**ECE 211: Digital Circuits I — Spring 2025**

**Lab 10: Traffic Light Controller**

By: *Efe CIVISOKEN, Knycalus GREEN-JONES*

07/05/2025

**Statement of Collaboration:** Lab demonstration was done by the both authors together. Efe derived the behavior and logic of the FSM, the boolean equations associated with it, designed/wrote the whole code in SystemVerilog, modularly tested the code, and ensured proper implementation through the top module; while Knycalus was focused on helping Efe with minor and simultaneous fix of syntax errors and debugging while Efe was writing the code.

Concerning the lab report, Efe wrote INTRODUCTION, DESIGN, and IMPLEMENTATION, and every other part, whereas Knycalus wrote TESTING/RESULTS and CONCLUSION/DISCUSSION parts. At the end, Efe formatted paragraphs and styling.

**Time Spent**: 2.45 hours.

**INTRODUCTION**

This lab implements a synchronous traffic light controller using a finite state machine in SystemVerilog. The intersection has two traffic lights—LC for Cattell Street and LH for High Street—each consisting of red, yellow, and green LEDs. The controller receives input from two traffic sensors (Tc and Th) and a pedestrian crosswalk button. The circuit must ensure a safe transition between directions by introducing an all-red interlock before switching green lights and must also remember and respond to brief pedestrian button presses. The design uses one-hot encoded states with the intent of simplifying the Boolean logic and is verified via simulation and FPGA/Breadboard deployment.

**DESIGN**

The traffic\_light module implements the FSM and all related control logic. A one-hot encoding scheme is used, with six states represented by a 6-bit current\_state register. These states are:

* State 0: High Green, Cattell Red
* State 1: High Yellow, Cattell Red
* State 2: All Red
* State 3: Cattell Green, High Red
* State 4: Cattell Yellow, High Red
* State 5: All Red

A 6-bit next\_state register is computed combinationally. All state transitions follow this cyclic pattern described:  
State 0 → State 1 → State 2 → State 3 → State 4 → State 5 → repeat.

The FSM advances on the rising edge of the system clock (clk) only if the enable signal is high. This enable signal is generated by a separate clock\_enable module, which slows the FSM transitions to human-visible timing.

The reset signal (rst) is synchronous. When rst is high, the FSM returns to its initial state (High Green/State 0). If not in reset, and the enable signal is high, the FSM updates to the next state. Otherwise, the current state is held.

To support brief button presses, a dedicated flip-flop person\_waiting is introduced. When the button is high, this flip-flop is set to 1. It holds this value until the FSM advances, at which point it is cleared. This ensures that short button presses are not missed by the FSM.

Two intermediate conditions are computed using nand gates:

* simplifier1 is the NAND of ~Tc and Th, which is high unless High has traffic but Cattell does not.
* simplifier2 is the NAND of Tc and ~Th, which is high unless Cattell has traffic but High does not.

