



CEIS114 Final Course Project Traffic Controller Hector Acosta

Introduction

- Traffic management is a critical aspect of urban planning and development, impacting everything from commuter efficiency to safety and environmental concerns.
- Traditional traffic light systems are often static and inflexible, unable to adapt to dynamic traffic patterns or respond to emergencies effectively.
- With the increasing complexity of urban traffic, there is a growing need for intelligent traffic management solutions that can optimize traffic flow, enhance safety, and reduce congestion.



Project Overview

- The Smart Traffic Light Controller project is designed to address these challenges by creating an advanced traffic management system using modern technology.
- The project leverages an ESP32 microcontroller, LEDs, sensors, and web-based interfaces to develop a traffic light system that can adapt to various traffic conditions and respond to emergencies.
- The system aims to improve traffic flow, provide real-time monitoring, and ensure safety through innovative features such as emergency mode and remote access.



Objectives

Traffic Light Control: Develop a traffic light controller system that can manage multiple traffic signals at an intersection. The system will use colored LEDs to represent traffic lights for different directions and implement timing sequences to control the flow of traffic.

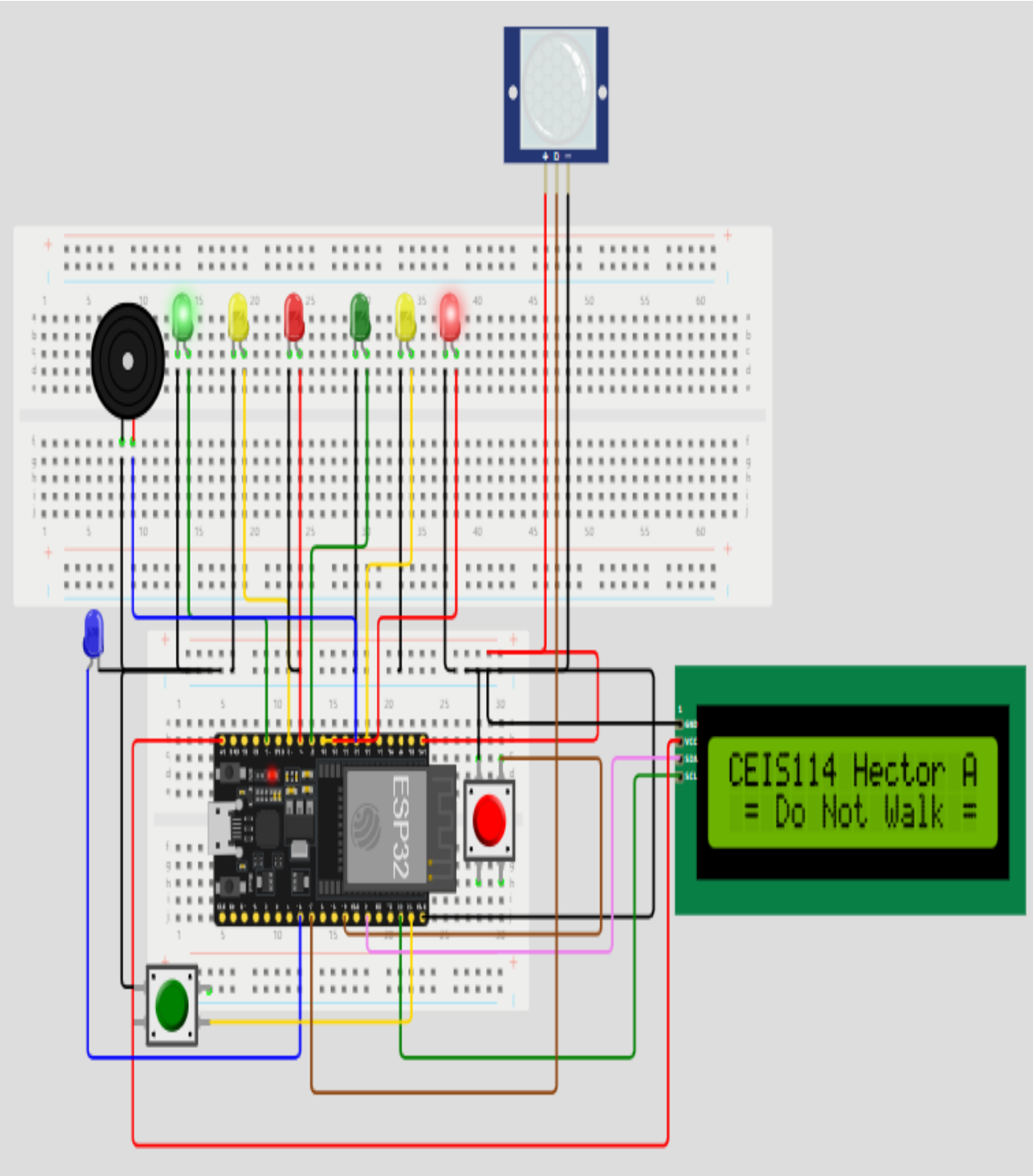
Emergency Mode: Implement an emergency mode feature that can be activated in response to emergency situations. This feature will turn all traffic lights red and activate a flashing blue signal to alert drivers, ensuring that emergency vehicles can pass through intersections safely and efficiently.

Remote Access: Create a remote access system that allows users to manage and monitor the traffic lights via a web browser. This will involve setting up a web server on the ESP32, providing a user-friendly interface to control the traffic lights and view their status from any location.

Alternative Control Mechanism: Explore an alternative control mechanism using a motion sensor. This system will enable traffic lights to operate continuously on the major street and switch to the minor street only when motion is detected, optimizing traffic flow and reducing unnecessary stops.

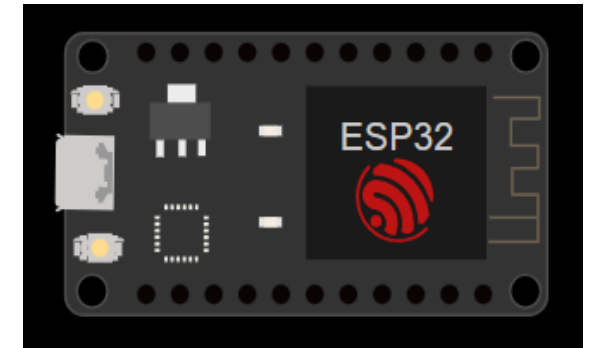
Components and Setup

- **ESP32 Microcontroller:** The central processing unit for controlling the traffic lights and handling web server functions.
- **LEDs:** Red, yellow, and green LEDs to represent traffic signals.
- **Buttons:** For manual control and testing of the system.
- **LCD Display:** To show system status and information.
- **Buzzer:** For alert signals and notifications.
- **Motion Sensor:** For implementing the alternative control mechanism.
- **Blue LED:** For emergency signaling.



Implementation

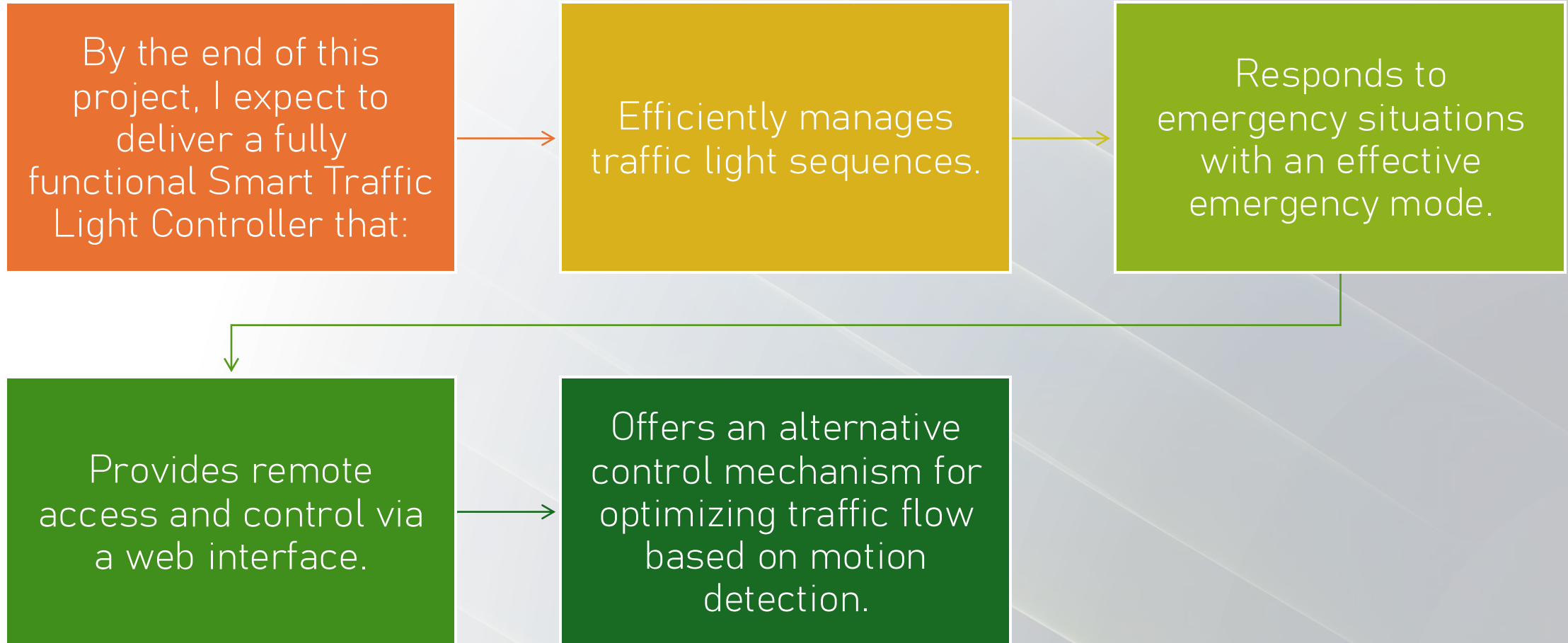
- **Design and Simulation:**
 - I use Wokwi to design and simulate the traffic light system circuit, including the integration of LEDs, buttons, and sensors.
 - Develop and test the code for controlling the traffic lights and implementing the emergency mode and remote access functionalities.
- **Web Interface:**
 - Set up a web server on the ESP32 to create a user interface for remote management.
 - This interface will allow users to control and monitor the traffic lights from a web browser.
- **Testing and Documentation:**
 - Conduct thorough testing of the traffic light system, including the emergency mode and remote access features.
 - Document the design, implementation, and testing process, providing screenshots of the circuit simulation, code simulation, and output from the Serial Monitor.



```
WOKWI SAVE SHARE Final Project Option2 Hector Acosta by oaxacarian
sketch.ino diagram.json libraries.txt Library Manager
1 // === Hector Acosta ===
2 // Final Project Component - Option 2
3 #include <Wire.h> //I2C
4 #include <LiquidCrystal_I2C.h> //LCD
5 LiquidCrystal_I2C lcd(0x27,16,2); //set the LCD address to 0x27 for a 16 by 2 display
6 const int bzz=32; // GPIO32 to connect the Buzzer
7 // Set GPIOs for LED and PIR Motion Sensor
8 const int led = 16; // Flashing White (Blue) Led
9 const int motionSensor = 17;
10 int pinState = 0 ;
11 int j,Em_value,Xw_value;
12 const int Em_button = 23; // Emergency button
13 const int Xw_button = 19; //Cross Walk button
14 //===== LCD =====
15 const int red_LED1 = 14; // The red LED1 is wired to ESP32 board pin GPIO14
16 const int yellow_LED1 =12; // The yellow LED1 is wired to ESP32 board pin GPIO12
17 const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
18 const int red_LED2 = 25; // The red LED2 is wired to Mega board pin GPIO25
19 const int yellow_LED2 = 26; // The yellow LED2 is wired to ESP32 pin GPIO 26
20 const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27
21 void setup() {
22 //=====Motion Detector initialization =====
23 // PIR Motion Sensor mode INPUT
24 pinMode(motionSensor, INPUT);
25 pinMode(Em_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
26 pinMode(Xw_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
27 pinMode(bzz,OUTPUT);
28 pinMode(led, OUTPUT);
29 digitalWrite(led, LOW); // Set Flashing White (Blue) Light to LOW
30 //=====
31 Serial.begin(115200);
32 lcd.init(); // initialize the lcd
33 lcd.backlight();
34 lcd.setCursor(0,0); // column #1 and Row #1
```

```
== Do Not Walk ==
== Do Not Walk ==
Emergency button was pressed
== 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 ==
== Do Not Walk ==
motion detected
```

