CS223-002

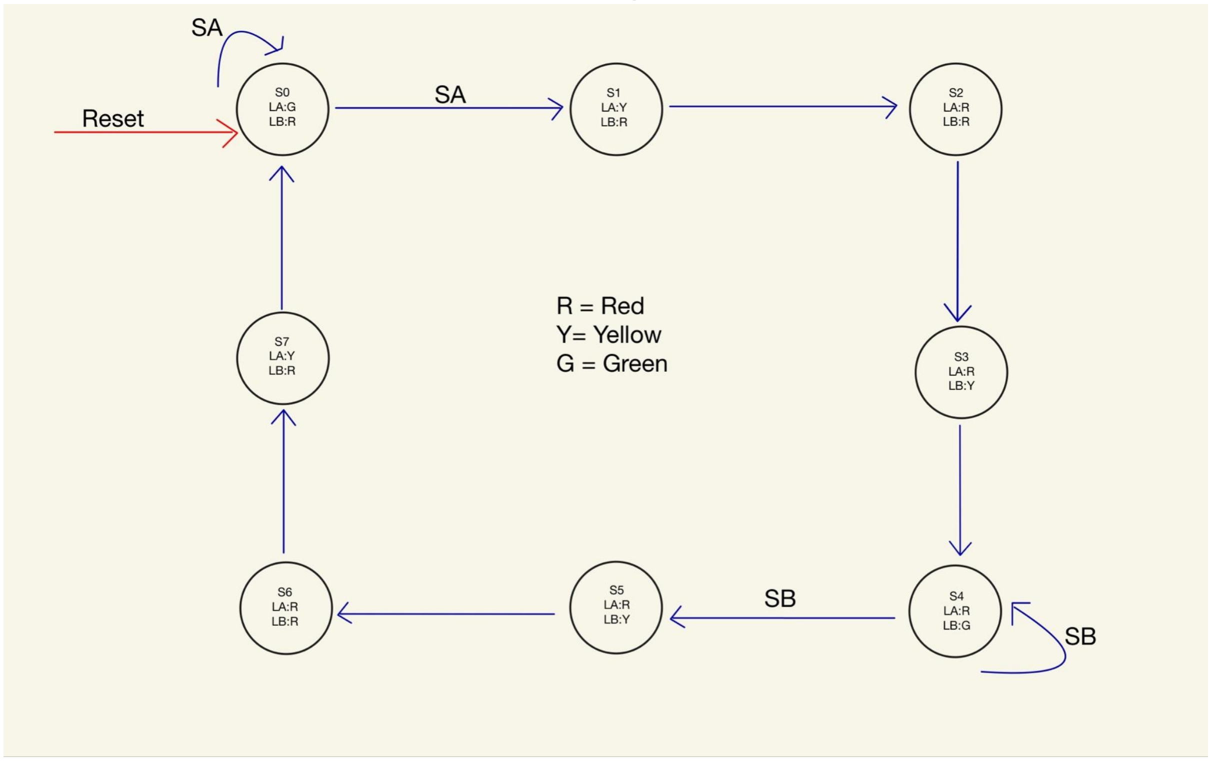
Lab 3

Alper Bozkurt
21802766

14.05.2023

a) Your improved Moore machine state transition diagram, state encodings, state transition table, output table, next state, and output equations.

Moore Machine State Transition Diagram



State Encodings

STATE	ENCODING
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

State Transition Table

Notes:
s0, s1, s2 => current state
SA, SB => inputs
s’0, s’1, s’2 => next state

If there is no traffic on roadA, sensor SA is false and then goes to the next state. If there is traffic, it is true and stays in its current state. Every 3 seconds, the censor will report the situation. After going to the other state every 3 seconds, the states change without inputs until the same starting is provided for Light B.

s0	**s1**	**s2**	**SA**	**SB**	**s’0**	**s’1**	**s’2**
0	0	0	0	X	0	0	1
0	0	0	1	X	0	0	0
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	0	1	0	1
1	0	0	X	1	1	0	0
1	0	1	X	X	1	1	0
1	1	0	X	X	1	1	1
1	1	1	X	X	0	0	0

Output Table

Notes:
Yellow => 001
Green => 011
Red => 111

Inputs => s0, s1, s2
Outputs => lA2, lA1, lA0, lB2, LB1, LB0

It is a table that demonstrates the conclusion of the censors s0, s1, s2 into the lights lA2, lA1, lA0, lB2, LB1, LB0.

