BILKENT UNIVERSITY

COMPUTER ENGINEERING DEPT.

CS223 LAB 4 SECTION 4

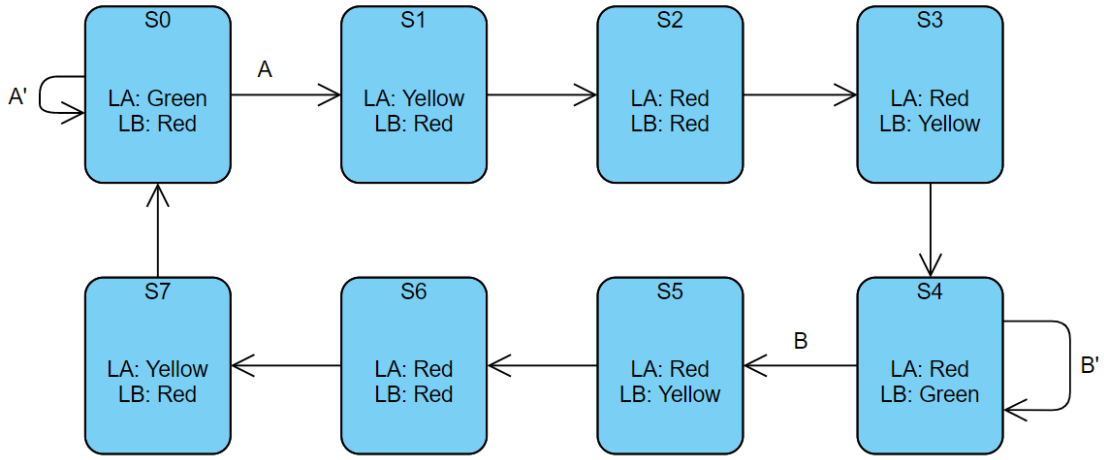
LAB REPORT

ZÜBEYİR BODUR

21702382

1. FSM Design

a) State Transition Diagram



For convenience, inputs are named into A & B from SA & SB.

b) State & Output Encoding

State	S_2	S_1	S_0
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0
S5	1	0	1
S6	1	1	0
S7	1	1	1

Output encoding for the traffic lights at road X (A or B) is the following. In total, we have 6 outputs: LA_2, LA_1, LA_0 for the road A and LB_2, LB_1, LB_0 for the road B.

Output	LX_2	LX_1	LX_0
Red	1	0	0
Yellow	1	1	0
Green	1	1	1

c) State Transition Table

Current State	S_2	S_1	S_0	A	B	S'_2	S'_1	S'_0	Next State
S0	0	0	0	0	X	0	0	1	S1
S0	0	0	0	1	X	0	0	0	S0
S1	0	0	1	X	X	0	1	0	S2
S2	0	1	0	X	X	0	1	1	S3
S3	0	1	1	X	X	1	0	0	S4
S4	1	0	0	X	0	1	0	1	S5
S4	1	0	0	X	1	1	0	0	S4
S5	1	0	1	X	X	1	1	0	S6
S6	1	1	0	X	X	1	1	1	S7
S7	1	1	1	X	X	0	0	0	S0

In order not to have 32 rows and for the sake of legibility, some inputs were given DON'T CARE values when necessary.

d) Output Table

State	LA	LB
S0	Green	Red
S1	Yellow	Red
S2	Red	Red
S3	Red	Yellow
S4	Red	Green
S5	Red	Yellow
S6	Red	Red
S7	Yellow	Red

State			Output					
S_2	S_1	S_0	LA_2	LA_1	LA_0	LB_2	LB_1	LB_0
0	0	0	1	1	1	1	0	0
0	0	1	1	1	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	1	1	0	0	1	1	0
1	0	0	1	0	0	1	1	1
1	0	1	1	0	0	1	1	0
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	1	0	0

e) Boolean Equations

$$S'_2 = \bar{S_2}S_1S_0 + S_2\bar{S_1} + S_2S_1\bar{S_0} = F(S_2, S_1, S_0) = \sum m(3,4,5,6)$$

$$S'_1 = \bar{S_1}S_0 + S_1\bar{S_0} = G(\ S_1, S_0) = \sum m(1,2)$$

$$S'_0 = \bar{S_2}\ \bar{S_1}\ \bar{S_0}\ \bar{A} + S_2\ \bar{S_1}\ \bar{S_0}\ \bar{B} + S_1\ \bar{S_0}$$
$$= P(\ H(S_2, S_1, S_0), \quad \bar{A}) + P(\ K(S_2, S_1, S_0), \quad \bar{B}) + R(S_1, S_0)$$

