ECE451Lab8

Huanjia Liu (ID:831993149)

Dec 11, 2020

Contents

1 Objective

During this lab, we are asked to improve the design of a traffic light controller system. 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.

Develop the excitation table using T flip-flops.

Design the logic and implement the finite state machine.

2 Introduction

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.

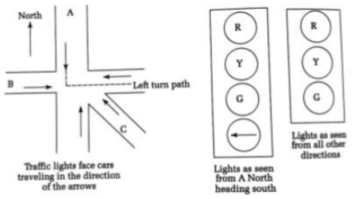

Figure 1: Traffic light controller

3 Procedure and analysis

3.1 State transition diagrams

Before designing FSM, we need to figure out the state transition diagram, the diagram below is my state transition diagram to describe traffic light system. The light behavior of each state are listed below.

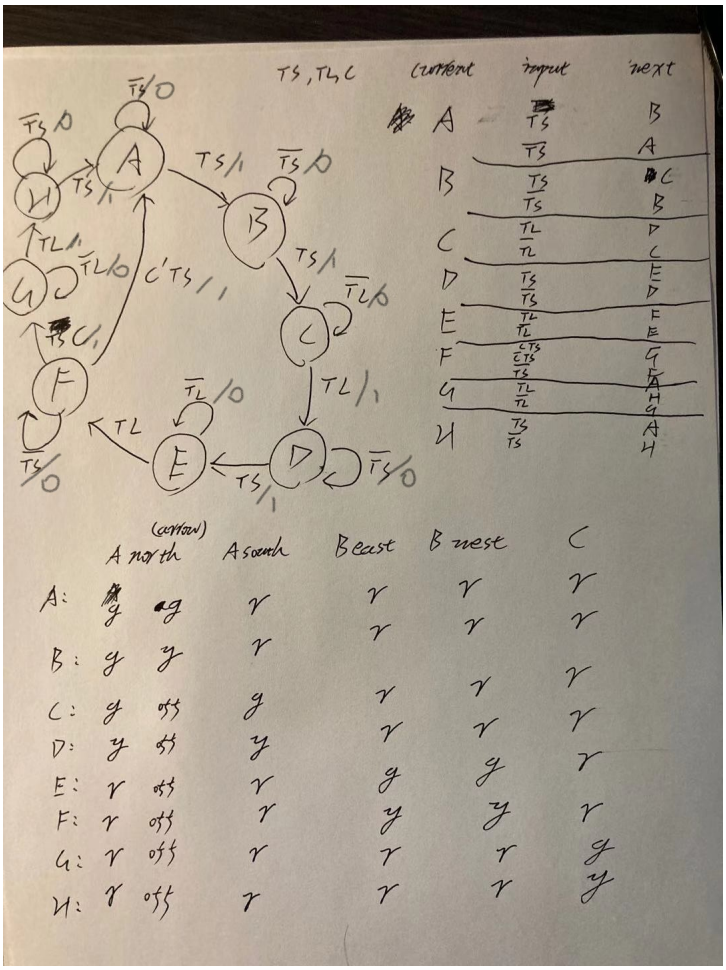

Figure 2: State transition diagrams

3.2 State transition tables

According to the diagram above, we can encode it and generate the state transition tables. We assigned state A H to the binary number 000 111.

Current State		C	TL	TS	Next State		Output(ST)
A	000	X	X	0	A	000	0
		X	X	1	B	001	1
B	001	X	X	0	B	001	0
		X	X	1	C	010	1
C	010	X	0	X	C	010	0
		X	1	X	D	011	1
D	011	X	X	0	D	011	0
		X	X	1	E	100	1
E	100	X	0	X	E	100	0
		X	1	X	F	101	1
F	101	X	X	0	F	101	0
		1	X	1	G	110	1
		0	X	1	A	000	1
G	110	X	0	X	G	110	0
		X	1	X	H	111	1
H	111	X	X	0	H	111	0
		X	X	1	A	000	1

