DIGITAL DESIGN CS223

SECTION 02

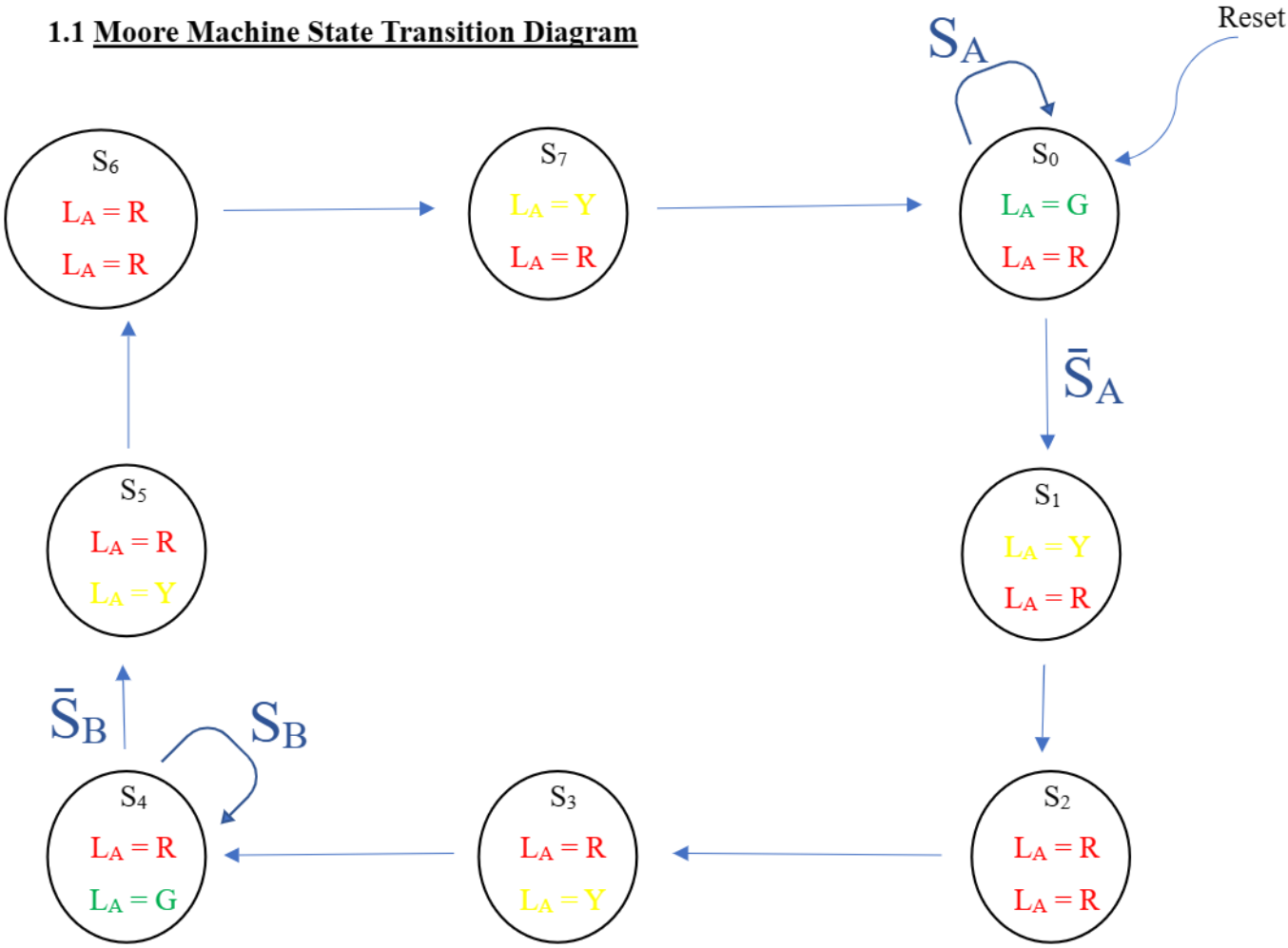
LAB 04

Sümeyye ACAR

22103640

16.05.2023

1.1 Moore Machine State Transition Diagram



1.2 State Encoding

S_2	S_1	S_0	State
0	0	0	S_0
0	0	1	S_1
0	1	0	S_2
0	1	1	S_3
1	0	0	S_4
1	0	1	S_5
1	1	0	S_6
1	1	1	S_7