Expected Outcomes



CEIS 114

Module 2

**Project Plan for IoT Traffic
Controller**

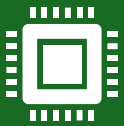
ThePhoto by PhotoAuthor is licensed under
CCYSA.



Introduction to Module 2:



In this module, I will dive into the practical setup of our Smart Traffic Light Controller system. First, I'll ensure our ESP32 is properly installed and powered on. This step is crucial for establishing the foundation of our project.



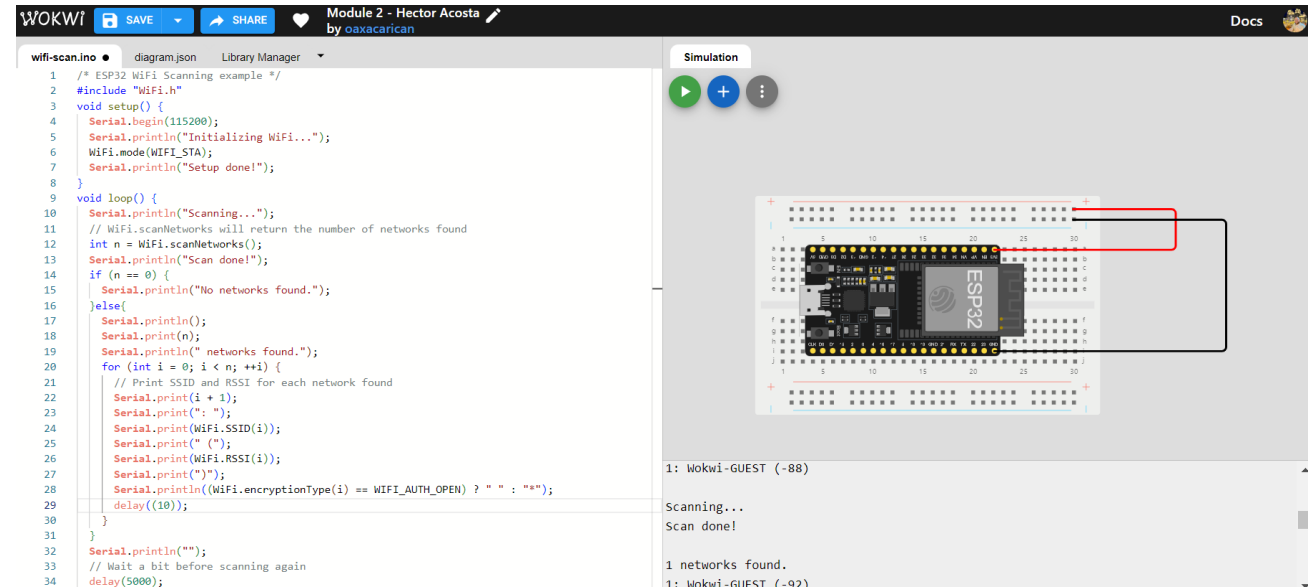
I'll start by examining the circuit setup, as depicted in the screenshot on this slide. Once the ESP32 is up and running, I'll proceed with scanning for available WiFi networks, which will be illustrated through the Serial Monitor screenshot.



Let's move forward and get our ESP32 configured for network communication. On the next slide, we'll see the detailed circuit setup, followed by the WiFi scan results.

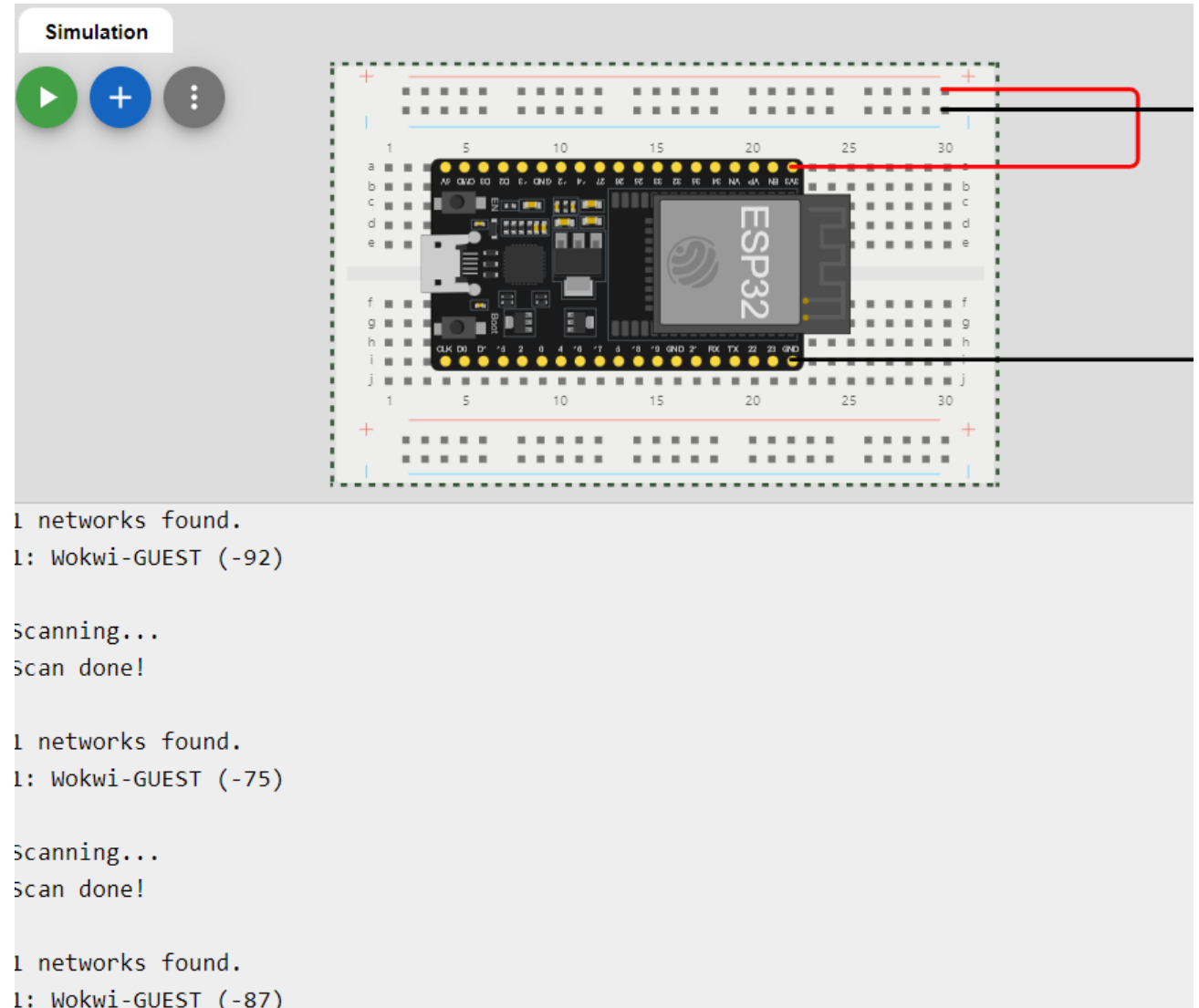
ESP32

- Microcontroller mounted and powered ON
- Here you see the screenshot of my circuit setup, showing the ESP32 installed and powered.
- This visual confirms that I have correctly connected my components and the microcontroller is operational.



