2018-2019 FALL SEMESTER

CS 223 – DIGITAL DESIGN

LAB 4 – 25.11.2018

SECTION: 1

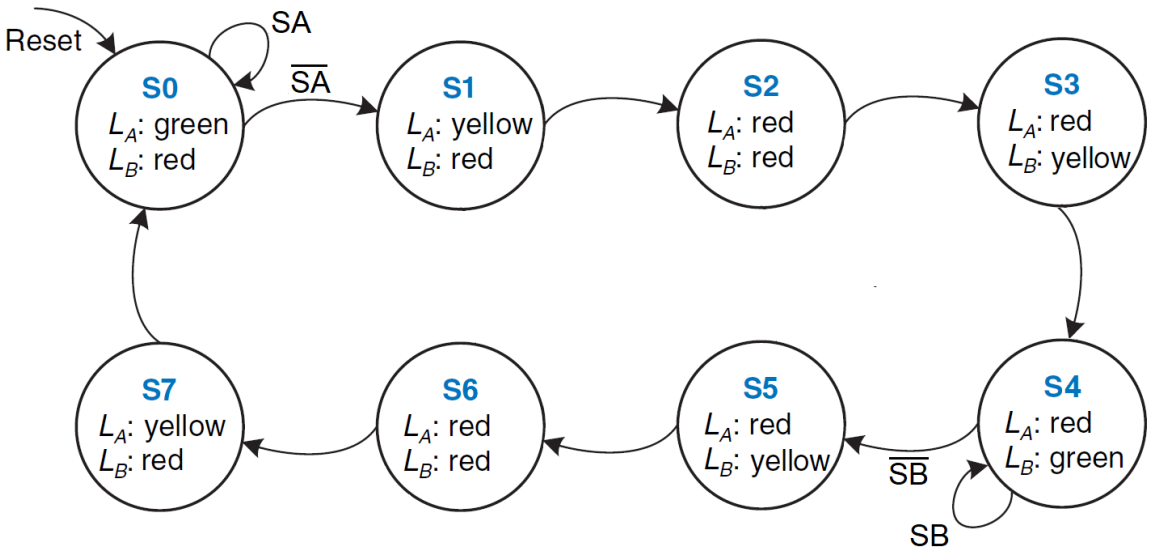
NAME: MUNIB EMRE

SURNAME: SEVILGEN

STUDENT ID: 21602416

TRAINER PACK: 09

a)



CURRENT STATE			INPUT		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

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

CURRENT STATE			OUTPUT			
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

OUTPUT ENCODING	
RED	00
YELLOW	01
GREEN	10

Equations:

$S2' = (\, S0\, S1\,) \oplus S2$

$S1' = S0 \oplus S1$

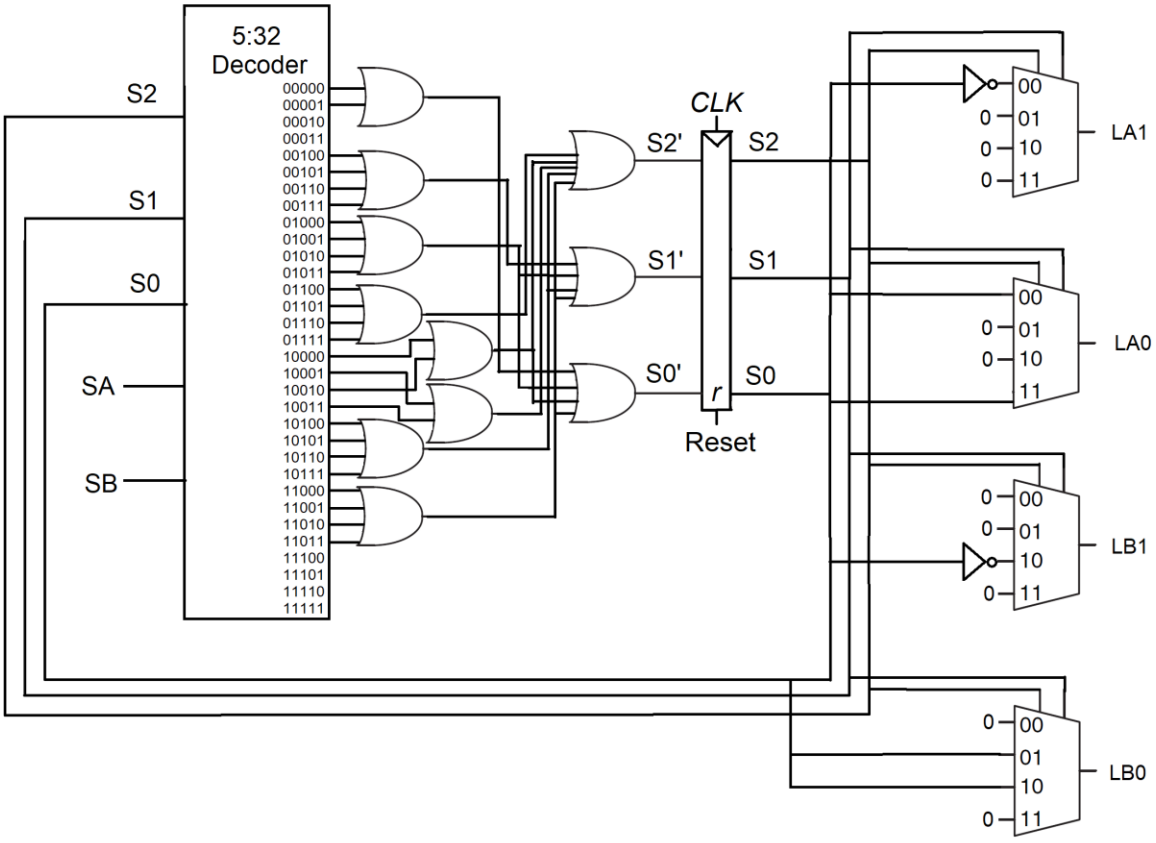
$S0' = \overline{S0}\,(\, S1 + \overline{S1}\,(\, \overline{S2}\,\overline{SA} + S2\,\overline{SB}\,))$

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

$LA0 = S0\,\overline{(\, S1 \oplus S2\,)}$

$LB1 = \overline{S0}\,\overline{S1}\,S2$

$LB0 = S0\,(\, S1 \oplus S2)$



b) We need 3 flip flops to implement this problem.

