DIGITAL DESIGN 4th LAB

PRELIMINARY REPORT

Course Code: CS 223

Course Name: Digital Design

Section: 1

Name: Arda

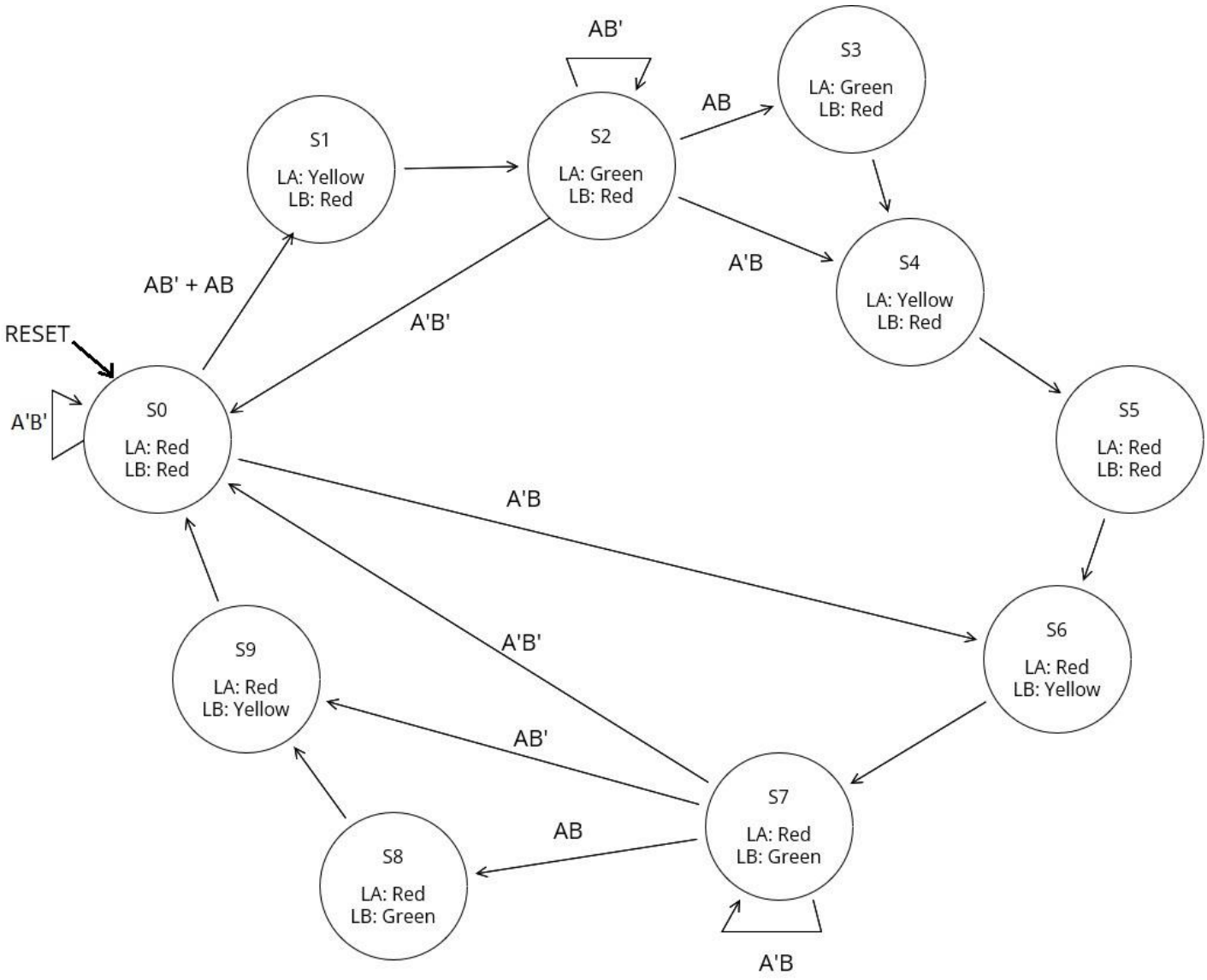
Surname: İynem

Student ID: 22002717

Date: 03.04.2022

STATE TRANSITION DIAGRAM

(a) Moore Machine State Transition Diagram



NOTE: SA and SA inputs are represented with A and B labels in the Moore State Transition Diagram for the sake of simplicity.

