CS223 Digital Design Section 2

Lab 4 – Preliminary Report

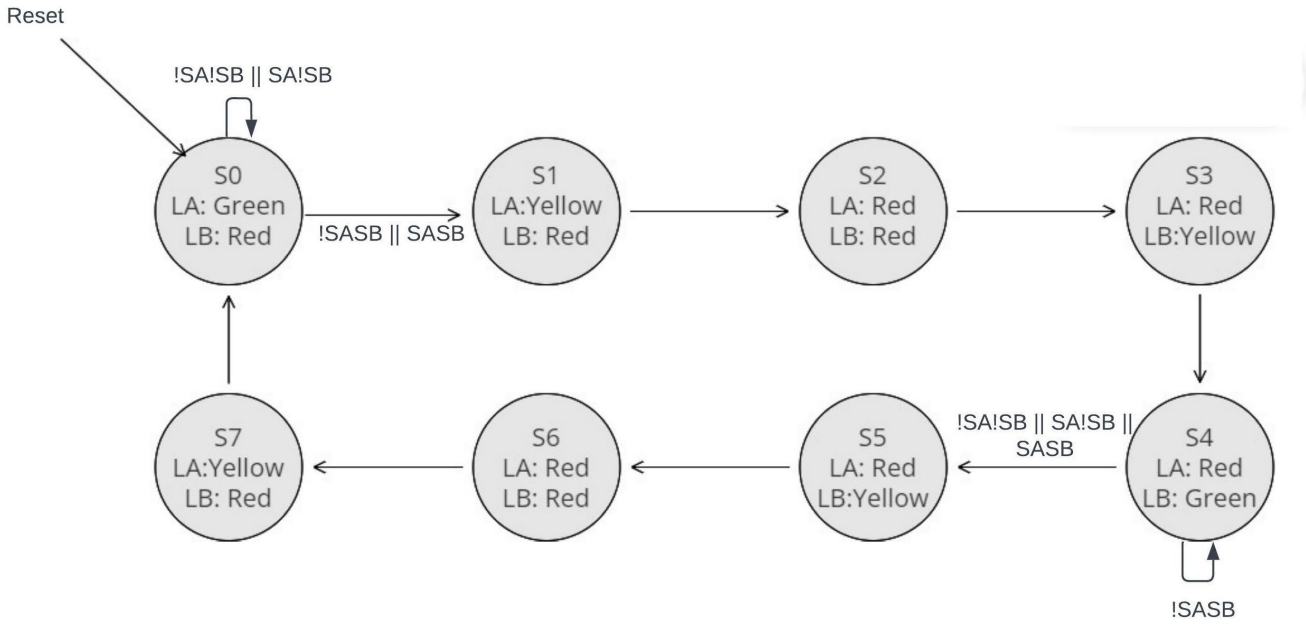
Seçkin Alp Kargı

22001942

CS

15.05.2023

State Transition Diagram



State Encoding & Output

	S2	S1	S0	LA1	LA0	LB1	LB0
S0	000			10		00	
S1	001			01		00	
S2	010			00		00	
S3	011			00		01	
S4	100			00		10	
S5	101			00		01	
S6	110			00		00	
S7	111			01		00	

Output Encoding

Red	00
Yellow	01

Green	10

State Transition Table

S2	S1	S0	SA	SB	S2’	S1’	S0’
0	0	0	X	0	0	0	0
0	0	0	X	1	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	0	1	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	X	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