s0	**s1**	**s2**	**lA2**	**lA1**	**lA0**	**lB2**	**lB1**	**lB0**
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

Next State and Output Equations

Notes:
’ is for the complement
^ is for indicating the next state

Next State Equations:
s2^ = s2(s1’ + s0’) + s2’s1s0
s1^ = s1 ⊕ s0
s0^ = s0’(s1 + s2’sA’ + s2sB’)

Output Equations:

$IA2 = s2s0' + s2 \oplus s1$

$IA1 = s0' + s2 \oplus s1$

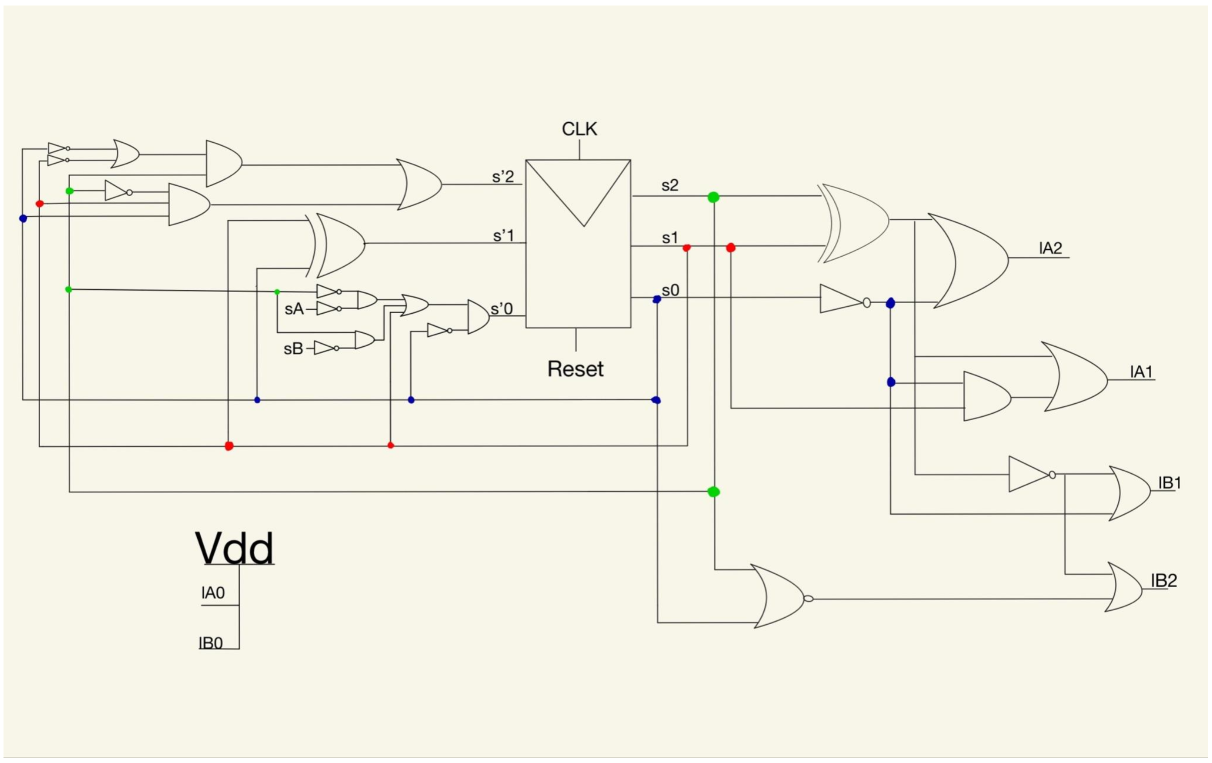
$IA0 = 1$

$IB2 = s1s0' + (s2 \oplus s1)'$

$IB1 = s0' + (s2 \oplus s1)'$

$IB0 = 1$

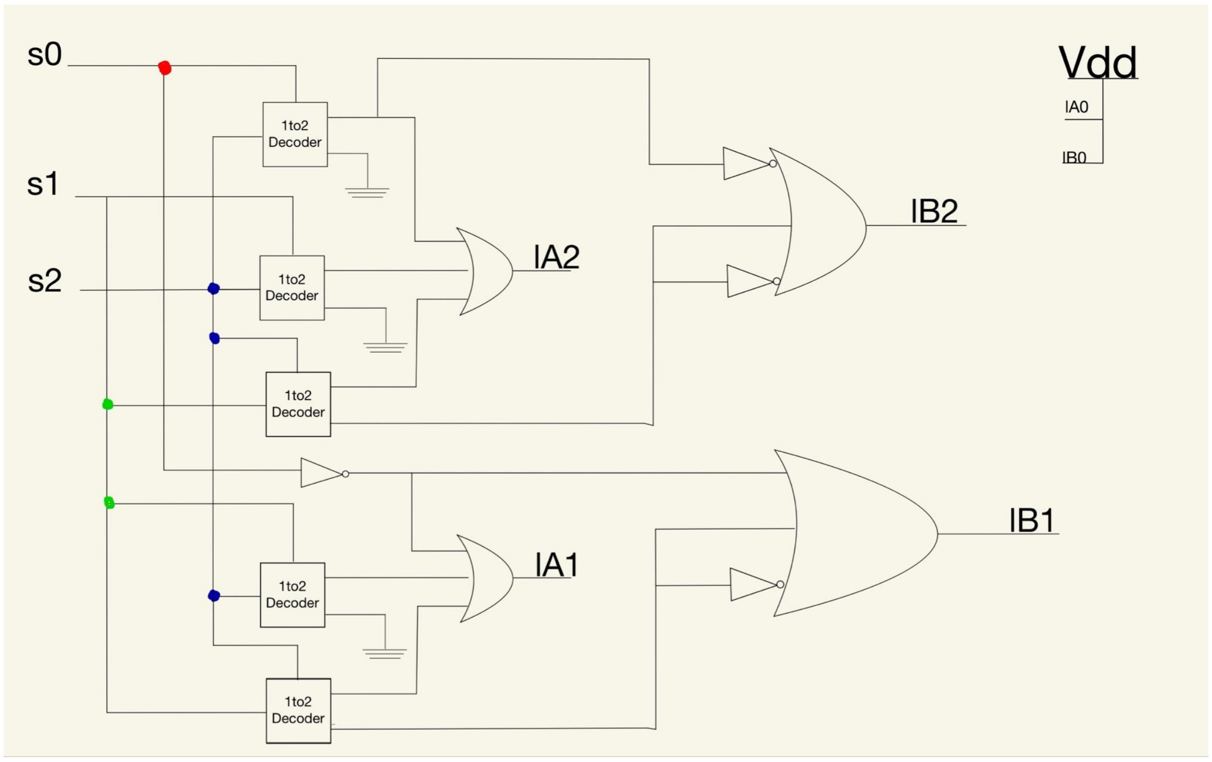
b) Your Finite State Machine schematic.



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

Because there are 3 state bits, 3 flip flops will be enough to implement this problem.

d) Redesign of your outputs using decoders.



e) Your System Verilog design code and testbench.

Code:

```
module tls(input logic clk, reset, sA, sB,

output logic lA0, lA1, lA2, lB0, lB1, lB2);

typedef enum logic [2:0] {S0,S1,S2,S3,S4,S5,S6,S7} statetype;