ESP32 WiFi Scan

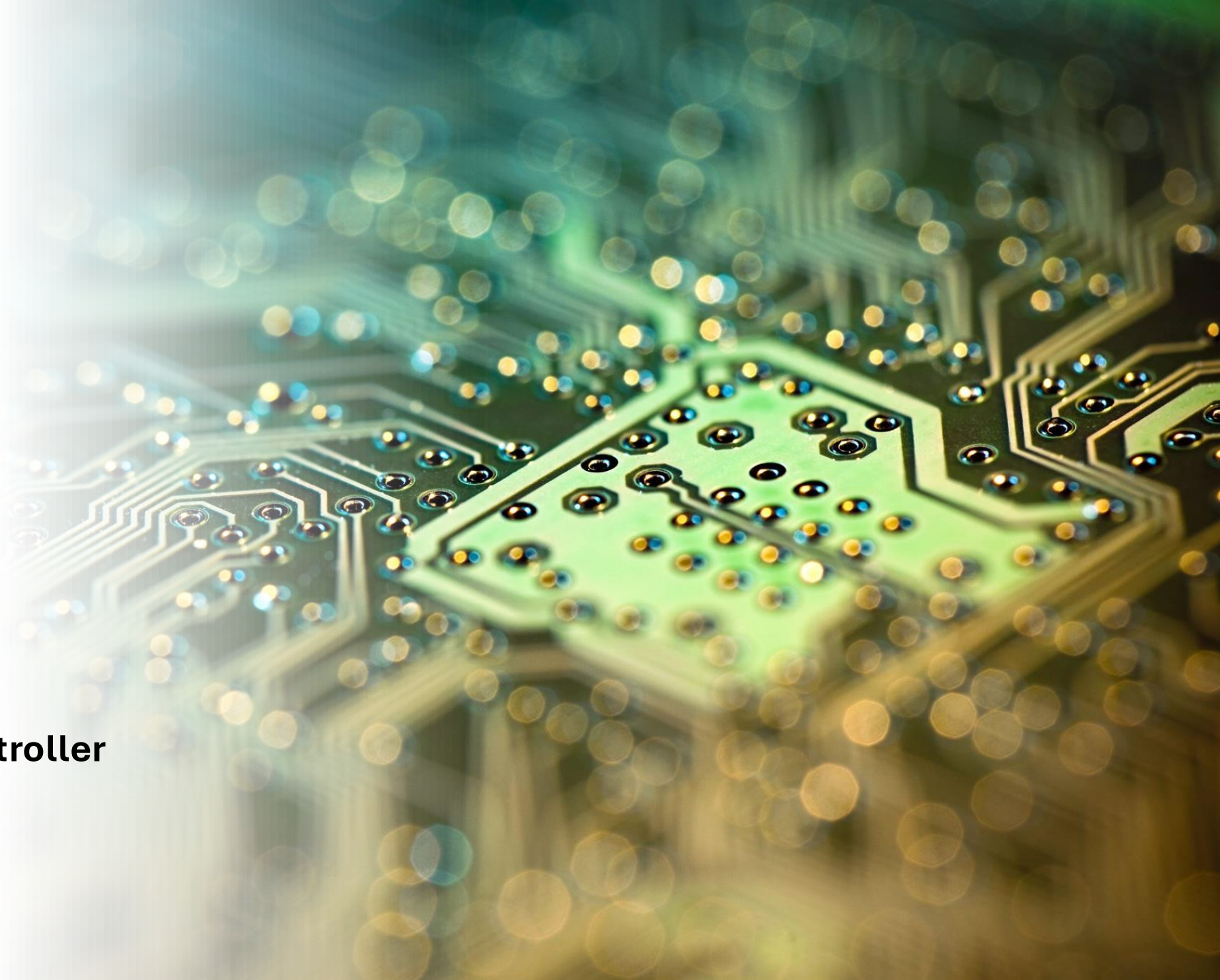
- **Serial Monitor** showing the available networks
- I perform a WiFi scan to detect available networks.
- The Serial Monitor screenshot on this slide demonstrates the scanning process, allowing me to identify and connect to my desired network for further communication.



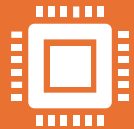
CEIS 114

Module 3

Creating the Traffic Controller



Introduction for Module 3:



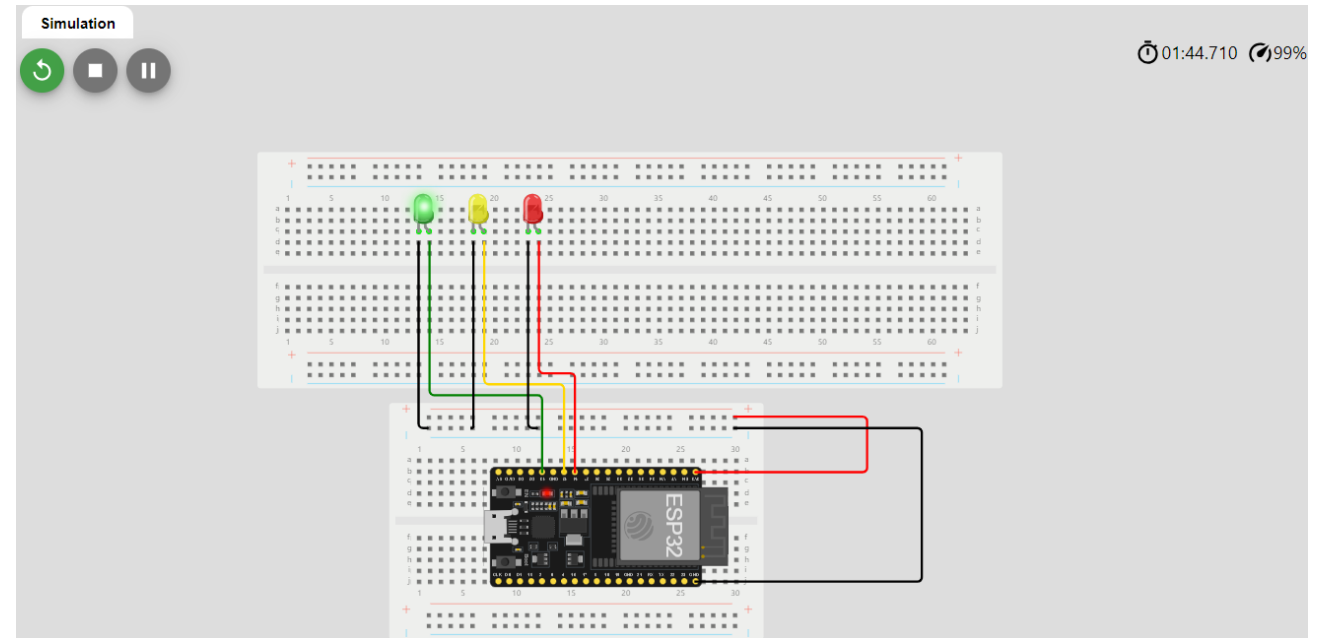
In this module, I move forward with our Smart Traffic Light Controller project by integrating LED indicators to visualize the system's operation.



I'll begin by connecting the LEDs to my circuit and ensuring they function as expected. The focus will be on demonstrating the physical setup of the LEDs and validating their performance through my breadboard configuration.

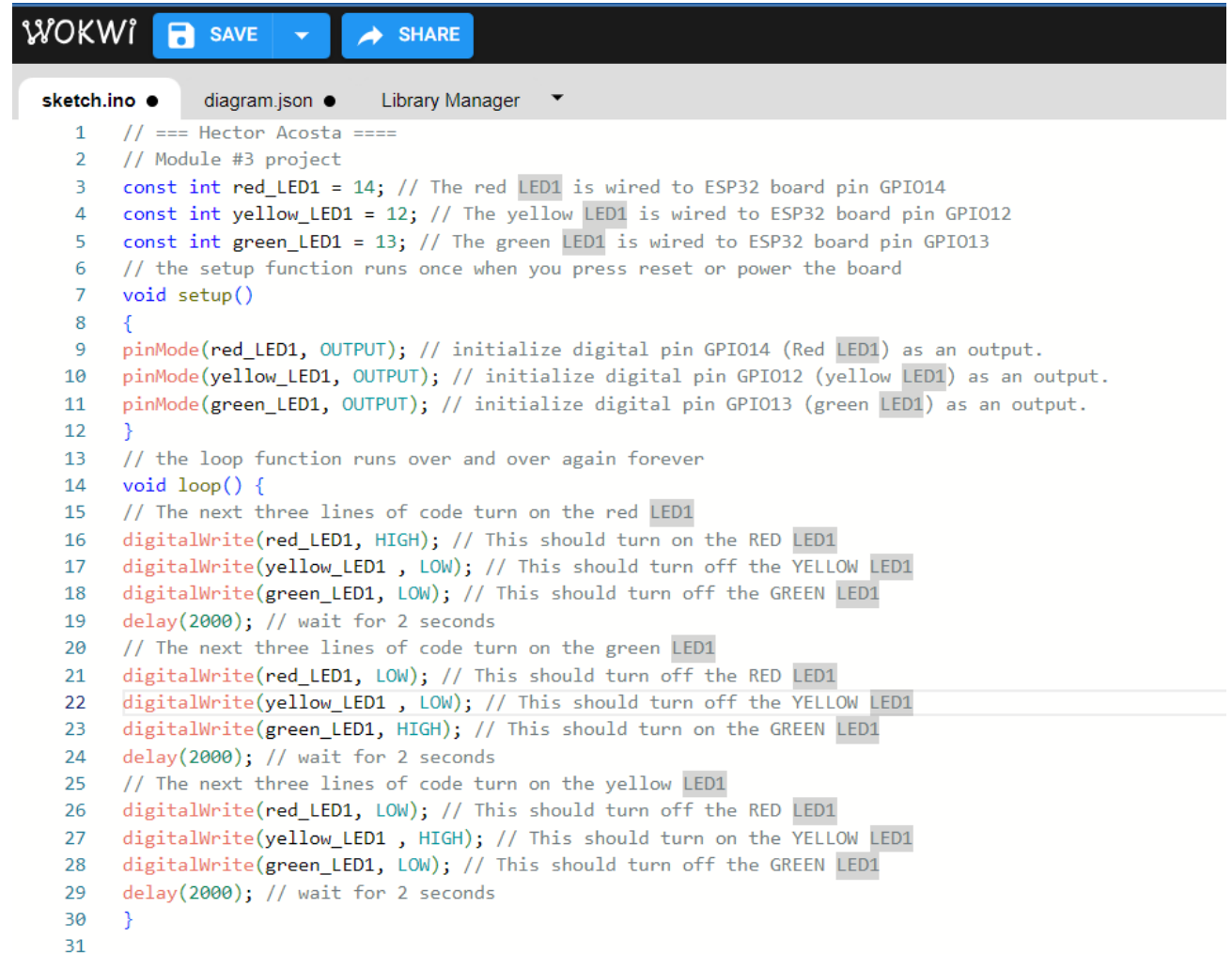
Circuit with working LEDs

- This slide showcases the updated circuit with the LEDs powered ON.
- This visual representation highlights how the LEDs are integrated into the breadboard, providing a clear view of their placement and connection.



Code Commenting

- This slide presents a screenshot of the code editor, where I include a comment with my name.
- This serves to illustrate the code I implemented for controlling the LEDs, highlighting my personal contribution to the project.