Figure 3: State transition table

From the table above we can generate the equations for next state and output. Here we use the T flip flop, setting Q2,Q1,Q0 for current state and T2,T1,T0 for next state. ST is the output.

$$T_2 = Q_1Q_0TS + Q_2Q_0C'TS$$

$$T_1 = Q'_2Q_0TS + Q_0CTS + Q_1Q_0TS$$

$$T_0 = Q'_2Q'_1TS + Q_1Q'_0TL + Q_0TS + Q_2Q'_0TL$$

$$ST = Q'_2Q'_1TS + Q_1Q'_0TL + Q_0TS + Q_2Q'_0TL$$

3.3 Schematic of T flip flop

Here is my schematic of T flip flop. The PDFF is the risen edge trigger D flip flop.

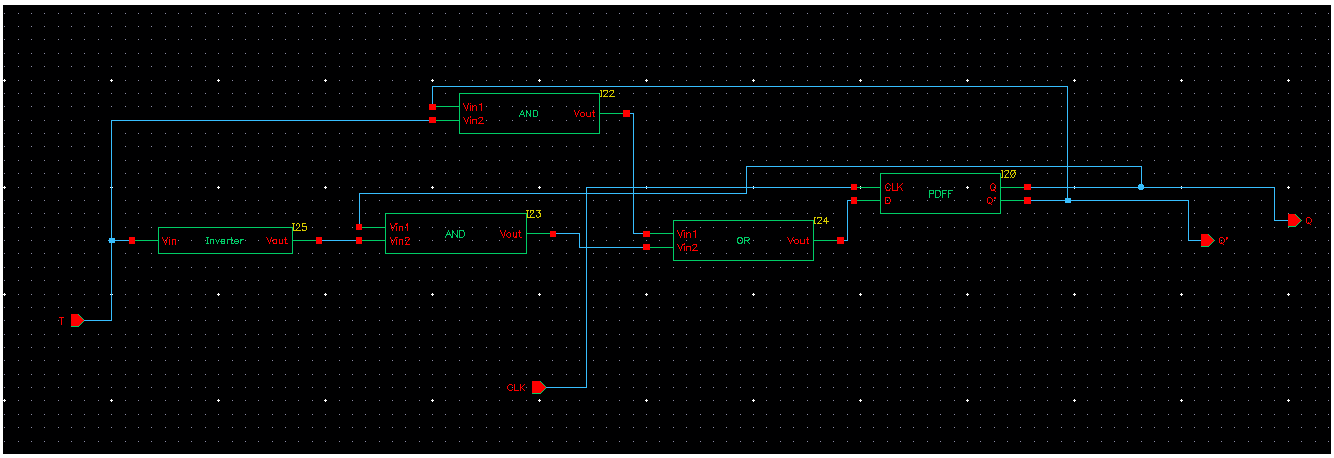

Figure 4: T flip flop

3.4 Schematic of equations

Now we can generate the schematic according to the equations above. Here are my schematic.