STATE ENCODINGS

(a) State and Output Encodings

Current State S	Encoding $S_{3:0}$
S0	0000
S1	0001
S2	0010
S3	0011
S4	0100
S5	0101
S6	0110
S7	0111
S8	1000
S9	1001

Output	Encoding $L_{1:0}$
Green	00
Yellow	01
Red	10

STATE TRANSITION TABLE

(a) State Transition Table

Current State S	Inputs		Next State S’
	SA	SB	
S0	1	X	S1
S0	0	0	S0
S0	0	1	S6
S1	X	X	S2
S2	0	0	S0
S2	0	1	S4
S2	1	0	S2
S2	1	1	S3
S3	X	X	S4
S4	X	X	S5
S5	X	X	S6
S6	X	X	S7
S7	0	0	S0
S7	0	1	S7
S7	1	0	S9
S7	1	1	S8
S8	X	X	S9
S9	X	X	S0

STATE TRANSITION TABLE

USING BINARY ENCODINGS

(a) State Transition Table with Binary Encodings

Current State S				Inputs		Next State S'			
S_3	S_2	S_1	S_0	S_A	S_B	S'_3	S'_2	S'_1	S'_0
0	0	0	0	1	X	0	0	0	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	1	0
0	0	0	1	X	X	0	0	1	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0	1	0
0	0	1	0	1	1	0	0	1	1
0	0	1	1	X	X	0	1	0	0
0	1	0	0	X	X	0	1	0	1
0	1	0	1	X	X	0	1	1	0
0	1	1	0	X	X	0	1	1	1
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	0	1
0	1	1	1	1	1	1	0	0	0
1	0	0	0	X	X	1	0	0	1
1	0	0	1	X	X	0	0	0	0

OUTPUT TABLE

(a) Output Table

Current State S	Output	
	L_A	L_B
S0	Red	Red
S1	Yellow	Red
S2	Green	Red
S3	Green	Red
S4	Yellow	Red
S5	Red	Red
S6	Red	Yellow
S7	Red	Green
S8	Red	Green
S9	Red	Yellow

(a) Output Table with Binary Encodings

Current State S				Outputs			
S_3	S_2	S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0	1	0
0	0	0	1	0	1	1	0
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1

