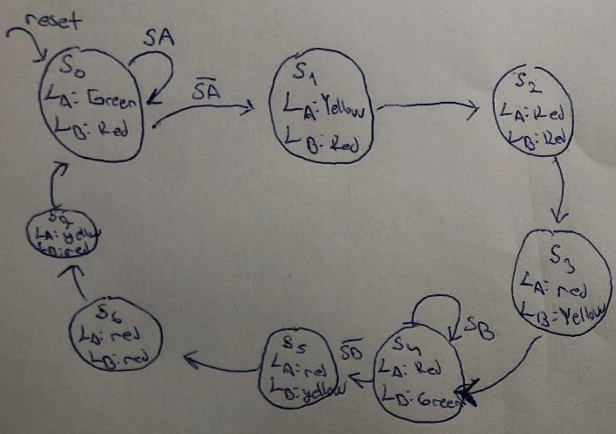
CS 223 Digital Design
Section 01
Lab 04

NAME: Ferhat
SURNAME: Korkmaz
ID: 21901940
DATE: 08.04.2021

a-)



Encoding of states

S_0 000
S_1 001
S_2 010
S_3 011
S_4 100
S_5 101
S_6 110
S_7 111

Inputs

S_A
S_B

Outputs

L_A, L_B → green 00
yellow 01
red 10

$L_A \to L_{A_1}, L_{A_2}$ encoding (2 bit)

$L_B \to L_{B_1}, L_{B_2}$ encoding (2 bit)

State transition and output table

S_2	S_1	S_0	S_A	S_B	S_2'	S_1'	S_0'	L_{A_1}	L_{A_0}	L_{B_1}	L_{B_0}
0	0	0	0	X	0	0	1	0	0	1	0
0	0	0	1	X	0	0	0				
0	0	1	X	X	0	1	0	0	1	1	0
0	1	0	X	X	0	1	1	1	0	1	0
0	1	1	X	X	1	0	0	1	0	0	1
1	0	0	X	0	1	0	1	1	0	0	0
			X	1	1	0	0				
1	0	1	X	X	1	1	0	1	0	0	1
1	1	0	X	X	1	1	1	1	0	1	0
1	1	1	X	X	0	0	0	0	1	1	0

$S_0' = \bar{S_0} S_1 + \bar{S_0} \bar{S_2} \overline{SA} + \bar{S_0} S_2 \bar{S_B}$