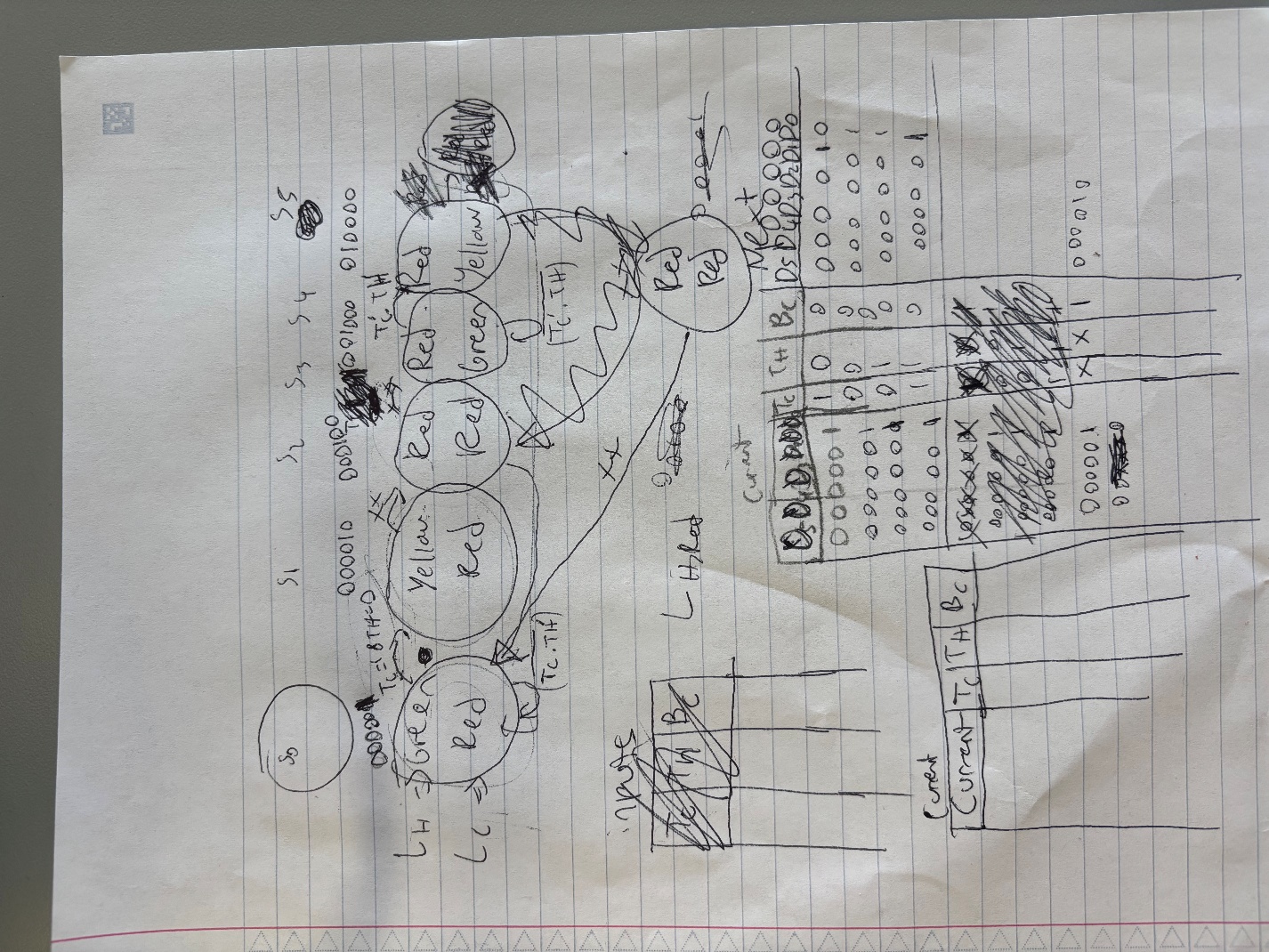
Next state logic is implemented as combinational logic and was derived according to the one-hot encoded FSM and truth table calculations in *Images 1 & 2* and implemented accordingly as can be seen in *Code 1.* This sequence ensures that every switch between green lights passes through a red phase. Pedestrian requests are given priority during Cattell Green, causing an early transition to yellow and eventually allowing High Green to proceed.

