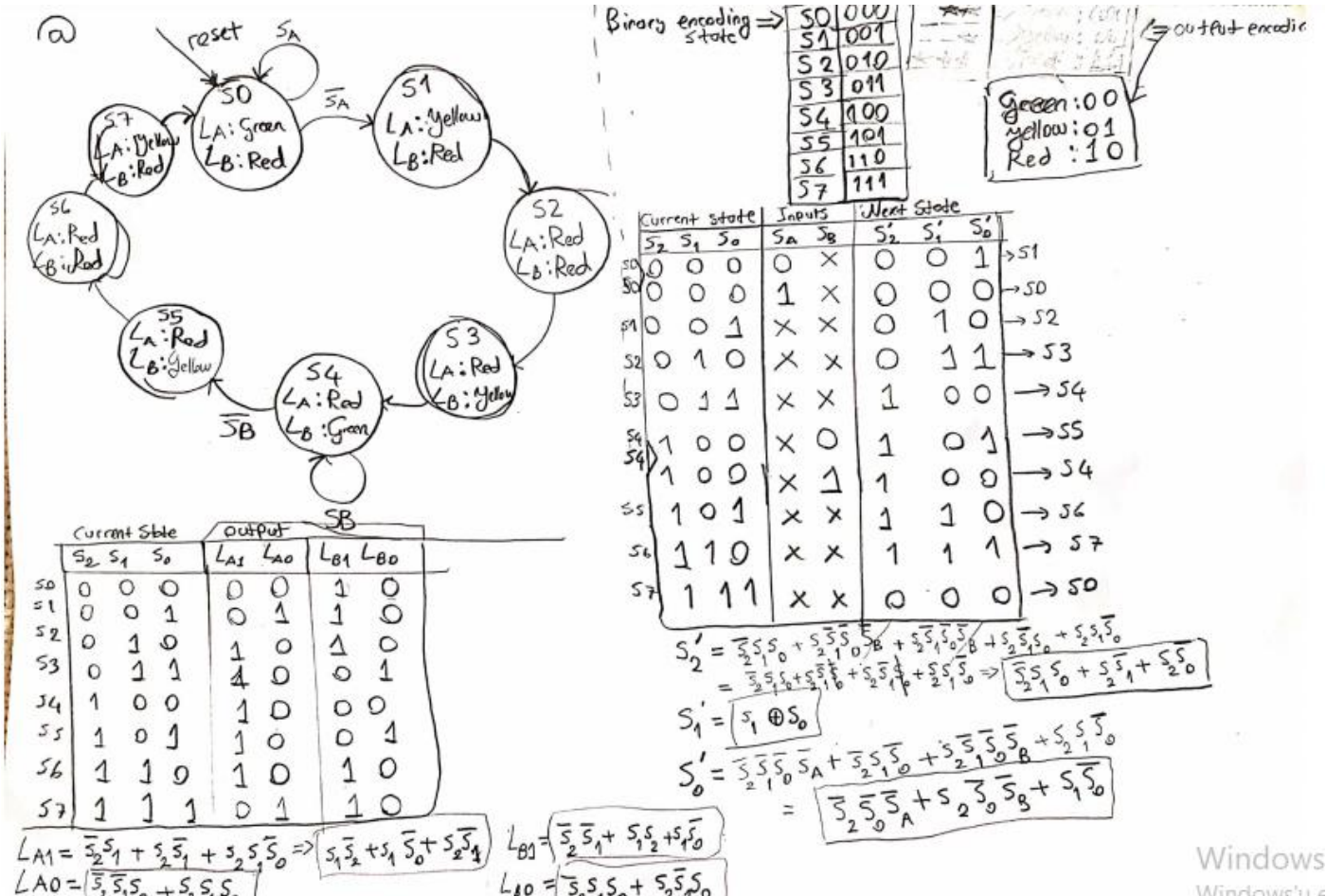
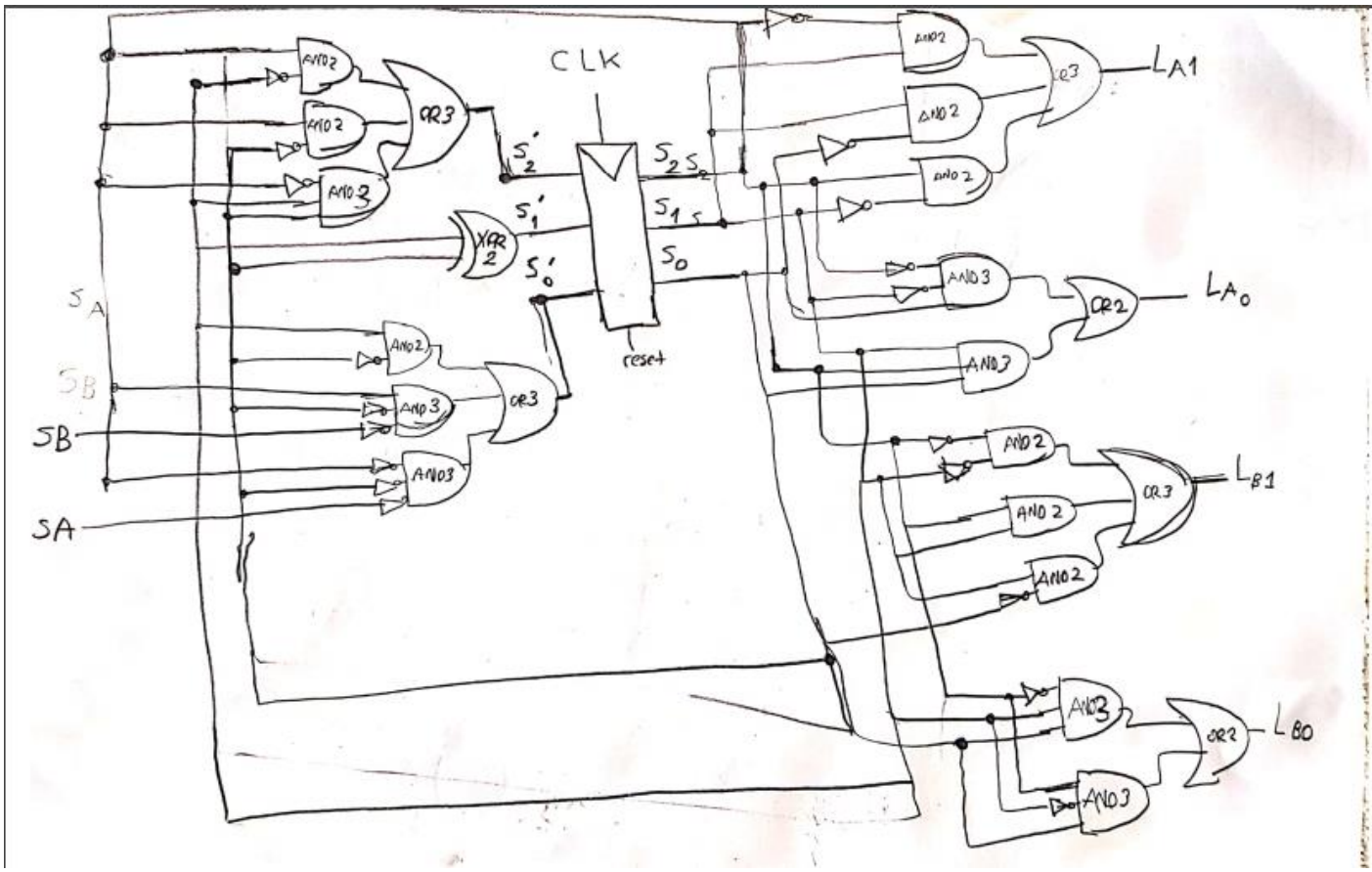
# CS223 DIGITAL DESIGN
# SECTION 3
# Laboratory Assignment 4:
# Traffic Light System
Nurettin Onur Vural 21902330
29.11.2020
PRELIMINARY REPORT

