

**ESP32 Basics-1 Using ESP-IDF (Industry style Projects using LED and buzzer)**

**1. INTRODUCTION**

Embedded systems are dedicated computing systems designed to perform a **specific function** within a larger system. Unlike general-purpose computers, embedded systems are optimized for **reliability, real-time operation, and low power consumption**.

The **ESP32** microcontroller is a popular and powerful embedded platform widely used in **IoT, industrial control, automation, and learning environments**. It combines high performance with integrated communication features, making it suitable for both beginners and advanced developers.

In this course/lab, we will learn **basic embedded programming concepts** using the **ESP32 microcontroller** and the **ESP-IDF (Espressif IoT Development Framework)**.

**2. ABOUT ESP32 MICROCONTROLLER**

The ESP32 is a **32-bit microcontroller** developed by Espressif Systems. It is designed for embedded applications that require **high processing capability**, **real-time response**, and **wireless connectivity**.

**Key Features of ESP32:**

* Dual-core 32-bit processor
* Operates at up to **240 MHz**
* Built-in **Wi-Fi** and **Bluetooth**
* Rich set of **GPIO pins**
* Multiple timers, PWM, ADC, DAC, UART, SPI, I2C
* Low-power modes for energy-efficient applications

Because of these features, ESP32 is widely used in:

* Embedded learning labs
* Industrial monitoring systems
* IoT-based automation
* Smart devices

**3. ESP32 DEVELOPMENT BOARD OVERVIEW**

An ESP32 development board integrates the ESP32 chip along with supporting components such as:

* USB-to-Serial converter
* Voltage regulator
* Boot and reset buttons
* Header pins for GPIO access

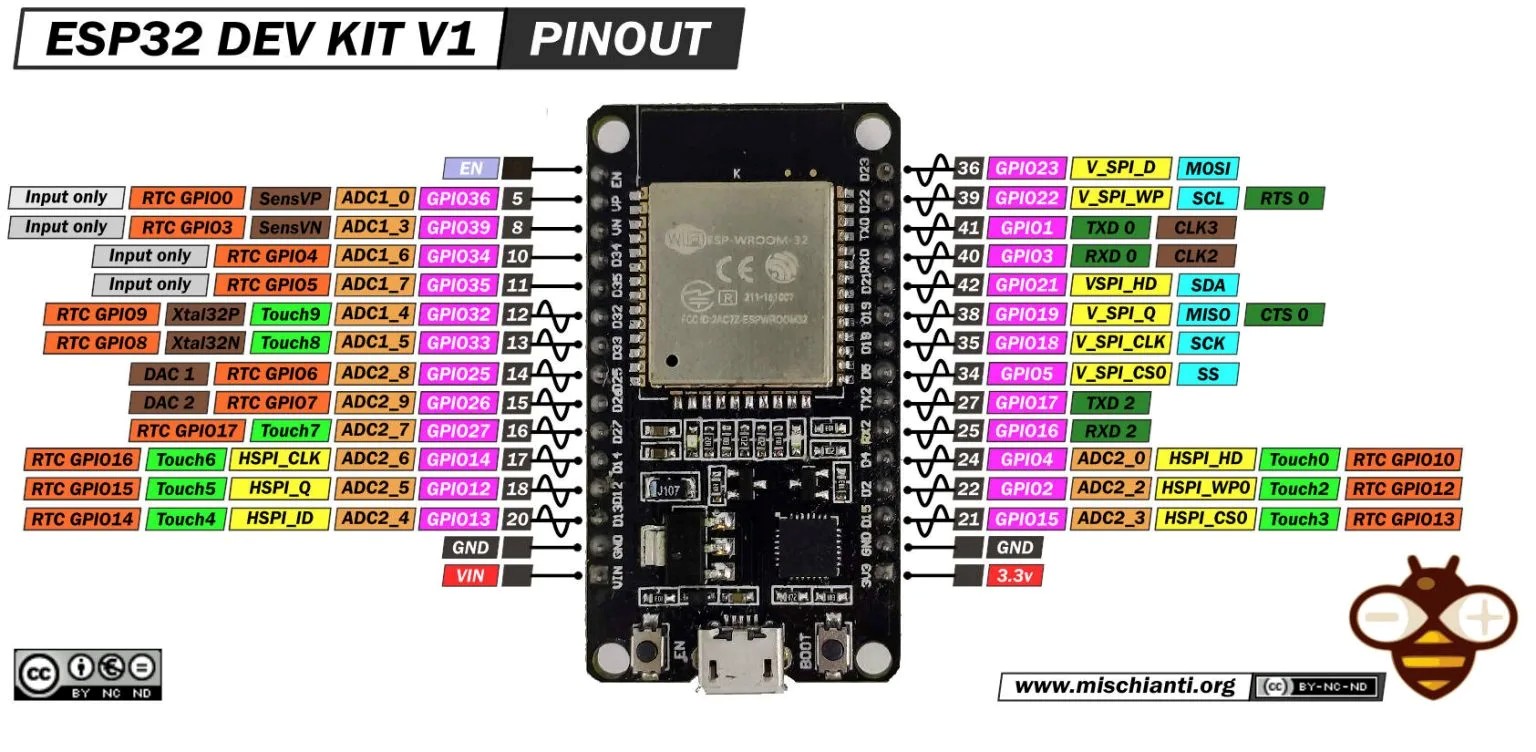
The development board allows easy connection to a computer for **programming and debugging**.