Equations

S2’ = !S2S1S0 + S2!S1!S0!SASB + S2!S1!S0!SA!SB +S2!S1!S0SA + S2!S1S0 + S2S1!S0 = !S2S1S0 + S2!S0 + S2!S1

S1’ = !S2!S1S0 + !S2S1!S0 + S2!S1S0 + S2S1!S0 = S0 $\oplus$ S1

S0’ = !S2!S1!S0SB + !S2S1!S0 + S2!S1!S0!SA!SB + S2!S1!S0SA + S2S1!S0 = !S2!S0SB + S1!S0 + S2!S0!SB + S2!S0SA

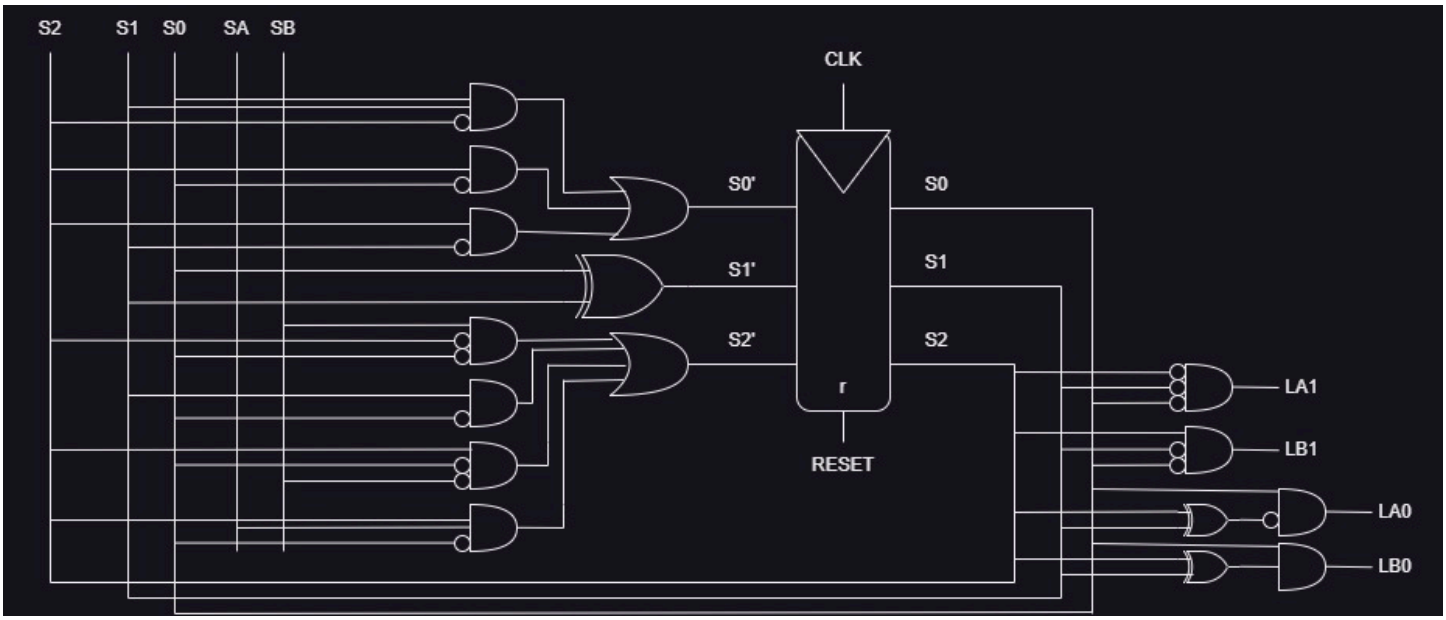
LA0’ = !S2!S1S0 + S2S1S0 = S0 !(S1 $\oplus$ S2)

LA1’ = !S2!S1!S0

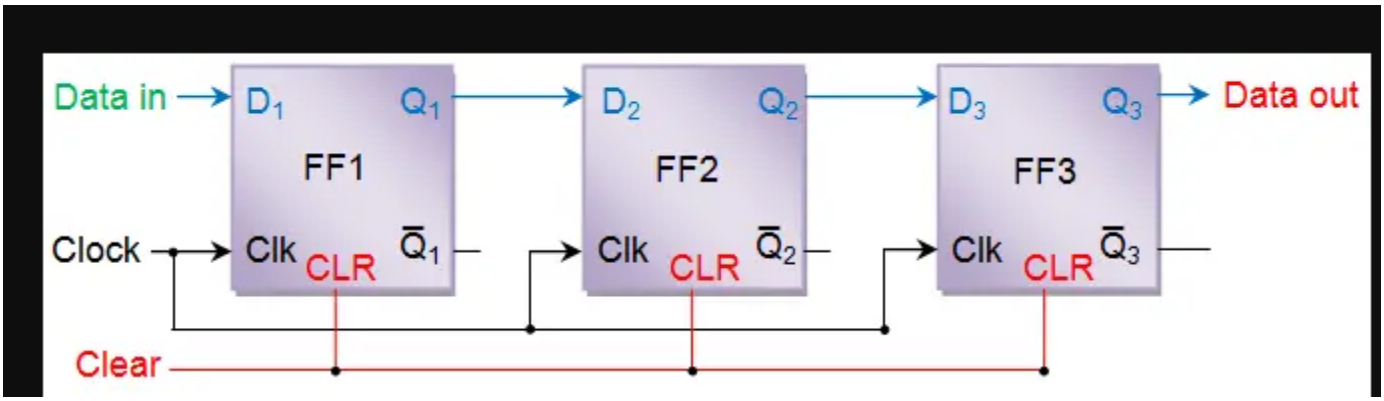
LB0’ = S0 (S1 $\oplus$ S2)