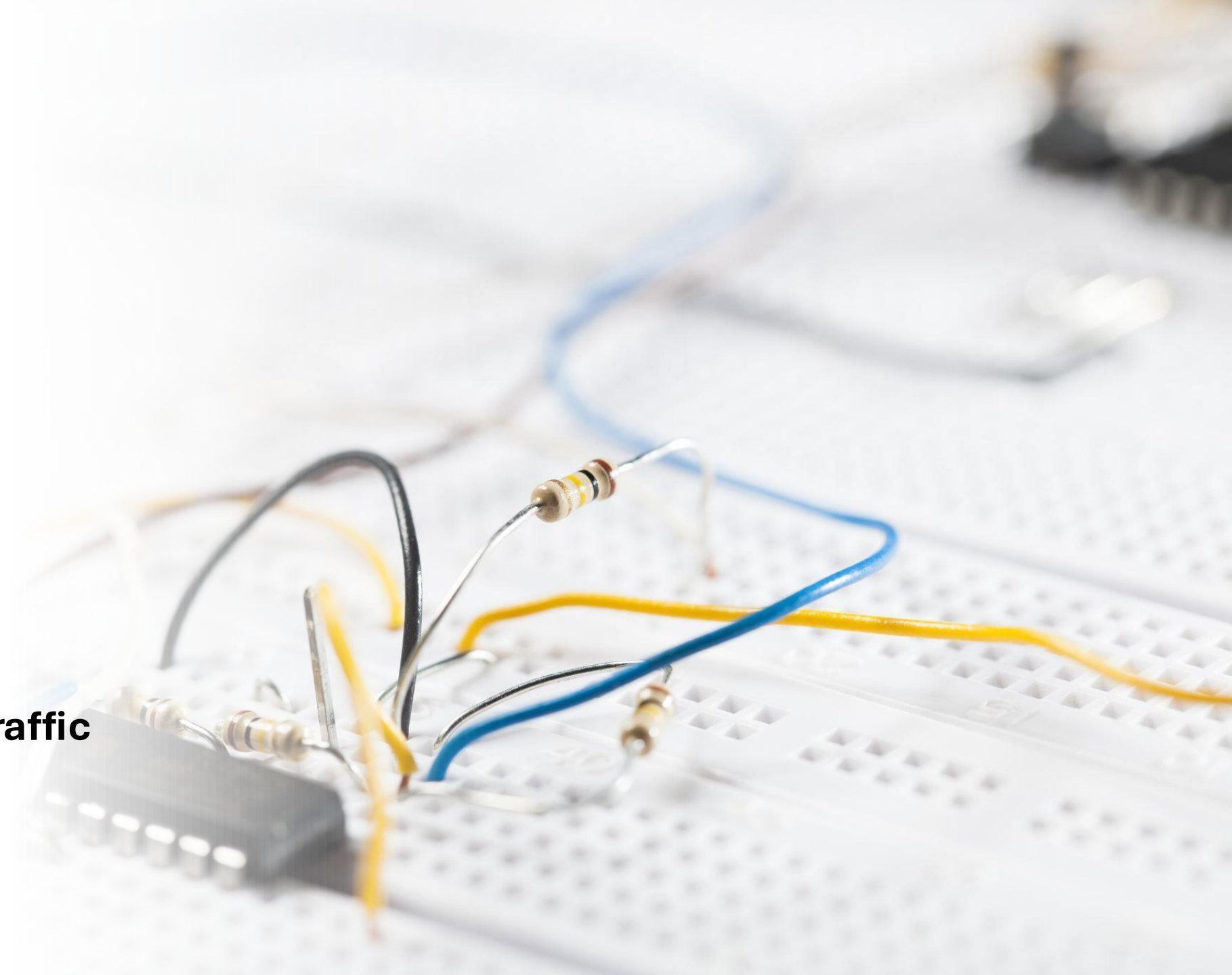


```
WOKWI SAVE SHARE
sketch.ino • diagram.json • Library Manager
1 // === Hector Acosta ===
2 // Module #3 project
3 const int red_LED1 = 14; // The red LED1 is wired to ESP32 board pin GPIO14
4 const int yellow_LED1 = 12; // The yellow LED1 is wired to ESP32 board pin GPIO12
5 const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
6 // the setup function runs once when you press reset or power the board
7 void setup()
8 {
9   pinMode(red_LED1, OUTPUT); // initialize digital pin GPIO14 (Red LED1) as an output.
10  pinMode(yellow_LED1, OUTPUT); // initialize digital pin GPIO12 (yellow LED1) as an output.
11  pinMode(green_LED1, OUTPUT); // initialize digital pin GPIO13 (green LED1) as an output.
12 }
13 // the loop function runs over and over again forever
14 void loop() {
15   // The next three lines of code turn on the red LED1
16   digitalWrite(red_LED1, HIGH); // This should turn on the RED LED1
17   digitalWrite(yellow_LED1, LOW); // This should turn off the YELLOW LED1
18   digitalWrite(green_LED1, LOW); // This should turn off the GREEN LED1
19   delay(2000); // wait for 2 seconds
20   // The next three lines of code turn on the green LED1
21   digitalWrite(red_LED1, LOW); // This should turn off the RED LED1
22   digitalWrite(yellow_LED1, LOW); // This should turn off the YELLOW LED1
23   digitalWrite(green_LED1, HIGH); // This should turn on the GREEN LED1
24   delay(2000); // wait for 2 seconds
25   // The next three lines of code turn on the yellow LED1
26   digitalWrite(red_LED1, LOW); // This should turn off the RED LED1
27   digitalWrite(yellow_LED1, HIGH); // This should turn on the YELLOW LED1
28   digitalWrite(green_LED1, LOW); // This should turn off the GREEN LED1
29   delay(2000); // wait for 2 seconds
30 }
31
```

CEIS 114

Module 4

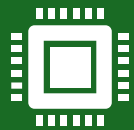
**Creating a Multiple Traffic
Light Controller**



Introduction to Module 4:



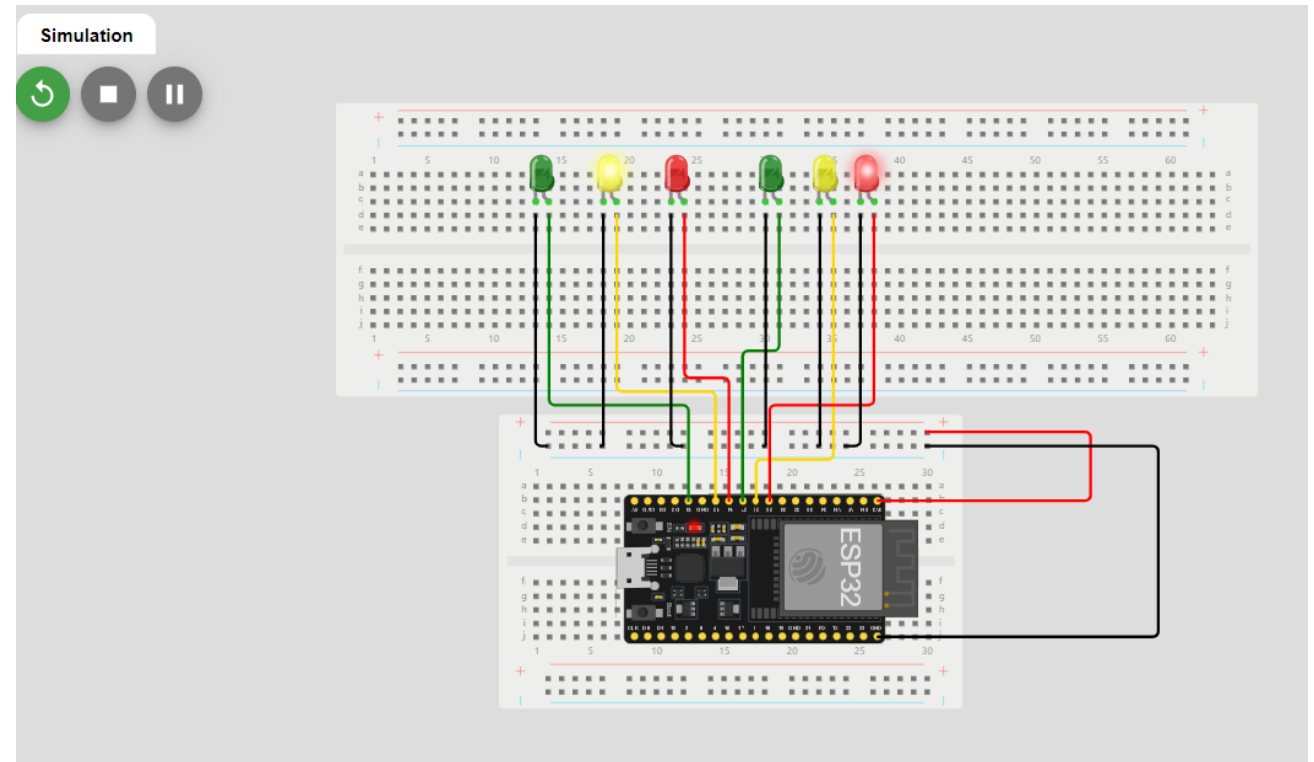
Having successfully set up our ESP32 and connected it to my WiFi network, I'm now ready to dive into Module 4. This module focuses on integrating my circuit components and programming the ESP32 to control them.



In this Module 4, I'm setting the stage for interactive and responsive control, which is a key aspect of my Smart Traffic Light Controller project. Let's explore how these elements come together to bring our system to life.


The circuit with working LEDs

- I start by examining the circuit I been assembled on the breadboard, which includes the LEDs I be controlling.
- This screenshot will showcase how the components are arranged and how the connections are made to ensure proper functionality.



