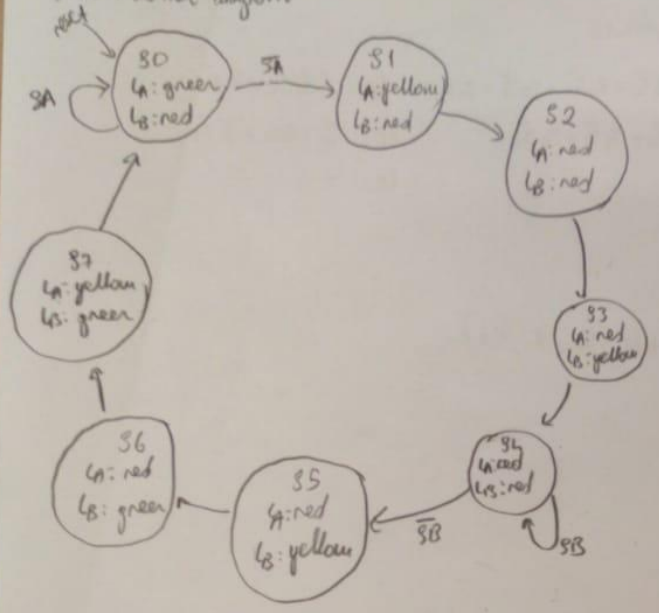
CS223-05 LAB-04

Remzi Tepe

21802713

23/11/2020

State transition diagram



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

Output encoding	
green	011
yellow	001
red	111

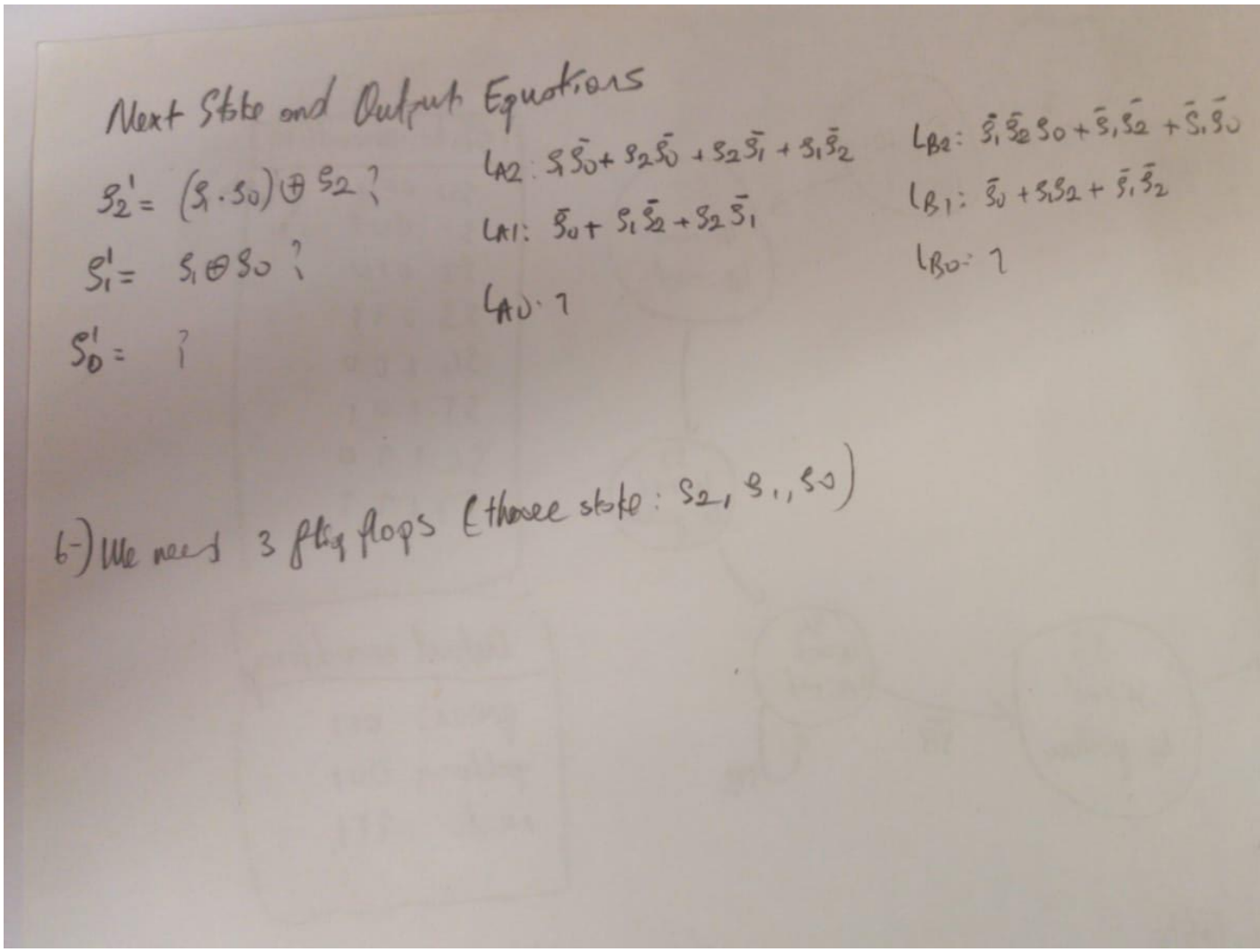
State Transition Table

Current State			Inputs		Next State		
S_2	S_1	S_0	SA	SB	S_2'	S_1'	S_0'
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 Table

Current State			Outputs					
S_2	S_1	S_0	L_{A2}	L_{A1}	L_{A0}	L_{B2}	L_{B1}	L_{B0}
0	0	0	0	1	1	1	1	1
0	0	1	0	0	1	1	1	1
0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	0	0	1
1	0	0	1	1	1	1	1	1
1	0	1	1	1	1	0	0	1
1	1	0	1	1	1	0	1	1
1	1	1	0	0	1	0	1	1

x: don't care



SystemVerilog code for trafficLights

```
module trafficLights(input logic SA,

                     input logic SB,

                     input logic clk,

                     input logic reset,

                     output logic ledA1, ledA0,
```

```
            output logic  ledB1, ledB0);

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

            statetype[1:0] state, nextstate;



always_ff @(posedge clk, posedge reset)

   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 = S0;
