CS223: Digital Design

Section 01

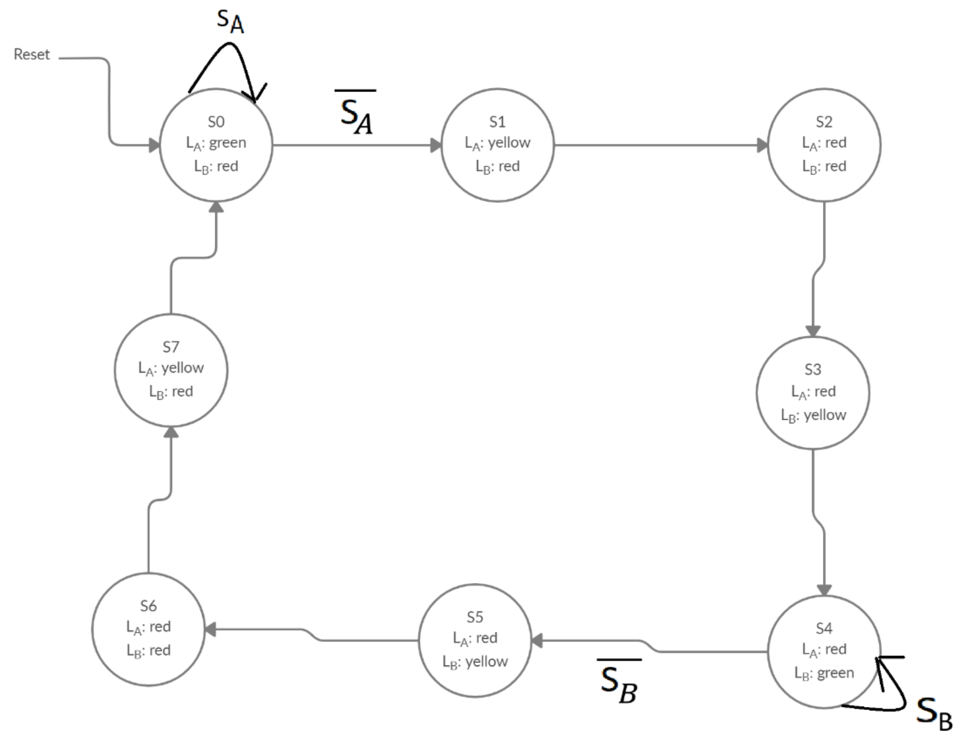
Lab 4

Efe Beydoğan

21901548

28.11.2020

a) Sketch your improved Moore state machine transition diagram, state encodings, state transition table, output table, next state and output equations and your Finite State Machine schematic:



Reset
S_A
$\overline{S_A}$
S0 L_A: green L_B: red
S1 L_A: yellow L_B: red
S2 L_A: red L_B: red
S3 L_A: red L_B: yellow
S4 L_A: red L_B: green
S5 L_A: red L_B: yellow
S6 L_A: red L_B: red
S7 L_A: yellow L_B: red
$\overline{S_B}$
S_B

state	encoding $s_{1:0}$
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

current state			inputs		next state		
s_2	s_1	s_0	s_a	s_b	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	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

Output	Encoding $L_{2:0}$
Green	011
Yellow	001
Red	111

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	0	1	1
1	0	1	1	1	1	0	0	1
1	1	0	1	1	1	1	1	1
1	1	1	0	0	1	1	1	1