Code in Wokwi

- Following the circuit overview, I'll review the code that powers our setup.
- This code snippet, displayed in the Code Editor, not only shows the logic I've implemented that includes a personal touch—my name is embedded in the comments to track contributions and provide context.



```
WOKWI SAVE SHARE Module 4 Hector Acosta by oaxacarican
sketch.ino diagram.json Library Manager
1 // === Hector Acosta ===
2 // Module #4 project
3 // Define some labels
4 const int red_LED1 = 14; // The red LED1 is wired to ESP32 board pin GPIO14
5 const int yellow_LED1 = 12; // The yellow LED1 is wired to ESP32 board pin GPIO12
6 const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
7 const int red_LED2 = 25; // The red LED2 is wired to Mega board pin GPIO25
8 const int yellow_LED2 = 26; // The yellow LED2 is wired to Mega board pin GPIO 26
9 const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27
10 // the setup function runs once when you press reset or power the board
11 void setup() {
12   pinMode(red_LED1, OUTPUT); // initialize digital pin GPIO14 (Red LED1) as an output.
13   pinMode(yellow_LED1, OUTPUT); // initialize digital pin GPIO12 (yellow LED1) as an output.
14   pinMode(green_LED1, OUTPUT); // initialize digital pin GPIO13 (green LED1) as an output.
15   pinMode(red_LED2, OUTPUT); // initialize digital pin GPIO25 (Red LED2) as an output.
16   pinMode(yellow_LED2, OUTPUT); // initialize digital pin GPIO26 (yellow LED2) as an output.
17   pinMode(green_LED2, OUTPUT); // initialize digital pin GPIO27 (green LED2) as an output.
18 }
19 // the loop function runs over and over again forever
20 void loop() {
21   // The next three lines of code turn on the red LED1
22   digitalWrite(red_LED1, HIGH); // This should turn on the RED LED1
23   digitalWrite(yellow_LED1, LOW); // This should turn off the YELLOW LED1
24   digitalWrite(green_LED1, LOW); // This should turn off the GREEN LED1
25   delay(1000); // Extended time for Red light#1 before the Green of the other side turns on
26   // The next three lines of code turn on the green LED2 for 2 seconds
27   digitalWrite(red_LED2, LOW); // This should turn off the RED LED2
28   digitalWrite(yellow_LED2, LOW); // This should turn off the YELLOW LED2
29   digitalWrite(green_LED2, HIGH); // This should turn on the GREEN LED2
30   delay(2000); // wait for 2 seconds
31   // The next three lines of code turn on the red LED1
32   digitalWrite(red_LED1, HIGH); // This should turn on the RED LED1
33   digitalWrite(yellow_LED1, LOW); // This should turn off the YELLOW LED1
34   digitalWrite(green_LED1, LOW); // This should turn off the GREEN LED1
35   // The next three lines of code turn on the yellow LED2
36   digitalWrite(red_LED2, LOW); // This should turn off the RED LED2
```

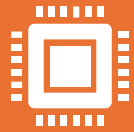
CEIS 114

Module 5

**Creating a Multiple Traffic
Light Controller with a Cross
Walk**



Introduction to Module 5:



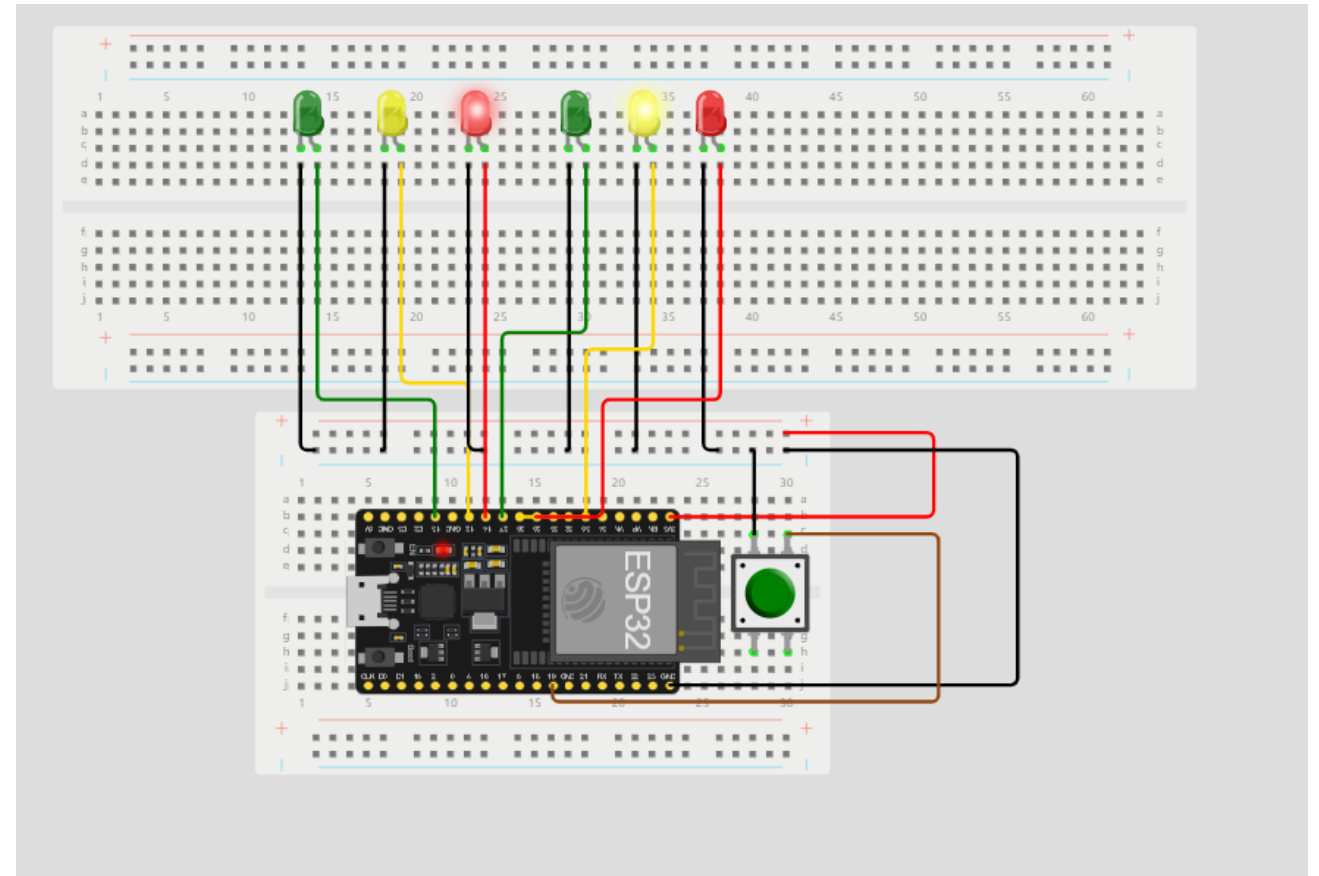
In Module 5, I focus on testing the functionality of our Smart Traffic Light Controller.



This module is crucial for ensuring that our system operates as intended and can effectively manage traffic lights in real-time.

The circuit with working LEDs

- I'll start by showcasing my circuit setup.
- The picture of the breadboard with the LEDs illuminated will give you a clear view of the hardware configuration and how the LEDs are integrated into the system.



