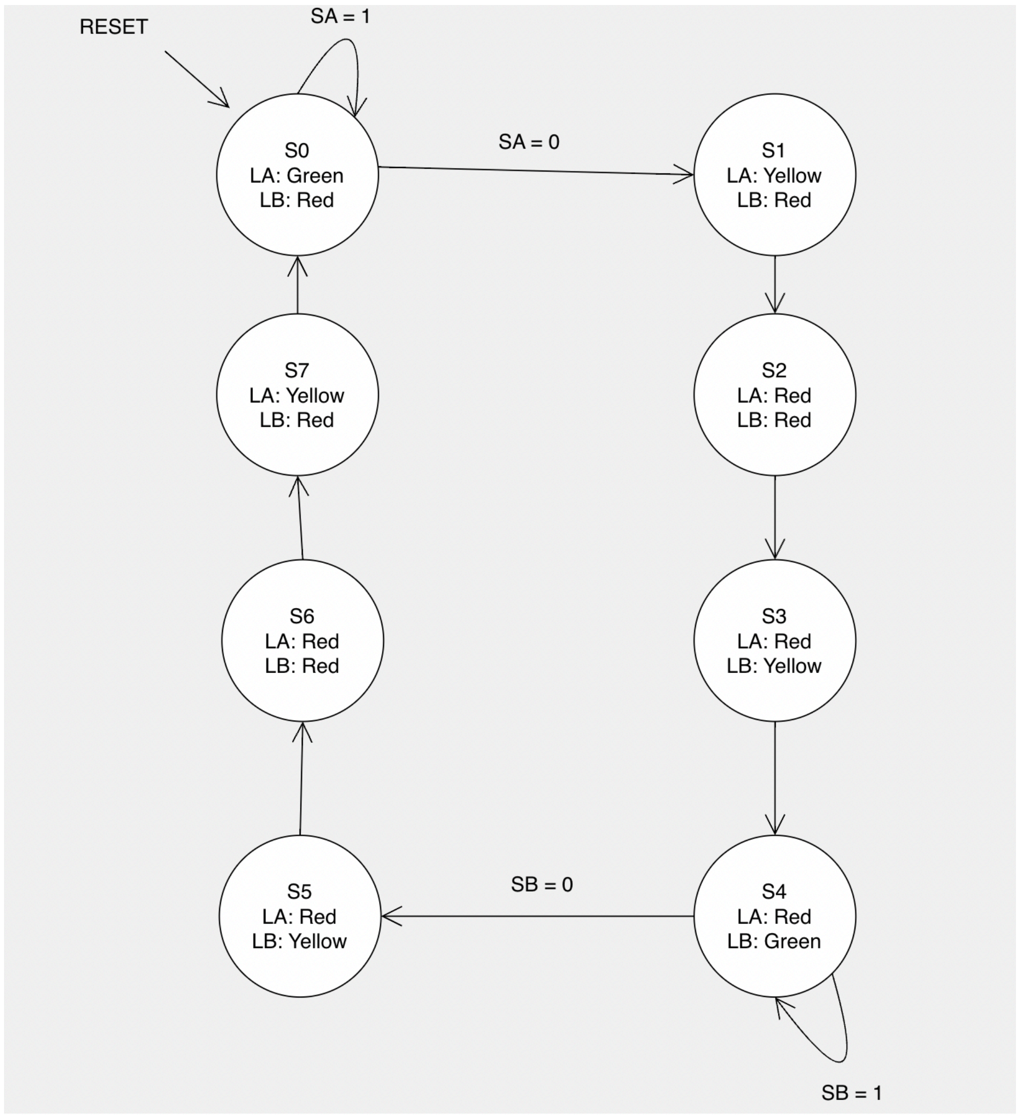
DIGITAL DESIGN CS223
LAB 4

YASEMİN AKIN
22101782

28 NOVEMBER 2022

a) Improved Moore machine state transition diagram, state encodings, state transition table, output table, next state, and output equations.