$S_1' = S_1 \oplus S_0$

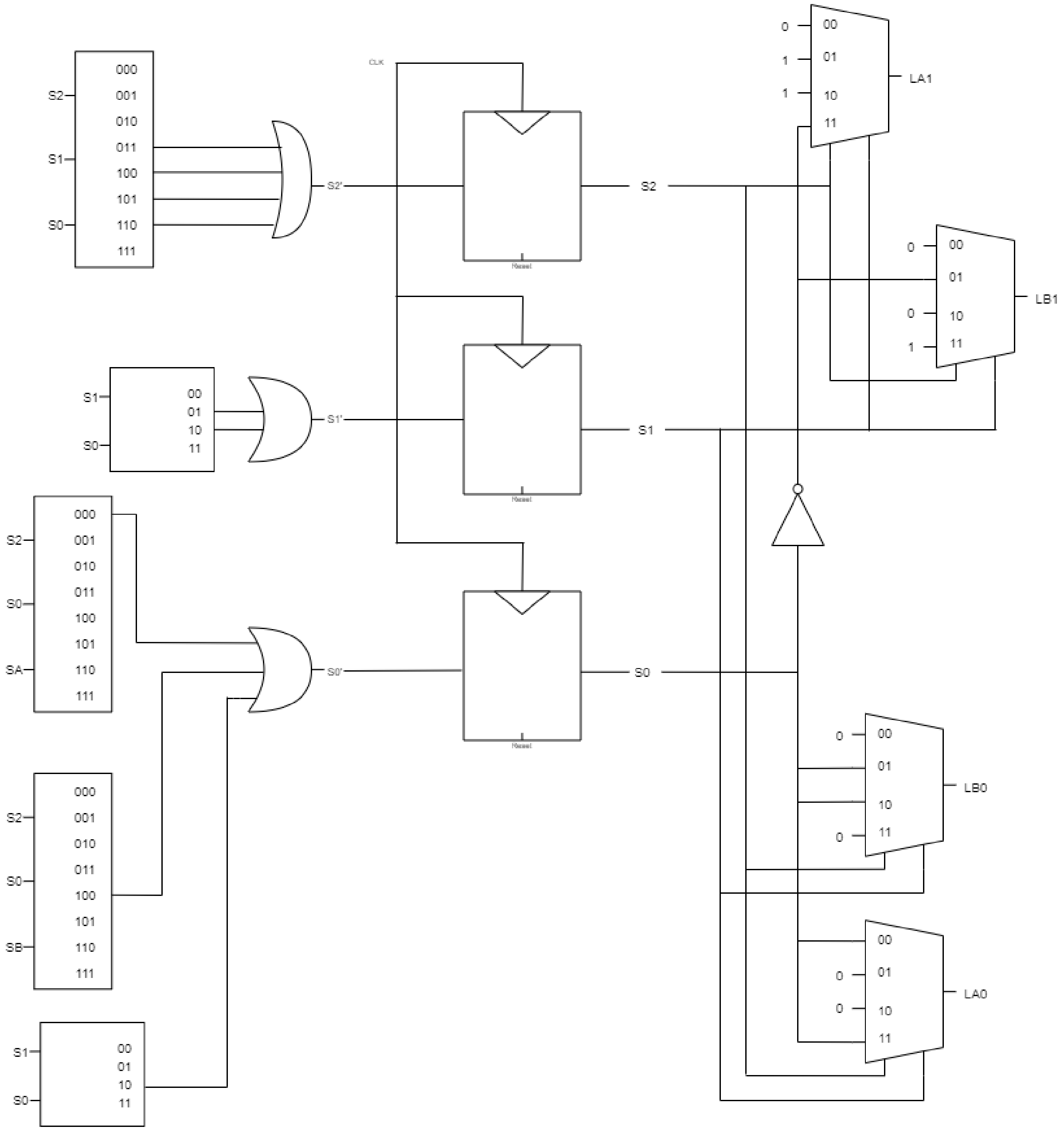
$S_2' = \bar{S_2} \oplus S_1 S_0$

$L_{A_1} = S_2 \oplus S_1 + S_1 \bar{S_0}$

$L_{A_0} = S_0 (\overline{S_2 \oplus S_1})$

$L_{B_1} = \overline{S_2 \oplus S_1} + S_1 \bar{S_0}$

$L_{B_0} = S_0 (S_2 \oplus S_1)$



b-) 3 Flip-flops are needed in order to implement this problem since we have 8 states that are encoded as 3 bits.

c-)

module twoToFourDecoder(input logic input1, input0 ,output logic[3:0] myoutput);

assign myoutput[3] = input1 & input0;

assign myoutput[2] = input1 & ~input0;

```
 assign myoutput[1] = ~input1 & input0;
 assign myoutput[0] = ~input1 & ~input0;

endmodule

module fourToOneMux( input logic d0, d1, d2, d3, s0, s1, output logic y);
    assign y = s1 ? ( s0 ? d3: d2 ) : ( s0 ? d1: d0);
endmodule

module threeToEightDecoder( input logic input2, input1, input0, output logic [7:0]
myoutput);

assign myoutput[0] = ~input2 & ~input1 & ~input0;
assign myoutput[1] = ~input2 & ~input1 & input0;
assign myoutput[2] = ~input2 & input1 & ~input0;
assign myoutput[3] = ~input2 & input1 & input0;
assign myoutput[4] = input2 & ~input1 & ~input0;
assign myoutput[5] = input2 & ~input1 & input0;
assign myoutput[6] = input2 & input1 & ~input0;
assign myoutput[7] = input2 & input1 & input0;

endmodule


module green(input logic[2:0] current_state, output logic [1:0] LA, LB);
    fourToOneMux assignLA0(current_state[0],0,0, current_state[0],
current_state[1],current_state[2], LA[0]);
    fourToOneMux assignLA1(0, 1, 1, ~current_state[0], current_state[1], current_state[2],
LA[1]);
    fourToOneMux assignLB0(0,current_state[0], current_state[0], 0, current_state[1],
current_state[2], LB[0]);
```

```
    fourToOneMux assignLB1(1, ~current_state[0], 0, 1, current_state[1], current_state[2], LB[1]);
endmodule



module blue(input logic SA, SB, input logic [2:0] state, output logic[2:0] nextstate);
    logic [3:0] n3;
    logic [7:0]n1, n2, n4;
    threeToEightDecoder s2next(~state[2], state[1], state[0], n4);
    twoToFourDecoder s1next(state[1], state[0], n3);
    threeToEightDecoder s0next(state[2], state[0],SA , n1);
    threeToEightDecoder s0next2(state[2], state[0],SB , n2);
    twoToFourDecoder s0next3(state[1], state[0], n3);
    assign nextstate[0] = n1[0] | n2[4] | n3[2];
    assign nextstate[1] = n3[1] | n3[2];
    assign nextstate[2] = n4[0] |n4[1] | n4[2] | n4[7];
endmodule

module TrafficLightModule(input logic reset, clk , SA, SB,
                        output logic [2:0] next_state, LA3, LB3
                         );
typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6, S7} statetype;
logic [2:0]  current_state;

//typedef enum logic [1:0] {red, yellow, green} lights;
logic [1:0] LA, LB;

blue firstcall(SA, SB, current_state, next_state);
DFlipFlop flop0(clk, reset, next_state[0], current_state[0]);
DFlipFlop flop1(clk, reset, next_state[1], current_state[1]);
DFlipFlop flop2(clk, reset, next_state[2], current_state[2]);
```

```
green secondcall(current_state, LA, LB);



assign LA3[2] = ~LA[1] & ~LA[0];
assign LA3[1] = ~LA[1];
assign LA3[0] = 1;
assign LB3[2] = ~LB[1] & ~LB[0];
assign LB3[1] = ~LB[1];
assign LB3[0] = 1;

endmodule

// Flip-flop D
module DFlipFlop(
        input clk, rst, d,
        output logic q);

  always@(posedge clk or posedge rst)
      if (rst)
        q <= 0;
      else
        q <= d;
endmodule
```