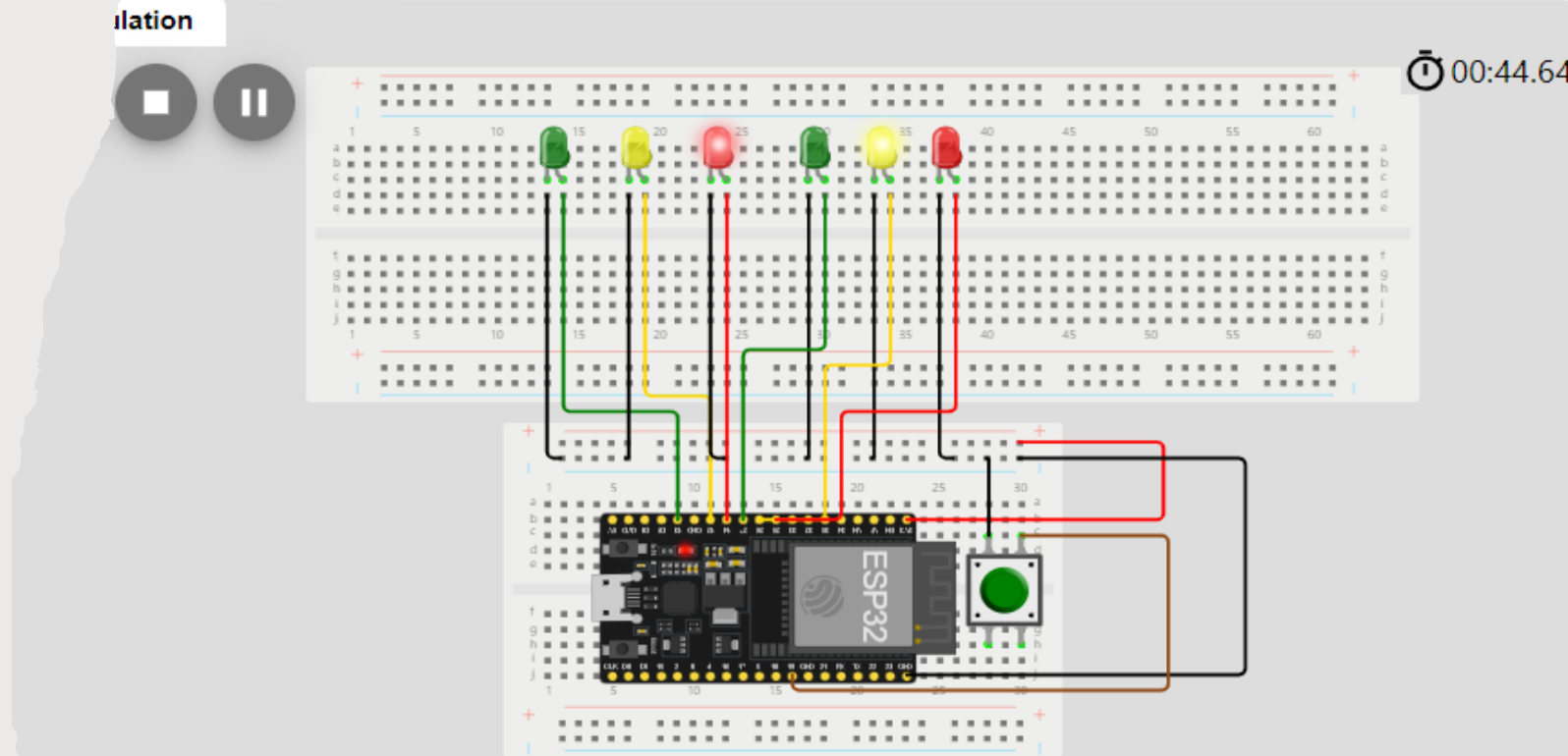
The code in Wokwi

- This code in Wokwi will highlight the segment where I've included my name in the comments, demonstrating the code's customization and personal touch.

```
WOKWI SAVE SHARE Module 5 Hector Acosta by oaxacarican
sketch.ino diagram.json Library Manager
1 // === Hector Acosta ===
2 // Module #5 project
3 const int red_LED1 = 14; // The red LED1 is wired to ESP32 board pin GPIO14
4 const int yellow_LED1 = 12; // The yellow LED1 is wired to ESP32 board pin GPIO12
5 const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
6 const int red_LED2 = 25; // The red LED2 is wired to Mega board pin GPIO25
7 const int yellow_LED2 = 26; // The yellow LED2 is wired to Mega board pin GPIO 26
8 const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27
9 int Xw_value;
10 const int Xw_button = 19; //Cross Walk button
11 // the setup function runs once when you press reset or power the board
12 void setup() {
13   pinMode(Xw_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
14   Serial.begin(115200);
15   pinMode(red_LED1, OUTPUT); // initialize digital pin 14 (Red LED1) as an output.
16   pinMode(yellow_LED1, OUTPUT); // initialize digital pin 12 (yellow LED1) as an output.
17   pinMode(green_LED1, OUTPUT); // initialize digital pin 13 (green LED1) as an output.
18   pinMode(red_LED2, OUTPUT); // initialize digital pin 25 (Red LED2) as an output.
19   pinMode(yellow_LED2, OUTPUT); // initialize digital pin 26 (yellow LED2) as an output.
20   pinMode(green_LED2, OUTPUT); // initialize digital pin 27 (green LED2) as an output.
21 }
22 // the loop function runs over and over again forever
23 void loop() {
24   // read the cross walk button value:
25   Xw_value=digitalRead(Xw_button);
26   if (Xw_value == LOW) { // if crosswalk button (X-button) pressed
27     digitalWrite(yellow_LED1, LOW); // This should turn off the YELLOW LED1
28     digitalWrite(green_LED1, LOW); // This should turn off the GREEN LED1
29     digitalWrite(yellow_LED2, LOW); // This should turn off the YELLOW LED2
30     digitalWrite(green_LED2, LOW); // This should turn off the GREEN LED2
31     for (int i=10; i>0; i--)
32     {
33       Serial.print(" Count = "); Serial.print(i);
34       Serial.println(" == Walk == ");
35       digitalWrite(red_LED1, HIGH); // This should turn on the RED LED1
36       digitalWrite(red_LED2, HIGH); // This should turn on the RED LED2
37       delay(500); //wait 0.5 seconds
```

The Serial Monitor in Wokwi

- I review the output from the Serial Monitor.
- This screenshot will show the live data being transmitted and received, confirming that my system is functioning correctly and that the traffic light controls are responding as expected.



```
== 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 ==
```

CEIS 114

Module 6

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

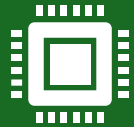
ThePhoto by PhotoAuthor is licensed under CCYYSA.



Introduction to Module 6:



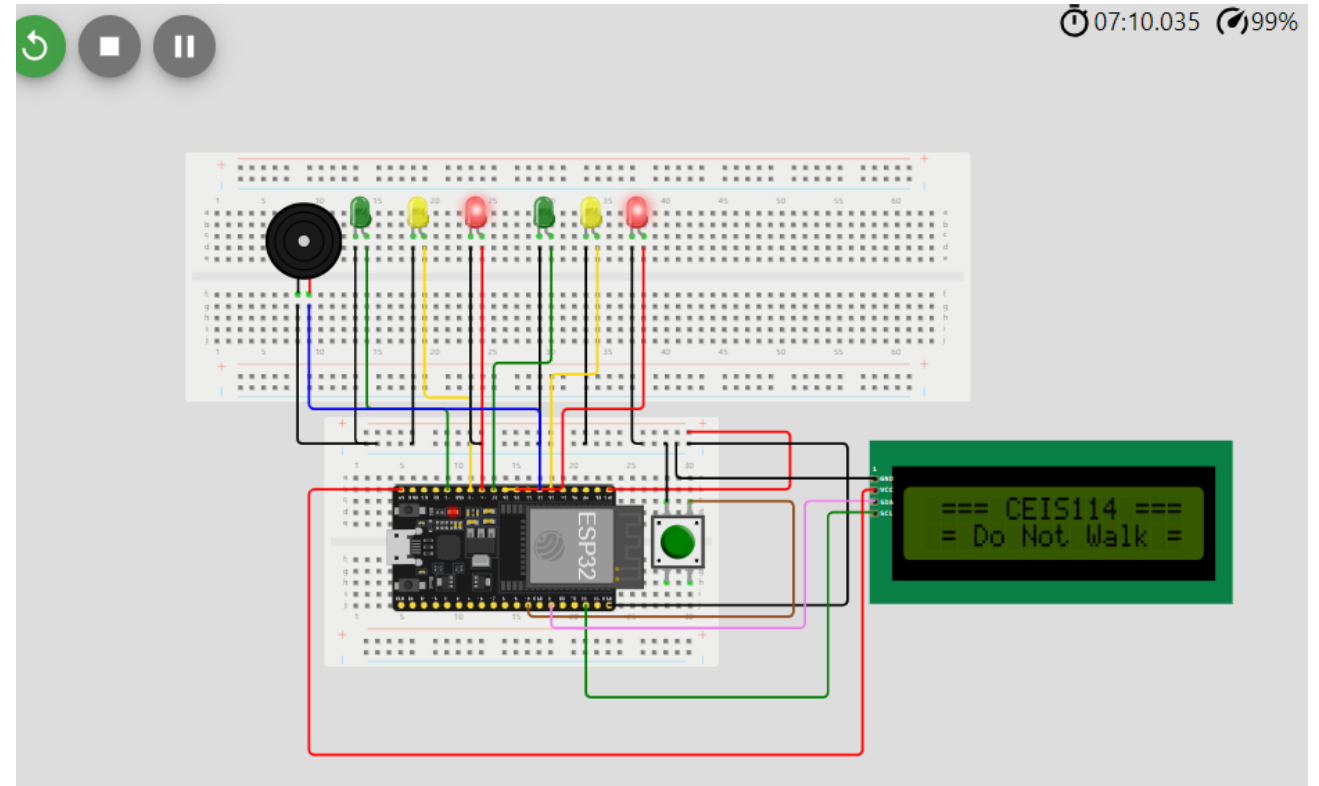
In Module 6, I focus on the crucial phase of testing and validating my Smart Traffic Light Controller project.



This module is designed to ensure that my circuit operates correctly and meets the project requirements.

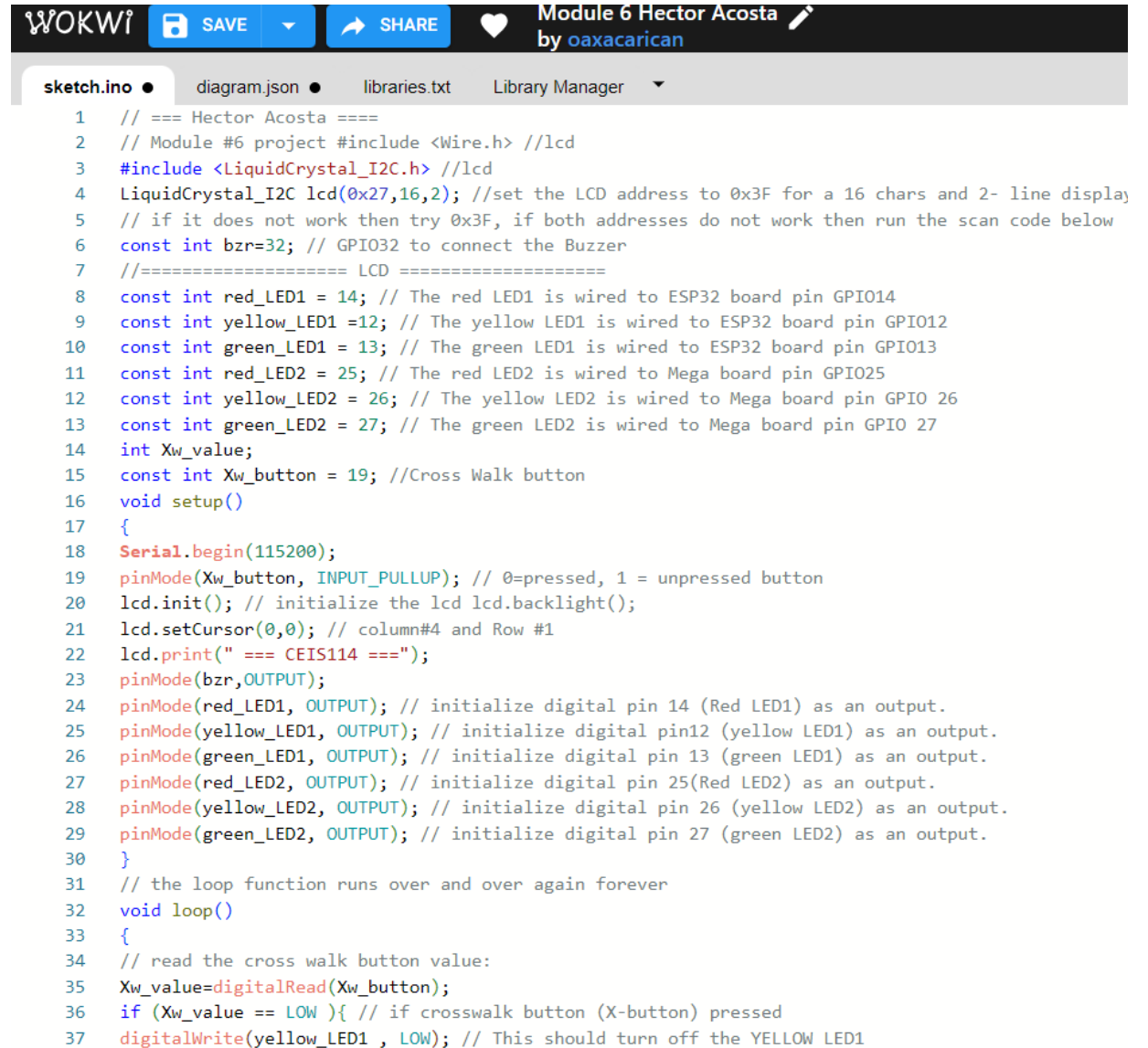
The circuit with working LEDs and LCD display

- I start by examining the physical setup of my project.
- This picture will show the breadboard with the LEDs illuminated and the LCD displaying the intended messages.
- This visual will help me to confirm that the hardware is assembled correctly and functioning as expected.