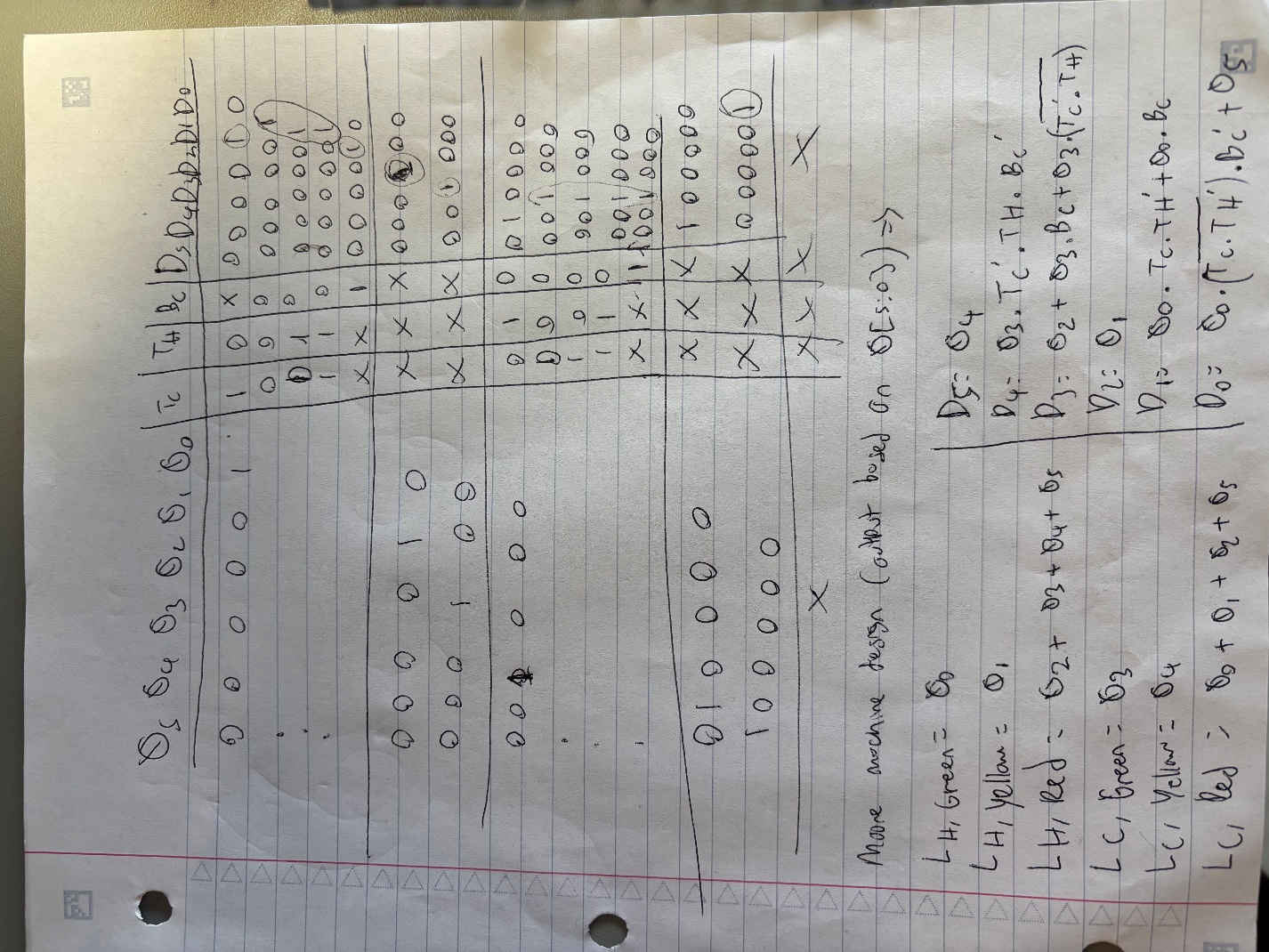
The output lights are determined directly from the current state thanks to one-hot encoded design choice:

* LhGreen is high when current\_state[0] is active
* LhYellow is high in current\_state[1]
* LhRed is high in all other states (states 2,3,4,5)
* LcGreen is high in current\_state[3]
* LcYellow is high in current\_state[4]
* LcRed is high in all other states (states 5,0,1,2)

