CEIS 114
Final Project Deliverables PowerPoint

Name: Santiago Donohue

Professor: Mostafa Mortezaie

Session: CEIS114 – Week 8

Date: 06/28/2025

Final Project Deliverable for CEIS114

This presentation is a collection of all my final deliverables for the DeVry CEIS114 - Introduction to Digital Devices course. The last few slides are the challenges, take-aways and conclusiton.

At the end you'll find my published portfolio on my website.

CEIS 114
Module 2

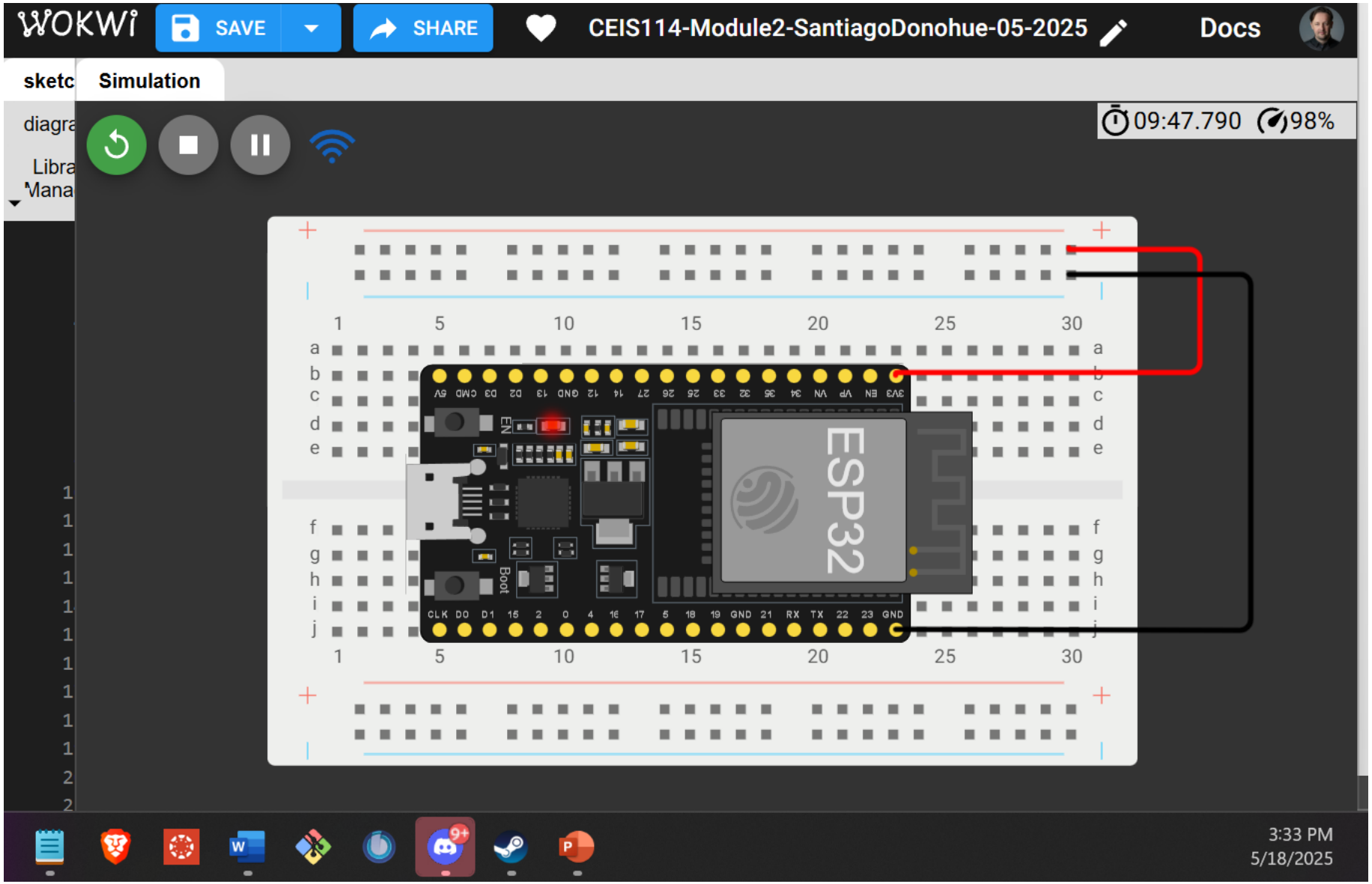
Project Plan for IoT Traffic Controller

Student: Santiago (James) Donohue

05/2025

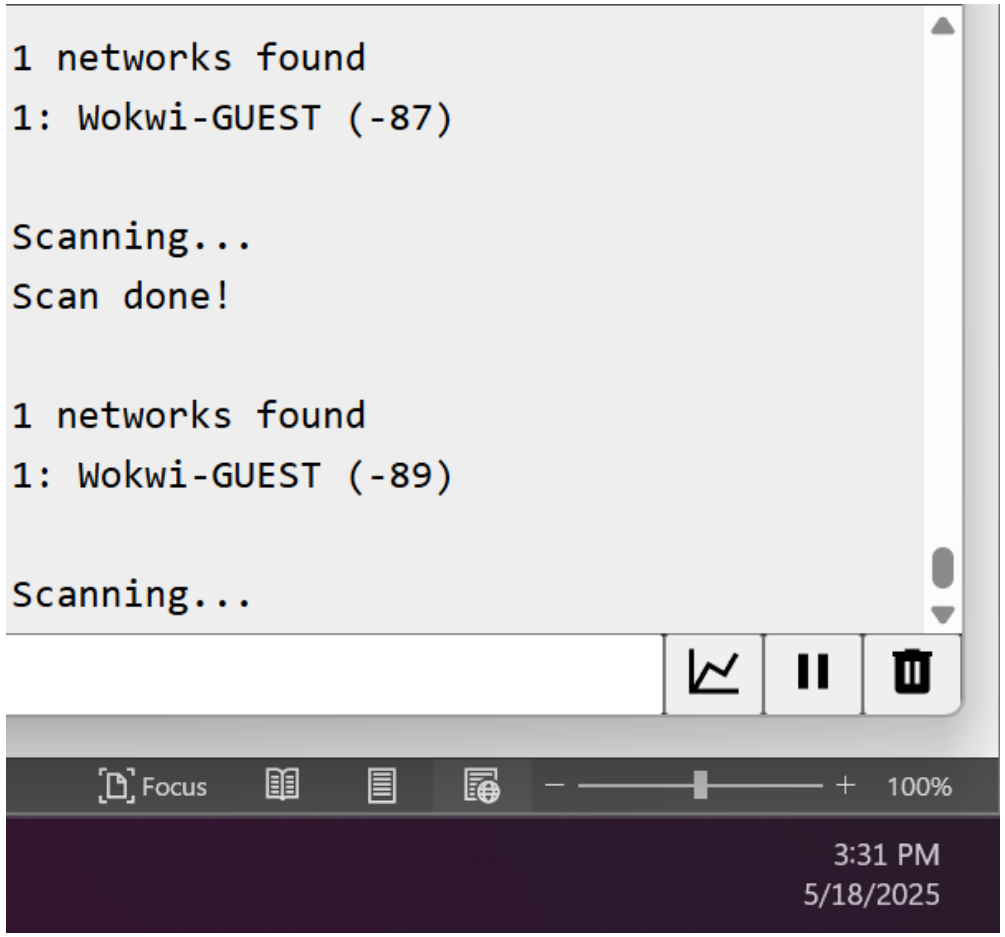
ESP32 (Screenshot)

