# CS M152B Lab 2

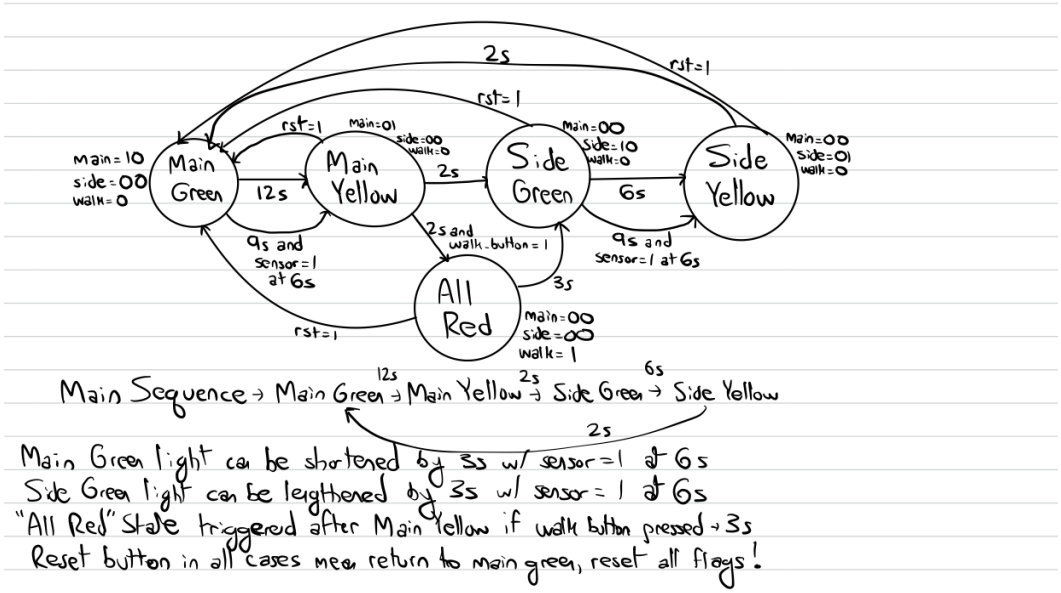
Akrit Shrikant, Patrick Parsegian

## Introduction:

In this lab, we implemented a traffic light controller that controlled a main street, side street, walk light, and sensors. We utilized a finite state machine to implement our design. We implemented the red, yellow, and green lights for both the main and side street while also putting together a working pedestrian walk button with a walklight. The walk button triggers both the main and side street's lights to turn red for 3 seconds only when the main street's yellow light has finished displaying. Our sensor signal, which represents high levels of traffic on the side street, will shorten the length of the main street's green light and make the side street's light remain green for longer if it is enabled. Finally, we implemented a reset button which, regardless of the state of the streetlights at the time, sets the main street light back to green, the side street light back to red, and restarts the progression of the lights.

## Implementation:

An explanation of each state can be seen in the diagram. The normal progression can be altered with the presence of either the sensor or walk_button flags, as explained in the image.



```
module traffic_controller(
    input clk,
    input walk_button,
    input sensor,
    input rst,
    output reg [1:0] main_light,
    output reg [1:0] side_light,
    output walk_light
    );

    parameter SIZE = 3;
    parameter GREEN = 2'b10, YELLOW = 2'b01, RED = 2'b00;
    parameter main_green = 3'b000, main_yellow = 3'b001, side_green = 3'b010, side_yellow = 3'b011, all_red = 3'b100;

    reg [SIZE-1:0] state = main_green;
    reg [SIZE-1:0] next_state;

    integer count = 0;
    integer walk_on = 0;
    integer walk_light = 0;
    integer main_target = 12;
    integer side_target = 6;

    //State Control
    always @(posedge clk) begin
        if (rst == 1) begin
            count = 0;
            state <= main_green;
        end
        else if (walk_button == 1) begin
            walk_on = 1;
        end
    end
```

