Ethan Nagelvooort
821234668

# Traffic Light Project Report

## Introduction:

This project implements a traffic light intersection through Verilog, using the Vivado IDE. It has four modules; one for the traffic lights, one for the pedestrian crosswalks, one for a red light sensor, and a top module. There is a test bench for each module, excluding the top module, to test each of them separately.  The test bench is used to input values required from each module to see if accurate outputs occur in the waveform. If so, then the modules were coded and implemented correctly in Verilog. The traffic light module dictates the configuration of the traffic lights, as in which traffic light, in all directions (south, north, east, west), is green, red, or yellow. The pedestrian crossing module dictates which crosswalk pedestrians are able to cross depending on the traffic light configuration. The red light module senses when a car runs a red light, and sends a signal to a camera to take a picture of that car. It uses the configuration of the traffic lights to determine which light is red. This project is state based, and mainly depends on the traffic light configurations as states. It is the traffic light module that determines which state the traffic lights are in, and the other two modules need to know what state the traffic lights are currently in in order to operate correctly.

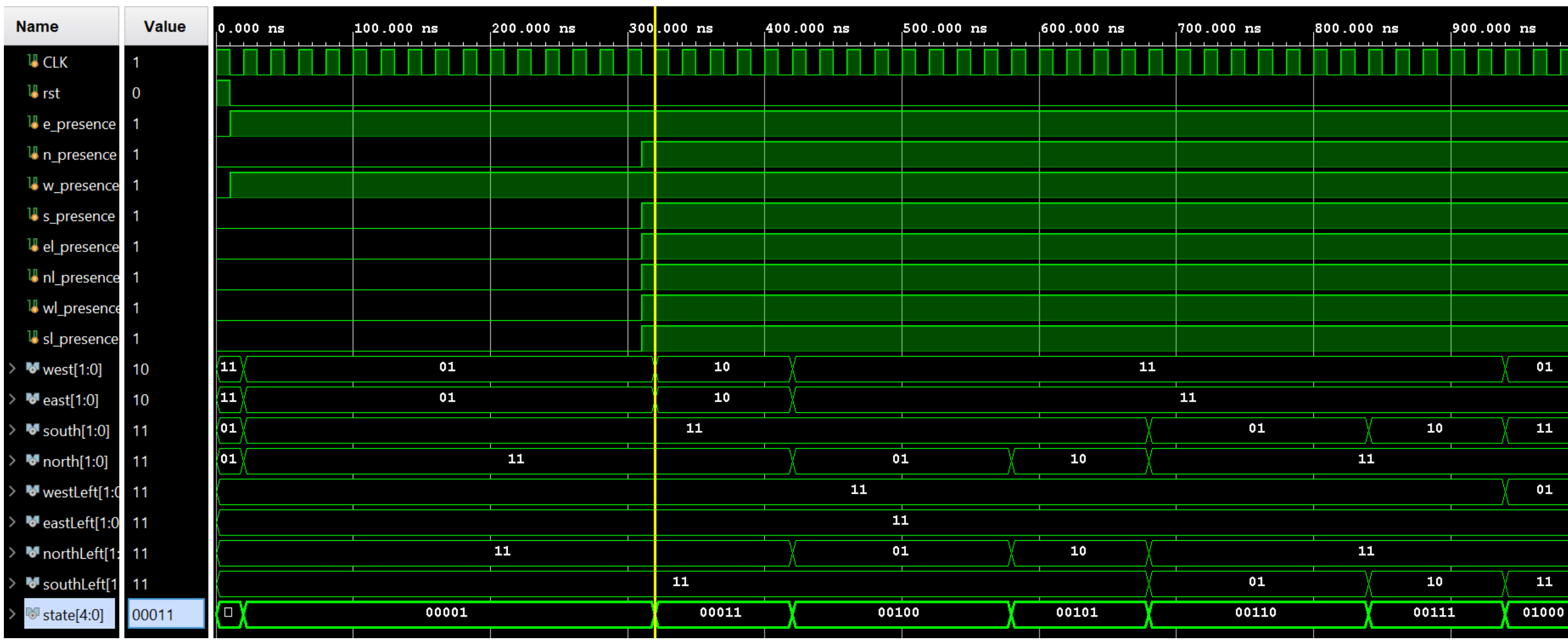
## Tool Set Used:

Vivado was used to code the modules and test benches of this project.

## Traffic Light module and its Test Bench:

The traffic light module takes in 10 inputs, a clock, an rst, and eight sensors for west, east, north, south, left lane of east, left lane of west, left lane of north, and left lane of south. It then has 9 outputs, one that represents the state and 8 that represent the lights of the intersection. I then set parameters, three of which represent green, yellow, and red for each of the light outputs. The other 16 states describe the traffic light configuration. 8 of these states are the green version of the configuration, an example of this type of configuration would be like “n_nl” which would represent the forward north lane and the left lane of the north side as green. This means that cars can go forward while the lights are in this configuration. The other 8 configurations describe the yellow version of the same configurations that were described as green. Hence, there is a configuration that describes the forward north lane and left lane of the north side as yellow. This module uses two arbitrary registers, one for a time counter, which dictates how long the traffic lights will be in a configuration, and one to keep track of the state. It is also split up into two always blocks and the first one is always at the positive edge of the clock. Then there is an if statement that occurs if rst is one and it initializes the registers with state initialized to “n_s”(north is green and south is green). Then, in an else statement, a case statement occurs that is used to describe each configuration based on what the state register equals. Since state was initialized at “n_s”, the case statement will start there. In this part of the case statement, there is an if statement that checks if cars are present in the lane using the sensors from the input. If there is, the module then checks if time_count is over than 6. If it is, it then checks the other sensors to see if there are cars in any other lane. This prevents the state from changing if there are no other cars in the other lanes since there is no need to change then. If so, time_count is set back to 0 and state is changed to n_s_y (north yellow and south yellow). If time_count was not greater than 6, then time_count is incremented and the state remains “n_s”. The manipulation of the time_count register also shows that this state will last for at least 6 clock cycles. Now, if the north sensor and south sensor did not detect any cars from the beginning if statement, then the state would have directly changed to e_w (east green and west green), skipping the n_s_y state. All states that describe an all green light configuration operate in this way that was just described. Now, to explain how cases that describe an all yellow light configuration, I will explain how the state n_s_y works. There is first an if statement that checks if time_count is greater than 3. If it is not greater than three, then time_count is incremented and the state remains n_s_y. This shows that states that describe an all yellow light configuration last for three clock cycles. Then, if time_count is greater than three, time_count is set back to 0 and state is changed to e_w. Then, after this always block, another is used that uses state as the changing edge. Another case statement occurs that describes each light configuration again. In each case, the outputs are described. The output state1 is set to the current state, and all the lights are set to the color it would be in the corresponding traffic light configuration.