### Power Supply:

* Powered through **USB (5V)**
* Onboard regulator converts 5V to **3.3V**
* ESP32 operates internally at **3.3V**

**Important Note:**  
ESP32 GPIO pins are **NOT 5V tolerant**. Applying 5V directly to GPIO pins may permanently damage the chip.



## 4. ESP-IDF (ESPRESSIF IOT DEVELOPMENT FRAMEWORK)

ESP-IDF is the **official software development framework** provided by Espressif for programming ESP32.

Unlike Arduino ide, ESP-IDF:

* Uses **industry-standard C programming**
* Follows a **modular and professional project structure**
* Supports **FreeRTOS** (Real-Time Operating System)
* Provides better control over hardware resources

### Why ESP-IDF?

* Used in **real industrial products**
* Encourages **good coding practices**
* Helps students transition from hobby-level coding to **professional embedded firmware development**



## 5. BASIC ESP-IDF PROJECT STRUCTURE

An ESP-IDF project typically contains the following folders:

project\_folder/

│

├── main/

│ └── main.c

│

├── CMakeLists.txt

└── sdkconfig

### Description:

* **main.c** – contains application logic
* **CMakeLists.txt** – build configuration file
* **sdkconfig** – ESP32 system configuration

The app\_main() function is the **entry point** of every ESP-IDF application.

**ESP-IDF installation in VS code with Platform-IO extension**[**: https://www.youtube.com/watch?v=zNZYxm92T6I**](:%20https:/www.youtube.com/watch?v=zNZYxm92T6I%20)



**Project-1: Blink on-board LED**

### ****Objective:**** To understand basic GPIO configuration and control by blinking an LED using the ESP32 microcontroller with ESP-IDF.

### ****Learning Outcome:**** Able to configure an ESP32 GPIO pin as output and control an LED ON and OFF using embedded C programming in ESP-IDF.

