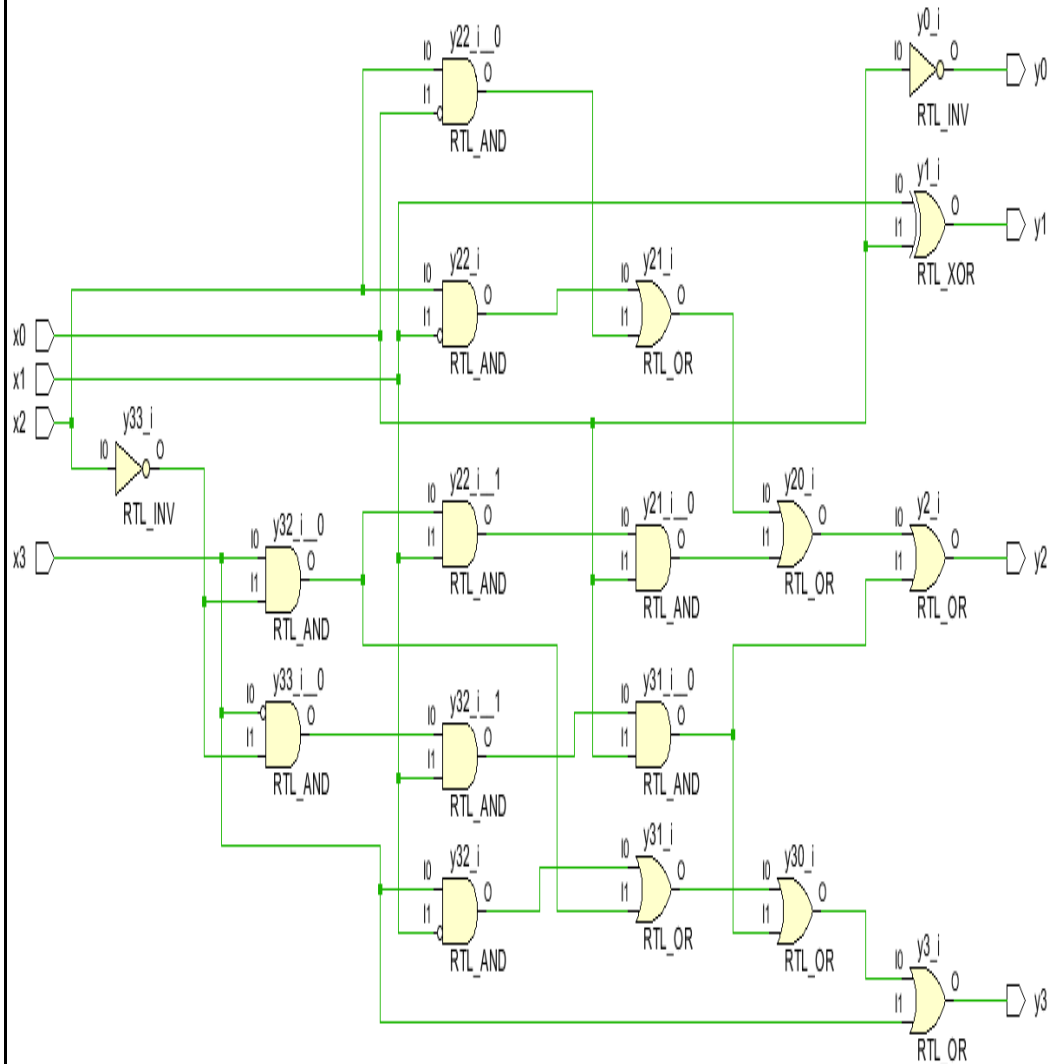
Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module halfSub(d,bo,a,b); 22 input a,b; 23 output d,bo; 24 wire t1; 25 assign t1=~a; 26 assign d=a^b; 27 assign bo=t1&amp;b; 28 endmodule </pre>
RTL View	