Microcontroller mounted and powered ON



ESP32 WiFi Scan

Screenshot of **Serial Monitor** showing the available networks



CEIS 114
Module 3

Creating the Traffic Controller

Student: Santiago (James) Donohue

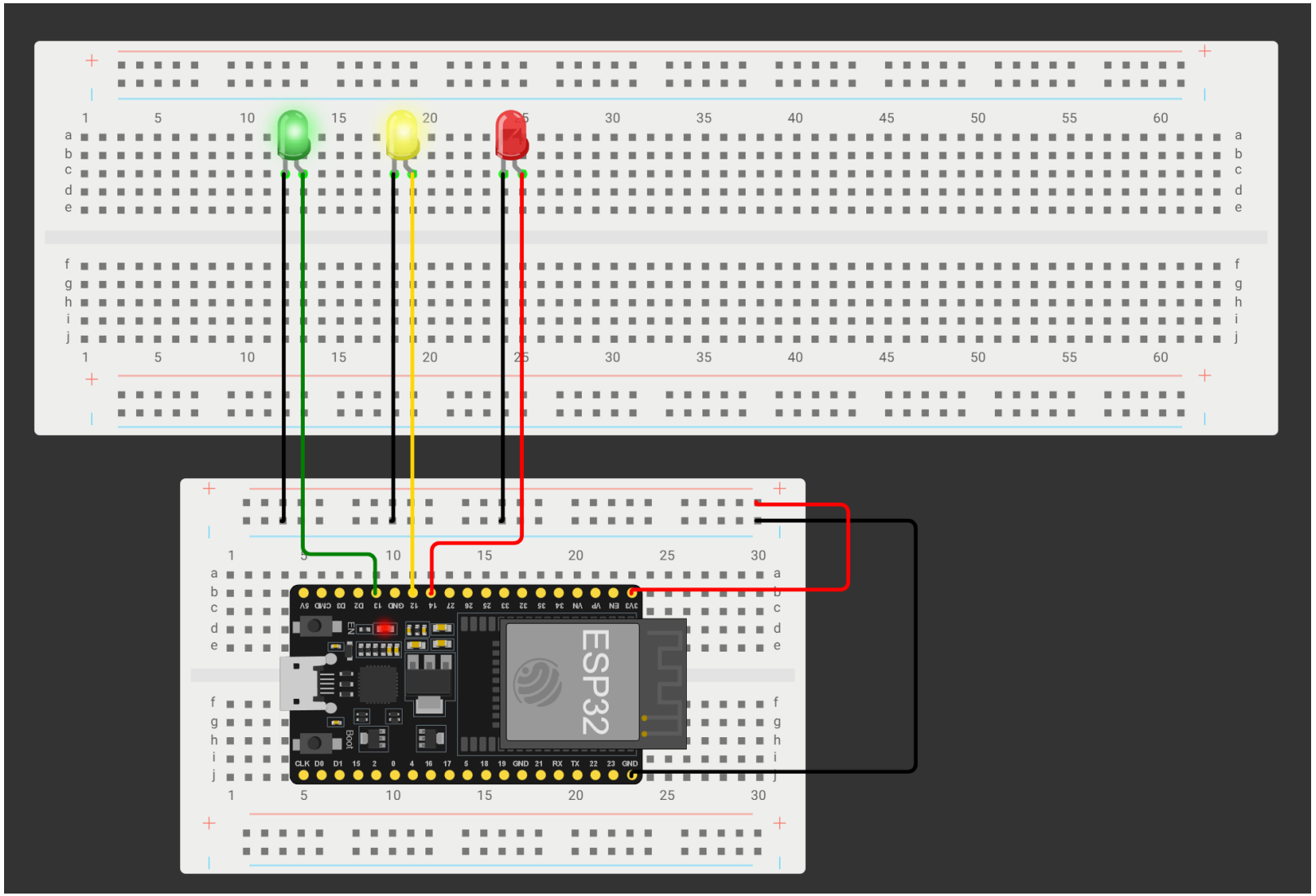
Picture of circuit with working LEDs

ESP 32 Board

Colored LEDs: Red, Yellow and Green

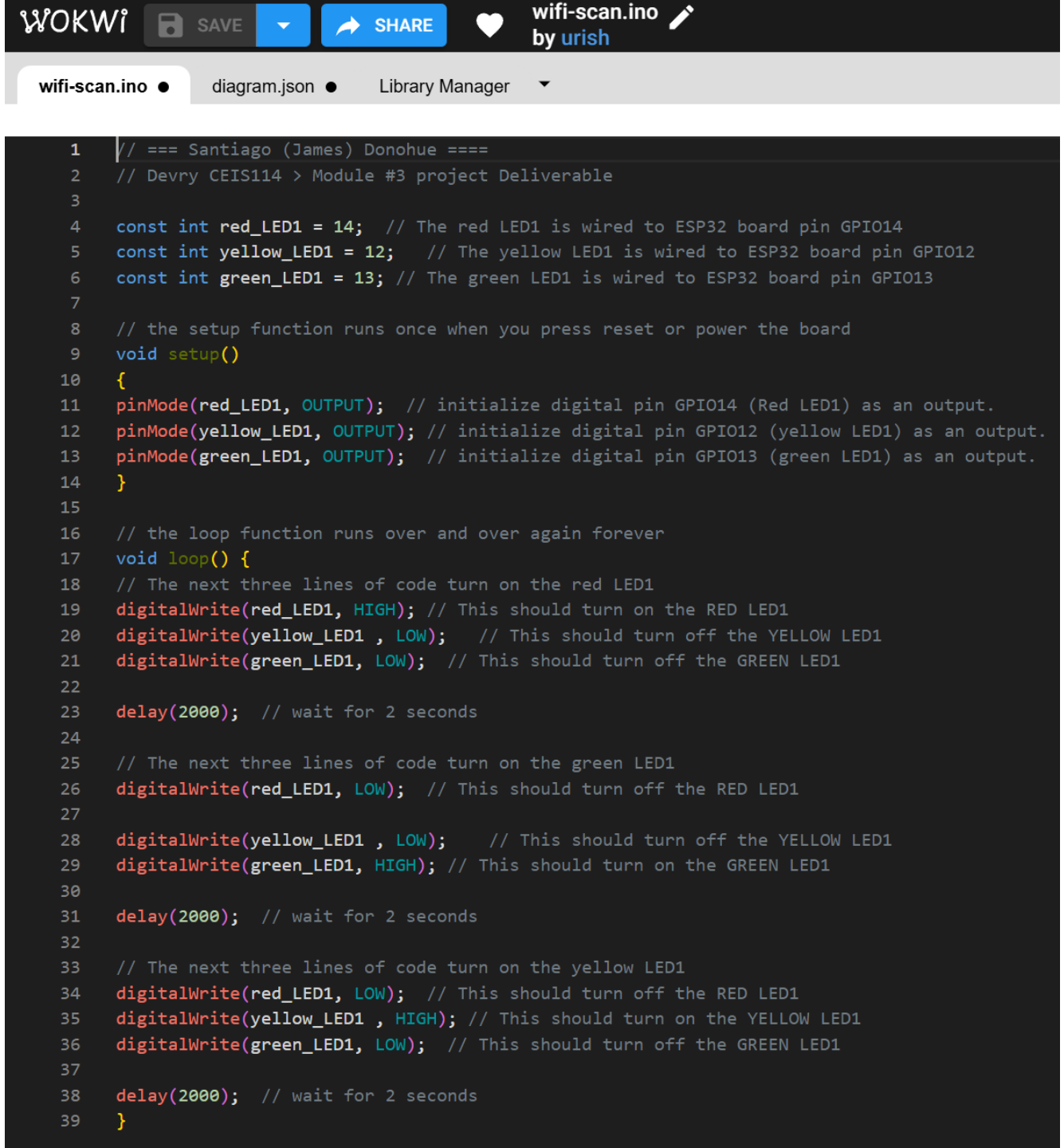