where

$$H(X, Y, Z) = \bar{X}\ \bar{Y}\ \bar{Z} = \sum m(0)$$

$$K(X, Y, Z) = X\ \bar{Y}\ \bar{Z} = \sum m(4)$$

$$P(X, Y) = X\ Y = \sum m(3)$$

$$R(X, Y) = X\ \bar{Y}\ = \sum m(2)$$

$$LA_2 = 1$$

$$LA_1 = \bar{S_2}\ \bar{S_1} + S_2S_1S_0$$

$$LA_0 = \bar{S_2}\ \bar{S_1}\ \bar{S_0}$$

$$LB_2 = 1$$

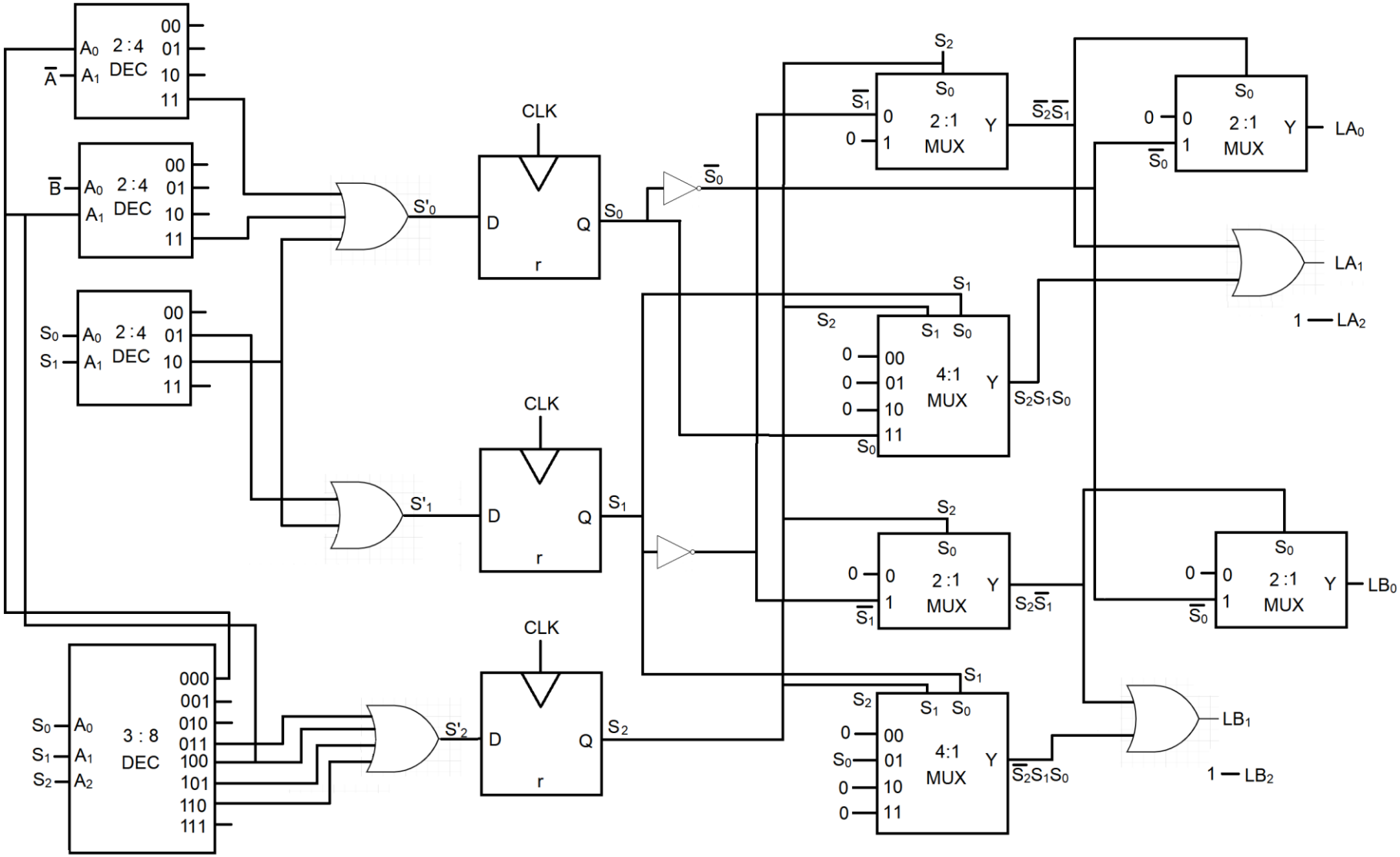
$$LB_1 = S_2\ \bar{S_1} + \bar{S_2}S_1S_0$$

$$LB_0 = S_2\ \bar{S_1}\ \bar{S_0}$$

f) Number of Flip Flops

Only **3** D-Flip Flops are needed for this FSM, as there are three state bits.

