# Pre-Lab 11: The Traffic Light Controller Lab

ECEN 248 - 505

TA: Younggyun Cho

Date: November 10, 2020

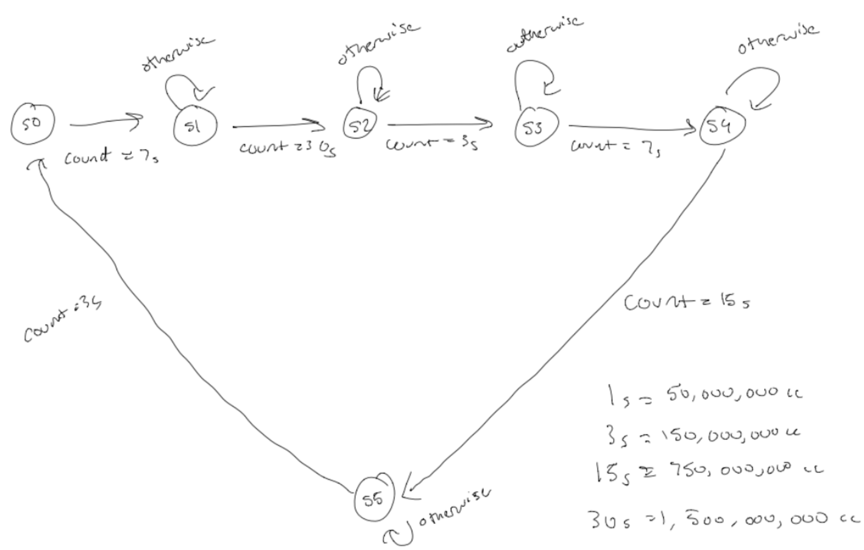
**1. Fill in the remaining entries in Table 1. You may find the discussion on Generating Timing Delays in the background section helpful.**

| State | Highway Output | Farm Road Output | Delay (s) | Delay (cc) |
|---|---|---|---|---|
| **S0** | red | red | 1 | 50,000,000 |
| **S1** | green | red | 30 | 1,500,000,000 |
| **S2** | yellow | red | 3 | 150,000,000 |
| **S3** | red | red | 1 | 50,000,000 |
| **S4** | red | green | 15 | 750,000,000 |
| **S5** | red | yellow | 3 | 150,000,000 |

**2. Based on the column entries you just calculated, what is the necessary value of n in Figure 5?**

The necessary value of n in Figure 5 is 31 bits.

**3. Given Table 1 and Figure 6, create a state diagram for the traffic light controller FSM. Be sure to include the appropriate input and output labels. Assume S0 is the reset state.**



**4. Now describe the traffic light controller FSM in Verilog using the following module interface:**

```
module tlc_fsm(
    output reg [2:0] state, // output for debugging
    output reg RstCount, // use an always block
    output reg [1:0] highwaySignal, farmSignal,
    input wire [n-1:0] Count, // use n computed earlier
    input wire Clk, Rst // clock and reset
);
```

```
parameter S0 = 3'b000,
S1 = 3'b001,
S2 = 3'b010,
S3 = 3'b011,
S4 = 3'b100,
S5 = 3'b101;

/*intermediate nets*/
reg [2:0] nextState;

//defining colors
parameter green = 2'b00,
yellow = 2'b01,
red = 2'b10;

/*describe next state logic*/
always@(state or Count)

case(state)
S0: begin
if(Count == `one_sec)
nextState = S1; //transition
else
nextState = S0;
end
S1: begin
if(Count == `thirty_sec)
nextState = S2;
else
```

```
nextState = S1;
end
S2: begin
if(Count == `three_sec)
nextState = S3;
else
nextState = S2;
end
S3: begin
if(Count == `one_sec)
nextState = S4; //transition
else
nextState = S3;

end
S4: begin
if(Count == `fifteen_sec)
nextState = S5;
else
nextState = S4;
end
S5: begin
if(Count == `three_sec)
nextState = S0;
else
nextState = S5;
end
default:
nextState = S0;
endcase

/*describe output logic*/
always@(state or Count)
case(state)
S0: begin
highwaySignal = red;
farmSignal = red;
if(Count == `one_sec)

RstCount = 1;
else
RstCount = 0;
end
S1: begin
```

```
highwaySignal = green;
farmSignal = red;
if(Count == `thirty_sec)
RstCount = 1;
else
RstCount = 0;
end
S2: begin
highwaySignal = yellow;
farmSignal = red;
if(Count == `three_sec)
RstCount = 1;
else
RstCount = 0;
end
S3: begin
highwaySignal = red;
farmSignal = red;
if(Count == `one_sec)

RstCount = 1;
else
RstCount = 0;
end
S4: begin
highwaySignal = red;
farmSignal = green;
if(Count == `fifteen_sec)
RstCount = 1;
else
RstCount = 0;
end
S5: begin
highwaySignal = red;
farmSignal = yellow;
if(Count == `three_sec)
RstCount = 1;
else
RstCount = 0;
end
default: begin
highwaySignal = red;
farmSignal = red;
RstCount = 1;
```

```
end
endcase

/*behavior for input clock*/
always@(posedge Clk)
if(Rst)
state <= S0;
else
state <= nextState;

endmodule
```