BOOLEAN EQUATIONS

NEXT STATE EQUATIONS
OUTPUT EQUATIONS

(a) Next State Equations

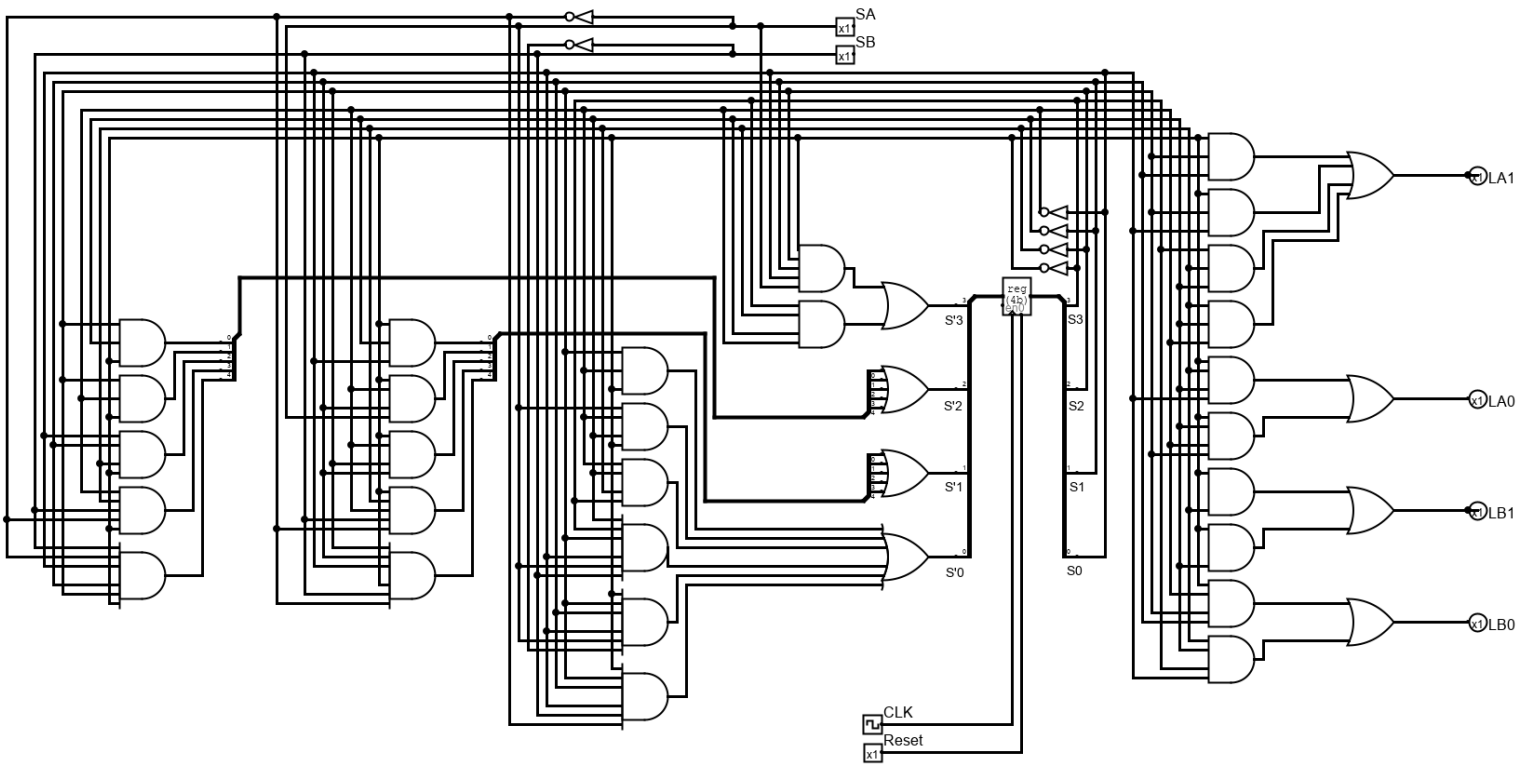
S'_3	$\bar{S}_3 S_2 S_1 S_0 A + S_3 \bar{S}_2 \bar{S}_1 \bar{S}_0$
S'_2	$\bar{S}_3 \bar{S}_2 \bar{S}_0 \bar{A} B + \bar{S}_3 \bar{S}_2 S_1 S_0 + \bar{S}_3 S_2 \bar{S}_0 + \bar{S}_3 S_2 \bar{S}_1 + \bar{S}_3 S_2 S_1 S_0 \bar{A} B$
S'_1	$\bar{S}_3 \bar{S}_2 \bar{S}_0 \bar{A} B + \bar{S}_3 \bar{S}_1 S_0 + \bar{S}_3 S_1 \bar{S}_0 A + \bar{S}_3 S_2 S_1 \bar{S}_0 + \bar{S}_3 S_2 S_1 S_0 \bar{A} B$
S'_0	$\bar{S}_3 \bar{S}_1 \bar{S}_0 A + \bar{S}_3 S_2 \bar{S}_0 + S_3 S_2 \bar{S}_1 S_0 A B + \bar{S}_3 S_2 S_1 S_0 A \bar{B} + \bar{S}_3 S_2 S_1 S_0 \bar{A} B + S_3 \bar{S}_2 \bar{S}_1 \bar{S}_0$

