LAB 08

Finite State Machine: Traffic Light Controller

Ishani Gowaikar
CSU ID: 831702439

Objective:

The objective of this lab is to gain experience with state machine design by using the six-step design process. From the project specifications you will:
Develop a state diagram,

- Reduce the number of states using the implication chart.
- Assign the reduced states the optimal binary values.
- Design the logic and implement the finite state machine in Verilog.

Lab Description and Specs:

Function:

Design a variation of the classical traffic light controller. The intersection is shown in the following diagram.
"A" street runs north-south, "B" street runs east-west, and "C" street enters the intersection from the southeast. "A" street is quite busy, and it is frequently difficult for cars heading south on "A" to make the left turn onto either "B" or "C". In addition, cars rarely enter the intersection from "C" street. Design a traffic light state diagram for this three-way intersection to the following specifications:
a. There are five sets of traffic lights, facing cars coming from "A" north, "A" south, "B" east, "B" west, and "C" street.
b. The red, yellow, and green lights facing cars that are heading South on road A are augmented with a left turn arrow that can be lit up as either green or yellow or not lit up at all.
c. The normal sequencing of lights facing the cars heading South on road A is arrow green, arrow yellow, traffic light green, traffic light yellow, traffic light red, and repeat. In other words, the left arrow light is illuminated in every complete cycle of the lights.
d. However, it should be possible for traffic going from north to south on "A" street to cross the intersection even when the left turn arrow is illuminated. Therefore, the traffic light green should also be illuminated while the turn arrow is lit up.
e. Cars traveling from south to north on "A" street (and all directions on "B" and "C" streets) must see a red light while the left turn arrow is illuminated for the traffic heading south.
f. A car sensor C is embedded in "C" street to detect whether a car is waiting to enter the intersection from the southeast.
g. A timer generates a long interval signal, Time Long (TL), and a short interval signal, Time Short (TS) when set by a start timer (ST) signal.
h. Red and green lights are lit up for at least a TL unit of time. Yellow lights, the green arrow, and the yellow arrow are lit up for exactly a TS unit of time.
i. The "C" street lights cycle from red to green only if the embedded car sensor indicates that a car is waiting. The lights cycle to yellow and then red as soon as no cars are waiting. Under no circumstances is the "C" street green light to be lit for longer than a TL unit of time.

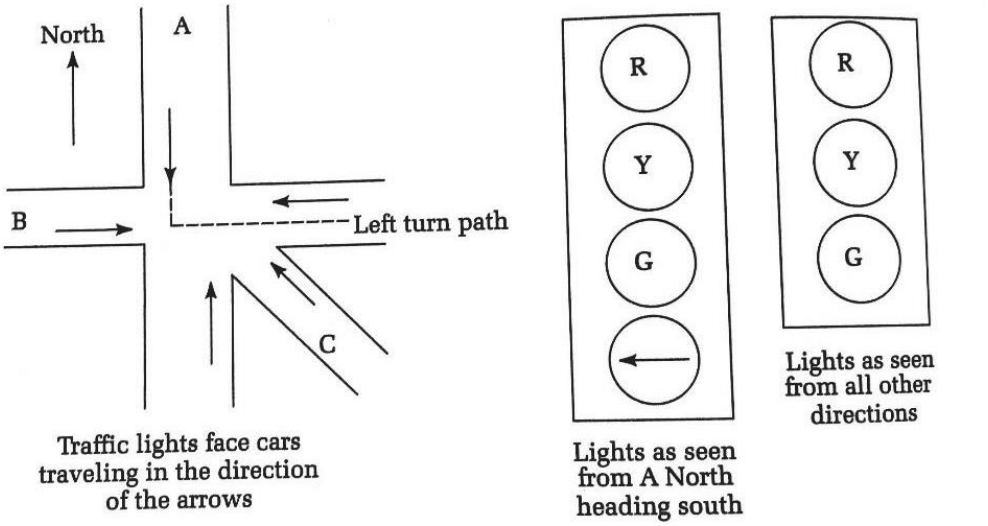


Fig. The Head of each arrow is a traffic light facing the back of the arrow

Inputs:

- Sensor C, short and long time intervals TS and TL.

Outputs:

- "A" street north arrow green/yellow/off (three separate signals)
- "A" street north light green/yellow/red
- "A" street south light green/yellow/red
- "B" street east light green/yellow/red
- "B" street west light green/yellow/red
- "C" street southeast green/yellow/red and the start timer (ST) signal

Recommended Procedures:

Find all possible loops for FSM(s) (if more than one).
Write a Verilog model without state minimization and the simple binary encoding.

