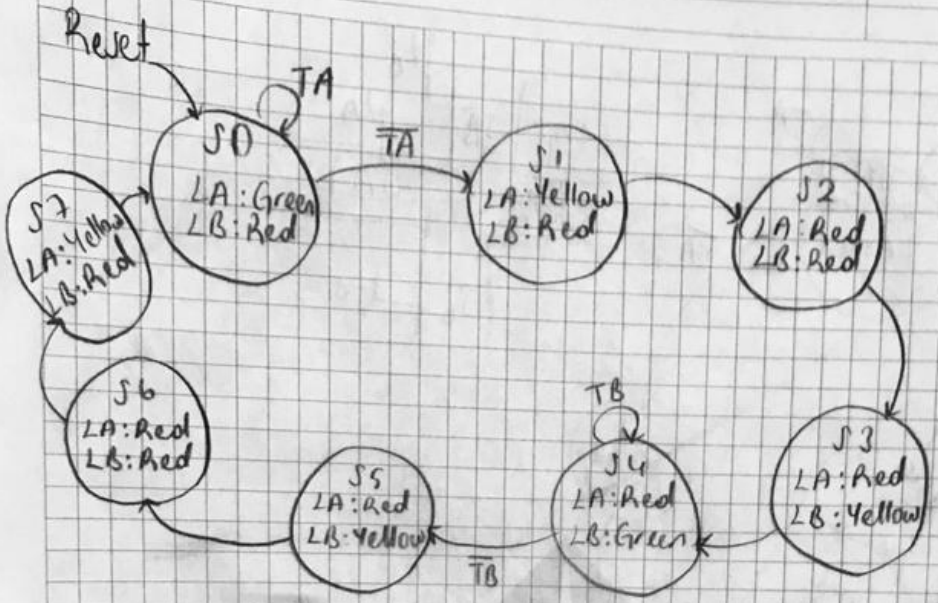
Selen Görgün

21802674

CS 223-006

Lab 4

a)



Current State S_2 S_1 S_0	Inputs $\overline{TA}$ $\overline{TB}$	Next State S_2' S_1' S_0'
0 0 0	1 X	0 0 0
0 0 0	0 X	0 0 1
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 1	1 0 0
1 0 0	X 0	1 0 1
1 0 1	X X	1 1 0
1 1 0	X X	1 1 1
1 1 1	X X	0 0 0

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

$S_2' = S_2\overline{S_1}\,\overline{S_0} + S_2\overline{S_1}S_0 + S_2S_1\overline{S_0} + \overline{S_2}S_1S_0$

$S_2' = S_2\overline{S_1} + S_1(S_2 \oplus S_0)$

$S_1' = \overline{S_2}S_1S_0 + S_2\overline{S_1}S_0 + S_2\overline{S_1}\,\overline{S_0} + S_2S_1\overline{S_0}$

$S_1' = S_1\overline{S_0} + \overline{S_1}S_0$

$S_1' = S_1 \oplus S_0$

$S_0' = \overline{S_0}(S_1 + \overline{S_2}\,\overline{TA} + S_2\,\overline{TB})$

$S_0' = \overline{S_2}\,\overline{S_1}\,\overline{S_0}\,\overline{TA} + \overline{S_2}S_1\overline{S_0} + S_2\overline{S_1}\,\overline{S_0}\,\overline{TB} + S_2S_1\overline{S_0}$

$S_0' = \overline{S_2}\,\overline{S_0}(\overline{S_1}\,\overline{TA} + S_1) + S_2\overline{S_0}(\overline{TB}\,\overline{S_1} + S_1)$

$S_0' = \overline{S_2}\,\overline{S_0}(S_1 + \overline{TA}) + S_2\overline{S_0}(S_1 + \overline{TB})$

$S_0' = \overline{S_0}(S_1\overline{S_2} + \overline{S_2}\,\overline{TA} + S_1S_2 + \overline{TB}S_2)$

Output Table

Binary Encoding	Output
00	Red
01	Green
10	Yellow

Current State S_2 S_1 S_0	Outputs LA_1	LA_0	LB_1	LB_0
0 0 0	0	1	0	0
0 0 1	1	0	0	0
0 1 0	0	0	0	0
0 1 1	0	0	1	0
1 0 0	0	0	0	1
1 0 1	0	0	1	0
1 1 0	0	0	0	0
1 1 1	1	0	0	0