Wires

Breadboard



Screenshot of code in the Code Editor

Screenshot of code in the Wokwi Code Editor showing **your name in the comment**



CEIS 114
Module 4

Creating a Multiple Traffic Light Controller

Student: Santiago (James) Donohue

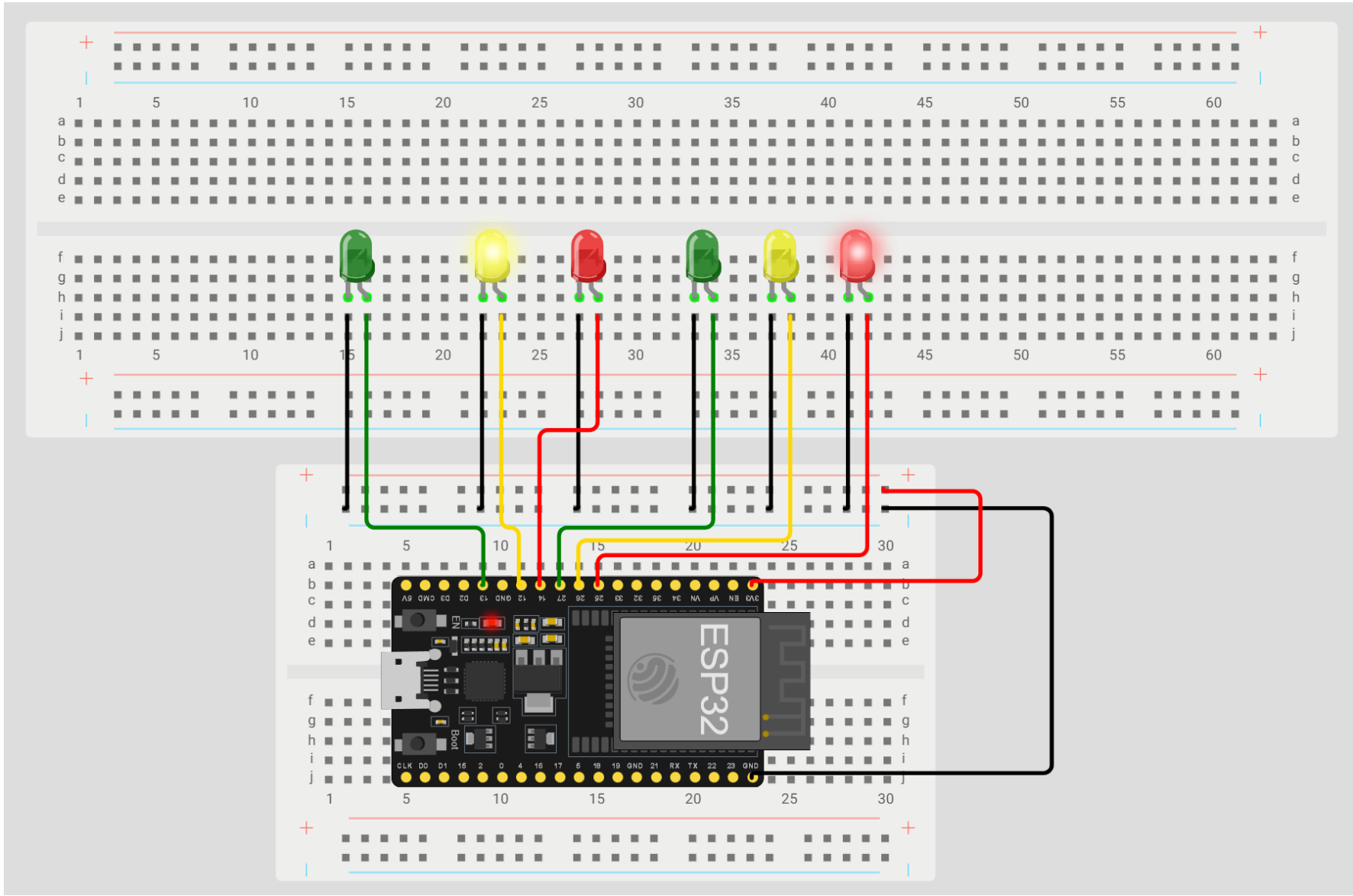
Picture of circuit with working LEDs

ESP 32 Board

Colored LEDs: Red, Yellow and Green (two sets)