        default: nextstate = S0;
      endcase
  assign ledA1 = (state[2]^state[1]) | (state[1] & state[0]);
  assign ledA0 = state[0] & ((~state[2])^state[1]);
  assign ledB1 = (~state[2]^state[1]) | (~state[2] & state[0]);
  assign ledB0 = state[0] & ((state[2])^state[1]);
endmodule
```

SystemVerilog code for clock generator

```
module newClock(
  input  clk,
  input  rst,
  output logic clk_div
  );
  localparam constantNumber = 150000000;
```

```
    logic [31:0] count;



always @ (posedge(clk), posedge(rst))

begin

    if (rst == 1'b1)

        count <= 32'b0;

    else if (count == constantNumber - 1)

        count <= 32'b0;

    else

        count <= count + 1;

end



always @ (posedge(clk), posedge(rst))

begin

    if (rst == 1'b1)

        clk_div <= 1'b0;

    else if (count == constantNumber - 1)
```

```
      clk_div <= ~clk_div;

   else

      clk_div <= clk_div;

end




endmodule
```

testbench for traffic lights

```
module trafficLights_tb;



logic SA, SB, clk, reset, ledA1, ledA0, ledB1, ledB0;



trafficLights func (.SA(SA), .SB(SB), .clk(clk), .reset(reset), .ledA1(ledA1), .ledA0(ledA0),

                .ledB1(ledB1), .ledB0(ledB0));
```