c)

```
module blue_function (input logic [2:0] s, sa, sb, output logic [2:0] snext);
	logic [31:0] o;
	decoder(s[2], s[1], s[0], sa, sb, o [31:0])
	assign snext[0] = o[0] || o[1] || o[8] || o[9] || o[10] || o[11] || o[16] || o[18] || o[24] || o[25]
		|| o[26] || o[27];
	assign snext[1] = o[4] || o[5] || o[6] || o[7] || o[8] || o[9] || o[10] || o[11] || o[20] || o[21] ||
		o[22] || o[23] || o[24] || o[25] || o[26] || o[27];
	assign snext[2] = o[12]  || o[13] || o[14] || o[15] || o[16] || o[17] || o[18] || o[19] || o[20] ||
		o[21] || o[22] || o[23] || o[24] || o[25] || o[26] || o[27];

endmodule

module decoder(input logic a, b, c, d, e, output logic [31:0] o);
	assign o[0] = (~a)&(~b)&(~c)&(~d)&(~e);
	assign o[1] = (~a)&(~b)&(~c)&(~d)&(e);
	assign o[2] = (~a)&(~b)&(~c)&(d)&(~e);
	assign o[3] = (~a)&(~b)&(~c)&(d)&(e);
	assign o[4] = (~a)&(~b)&(c)&(~d)&(~e);
	assign o[5] = (~a)&(~b)&(c)&(~d)&(e);
	assign o[6] = (~a)&(~b)&(c)&(d)&(~e);
	assign o[7] = (~a)&(~b)&(c)&(d)&(e);
	assign o[8] = (~a)&(b)&(~c)&(~d)&(~e);
	assign o[9] = (~a)&(b)&(~c)&(~d)&(e);
	assign o[10] = (~a)&(b)&(~c)&(d)&(~e);
	assign o[11] = (~a)&(b)&(~c)&(d)&(e);
	assign o[12] = (~a)&(b)&(c)&(~d)&(~e);
	assign o[13] = (~a)&(b)&(c)&(~d)&(e);
	assign o[14] = (~a)&(b)&(c)&(d)&(~e);
	assign o[15] = (~a)&(b)&(c)&(d)&(e);
	assign o[16] = (a)&(~b)&(~c)&(~d)&(~e);
	assign o[17] = (a)&(~b)&(~c)&(~d)&(e);
	assign o[18] = (a)&(~b)&(~c)&(d)&(~e);
	assign o[19] = (a)&(~b)&(~c)&(d)&(e);
	assign o[20] = (a)&(~b)&(c)&(~d)&(~e);
	assign o[21] = (a)&(~b)&(c)&(~d)&(e);
	assign o[22] = (a)&(~b)&(c)&(d)&(~e);
	assign o[23] = (a)&(~b)&(c)&(d)&(e);
	assign o[24] = (a)&(b)&(~c)&(~d)&(~e);
	assign o[25] = (a)&(b)&(~c)&(~d)&(e);
	assign o[26] = (a)&(b)&(~c)&(d)&(~e);
	assign o[27] = (a)&(b)&(~c)&(d)&(e);
	assign o[28] = (a)&(b)&(c)&(~d)&(~e);
	assign o[29] = (a)&(b)&(c)&(~d)&(e);
	assign o[30] = (a)&(b)&(c)&(d)&(~e);
	assign o[31] = (a)&(b)&(c)&(d)&(e);
endmodule
```

```
module green_function (input logic [2:0] s, output logic [2:0] la, [2:0] lb);
	four_to_one_mux mux1 (~s[0], 0, 0, 0, la[1]);
	four_to_one_mux mux2 (s[0], 0, 0, s[0], la[0]);
	four_to_one_mux mux3 (0, 0, ~s[0], 0, lb[1]);
	four_to_one_mux mux4 (0, s[0], s[0], 0, lb[0]);
	assign la[2] = la[1];
	assign la[1] = (~la[1]&la[0])||(la[1]& ~la[0]);
	assign la[0] = 1;
	assign lb[2] = lb[1];
	assign lb[1] = (~lb[1]&lb[0])||(lb[1]& ~lb[0]);
	assign lb[0] = 1;
endmodule


module four_to_one_mux(input logic [3:0] i, [1:0] sel, output logic o);
	assign out = sel[1] ? (sel[0] ? i[3] : i[2]) : (sel[0] ? i[1] : i[0]);
endmodule


module trafficLight ( input logic clk_in, reset, sa, sb, output logic [2:0]la, output logic [2:0]lb);

	logic [2:0] state, nextstate;
	logic [2:0] s0;
	s0[0] = 0;
	s0[1] = 0;
	s0[2] = 0;
	logic clk_out;
	Clock_Divider clock_divider(clk_in, clk_out);

	always_ff@ (posedge clk_out, posedge reset)
		if (reset) state <= s0;
		else	state <= nextstate;

	always @(*)
		blue_function (state, sa, sb, nextstate);

	always @(*)
		green_funtion (state, la, lb);
endmodule

module ClockDivider( input clk_in, output clk_out );
	logic [25:0] count = {26{1'b0}};
	logic clk_NoBuf;
	always@ (posedge clk_in) begin
		count <= count + 1;
	end
	// you can modify 25 to have different clock rate
```

```
	assign clk_NoBuf = count[25];
	BUFG BUFG_inst (
		.I(clk_NoBuf), // 1-bit input: Clock input
		.O(clk_out) // 1-bit output: Clock output
	);
endmodule
```