<u>Output Equations:</u>

$L_{a2} = (S_1 \oplus S_2) + S_1\overline{S}_0$

$L_{a1} = (S_1 \oplus S_2) + \overline{S}_0$

$L_{a0} = 1$

$L_{b2} = (\overline{S_2 \oplus S_1}) + \overline{S}_2\ \overline{S}_0$

$L_{b1} = (\overline{S_2 \oplus S_1}) + \overline{S}_0$

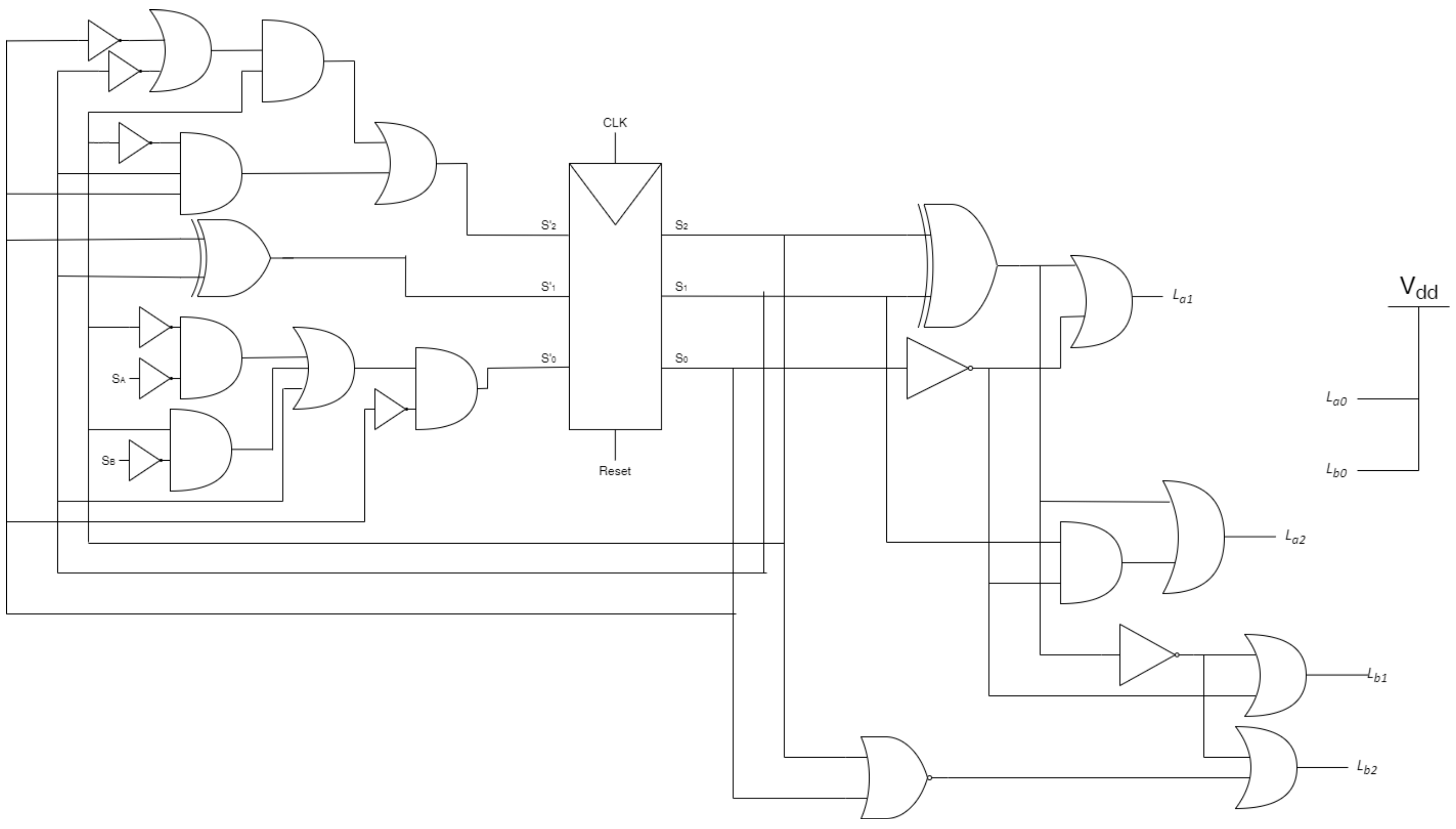
$L_{b0} = 1$

<u>Next State Equations:</u>

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

$S_1' = S_1 \oplus S_0$

$S_0' = \overline{S}_0\ (S_1 + \overline{S}_2\overline{S}_A + S_2\overline{S}_B)$



b) How many flip-flops are needed to implement this problem?

Since there are 3 state bits, 3 flip-flops are needed to implement this problem.

c) How many rising edges do you need to count from the Basys3 clock in order to obtain such frequency? Write the SystemVerilog code that will count original clock cycles output HIGH in every 3 seconds.

The clock changes state 10^8 times in a second, so it changes state 3 x 10^8 times in 3 seconds. Half of those state changes are rising edges where the clock goes from 0 to 1. So, we need to count $\frac{3 x 10^8}{2}$ = 15 x 10^7 rising edges from the BASYS3 clock to obtain a 3 second delay.

SystemVerilog code for the clock:

```
module clock_counter( input clk, input reset, output logic clock);

localparam number = 150000000;

   logic [31:0] count;

   always @ (posedge(clk), posedge(reset))

   begin

      if (reset == 1'b1)

         count <= 32'b0;

      else if (count == number - 1)

         count <= 32'b0;

      else

         count <= count + 1;

    end

  always @ (posedge(clk), posedge(reset))

   begin

      if (reset == 1'b1)

         clock <= 1'b0;

      else if (count == number - 1)

         clock <= ~clock;

      else

         clock <= clock;

   end

endmodule
```

<u>d) Write the SystemVerilog code for your traffic light system and a testbench for it:</u>

<u>SystemVerilog code for the traffic light system:</u>

```
module traffic_light_system( input logic clk, input logic reset, input logic Sa, Sb, output logic [2:0] La,
Lb);

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

   statetype state, nextstate;


   // state register

   always_ff @(posedge clk, posedge reset)

   if (reset) state <= S0;

   else state <= nextstate;


   // next state logic

   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 = S6;

         S6: nextstate = S7;

         S7: nextstate = S0;

         default: nextstate = S0;

   endcase
```

```
  // output logic
  logic n0, n1;

  xor( n0, state[1], state[2]);

  assign La[2] = n0 | (state[1] & ~state[0]);
  assign La[1] = n0 | ~state[0];
  assign La[0] = 1;

  assign Lb[2] = ~n0 | (~state[2] & ~state[0]);
  assign Lb[1] = ~n0 | ~state[0];
  assign Lb[0] = 1;
endmodule
```

<u>Testbench:</u>

```
module testbench_traffic();
  logic clk, reset, Sa, Sb;
  logic [2:0] La, Lb;

  // instantiate device under test
  traffic_light_system dut(clk, reset, Sa, Sb, La, Lb);

  // apply inputs one at a time
  always
    begin
      clk = 1; #10;
      clk = 0; #10;
    end
  initial begin
    reset = 1; Sa = 1; Sb = 0; #10;
    reset = 0; #100;
    Sb = 1; #100;
    Sa = 0; #100;
    Sa = 0; Sb = 0; #100;
  end
endmodule
```

e) Write the SystemVerilog code for the top design where you will combine the new clock generator in part c and traffic light system in part d:

```
module clock_fsm(input logic clk, reset, Sa, Sb, output logic [2:0] La, Lb);
  logic clock;
  clock_counter clockdiv(clk, reset, clock);
  traffic_light_system fsm(clock, reset, Sa, Sb, La, Lb);
endmodule
```