DIGITAL DESIGN

CS223

LAB-04 PRELIMINARY REPORT

NAME: MUHAMMAD ARHAM KHAN

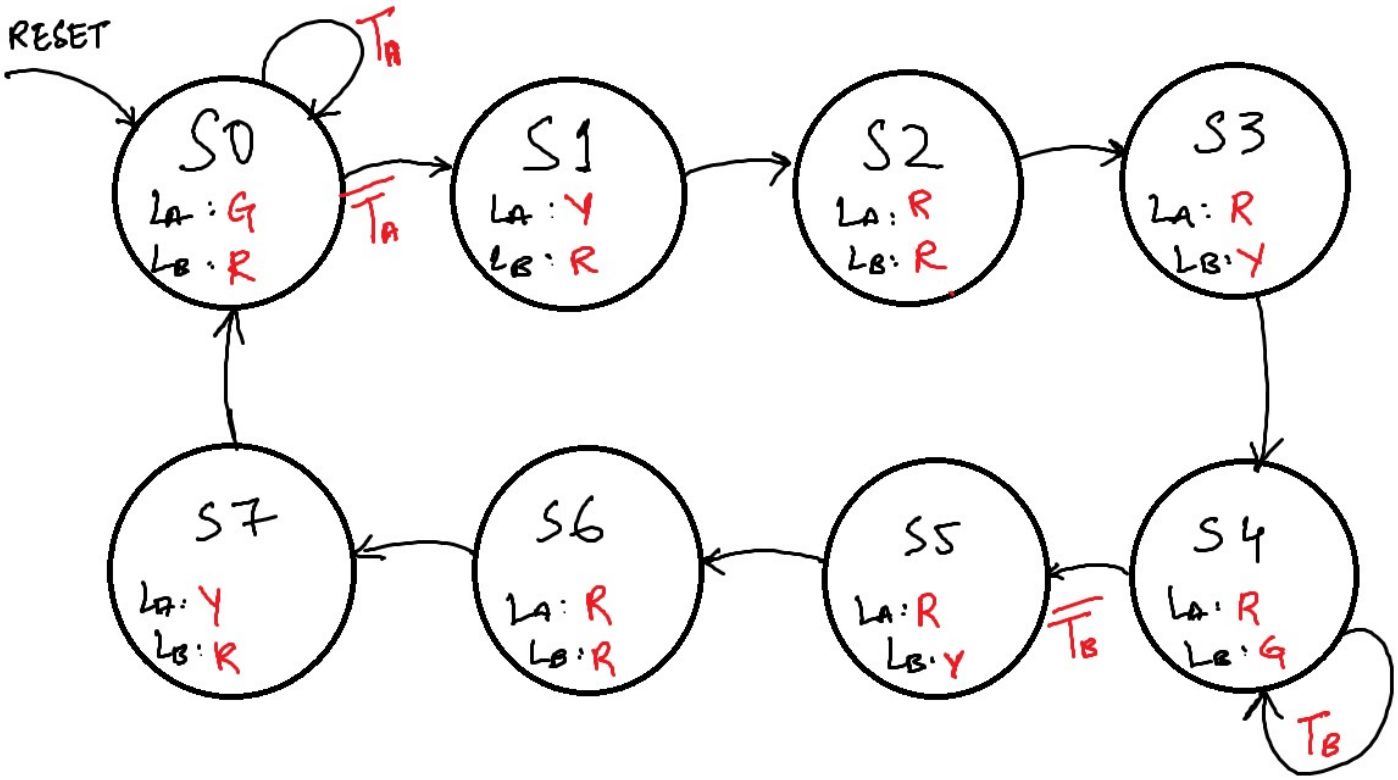
BILKENT ID: 21701848

SECTION: 2

DATE: 26 NOV, 2018

TRAINING PACK: 47

MOORE FINITE STATE MACHINE (FSM) TRANSITION DIAGRAM



STATE ENCODINGS

STATE	ENCODING $S_{2:0}$		
	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