In this screenshot, we have our initial design of our state machine. Here, we implement the state control to check for the different inputs of the reset switch and the walk button. We also initialize parameters for the lights, setting GREEN to the bits 10, YELLOW to 01, and RED to 00. This will be visible in our waveforms.

```
//OUTPUTS
always @(posedge clk) begin
    if (rst == 1) begin
        main_light <= GREEN;
        side_light <= RED;
        count = count + 1;
    end
    else begin
        case(state)
            main_green: begin
                main_light <= GREEN;
                side_light <= RED;
                count = count + 1;
                if (count == 6) begin
                    if (sensor == 1) begin
                        main_target = 9;
                    end
                end
                if (count == main_target) begin
                    state <= main_yellow;
                    count = 0;
                    main_target = 12;
                end
            end
            main_yellow: begin
                main_light <= YELLOW;
                side_light <= RED;
                count = count + 1;
                if (count == 2) begin
                    if (walk_on == 1) begin
                        count = 0;
                        state <= all_red;
                    end
                    else begin
                        state <= side_green;
                        count = 0;
                    end
                end
            end
```

This is a screenshot of our output control. Here we decide on the outputs based on the control signals received. We designed our states as main_green, main_yellow, side_green, side_yellow, and all_red. By doing this, we are able to control the red light output within the green and yellow output of the opposite traffic light. We also implemented a count variable which keeps track of how many seconds have passed between states. Once the requisite amount of time has passed in a certain state, we switch states and reset the count variable to zero. The code for the reset button and walk button functionalities can be found here as well. The reset button immediately sets the state to main_green, the side_light to RED, and the main_light to GREEN, along with setting the

count variable to 0. The walk button being pressed sets a flag called walk_on to 1, which is then checked once the main_street's yellow light state has run for 2 seconds. If walk_on is 1, we set the state to all_red for 3 seconds, and then continue our progression with the side_light turning GREEN.

```
        side_green: begin
            side_light <= GREEN;
            main_light <= RED;
            count = count + 1;
            walk_light = 0;
            if (count == 6) begin
                if (sensor == 1) begin
                    side_target = 9;
                end
            end
            if (count == side_target) begin
                state <= side_yellow;
                count = 0;
                side_target = 6;
            end
        end
        side_yellow: begin
            side_light <= YELLOW;
            main_light <= RED;
            count = count + 1;
            if (count == 2) begin
                state <= main_green;
                count = 0;
            end
        end
        all_red: begin
            side_light <= RED;
            main_light <= RED;
            count = count + 1;
            walk_on = 0;
            walk_light = 1;
            if (count == 3) begin
                state <= side_green;
                count = 0;
            end
        end
    endcase
```

## Testing:

```
module testbench();
    reg clk;
    always #500 clk = ~clk;
    reg walk_button = 0;
    reg sensor = 0;
    reg rst = 0;
    wire [1:0] main_light;
    wire [1:0] side_light;
    wire walk_light;

    traffic_controller traf(.clk(clk), .walk_button(walk_button), .sensor(sensor),
    .rst(rst), .main_light(main_light), .side_light(side_light), .walk_light(walk_light));

    initial begin
        clk = 1;
        rst = 0;
        #6000;
        rst = 1;        //check reset during main_green
        #1000;
        rst = 0;        //remove reset
        #5000;
        walk_button = 1;    //check walk button
        #1000;
        walk_button = 0;    //turn off walkbutton
        #6000;
        sensor = 1;         //check sensor
        #15000;
        sensor = 0;         //turn off sensor
        walk_button = 1;    //turn on walk button
        #1000;
        walk_button = 0;    //turn walk button off
        #17000;
        rst = 1;            //reset
        #1000;
        rst = 0;
        #5000;
        sensor = 1;
        #12000;
        sensor = 0;
        #1000;
        walk_button = 1;        //check walk during side street green
        #1000;
        walk_button = 0;
    end
```

Our Testbench Implementation:

At 6 seconds, we assert a reset. This resets the state to main_green with a count of 0 seconds at 6 seconds, which is why we see the initial green light run for 18 seconds instead of the normal 12.

We then wait 5 seconds and then push the walk button, which sets our walk_on flag to 1 and changes nothing else immediately. We see at 20 seconds that both the main_light and side_light are RED for 3 seconds. This is our all_red state, and we can see our walk_light is illuminated for those same 3 seconds.