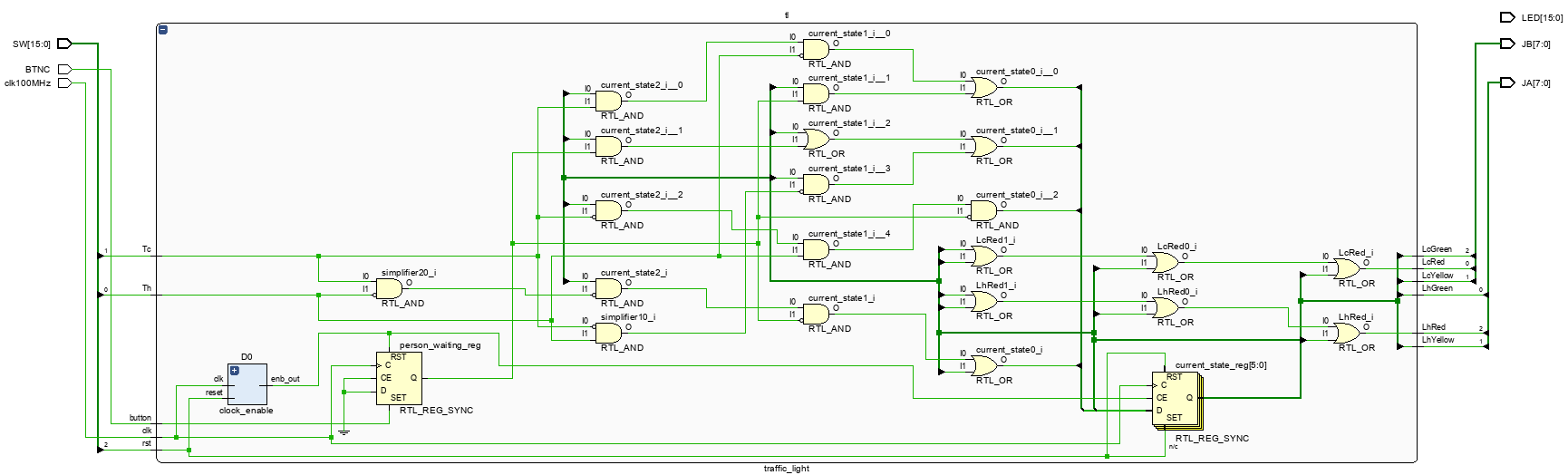
This guarantees that only one direction is green at a time, and both lights are red during transitions.

More detailed information including logic derivation steps and full SystemVerilog design can be found in the following attachments *Images 1 & 2, Schematic 1, and Code 1.*





**Images 1 & 2:** Sketch Design of the Traffic Light FSM through one-hot encoding and related truth-tables.



**Schematic 1.** The Final Schematic of traffic\_light.sv Module, Which Also Includes How It Is Connected to FPGA Board in the Top Module.

module traffic\_light( input logic rst, clk, Tc, Th , button, output logic LcRed,LcYellow,LcGreen,LhGreen,LhYellow,LhRed

);

logic [5:0] current\_state;

logic [5:0] next\_state;

logic enable; // Declare enable signal

clock\_enable #(.PERIOD\_MS(500)) D0(.clk(clk), .reset(rst),

.enb\_out(enable));

logic simplifier1;

logic simplifier2;

nand(simplifier1, ~Tc, Th);

nand(simplifier2, Tc, ~Th);

always\_ff @(posedge clk) begin

if (rst == 0)

if (enable)

current\_state <= next\_state ;

else

current\_state <= current\_state ;

else

current\_state <= 6'b000001;

end

logic person\_waiting;

always\_ff @(posedge clk) begin

if (button)

person\_waiting <= button;

else

if (enable)

person\_waiting <= 0;

end

always\_comb begin

next\_state[5] = current\_state[4];

next\_state[4] = current\_state[3] & ~Tc & Th & ~person\_waiting;

next\_state[3] = current\_state[2] | current\_state[3] & person\_waiting | current\_state[3] & simplifier1;

next\_state[2] = current\_state[1];