LB1’ = S2!S1!S0

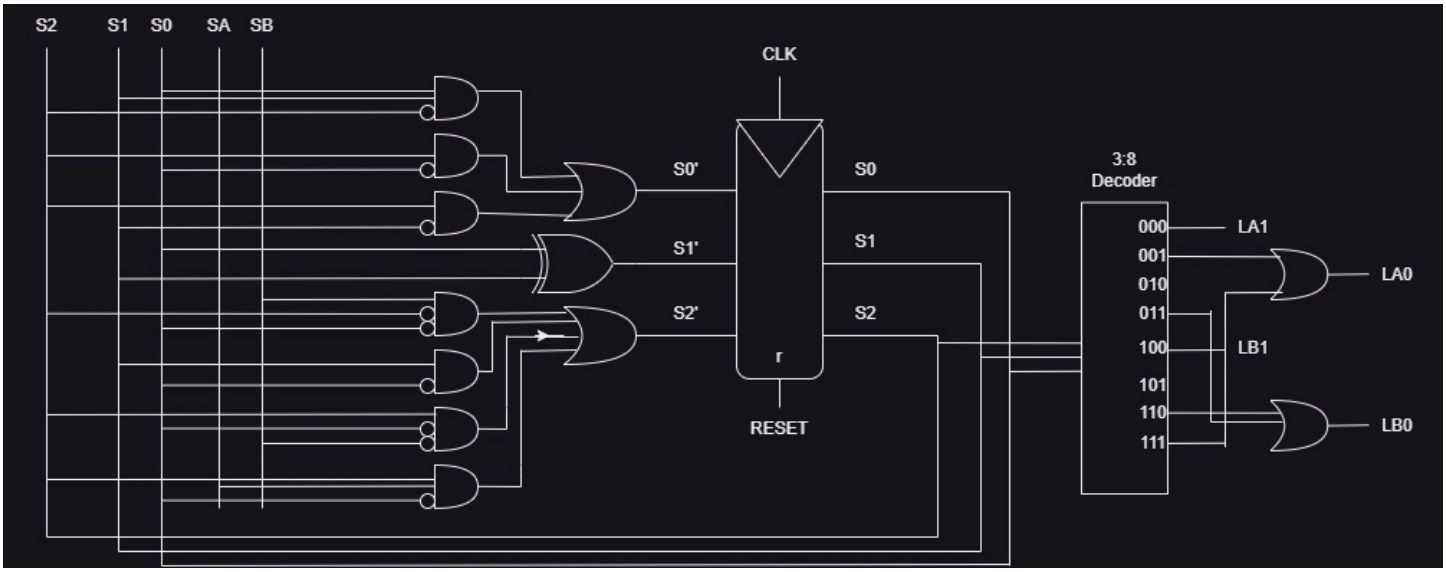
Finite State Machine schematic



I will use 3 flip-flops, 3 bit shift register.



Finite State Machine using Decoder in outputs, I can use 5:32 decoder for s2,s1,s0,sa,sb too.