Wires

Breadboard



Screenshot of code in Wokwi

Screenshot of code Wokwi Code Editor showing **your name in the comment**

CEIS 114
Module 5

Creating a Multiple Traffic Light Controller with a Cross Walk

Student: Santiago (James) Donohue

Screenshot of circuit with working LEDs

ESP 32 Board

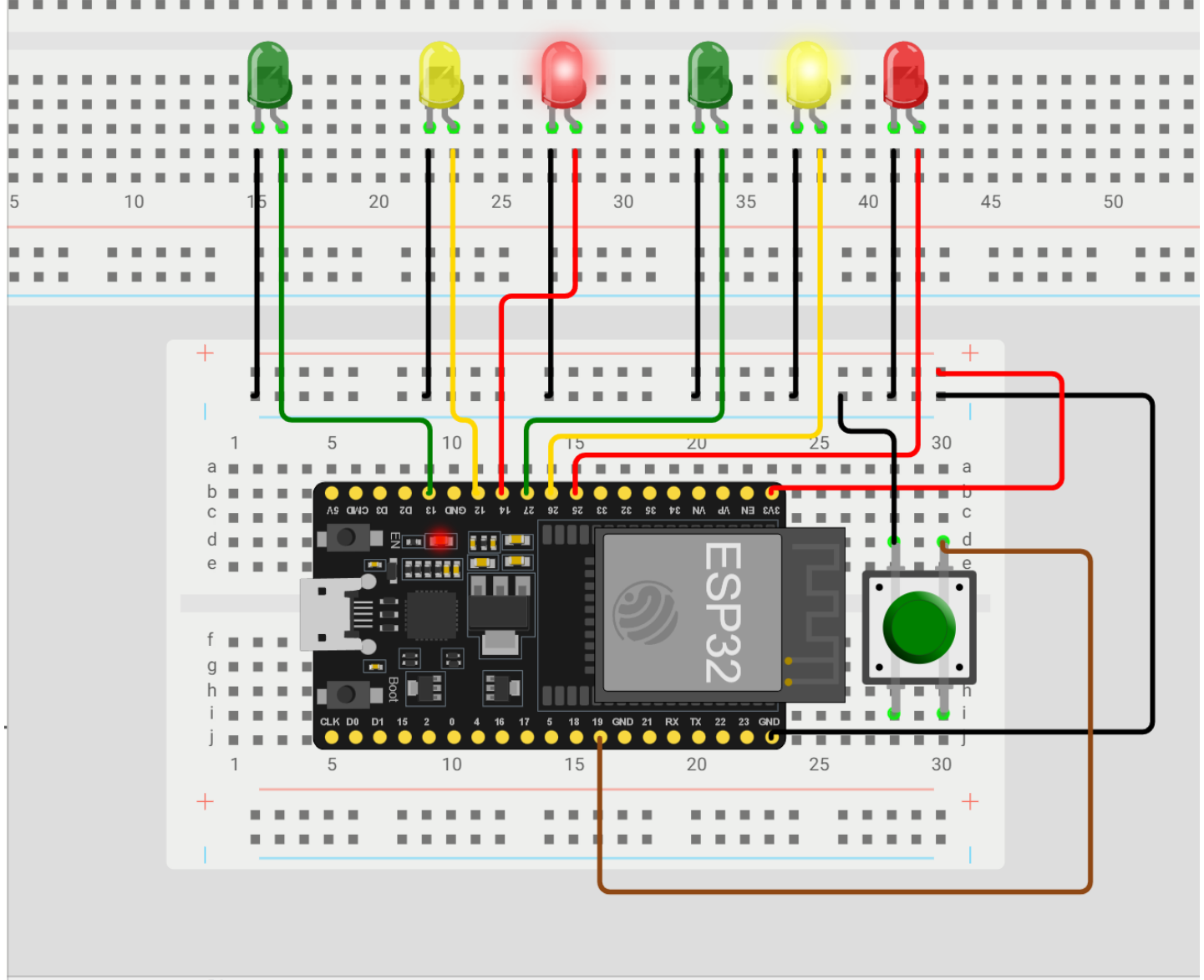
Colored LEDs: Red, Yellow and Green (two sets)

220 Ohm Resistors (optional)

Push Button

Wires

Breadboard




```
Count = 5 == Walk ==
Count = 4 == Walk ==
Count = 3 == Walk ==
Count = 2 == Walk ==
Count = 1 == Walk ==
== Do Not Walk ==
== Do Not Walk ==
```

Screenshot of code in Wokwi

Screenshot of code in Wokwi Code Editor showing **your name in the comment**


```
// Module #5 project - Santiago (James) Donohue

const int red_LED1 = 14;  // The red LED1 is wired to ESP32 board pin GPIO14
const int yellow_LED1 =12;  // The yellow LED1 is wired to ESP32 board pin GPIO12

const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
const int red_LED2 = 25;  // The red LED2 is wired to Mega board pin GPIO25
const int yellow_LED2 = 26; // The yellow LED2 is wired to Mega board pin GPIO 26
const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27

int Xw_value;

const int Xw_button = 19; //Cross Walk button

// the setup function runs once when you press reset or power the board
void setup() {

pinMode(Xw_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
Serial.begin(115200);
pinMode(red_LED1, OUTPUT); // initialize digital pin 14 (Red LED1) as an output.
pinMode(yellow_LED1, OUTPUT); // initialize digital pin 12 (yellow LED1) as an output.
pinMode(green_LED1, OUTPUT);  // initialize digital pin 13 (green LED1) as an output.

pinMode(red_LED2, OUTPUT); // initialize digital pin 25(Red LED2) as an output.
pinMode(yellow_LED2, OUTPUT); // initialize digital pin 26 (yellow LED2) as an output.
pinMode(green_LED2, OUTPUT); // initialize digital pin 27 (green LED2) as an output.
}

// the loop function runs over and over again forever
void loop() {

// read the cross walk button value:
Xw_value=digitalRead(Xw_button);

if (Xw_value == LOW ){ // if crosswalk button (X-button) pressed

digitalWrite(yellow_LED1 , LOW);  // This should turn off the YELLOW LED1
digitalWrite(green_LED1, LOW);  // This should turn off the GREEN LED1
digitalWrite(yellow_LED2 , LOW); // This should turn off the YELLOW LED2
digitalWrite(green_LED2, LOW);  // This should turn off the GREEN LED2

for (int i=10; i>0; i--)

{