OUTPUT ENCODING

OUTPUT	ENCODING	
	L_1	L_0
RED	0	0
YELLOW	0	1
GREEN	1	0

OUTPUT TABLE

STATE	L_A		L_B	
	L_{A1}	L_{A0}	L_{B1}	L_{B0}
S0	1	0	0	0
S1	0	1	0	0
S2	0	0	0	0
S3	0	0	0	1
S4	0	0	1	0
S5	0	0	0	1
S6	0	0	0	0
S7	0	1	0	0

STATE TRANSITION TABLE

CURRENT STATE			INPUTS		NEXT STATE		
S_2	S_1	S_0	T_A	T_B	S_2	S_1	S_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

NEXT STATE EQUATIONS

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

$$S_1' = S_1\overline{S}_0 + \overline{S}_1S_0$$

$$S_2' = S_2\overline{S}_1 + S_2\overline{S}_0 + S_1S_0\overline{S}_2$$

OUTPUT EQUATIONS

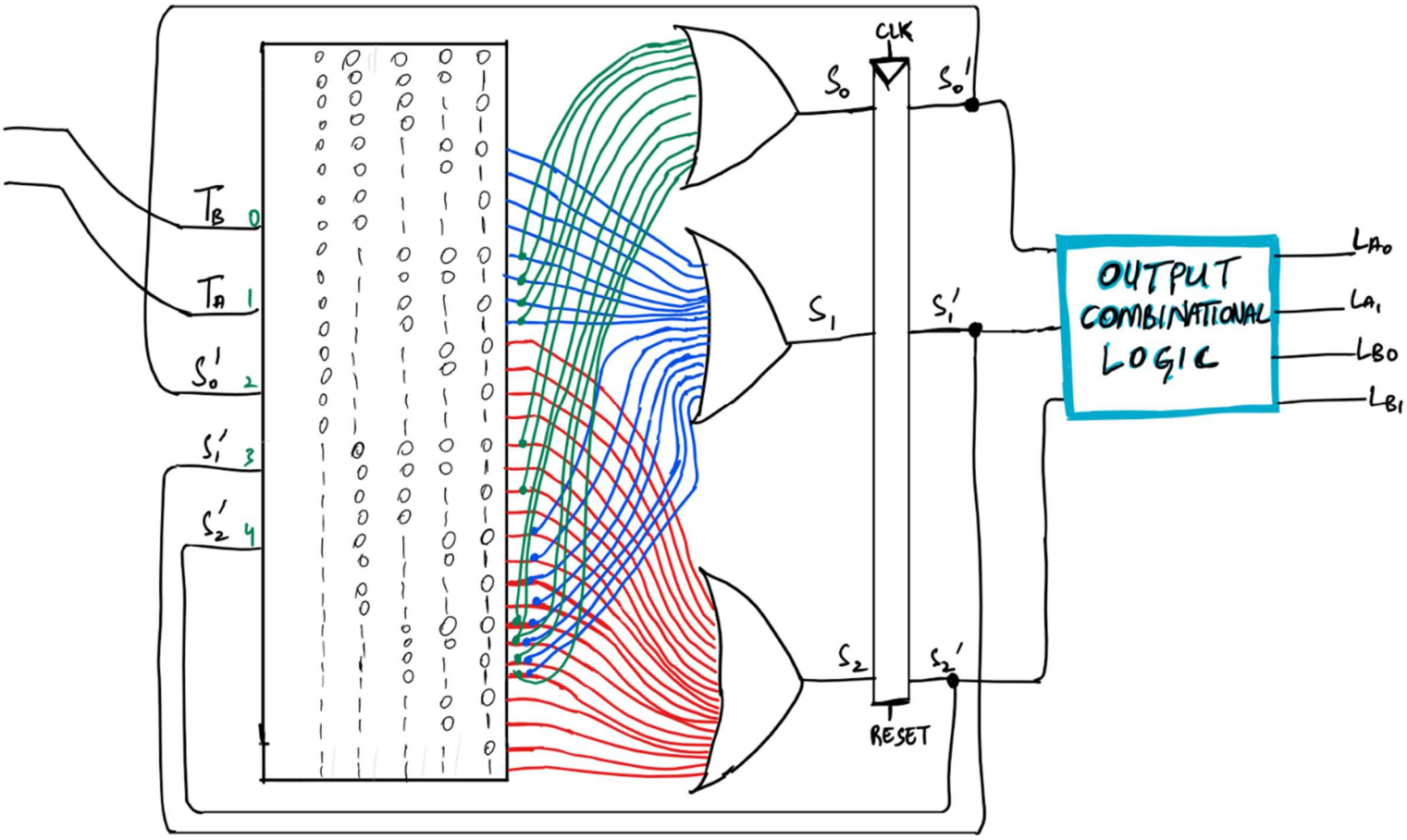
$$L_{A0} = \overline{S}_2\overline{S}_1S_0 + S_2S_1S_0$$

