Traffic Light System with Train Detection - Code Explanation

This code is designed to control a traffic light system, which is influenced by an external event—specifically, the arrival of a train. The system manages two sets of traffic lights (labeled A and B) and includes a button to simulate the detection of an approaching train, a buzzer to warn of the train's presence, and a timing mechanism to cycle through the lights.

# Pin Assignments and Variables:

At the start of the code, different hardware components are assigned specific pins. This includes pins for two sets of LEDs representing traffic lights (red, green, and blue for each set A and B), a potentiometer for potential manual adjustments, a button to detect a train, and a buzzer for an auditory warning. Several timing-related variables are initialized to manage the duration of green lights and orange lights, as well as a delay representing the train crossing time. Flags and timers are set up to manage the states of the system, particularly for debouncing the button and keeping track of whether the train is approaching.

# Setup Function:

The setup() function initializes the traffic light system. It starts by setting the serial communication for debugging purposes and then configures the pin modes for the LEDs, button, and buzzer. Initially, all LEDs and the buzzer are turned off to ensure a clean start for the system.

# Loop Function:

In the main loop, the code constantly checks for two conditions: (1) whether a train is approaching, and (2) how to manage the traffic light cycle when the train is not present. The millis() function is used to track time without pausing the entire system.

The button state is checked using digitalRead(), and a debounce logic ensures that the button isn't falsely triggered by noise. If the button is pressed, indicating a train's approach, the system enters a special mode to handle the train's presence (handleTrainApproaching).

# Traffic Light Management:

If no train is detected, the system runs the normal traffic light cycle in the handleTrafficLights() function. The traffic lights for road A and road B alternate between green, orange (simulated by blue LEDs), and red, with specific durations for each. State management using a switch-case structure ensures that the lights transition between these states in a coordinated manner. For example, in state 0, the green light for road A is on while road B’s red light is on. The system then transitions to state 1, where the green light on road A turns to orange, and so on.

# Train Handling:

When a train is detected (button press), the handleTrainApproaching() function is activated. This function turns both traffic lights red to stop all vehicles. At the same time, a warning signal is triggered by flashing an LED (representing a warning light) and activating the buzzer in a blinking fashion. This continues for the duration defined by trainDelay, simulating the train crossing time. After the train has passed, the system resets back to normal traffic light operation.

# Summary:

This code effectively manages a traffic light system with a train crossing simulation. It ensures that when a train is approaching, all vehicles are stopped with a visual and auditory warning, and the system safely returns to its normal state after the train has passed. The use of timing and state management allows for smooth transitions between traffic light signals and train alerts (**See the code in Appendix A**)

State Machine for Traffic Light and Train Control

This document describes the state machine for controlling traffic lights and handling a train approaching scenario. The state machine has two major functions: normal traffic light operation and train approaching control.

# States and Transitions

## State 0

Green light A, Red light B

## State 1

Orange light A, Red light B

## State 2

Red light A, Green light B

## State 3

Red light A, Orange light B

## Train Approaching

All lights red, buzzer and warning light flashing

# State Machine Diagram

The following diagram represents the state machine for the traffic light and train control system:

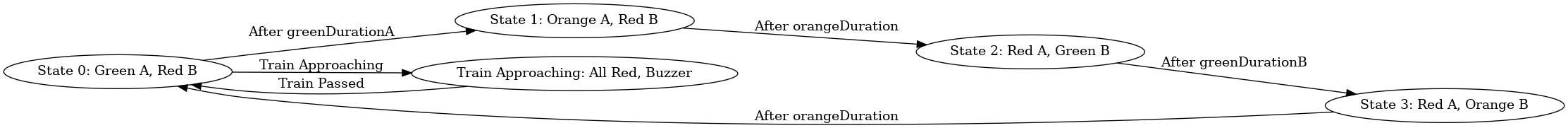


Figure 1: State Machine Diagram

Use of Potentiometer in ESP32-S3 Code - Summary

In ESP32-S3 code, a potentiometer is typically used as an analog input device to adjust parameters dynamically. It is connected to an analog pin, and the `analogRead()` function is used to read its value. The potentiometer outputs a variable voltage, which can be mapped to control various features in the program.

