CS223 Section-002

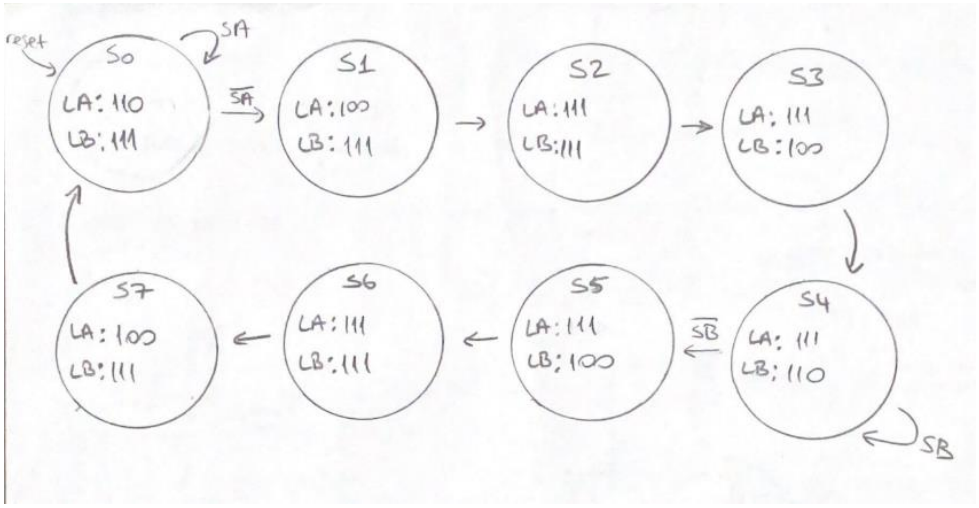
Lab-5

Elif Ercan

22201601

26/11/23

Moore Machine State Transition Diagram



State Encodings

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

State Transition Table

Current state			Inputs		Next state		
S2	S1	S0	SA	SB	S2’	S1’	S0’
0	0	0	0	X	0	0	1
0	0	0	1	X	0	0	0
0	0	1	0	X	0	1	1
0	0	1	1	X	0	1	0
0	1	0	X	X	0	1	1
0	1	1	X	X	1	0	0
1	0	0	X	0	1	0	1
1	0	0	X	1	1	0	0
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 Encodings

Output	Encoding
Red	111
Yellow	100
Green	110

Output Table

S2	S1	S0	LA2	LA1	LA0	LB2	LB1	LB0
0	0	0	1	1	0	1	1	1
0	0	1	1	0	0	1	1	1
0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	0	0
1	0	0	1	1	1	1	1	0
1	0	1	1	1	1	1	0	0
1	1	0	1	1	1	1	1	1
1	1	1	1	0	0	1	1	1

Next State Equations

$S2' = S2\ \overline{S}1 + S2\ \overline{S}0 + \overline{S}2\ S1\ S0$

$S1' = \overline{S}1\ S0 + S1\ \overline{S}0$

$S0' = S1\ \overline{S}0 + \overline{S}2\ \overline{S}1\ \overline{S}A + S2\ \overline{S}0\ \overline{S}B$

Output Equations

$LA2 = 1$

$LA1 = \overline{S}0 + \overline{S}2\ S1 + S2\ \overline{S}1$

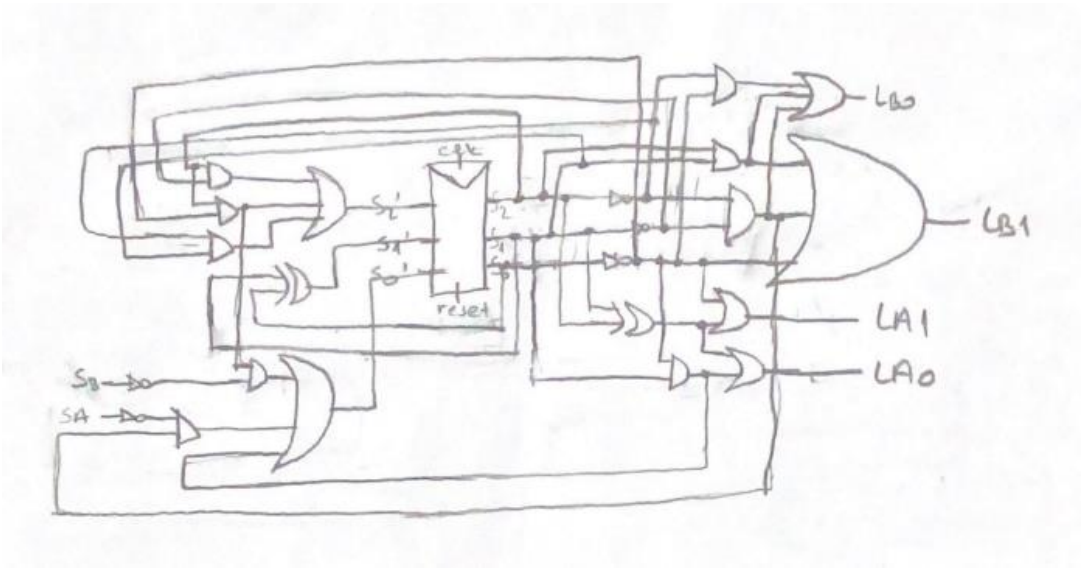
$LA0 = \overline{S}2\ S1 + S1\ \overline{S}0 + S2\ \overline{S}1$