next\_state[1] = current\_state[0] & Tc & ~Th | current\_state[0] & person\_waiting;

next\_state[0] = current\_state[0] & simplifier2 & ~person\_waiting | current\_state[5];

end

or(LhRed, current\_state[2], current\_state[3],current\_state[4],current\_state[5]);

assign LhYellow = current\_state[1];

assign LhGreen = current\_state[0];

assign LcGreen = current\_state[3];

assign LcYellow =current\_state[4];

or(LcRed, current\_state[0], current\_state[1],current\_state[2],current\_state[5]);

endmodule

**Code 1.** Full code of traffic\_light.sv module, containing the full logic/backbone of this lab.

**IMPLEMENTATION**

The lab10\_top module connects the FSM to the physical hardware interface of the Nexys A7 FPGA. It handles input mapping from the board's switches and pushbutton, and output mapping to the LEDs via PMOD connectors.

The clock input clk100MHz is passed directly into the FSM. Switches SW[0], SW[1], and SW[2] are assigned to the High Street traffic sensor (Th), Cattell Street traffic sensor (Tc), and reset (rst), respectively. The pedestrian button (BTNC) is passed as the button input.

Outputs from the FSM are connected to two PMOD headers. PMOD JB controls Cattell’s LEDs:

* JB[2] = Cattell Green
* JB[1] = Cattell Yellow
* JB[0] = Cattell Red

PMOD JA controls High’s LEDs:

* JA[0] = High Green
* JA[1] = High Yellow
* JA[2] = High Red

No additional logic is present in the top-level module. It serves purely as a wrapper that routes physical inputs and outputs to the traffic\_light FSM. The module instantiates the FSM once and provides it with all required signals.

The enable signal, button capture logic, state logic, and output assignments are entirely contained in the FSM module. This modular structure makes the design clean and easily testable.

module lab10\_top(

input logic clk100MHz,

input logic BTNC,

input logic [15:0] SW,

output logic [15:0] LED,

output logic [7:0] JA,

output logic [7:0] JB

);

logic rst, Tc, Th , button;

logic LcRed,LcYellow,LcGreen,LhGreen,LhYellow,LhRed;

assign Th = SW[0];

assign Tc = SW[1];

assign rst = SW[2];

assign button = BTNC;

assign JB[2] = LcGreen;

assign JB[1] = LcYellow;

assign JB[0] = LcRed;

assign JA[0] = LhGreen;

assign JA[1] = LhYellow;

assign JA[2] = LhRed;

traffic\_light tl(.rst(rst), .clk(clk100MHz), .Tc(Tc), .Th(Th) , .button(button),

.LcRed(LcRed),.LcYellow(LcYellow),.LcGreen(LcGreen),.LhGreen(LhGreen),.LhYellow(LhYellow),.LhRed(LhRed)

);

endmodule

**Code 2.** Full code of the lab10\_top.sv module, that handles implementation of the lab and integration with the FPGA board.