System Verilog design code

```
`timescale 1ns / 1ps
module decoder3_8
(
  input A,
  input B,
  input C,
  output y1,
  output y2,
  output y3,
  output y4,
  output y5,
  output y6,
  output y7,
  output y8
);
  assign y1 = ~A & ~B & ~C;
  assign y2 = ~A & ~B & C;
  assign y3 = ~A & B & ~C;
  assign y4 = ~A & B & C;
  assign y5 = A & ~B & ~C;
  assign y6 = A & ~B & C;
  assign y7 = A & B & ~C;
  assign y8 = A & B & C;
endmodule
`timescale 1ns / 1ps
module decoder5_32
(
  input A,
  input B,
  input C,
  input D,
```

```
input E,
output y1,
output y2,
output y3,
output y4,
output y5,
output y6,
output y7,
output y8,
output y9,
output y10,
output y11,
output y12,
output y13,
output y14,
output y15,
output y16,
output y17,
output y18,
output y19,
output y20,
output y21,
output y22,
output y23,
output y24,
output y25,
output y26,
output y27,
output y28,
output y29,
output y30,
output y31,
```

```
  output y32
);
  assign y1  = ~(A & B & C & D & E);
  assign y2  = ~(~A & B & C & D & E);
  assign y3  = ~(A & ~B & C & D & E);
  assign y4  = ~(~A & ~B & C & D & E);
  assign y5  = ~(A & B & ~C & D & E);
  assign y6  = ~(~A & B & ~C & D & E);
  assign y7  = ~(A & ~B & ~C & D & E);
  assign y8  = ~(~A & ~B & ~C & D & E);
  assign y9  = ~(A & B & C & ~D & E);
  assign y10 = ~(~A & B & C & ~D & E);
  assign y11 = ~(A & ~B & C & ~D & E);
  assign y12 = ~(~A & ~B & C & ~D & E);
  assign y13 = ~(A & B & ~C & ~D & E);
  assign y14 = ~(~A & B & ~C & ~D & E);
  assign y15 = ~(A & ~B & ~C & ~D & E);
  assign y16 = ~(~A & ~B & ~C & ~D & E);
  assign y17 = ~(A & B & C & D & ~E);
  assign y18 = ~(~A & B & C & D & ~E);
  assign y19 = ~(A & ~B & C & D & ~E);
  assign y20 = ~(~A & ~B & C & D & ~E);
  assign y21 = ~(A & B & ~C & D & ~E);
  assign y22 = ~(~A & B & ~C & D & ~E);
  assign y23 = ~(A & ~B & ~C & D & ~E);
  assign y24 = ~(~A & ~B & ~C & D & ~E);
  assign y25 = ~(A & B & C & ~D & ~E);
  assign y26 = ~(~A & B & C & ~D & ~E);
  assign y27 = ~(A & ~B & C & ~D & ~E);
  assign y28 = ~(~A & ~B & C & ~D & ~E);
  assign y29 = ~(A & B & ~C & ~D & ~E);
  assign y30 = ~(~A & B & ~C & ~D & ~E);
```

```
  assign y31 = ~(A & ~B & ~C & ~D & ~E);
  assign y32 = ~(~A & ~B & ~C & ~D & ~E);
endmodule
`timescale 1ns / 1ps
module part1
(
   input logic s0,s1,s2,sa,sb,
   output logic s00,s11,s22
);
assign s22 = (~s2 & s1 & s0) || (s2 & ~s0) || (s2 & ~s1);
assign s11 = (~s2 & ~s1 & s0) || (~s2 & s1 & ~s0) || (s2 & ~s1 & s0) || (s2 & s1 & ~s0);
assign s00 = (~s2 & ~s0 & sb) || (s1 & ~s0) ||(s2 & ~s0 & ~sb) || (s2 & ~s0 & sa);
endmodule
`timescale 1ns / 1ps
module outputpart
(
   input logic s0,s1,s2,
   output logic la0,la1,lb0,lb1
 );
assign la0 = (~s2 & ~s1 & s0) || (s2 & s1 & s0);
assign la1= ~s2 || ~s1 ||~s0;
assign lb0 = (~s2 & s1 &s0) || (s2 & ~s1 & s0);
assign lb1 = s2 || ~s1 || ~s0;
endmodule

`timescale 1ns / 1ps



module fsm2(

  input logic sa,sb,
  input logic clockin,
```

```
  input logic reset,
  output logic la0,la1,
  output logic lb0,lb1

);

  logic clockout;
  logic clock;
  logic[31:0] cntr = 0;
  logic ss0 = 0;
  logic ss1 = 0;
  logic ss2 = 0;
  logic s00,s11,s22;
  logic s0,s1,s2;

  initial
  begin
 assign s0 = 0;
  assign s1 = 0;
 assign s2 = 0;
  assign la1 = 1;
 assign la0 = 0;
assign  lb1 = 0;
 assign lb0 =0;
  end

  always@ (posedge clockin)
  begin
    cntr <= cntr + 1;
  end

  assign clock = cntr[31];
```

```
  BUFG BUFG_inst
  (
    .I(clock),
    .O(clockout)
);

always_ff@ (posedge clockout, posedge reset)
if (reset) begin
    s0 <= ss0;
    s1 <= ss1;
    s2 <= ss2;
end
else begin
    s0 <= s00;
    s1 <= s11;
    s2 <= s22;
    end
    part1 p11
    (
        .s0(s0),
        .s1(s1),
        .s2(s2),
        .sa(sa),
        .sb(sb),
        .s00(s00),
        .s11(s11),
        .s22(s22)
    );
    outputpart o1
    (
        .s0(s0),
        .s1(s1),
```

```
      .s2(s2),
      .la0(la0),
      .la1(la1),
      .lb0(lb0),
      .lb1(lb1)
   );
endmodule
`timescale 1ns / 1ps
```