$LA_1 = \bar{S}_2 \bar{S}_1 S_0 + S_2 S_1 S_0$

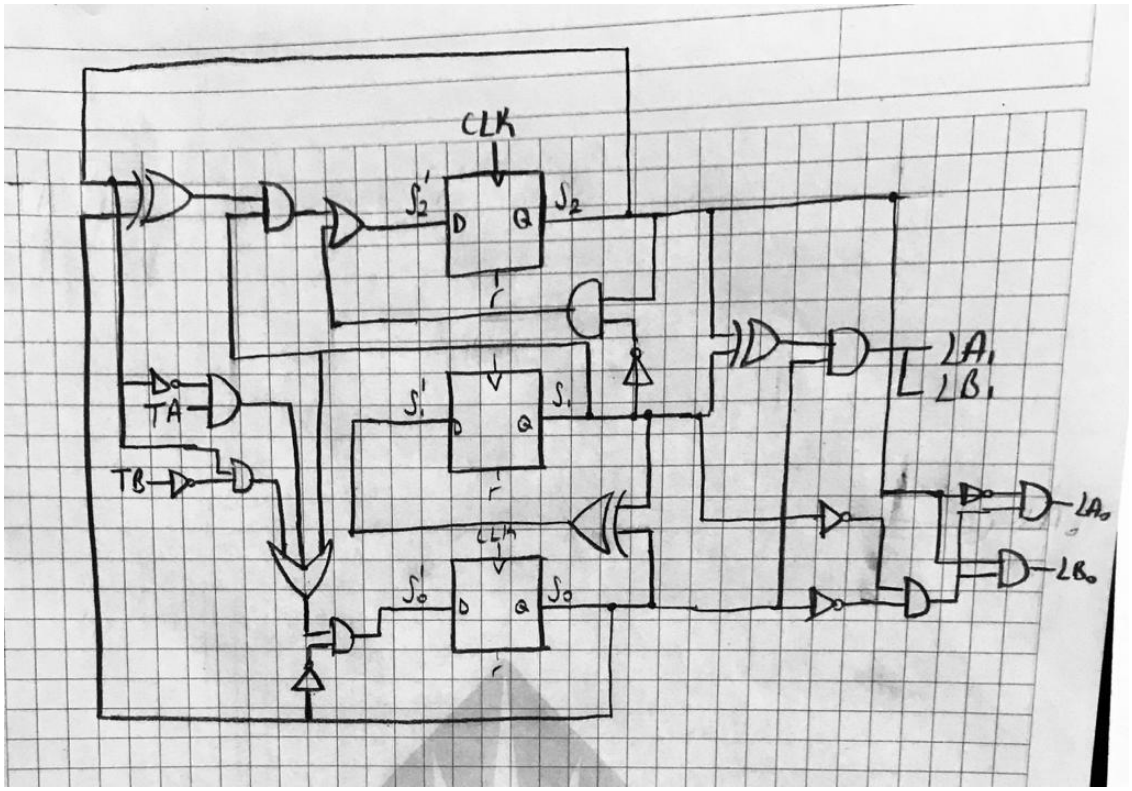
$LA_1 = S_0 (\overline{S_2 \oplus S_1})$

$LA_0 = \bar{S}_2 \bar{S}_1 \bar{S}_0$

$LB_1 = \bar{S}_2 S_1 S_0 + S_2 \bar{S}_1 S_0$

$LB_1 = S_0 (S_2 \oplus S_1)$

$LB_0 = S_2 \bar{S}_1 \bar{S}_0$



b) 3 flip-flops are needed to solve the problem.

c) 3 x 10^8 ns

d)

Clock Divider Module

```
module clock_divider( input clk, output logic out_clk);
    localparam max = 300000000;
    logic [31:0] count;

    always @ (posedge(clk))
    begin
        if (count == max - 1)
            count <= 32'b0;
        else
            count <= count + 1;
    end

    always @ (posedge(clk))
```

```
    begin
        if (count == max - 1)
            out_clk <= ~out_clk;
        else
            out_clk <= out_clk;
    end
endmodule
```