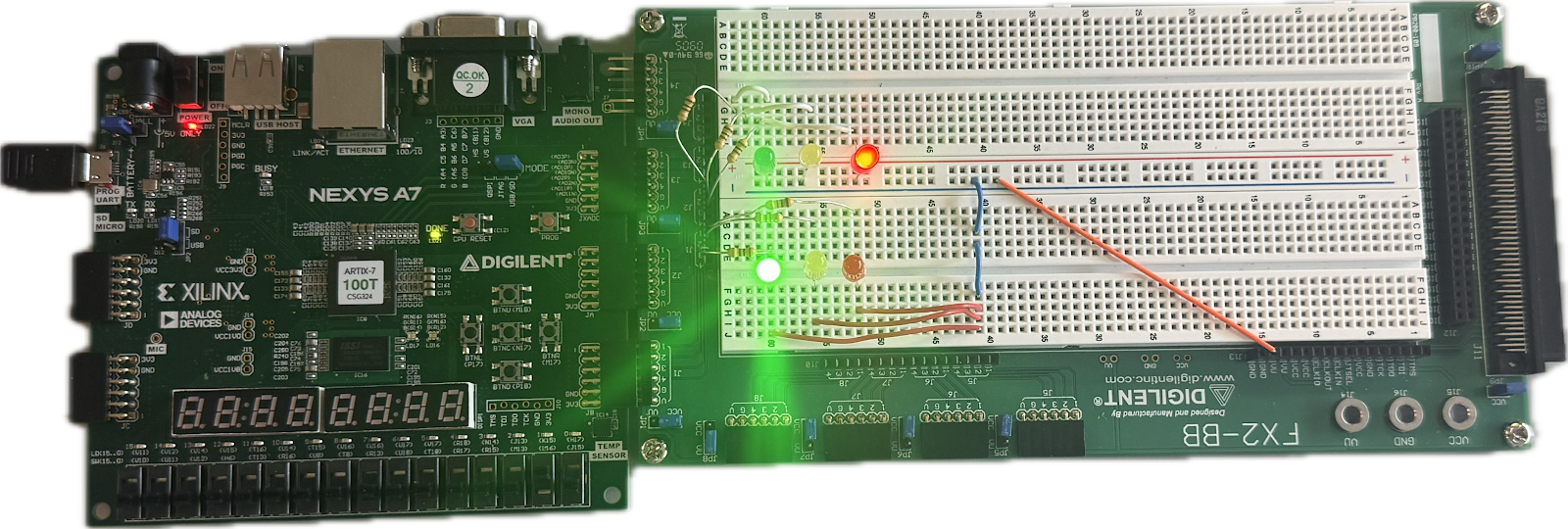
**TESTING/RESULTS**

Tests for this lab went as expected. Each LED was assigned a certain state and a next state based on its current one (as well as user input). The top row of LEDs represents Cattell Street and the bottom row is High Street. As it can be seen in demonstrations 1 and 2, the states for Scenario 1 would be that Cattell Street has a red light, but High Street has a green. Scenario 2 would be the opposite.

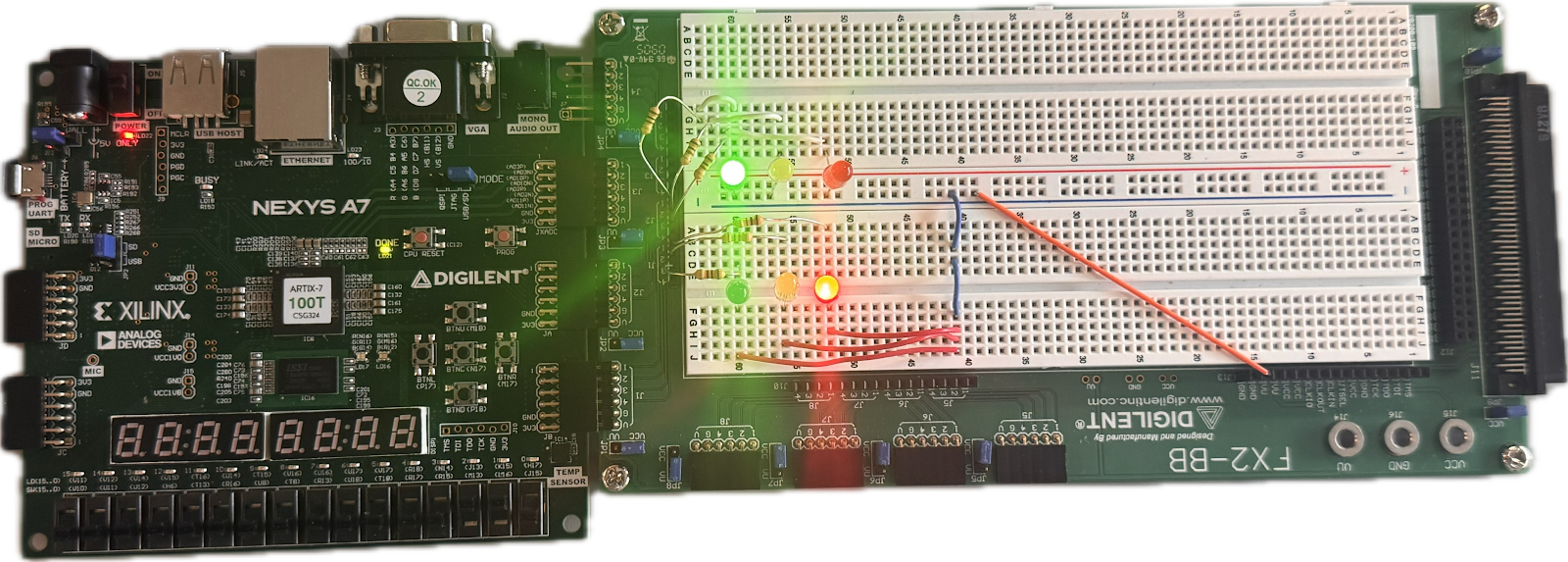
SW[0] is connected to Cattell Stso its default light sequence is as it appears in Demonstration 1.