List of Items Completed:

a. Lab Procedures
b. Verilog code
c. Results and Simulations
d. Analysis and Explanations (includes a brief description of the schematics, results and simulations)
e. Conclusion
f. Answers to Questions
g. Prelab (attached with the report, state diagram and state minimization included)

Code:

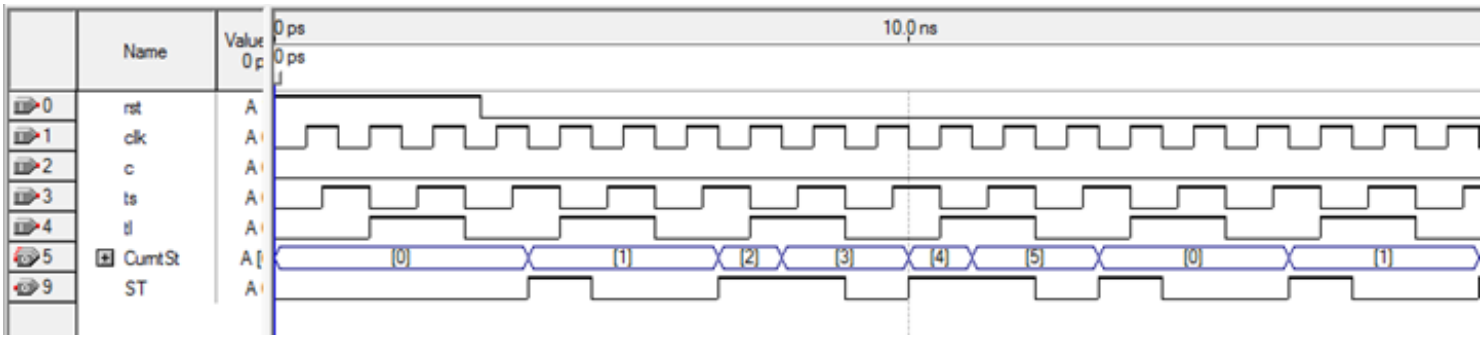
```
module Traffic_cntr(clk, rst, c, ts, tl, CurntSt, nxtSt, ST);
input clk, rst, c, ts, tl;
output ST, CurntSt, nxtSt;

reg[2:0] CurntSt;
reg[2:0] nxtSt;
reg out, ST;

localparam s0=3'b000, s1=3'b001, s2=3'b010, s3=3'b011, s4=3'b100, s5=3'b101, s6=3'b110, s7=3'b111;

always@ (negedge clk)

   if(rst)
      begin
         CurntSt <= s0;
         ST <= 0;
      end
   else
      begin
         CurntSt <= nxtSt;
         ST <= out;
       end

always@ (ts or tl or CurntSt)

   begin

      nxtSt <= CurntSt;

      case(CurntSt)

         s0:
            begin

               if (ts==0)
                  begin
                     nxtSt<=s0;
                     out<=0;
                  end
               else if (ts==1)
                  begin
                     nxtSt<=s1;
                     out=1;
                  end

            end

         s1:
            begin

               if (ts==0)
                  begin
```

```
                    nxtSt<=s1;
                    out<=0;
                end
            else if(ts==1)
                begin
                    nxtSt<=s2;
                    out<=1;
                end

        end

    s2:
        begin

            if (tl==0)
                begin
                    nxtSt<=s2;
                    out<=0;
                end
            else if (tl==1)
                begin
                    nxtSt<=s3;
                    out=1;
                end
        end

    s3:
        begin

            if (ts==0)
                begin
                    nxtSt<=s3;
                    out<=0;
                end
            else if(ts==1)
                begin
                    nxtSt<=s4;
                    out<=1;
                end

        end

    s4:
        begin

            if (tl==0)
                begin
                    nxtSt<=s4;
                    out<=0;
                end
            else if(tl==1)
                begin
                    nxtSt<=s5;
```