1.3 State Transition Table

Current State			Input		Next State		
S_2	S_1	S_0	S_A	S_B	N_2	N_1	N_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

1.4 Output Table

State			Output (L_A)			Output (L_B)		
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

111 = RED

011 = GREEN

001 = YELLOW

1.5 Equations

Next State:

$N_2 = \bar{S}_2 S_1 S_0 + S_2 \bar{S}_1 + S_2 S_1 \bar{S}_0$

$N_1 = S_1 \oplus S_0$

$N_0 = \bar{S}_2 \bar{S}_1 \bar{S}_0 \bar{S}_A + S_2 \bar{S}_1 \bar{S}_0 \bar{S}_B + S_1 \bar{S}_0$

Output:

$L_{A2} = S_2 S_1 \bar{S}_0 + (S_2 \oplus S_1)$

$L_{A1} = \bar{S}_0 + (S_2 \oplus S_1)$

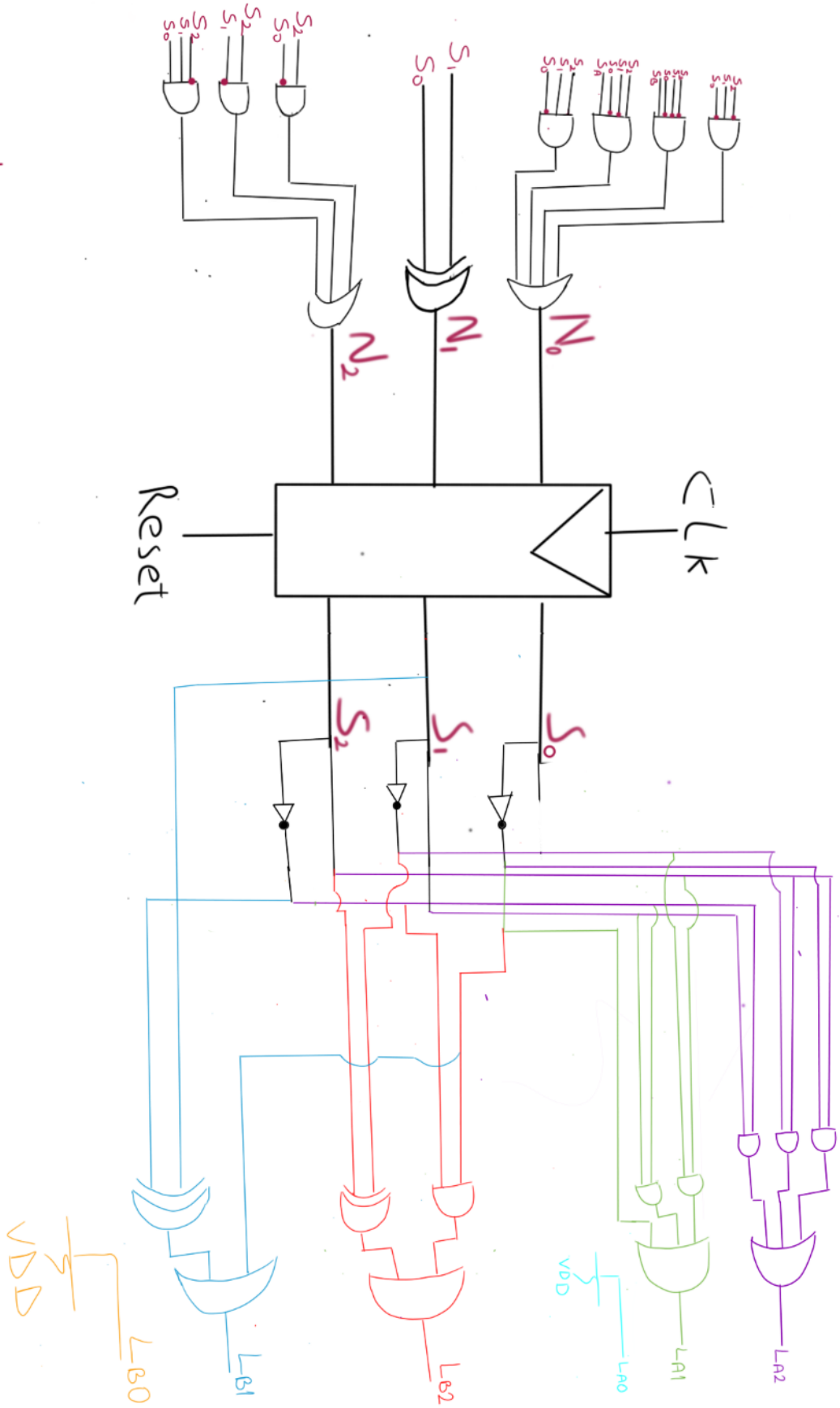
$L_{A0} = 1$

$L_{B2} = (S_2 \oplus \bar{S}_1) + \bar{S}_2 S_1 \bar{S}_0$

$L_{B1} = \bar{S}_0 + (\bar{S}_2 \oplus S_1)$

$L_{B0} = 1$

2.0 Finite State Machine Schematic



3.0 Number of Flip-Flops

The number of needed flip-flops is **3** because the number of states which is 8. 8 states can be represented with 3-bit binary numbers; so, 3 flip-flops will be needed.

4.0 SystemVerilog Design Code

4.1 FSM

```
module lab4_fsm (input logic clk, sa, sb, reset,
                 output logic [2:0] la, logic [2:0] lb);

  // Slowed Clock
  //logic slower;
  //ClockDiv CD(.clk(clk), .slow(slower));

  //Possible States
  typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6, S7} states; // all states are 3 bit
  states [2:0] current, next;

  // Lights
  parameter red_stop    = 3'b111;
  parameter green_go    = 3'b011;
  parameter yellow_wait = 3'b001;

  // Reset (CHANGE clk WITH slower TO SLOW THE CLOCK)!!!
  always_ff @(posedge clk, posedge reset)
  if (reset)
    current <=  S0;
  else
    current <= next;

  // State Transition
  always_comb
    case (current)
      S0:
        if (sa)
          next = S0;
        else
          next = S1;
      S1:
        next = S2;
      S2:
        next = S3;
      S3:
        next = S4;
      S4:
```

```
        if (sb)
            next = S4;
        else
            next = S5;
    S5:
        next = S6;
    S6:
        next = S7;
    S7:
        next = S0;
  endcase