### Procedure:

### Step-1: Open VS code

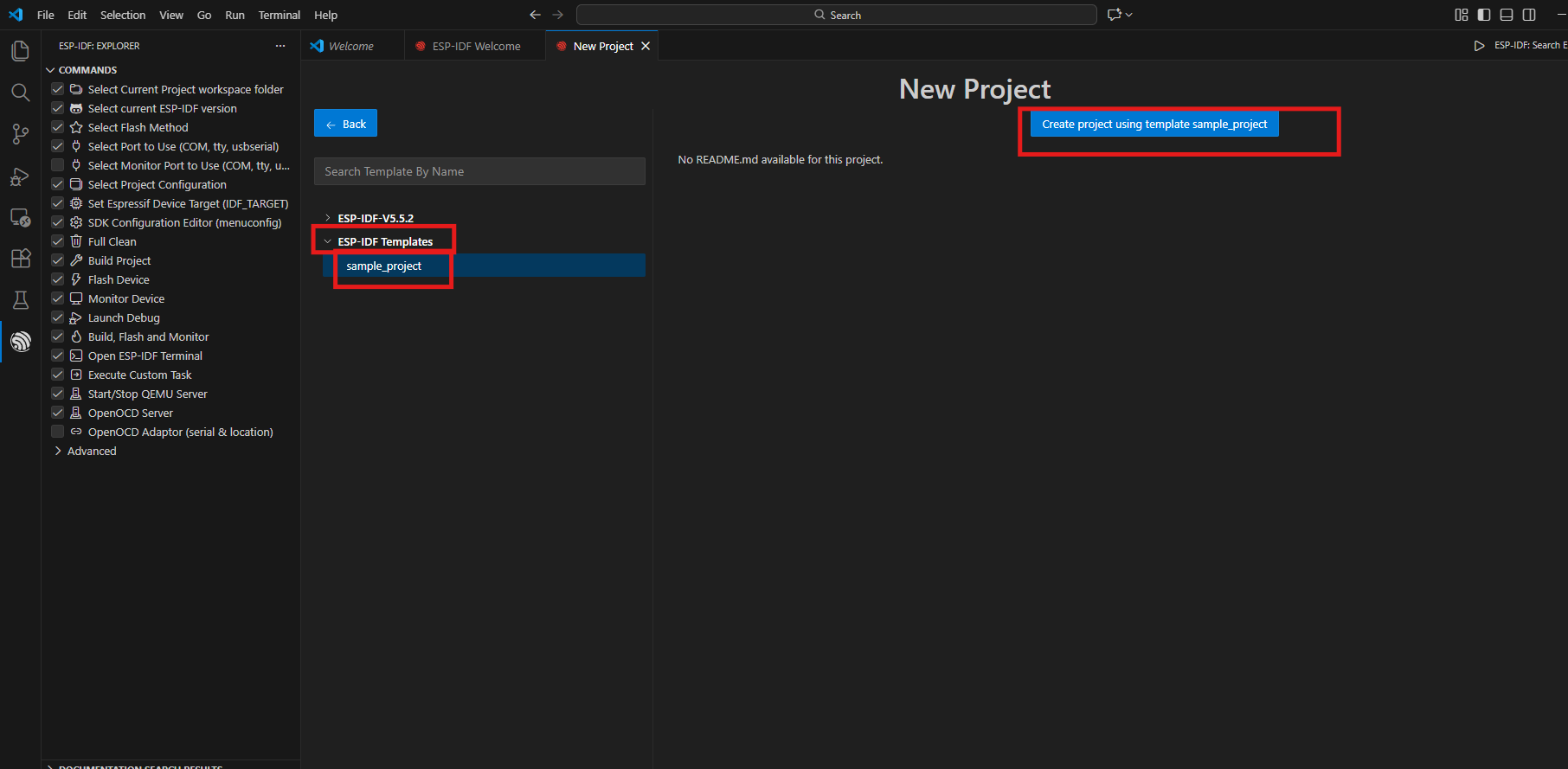
### 

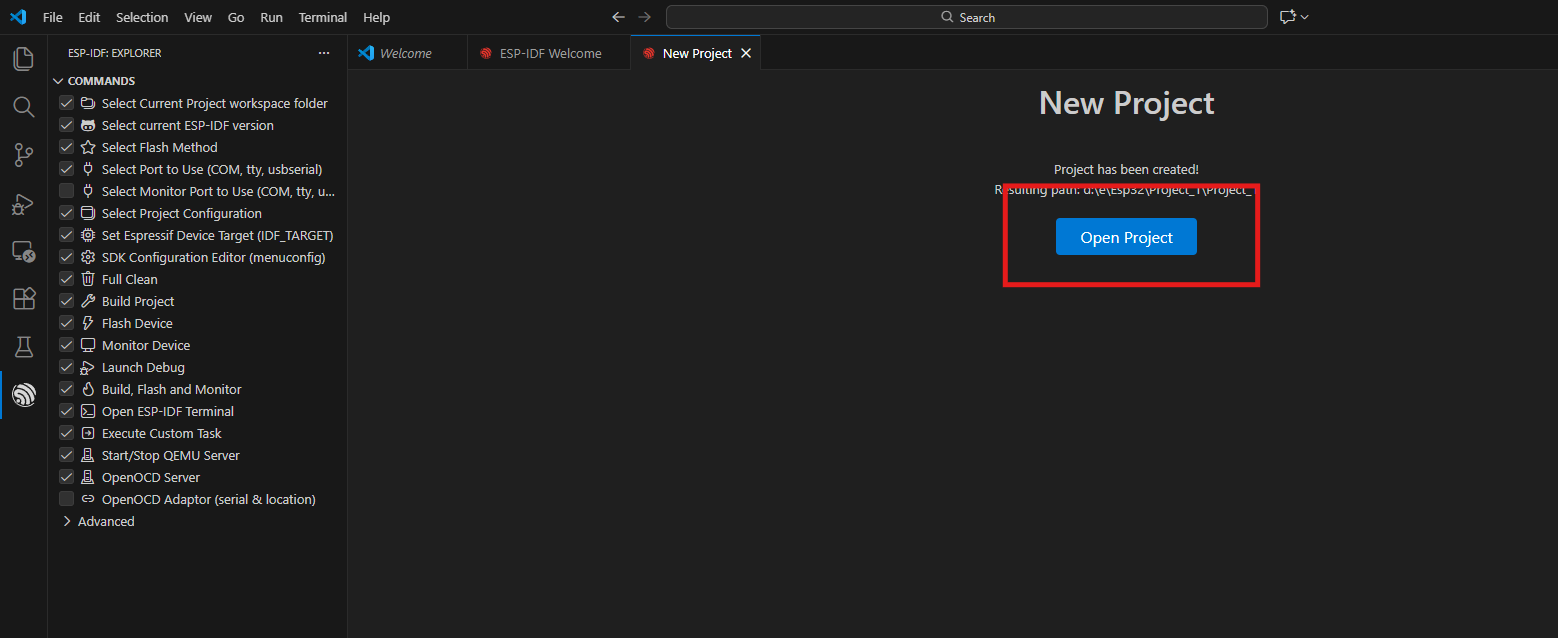
### Step-2:

### 

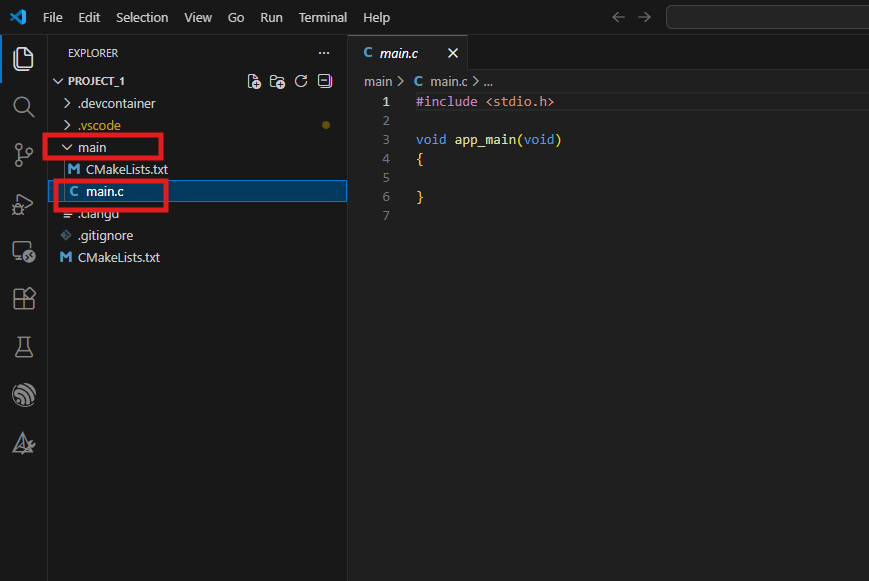


**Step-3:**



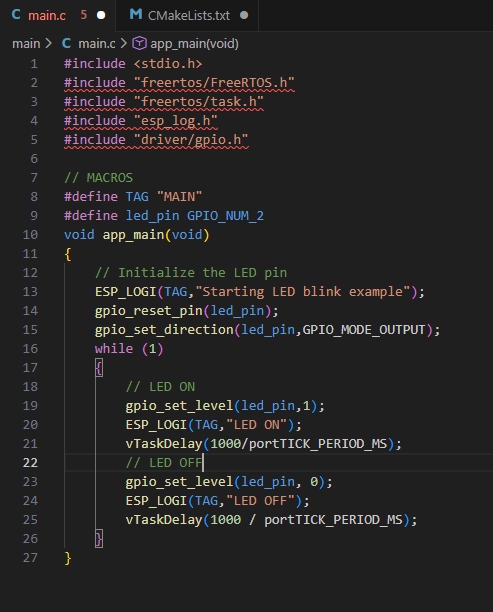


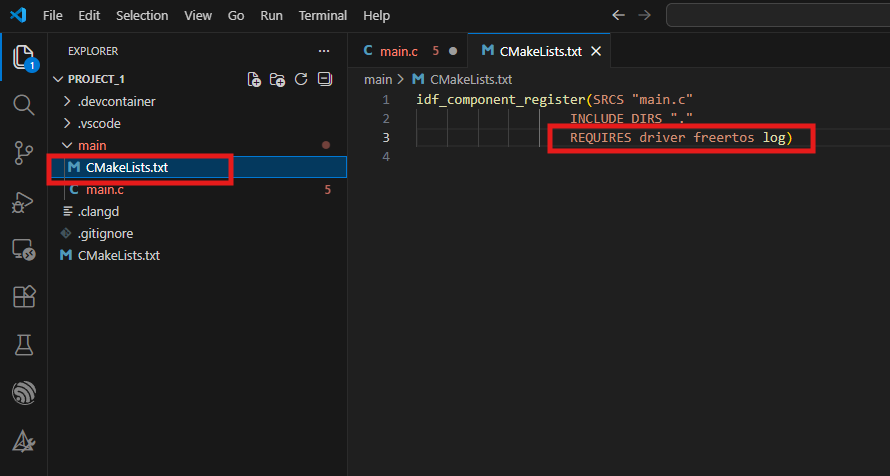