The code

- I delve into the code behind the project.
- This slide will feature a screenshot of the code with my name included in the comments, showcasing my contribution and understanding of the programming aspects.



```
1 // === Hector Acosta ===
2 // Module #6 project #include <Wire.h> //lcd
3 #include <LiquidCrystal_I2C.h> //lcd
4 LiquidCrystal_I2C lcd(0x27,16,2); //set the LCD address to 0x3F for a 16 chars and 2- line display
5 // if it does not work then try 0x3F, if both addresses do not work then run the scan code below
6 const int bzt=32; // GPIO32 to connect the Buzzer
7 //===== LCD =====
8 const int red_LED1 = 14; // The red LED1 is wired to ESP32 board pin GPIO14
9 const int yellow_LED1 =12; // The yellow LED1 is wired to ESP32 board pin GPIO12
10 const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
11 const int red_LED2 = 25; // The red LED2 is wired to Mega board pin GPIO25
12 const int yellow_LED2 = 26; // The yellow LED2 is wired to Mega board pin GPIO 26
13 const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27
14 int Xw_value;
15 const int Xw_button = 19; //Cross Walk button
16 void setup()
17 {
18   Serial.begin(115200);
19   pinMode(Xw_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
20   lcd.init(); // initialize the lcd lcd.backlight();
21   lcd.setCursor(0,0); // column#4 and Row #1
22   lcd.print(" === CEIS114 ===");
23   pinMode(bzt,OUTPUT);
24   pinMode(red_LED1, OUTPUT); // initialize digital pin 14 (Red LED1) as an output.
25   pinMode(yellow_LED1, OUTPUT); // initialize digital pin12 (yellow LED1) as an output.
26   pinMode(green_LED1, OUTPUT); // initialize digital pin 13 (green LED1) as an output.
27   pinMode(red_LED2, OUTPUT); // initialize digital pin 25(Red LED2) as an output.
28   pinMode(yellow_LED2, OUTPUT); // initialize digital pin 26 (yellow LED2) as an output.
29   pinMode(green_LED2, OUTPUT); // initialize digital pin 27 (green LED2) as an output.
30 }
31 // the loop function runs over and over again forever
32 void loop()
33 {
34   // read the cross walk button value:
35   Xw_value=digitalRead(Xw_button);
36   if (Xw_value == LOW ){ // if crosswalk button (X-button) pressed
37     digitalWrite(yellow_LED1 , LOW); // This should turn off the YELLOW LED1
```

Serial Monitor

- I review the output in the Serial Monitor.
- This screenshot will provide insight into the real-time data and status updates from my ESP32, validating that the data is being processed and displayed as intended.

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 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
no 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
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 ==
== Do Not Walk ==
```


CEIS 114 Module 7 Project- Option 2

**Multiple Traffic Light
Controller with Cross Walk
and an Emergency Buzzer
with a SMART Sensor**



Introduction to Module 7:



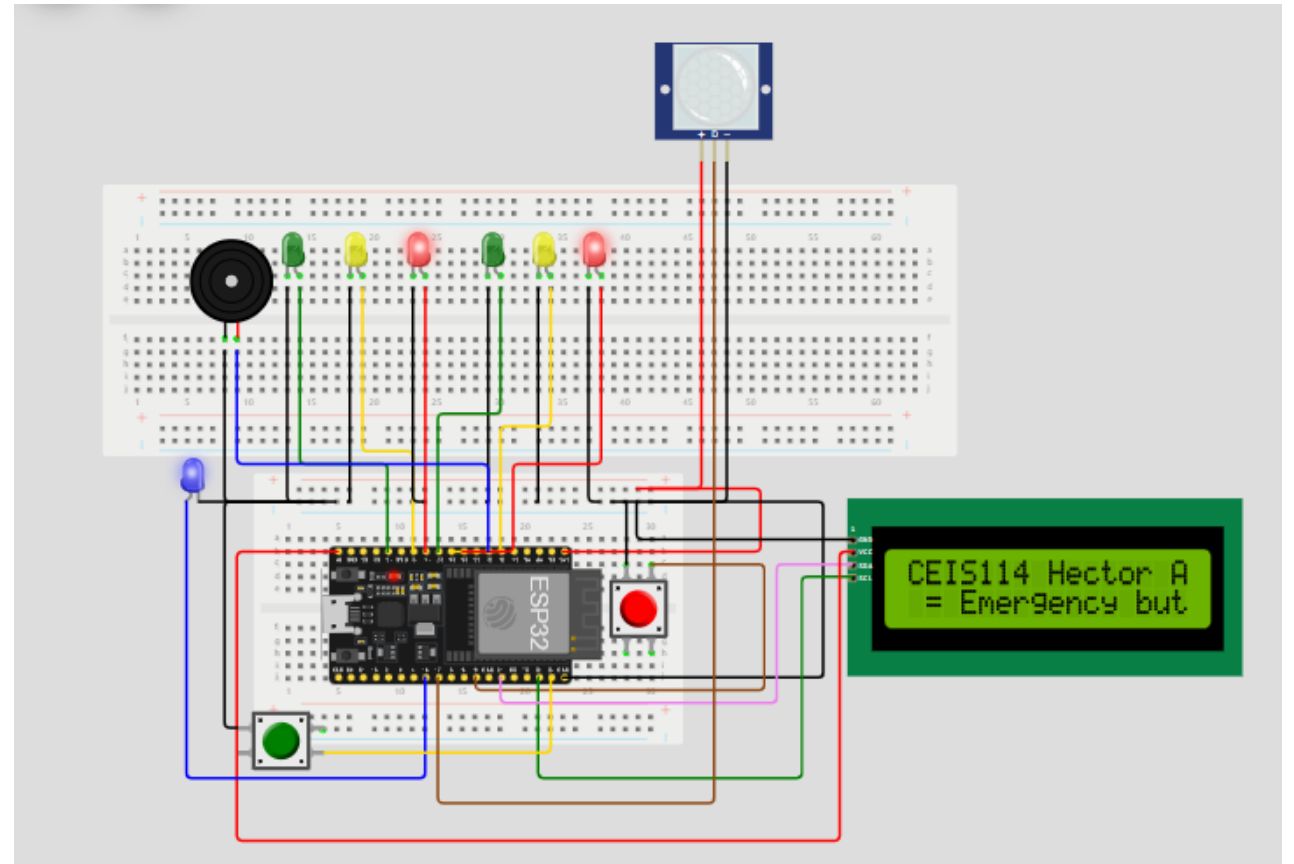
Welcome to Module 7, where I finalize my Smart Traffic Light Controller project. In this module, I focus on ensuring that all components work together seamlessly and meet my project objectives.



In this module, I validate my design, troubleshoot any issues, and confirm that my Smart Traffic Light Controller is ready for deployment. Let's dive into the details of our final testing and operational phase!!

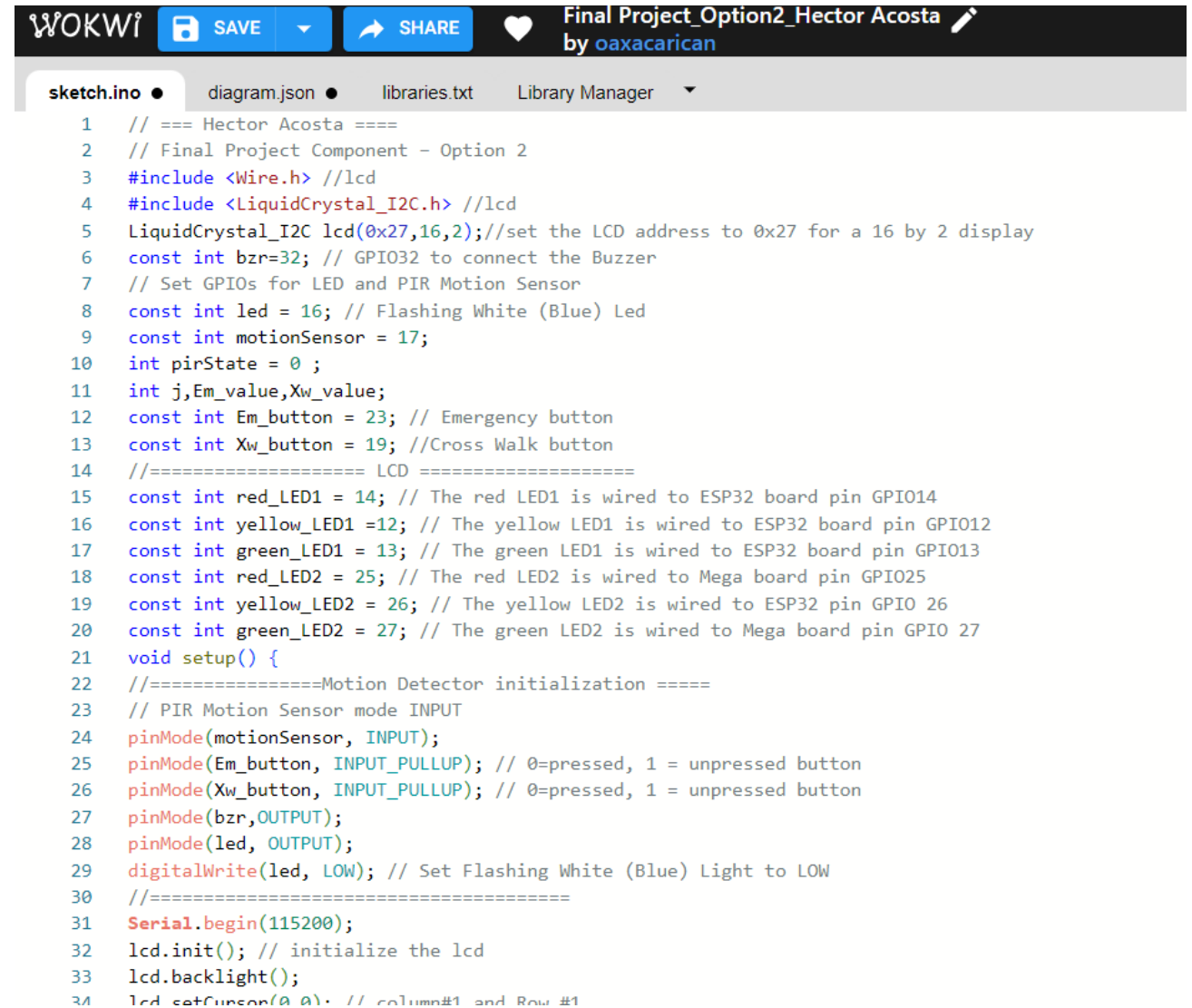
The circuit with working LEDs and LCD display (Building/Operation)

- I start by examining the circuit setup with the working LEDs and LCD display.
- This screenshot will demonstrate the integrated system's operation, highlighting how the LEDs and display respond to various inputs and conditions.



The code

- I delve into the code that drives this operation.
- This slide will showcase the code editor with my final code, providing insight into the logic and functionality behind our system.