STATE	ENCODINGS $S_{2:0}$
S0	000
S1	001
S2	010
S3	011
S5	100
S5	101
S6	110
S7	111

CURRENT STATE S	**INPUTS**		**NEXT STATE S'**
	SA	**SB**	
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	X	S3
S3	X	X	S4
S4	X	0	S5
S4	X	1	S4
S5	X	X	S6
S6	X	X	S7
S7	X	X	S0

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	X	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	**ENCODING $L_{1:0}$**
Green	10
Yellow	01
Red	11

CURRENT STATE			**OUTPUTS**			
S2	**S1**	**S0**	**LA1**	**LA0**	**LB1**	**LB0**
0	0	0	1	0	1	1
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	0	1
1	0	0	1	1	1	0
1	0	1	1	1	0	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1

$$SO' = \overline{S2}\,.\,\overline{S1}\,.\,\overline{S0}\,.\,\overline{SA} + S1\,.\,\overline{S0} + S2\,.\,\overline{S1}\,.\,\overline{S0}\,.\,\overline{SB}$$

$$S1' = S1 \oplus S0$$

$$S2' = \overline{S2}\,.\,S1\,.\,S0 + S2\,.\,\overline{S1} + S2\,.(S1 \oplus S0)$$

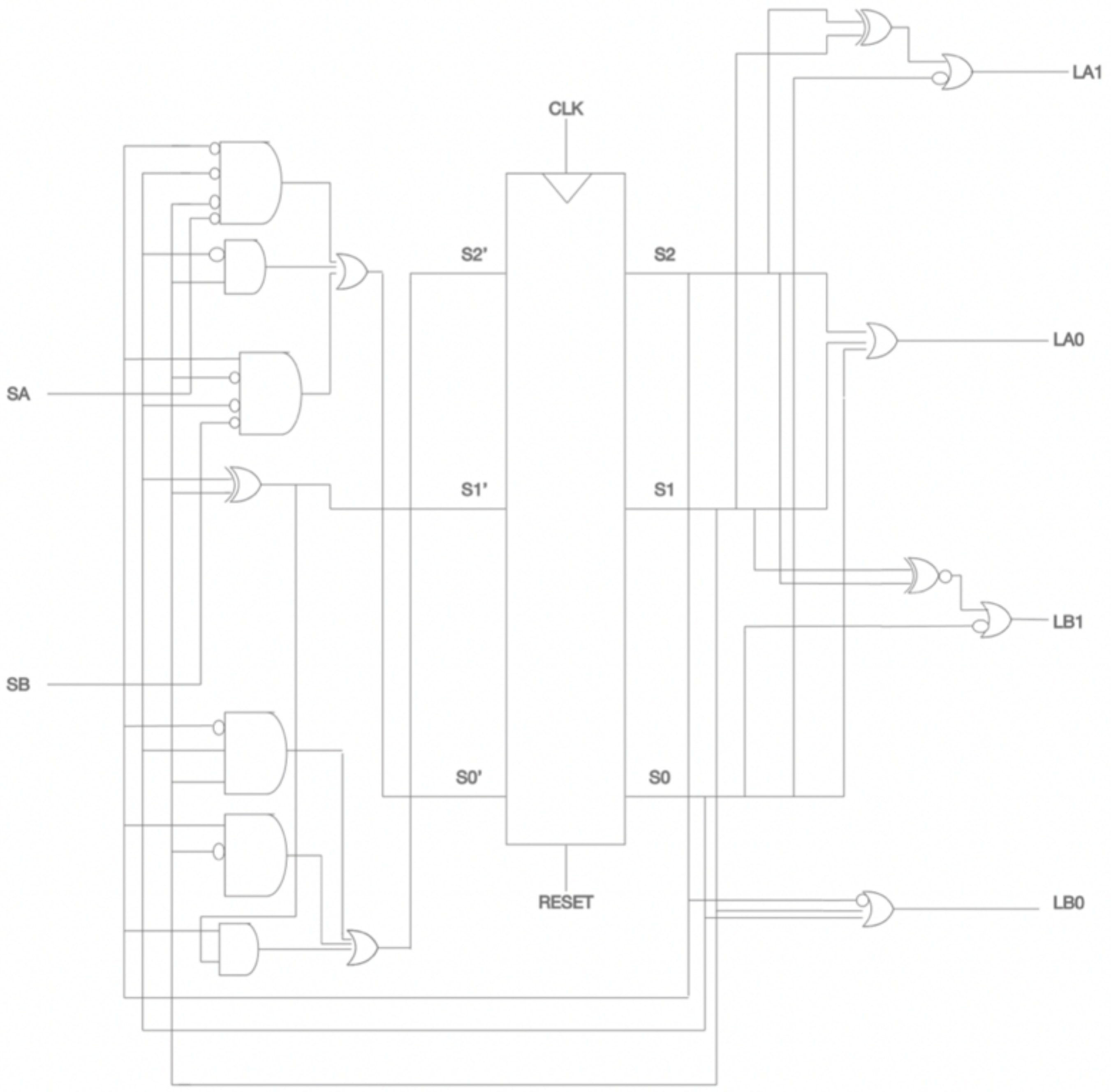
$$LA0 = S0 + S1 + S2$$

$$LA1 = \overline{S0} + (S1 \oplus S2)$$

$$LB0 = S0 + S1 + \overline{S2}$$

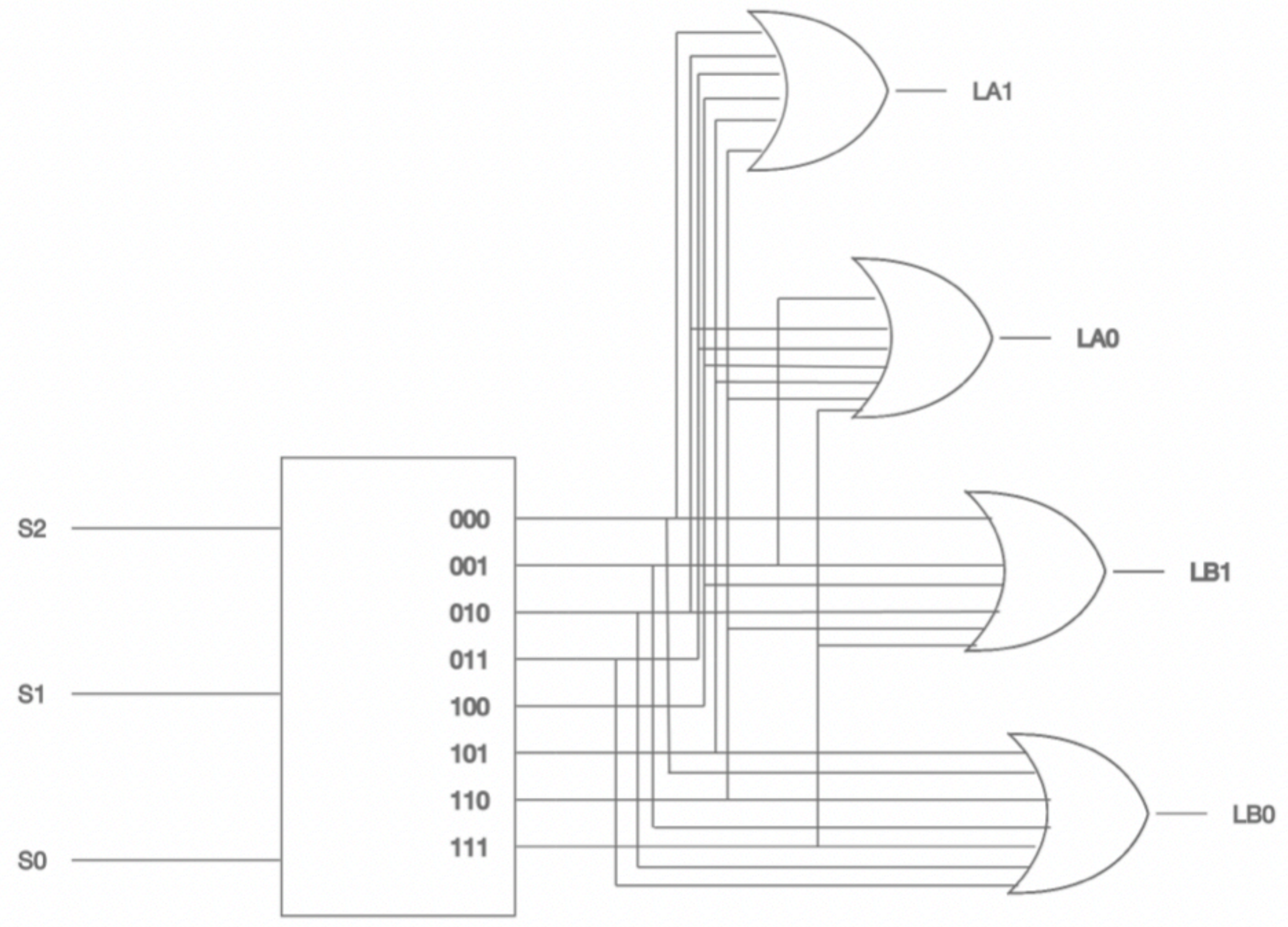
$$LB1 = \overline{S0} + \overline{(S1 \oplus S2)}$$