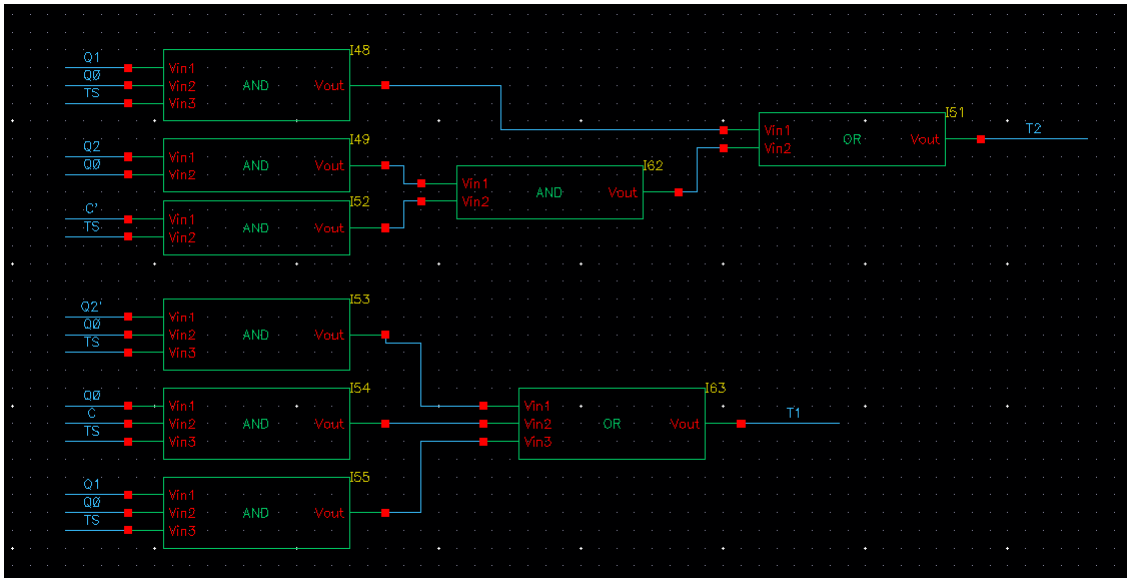

Figure 5: T_2T_1 schematic

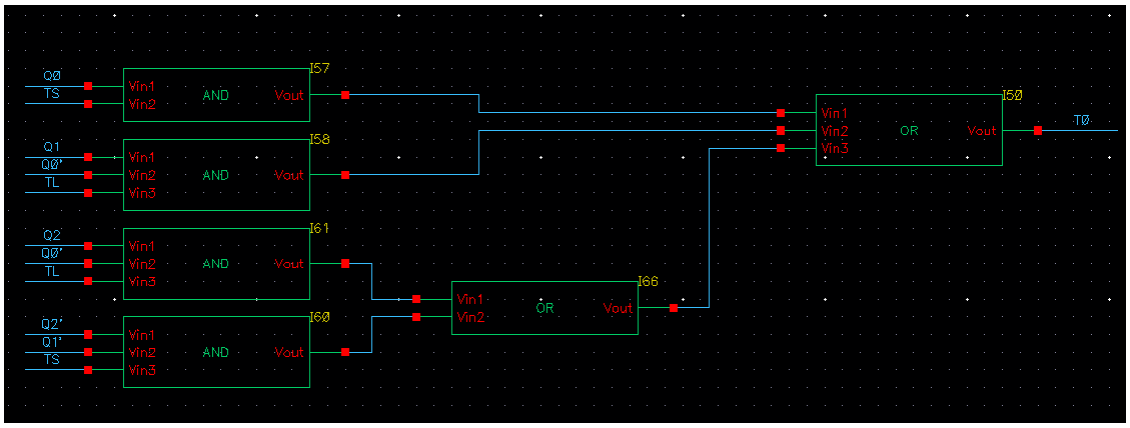

Figure 6: T_0 schematic

Finally, combining them together, here is my main schematic.