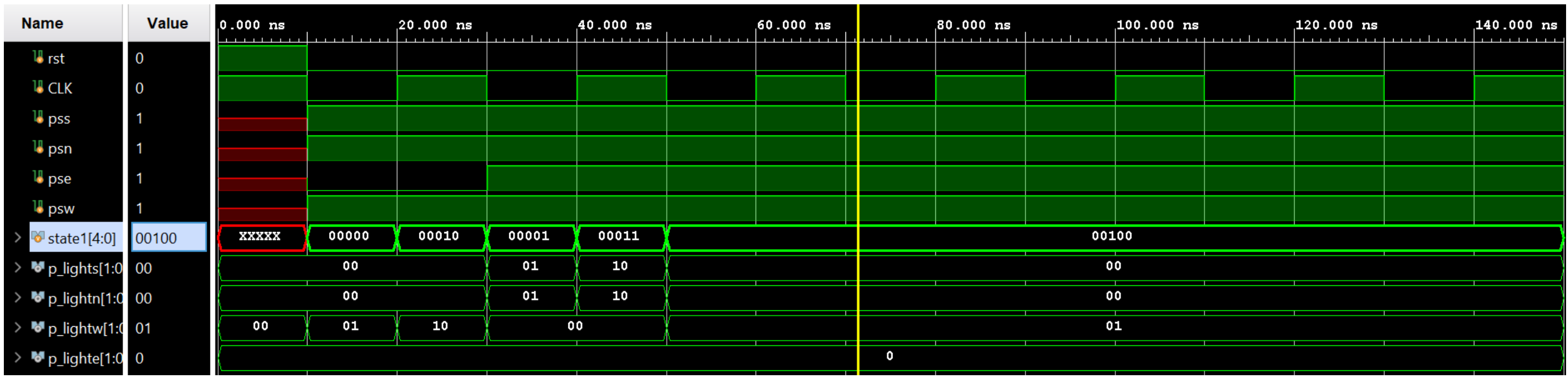
In the test bench for this module, “Test_bench”, first run the rst state. Then after ten clock cycles, I set only e_presence and w_presence to 1. This is so I can test that the module skips the states n_s and n_s_y since there are no cars at those sides. Also, so I can test that the state will not change since there are no cars present in the other sides. Then after 300 clock cycles, I set all the other presence variables to 1, meaning that there are cars on all sides. After running the test bench, I saw in my waveform that all the states configurations appeared with the correct lights. Also n_s and n_s_y was skipped, e_w was the current state for a long time like I wanted and all the other states appeared in the correct order. This shows that my module operates exactly as I intended. The waveform can be seen below.



## Pedestrian Crosswalk Module and its Test Bench:

In the next module, I implement how the pedestrian crosswalk operates. The inputs for this module were CLK, rst, state1 and four sensors for the four sides of the intersection. The outputs were the four pedestrian lights. These lights can have three different configurations: off, red hand, and walking man. I set up these configurations as parameters. I also bring in the 16 different state configurations from the traffic light module. This module uses one always statement and begins with an if statement that's true if rst is one and then initializes the outputs to off. Then in the else statement a case statement occurs with state1 as the input. In each state that describes a green traffic light configuration, if first turns off the necessary pedestrian lights, in other ones on the side where the cars are gonna cross. Then if the sensors detect a pedestrian on their side and it is safe to walk, that pedestrian light is set to walking man. If there is no pedestrian, the light is set to off. In the cases that involve the yellow configurations, the module checks if the pedestrian lights that are on the sides that are safe to cross are set to walking man. If they are, then these are set to red hand.

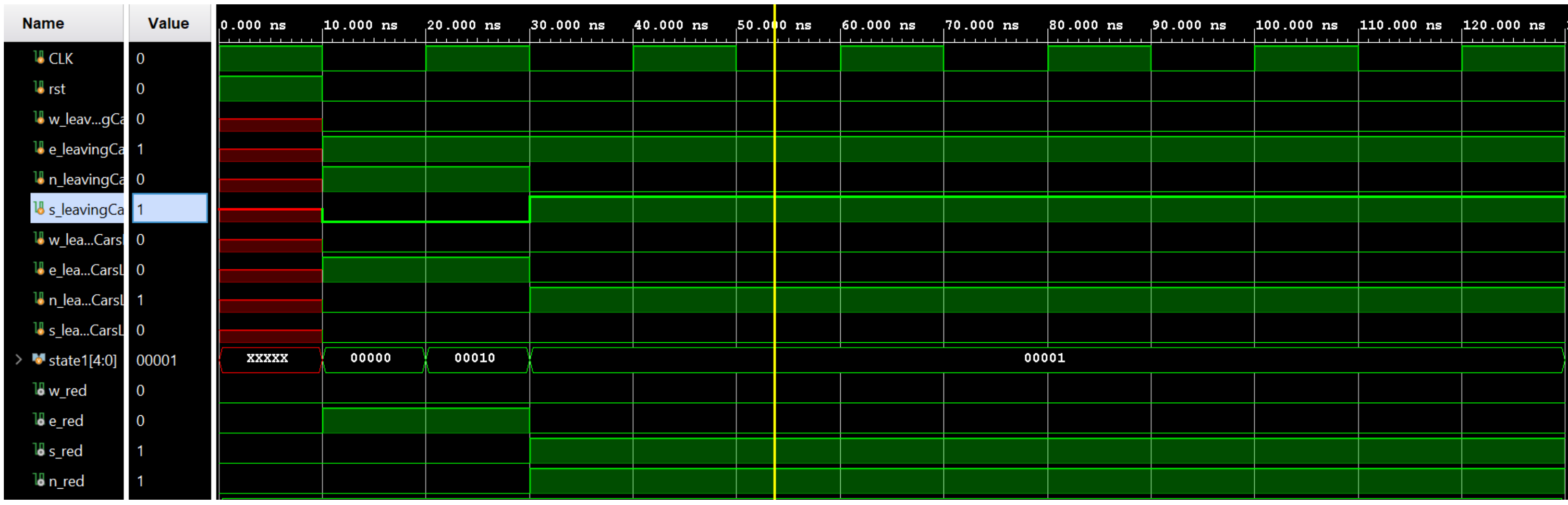
In the test bench for this module, “test_level1”, I first set rst to 1. I then set rst to 0 and state to n_s. I set all the pedestrian sensors to 1 except for the one on the east side. This means that for the duration of this state, p_lighte will be set to off, as well as p_lightn and p_lights. Then after 10 clock cycles, the state changes to n_s_y and then after another 10 clock cycles, it changes to e_w, e_w_y, and finally n_nl. During the duration of those states, p_lighte should be off since there has been no state it can be on in. When I ran this test bench, the correct p_light configuration corresponded to the states. This shows that my module was coded correctly.