```

Screenshot of Serial Monitor in Wokwi

Screenshot of output in Serial Monitor

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 Count = 10 == Walk ==
 Count = 9 == Walk ==
 Count = 8 == Walk ==
 Count = 7 == Walk ==
 Count = 6 == Walk ==
 Count = 5 == Walk ==
 Count = 4 == Walk ==
 Count = 3 == Walk ==
 Count = 2 == Walk ==
 Count = 1 == Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
```

CEIS 114
Module 6

Creating a Multiple Traffic Light Controller with a Cross Walk and an Emergency Buzzer

Student: Santiago (James) Donohue

Picture of circuit with working LEDs and LCD display

ESP 32 Board

Colored LEDs: Red, Yellow and Green (two sets)

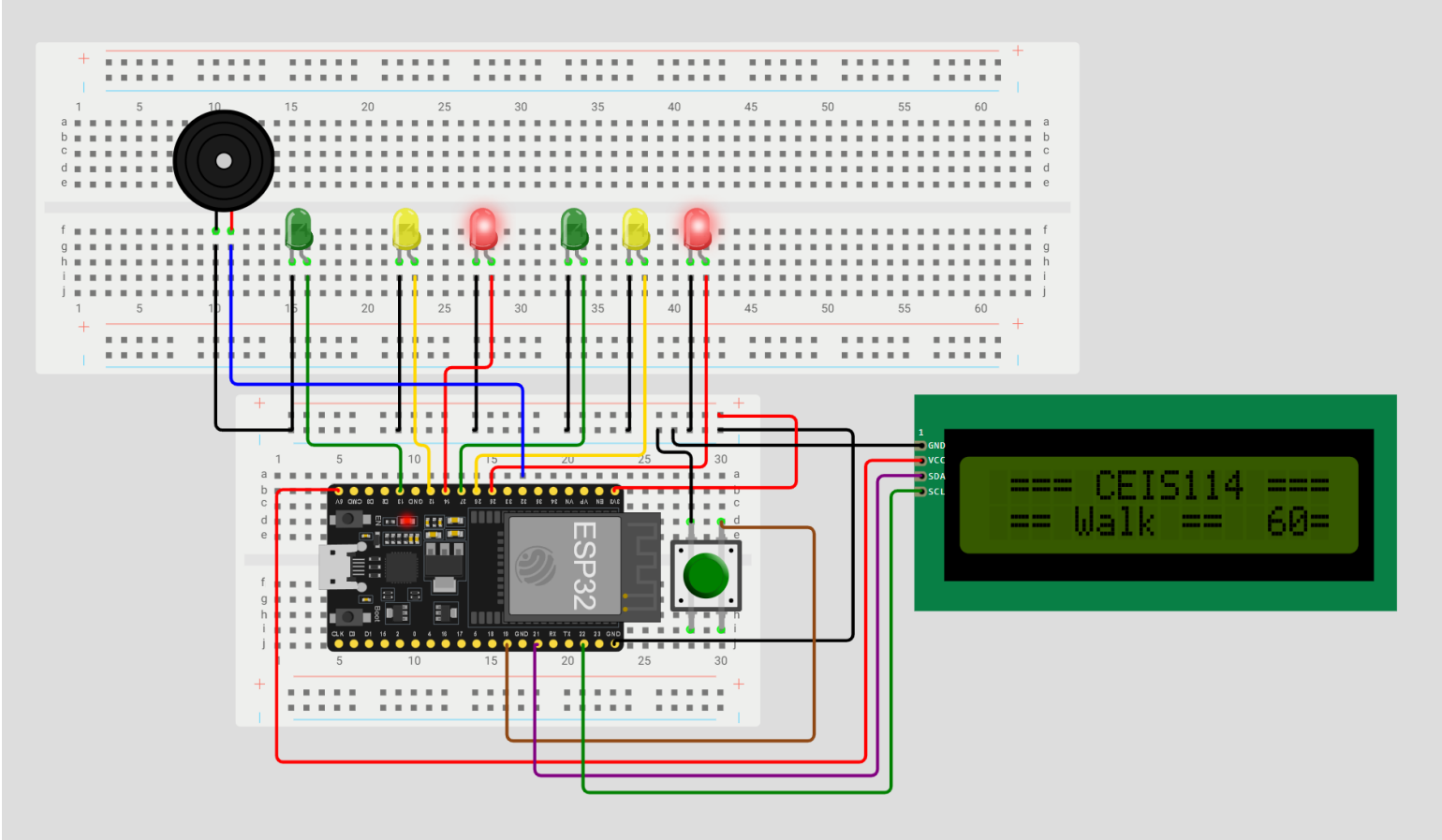
220 Ohm Resistors (optional)

Push Button

LCD Unit with Message Display

Wires

Breadboard



Screenshot of code in Code Editor

Screenshot of code in Code Editor showing **your name in the comment**


```
// === Santiago (James) Donohue ====
// Module #6 project #include <Wire.h> //lcd
#include <LiquidCrystal_I2C.h> //lcd

LiquidCrystal_I2C lcd(0x27,16,2); //set the LCD address to 0x3F for a 16 chars and 2-line display
// if it does not work then try 0x3F, if both addresses do not work then run the scan code below

const int bzr=32; // GPIO32 to connect the Buzzer
//=================== LCD ====================
const int red_LED1 = 14;  // The red LED1 is wired to ESP32 board pin GPIO14
const int yellow_LED1 =12;  // The yellow LED1 is wired to ESP32 board pin GPIO12
const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
const int red_LED2 = 25;  // The red LED2 is wired to Mega board pin GPIO25
const int yellow_LED2 = 26; // The yellow LED2 is wired to Mega board pin GPIO 26
const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27

int Xw_value;
const int Xw_button = 19; //Cross Walk button

void setup()
{

Serial.begin(115200);
pinMode(Xw_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button

lcd.init(); // initialize the lcd lcd.backlight();
lcd.setCursor(0,0); // column#4 and Row #1
lcd.print(" === CEIS114 ===");
pinMode(bzr,OUTPUT);

pinMode(red_LED1, OUTPUT); // initialize digital pin 14 (Red LED1) as an output.
pinMode(yellow_LED1, OUTPUT); // initialize digital pin12 (yellow LED1) as an output.
pinMode(green_LED1, OUTPUT);  // initialize digital pin 13 (green LED1) as an output.

pinMode(red_LED2, OUTPUT); // initialize digital pin 25(Red LED2) as an output.
pinMode(yellow_LED2, OUTPUT); // initialize digital pin 26 (yellow LED2) as an output.
pinMode(green_LED2, OUTPUT); // initialize digital pin 27 (green LED2) as an output.