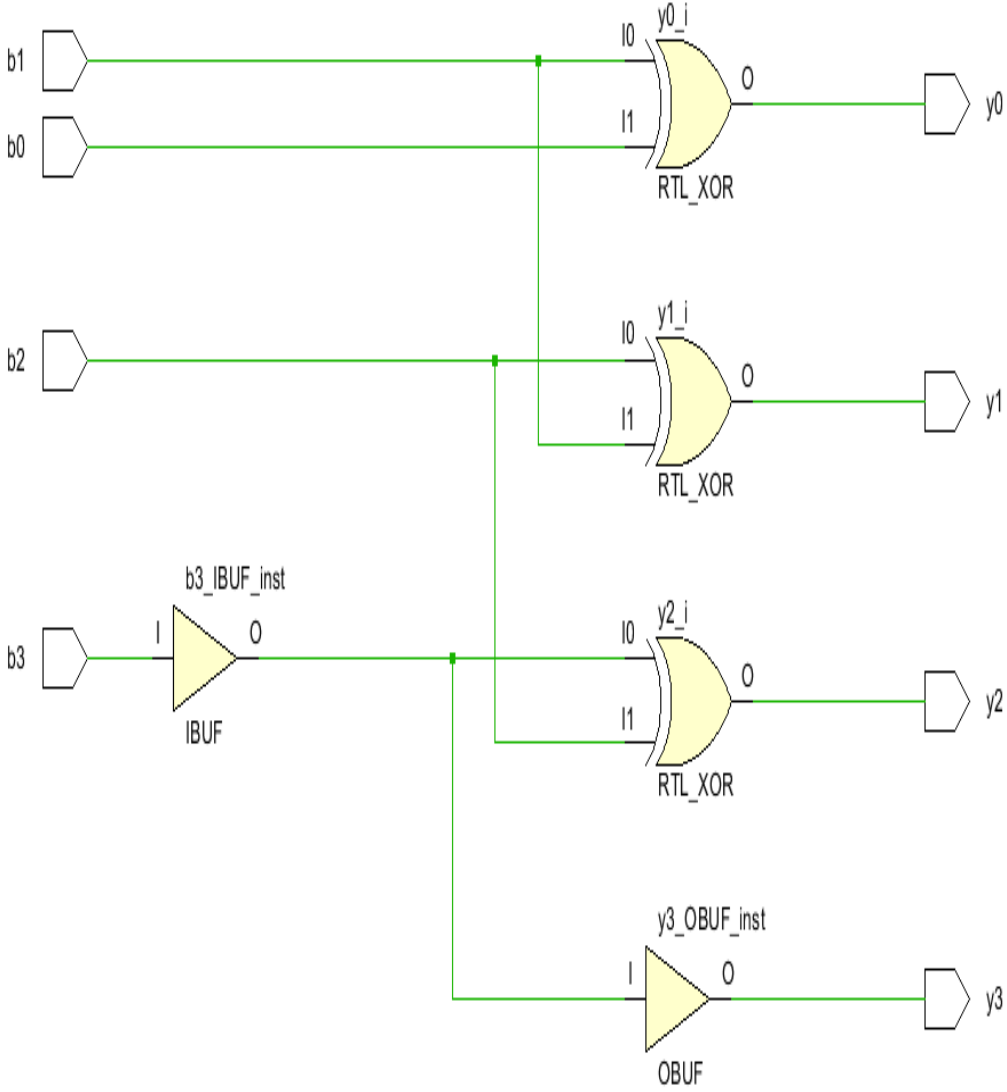
Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module fullSub(d,bo,a,b,cin); 22 input a,b,cin; 23 output d,bo; 24 wire t1,t2,t3,t4,t5; 25 assign t1=~a; 26 assign t2=~cin; 27 assign t3=a^b; 28 assign d=t3^cin; 29 assign t4=b&amp;t1; 30 assign t5=t2&amp;t3; 31 assign bo=t4 t5; 32 endmodule </pre>
RTL View	

Code

```
20 //SHIVAM SALVE VLSI 56
21 module incremental_code(y3,y2,y1,y0,x3,x2,x1,x0);
22     output y3,y2,y1,y0;
23     input x3,x2,x1,x0;
24     assign y3=(x3&~ x1)|(x3&~x2)|((~x3)&(~x2)&x1&x0)|(x3);
25     assign y2=(x2&(~x1))|(x2&~x0)|((x3)&(~x2)&x1&x0)|((~x3)&(~x2)&x1&x0);
26     assign y1=x1^x0;
27     assign y0=~x0;
28 endmodule
```

RTL  
View



Code	<pre> 20  // SHIVAM SALVE VLSI 56 21  module gray_code(y3,y2,y1,y0,b3,b2,b1,b0); 22     output y3,y2,y1,y0; 23     input b3,b2,b1,b0; 24     assign y3=b3; 25     assign y2=b3^b2; 26     assign y1=b2^b1; 27     assign y0=b1^b0; 28  endmodule </pre>
RTL View	 <p>The RTL schematic diagram illustrates the implementation of the gray_code module. It features four input signals: b1, b0, b2, and b3. The outputs are y0, y1, y2, and y3. The logic is as follows:</p> <ul style="list-style-type: none"> <li>Inputs b1 and b0 are connected to an RTL_XOR gate (labeled y0_i) to produce output y0.</li> <li>The output of the first XOR gate (y0) and input b2 are connected to a second RTL_XOR gate (labeled y1_i) to produce output y1.</li> <li>Input b3 is connected to an IBUF block (labeled b3_IBUF_inst) to produce an internal signal.</li> <li>The internal signal from the IBUF block and the output of the second XOR gate (y1) are connected to a third RTL_XOR gate (labeled y2_i) to produce output y2.</li> <li>The internal signal from the IBUF block is also connected to an OBUF block (labeled y3_OBUF_inst) to produce output y3.</li> </ul>

