CS 223

Section – 6

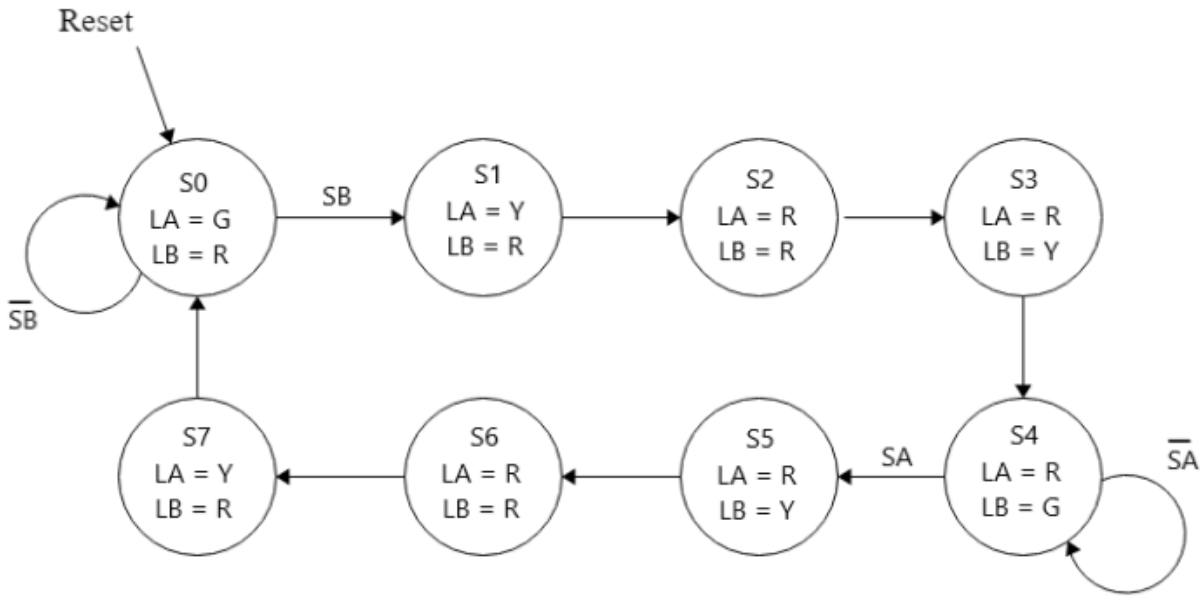
Lab – 04

Mert Fidan

22101734

26.11.2022

State Transition Diagram:



State Encoding

S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

Output Encoding

Red(R)	0	0
Yellow(Y)	0	1
Green(G)	1	0

State Transition Table:

Current State			Inputs		Next State		
S2	S1	S0	SA	SB	S2'	S1'	S0'
0	0	0	X	0	0	0	0
0	0	0	X	1	0	0	1
0	0	1	X	X	0	1	0
0	1	0	X	X	0	1	1
0	1	1	X	X	1	0	0
1	0	0	0	X	1	0	0
1	0	0	1	X	1	0	1
1	0	1	X	X	1	1	0
1	1	0	X	X	1	1	1
1	1	1	X	X	0	0	0

Output Table:

Current State			Outputs			
S2	S1	S0	LA1	LA0	LB1	LB0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	1
1	0	0	0	0	1	0
1	0	1	0	0	0	1
1	1	0	0	0	0	0
1	1	1	0	1	0	0

Equations:

$$S'_2 = \bar{s}_2 s_1 s_0 + s_2 s_1 \overline{s_0} + s_2 \bar{s}_1$$

$$S'_1 = \bar{s}_1 s_0 + s_1 \overline{s_0}$$

$$S'_0 = s_1 \overline{s_0} + \bar{s}_2 \bar{s}_1 s_0$$

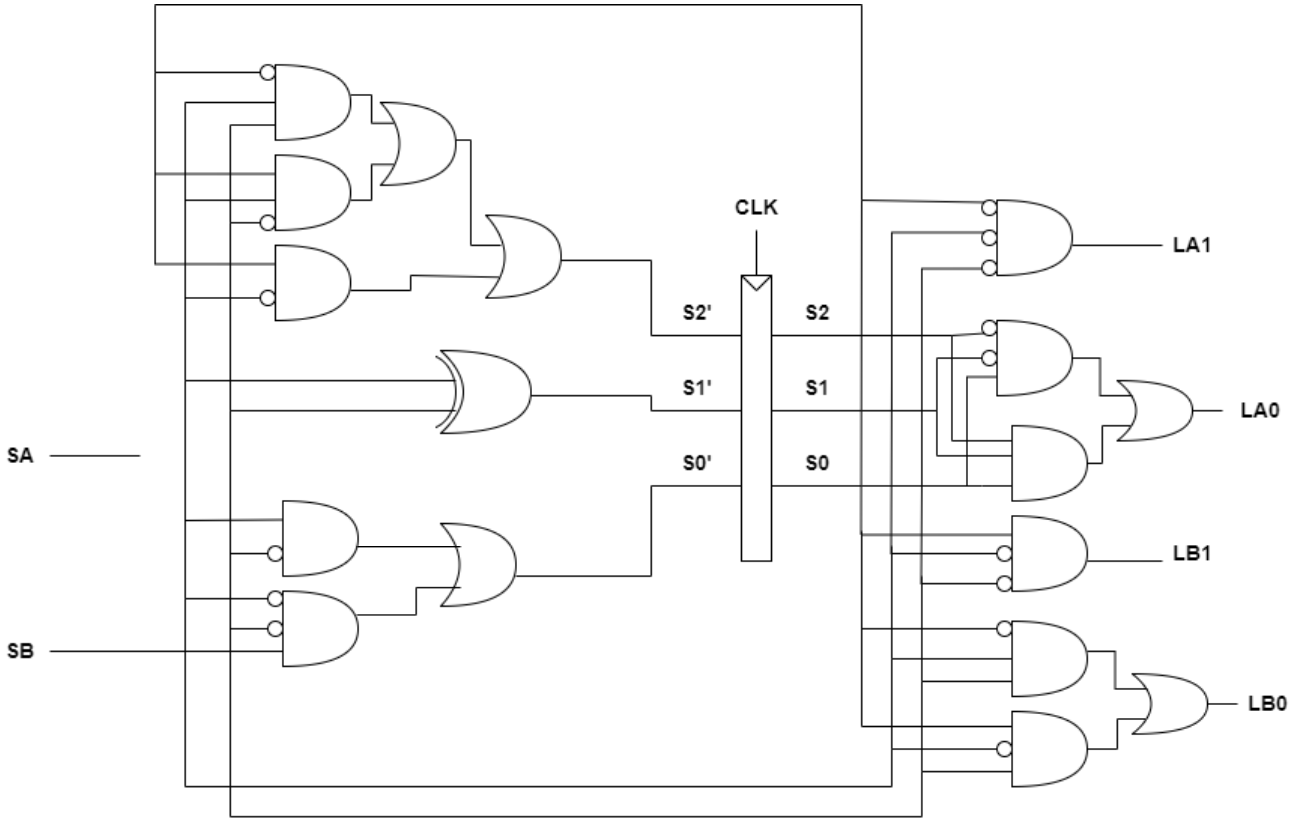
$$LA_1 = \bar{s}_2 \bar{s}_1 s_0$$

$$LA_2 = \bar{s}_2 \bar{s}_1 s_0 + s_2 s_1 s_0$$

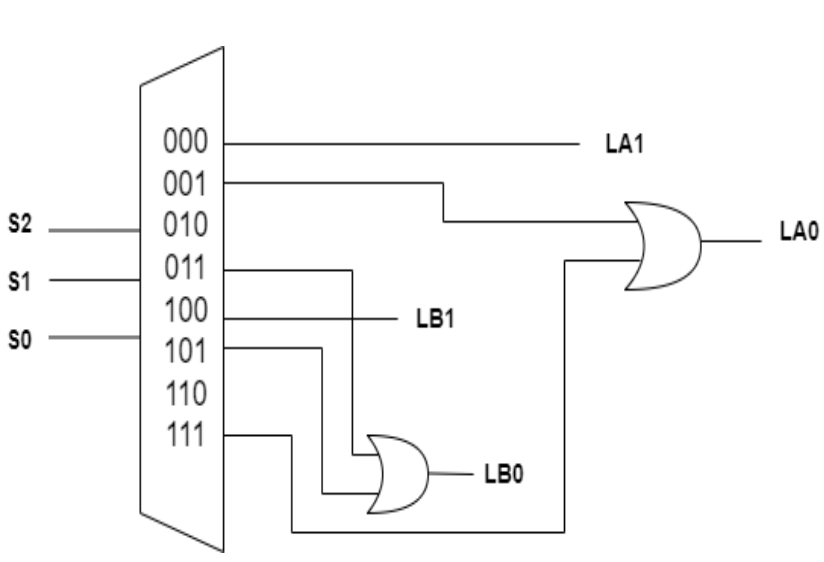
$$LB_1 = s_2 \bar{s}_1 \overline{s_0}$$

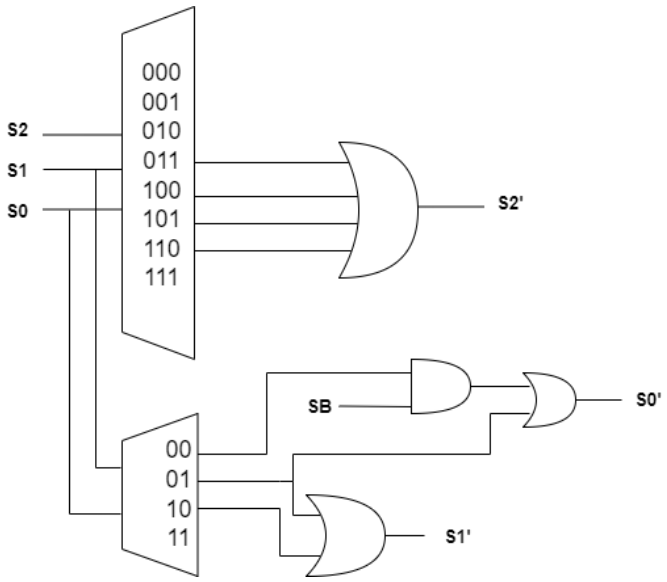
$$LB_0 = \bar{s}_2 s_1 s_0 + s_2 \overline{s_1} s_0$$

FSM Design:



Redesign using decoders:





- **3 Flip Flops are needed to implement this FSM**

Code:

```
module fsm(input logic clk, res, sa, sb, output logic [1:0]la, [1:0]lb);
  typedef enum logic [2:0] {s0, s1, s2, s3, s4, s5, s6, s7} statetype;
  statetype [1:0] state, nextstate;
  parameter red = 2'b00;
  parameter yellow = 2'b01;
  parameter green = 2'b10;
  logic newClk;
  clockDivider divider(clk, res, newClk);
  always_ff@(posedge newClk, posedge res)
    if(res) state <= s0;
    else state <= nextstate;
  always_comb
    case(state)
      s0:   if(sb) nextstate <= s1;
            else nextstate <= s0;
      s1:   nextstate <= s2;
      s2:   nextstate <= s3;
      s3:   nextstate <= s4;
      s4:   if(sa) nextstate <= s5;