$LB2 = 1$

$LB1 = \overline{S}0 + \overline{S}2\ \overline{S}1 + S2\ S1$

$LB0 = \overline{S}2\ \overline{S}1 + \overline{S}2\ \overline{S}0 + S2\ S1$

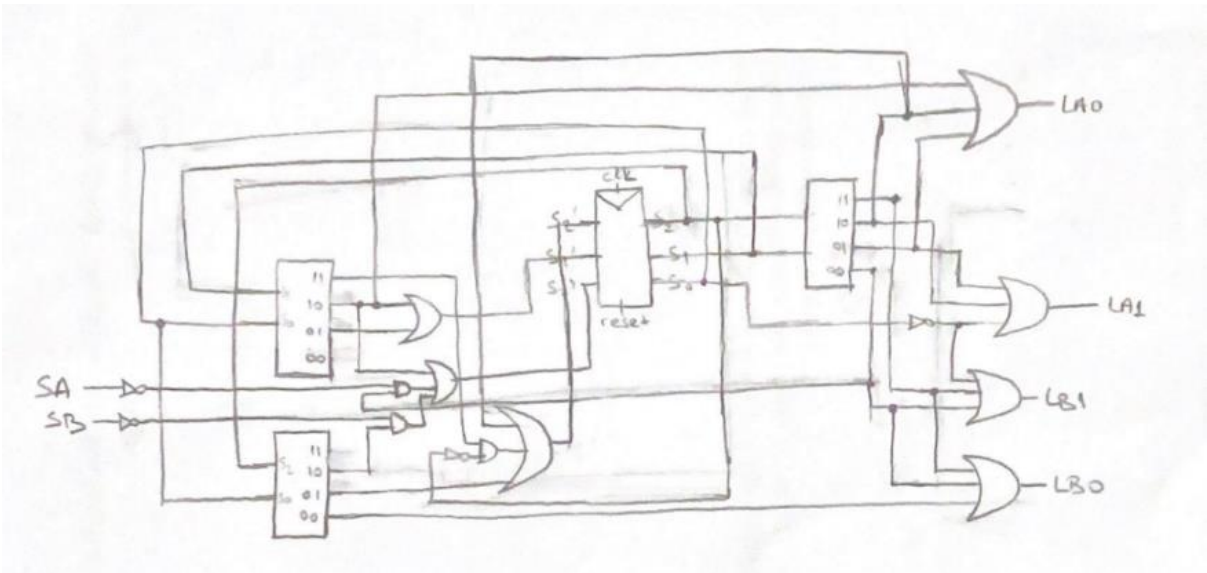
Finite State Machine Schematic



How many flip-flops you need to implement this problem?

I need to implement 3 flip-flops for this problem.

Redesigned Machine Using Decoders



SystemVerilog Codes for the Register

```
module register_lab5(
  input clk,
    input reset,
    input sa,
    input sb,
    output la2,
    output la1,
    output la0,
    output lb2,
    output lb1,
    output lb0
     );

     typedef enum {S0,S1,S2,S3,S4,S5,S6,S7} statetype;
     statetype state, nextstate;

     wire w1;
     clock_divider c (clk,reset,w1);

     always_ff@(posedge w1,posedge reset)
     if(reset) state<=S0;
     else state<=nextstate;

     always_comb
     case(state)
     S0: if(sa) nextstate=S0;
         else nextstate=S1;
     S1: nextstate=S2;
     S2: nextstate=S3;
     S3: nextstate=S4;
     S4: if(sb) nextstate=S4;
         else nextstate=S5;
     S5: nextstate=S6;
     S6: nextstate=S7;
     S7: nextstate=S0;
     default: nextstate=S0;
     endcase

     assign la2=(1);
     assign la1=(state==S0 | state==S2 | state==S3 | state==S4 |
state==S5 | state==S6);
     assign la0=(state==S2 | state==S3 | state==S4 | state==S5 |
state==S6);

     assign lb2=(1);
     assign lb1=(state==S0 | state==S1 | state==S2 | state==S4 |
state==S6 | state==S7);
     assign lb0=(state==S0 | state==S1 | state==S2 | state==S6 |
state==S7);


endmodule
```

SystemVerilog Codes for the Register's Testbench

```
module register_tb;

reg sa;
reg sb;
reg clk;
reg reset;
wire la2;
wire la1;
wire la0;
wire lb2;
wire lb1;
wire lb0;

register_lab5 uut (.sa(sa), .sb(sb), .clk(clk), .reset(reset),
.la2(la2), .la1(la1), .la0(la0), .lb2(lb2), .lb1(lb1), .lb0(lb0));

  always begin
     clk = 0; #5; clk=1; #5;
     end

     initial begin
     #5  sa=0; sb=0; reset=1;
      #5 sa=0; sb=0; reset=0;
      #20 sa=0; sb=1; reset=0;
      #20 sa=1; sb=0; reset=0;
      #20 sa=1; sb=1; reset=0;
      #20;
      $finish;
      end
endmodule
```

SystemVerilog Codes for the Clock Divider

```
module clock_divider(
input clk,
     input reset,
     output clk_divd
     );
     reg clk_divd;
     reg[100:0] count;
     always@(posedge clk)
     begin
     if(reset)
     count <= 0;
      if(count==150_000_000)
     count <=0;
     else
     count <= count+1;
     end
     always@(posedge clk)
        if(reset)
        clk_divd <= 0;
        else if(count==150_000_000)
        clk_divd <= ~clk_divd;
        else
        clk_divd <= clk_divd;
endmodule
```