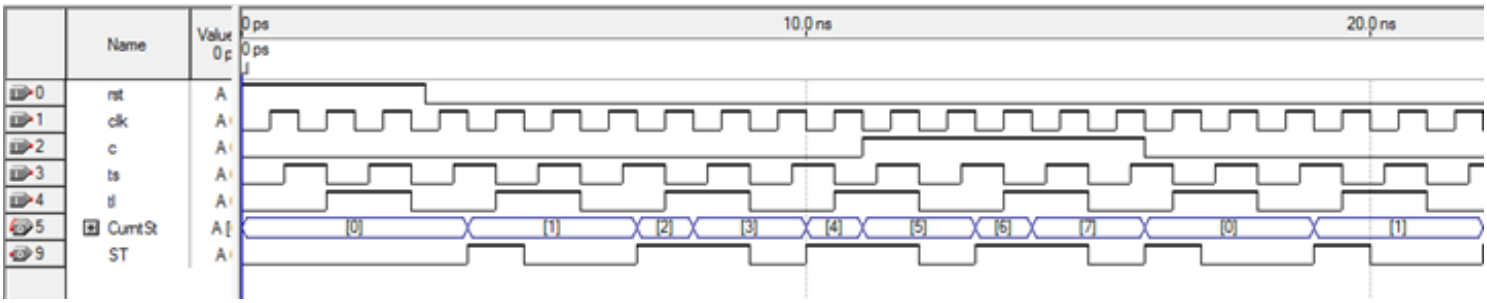
```
                        nxtSt<=s5;
                        out<=1;
                    end
            end

        s5:

            begin

                if (ts==0)
                    begin
                        nxtSt<=s5;
                        out<=0;
                    end
                else if(ts==1 & c==0)
                    begin
                        nxtSt<=s0;
                        out<=1;
                    end
                else if(ts==1 & c==1)
                    begin
                        nxtSt<=s6;
                        out<=1;
                    end
            end

            s6:

            begin

                if(tl==0)
                    begin
                        nxtSt<=s6;
                        out<=0;
                    end
                else if(tl==1)
                    begin
                        nxtSt<=s7;
                        out<=1;
                    end
            end

            s7:

            begin

                if (ts==0)
                    begin
                        nxtSt<=s7;
                        out<=0;
                    end
                else if(ts==1)
                    begin
                        nxtSt<=s0;
                        out<=1;
                    end
            end

    endcase
  end
endmodule
```

Results and Simulations:

a. Car not present on C street



b. Car present on C street



Analysis and Explanations:

The description of each state, state diagram and design of the FSM is mentioned in the prelab. Some of the assumptions we made as per our understanding of lab instructions is as follows:

- We do not check the "C" street sensor for cars at every instant. Instead, we check it once in every complete cycle when the B street W and E signals are yellow and ready to switch red (State S5). If there is a car present, we move to S6 and then S7 and S0. If not, we directly jump to S7 and start next cycle of whole sequence.
- In the design we encode the values for each signal light in the state and thus in the interest of time, we did not decode the states back into signals, but we intend to imply the signal outputs through state description.

The Verilog code provided above is divided into a total of 8 states, from S0-S7, which has been programmed according to the state transition table, and the state definition table and also the state minimization table, which has been provided in the prelab.

Conclusion:

In this lab we designed an FSM for a traffic light controller for the given design parameters and gained experience regarding design methodologies and techniques in Verilog. We successfully implemented the design (with our mentioned assumptions) in 9 states taking care of additional transitions to make the FSM stable.

Answers to Questions:

1. Explain a "race condition" in the context of a Finite State Machine (FSM).

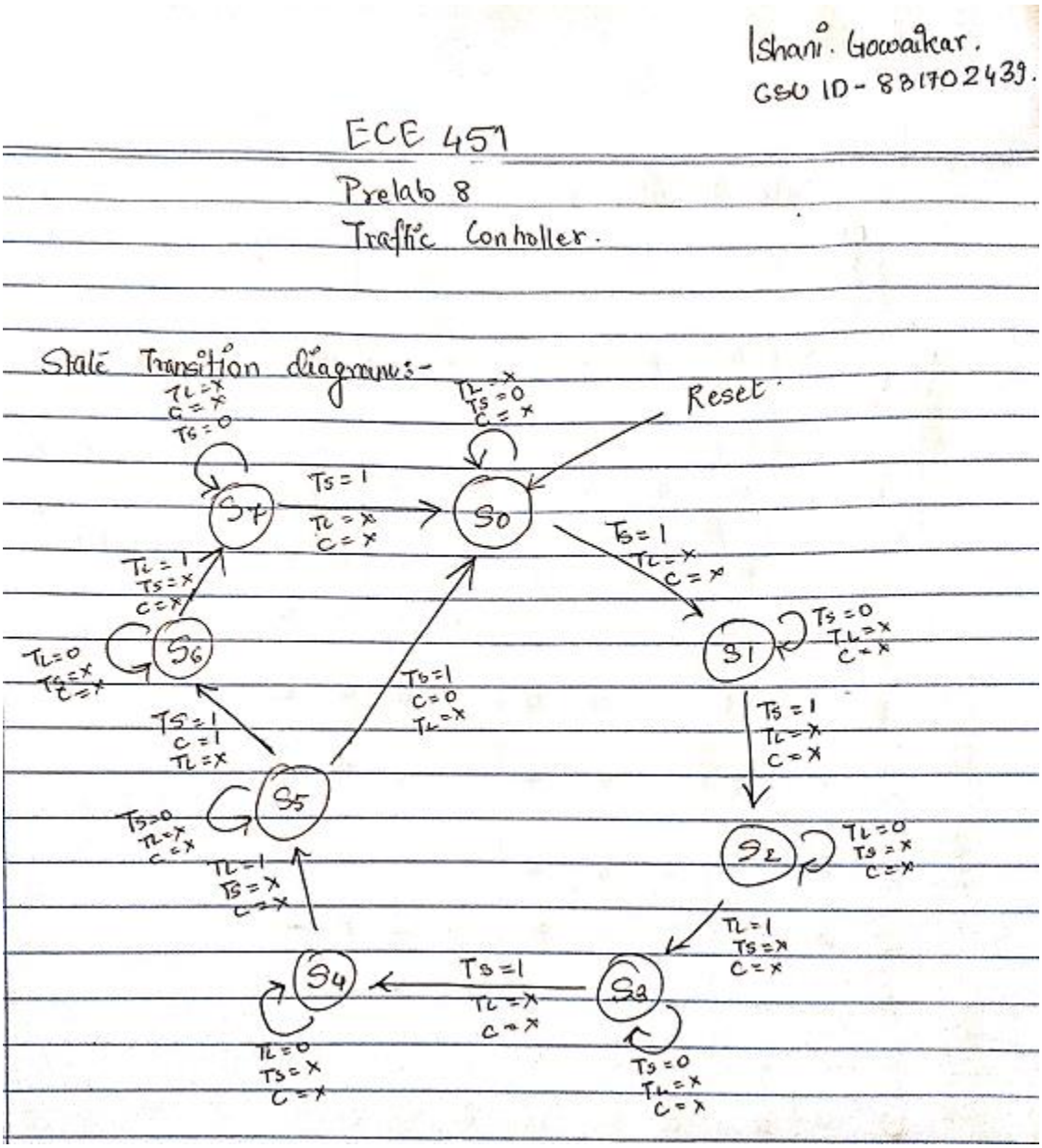
Race Condition in FSM is when more than one state transition occurs in a single clock cycle of a finite state machine. It is caused due to set-up and hold –time violations.