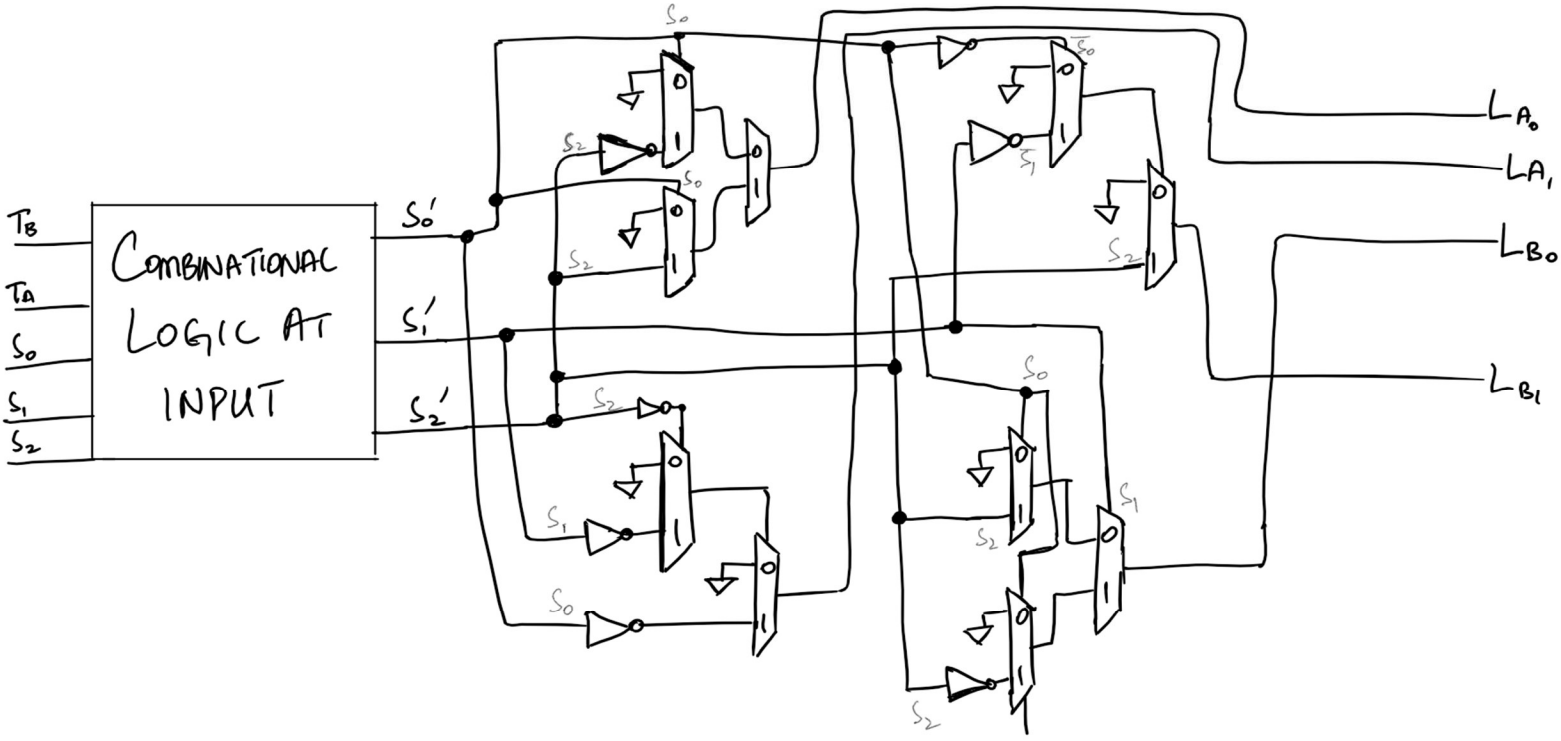
$$L_{A1} = \overline{S}_2\overline{S}_1\overline{S}_0$$