**Step-4:**





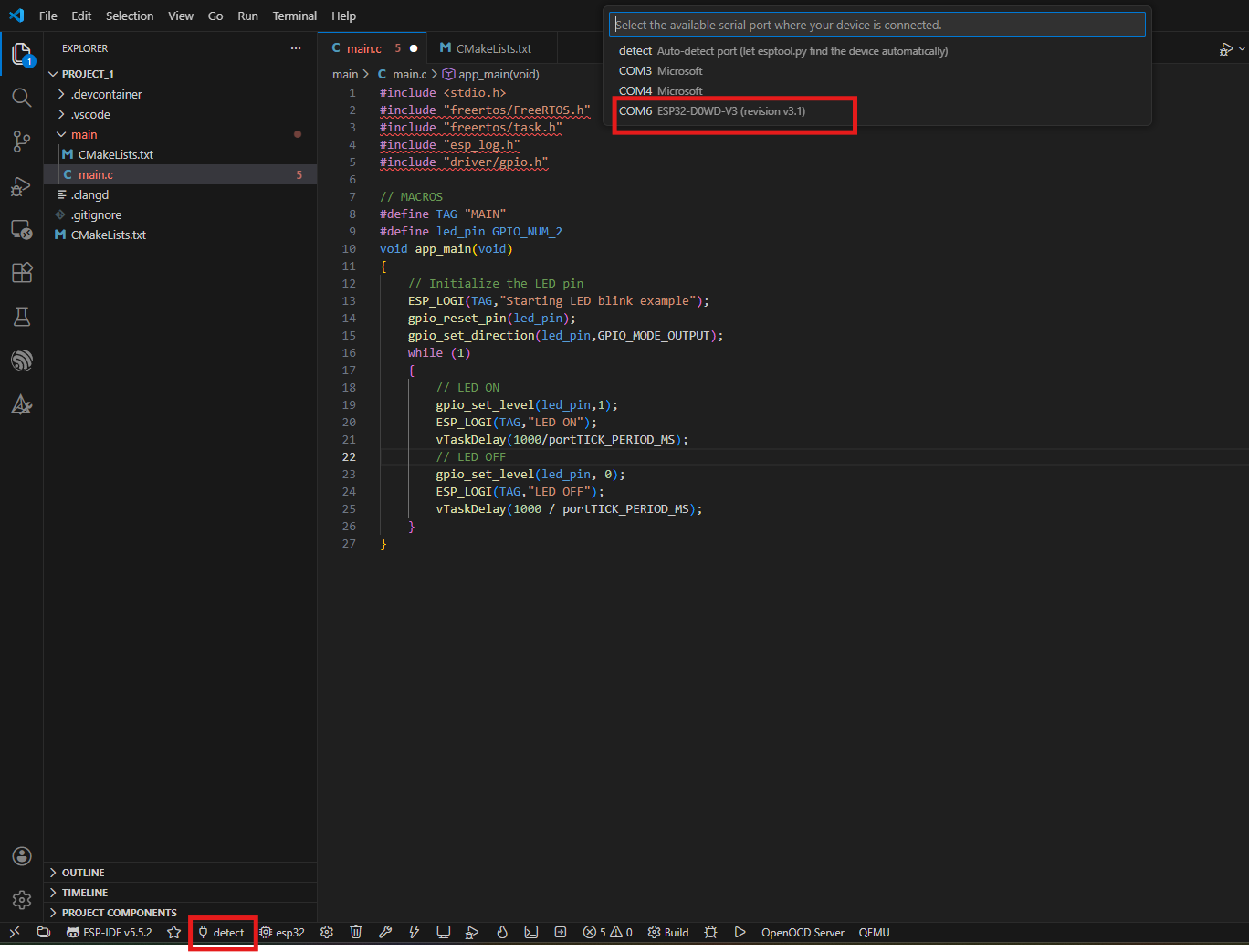
**Code:**



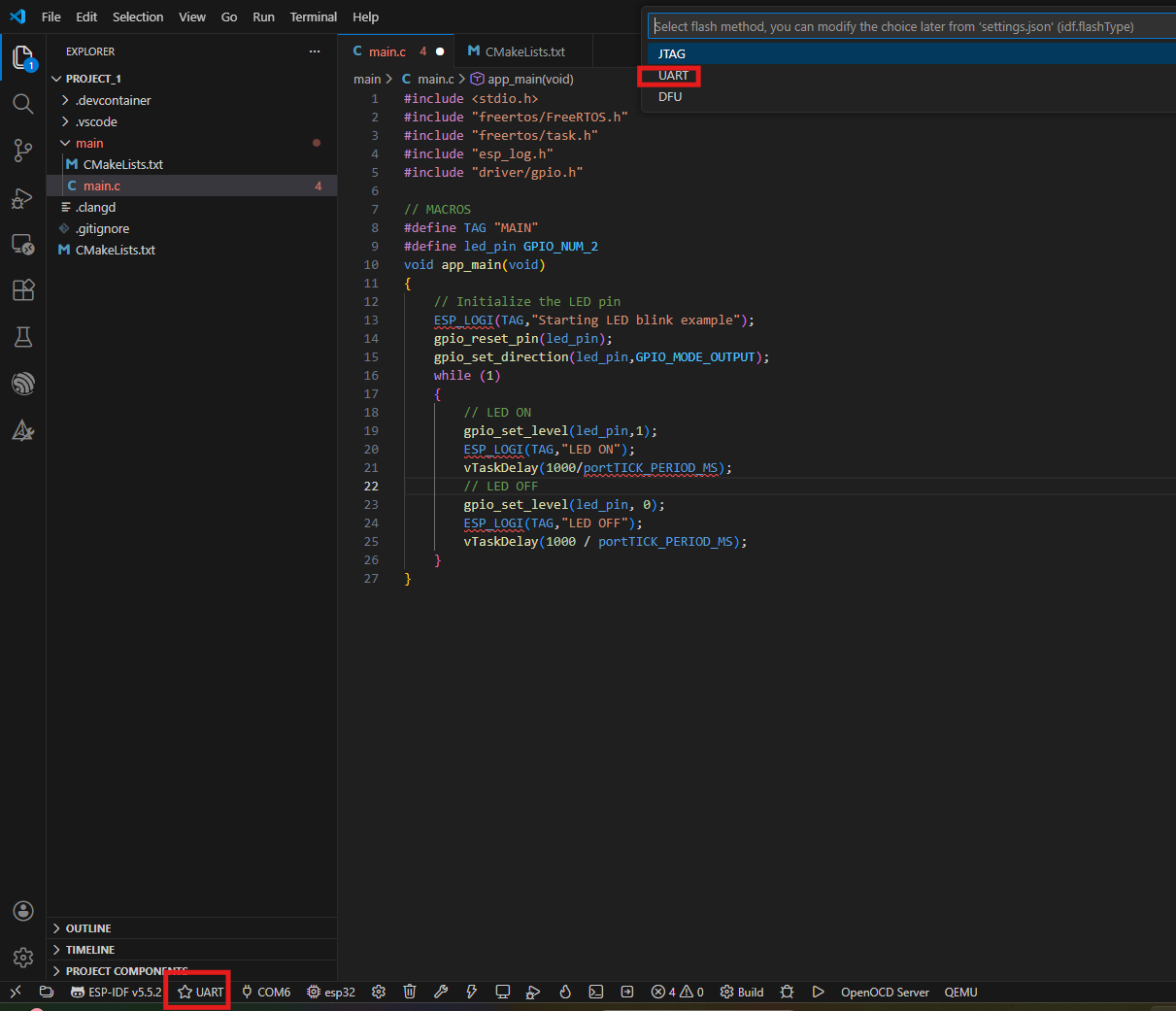