Another Implementation

```
module finitestate
(
   input logic sa,
   input logic sb,
   output logic la0,la1,
   output logic lb0, lb1,
   input logic clck,
   input logic reset
);
typedef enum logic [2:0] {s0,s1,s2,s3,s4,s5,s6,s7} State;
State currentState;
State nextState;
int unsigned cntr = 0;
logic CLK100MHZ;
localparam int threesecond = 300000000;
logic clock2;
always @(posedge CLK100MHZ)
begin
   if(cntr ==  1000000)
   begin
      clock2 = 0;
   end
   cntr = cntr +  1;
```

```
    if( cntr == threesecond)
    begin
        clock2 = 1;
        cntr = 0;
    end
end
always_ff @(posedge clck)
    if(reset) currentState <= s0;
    else currentState <= nextState;
always_comb
    case(currentState)
        s0: if(sb) nextState = s1;
            else nextState = s0;
        s1: nextState = s2;
        s2: nextState = s3;
        s3: nextState = s4;
        s4: if(sb && ~sa) nextState = s4;
            else nextState = s5;
        s5: nextState = s6;
        s6: nextState = s7;
        s7: nextState = s0;
        default: nextState = s0;
        endcase
 assign la1 = currentState == s0;
 assign la0 = (currentState == s1 | currentState == s7 );
 assign lb1 = currentState == s4;
 assign lb0 = (currentState == s3 | currentState == s5 );
endmodule
```

Test Bench

```
`timescale 1ns / 1ps
module finitestateverilog(
);
logic clck;
logic reset;
logic sa;
logic sb;
logic lb0,lb1;
logic la1 ,la0;

finitestate fs
(
   .clck(clck),
   .reset(reset),
   .sa(sa),
   .sb(sb),
   .lb1(lb1),
   .lb0(lb0),
   .la1(la1),
   .la0(la0)
);

initial
begin
reset <= 1; #10;
reset <= 0;
sa = 0; sb = 0; #20;
sa = 0; sb = 1; #20;
sa = 1; sb = 0; #20;

sa = 1; sb = 1; #20;
```

```
sa = 1; sb = 1; #20;
sa = 0; sb = 1; #20;
sa = 1; sb = 0; #20;
sa = 1; sb = 1; #20;

sa = 0; sb = 0; #20;
sa = 1; sb = 0; #20;
sa = 0; sb = 1; #20;
sa = 0; sb = 1; #20;
sa = 0; sb = 0; #20;

sa = 0; sb = 1; #20;
sa = 1; sb = 1; #20;

sa = 1; sb = 1; #20;
sa = 1; sb = 1; #20;

sa = 0; sb = 1; #20;
sa = 1; sb = 0; #20;

sa = 1; sb = 1; #20;
sa = 1; sb = 0; #20;

sa = 0; sb = 1; #20;
sa = 0; sb = 0; #20;

sa = 1; sb = 1; #20;
sa = 0; sb = 0; #20;
end
always
   begin
   clck <= 1; #5;
```

```
    clck <= 0; #5;
end
endmodule
```