Light Controller Module

```
module lightController(input logic reset, clk,
            input logic Ta, Tb,
            output logic La1, La0, Lb1, Lb0);
    typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6, S7} statetype;
statetype state, nextState;

    always_ff@(posedge clk, posedge reset)
if(reset) state <= S0;
else      state <= nextState;

always_comb
case(state)
  S0: if(Ta == 0)nextState = S1;
      else nextState = S0;
  S1: nextState = S2;
  S2: nextState = S3;
  S3: nextState = S4;
  S4: if(Tb == 0) nextState = S5;
      else nextState = S4;
  S5: nextState = S6;
  S6: nextState = S7;
  S7: nextState = S0;
```

```
  default: nextState = S0;
endcase

    assign La1 = (state == S1) | (state == S7);
    assign La0 = (state == S0);
    assign Lb1 = (state == S3) | (state == S5);
    assign Lb0 = (state == S4);
endmodule
```

Testbench for controller

```
module controller_test();
logic Ta, Tb;
logic La1, La0, Lb1, Lb0;
logic reset,clk;
lightController lc(reset, clk, Ta, Tb, La1, La0, Lb1, Lb0);

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

always begin
reset = 1; #10;
assert(La1 === 0 & La0 === 1 & Lb1 === 0 & Lb0 === 0);

    reset = 0; Ta = 1; #10;
assert(La1 === 0 & La0 === 1 & Lb1 === 0 & Lb0 === 0);

Ta = 0; #10;
assert(La1 === 1 & La0 === 0 & Lb1 === 0 & Lb0 === 0);
```

```
Ta = 1'bx; Tb = 1'bx;#10;
assert(La1 === 0 & La0 === 0 & Lb1 === 0 & Lb0 === 0);

Ta = 1'bx; Tb = 1'bx;#10;
assert(La1 === 0 & La0 === 0 & Lb1 === 1 & Lb0 === 0);

Ta = 1'bx; Tb = 1'bx;#10;
assert(La1 === 0 & La0 === 0 & Lb1 === 0 & Lb0 === 1);

Tb = 1; #10;
assert(La1 === 0 & La0 === 0 & Lb1 === 0 & Lb0 === 1);

Tb = 0; #10;
assert(La1 === 0 & La0 === 0 & Lb1 === 1 & Lb0 === 0);

Ta = 1'bx; Tb = 1'bx;#10;
assert(La1 === 0 & La0 === 0 & Lb1 === 0 & Lb0 === 0);

Ta = 1'bx; Tb = 1'bx;#10;
assert(La1 === 1 & La0 === 0 & Lb1 === 0 & Lb0 === 0);

Ta = 1'bx; Tb = 1'bx;#10;
assert(La1 === 0 & La0 === 1 & Lb1 === 0 & Lb0 === 0);

end

endmodule
```

Module for Controller with clock

```
module ControllerWithClock(input logic reset, clk,
                input logic Ta, Tb,
```

```
        output logic LedsA2, LedsA1, LedsA0, LedsB2, LedsB1, LedsB0);
logic out_clk;
logic La1, La0, Lb1, Lb0;
assign La2 = 1;
assign Lb2 = 1;
clock_divider clkk(clk, out_clk);
lightController lc(reset, out_clk, Ta, Tb, La1, La0, Lb1, Lb0);

always @(La1, La0)
begin
if(La1 == 0 & La0 == 0)
begin
LedsA2 = 1;
LedsA1 = 1;
LedsA0 = 1;
end
else if(La1 == 0 & La0 == 1)
begin
LedsA2 = 0;
LedsA1 = 1;
LedsA0 = 1;
end
else if(La1 == 1 & La0 == 0)
begin
LedsA2 = 0;
LedsA1 = 0;
LedsA0 = 1;
end
end
```

```
    always@ (Lb1, Lb0)
    begin
    if(Lb1 == 0 & Lb0 == 0)
    begin
    LedsB2 = 1;
    LedsB1 = 1;
    LedsB0 = 1;
    end
    else if(Lb1 == 0 & Lb0 == 1)
    begin
    LedsB2 = 0;
    LedsB1 = 1;
    LedsB0 = 1;
    end
    else if(Lb1 == 1 & Lb0 == 0)
    begin
    LedsB2 = 0;
    LedsB1 = 0;
    LedsB0 = 1;
    end
    end

endmodule
```