**Step-5:**



**Step-6:**

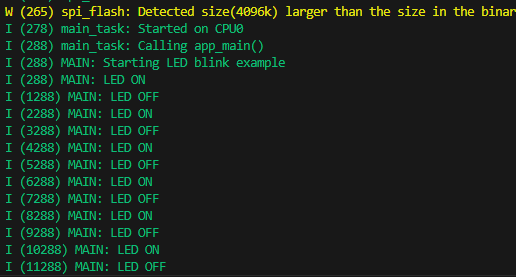


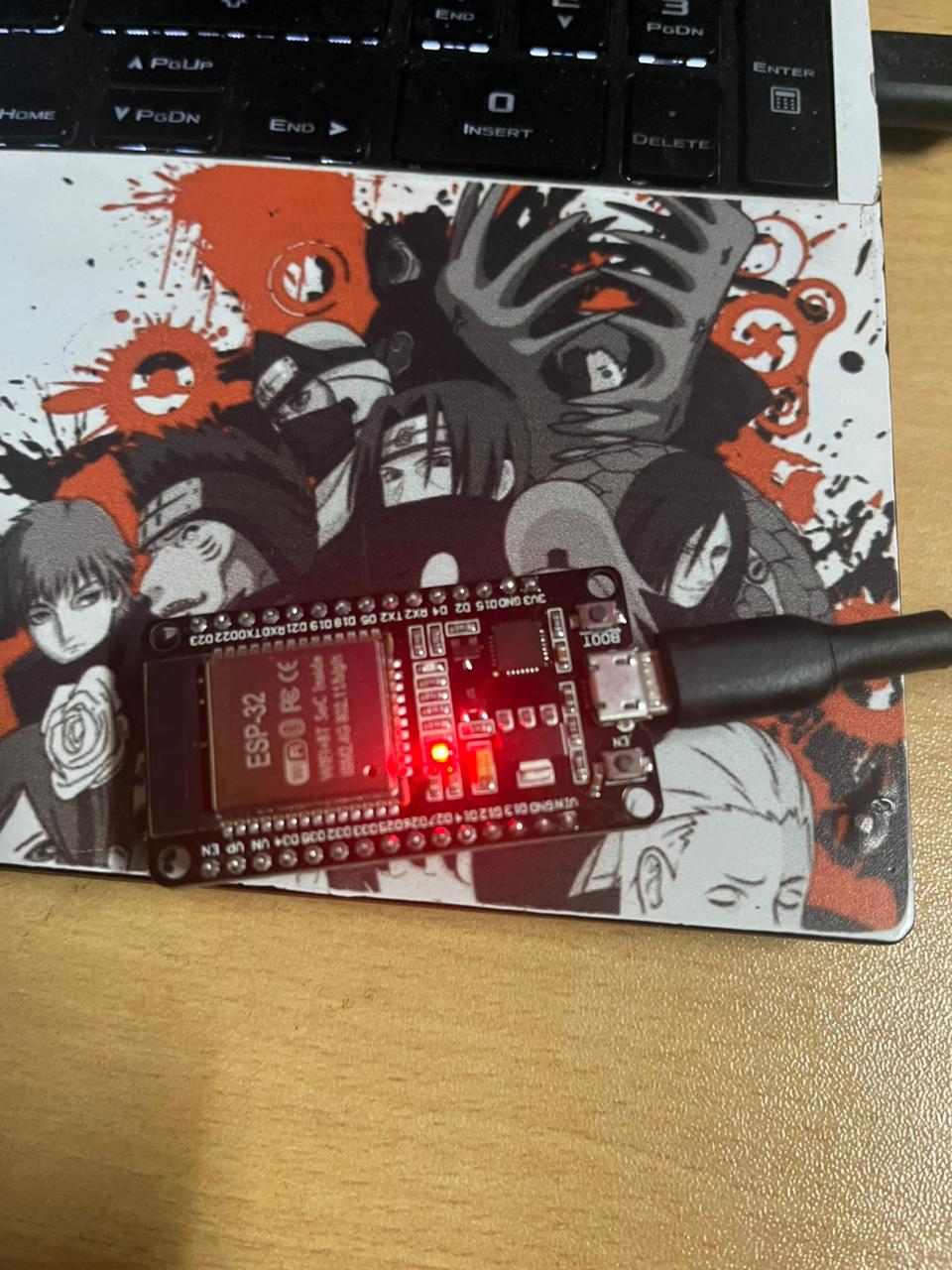
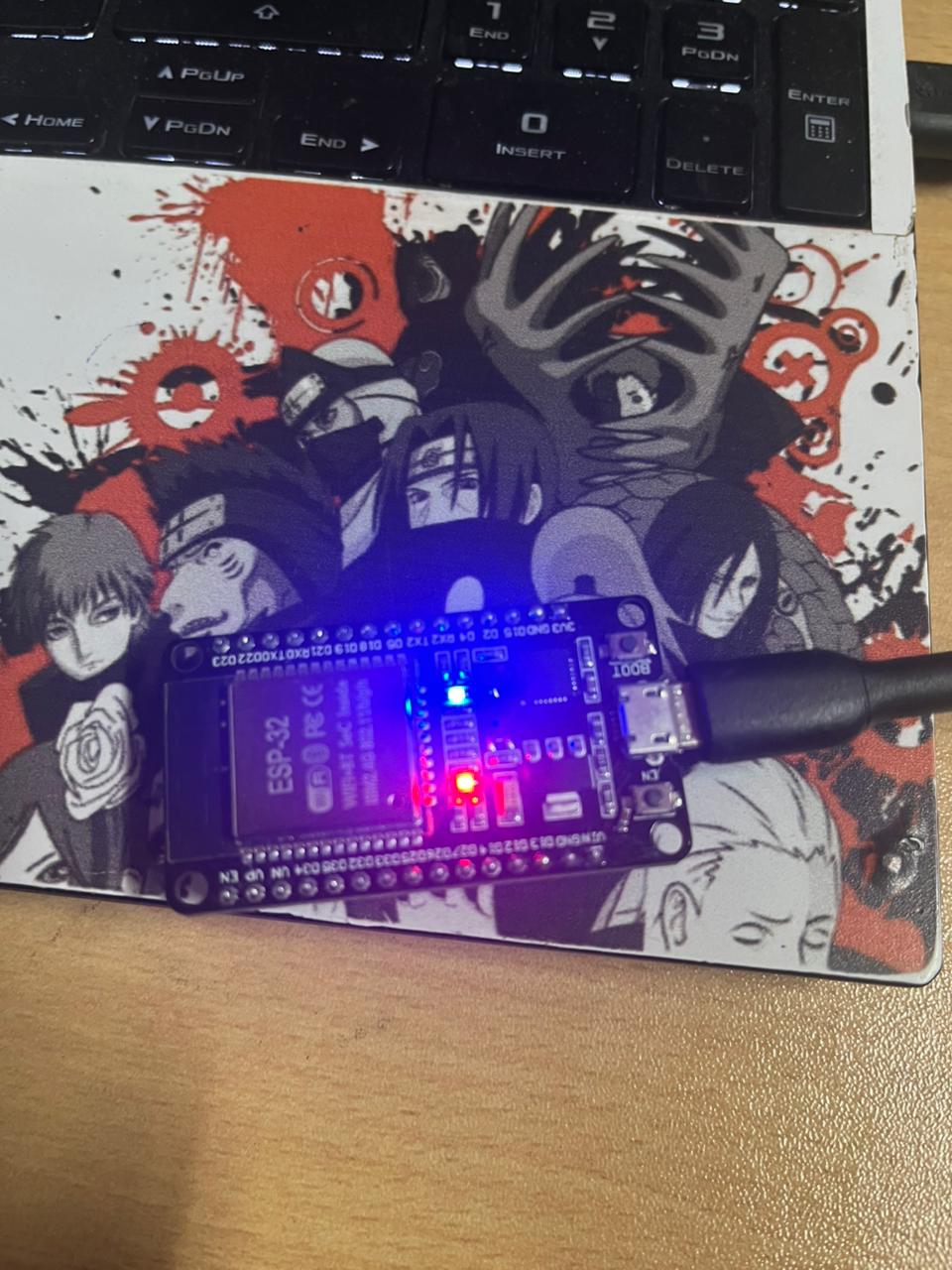