## Red Light Module and its Test Bench:

In the module, red_light, I had the inputs: rst, CLK, w_leavingCars, e_leavingCars, n_leavingCars, s_leavingCars, e_leavingCarsL, w_leavingCarsL, n_leavingCarsL, s_leavingCars, and state1. The inputs that have "leaving" in the name senses when a car leaves their side. For example, w_leavingCarsL will become 1 when a car leaves the left lane on the west side. The outputs are w_red, s_red, n_red, and e_red. These are basically sensors for when a car runs a red light. For example, if a w_leavingCarsL and/or w_leavingCars were 1 and the light on the west and/or west left was red, w_red will become 1. The module is based on the positive edge of CLK and begins with an if statement for if rst is 1. In this statement, we set all the red sensors to 0. Then in a separate always block that is based on state1, the module utilizes a case statement that is based on state1. These states are the same as from the traffic light module, where they describe the light configuration of the intersection. In each case, I set the red sensors to 0 where the light in that state is green or yellow. Then for the rest of the red sensors, an if statement to detect if a car leaves the side that has a red light, using the inputs previously described. Then if a car does leave a side with a red light, then the corresponding sensor will become 1.

In the test bench, "test_bench2", I first set rst to 1 and then after 10 clock cycles, set it to 0. Then I change state to n_s and set all leaving inputs to 0 except for the ones that represent north, east and east left. Thus e_red should be one here. Then after 10 clock cycles, I change the state to n_s_y and e_red should still remain 1. Then after another  10 clock cycles, the state changes  to e_w and leaving inputs east left to 0 and south and north to 1. Thus, n_red and s_red should change to 1. When I run the test bench,  all the desired outputs come out on the waveform, hence showing that my code is working properly.

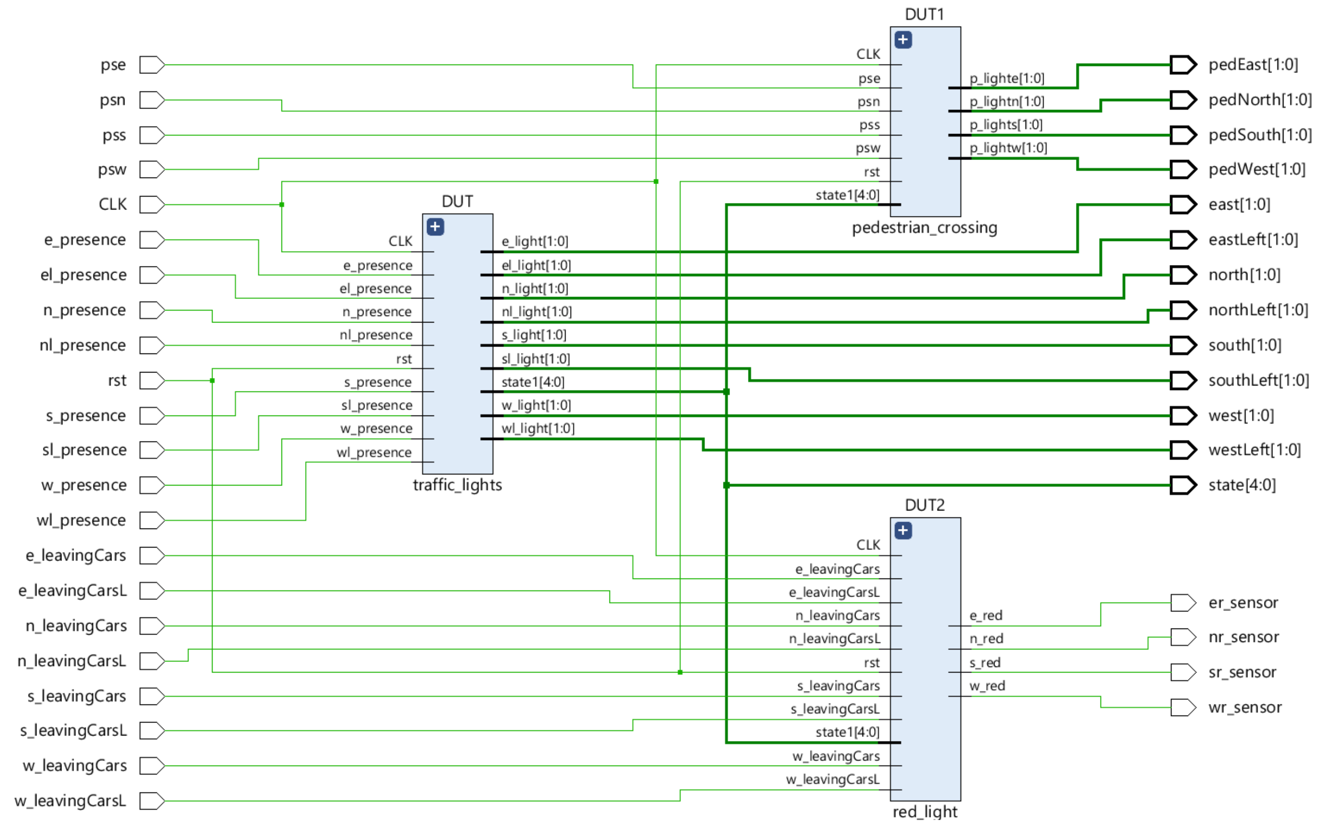


## Top Module:

In order to properly connect all the modules together, I created a top module called "top_level". To create this module, I first put all the inputs and outputs that I needed to instantiate the three modules. I then created wires that represent all the outputs. This is useful when there are outputs that serve as inputs for a different module. Since the state1 output from the traffic light module is used as an input for the other two modules, a wire is useful here. After instantiating all the modules in my top module, I then assign the outputs to the wires that represent them. This connects the outputs of each module to the outputs of the top module.

## Block Design:

______By creating a top level module that connects the three other modules; traffic light, pedestrian crossing, and red light, I was able to get a block design generated by Vivado.



## What I Would Have Done Differently:

There are a number of things that I would have done differently now that I have finished the project. One is that I never got to include a car counter module into the project. The reason for this is that I ran into too many errors when I first made it and I ran out of time to complete this module. I would have compensated for the short time that I had and began working on this module sooner. Another thing I would have done differently was to figure out a way to include the left turns without adding a bunch of more states to the traffic lights module. This would have either been a different module or just a better implementation in the traffic light module. The reason for this is that I feel the traffic light module is too long and complicated. Having another module or figuring out a simpler way for traffic lights to use left turns would have been better. Lastly, I would have figured out a way for the Pedestrian Crossing module to actually count down how many clock cycles there are before p_lights changes from the red hand to off. This would replicate real pedestrian crosswalks more accurately. I would have done it by having this module have four more outputs that would be a counter/timer that starts at a certain number and would decrease by one during the time the red hand is shown. The four more outputs would represent counters/timers for each direction.