(a) Output Equations

L_{A1}	$\bar{S}_3 S_2 S_1 + \bar{S}_3 S_2 S_0 + S_3 \bar{S}_2 \bar{S}_1 + \bar{S}_2 \bar{S}_1 \bar{S}_0$
L_{A0}	$\bar{S}_3 \bar{S}_2 \bar{S}_1 S_0 + \bar{S}_3 S_2 \bar{S}_1 \bar{S}_0$
L_{B1}	$\bar{S}_3 \bar{S}_2 + \bar{S}_3 \bar{S}_1$
L_{B0}	$\bar{S}_3 S_2 S_1 \bar{S}_0 + S_3 \bar{S}_2 \bar{S}_1 S_0$

SCHEMATICS

(b) Finite State Machine Schematic



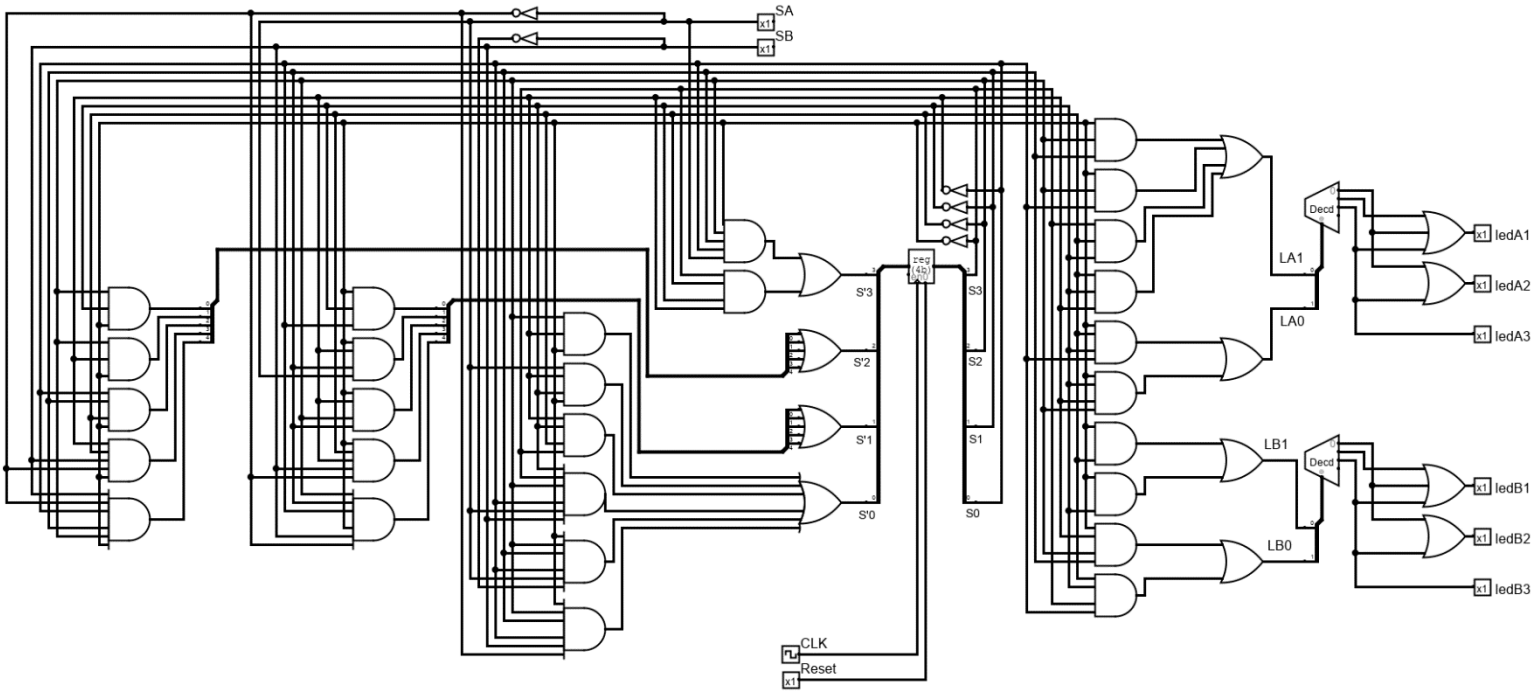
NOTE: “CLK” and “Reset” are two inputs connected to the register, in addition to SA and SB inputs. CLK is the clock signal with 3 seconds period whereas Reset input sets all current state bits to the 0.