**Step-7:**



**Output:**





**Project-2: Police Siren**

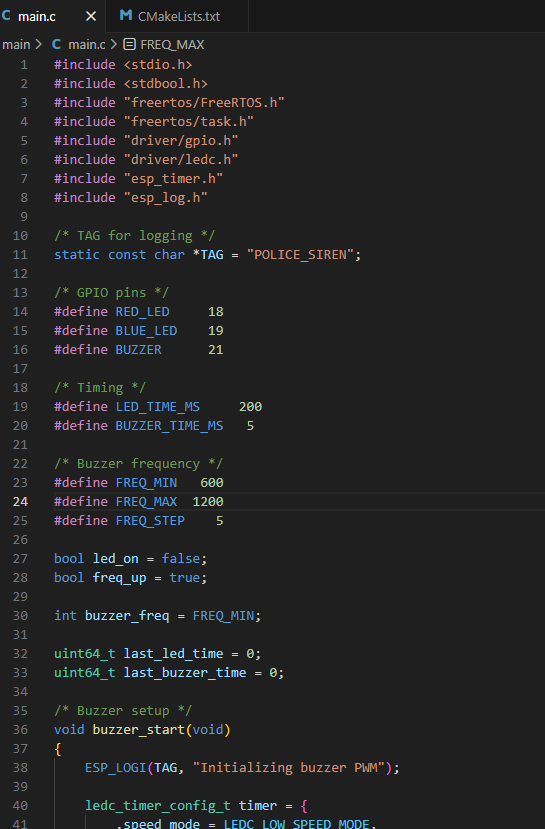
**Objective:** Simulates a **police vehicle warning system** using two LEDs and a buzzer.

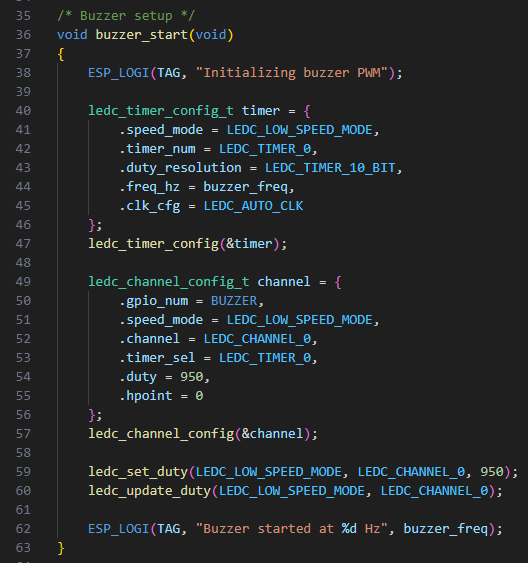
**Learning outcomes:** Using millis() for task scheduling. Running multiple time-based actions in a single loop. Writing clean, readable logic without blocking delays

**Circuit connections:**

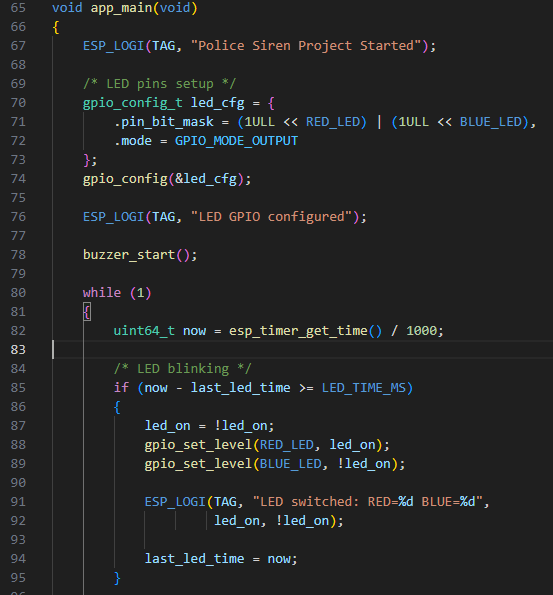
| **Component** | **ESP32 Pin** | **Other Connection** |
| --- | --- | --- |
| Red LED | GPIO 18 | LED cathode → 220Ω → GND |
| Blue LED | GPIO 19 | LED cathode → 220Ω → GND |
| Buzzer (+) | GPIO 21 | Buzzer (–) → GND |
| GND | GND pin | Common ground |