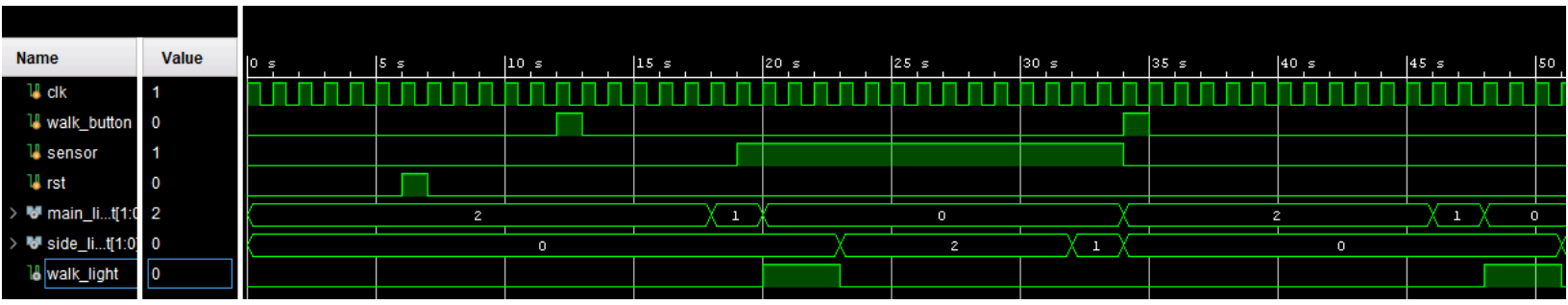
From 19 to 34 seconds, our sensor is activated, meaning high levels of traffic on the side street. Because of this, we can see our side_light is GREEN for 9 seconds instead of the normal 6. The main_light after this (34s-46s) runs for the normal 12, and that is because the sensor reads 0 at 6 seconds into the main_light's green state.

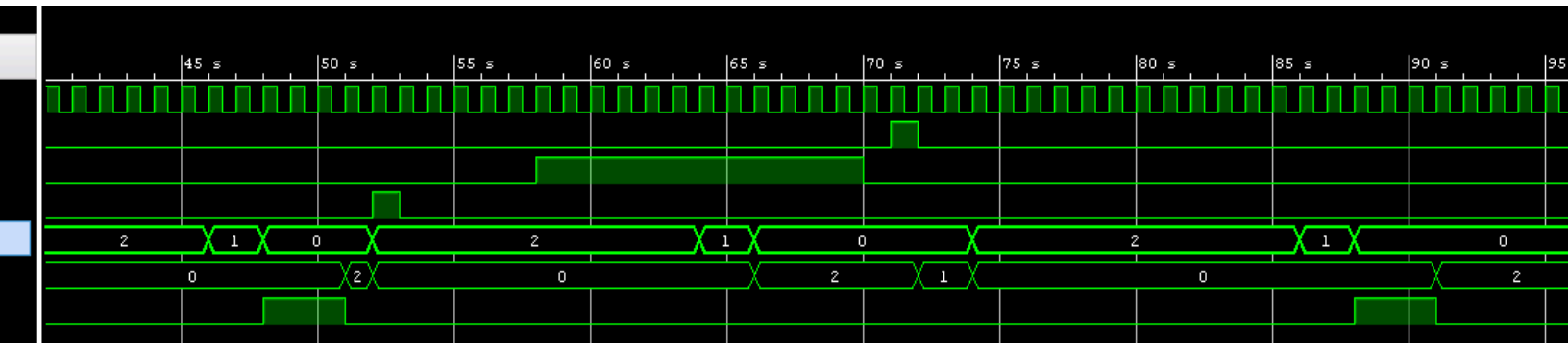
We assert another walk_button at 34 seconds, and we see both lights turn RED from 48-51 seconds. We assert a reset at 52 seconds, and this means our state is immediately set to main_green, and progresses as expected from there.

We assert the sensor signal from 58-70s, which falls exactly right after the 6 second mark for main_light, and right before the 6 second mark for side_light. This results in both lights remaining green for the normal 12 and 6 seconds respectively, as the sensor must be at 1 at the 6 second mark, and in neither case was that true.

We finally push the walk_button again at 71s, and we see the walk_light turn on and both lights turn RED from 88-91 seconds, waiting until after the main_light finishes its yellow state.

All of these test cases can be seen in the waveform screenshots below.



## Problems Faced:

Our greatest problem with this lab was our clock signal. For the first portion of our lab, our clock was running such that our time units were 5 nanoseconds as opposed to the expected 1 second frequency. This meant that instead of our main street light being green for 12 seconds, it was green for 60 nanoseconds. This was a simple fix that took us longer to find than expected. We fixed this issue by simply adjusting our timescale in our test bench and extending the amount of time to simulate in our simulation settings.

We additionally had trouble with the reset signal, as the first iteration of our code had the reset only being asserted in the following clock cycle instead of immediately. We had to add another always block and change the behavior of our lights immediately given a reset signal, and that took some time to work out syntactically. Apart from these minor issues, there were no blockers for us in the process of completing this lab.