SW[1] is connected to High Street, so its default light sequence is as it appears in Demonstration 2.

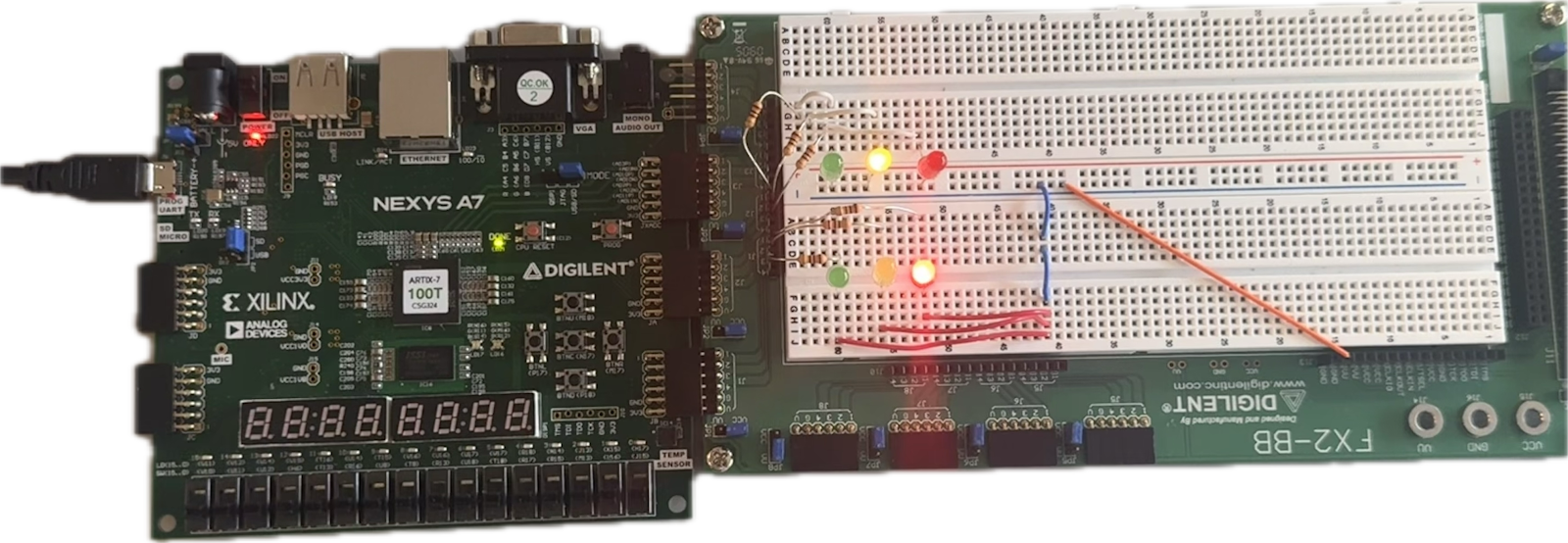
In Figure 3, it can be observed that Cattell Street is in the process of switching to a low state, and High Street a high state. This was triggered by the input BTNC (“button”) on the Nexys Board. The yellow LED (representing a “slow down” traffic light) sits between that transition. In Figure 4, BTNC’s wave pattern can be seen as referred earlier — which resets the current state of the FSM entirely.