2. Schematic



3. SystemVerilog Modules

```
/**
 * Title: Traffic Light Controller
 * Author: Zubeyir Bodur
 * ID: 21702382
 * Section: 4
 * Lab : 4
 * Description: Design sources
`timescale 1ns / 1ps

module trafficLightController(
    output logic [2:0] LA, LB,
    input logic clk, rst,
    input logic A, B
    );
    logic [2:0] S, S_next;
    logic slow_clk;
    nextStateLogic NSL(S_next, S, A, B);
    clockDivider#(27) div(slow_clk, clk);
    register#(3) stateRegister(S, slow_clk, rst, S_next);
    outputLogic OL(LA, LB, S);
endmodule

module trafficLightControllerSim(
    output logic [2:0] LA, LB,
    input logic clk, rst,
    input logic A, B
    );
    logic [2:0] S, S_next;
    logic slow_clk;
    nextStateLogic NSL(S_next, S, A, B);
    register#(3) stateRegister(S, clk, rst, S_next);
    outputLogic OL(LA, LB, S);
endmodule
```

```
// Divides the the period of CLK to N
module clockDivider#(parameter N = 26)(
    output logic clk_out,
    input logic clk_in
    );
    logic [N-1:0] count = {N{1'b0}};
    always@ (posedge clk_in) begin
        count <= count + 1;
    end
    assign clk_out = count[N-1];
endmodule

module nextStateLogic(
    output logic [2:0] S_next,
    input logic [2:0] S,
    input logic A, B
    );
    logic [7:0] minterms4;
    logic [3:0] minterms1, minterms2, minterms3;
    dec#(2) d2_1(minterms1, {minterms4[0], ~A});
    dec#(2) d2_2(minterms2, {minterms4[4], ~B});
    dec#(2) d2_3(minterms3, S[1:0]);
    dec#(3) d3_4(minterms4,S[2:0]);
    assign S_next = {minterms4[3] + minterms4[4] + minterms4[5] +
minterms4[6],
                     minterms3[1] + minterms3[2],
                     minterms3[2] + minterms2[3] + minterms1[3]};
endmodule
```

```
module outputLogic(
    output logic [2:0] LA, LB,
    input logic [2:0] S
    );
    logic S2_NOR_S1, S2_S1_S0, S2_S1NOT, S2NOT_S1_S0, S0NOT,
S1NOT;
    logic [1:0] in2_1, in2_2, in2_3, in2_4;
    logic [3:0] in4_1, in4_2;

    assign S0NOT = ~S[0];
    assign S1NOT = ~S[1];

    assign in2_1 = {1'b0, S1NOT};
    assign in2_2 = {S0NOT, 1'b0};
    assign in2_3 = {S1NOT, 1'b0};
    assign in2_4 = {S0NOT, 1'b0};
    assign in4_1 = {S[0], {3{1'b0}} };
    assign in4_2 = {{2{1'b0}}, S[0], 1'b0};

    mux2_1 m2_1(S2_NOR_S1, in2_1, S[2]);
    mux2_1 m2_2(LA[0], in2_2, S2_NOR_S1);
    mux2_1 m2_3(S2_S1NOT, in2_3, S[2]);
    mux2_1 m2_4(LB[0], in2_4, S2_S1NOT);
    mux4_1 m4_1(S2_S1_S0, in4_1, S[2:1]);
    mux4_1 m4_2(S2NOT_S1_S0, in4_2, S[2:1]);

    assign LA[2:1] = {1'b1, S2_NOR_S1 + S2_S1_S0};
    assign LB[2:1] = {1'b1, S2_S1NOT + S2NOT_S1_S0};
endmodule
```

```
// register with synchronous reset
module register#(parameter N = 1)(
    output logic [N - 1: 0] Q,
    input logic clk, rst,
    input logic [N - 1: 0] D
    );
    logic [N - 1 : 0] d_;
    // mux2_1#(N) enmux(d_, D, Q, en); // multiplexer for enable
    logic [1:0] [N - 1 : 0] inputs;
    assign inputs = {{N{1'b0}}, D};
    mux2_1#(N) rstmux(d_, inputs, rst); // multiplexer for reset
    always_ff @(posedge clk) begin
        Q <= d_;
    end
endmodule

// 2 to 1 mux, N bits
module mux2_1#(parameter N = 1)(
    output logic [N - 1 : 0] Y,
    input logic [1:0] [N - 1 : 0] D,
    input logic S
    );
    always_comb begin
        case (S)
            1'b0: Y = D[0];
            1'b1: Y = D[1];
            default: Y = {N{1'bZ}}; // handle the contention select
        endcase
    end
endmodule
```

```
// 4 to 1 mux, N bits
module mux4_1#(parameter N = 1)(
    output logic [N - 1 : 0] Y,
    input logic [3:0] [N - 1 : 0] D,
    input logic [1:0] S
    );
    always_comb begin
        case (S)
            2'b00: Y = D[0];
            2'b01: Y = D[1];
            2'b10: Y = D[2];
            2'b11: Y = D[3];
            default: Y = {N{1'bZ}}; // handle the contention select
        endcase
    end
endmodule

// N : 2 ^ N - 1 decoder
module dec#(parameter N = 2)(
    output logic [2**N - 1:0] Y,
    input logic [N - 1:0] A
    );
    always_comb begin
        Y = 0;
        Y[A] = 1;
    end
endmodule
```