}

// the loop function runs over and over again forever
void loop()
{

// read the cross walk button value:
Xw_value=digitalRead(Xw_button);

if (Xw_value == LOW ){ // if crosswalk button (X-button) pressed
digitalWrite(yellow_LED1 , LOW);  // This should turn off the YELLOW LED1
digitalWrite(green_LED1, LOW);  // This should turn off the GREEN LED1
digitalWrite(yellow_LED2 , LOW); // This should turn off the YELLOW LED2
digitalWrite(green_LED2, LOW); // This should turn off the GREEN LED2

```

Screenshot of Serial Monitor

Screenshot of output in Serial Monitor

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
 == Do Not Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
 Count = 10 == Walk ==
 Count = 9 == Walk ==
 Count = 8 == Walk ==
 Count = 7 == Walk ==
 Count = 6 == Walk ==
 Count = 5 == Walk ==
 Count = 4 == Walk ==
 Count = 3 == Walk ==
 Count = 2 == Walk ==
 Count = 1 == Walk ==
 Count = 0 == Walk ==
 == Do Not Walk ==
 == Do Not Walk ==
```

CEIS 114
Week 7 Project

Creating a Multiple Traffic Light Controller with a Cross Walk and an Emergency Buzzer with secured IoT Control via Web

Student: Santiago (James) Donohue

Screenshot of circuit with working LEDs and LCD display (Building/Operation)

ESP 32 Board

Colored LEDs: Red, Yellow and Green (two sets)

One Blue LED – Emergency Light

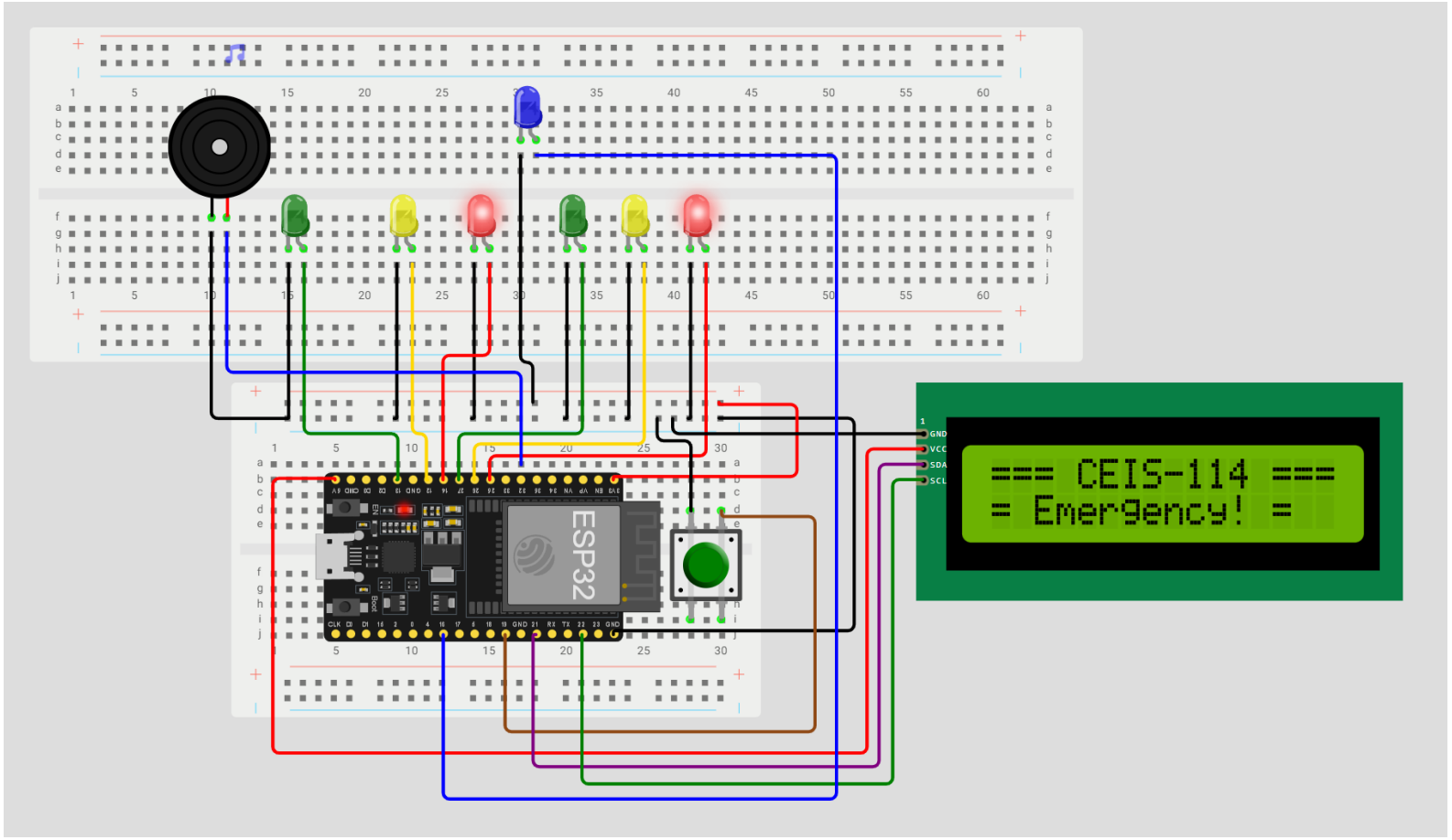
Push Button

LCD Unit

Buzzer

Wires

Breadboard



Screenshot of code in Code Editor (Testing)