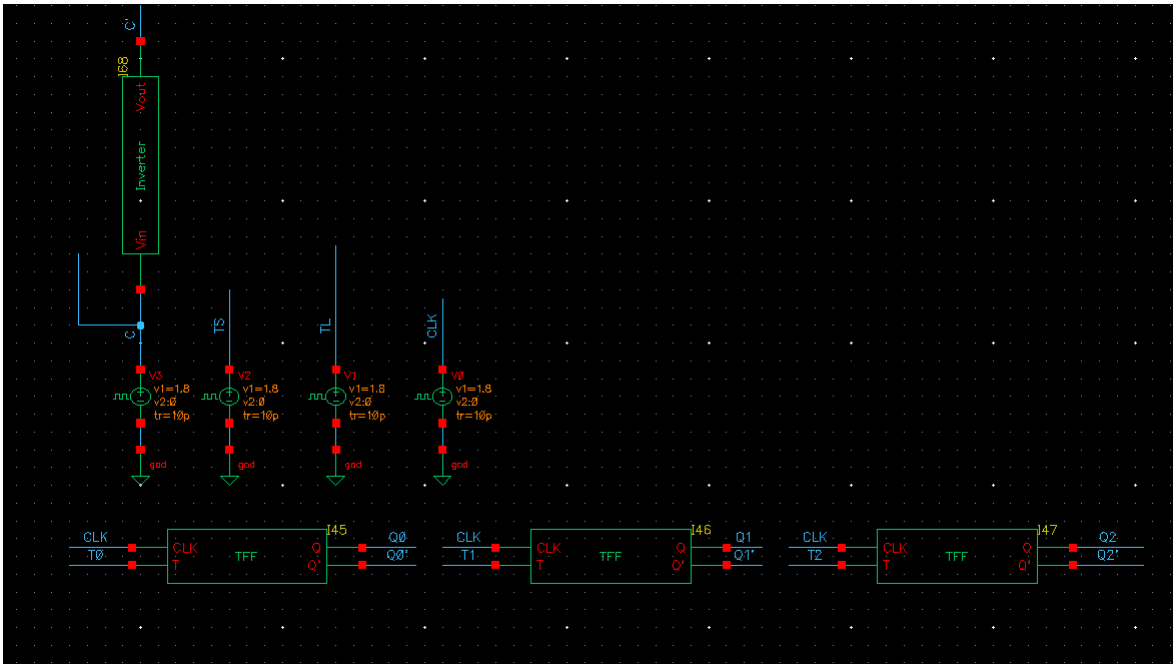

Figure 7: Combining schematic

3.5 Final output waveforms

After running the simulation. we got the figure below. All order of our state transition follows our state transition. The most important is State 101, we can find during the first state 101, the input is 011(C=0, TS=1), then the next state is 000. During the second state 101, the input is 111(C=1, TS=1), then the next

state is 110.

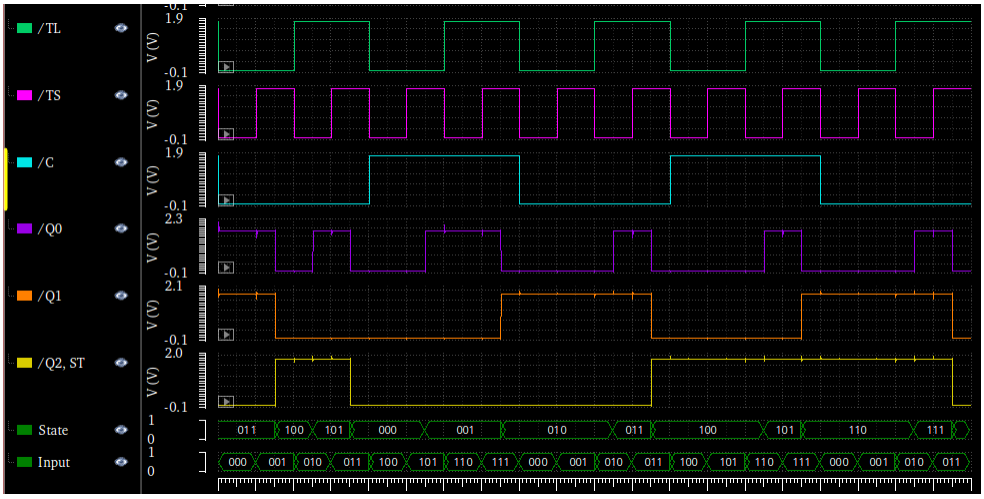

Figure 8: Final output waveforms

3.6 Verilog Part

I also implement the system in Verilog. Here is my code. Here I also add all the lights into my system output. There are 6 lights in system, and each of light has 3 color, which means we need to use 2 bit to express each light. Thus, I add a 12 bit Light output here.