## Hours Spent:

______I spent a total of around 20 hours on this project. The module, traffic lights, took me 7 hours to figure out and complete. Then I spent around 1 hour completing its test bench and running it. After finding errors, I spent around another hour trying to fix these errors. Then the module, pedestrian crossing, took me 3 hours to figure out and code. Its test bench took me 1 hour to code and after finding errors, I spent around another hour fixing this module. I then created a top module that took around 2 hours to figure out and code. I then created the red light module, which took about 2 hours to make. Creating its test bench and fixing its errors took me another 2 hours. I also modified my top module to include red light in that 2 hours as well.

## Conclusion:

This project is about a traffic intersection and consists of three parts: the traffic lights, the pedestrian crosswalk, and sensors that detect if cars run a red light. These parts are implemented in modules that are all state based and the states mainly represent the traffic light configurations. Other modules include the three test benches for those three modules and a top level module that connects them all. Throughout my 20 hours of working on this project I was able to learn a lot. This project showed me exactly how verilog is coded and just how different it is from normal sequential coding. One of the most important things this project has shown me is how to connect modules through a top level module. This includes having one module's output become another module's input through the manipulation of wires. It also taught me how to detect errors through observing the waveform. I spent a lot of time examining my waveforms to make sure my modules were operating correctly.  This project has really opened my eyes about verilog coding.

## Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Ethan Nagelvoort
//
// Create Date: 04/05/2020 11:21:26 AM
// Design Name:
// Module Name: traffic_lights
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

/*
CLK - clock
rst - reset
w_presence - senses if there is a car at the west light
e_presence - senses if there is a car at the east light
n_presence - senses if there is a car at the north light
s_presence - senses if there is a car at the south light
w_light - west traffic light
e_light - east traffic light
s_light - south traffic light
n_light - north traffic light
wl_light - west left traffic light
el_light - east left traffic light
sl_light - south left traffic light
nl_light - north left traffic light

*/
module traffic_lights(input CLK, rst,
w_presence, e_presence, n_presence, s_presence, nl_presence, sl_presence, el_presence,
wl_presence,
output reg [1:0] w_light, e_light, s_light, n_light, wl_light, el_light, sl_light, nl_light,
output reg [4:0] state1);
reg [4:0] state; //determines the next state of the traffic lights
parameter [1:0] g = 2'b01; //green
parameter [1:0] y = 2'b10; // yellow
parameter [1:0] r = 2'b11; //red
parameter [4:0] n_s = 5'b00000; //north and south light
parameter [4:0] e_w = 5'b00001; // east and west light
parameter [4:0] n_s_y = 5'b00010;//north and south yellow light
parameter [4:0] e_w_y = 5'b00011;//east and west yellow light
parameter [4:0] n_nl = 5'b00100;//north and north left
parameter [4:0] n_nl_y = 5'b00101;//north and north left yellow
parameter [4:0] s_sl = 5'b00110;//south and south left
parameter [4:0] s_sl_y = 5'b00111;//south and south left yellow
parameter [4:0] w_wl = 5'b01000;//west and west left
parameter [4:0] w_wl_y = 5'b01001;//west and west left yellow
parameter [4:0] e_el = 5'b01010;//east and east left
parameter [4:0] e_el_y = 5'b01011;//east and east left yellow
parameter [4:0] sl_nl = 5'b01100;//south left and north left
parameter [4:0] sl_nl_y = 5'b01101;//south left and north left yellow
parameter [4:0] el_wl = 5'b01110;//east left and west left
parameter [4:0] el_wl_y = 5'b01111;//east left and west left yellow

reg [3:0] time_count;//time counter for how long traffic lights will be on
always @ (posedge CLK) begin
if(rst==1'b1) begin//if rst is on, state is n_s
state <= n_s;
time_count <= 3'b000;
end
else begin

case(state)
//state for north and south light-green
n_s: begin
if(n_presence>1'b0 || s_presence>1'b0) begin//check is sensors detect cars in these lanes
if((time_count > 3'b110))
begin
if(e_presence> 1'b0 || w_presence> 1'b0  || el_presence> 1'b0 ||
wl_presence> 1'b0 || sl_presence> 1'b0 || nl_presence> 1'b0) begin //check if there are cars in any
other lane
time_count <= 3'b000;//set time counter back to zero
state <= n_s_y; // state goes to north and south yellow light
end end
else
begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= n_s;//state does not change
end
end
```

```
if(n_presence == 1'b0 && s_presence == 1'b0)
begin
state <= e_w;
end end

//state for north and south light - yellow
n_s_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= e_w; //state goes to east west light
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= n_s_y;//state does not change
end end

//state for east and west light-green
e_w: begin
if(e_presence > 1'b0 || w_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if( s_presence> 1'b0 || n_presence> 1'b0 || el_presence> 1'b0 ||
wl_presence> 1'b0 || sl_presence> 1'b0 || nl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= e_w_y; // state goes to east and west yellow light
end end
else begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= e_w;//state does not change
end
end
if(e_presence == 1'b0 && w_presence == 1'b0) begin
state <= n_s;
end end

//state for north and south light - yellow
e_w_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
```

```
state <= n_nl; //state goes to north north left light
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= e_w_y;//state does not change
end end

//state for north and north left green
n_nl: begin
if(n_presence > 1'b0 || nl_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if(e_presence> 1'b0 || w_presence> 1'b0 || s_presence> 1'b0 || el_presence> 1'b0 ||
wl_presence> 1'b0 || sl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= n_nl_y; // state goes to north north left yellow light
end end
else begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= n_nl;//state does not change
end
end
if(n_presence == 1'b0 && nl_presence == 1'b0) begin
state <= s_sl; // go directly to next state skip yellow state
end end

//state for north north left - yellow
n_nl_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= s_sl; //state goes to south south left
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= n_nl_y;//state does not change
end end

//state for south and south left green
s_sl: begin
```

```
if(s_presence > 1'b0 || sl_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if(e_presence> 1'b0 || w_presence> 1'b0 ||  n_presence> 1'b0 || el_presence> 1'b0 ||
wl_presence> 1'b0 || nl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= s_sl_y; // state goes to south south left yellow light
end end
else begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= s_sl;//state does not change
end
end
if(s_presence == 1'b0 && sl_presence == 1'b0) begin
state <= w_wl; // go directly to next state skip yellow state
end end

//state for south south left - yellow
s_sl_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= w_wl; //state goes to west west left
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= s_sl_y;//state does not change
end end

//state for west and west left green
w_wl: begin
if(w_presence > 1'b0 || wl_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if(e_presence> 1'b0 ||  s_presence> 1'b0 || n_presence> 1'b0 || el_presence> 1'b0 ||
 sl_presence> 1'b0 || nl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= w_wl_y; // state goes to west west left yellow light
end end
else begin
```