Code	<pre> 20 // SHIVAM SALVE VLSI 56 21 module mux_2X1(y,a,b,s); 22     input a,b,s; 23     output y; 24     assign y=((~s)&amp;a) (s&amp;b); 25 endmodule </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module nand_gate (a,b,y); 22     input a, b; 23     output y; 24     assign y = ~(a &amp; b); 25 endmodule </pre>
RTL View	

## EXTRA EXAMPLES TAKEN IN CLASS

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module test_1(y); 22     output wire y; 23     assign y=1'b1; 24 endmodule 25 </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module test_3(y,a); 22     output y; 23     input a; 24     assign y=~a; 25 endmodule </pre>
RTL View	

Code	<pre> 20 //SHIVAM SALVE VLSI 56 21 module test_4(p,q,r,s,a,b,c); 22     output p,q,r,s; 23     input a,b,c; 24     assign p=a; 25     assign q=c; 26     assign r=b; 27     assign s=c; 28 endmodule </pre>
RTL View	



### Design Any Combinational Circuit ( 5 Marks )

Code	<pre> 20 : //SHIVAM SALVE VLSI 56 21 : //DECODER 2TO4 22 : module Dec2to4(a,b,y0,y1,y2,y3); 23 : output y0,y1,y2,y3; 24 : input a,b; 25 : assign y0 = (~a)&amp;(~b); 26 : assign y1 = (~a)&amp;(b); 27 : assign y2 = (a)&amp;(~b); 28 : assign y3 = (a)&amp;(b); 29 : endmodule </pre>
RTL View	<p>The RTL View diagram illustrates the implementation of the 2-to-4 decoder circuit. It features two input signals, 'a' and 'b', each connected to an inverter (RTL_INV). The output of the first inverter (NOT a) is connected to the 'I1' input of the first two AND gates (RTL_AND). The output of the second inverter (NOT b) is connected to the 'I1' input of the last two AND gates (RTL_AND). The output of the first AND gate is labeled y0, the second is y1, the third is y2, and the fourth is y3. The 'I0' input of all four AND gates is connected to the input signal 'a'.</p>

## Design combinational design project ( 5 Marks )

Logic	<p>Vending Machine with Change</p> <p>Accepts <b>3 types of coins/currency notes</b>: 10, 20, 50.</p> <p>Product costs <b>40</b>.</p> <p>FSM keeps track of balance using states:</p> <ul style="list-style-type: none"> <li>• S10 = balance 10</li> <li>• S20 = balance 20</li> <li>• S30 = balance 30</li> <li>• S40 = balance 40 → Product given (Z=1, no change)</li> <li>• S50, S60, S70, S80 → Product given (Z=1, with change)</li> </ul> <p>After product delivery, FSM goes back to <b>Sin</b>.</p>
Gate Level diagram	<p>The diagram illustrates the gate-level implementation of the Vending Machine FSM. It features a state register (state reg[3:0]) that stores the current state. The register is connected to a series of 11 RTL_MUX blocks, each representing a state (next_state_i_0 to next_state_i_11). Each RTL_MUX block has inputs for coin values (10, 20, 50) and the current state. The outputs of the RTL_MUX blocks are connected to a final RTL_MUX block, which produces the next state (next_state_i_11). The final RTL_MUX block also produces the output Z (change given) and change_given_i. The diagram includes a clock (clk) and reset (reset) signal, and a constant value of 10.</p>