statetype [2:0] state, nextstate;


//register

always_ff @(posedge clk, posedge reset)

if(reset) state <= S0;

else state <= nextstate;
```

```
//next state

always_comb

case(state)

S0: if ( sA) nextstate = S0;

else nextstate = S1;

S1: nextstate = S2;

S2: nextstate = S3;

S3: nextstate = S4;

S4:  if ( sB) nextstate = S4;

else nextstate = S5;

S5: nextstate = S6;

S6: nextstate = S7;

S7: nextstate = S0;

default: nextstate = S0;

endcase

//output

logic xxx;

xor(xxx, state[1], state[2]);

assign lA2 = xxx | (state[2] & ~state[0]);

assign lA1 = xxx |  ~state[0];

assign lA0 = 1;

assign lB2 = ~xxx | (state[1] & ~state[0]);

assign lB1 = ~xxx | ~state[0];

assign lB0 = 1;

endmodule
```

Testbench:

```
module tls_testbench();

logic clk, reset, sA, sB;

logic lA0, lA1, lA2, lB0, lB1, lB2;

tls dut1(clk, reset, sA, sB, lA0, lA1, lA2, lB0, lB1, lB2);


initial begin

reset = 0;

sA = 0; sB = 0; #10;

sA = 0; sB = 1; #10;

sA = 1; sB = 0; #10;

sA = 1; sB = 1; #10;

always

begin

clk = 1; #100;

clk = 0; #100;

end
```