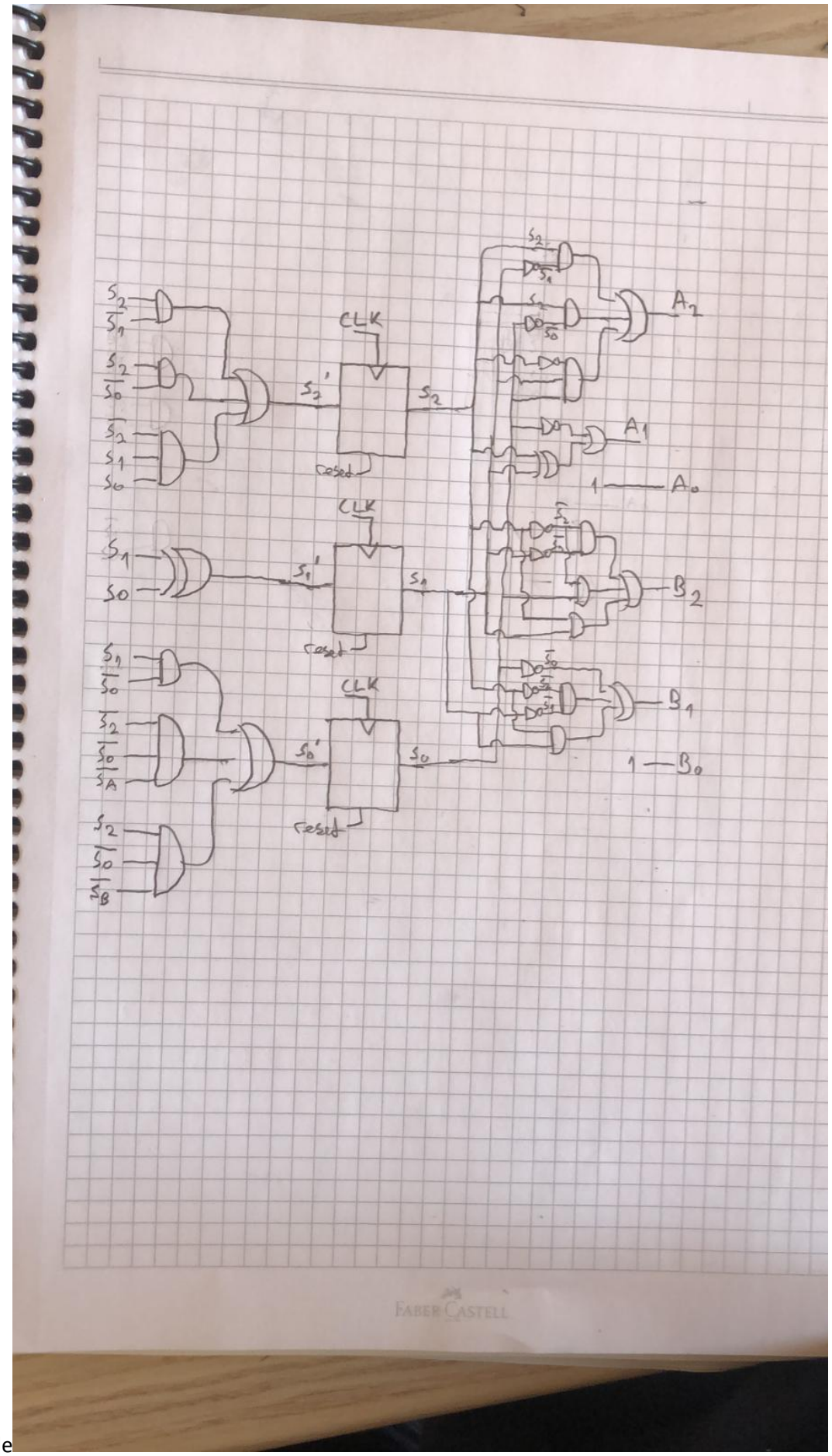
```
initial

 begin

   reset <= 1; #10;

   reset <= 0; SA <= 1; #10;

   SB <= 0; #10;

   SB <= 1; #10;

   SB <= 0; # 10;

   end



always begin

  clk <= 1; #5;

  clk <= 0; #5;

 end

endmodule
```

trafficlights top design systemverilog code

```
module trafficLights_top(input logic SA,
```

```
input logic SB,

input logic clk,

input logic reset,

output logic lightA2,

output logic lightA1,

output logic lightA0,

output logic lightB2,

output logic lightB1,

output logic lightB0);

logic clk_div;

logic ledA1, ledA0, ledB1, ledB0;

newClock( clk, reset, clk_div);

trafficLights( SA, SB, clk, reset, ledA1, ledA0, ledB1, ledB0);

assign lightA2 = ~ledA1 & ~ledA0;

assign lightA1 = ~ledA1;

assign lightA0 = 1;

assign lightB2 = ~ledB1 & ~ledB0;
```

```
        assign lightB1 = ~ledB1;

        assign lightB0 = 1;

endmodule
```



endmodule