For example, in a traffic light system, the potentiometer can be used to adjust the duration of the green or orange lights by reading its value and mapping it to a desired time range. This provides real-time control over the timing without needing to hard-code specific values.

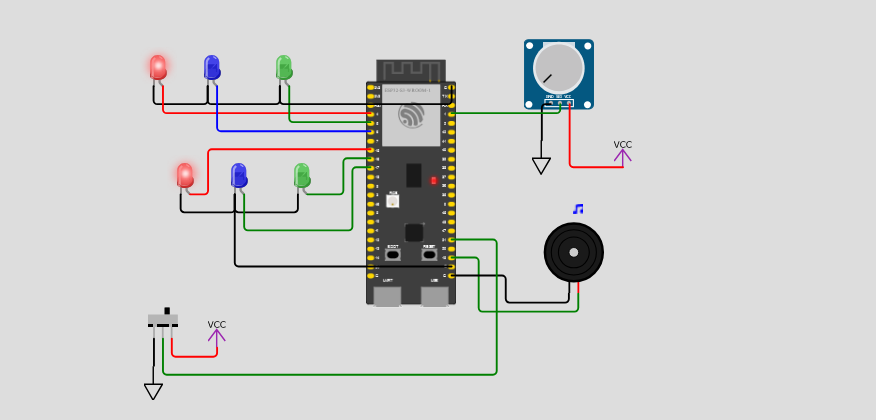


Figure 2: Train is coming

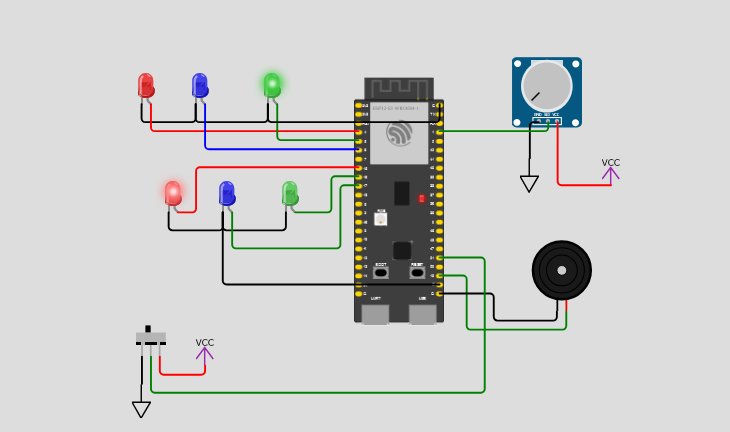


Figure 3: Normal operation for Road A

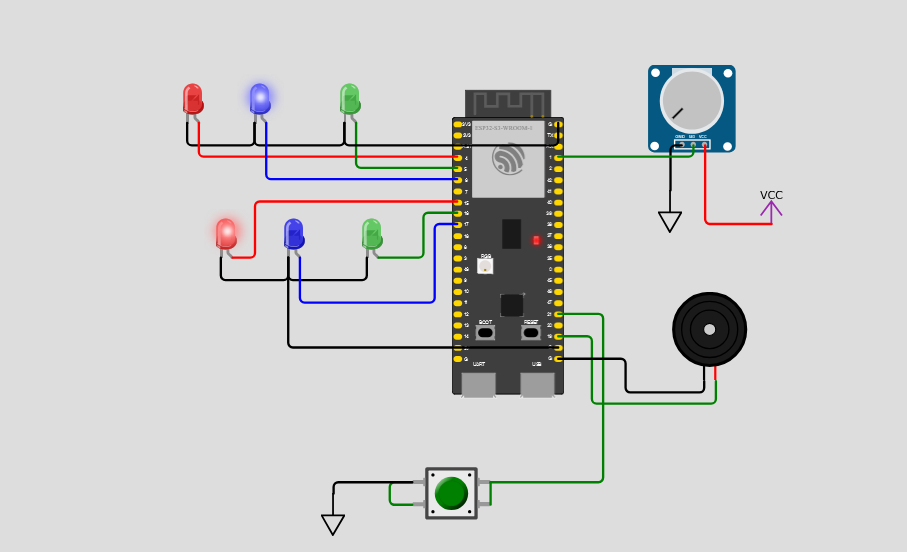


Figure 4: Normal operation for Road A

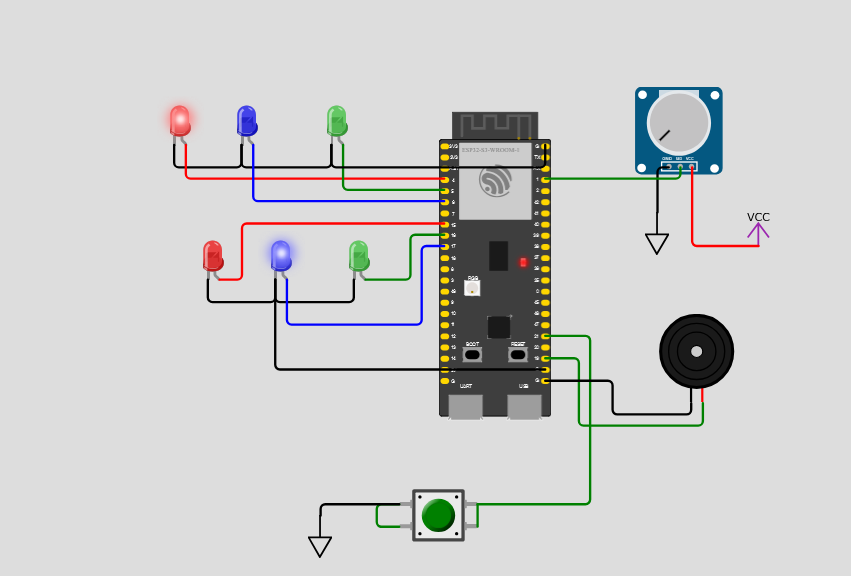


Figure 5: Normal operation for Road B

**Appendix A**

**Code:**

#define RED\_LED\_A 4

#define GREEN\_LED\_A 5

#define BLUE\_LED\_A 6

#define RED\_LED\_B 15

#define GREEN\_LED\_B 16

#define BLUE\_LED\_B 17

#define POTENTIOMETER\_PIN 1

#define BUTTON\_PIN 21

#define BUZZER\_PIN 19

// Timing Variables

int greenDurationA = 10;   // Initial green light duration for A (can be adjusted)

int greenDurationB = 20;   // Initial green light duration for B (can be adjusted)

int orangeDuration = 3;    // Orange light duration for all

int trainDelay = 3;       // Red light delay when the train is passing

// Train status

bool trainApproaching = false;   // Train is not approaching initially

bool buttonPressed = false;      // Flag to track if button is pressed