$$L_{B0} = \overline{S}_2S_1S_0 + S_2\overline{S}_1S_0$$

$$L_{B1} = S_2\overline{S}_1\overline{S}_0$$

FSM CIRCUIT SCHEMATIC

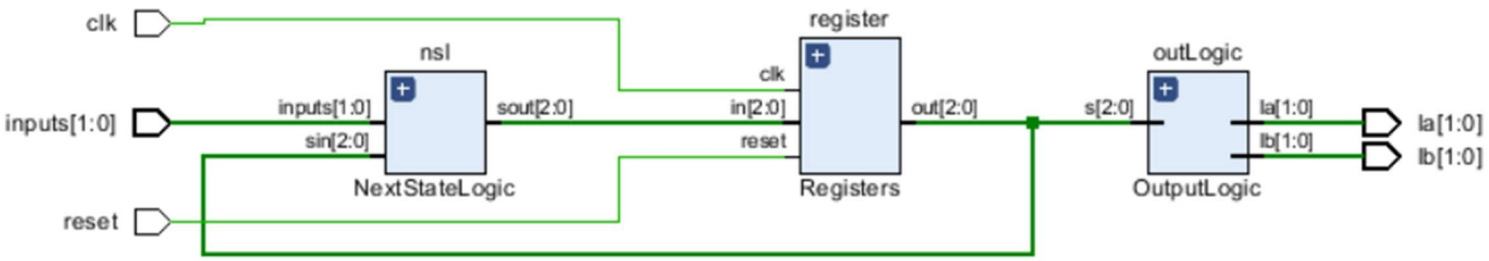




HOW MANY FLIP-FLOPS ARE NEEDED?

Since there are total 8 states and using the binary encoding, we have three bits used to store the 8 states, we will need to use 3 Flip-Flops for the implementation.

DETAILED CIRCUIT DESIGN



TRAFFIC LIGHTS (SYSTEM VERILOG)

```
module TrafficLights( input logic clk, reset,
                      input logic[1:0] inputs,
                      output logic[1:0] la, lb);
    logic[2:0] state;
    logic[2:0] nextState;
    Registers register( clk, reset, nextState, state);
    NextStateLogic nsl( state, inputs, nextState);
    OutputLogic outLogic( state, la, lb);
endmodule
```

