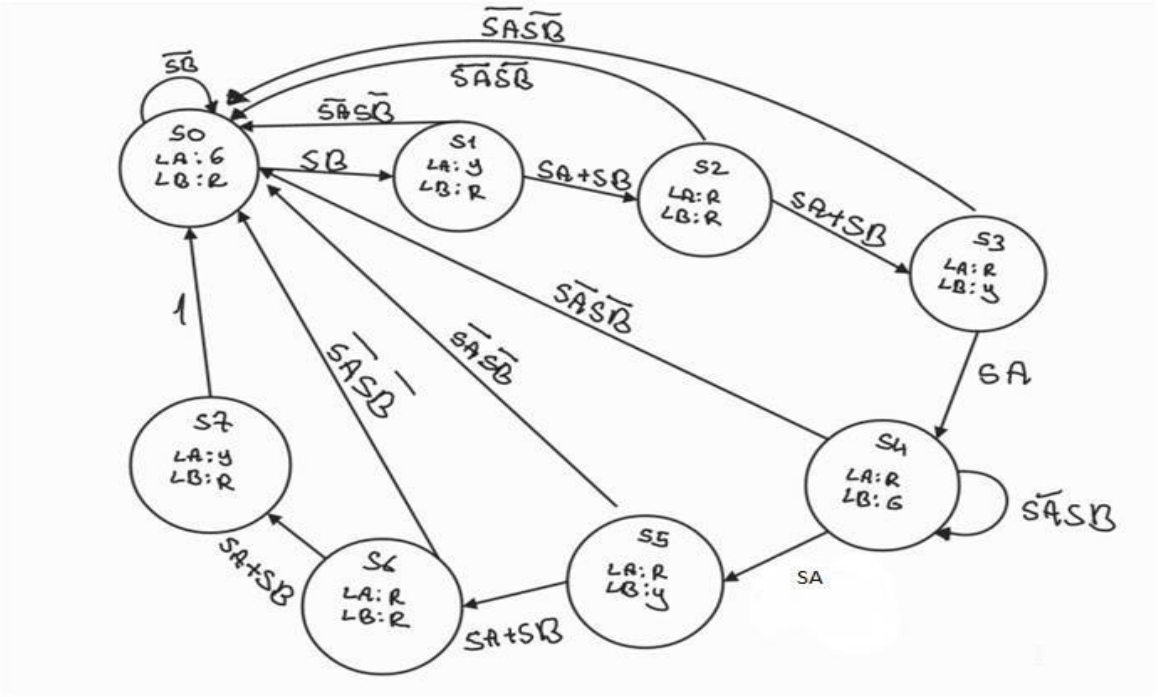
CS 223 Digital Design
Lab4 Preliminary Report

Ege Ipekci 21902333

15.05.2023

State Transition Diagram



*Since it wasn't possible to fit all states in the drawer program, I draw some parts on my tablet.

State Encodings

State	S2	S1	S0
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

State Transition Table

S2	S1	S0	SA	SB	S'2	S'1	S'0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	o	1
0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	1
0	0	1	0	0	0	0	0
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	0	1	0
0	1	0	0	0	0	0	0
0	1	0	0	1	0	1	1
0	1	0	1	0	0	1	1
0	1	0	1	1	0	1	1
0	1	1	0	0	0	0	0
0	1	1	0	1	1	0	0
0	1	1	1	0	1	0	0
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	1
1	0	0	1	1	1	0	1
1	0	1	0	0	0	0	0
1	0	1	0	1	1	1	0
1	0	1	1	0	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	0	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0

Output Encodings

State	LA1/LB1	LA0/LB0
RED	0	0
YELLOW	0	1
GREEN	1	0
X	1	1

Output Table

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

Code

```
module clockDivider(input logic clk, output logic outClk);
    logic [31:0] ctr = 1;
    logic temporaryClk = 0;
    always @ (posedge clk)
      begin
        if(ctr == 300000000)
          begin
             ctr <= 0;
            temporaryClk = ~temporaryClk;
          end
         else
          ctr <= ctr + 1;
       end
     assign outClk = temporaryClk;
 endmodule

module trafficsystem( input logic Sa, Sb, clk, reset,
                      output logic [ 5:0 ] TL );

   //logic clk, reset;
   logic outClk;
   //logic [ 1:0 ] La;
   //logic [ 1:0 ] Lb;
   typedef enum logic [ 2:0 ] { s0, s1, s2, s3, s4, s5, s6, s7 } statetype;
   statetype state, nextstate;
   //freq_divider fd( clk, newclk );

   clockDivider fd( clk,  outClk );



   always_ff @ ( posedge outClk /*posedge reset*/ )
   begin
      if( reset )
         state <= s0;
      else
         state <= nextstate;
   end
   always_comb
   begin
      case( state )
         s0: if ( ~Sb ) nextstate = s0;
            else nextstate = s1;
         s1: if( Sa || Sb) nextstate = s2;
            else nextstate = s0;
         s2: if( Sa || Sb) nextstate = s3;
            else nextstate = s0;
         s3: if( Sa || Sb) nextstate = s4;
            else nextstate = s0;
         s4: if( ~Sa && Sb ) nextstate = s4;
            else if( Sa ) nextstate = s5;
            else nextstate = s0;
         s5: if( Sa || Sb) nextstate = s6;
```

```
            else nextstate = s0;
         s6: if( Sa || Sb) nextstate = s7;
            else nextstate = s0;
         s7: nextstate = s0;
      endcase
   end

   always_comb
   begin
      case( state )
      s0: TL = 6'b011111;
      s1: TL = 6'b001111;
      s2: TL = 6'b111111;
      s3: TL = 6'b111001;
      s4: TL = 6'b111011;
      s5: TL = 6'b111001;
      s6: TL = 6'b111111;
      s7: TL = 6'b001111;
      endcase
   end
endmodule
```