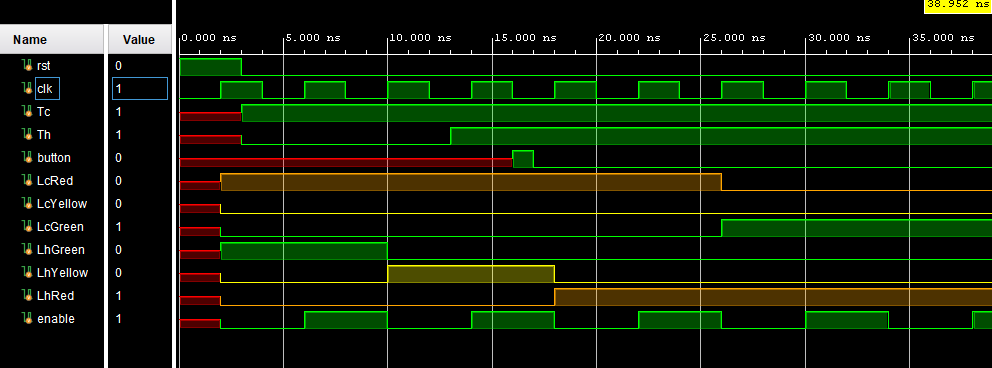
**Demonstration 1**: Cattell St default state (low)



**Demonstration 2**: High St default state (high)



**Demonstration 3:** Transition between High and Cattell St (switching high and low state)



**Figure 4:** Waveform for states

`timescale 1ns / 1ps

module lab10\_tb();

logic rst;

logic clk;

logic Tc;

logic Th;

logic button;

logic LcRed,LcYellow,LcGreen,LhGreen,LhYellow,LhRed;

always begin

clk = 0;

#2;

clk = 1;

#2;

end

// Instantiate the devices under test (DUTs)

traffic\_light tl(.rst(rst), .clk(clk), .Tc(Tc), .Th(Th) , .button(button),

.LcRed(LcRed),.LcYellow(LcYellow),.LcGreen(LcGreen),.LhGreen(LhGreen),.LhYellow(LhYellow),.LhRed(LhRed)

);

initial begin

$display("[ECE 211] Simulation starting...");

rst = 1;

#3;

rst = 0;

Tc = 1;

Th = 0;

#10;

Tc = 1;

Th = 1;

#3

button = 1;

#1

button = 0;

#22

$display("[ECE 211] Simulation complete...");

$stop;

end

endmodule

**Code** 3. Simulational Testbench for the Main Design Module traffic\_light.sv

**CONCLUSION/DISCUSSION**

The lab went as expected in the end. There was one error, however, and that was with the clock speed. Initially, it appeared as if there was no transition state that had Cattell St high with High St low, and vice versa. The problem ended up being with the frequency. While it was supposed to be 10 megahertz, the unit was set to “Hz” (Hertz) instead of “MHz” (megahertz). Due to the unit being smaller, the transition wasn’t visible to the human eye; therefore making the illusion that the state wasn’t occurring at all. During modular testing, the testbenches included events where “rst” (reset) was high and low, and how it affected the value of each light on both streets. In addition, BTNC, or button, is included in the test bench to represent the high and low state of both. All test have at least 1 nanosecond of delay.