Final Project: Traffic Light Controller

§1 Project Statement

Use system verilog to develop a controller unit for a simple two street intersection complete with corresponding lights and traffic sensors. Given that the configuration of the lights is finite, a state-machine approach is logical to describe the intersection's behavior. The lights will remain in their current state until the sensors on the opposing street trigger a change in traffic flow. In such a case, a transient state occurs which slows the current traffic prior to introducing opposing traffic.

§2 State Diagram

Given the sensory inputs and the procedural flow of some states logically into other states (e.g. green through red cycle) the following state truth table can be derived:

Present state	Next state	Output
state0	(0,0) → state1 (0,1) → state2 (1,0) → state0 (1,1) → state1	$green1 = 1$, $red2 = 1$ N-S street open
state1	state2	$green1 = 1$, $red2 = 1$
state2	state3	$green1 = 1$, $red2 = 1$
state3	state4	$yellow1 = 1$, $red2 = 1$ Unprotected left
state4	(0,0) → state5 (0,1) → state4 (1,0) → state6 (1,1) → state4	$red1 = 1$, $green2 = 1$
state5	state6	$red1 = 1$, $green2 = 1$
state6	state7	$red1 = 1$, $green2 = 1$
state7	state0	$red1 = 1$, $yellow2 = 1$

Figure 1: State truth table.

From this truth table, a more detailed flow diagram has been produced below.

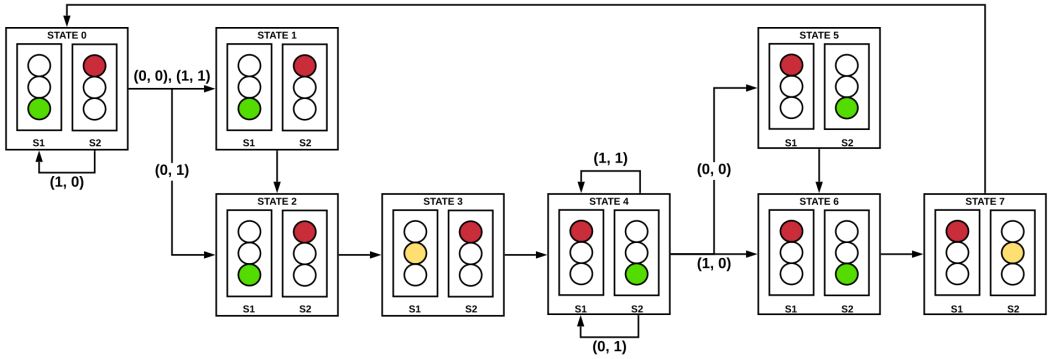

Figure 2: Traffic Light Controller flow diagram.

§3 Solution Approach

In order to describe the behavior of the controller, a local variable was used to define the current state. This variable is a customized enumeration that lists the possible states for the controller unit.

```
typedef enum {state0, state1, state2, state3, state4,
        state5, state6, state7} state_t;

// Define state enumeration variables
state_t state, next_state;
```

System Verilog allows for switch case statements to be applied to enumeration variables. Ergo, the remainder of the solution can be solved in a single large switch case against the current state. To prevent against confusing the system, the $next_state$ variable is used to delay updating the current state until the next switch case iteration like so:

```
always_ff @(posedge clk, posedge reset)
        begin
                if(reset)
                        state = state0;
                else
                        state = next_state;
        end
```

From there, the switch case explicitly defines the logic described by both the flow diagram as well as the state truth table. When simulating the controller, the clock must be modified to slow the system. Rather than adjusting the time signature, a clock divider can be implemented where T is a finite local parameter[1]

```
// Divide clock manually
  always
    begin
      clk = 1;
      #(T/2);
      clk = 0;
      #(T/2);
    end
```

Then the testbench can iterate between signal configurations by explicitly changing the sensory input and waiting for the states to change, again as a function of the local parameter T:

```
sen1 = 0;
sen2 = 0;
#(T*5);

sen1 = 1;
sen2 = 0;
#(T*5);

sen1 = 0;
sen2 = 1;
#(T*5);
...
```

[1]Similar to a *#define* statement in C-languages.

The code is then loaded into EDA Playground in an environment which can be accessed through the link included below. The Mentor Questa simulator is used and the wave output is included in the figure below.

`https://www.edaplayground.com/x/6NwQ`

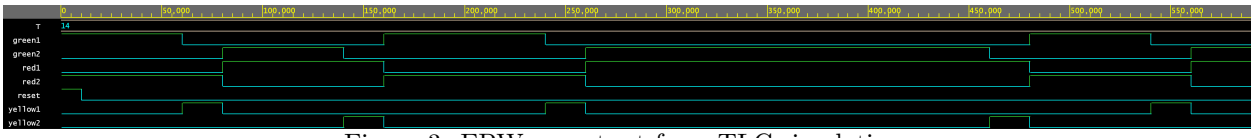

Figure 3: EPWave output from TLC simulation.