// Output - State Matching
always_comb
  case (current)
    S0:
      begin
        la =   green_go;
        lb =   red_stop;
      end
    S1:
      begin
        la = yellow_wait;
        lb =   red_stop;
      end
    S2:
      begin
        la =   red_stop;
        lb =   red_stop;
      end
    S3:
      begin
        la =   red_stop;
        lb = yellow_wait;
      end
    S4:
      begin
        la =   red_stop;
        lb =   green_go;
      end
    S5:
      begin
        la =   red_stop;
        lb = yellow_wait;
      end
    S6:
```

```
            begin
               la =    red_stop;
               lb =    red_stop;
            end
         S7:
            begin
               la = yellow_wait;
               lb =    red_stop;
            end
      endcase
endmodule
```

4.2 The Clock

```
module ClockDiv( input logic clk, output logic slow );

   // Clock Counter
   logic [28:0] counter;

   always @(  posedge clk  )
      begin
         if(  counter >= 199999999  )
            begin
               counter <= 0;
               slow    <= 0;
            end
         else
            begin
               counter <= counter + 1;
               slow    <=          1;
            end
      end

endmodule
```

5.0 SystemVerilog Testbench

```
module testBench();

   // Parameters of the Finite State Machine
   logic clk, reset, sa, sb;
   logic [2:0] LA, LB;

   // Device Under Test
  lab4_fsm dut(.clk(clk), .sa(sa), .sb(sb), .reset(reset), .la(LA), .lb(LB));
```

```
    // Clock Signal
    always
        begin
            clk <= 1;           #5;
            clk <= 0;           #5;
        end

    initial
        begin
            reset = 1;          #100; // beginnig

            reset = 0;
            sa = 0; sb = 0;     #100;
            sa = 0; sb = 1;     #100;
            sa = 1; sb = 0;     #100;
            sa = 1; sb = 1;     #100;

            reset = 1;
            sa = 0; sb = 0;     #40;
            sa = 0; sb = 1;     #40;
            sa = 1; sb = 0;     #40;
            sa = 1; sb = 1;     #40;
            $finish;
        end

endmodule
```