```
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= w_wl;//state does not change
end
end
if(w_presence == 1'b0 && wl_presence == 1'b0) begin
state <= e_el; // go directly to next state skip yellow state
end end

//state for west west left - yellow
w_wl_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= e_el; //state goes to east east left
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= w_wl_y;//state does not change
end end

//state for east and east left green
e_el: begin
if(e_presence > 1'b0 || el_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if( w_presence> 1'b0 || s_presence> 1'b0 || n_presence> 1'b0 ||
wl_presence> 1'b0 || sl_presence> 1'b0 || nl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= e_el_y; // state goes to east east left yellow light
end end
else begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= e_el;//state does not change
end
end
if(e_presence == 1'b0 && el_presence == 1'b0) begin
state <= sl_nl; // go directly to next state skip yellow state
end end

//state for east east left - yellow
```

```
e_el_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= sl_nl; //state goes to south left north left
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= e_el_y;//state does not change
end end

//state for south left and north left green
sl_nl: begin
if(sl_presence > 1'b0 || nl_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if(e_presence> 1'b0 || w_presence> 1'b0 || s_presence> 1'b0 || n_presence> 1'b0 || el_presence>
1'b0 ||
wl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= sl_nl_y; // state goes to south left north left yellow light
end end
else begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= sl_nl;//state does not change
end
end
if(sl_presence == 1'b0 && nl_presence == 1'b0) begin
state <= el_wl; // go directly to next state skip yellow state
end end

//state for south left and north left - yellow
sl_nl_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= el_wl; //state goes to east left west left
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= sl_nl_y;//state does not change
```

```
end end

//state for east left and west left green
el_wl: begin
if(el_presence > 1'b0 || wl_presence > 1'b0) begin
if(time_count > 3'b110)
begin
if(e_presence> 1'b0 || w_presence> 1'b0 || s_presence> 1'b0 || n_presence> 1'b0 || sl_presence>
1'b0 || nl_presence> 1'b0) begin
time_count <= 3'b000;//set time counter back to zero
state <= el_wl_y; // state goes to east left west left yellow light
end end
else begin
time_count <= time_count + 3'b001; // increment timer, will be on for at least 4 clock cycles
state <= el_wl;//state does not change
end
end
if(el_presence == 1'b0 && wl_presence == 1'b0) begin
state <= n_s; // go directly to next state skip yellow state
end end

//state for east left and west left - yellow
el_wl_y: begin
if (time_count > 3'b011) begin
time_count <= 3'b000;//set time counter back to zero
state <= n_s; //state goes to north south
end
else begin
time_count <= time_count + 3'b001;//increment time count, yellow light last for 3 clock cycles
state <= el_wl_y;//state does not change
end end

endcase
end end


always @(state) begin //always block with states, will describe light color
case(state)
```

```
/*
In each state, we set state1 to what the current state is and set the lights to what they are
according the the state
*/
n_s: begin//north and south light is green
state1 = n_s;
n_light = g;
s_light = g;
e_light = r;
w_light = r;

nl_light = r;
sl_light = r;
el_light = r;
wl_light = r;
end

n_s_y:begin//north and south light is yellow
state1 = n_s_y;
n_light = y;
s_light = y;
e_light = r;
w_light = r;

nl_light = r;
sl_light = r;
el_light = r;
wl_light = r;
end

e_w:begin //east and west light is green
state1 = e_w;
n_light = r;
s_light = r;
e_light = g;
w_light = g;

nl_light = r;
sl_light = r;
```

```
el_light = r;
wl_light = r;
end

e_w_y: begin//east and west ligh is yellow
state1 = e_w_y;
n_light = r;
s_light = r;
e_light = y;
w_light = y;

nl_light = r;
sl_light = r;
el_light = r;
wl_light = r;
end

n_nl: begin //north and north left
state1 = n_nl;
n_light = g;
s_light = r;
e_light = r;
w_light = r;

nl_light = g;
sl_light = r;
el_light = r;
wl_light = r;
end

n_nl_y: begin // north and north left yellow
state1 = n_nl_y;
n_light = y;
s_light = r;
e_light = r;
w_light = r;

nl_light = y;
sl_light = r;
```

```
el_light = r;
wl_light = r;
end

s_sl: begin // south and south left
state1 = s_sl;
n_light = r;
s_light = g;
e_light = r;
w_light = r;

nl_light = r;
sl_light = g;
el_light = r;
wl_light = r;
end

s_sl_y: begin // south and south left yellow
state1 = s_sl_y;
n_light = r;
s_light = y;
e_light = r;
w_light = r;

nl_light = r;
sl_light = y;
el_light = r;
wl_light = r;
end

w_wl: begin // west and west left
state1 = w_wl;
n_light = r;
s_light = r;
e_light = r;
w_light = g;

nl_light = r;
sl_light = r;
```

```
el_light = r;
wl_light = g;
end

w_wl_y: begin // west and west left yellow
state1 = w_wl_y;
n_light = r;
s_light = r;
e_light = r;
w_light = y;

nl_light = r;
sl_light = r;
el_light = r;
wl_light = y;
end

e_el: begin // east and east left
state1 = e_el;
n_light = r;
s_light = r;
e_light = g;
w_light = r;

nl_light = r;
sl_light = r;
el_light = g;
wl_light = r;
end

e_el_y:begin // east and east left yellow
state1 = e_el_y;
n_light = r;
s_light = r;
e_light = y;
w_light = r;

nl_light = r;
sl_light = r;
```

```
el_light = y;
wl_light = r;
end

sl_nl: begin// south left and north left
state1 = sl_nl;
n_light = r;
s_light = r;
e_light = r;
w_light = r;

nl_light = g;
sl_light = g;
el_light = r;
wl_light = r;
end

sl_nl_y: begin// south left and north left yellow
state1 = sl_nl_y;
n_light = r;
s_light = r;
e_light = r;
w_light = r;

nl_light = y;
sl_light = y;
el_light = r;
wl_light = r;
end

el_wl: begin // west left and east left
state1 = el_wl;
n_light = r;
s_light = r;
e_light = r;
w_light = r;

nl_light = r;
sl_light = r;
```

```
el_light = g;
wl_light = g;
end

el_wl_y:begin // west left and east left yellow
state1 = el_wl_y;
n_light = r;
s_light = r;
e_light = r;
w_light = r;

nl_light = r;
sl_light = r;
el_light = y;
wl_light = y;
end

endcase
end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/05/2020 11:34:24 PM
// Design Name:
// Module Name: Test_bench
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
////////////////////////////////////////////////////////////////////////////////


module Test_bench;
reg CLK, rst, e_presence, n_presence, w_presence, s_presence,
el_presence, nl_presence, wl_presence, sl_presence;
wire [1:0] west, east, south, north, westLeft, eastLeft,
northLeft, southLeft;
wire [4:0] state;