Testbench

```
module testbench();

    logic Sa, Sb, clk, reset;
   logic  [5:0]TL;

    trafficsystem uut( Sa, Sb, clk, reset, TL );

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

  initial begin
  Sa = 1; Sb=0; #5;
   reset = 1;
    #100;
    reset = 0;
    #100;

    Sa=1; Sb=0;  #50;
    Sa=0; Sb=1; #50;
    Sa=1; ; #50;



    end

endmodule
```

Next State Equations

S’2 = S2 & ~S0 & SA || SA & S1 & S0 & ~S2 ||S2 & ~S1 & SA

|| S2 & ~S0 & SB || SB & S1 & S0 & ~S2 || S2 & ~S1 & SB

S’1 = (S1 ⊕ S0) & (SA || SB)

S’0 = S2 & ~S0 & SB || SA & S1 & ~S0 & S2 ||S2 & ~S0 & SA || S2 & ~S0 & SB

Output Equations

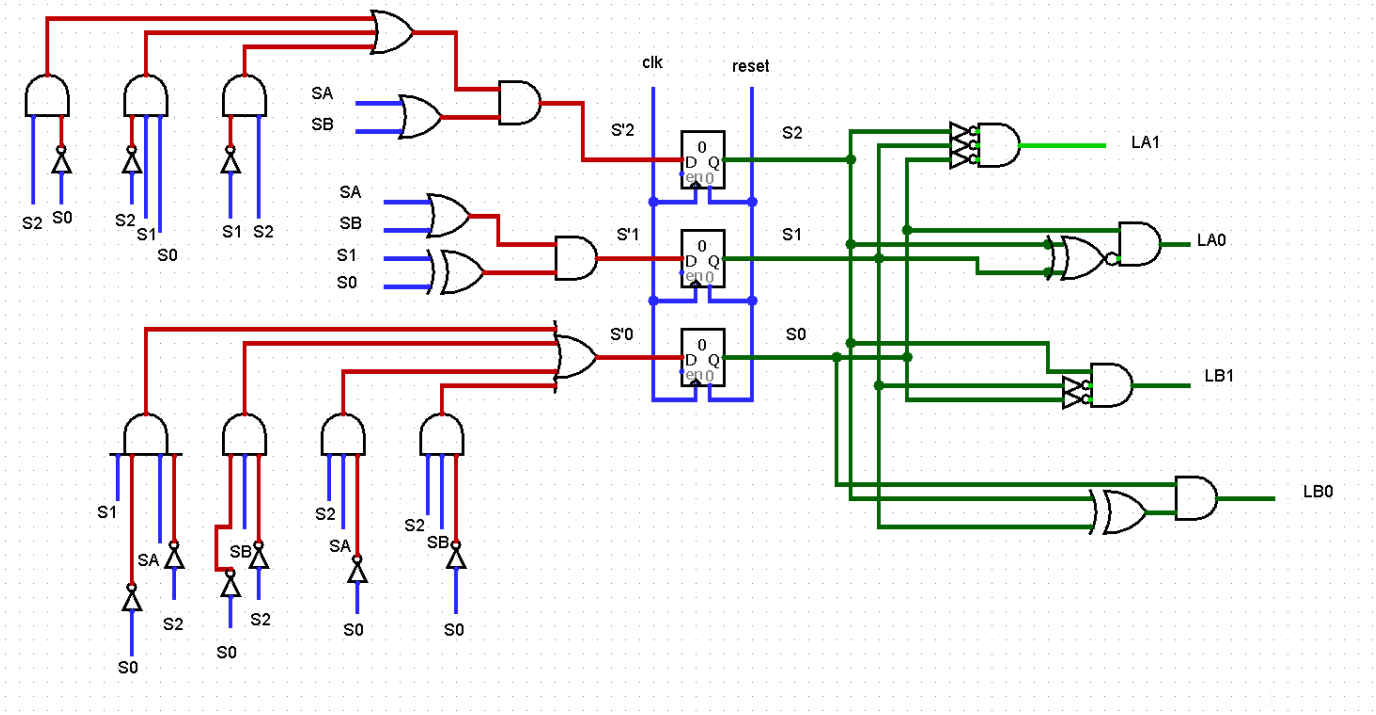
LA1 = ~S2 & ~S1 & ~S0

LA0 = S0 & ~ (S2 ⊕ S1)

LB1 = S2 & ~S1 & ~S0

LB0 =S0 & (S2 ⊕ S1)

Finite State Machine Schematic



Since there are 8 states, 3 flip flops would be sufficient to cover up all the possible states (2^3).

Traffic Light System with Decoders