(c) How many flip-flops do you need to implement this problem?

Since the register consists of flip flops for each input bit and 4 bits are needed to represent corresponding bit numbers for 10 states (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9), 4 Flip Flops are needed to implement this problem with a circuit.

SCHEMATICS

(d) Redesign of Outputs using Decoders.



SYSTEM VERILOG

DESIGN CODE

(e) SystemVerilog Design with outputs LA1, LA0, LB1, LB0

```
module fsm_module(clk, reset, sa, sb, la1, la0,
lb1, lb0);
   input logic clk, reset, sa, sb;
   output logic la1, la0, lb1, lb0;

   typedef enum logic [3:0] {S0 = 4'b0000 , S1 =
4'b0001, S2 = 4'b0010, S3 = 4'b0011, S4 =
4'b0100,
                          S5 = 4'b0101, S6 = 4'b0110,
S7 = 4'b0111, S8 = 4'b1000, S9 = 4'b1001}
statetype;
   statetype state, nextstate;
   parameter green = 2'b00;
   parameter yellow = 2'b01;
   parameter red = 2'b10;

   // 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 = S1;
         else if (~sb) nextstate = S0;
         else nextstate = S6;
      S1: nextstate = S2;
      S2: if(~sa & ~sb) nextstate = S0;
         else if (~sa & sb) nextstate = S4;
         else if (sa & ~sb) nextstate = S2;
         else nextstate = S3;
      S3: nextstate = S4;
      S4: nextstate = S5;
```

```
// Code Continues
      S5: nextstate = S6;
      S6: nextstate = S7;
      S7: if(~sa & ~sb) nextstate = S0;
         else if (~sa & sb) nextstate = S7;
         else if (sa & ~sb) nextstate = S9;
         else nextstate = S8;
      S8: nextstate = S9;
      S9: nextstate = S0;
      default: nextstate = S0;
      endcase


   // output logic
   always_comb
      case (state)
      S0: {la1, la0, lb1, lb0} = {red, red};
      S1: {la1, la0, lb1, lb0} = {yellow, red};
      S2: {la1, la0, lb1, lb0} = {green, red};
      S3: {la1, la0, lb1, lb0} = {green, red};
      S4: {la1, la0, lb1, lb0} = {yellow, red};
      S5: {la1, la0, lb1, lb0} = {red, red};
      S6: {la1, la0, lb1, lb0} = {red, yellow};
      S7: {la1, la0, lb1, lb0} = {red, green};
      S8: {la1, la0, lb1, lb0} = {red, green};
      S9: {la1, la0, lb1, lb0} = {red, yellow};
      default: {la1, la0, lb1, lb0} = {red, red};
      endcase
endmodule
```

SYSTEM VERILOG

TESTBENCH CODE

(e) SystemVerilog Testbench with Outputs LA1, LA0, LB1, LB0

```
module tb_fsm();
    logic clk, reset, sa, sb;
    logic la1, la0, lb1, lb0;

    fsm_module dut(clk, reset, sa, sb, la1, la0, lb1, lb0);


    always
        begin
            clk = 0; #1500000000; clk = 1; #1500000000;
        end

    initial
        begin
        reset <= 0; sa <= 0; sb <= 0; @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        @(posedge clk);
        reset <= 1; sa <= 0; sb <= 0; @(posedge clk);
        reset <= 0; sa <= 1; sb <= 1; @(posedge clk);
        @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        @(posedge clk);
        sa <= 0; sb <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        sa <= 0; sb <= 0; @(posedge clk);
        sa <= 1; sb <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        sa <= 0; sb <= 1; @(posedge clk);
        @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        $stop; // End the simulation.
        end
endmodule
```