unsigned long buttonPressTime = 0; // To debounce the button

// Timer Variables

unsigned long previousMillis = 0;

int state = 0;

void setup() {

    // Initialize serial communication

**Serial**.begin(115200);

    // Set pin modes

    pinMode(RED\_LED\_A, OUTPUT);

    pinMode(GREEN\_LED\_A, OUTPUT);

    pinMode(BLUE\_LED\_A, OUTPUT);

    pinMode(RED\_LED\_B, OUTPUT);

    pinMode(GREEN\_LED\_B, OUTPUT);

    pinMode(BLUE\_LED\_B, OUTPUT);

    pinMode(POTENTIOMETER\_PIN, INPUT);

    pinMode(BUTTON\_PIN, INPUT);

    pinMode(BUZZER\_PIN, OUTPUT);

    // Initialize LEDs and buzzer off

    digitalWrite(RED\_LED\_A, LOW);

    digitalWrite(GREEN\_LED\_A, LOW);

    digitalWrite(BLUE\_LED\_A, LOW);

    digitalWrite(RED\_LED\_B, LOW);

    digitalWrite(GREEN\_LED\_B, LOW);

    digitalWrite(BLUE\_LED\_B, LOW);

    digitalWrite(BUZZER\_PIN, LOW);

}

void loop() {

    unsigned long currentMillis = millis();

    // Check if train is approaching (button press)

    int buttonState = digitalRead(BUTTON\_PIN);

    // Debounce the button

    if (buttonState == HIGH && !buttonPressed && (currentMillis - buttonPressTime) > 200) {

        buttonPressed = true;

        trainApproaching = true;

        buttonPressTime = currentMillis;

    }

    if (trainApproaching) {

        handleTrainApproaching(currentMillis);

