30.04.2020

CS 223-4

LAB-5

PRELIMINARY REPORT

ZÜBEYİR BODUR

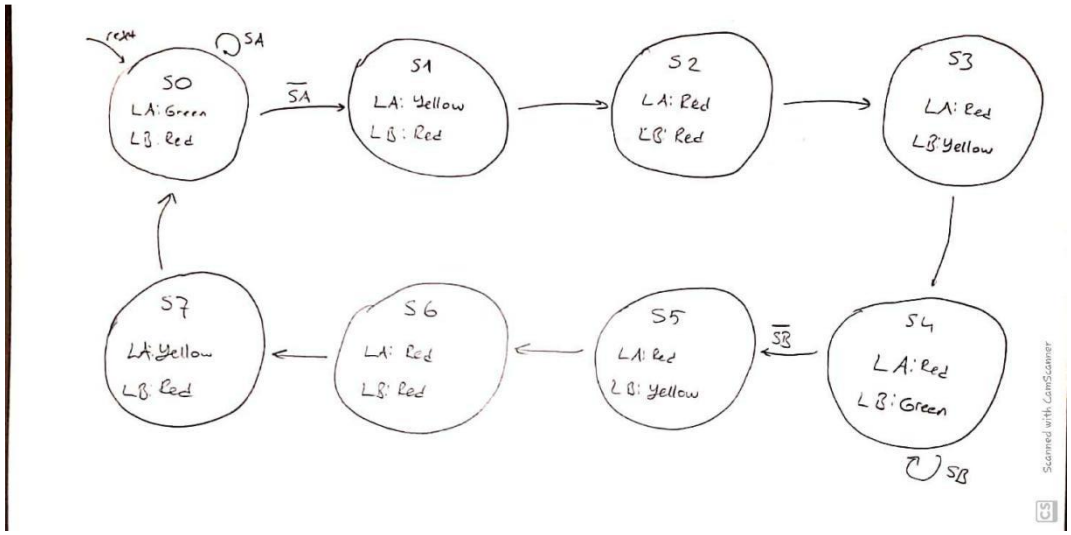
21702382

Spring 2020

1. Number of Flip-Flops

3 flip-flops are required for this FSM. Since we will have 8 states, we will have 3-bits (S_2 S_1 and S_0) to represent those states. To build a 3-bit register, 3 flip-flops will be needed.

2. State Transition Diagram



SA/SB : SA/SB sensor indicates TRUE
SA'/SB' : SA/SB sensor indicates FALSE

3. Binary Encodings

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

State Encodings

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

Output Encodings. Since LA and LB's encodings would be the same, LX is used to represent both LA and LB.

4. State Transition Table

Current State	SA	SB	Next State
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	X	S3
S3	X	X	S4
S4	X	0	S5
S4	X	1	S4
S5	X	X	S6
S6	X	X	S7
S7	X	X	S0

In this table, don't cares are used where a states goes to the same state after the clock edge, regardless of the input.

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

State transition table with encodings

5. 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

Output table with encodings

6. Boolean Equations

For next state logic, equations are;

$$S'_2 = \overline{S}_2 S_1 S_0 + S_2 \overline{S}_1 + S_2 S_1 \overline{S}_0$$

$$S'_1 = \overline{S}_1 S_0 + S_1 \overline{S}_0 = S_1 \oplus S_0$$

$$S'_0 = \overline{S}_2 \overline{S}_1 \overline{S}_0 \overline{S}A + \overline{S}_2 S_1 \overline{S}_0 + S_2 \overline{S}_1 \overline{S}_0 \overline{S}B + S_2 S_1 \overline{S}_0$$

And for the output logic, equations are;

$$LA_2 = 1$$

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

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

$$LB_2 = 1$$

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

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

7. System Verilog Module for the FSM

```
// Next state logic of the traffic light controller
module nextStateLogic(    input logic s_a, s_b,
                          input logic [2:0]s,
                          output logic [2:0]n_s
);
    logic [2:0] mint_2;
    logic [3:0] mint_0;

    and and1( mint_2[0], ~s[2], s[1], s[0] );
    and and2( mint_2[1], s[2], ~s[1] );
    and and3( mint_2[2], s[2], s[1], ~s[0] );

    or or1( n_s[2], mint_2[0], mint_2[1], mint_2[2] );

    xor xor1( n_s[1], s[1], s[0]);

    and and4( mint_0[0], ~s[2], ~s[1], ~s[0], ~s_a );
    and and5( mint_0[1], ~s[2], s[1], ~s[0] );
    and and6( mint_0[2], s[2], ~s[1], ~s[0], ~s_b );
    and and7( mint_0[3], s[2], s[1], ~s[0] );

    or  or2(n_s[0], mint_0[0], mint_0[1] ,mint_0[2], mint_0[3]);
endmodule

// Output logic of the traffic light controller
module outputLogic( input logic [2:0]s,
                    output logic [2:0]l_a, l_b
);
    and and1  ( l_a[2], 1);
    or  or1   ( l_a[1], ~s[2] & ~s[1], s[2] & s[1] & s[0]);
    and and2  ( l_a[0], ~s[2], ~s[1], ~s[0]);

    and and3  ( l_b[2], 1);
    or  or2   ( l_b[1], s[2] & ~s[1], ~s[2] & s[1] & s[0]);
    and and4  ( l_b[0], s[2], ~s[1], ~s[0]);
endmodule
```

```
// Module for clock divider to reduce CLK frequency from 100 Mhz
// to 0.5 Hz
module clkDivider( input logic[31:0] divider,
                   input logic clk, rst,
                   output logic clk_prime
);
    logic [31:0]count = 32'b0;

    always_ff @( posedge clk, posedge rst) begin
       if     ( rst == 1 )               count <= 32'b0;
       else if ( count >= divider - 1)   count <= 32'b0;
       else                              count <= count + 1;
    end



    always_ff @(posedge clk, posedge rst) begin
       if    (rst == 1)            clk_prime <= 32'b0;
       else if  (count == divider - 1) clk_prime <= ~clk_prime;
       else                            clk_prime <= clk_prime;
    end
endmodule

// Module for the FSM
module trafficLightController(  input logic s_a, s_b, clk, rst,
                                output logic [2:0]l_a, l_b
    );
       logic[2:0]s;
       logic[2:0]n_s;
       logic[31:0]divider = 32'b10111110101111000010000000;
       Logic clk_prime;

       nextStateLogic nextState(s_a, s_b, s, n_s);
       clkDivider divide(divider, clk, rst, clk_prime);

       always_ff@ (posedge clk_prime, posedge rst) begin
          if (rst)  s <= 3'b0;
          else      s <= n_s;
       end

       outputLogic out(s, l_a, l_b);
endmodule
```

8. Testbench for the FSM

```
module testTrafficLightController();

   logic s_a, s_b, clk, rst;
   logic [2:0]l_a;
   logic [2:0]l_b;

   trafficLightController test( s_a, s_b, clk, rst, l_a, l_b);
   initial begin
      s_a = 0; s_b = 0; clk = 0; rst = 1; #1; rst = 0;
      repeat (2) begin
         repeat (2) begin
            repeat (2) begin
               clk = ~clk; #10;
            end
            s_b = ~s_b; #10;
         end
         s_a = ~s_a; #10;
      end
      $stop;
   end
endmodule
```