```
WOKWI SAVE SHARE Final Project_Option2_Hector Acosta by oaxacarican
sketch.ino diagram.json libraries.txt Library Manager
1 // === Hector Acosta ===
2 // Final Project Component - Option 2
3 #include <Wire.h> //lcd
4 #include <LiquidCrystal_I2C.h> //lcd
5 LiquidCrystal_I2C lcd(0x27,16,2); //set the LCD address to 0x27 for a 16 by 2 display
6 const int bzt=32; // GPIO32 to connect the Buzzer
7 // Set GPIOs for LED and PIR Motion Sensor
8 const int led = 16; // Flashing White (Blue) Led
9 const int motionSensor = 17;
10 int pirState = 0 ;
11 int j,Em_value,Xw_value;
12 const int Em_button = 23; // Emergency button
13 const int Xw_button = 19; //Cross Walk button
14 //===== LCD =====
15 const int red_LED1 = 14; // The red LED1 is wired to ESP32 board pin GPIO14
16 const int yellow_LED1 =12; // The yellow LED1 is wired to ESP32 board pin GPIO12
17 const int green_LED1 = 13; // The green LED1 is wired to ESP32 board pin GPIO13
18 const int red_LED2 = 25; // The red LED2 is wired to Mega board pin GPIO25
19 const int yellow_LED2 = 26; // The yellow LED2 is wired to ESP32 pin GPIO 26
20 const int green_LED2 = 27; // The green LED2 is wired to Mega board pin GPIO 27
21 void setup() {
22 //=====Motion Detector initialization =====
23 // PIR Motion Sensor mode INPUT
24 pinMode(motionSensor, INPUT);
25 pinMode(Em_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
26 pinMode(Xw_button, INPUT_PULLUP); // 0=pressed, 1 = unpressed button
27 pinMode(bzt,OUTPUT);
28 pinMode(led, OUTPUT);
29 digitalWrite(led, LOW); // Set Flashing White (Blue) Light to LOW
30 //=====
31 Serial.begin(115200);
32 lcd.init(); // initialize the lcd
33 lcd.backlight();
34 lcd.setCursor(0,0); // column#1 and Row #1
```

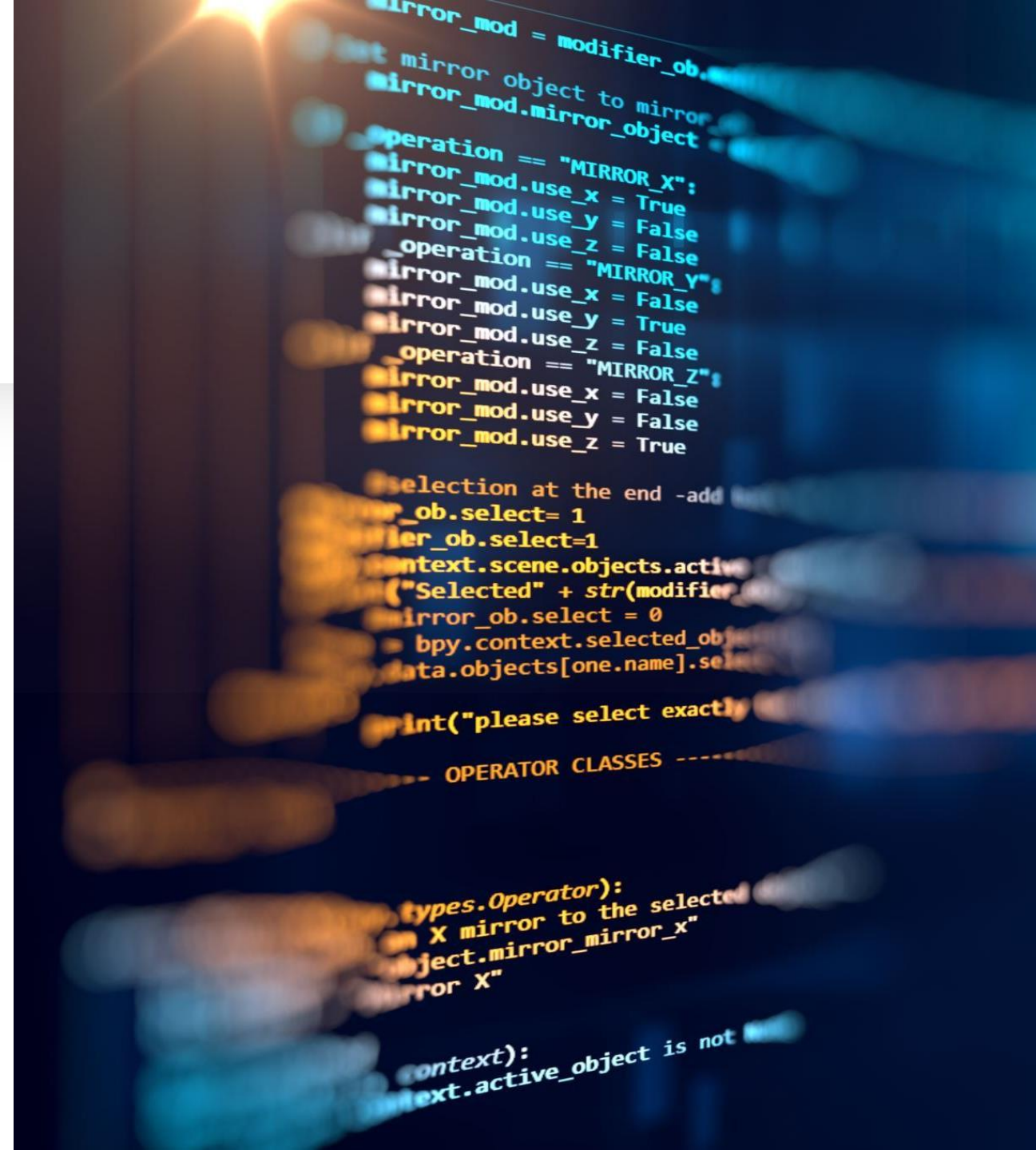

Serial Monitor

- I review the Serial Monitor output, which confirms that my system is functioning as intended.
- This screenshot will illustrate how the system processes and displays real-time data, ensuring that everything operates smoothly.

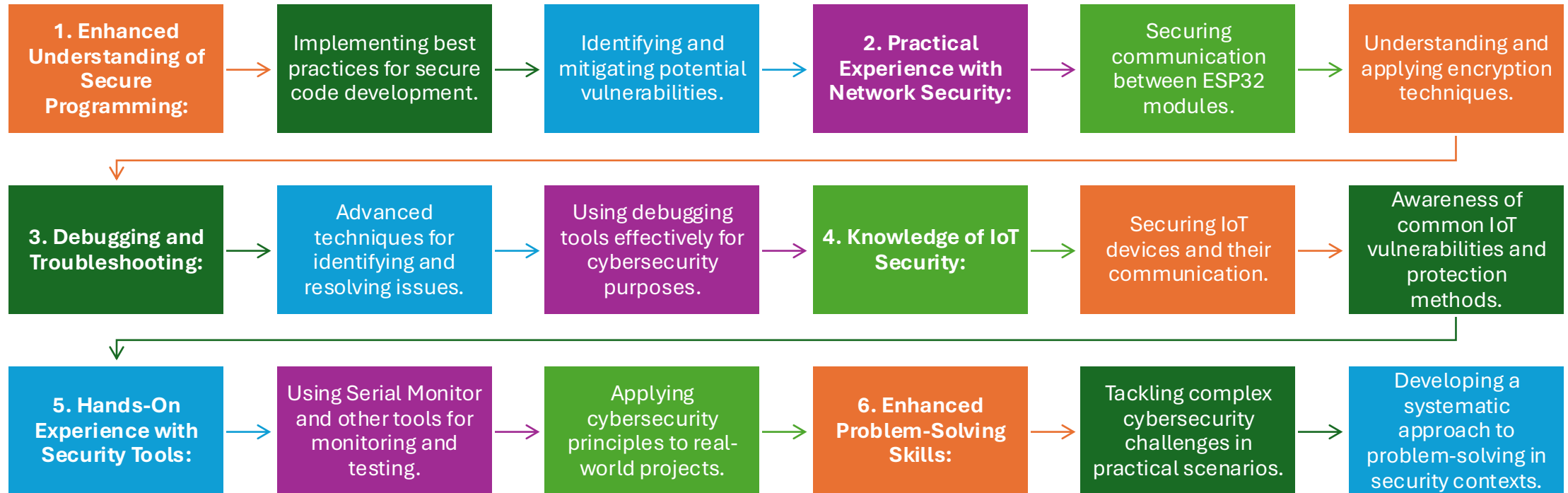
```
== Do Not Walk ==  
== Do Not Walk ==  
Emergency button was pressed  
== Do Not Walk ==  
== Do Not Walk ==  
Count = 10 == Walk ==  
Count = 9 == Walk ==  
Count = 8 == Walk ==  
Count = 7 == Walk ==  
Count = 6 == Walk ==  
Count = 5 == Walk ==  
Count = 4 == Walk ==  
Count = 3 == Walk ==  
Count = 2 == Walk ==  
Count = 1 == Walk ==  
Count = 0 == Walk ==  
== Do Not Walk ==  
== Do Not Walk ==  
== Do Not Walk ==  
motion detected
```

Challenges Faced

- **Debugging the Programs**
- **Issue Identification:**
 - Difficulty pinpointing errors in code.
 - Complex interactions between hardware and software.
- **Troubleshooting Steps:**
 - Extensive code reviews.
 - Implementing debug statements and using Serial Monitor.
 - Testing different configurations and scenarios.
- **Resolution:**
 - Refined code through iterative testing.
 - Adjusted hardware connections for better stability.
- **Lessons Learned:**
 - Importance of systematic debugging.
 - Enhanced understanding of ESP32 programming and hardware integration.



Skills Gained



Conclusion

To conclude, my Smart Traffic Light Controller project successfully demonstrated the integration of ESP32 with LEDs and an LCD display to manage traffic light sequences.

I effectively addressed design challenges, optimized our system through testing, and gained valuable skills in both hardware and software development.

This project not only met its objectives but also paved the way for future enhancements and applications.

Sources

- **Course CEIS114 Project: Overview and Expectations**
 - Detailed module guides
 - Instructional videos
- **Live Lectures**
 - Various Professors from DeVry University