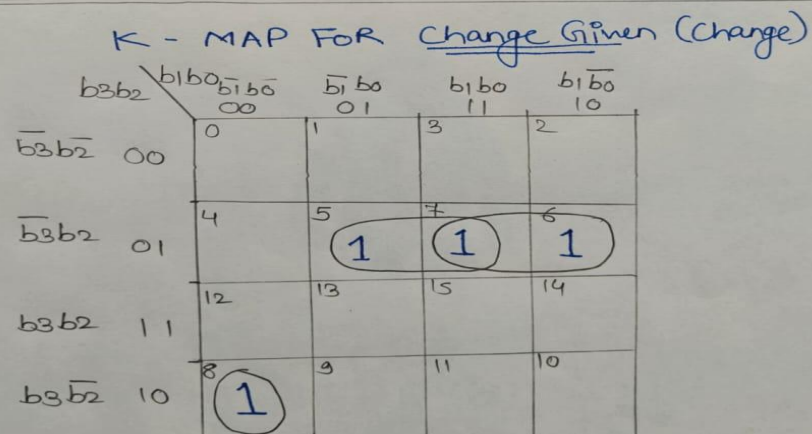
Truth  
Table

State	Balance	Z (Product)	Change
Sin	0	0	0
S10	10	0	0
S20	20	0	0
S30	30	0	0
S40	40	1	0
S50	50	1	1
S60	60	1	1
S70	70	1	1
S80	80	1	1

Kmap  
And  
Equation  
s



$$\text{eq:- } Z = b_2 \cdot \bar{b}_3 + b_3 \bar{b}_2 \cdot \bar{b}_1 \cdot \bar{b}_0$$



$$\begin{aligned} \text{eq:- } \text{Change} &= \bar{b}_3 \cdot b_2 \cdot \bar{b}_0 + \bar{b}_3 b_2 \cdot b_1 + b_3 \cdot \bar{b}_2 \cdot \bar{b}_1 \cdot \bar{b}_0 \\ \text{Change} &= \bar{b}_3 \cdot b_2 (b_0 + b_1) + b_3 \cdot \bar{b}_2 \cdot \bar{b}_1 \cdot \bar{b}_0 \end{aligned}$$

## Code

```

21 //Shivam Salve VLSI 56 MSPA 1 PROJECT
22 module Vending_machine(
23     input wire clk,
24     input wire reset,
25     input wire [1:0] coin,    // 2-bit coin input
26     output reg Z,             // Product given
27     output reg change_given   // Change return
28 );
29
30 parameter Sin = 4'b0000,    // Initial state
31             S10 = 4'b0001,
32             S20 = 4'b0010,
33             S30 = 4'b0011,
34             S40 = 4'b0100,
35             S50 = 4'b0101,
36             S60 = 4'b0110,
37             S70 = 4'b0111,
38             S80 = 4'b1000;
39
40 parameter ten = 2'b00,      // 10 Rs coin
41             twenty = 2'b01, // 20 Rs coin
42             fifty = 2'b10;  // 50 Rs coin
43
44 reg [3:0] state, next_state;
45
46 // ----- State Update -----
47 always @(posedge clk or posedge reset) begin
48     if (reset)
49         state <= Sin;
50     else
51         state <= next_state;
52 end
53
54 // ----- Next State Logic -----
55 always @(*) begin
56     case (state)
57     Sin: begin
58         if (coin == ten)      next_state = S10;
59         else if (coin == twenty) next_state = S20;
60         else if (coin == fifty) next_state = S50;
61         else next_state = Sin;
62     end
63
64     S10: begin
65         if (coin == ten)      next_state = S20;
66         else if (coin == twenty) next_state = S30;
67         else if (coin == fifty) next_state = S60;
68         else next_state = S10;
69     end
70
71     S20: begin
72         if (coin == ten)      next_state = S30;
73         else if (coin == twenty) next_state = S40;
74         else if (coin == fifty) next_state = S70;
75         else next_state = S20;
76     end
77
78     S30: begin
79         if (coin == ten)      next_state = S40;
80         else if (coin == twenty) next_state = S50;
81         else if (coin == fifty) next_state = S80;
82         else next_state = S30;
83     end
84
85     S40, S50, S60, S70, S80: next_state = Sin;

```

```

86 |
87 |         default: next_state = Sin;
88 |     endcase
89 | end
90 |
91 | // ----- Output Logic -----
92 | always @(*) begin
93 |     case (state)
94 |         Sin, S10, S20, S30: begin
95 |             Z = 0; change_given = 0;
96 |         end
97 |
98 |         S40: begin
99 |             Z = 1; change_given = 0; // Product delivered, no change
100 |        end
101 |        S50, S60, S70, S80: begin
102 |            Z = 1; change_given = 1; // Product delivered, change returned
103 |        end
104 |
105 |        default: begin
106 |            Z = 0; change_given = 0;
107 |        end
108 |    endcase
109 | end
110 |
111 | endmodule

```

Evaluation by faculty (Dr. P.P. Zode )

5 Marks	5 Marks	5 Marks	15 Marks