            else nextstate <= s4;
      s5:   nextstate <= s6;
      s6:   nextstate <= s7;
      s7:   nextstate <= s0;
      default: nextstate <= s0;
     endcase
  always_comb
    case(state)
      s0:   {la, lb} = {green, red};
      s1:   {la, lb} = {yellow, red};
      s2:   {la, lb} = {red, red};
      s3:   {la, lb} = {red, yellow};
      s4:   {la, lb} = {red, green};
      s5:   {la, lb} = {red, yellow};
```

```
        s6:     {la, lb} = {red, red};
        s7:     {la, lb} = {yellow, red};
        default: {la, lb} = {green, red};
    endcase
endmodule
```

```
module clockDivider(input logic inClk, res, output logic outClk);
  logic [31:0] counter = {32{1'b0}};
  always @(posedge inClk)
    begin
      if(res) counter <= 0;
      else counter <= counter + 1;
    end
    logic last = counter[31];
    BUFG BUFG_inst (.I(last),.O(outClk));
endmodule
```

```
module fsm_tb();
  logic clk, res, sa, sb;
  logic [1:0] la;
  logic [1:0] lb;
  fsm test(clk, res, sa, sb, la, lb);
  always
    begin
      clk = 0; #3; clk = 1; #3;
    end
  initial
    begin
      res = 0; sa = 0; sb = 0;#6;
      sb = 1; #6;
      sa = 1; sb = 0; #6;
      sb = 1; #6;
    end
endmodule
```