**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. While doing this part, you can ignore the "3 second waiting" constraint between transition of lights.**






**b) How many flip-flops you need to implement this problem?**

As this implementation uses binary encoding for the states in ascending order, in order to represent seven states 3 bits must be used. And accordingly, for this particular implementation three flip flops connected together with the same clock signal can be used the represent the coresponding three bits.

**c) Each transition should happen in 3 seconds. However, BASYS3 has a clock speed of 100MHz. If you were going to use that clock directly in each state transition, waiting time between each state would be around 10 nanoseconds. However, you can obtain a slower clock frequency by counting the rising edges of the Basys3 clock and output your newly generated clock accordingly. In order to obtain 3 seconds delay, you need a clock frequency of 1/3 Hz. 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.**

In order to generate 1/3 Hz frequency from 100MHz, in order words 100 000 000 Hz, we should understand that 300 000 000 clock cycles are required. Here 300 000 000 clock cycles imply that there are 150 000 000 rising edges of the clock is needed.

300 000 000 / 2 = 150 000 000 rising edges of cycle

To store 150 000 000, we need 28 bits in total as $2^{28}$ = 268 435 456 first power of two that is bigger than 150 000 000

```
module clock3seconds(input logic clk_in, output logic clk_out);
   logic [27:0] count = 0;
   always @ (posedge clk_in)
   begin
   count<=count+1;
   if(count==150_000_000)
   begin
   count<=0;
   clk_out = ~clk_out;
   end
   end
endmodule
```

**d) Write the SystemVerilog code for your traffic light system and a testbench for it.**

```
module trafficFSM( input logic clk, reset, SA , SB,
                   output logic [2:0] lightA, lightB);
typedef enum logic [2:0] {S0,S1,S2,S3,S4,S5,S6,S7} statetype;
statetype [2:0] state, nextstate;

typedef enum logic [2:0] {green = 3'b011,yellow = 3'b001,red = 3'b111} light;
//red = 3'b111;
//yellow = 3'b001;
//green = 3'b011;



// state register
always_ff @ (posedge clk)
  if (reset) state <= S0;
  else state <= nextstate;

// next state logic
always_comb
```

```
case (state)
  S0: begin
     if (SA)
        nextstate = S0;
     else
        nextstate = S1;
     end
  S1: nextstate = S2;
  S2: nextstate = S3;
  S3: nextstate = S4;
  S4: begin
     if (SB)
        nextstate = S4;
     else
        nextstate = S5;
     end
  S5: nextstate = S6;
  S6: nextstate = S7;
  S7: nextstate = S0;

  default: nextstate = S0;
 endcase

//output logic
always_comb
   case (state)
   S0: begin
       lightA = green;
       lightB = red;
       end
   S1: begin
```

```
        lightA = yellow;
        lightB = red;
        end
    S2: begin
        lightA = red;
        lightB = red;
        end
    S3: begin
        lightA = red;
        lightB = yellow;
        end
     S4: begin
        lightA = red;
        lightB = green;
        end
     S5: begin
        lightA = red;
        lightB = yellow;
        end
     S6: begin
        lightA = red;
        lightB = red;
        end
     S7: begin
        lightA = yellow;
        lightB = red;
        end
     endcase
endmodule
```

---

```
module trafficFSMtestbench();
 logic clk, reset, SA , SB;
 logic [2:0] lightA, lightB;

// instantiate device under test
trafficFSM dut( clk, reset, SA , SB, lightA, lightB);

always begin
 clk <= 1; #5;
 clk <= 0; #5;
end

initial begin
reset <= 1; #10;

reset <= 0; SA <= 1; SB <= 1; #10; //while there is traffic on A ( SA = 1) should stay on S0

SB <= 0; #10; //this should not change S0 as long as there is traffic on street A

SA <= 0; #10; //now there is no traffic, the state will be S1

#10; //S2

SB <= 1; #10; //S3 Notice that even SB is 1 it will transfer to next stage
//when passes into S4 NOW it will stay

#10; //S4

#10; //S4 (stay in this state as long as traffic on B (SB = 1)

#10; //S4
```

```
#10; //S4

SB <= 0; #10; //transfer to S5

#10; //S6

#10; //S7

#10; //S0

#10; //S1

#10; //S2
//..... S3, S4, S5

#10; //AS LONG AS NO TRAFFIC THE STATES WILL CONTINUE TO CHANGE AND FOLLOW A CYCLE

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 clock3seconds(input logic clk_in, output logic clk_out);
  logic [27:0] count = 0;

  always @ (posedge clk_in)
  begin
  count<=count+1;
  if(count==150_000_000)
  begin
```

```
   count<=0;
   clk_out = ~clk_out;
   end
   end

endmodule

module trafficFSMUpgraded( input logic clk, reset, SA , SB,
                output logic [2:0] lightA, lightB);
typedef enum logic [2:0] {S0,S1,S2,S3,S4,S5,S6,S7} statetype;
statetype [2:0] state, nextstate;

typedef enum logic [2:0] {green = 3'b011,yellow = 3'b001,red = 3'b111} light;
//red = 3'b111;
//yellow = 3'b001;
//green = 3'b011;

logic clkNew;
clock3seconds c1(clk, clkNew );
// state register
always_ff @ (posedge clkNew, posedge reset)
  if (reset) state <= S0;
  else state <= nextstate;

// next state logic
always_comb
 case (state)
   S0: begin
       if (SA)
          nextstate = S0;
       else
```

```
        nextstate = S1;
    end
  S1: nextstate = S2;
  S2: nextstate = S3;
  S3: nextstate = S4;
  S4: begin
    if (SB)
        nextstate = S4;
    else
        nextstate = S5;
    end
  S5: nextstate = S6;
  S6: nextstate = S7;
  S7: nextstate = S0;

  default: nextstate = S0;
 endcase

//output logic
always_comb
   case (state)
   S0: begin
      lightA = green;
      lightB = red;
      end
   S1: begin
      lightA = yellow;
      lightB = red;
      end
   S2: begin
      lightA = red;
```

```
        lightB = red;
        end
    S3: begin
        lightA = red;
        lightB = yellow;
        end
    S4: begin
        lightA = red;
        lightB = green;
        end
    S5: begin
        lightA = red;
        lightB = yellow;
        end
    S6: begin
        lightA = red;
        lightB = red;
        end
    S7: begin
        lightA = yellow;
        lightB = red;
        end
    endcase
endmodule
```