2. Explain the advantages of state reduction, state assignment selection and type of flip-flop selection in an FSM.

a) State reduction or state optimization helps to reduce number of states in a state transition diagram, which in turn reduces the number of flip-flops to implement the design.
b) State assignment technique in an FSM like 'One-hot encoding' or grey code encoding helps reduce power and cost of the entire circuit and both these leads to easy debugging of circuits.
c) Flip flop selection techniques helps to reduce the number of gates requires and also the power requirement of the circuit.

Prelab:

State diagram, state definition table and state minimization included:



	Current State						Inputs			Next State						Output	
	ANc	AN	AS	BE	BW	C	C	TL	TS	ANc+	AN+	AS+	BE+	BW+	C+	ST	Cdt every state chng
[S0]	G	G	R	R	R	R	X	1	X	Y	G	R	R	R	R		1
[S1]	Y	G	R	R	R	R	X	X	X	R	G	G	R	R	R		1
[S2]	R	G	G	R	R	R	X	1	X	R	Y	Y	R	R	R		1
[S3]	R	Y	Y	R	R	R	X	X	1	R	R	R	G	G	R		1
[S4]	R	R	R	G	G	R	X	1	X	R	R	R	Y	Y	R		1
[S5]	R	R	R	Y	Y	R	1	X	1	R	R	R	R	R	G		1
[S5']	R	R	R	Y	Y	R	0	X	1	G	G	R	R	R	R		1
[S6]	R	R	R	R	R	G	X	1	X	R	R	R	R	R	Y		1
[S7]	R	R	R	R	R	Y	X	X	1	G	G	R	R	R	R		1

State Definitions :-

C.S	I/P			N.S					O/P.
Q_2 Q_1 Q_0	C	T_L	T_S	Q_2^+ Q_1^+ Q_0^+			T_2	T_1	T_0	'ST'
	X	X	0	S_0 [000]			0	0	0	0
S_0 [000]	X	X	1	S_1 [001]			0	0	1	1
	X	X	0	S_1 [001]			0	0	0	0
S_1 [001]	X	X	1	S_2 [010]			0	1	1	1
	X	0	X	S_2 [010]			0	0	0	0
S_2 [010]	X	1	X	S_3 [011]			0	0	1	1
	X	X	0	S_3 [011]			0	0	0	0
S_3 [011]	X	X	1	S_4 [100]			1	1	1	1
	X	0	X	S_4 [100]			0	0	0	0
S_4 [100]	X	1	X	S_5 [101]			0	0	1	1
	X	X	0	S_5 [101]			0	0	0	0
S_5 [101]	0	X	1	S_0 [000]			1	0	1	1
	1	X	1	S_6 [110]			0	1	1	1
	X	0	X	S_6 [110]			0	0	0	0
S_6 [110]	X	1	X	S_7 [111]			0	0	1	1
	X	X	0	S_7 [111]			0	0	0	0
S_7 [111]	X	X	1	S_0 [000]			1	1	1	1

S