b) Finite State Machine schematic.



c) How many flip-flops do you need to implement this problem? 3

d) Redesign of your outputs using decoders.



e) System Verilog design code and testbench.

```
//FSM SYSTEMVERILOG CODE
`timescale 1ns/1ps
module FSM_TL (input logic clk, reset, SA, SB, output
logic [2:0] LA, LB);

typedef enum logic [2:0] {S0,S1,S2,S3,S4,S5,S6,S7}
statetype;
statetype state, nextstate;
logic clkDivider;

assign clkDivider = clk;

always_ff@(posedge clkDivider)
    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;
    endcase

always_comb
    case(state)
        S0: begin
            LA <= 3'b110;
            LB <= 3'b111;
        end
        S1: begin
```

```
                LA <= 3'b100;
                LB <= 3'b111;
            end
            S2: begin
                LA <= 3'b111;
                LB <= 3'b111;
            end
            S3: begin
                LA <= 3'b111;
                LB <= 3'b100;
            end
            S4: begin
                LA <= 3'b111;
                LB <= 3'b110;
            end
            S5: begin
                LA <= 3'b111;
                LB <= 3'b100;
            end
            S6: begin
                LA <= 3'b111;
                LB <= 3'b111;
            end
            S7: begin
                LA <= 3'b100;
                LB <= 3'b111;
            end
        endcase
    endmodule
```

```
//CLOCK
`timescale 1ns/1ps
module clockDivider (input logic clk, output logic
clkDivider);

    logic [31:0] count;
    always @(posedge clk)
    begin
        if(count == 149999999)
            count <= 32'b0;
        else
            count <= count+1;
    end
    always @(posedge clk)
    begin
        if(count == 149999999)
            clkDivider <= ~clkDivider;
        else
            clkDivider <= clkDivider;
    end
endmodule

`timescale 1ns/1ps

//testbench
module FSM_TestBench();
    logic clk, reset, SA, SB;
    logic [2:0] LA, LB;
    FSM_TL testFSM(clk, reset, SA, SB, LA, LB);
    initial begin
        clk <= 0;
        reset = 1; SA = 1; SB = 1; #4;
        reset = 0; #4;

        for(int i = 0; i < 4; i++) begin
            SB = 0; #4;
            SB = 1; SA = 0; #4;
            {SA, SB} = i; #8;
            SB = 1; #4;
            SA = ~SA; #4;
            SB = 0; #4;
            {SA, SB} = i; #8;
            SA = 1; SB = 1; #4;
        end

        reset = 1; #4;
        for(int i = 0; i < 4; i++) begin
            SB = 0; #4;
            SB = 1; SA = 0; #4;
            {SA, SB} = i; #8;
            SB = 1; #4;
            SA = ~SA; #4;
            SB = 0; #4;
            {SA, SB} = i; #8;
            SA = 1; SB = 1; #4;
        end
    end

    always #2 clk <= ~clk;

endmodule
```