//instantiate module
top_level DUT (.CLK(CLK), .rst(rst),.n_presence(n_presence), .s_presence(s_presence),
.w_presence(w_presence),.e_presence(e_presence),
.west(west), .east(east), .south(south), .north(north),
.state(state), .westLeft(westLeft), .eastLeft(eastLeft),
.northLeft(northLeft), .southLeft(southLeft),  .wl_presence(wl_presence),
.el_presence(el_presence), .sl_presence(sl_presence),
.nl_presence(nl_presence));

initial begin
CLK <= 1;
forever #10 CLK <= ~CLK;
 end
 initial
 begin
  rst <= 1'b1;
  e_presence <= 1'b0;
  w_presence <= 1'b0;
  n_presence <= 1'b0;
  s_presence <= 1'b0;
  el_presence <= 1'b0;
  wl_presence <= 1'b0;
  sl_presence <= 1'b0;
  nl_presence <= 1'b0;
  #10
  rst <=1'b0;
  e_presence <= 1'b1;
  w_presence <= 1'b1;
  n_presence <= 1'b0;//test to see if module skips n_s and n_s_y state since there is not cars there
  s_presence <= 1'b0;
  el_presence <= 1'b0;
  wl_presence <= 1'b0;
  sl_presence <= 1'b0;
  nl_presence <= 1'b0;
  #300//all other presences are 0 so the state e_w should stay for a long time
  //now there is a car in each side so all states should start to appear in order
  n_presence <= 1'b1;
  s_presence <= 1'b1;
  s_presence <= 1'b1;
  el_presence <= 1'b1;
  wl_presence <= 1'b1;
  sl_presence <= 1'b1;
  nl_presence <= 1'b1;
  #1000
  $stop;
 end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/06/2020 06:25:19 PM
// Design Name:
// Module Name: pedestrian_crossing
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
////////////////////////////////////////////////////////////////////////////

module pedestrian_crossing(input rst, CLK,  pss, psn, pse, psw,
input [4:0] state1,
output reg [1:0] p_lights, p_lightn, p_lightw, p_lighte);
parameter [4:0] n_s = 5'b00000; //north and south light
parameter [4:0] e_w = 5'b00001; // east and west light
parameter [4:0] n_s_y = 5'b00010;//north and south yellow light
parameter [4:0] e_w_y = 5'b00011;//east and west yellow light
parameter [4:0] n_nl = 5'b00100;//north and north left
parameter [4:0] n_nl_y = 5'b00101;//north and north left yellow
parameter [4:0] s_sl = 5'b00110;//south and south left
parameter [4:0] s_sl_y = 5'b00111;//south and south left yellow
parameter [4:0] w_wl = 5'b01000;//west and west left
parameter [4:0] w_wl_y = 5'b01001;//west and west left yellow
parameter [4:0] e_el = 5'b01010;//east and east left
parameter [4:0] e_el_y = 5'b01011;//east and east left yellow
parameter [4:0] sl_nl = 5'b01100;//south left and north left
parameter [4:0] sl_nl_y = 5'b01101;//south left and north left yellow
parameter [4:0] el_wl = 5'b01110;//east left and west left
parameter [4:0] el_wl_y = 5'b01111;//east left and west left yellow
parameter [1:0] off = 2'b00;//pedlight off
parameter [1:0] walking_man = 2'b01; //pedlight displays walking man
parameter [1:0] red_hand = 2'b10; //pedlight displays red hand
always @ (posedge CLK) begin
if(rst == 1'b1)begin
p_lights <= off;
p_lightn <= off;
p_lightw <= off;
p_lighte <= off;
end
end

always @(state1)begin //split it up into two always blocks so there would not be an offset of 1
clock cycle
case(state1)
/*
```

```
In each green-state, the p_lights pedestrains can't cross are set to off. Then if the pedestrian sensor detects a pedestrain
then the p_light in the corresponding direction turns to walking man. If it doesn't detect a pedestrian, then that p_light
is set to off.
*/
n_s:begin
p_lightn<= off;
p_lights<= off;
if(psw == 1'b1)
p_lightw <= walking_man;
else
p_lightw <= off;
if(pse == 1'b1)
p_lighte <= walking_man;
else
p_lighte <= off;
end

/*
In each yellow state, the case detects if the p_light(s) the pedestrians can cross is set to walking man.
If so, then that p_light is set to red_hand.
*/
n_s_y:begin
if(p_lightw == walking_man)
p_lightw <= red_hand;
if(p_lighte == walking_man)
p_lighte <= red_hand;
end

e_w:begin
p_lightw <= off;
p_lighte <= off;
if(psn == 1'b1)
p_lightn <= walking_man;
else
p_lightn <= off;
if(pss == 1'b1)
```

```
p_lights <= walking_man;
else
p_lights <= off;
end

e_w_y: begin
if(p_lightn == walking_man)
p_lightn <= red_hand;
if(p_lights == walking_man)
p_lights <= red_hand;
end

n_nl:begin
p_lightn<= off;
p_lights<= off;
p_lighte<= off;
if(psw == 1'b1)
p_lightw <= walking_man;
else
p_lightw <= off;
end

n_nl_y:begin
if(psw == 1'b1)
p_lightw <= red_hand;
end

s_sl:begin
p_lightw<= off;
p_lights<= off;
p_lightn<= off;
if(pse == 1'b1)
p_lighte <= walking_man;
else
p_lighte <= off;
end

s_sl_y:begin
if(pse == 1'b1)
```

```
p_lighte <= red_hand;
end

w_wl:begin
p_lightw<= off;
p_lightn<= off;
p_lighte<= off;
if(pss == 1'b1)
p_lights <= walking_man;
else
p_lights <= off;
end

w_wl_y:begin
if(pss == 1'b1)
p_lights <= red_hand;
end

e_el:begin
p_lightw<= off;
p_lights<= off;
p_lighte<= off;
if(psn == 1'b1)
p_lightn <= walking_man;
else
p_lightn <= off;
end

e_el_y:begin
if(psn == 1'b1)
p_lightn <= red_hand;
end

/*
In the rest of these state, it is impossible for pedestrians to cross, so all p_lights are off
*/
sl_nl:begin
p_lightn<= off;
p_lights<= off;
```