NEXT STATE LOGIC (SYSTEM VERILOG)

```
module NextStateLogic( input logic[2:0] sin,
                       input logic[1:0] inputs,
                       output logic[2:0] sout);
    logic[31:0] decoderOut;
    logic[4:0] aa;

    assign aa = { sin[2], sin[1], sin[0], inputs[1], inputs[0]};
    Decoder dec( aa, decoderOut);
    assign sout[0] = decoderOut[0] || decoderOut[1] || decoderOut[8] || decoderOut[9] ||
                decoderOut[10] || decoderOut[11] || decoderOut[16] || decoderOut[18] ||
                    decoderOut[24] || decoderOut[25] || decoderOut[26] || decoderOut[27];
    assign sout[1] = decoderOut[4] || decoderOut[5] || decoderOut[6] || decoderOut[7] ||
                decoderOut[8] || decoderOut[9] || decoderOut[10] ||decoderOut[11] ||
                    decoderOut[20] || decoderOut[21] || decoderOut[22] ||decoderOut[23] ||
                        decoderOut[24] || decoderOut[25] || decoderOut[26] ||decoderOut[27];
    assign sout[2] = decoderOut[12] || decoderOut[13] || decoderOut[14] || decoderOut[15] ||
                decoderOut[16] || decoderOut[17] || decoderOut[18] || decoderOut[19] ||
                    decoderOut[20] || decoderOut[21] || decoderOut[22] || decoderOut[23] ||
                        decoderOut[24] || decoderOut[25] || decoderOut[26] || decoderOut[27];

endmodule
```

OUTPUT LOGIC (SYSTEM VERILOG)

```
module OutputLogic( input logic[2:0] s,
                    output logic[1:0] la, lb);

    logic la01, la02;
    logic la11;
    logic lb01, lb02;
    logic lb11;

    Mux LA01( 0, ~s[2], s[0], la01);
    Mux LA02(0, s[2], s[0], la02);
    Mux LA03( la01, la02, s[1], la[0]);

    Mux LA11( 0, ~s[1], ~s[2], la11);
    Mux LA12( 0, ~s[0], la11, la[1]);

    Mux LB01( 0, s[2], s[0], lb01);
    Mux LB02( 0, ~s[2], s[0], lb02);
    Mux LB03( lb01, lb02, s[1], lb[0]);

    Mux LB11( 0, ~s[1], ~s[0], lb11);
    Mux LB12( 0, s[2], lb11, lb[1]);

endmodule
```

32-BIT DECODER (SYSTEM VERILOG)

```
module Decoder( input logic[4:0] a,
                output logic[31:0] out);
    always_comb
        case( a)
            5'b00000:   out=32'b00000000000000000000000000000001;
            5'b00001:   out=32'b00000000000000000000000000000010;
            5'b00010:   out=32'b00000000000000000000000000000100;
            5'b00011:   out=32'b00000000000000000000000000001000;
            5'b00100:   out=32'b00000000000000000000000000010000;
            5'b00101:   out=32'b00000000000000000000000000100000;
            5'b00110:   out=32'b00000000000000000000000001000000;
            5'b00111:   out=32'b00000000000000000000000010000000;
```

```
            5'b01000:   out=32'b00000000000000000000000100000000;
            5'b01001:   out=32'b00000000000000000000001000000000;
            5'b01010:   out=32'b00000000000000000000010000000000;
            5'b01011:   out=32'b00000000000000000000100000000000;
            5'b01100:   out=32'b00000000000000000001000000000000;
            5'b01101:   out=32'b00000000000000000010000000000000;
            5'b01110:   out=32'b00000000000000000100000000000000;
            5'b01111:   out=32'b00000000000000001000000000000000;
            5'b10000:   out=32'b00000000000000010000000000000000;
            5'b10001:   out=32'b00000000000000100000000000000000;
            5'b10010:   out=32'b00000000000001000000000000000000;
            5'b10011:   out=32'b00000000000010000000000000000000;
            5'b10100:   out=32'b00000000000100000000000000000000;
            5'b10101:   out=32'b00000000001000000000000000000000;
            5'b10110:   out=32'b00000000010000000000000000000000;
            5'b10111:   out=32'b00000000100000000000000000000000;
            5'b11000:   out=32'b00000001000000000000000000000000;
            5'b11001:   out=32'b00000010000000000000000000000000;
            5'b11010:   out=32'b00000100000000000000000000000000;
            5'b11011:   out=32'b00001000000000000000000000000000;
            5'b11100:   out=32'b00010000000000000000000000000000;
            5'b11101:   out=32'b00100000000000000000000000000000;
            5'b11110:   out=32'b01000000000000000000000000000000;
            5'b11111:   out=32'b10000000000000000000000000000000;
            default:    out=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
        endcase
endmodule
```