Screenshot of code in Code Editor showing **your name in the comment**

WOKWI SAVE SHARE CEIS114-Module07-SantiagoDonohue-2025

wifi-scan.ino ● diagram.json ● libraries.txt ● Library Manager

```
// === Santiago (James) Donohue ====
// Final Project Component, Option 1

#include <WiFi.h> // WiFi header file
#include <PubSubClient.h> // MQTT publish and subscribe header file
#include <Wire.h> // I2C header file
#include <LiquidCrystal_I2C.h> // I2C lcd header file

const char* ssid = "Wokwi-GUEST"; // This is the access point to your wireless network.
const char* password = ""; // This is the password to the SSID. For the smart mini router
const char* mqttServer = "test.mosquitto.org"; // This is the free MQTT broker we will use.

int port = 1883; // MQTT brokers listen to port 1883 by default
String stMac; // C string used for convenience of comparisons.
char mac[50]; // C char array used to hold the MAC address of your ESP32 microconroller
char clientId[50]; // This client ID is used to identify the user accessing the MQTT broker.

// For our test.mosquitto.org broker, we just generate a random user client ID
WiFiClient espClient; // instantiate the WiFi client object
PubSubClient client(espClient); // instantiate the publish subscribe client object
LiquidCrystal_I2C lcd(0x27,16,2); //set the LCD address to 0x27 for a 16 chars and 2-line display
// if it does not work then try 0x3F, if both addresses do not work then run the scan code

const int redLightNorthSouth = 14; // The red LED NS is wired to ESP32 board pin GPIO 14
const int yellowLightNorthSouth = 12; // The yellow LED NS is wired to ESP32 board pin GPIO 12
const int greenLightNorthSouth = 13; // The green LED NS is wired to ESP32 board pin GPIO 13
const int redLightEastWest = 25; // The red LED EW is wired to ESP32 pin GPIO 25
const int yellowLightEastWest = 26; // The yellow LED EW is wired to ESP32 board pin GPIO 26
const int greenLightEastWest = 27; // The green LED EW is wired to ESP32 board pin GPIO 27

int crossWalkButtonState = 1 ; // Variable will store the state of the crosswalk button
const int crossWalkButton = 19; // Cross Walk button pin is GPIO 19
const int emergencyBlueLED = 16; // The blue LED is wired to ESP32 board pin GPIO 16
const int buzzerPin = 32; // Active Buzzer pin is GPIO 32

int loopCount; // Variable will keep count of the number of times the light pattern repeats
int secondsLeft; // counter to keep track of number of seconds left for crossing intersection
int iotControl = 0; // Variable will be used to switch between emergency and normal operations of

// traffic controller
void setup() {

Serial.begin(115200); // set baud rate of serial monitor to 115200 bits per second
randomSeed(analogRead(0)); // seed the random() function
delay(10); // wait 10 milliseconds

```

Screenshot of Serial Monitor (Testing)

Screenshot of output in Serial Monitor

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc

Connecting to Wokwi-GUEST
..
WiFi connected
IP address:
10.10.0.2
24:0A:C4:00:01:10
24_0A_C4_00_01_10
Attempting MQTT connection...clientId-260 connected
 == Do Not Walk ==
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
= Emergency! =
 == Do Not Walk ==
 == Do Not Walk ==
```

Challenges/Lessons Learned

Challenges:

Below I'll list some of the challenged that stood out to me while working on project modules throughout this course.

- I was surprised to learn that not all data processing happens on the device. I thought everything was handled by the sensor itself, but realizing that data often gets processed elsewhere made things feel more complicated.
- Getting different devices to work together was harder than I expected. It took time to understand how sensors, actuators, and communication systems all fit into one working system.
- Working with real-time data was also challenging. I had trouble making sure the system reacted fast enough, and it wasn't always clear whether issues were from the code or the hardware.

Lessons Learned:

- I came to understand that offloading data processing to the cloud or edge systems isn't just a workaround—it's a smart way to reduce the load on devices and improve performance. It's actually a core part of designing efficient IoT systems.
- Signal conditioning stood out as something I underestimated at first. Clean, accurate signals are essential if you want reliable results from your sensors, and that step can make or break how well your system works.
- Breaking the system into input, processing, and output made everything easier to troubleshoot and scale. It helped me think more like a systems designer instead of just wiring things together.

Career Skills

1. Systems thinking

I learned how to break down complex systems into parts like sensors and processors. This helps in jobs where you design or manage technology projects with many connected components.

2. Working with data

I practiced collecting and cleaning sensor data to make it useful. This skill is important for analyzing information to improve systems at work.

3. Problem-solving with devices

I gained experience troubleshooting hardware and software issues. This helps in jobs that require fixing technical problems and keeping systems running.

4. Explaining technical work

I improved how I explain my projects and solutions clearly. This is useful for working with teams and communicating technical ideas to others.

CONCLUTION

The conclusion of my project, I was able to build and program an IoT device that could improve the security of my home or work location.

This course helped me develop a much clearer understanding of how digital devices and IoT systems connect and work together, especially around data processing and device integration. Working through the hands-on projects gave me practical experience troubleshooting issues and showed me why clear communication about technical problems is so important.

Overall, I feel more prepared to handle real-world challenges in technology roles, particularly those involving embedded systems and IoT.

Personal website portfolio link referencing this course's final project

https://santiagodonohue.com/