SYSTEM VERILOG

DESIGN CODE
REDESIGNED OUTPUTS

(e) SystemVerilog Design with Outputs ledA1, ledA2, ledA3, ledB1, ledB2, ledB3 using Decoder

```
module fsm_module(clk, reset, sa, sb, ledA1, ledA2,
ledA3, ledB1, ledB2, ledB3);
   input logic clk, reset, sa, sb;
   output logic ledA1, ledA2, ledA3, ledB1, ledB2,
ledB3;
   logic la1, la0, lb1, lb0;

   typedef enum logic [3:0] {S0 = 4'b0000 , S1 =
4'b0001, S2 = 4'b0010, S3 = 4'b0011, S4 = 4'b0100,
                          S5 = 4'b0101, S6 = 4'b0110, S7 =
4'b0111, S8 = 4'b1000, S9 = 4'b1001} statetype;
   statetype state, nextstate;
   parameter green = 2'b00;
   parameter yellow = 2'b01;
   parameter red = 2'b10;

   // 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 = S1;
         else if (~sb) nextstate = S0;
         else nextstate = S6;
      S1: nextstate = S2;
      S2: if(~sa & ~sb) nextstate = S0;
         else if (~sa & sb) nextstate = S4;
         else if (sa & ~sb) nextstate = S2;
         else nextstate = S3;
      S3: nextstate = S4;
      S4: nextstate = S5;
      S5: nextstate = S6;
      S6: nextstate = S7;
      S7: if(~sa & ~sb) nextstate = S0;
         else if (~sa & sb) nextstate = S7;
         else if (sa & ~sb) nextstate = S9;
         else nextstate = S8;
      S8: nextstate = S9;
```

```
// Code Continues
      S9: nextstate = S0;
      default: nextstate = S0;
      endcase


   // output logic
   always_comb
   begin
      case (state)
      S0: {la1, la0, lb1, lb0} = {red, red};
      S1: {la1, la0, lb1, lb0} = {yellow, red};
      S2: {la1, la0, lb1, lb0} = {green, red};
      S3: {la1, la0, lb1, lb0} = {green, red};
      S4: {la1, la0, lb1, lb0} = {yellow, red};
      S5: {la1, la0, lb1, lb0} = {red, red};
      S6: {la1, la0, lb1, lb0} = {red, yellow};
      S7: {la1, la0, lb1, lb0} = {red, green};
      S8: {la1, la0, lb1, lb0} = {red, green};
      S9: {la1, la0, lb1, lb0} = {red, yellow};
      default: {la1, la0, lb1, lb0} = {red, red};
      endcase

      case ({la1,la0})
      2'b00: {ledA1, ledA2, ledA3} = 3'b110;
      2'b01: {ledA1, ledA2, ledA3} = 3'b100;
      2'b10: {ledA1, ledA2, ledA3} = 3'b111;
      default: {ledA1, ledA2, ledA3} = 3'b111;
      endcase

      case ({lb1,lb0})
      2'b00: {ledB1, ledB2, ledB3} = 3'b110;
      2'b01: {ledB1, ledB2, ledB3} = 3'b100;
      2'b10: {ledB1, ledB2, ledB3} = 3'b111;
      default: {ledB1, ledB2, ledB3} = 3'b111;
      endcase
   end

endmodule
```

SYSTEM VERILOG

TESTBENCH CODE REDESIGNED OUTPUTS

(e) SystemVerilog Testbench with Outputs ledA1, ledA2, ledA3, ledB1, ledB2, ledB3 using Decoder

```
module tb_fsm();
    logic clk, reset, sa, sb;
    logic ledA1, ledA2, ledA3, ledB1, ledB2, ledB3;

    fsm_module dut(clk, reset, sa, sb, ledA1, ledA2, ledA3, ledB1, ledB2, ledB3);


    always
        begin
            clk = 0; #1500000000; clk = 1; #1500000000;
        end

    initial
        begin
        reset <= 0; sa <= 0; sb <= 0; @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        @(posedge clk);
        reset <= 1; sa <= 0; sb <= 0; @(posedge clk);
        reset <= 0; sa <= 1; sb <= 1; @(posedge clk);
        @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        @(posedge clk);
        sa <= 0; sb <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        sa <= 0; sb <= 0; @(posedge clk);
        sa <= 1; sb <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        sa <= 0; sb <= 1; @(posedge clk);
        @(posedge clk);
        sa <= 1; sb <= 0; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        $stop; // End the simulation.
        end
endmodule
```