    } else {

        handleTrafficLights(currentMillis);

    }

}

void handleTrafficLights(unsigned long currentMillis) {

    switch (state) {

        case 0:  // Green light A, Red light B

            digitalWrite(GREEN\_LED\_A, HIGH);

            digitalWrite(RED\_LED\_B, HIGH);

            if (digitalRead(BUTTON\_PIN) == HIGH) {

              trainApproaching = true;

                    }

            if (currentMillis - previousMillis >= greenDurationA \* 1000) {

                previousMillis = currentMillis;

                state = 1;  // Move to Orange for A

            }

            break;

        case 1:  // Orange light A, Red light B

            digitalWrite(GREEN\_LED\_A, LOW);

            digitalWrite(BLUE\_LED\_A, HIGH);  // Orange simulated as blue

            if (currentMillis - previousMillis >= orangeDuration \* 1000) {

                previousMillis = currentMillis;

                state = 2;  // Move to Green B

            }

            break;

        case 2:  // Red light A, Green light B

            digitalWrite(BLUE\_LED\_A, LOW);

            digitalWrite(RED\_LED\_A, HIGH);

            digitalWrite(RED\_LED\_B, LOW);

            digitalWrite(BLUE\_LED\_B, HIGH);

            if (currentMillis - previousMillis >= orangeDuration \* 1000) {

                previousMillis = currentMillis;

                state = 3;  // Move to Orange for B

            }

            break;

        case 3:  // Red light A, Orange light B

            digitalWrite(RED\_LED\_B, LOW);

            digitalWrite(BLUE\_LED\_B, LOW);

            digitalWrite(GREEN\_LED\_B, HIGH);

            if (currentMillis - previousMillis >= greenDurationB \* 1000) {

                previousMillis = currentMillis;

                digitalWrite(RED\_LED\_A, LOW);

                digitalWrite(GREEN\_LED\_A, LOW);

                digitalWrite(BLUE\_LED\_A, LOW);

                digitalWrite(RED\_LED\_B, LOW);

                digitalWrite(GREEN\_LED\_B, LOW);

                digitalWrite(BLUE\_LED\_B, LOW);

                digitalWrite(BUZZER\_PIN, LOW);

                state = 0;  // Back to Green A

            }

            break;

    }

}

void handleTrainApproaching(unsigned long currentMillis) {

    // Turn all lights red

    digitalWrite(RED\_LED\_A, HIGH);

    digitalWrite(RED\_LED\_B, HIGH);

    digitalWrite(GREEN\_LED\_A, LOW);

    digitalWrite(GREEN\_LED\_B, LOW);

    digitalWrite(BLUE\_LED\_A, LOW);

    digitalWrite(BLUE\_LED\_B, LOW);

    // Flash built-in LED and sound buzzer

    if ((currentMillis / 500) % 2 == 0) {

        digitalWrite(BUZZER\_PIN, HIGH);  // Buzzer on

        digitalWrite(LED\_BUILTIN, HIGH); // Flash warning light

    } else {

        digitalWrite(BUZZER\_PIN, LOW);   // Buzzer off

        digitalWrite(LED\_BUILTIN, LOW);  // Warning light off

    }

    digitalWrite(RED\_LED\_A, LOW);

    digitalWrite(RED\_LED\_B, LOW);

    digitalWrite(GREEN\_LED\_A, LOW);

    digitalWrite(GREEN\_LED\_B, LOW);

    digitalWrite(BLUE\_LED\_A, LOW);

    digitalWrite(BLUE\_LED\_B, LOW);

    // Delay for train crossing time

    if (currentMillis - previousMillis >= trainDelay \* 1000) {

        previousMillis = currentMillis;

        trainApproaching = false;   // Train has passed

        buttonPressed = false;      // Reset button pressed flag

        digitalWrite(RED\_LED\_A, LOW);

        digitalWrite(RED\_LED\_B, LOW);

        digitalWrite(GREEN\_LED\_A, LOW);

        digitalWrite(GREEN\_LED\_B, LOW);

        digitalWrite(BLUE\_LED\_A, LOW);

        digitalWrite(BLUE\_LED\_B, LOW);

        state = 0;                  // Reset state to normal operation

    }

}