```
p_lighte<= off;
p_lightw<= off;
end

sl_nl_y:begin
p_lightn<= off;
p_lights<= off;
p_lighte<= off;
p_lightw<= off;
end

el_wl:begin
p_lightn<= off;
p_lights<= off;
p_lighte<= off;
p_lightw<= off;
end

el_wl_y:begin
p_lightn<= off;
p_lights<= off;
p_lighte<= off;
p_lightw<= off;
end
endcase
end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/03/2020 02:41:10 AM
// Design Name:
// Module Name: test_level1
// Project Name:
// Target Devices:
// Tool Versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//should be test_bench1
module test_level1;
reg rst, CLK,  pss, psn, pse, psw;
reg [4:0] state1;
wire [1:0] p_lights, p_lightn, p_lightw, p_lighte;
//instantiate module
pedestrian_crossing DUT1 (.CLK(CLK), .rst(rst), .pss(pss),.psn(psn),.pse(pse),
.psw(psw),.p_lights(p_lights),.p_lightn(p_lightn),.p_lightw( p_lightw),.p_lighte(p_lighte),
  .state1(state1));

parameter [4:0] n_s = 5'b00000; //north and south light
parameter [4:0] e_w = 5'b00001; // east and west light
parameter [4:0] n_s_y = 5'b00010;//north and south yellow light
parameter [4:0] e_w_y = 5'b00011;//east and west yellow light
parameter [4:0] n_nl = 5'b00100;//north and north left
parameter [4:0] n_nl_y = 5'b00101;//north and north left yellow
parameter [4:0] s_sl = 5'b00110;//south and south left
parameter [4:0] s_sl_y = 5'b00111;//south and south left yellow
parameter [4:0] w_wl = 5'b01000;//west and west left
parameter [4:0] w_wl_y = 5'b01001;//west and west left yellow
parameter [4:0] e_el = 5'b01010;//east and east left
parameter [4:0] e_el_y = 5'b01011;//east and east left yellow
parameter [4:0] sl_nl = 5'b01100;//south left and north left
parameter [4:0] sl_nl_y = 5'b01101;//south left and north left yellow
parameter [4:0] el_wl = 5'b01110;//east left and west left
parameter [4:0] el_wl_y = 5'b01111;//east left and west left yellow

initial begin
CLK <= 1;
```

```
forever #10 CLK <= ~CLK;
 end
 initial
 begin
 rst<= 1'b1;
 #10
 rst<= 1'b0;
 state1 <= n_s;
 pss <= 1'b1;
 pse <= 1'b0;//p_lighte should be set to off
 psn <= 1'b1;
 psw <= 1'b1;
 #10
 state1 <= n_s_y;
 #10
 state1 <= e_w;
 pse<=1'b1;// set pedestrian presence for east side to 1
 #10
 state1 <= e_w_y;
 #10
 state1 <= n_nl;
 #100
 $stop;
 end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/02/2020 11:14:01 PM
// Design Name:
// Module Name: red_light
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

/*
CLK-clock
rst - reset
w_leavingCars - detects if car leaves west side
e_leavingCars - detects if car leaves east side
n_leavingCars - detect if car leaves north side
s_leavingCars - detects if car leaves south side
w_leavingCarsL - detects if car leaves west left side
e_leavingCarsL - detects if car leaves east left side
n_leavingCarsL - detect if car leaves north left side
s_leavingCarsL - detects if car leaves south left side
state1 - current state/configuration of the traffic lights
w_red - sensor that detects if car runs red light on west or west left
e_red - sensor that detects if car runs red light on east or east left
n_red - sensor that detects if car runs red light on north or north left
s_red - sensor that detects if car runs red light on south or south left
*/
//if this sensor is 1, then picture is taken, sensor then reverts back to 0
//if any incoming car input is 1, then there is a car that ran a red light
module red_light(input CLK, rst,  w_leavingCars, e_leavingCars,
n_leavingCars, s_leavingCars, w_leavingCarsL, e_leavingCarsL,
 n_leavingCarsL, s_leavingCarsL,
input [4:0] state1,
output reg w_red, e_red, s_red, n_red);
parameter [1:0] g = 2'b01; //green
parameter [1:0] y = 2'b10; // yellow
parameter [1:0] r = 2'b11; //red
parameter [4:0] n_s = 5'b00000; //north and south light
parameter [4:0] e_w = 5'b00001; // east and west light
parameter [4:0] n_s_y = 5'b00010;//north and south yellow light
parameter [4:0] e_w_y = 5'b00011;//east and west yellow light
parameter [4:0] n_nl = 5'b00100;//north and north left
parameter [4:0] n_nl_y = 5'b00101;//north and north left yellow
parameter [4:0] s_sl = 5'b00110;//south and south left
parameter [4:0] s_sl_y = 5'b00111;//south and south left yellow
parameter [4:0] w_wl = 5'b01000;//west and west left
parameter [4:0] w_wl_y = 5'b01001;//west and west left yellow
parameter [4:0] e_el = 5'b01010;//east and east left
parameter [4:0] e_el_y = 5'b01011;//east and east left yellow
parameter [4:0] sl_nl = 5'b01100;//south left and north left
parameter [4:0] sl_nl_y = 5'b01101;//south left and north left yellow
parameter [4:0] el_wl = 5'b01110;//east left and west left
parameter [4:0] el_wl_y = 5'b01111;//east left and west left yellow
always @(posedge CLK) begin
if(rst == 1'b1)begin
w_red<= 1'b0;
e_red<= 1'b0;
s_red<= 1'b0;
n_red<= 1'b0;
end end
always @(state1)begin
case(state1)

/*
In each case, we first see set the sensors that represent the side with green lights to 0.
Then we have an if statement for each side where there is a red light. If there is a car leaving that side
then we set the sensor to that represents that side to 1, if not then it is set to 0.
*/

n_s: begin//north and south light is green
n_red <= 1'b0;
s_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
```

```
w_red <= 1'b0;
end

n_s_y:begin//north and south light is yellow
n_red <= 1'b0;
s_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
end

e_w:begin //east and west light is green
e_red <= 1'b0;
w_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
end

e_w_y: begin//east and west ligh is yellow
e_red <= 1'b0;
w_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
```

```
s_red <= 1'b0;
end

n_nl: begin //north and north left
n_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

n_nl_y: begin // north and north left yellow
n_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

s_sl: begin // south and south left
s_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
```

```
else
w_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

s_sl_y: begin // south and south left yellow
s_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

w_wl: begin // west and west left
w_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
```

```
e_red <= 1'b1;
else
e_red <= 1'b0;
end

w_wl_y: begin // west and west left yellow
w_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

e_el: begin // east and east left
e_red <= 1'b0;
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
end

e_el_y:begin // east and east left yellow
e_red <= 1'b0;
```

```
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
end

sl_nl: begin// south left and north left
if(s_leavingCars == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

sl_nl_y: begin// south left and north left yellow
if(s_leavingCars == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1)
```

```
n_red <= 1'b1;
else
n_red <= 1'b0;
if(w_leavingCars == 1'b1 || w_leavingCarsL == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(e_leavingCars == 1'b1 || e_leavingCarsL == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

el_wl: begin // west left and east left
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
else
n_red <= 1'b0;
if(w_leavingCars == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(e_leavingCars == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end

el_wl_y:begin // west left and east left yellow
if(s_leavingCars == 1'b1 || s_leavingCarsL == 1'b1)
s_red <= 1'b1;
else
s_red <= 1'b0;
if(n_leavingCars == 1'b1 || n_leavingCarsL == 1'b1)
n_red <= 1'b1;
```