3.6.1 Verilog Code

```
module Lab8(clk, ST, State, TS, TL, C, Light);
input clk,TS,TL,C;
output reg ST;
output reg [2:0] State;
reg [2:0] current;
output reg [11:0] Light;



always@ (posedge clk)
begin
```

```
current <= State;
if((current[1]&current[0]&TS) | (current[2]&current[0]&(~C)&TS))
begin
   State[2] <= ~current[2];
end

if(((~current[2])&current[0]&TS) | (current[0]&C&TS) | (current[1]&current[0]&TS))
begin
   State[1] <= ~current[1];
end

if( ((~current[2])&(~current[1])&TS) | (current[1]&(~current[0])&TL) | (current[0]&TS) |
    (current[2]&(~current[0])&TL))
begin
   State[0] <= ~current[0];
end
ST = ((~current[2])&(~current[1])&TS) | (current[1]&(~current[0])&TL) | (current[0]&TS) |
    (current[2]&(~current[0])&TL);

case (State)
   3'b000: Light = 12'b000010101010;
   3'b001: Light = 12'b010010101010;
   3'b010: Light = 12'b100000101010;
   3'b011: Light = 12'b100101101010;
   3'b100: Light = 12'b101010000010;
   3'b101: Light = 12'b101010010110;
   3'b110: Light = 12'b101010101000;
   3'b111: Light = 12'b101010101001;
   endcase
```

```
end


endmodule
```

3.6.2 Verilog Waveform

Here is my Verilog output waveform, same as the waveform in Candence, when I set C =0, the State 101 transfer to 000, when I set C = 1, the State transfer to 110. All the state transfer follow the transition table.

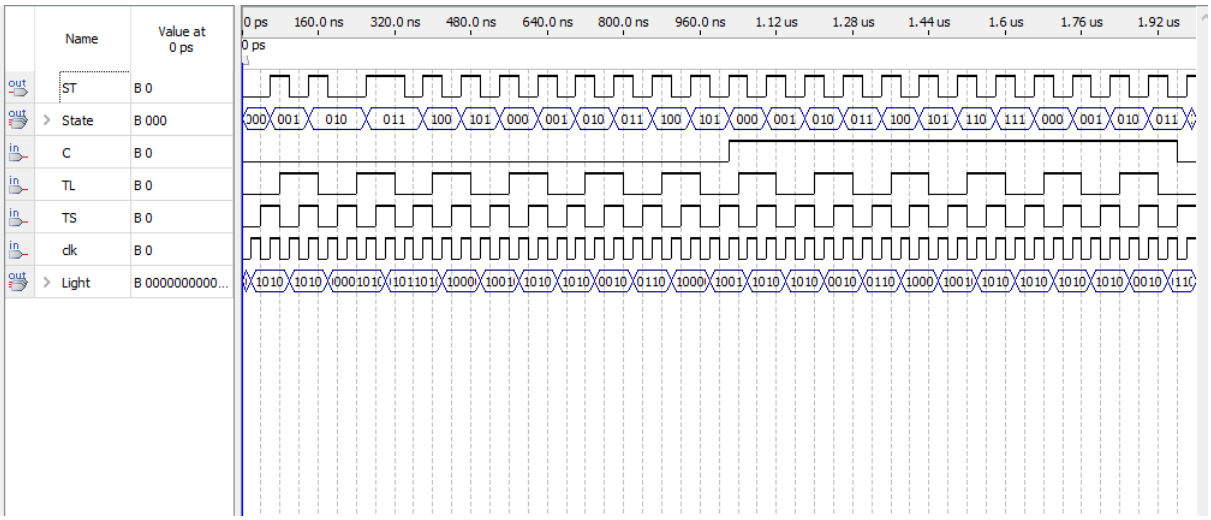

Figure 9: Final output waveforms

4 Questions

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

Because of the circuit delay, the FSM accept wrong input or wrong previous state, then go into the wrong state, causing system error.

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

State reduction and state assignment selection can help us simplify the circuit, remove redundant state and circuit, improve circuit speed and efficiency. Type of flip-flop selection can help us remove redundant circuit, keep the circuit stable.

5 Conclusion

Now we can get the conclusion, I finish this lab successfully, and achieve all the objective. I have learned that it's hard to consider a perfect circuit in the first time through the actual situation. So we can raise a redundant FSM at first, then use some minimization way to reduce state to make the FSM more simple and effect.