REGISTERS (SYSTEM VERILOG)

```
module Registers( input logic clk, reset,
                  input logic[2:0] in,
                  output logic[2:0] out);
    always_ff@(posedge clk)
        if( reset) out <= 3'b0;
        else       out <= in;
endmodule
```

TRAFFIC LIGHTS TEST BENCH

```
module TrafficLightTB();
       logic clk, reset;
       logic[1:0] inputs;
       logic[1:0] la, lb, laExpected, lbExpected;
       logic [31:0] vectornum, errors;
       logic[5:0] testvectors[31:0];



       TrafficLights t( clk, reset, inputs, la, lb);


       //generate clock
       always
          begin
               clk = 1; #5;
               clk = 0; #5;
       end


   //load vectors
   initial
        begin
            $readmemb( "example.tv", testvectors);
            vectornum = 0; errors = 0;
            reset = 1; #5; reset = 0;
        end
    //apply test vector on rising edge of clock
    always @( posedge clk)
        begin
            #1; { inputs[1], inputs[0], laExpected[1], laExpected[0], lbExpected[1],
lbExpected[0]} = testvectors[vectornum];
        end


    //checking results on falling edge
    always @( negedge clk)
        if( ~reset) begin
            if((la[0] !== laExpected[0]) || (la[1] !== laExpected[1]) ||
                    (lb[0] !== lbExpected[0]) || (lb[1] !== lbExpected[1])) begin


                $display( "Error: inputs = %b", {inputs[1], inputs[0]});
```

```
                $display( " outputs: la = %b and lb = %b (expected: la = %b and lb = %b)",
{la[1], la[0]}, {lb[1], lb[0]}, {laExpected[1], laExpected[0]}, {lbExpected[1],
lbExpected[0]});

                errors = errors + 1;

            end

            if((la[0] === laExpected[0]) && (la[1] === laExpected[1]) &&

                                (lb[0] === lbExpected[0]) && (lb[1] === lbExpected[1])) begin


                $display( "Worked: inputs = %b", {inputs[1], inputs[0]});

                $display( " outputs: la = %b and lb = %b (expected: la = %b and lb = %b)",
{la[1], la[0]}, {lb[1], lb[0]}, {laExpected[1], laExpected[0]}, {lbExpected[1],
lbExpected[0]});

            end

            vectornum = vectornum + 1;

            if( testvectors[ vectornum] === 6'bx) begin

                $display( "%d tests completed with %d errors", vectornum, errors);

                $finish;

            end

        end

endmodule
```

TRAFFIC LIGHTS TEST BENCH

```
set_property PACKAGE_PIN W5 [get_ports clk_in]

set_property IOSTANDARD LVCMOS33 [get_ports clk_in]

create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk_in]


# Switches

set_property PACKAGE_PIN V17 [get_ports {inputs[1]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {inputs[1]}]

set_property PACKAGE_PIN V16 [get_ports {inputs[0]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {inputs[0]}]

set_property PACKAGE_PIN W16 [get_ports {reset}]

    set_property IOSTANDARD LVCMOS33 [get_ports {reset}]



# LEDs

set_property PACKAGE_PIN U16 [get_ports {la[0]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {la[0]}]

set_property PACKAGE_PIN E19 [get_ports {la[1]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {la[1]}]

set_property PACKAGE_PIN U19 [get_ports {la[2]}]

      set_property IOSTANDARD LVCMOS33 [get_ports {la[2]}]
```

```
set_property PACKAGE_PIN V19 [get_ports {lb[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {lb[0]}]
set_property PACKAGE_PIN W18 [get_ports {lb[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {lb[1]}]
set_property PACKAGE_PIN U15 [get_ports {lb[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {lb[2]}]
```