```
else
n_red <= 1'b0;
if(w_leavingCars == 1'b1)
w_red <= 1'b1;
else
w_red <= 1'b0;
if(e_leavingCars == 1'b1)
e_red <= 1'b1;
else
e_red <= 1'b0;
end
endcase
end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/03/2020 01:43:06 AM
// Design Name:
// Module Name: test_bench2
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//tests red light module

module test_bench2;
reg CLK, rst, w_leavingCars, e_leavingCars,
n_leavingCars, s_leavingCars, w_leavingCarsL, e_leavingCarsL,
 n_leavingCarsL, s_leavingCarsL;
 reg [4:0] state1;
 wire w_red, e_red, s_red, n_red;
parameter [1:0] g = 2'b01; //green
parameter [1:0] y = 2'b10; // yellow
parameter [1:0] r = 2'b11; //red
parameter [4:0] n_s = 5'b00000; //north and south light
parameter [4:0] e_w = 5'b00001; // east and west light
parameter [4:0] n_s_y = 5'b00010;//north and south yellow light
parameter [4:0] e_w_y = 5'b00011;//east and west yellow light
parameter [4:0] n_nl = 5'b00100;//north and north left
parameter [4:0] n_nl_y = 5'b00101;//north and north left yellow
parameter [4:0] s_sl = 5'b00110;//south and south left
parameter [4:0] s_sl_y = 5'b00111;//south and south left yellow
parameter [4:0] w_wl = 5'b01000;//west and west left
parameter [4:0] w_wl_y = 5'b01001;//west and west left yellow
parameter [4:0] e_el = 5'b01010;//east and east left
parameter [4:0] e_el_y = 5'b01011;//east and east left yellow
parameter [4:0] sl_nl = 5'b01100;//south left and north left
parameter [4:0] sl_nl_y = 5'b01101;//south left and north left yellow
parameter [4:0] el_wl = 5'b01110;//east left and west left
parameter [4:0] el_wl_y = 5'b01111;//east left and west left yellow
//instantiate module
 red_light DUT3 (.CLK(CLK), .rst(rst), .w_leavingCars(w_leavingCars),
.e_leavingCars(e_leavingCars),
.n_leavingCars(n_leavingCars), .s_leavingCars(s_leavingCars),
.w_leavingCarsL(w_leavingCarsL), .e_leavingCarsL(e_leavingCarsL),
.n_leavingCarsL(n_leavingCarsL), .s_leavingCarsL(s_leavingCarsL),
.state1(state1), .w_red(w_red), .n_red(n_red), .s_red(s_red), .e_red(e_red));

initial begin
CLK <= 1;
forever #10 CLK <= ~CLK;
 end
 initial
 begin
 rst<= 1'b1;
 #10
 rst<=1'b0;
state1 <= n_s;
 n_leavingCars<=1'b1;
 s_leavingCars<=1'b0;
 w_leavingCars<=1'b0;
 e_leavingCars<=1'b1;//e_red is 1
 e_leavingCarsL<=1'b1;
 w_leavingCarsL<=1'b0;
 s_leavingCarsL<=1'b0;
 n_leavingCarsL<=1'b0;
 #10
 state1<=n_s_y;
 #10
 state1 <= e_w;
 n_leavingCars<=1'b0;
 s_leavingCars<=1'b1;//s_red is 1
 w_leavingCars<=1'b0;
 e_leavingCars<=1'b1;
 e_leavingCarsL<=1'b0;
 w_leavingCarsL<=1'b0;
 s_leavingCarsL<=1'b0;
 n_leavingCarsL<=1'b1;//n_red is 1
 #100
 $stop;
 end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/06/2020 06:46:03 PM
// Design Name:
// Module Name: top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_level(input CLK, rst, pss, psn, pse, psw,
w_presence, e_presence, n_presence, s_presence, wl_presence, el_presence, nl_presence,
sl_presence,
 w_leavingCars, e_leavingCars, n_leavingCars, s_leavingCars,
 w_leavingCarsL, e_leavingCarsL, n_leavingCarsL, s_leavingCarsL,
output [1:0] west, east, south, north, pedWest, pedEast, pedSouth, pedNorth, westLeft, eastLeft,
southLeft, northLeft,
output [4:0] state,
output wr_sensor, nr_sensor, er_sensor, sr_sensor);
wire[1:0] w_light, e_light, s_light, n_light, p_lights, p_lightn, p_lightw, p_lighte, wl_light,
el_light, sl_light,
nl_light;
wire[4:0] state1;
wire w_red, e_red, n_red, s_red;
//instantiating traffic lights
traffic_lights DUT (.CLK(CLK), .rst(rst), .w_presence(w_presence),
.e_presence(e_presence), .n_presence(n_presence), .s_presence(s_presence),
.w_light(w_light), .e_light(e_light), .s_light(s_light),
.n_light(n_light),.state1(state1), .wl_light(wl_light),.el_light(el_light),.sl_light(sl_light),
.nl_light(nl_light), .wl_presence(wl_presence), .el_presence(el_presence),
.sl_presence(sl_presence),
.nl_presence(nl_presence));
//instantiating pedestrian Crossing
pedestrian_crossing DUT1 (.CLK(CLK), .rst(rst), .pss(pss),.psn(psn),.pse(pse),
.psw(psw),.p_lights(p_lights),.p_lightn(p_lightn),.p_lightw( p_lightw),.p_lighte(p_lighte),
  .state1(state1));
//instantiating red light
red_light DUT2 (.CLK(CLK), .rst(rst), .w_leavingCars(w_leavingCars),
.e_leavingCars(e_leavingCars),
.n_leavingCars(n_leavingCars), .s_leavingCars(s_leavingCars),
.w_leavingCarsL(w_leavingCarsL), .e_leavingCarsL(e_leavingCarsL),
.n_leavingCarsL(n_leavingCarsL), .s_leavingCarsL(s_leavingCarsL),
.state1(state1), .w_red(w_red), .n_red(n_red), .s_red(s_red), .e_red(e_red));
//set wires to outputs
assign state = state1;
assign pedWest = p_lightw;
assign pedEast = p_lighte;
assign pedSouth = p_lights;
assign pedNorth = p_lightn;
assign west = w_light;
assign east = e_light;
assign north = n_light;
assign south = s_light;
assign westLeft = wl_light;
assign eastLeft = el_light;
assign northLeft = nl_light;
assign southLeft = sl_light;
assign wr_sensor = w_red;
assign er_sensor = e_red;
assign sr_sensor = s_red;
assign nr_sensor = n_red;
endmodule
```