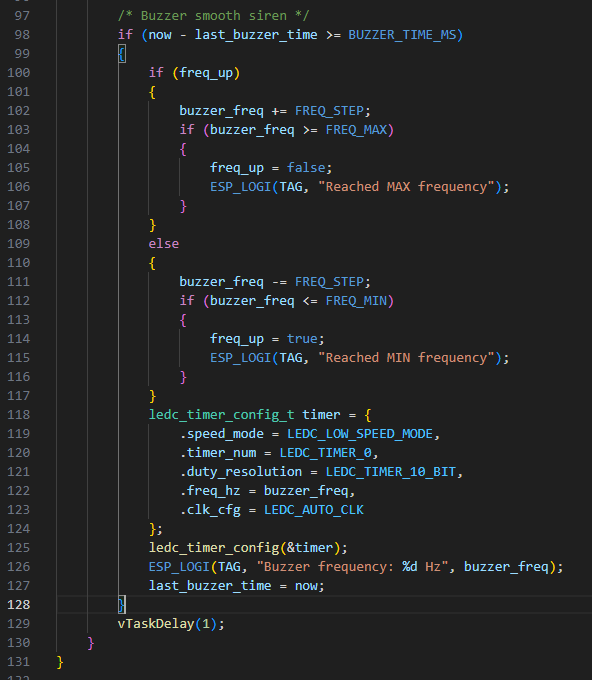
**Code:**











**Output:**





**Project-3: Digital Melody Player**

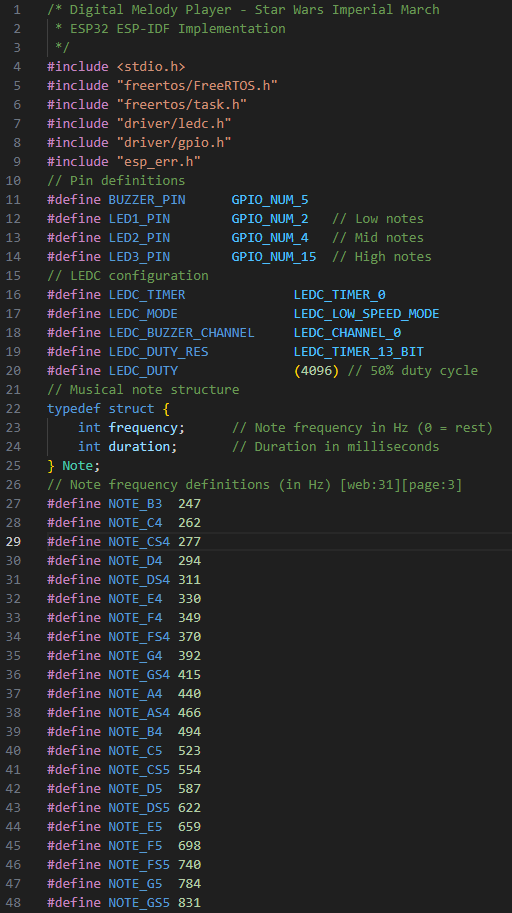
**Objective:** The **Digital Melody Player** is an embedded application where the ESP32 automatically plays a **predefined melody** using a buzzer, while LEDs visually indicate the **current note or beat** being played.

**Learning Outcomes:** The project demonstrates how an ESP32 can play predefined melodies using structured data and timing control while providing synchronized visual feedback through LEDs.

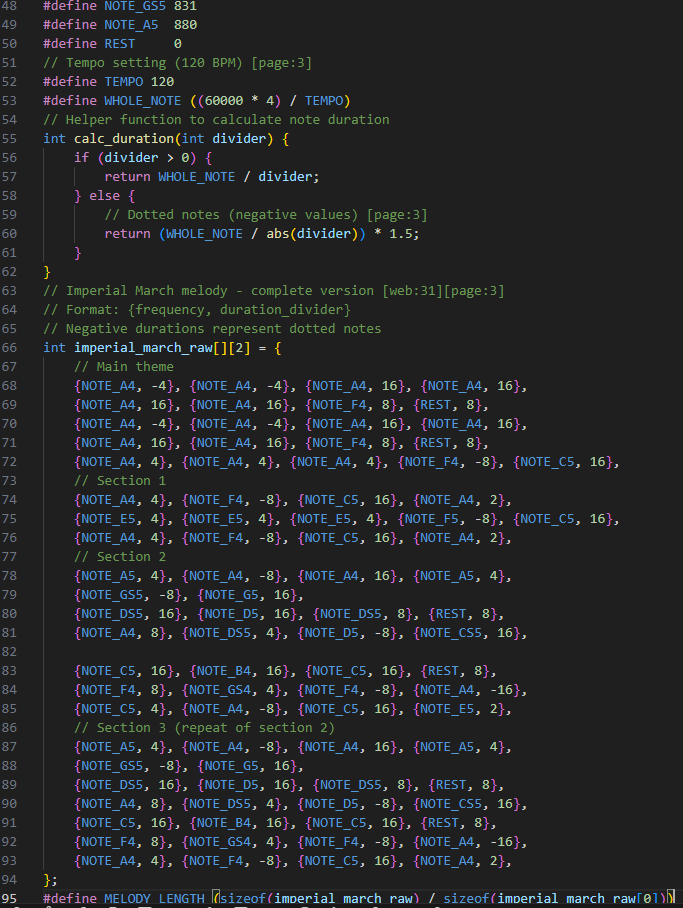
**Circuit connections:**

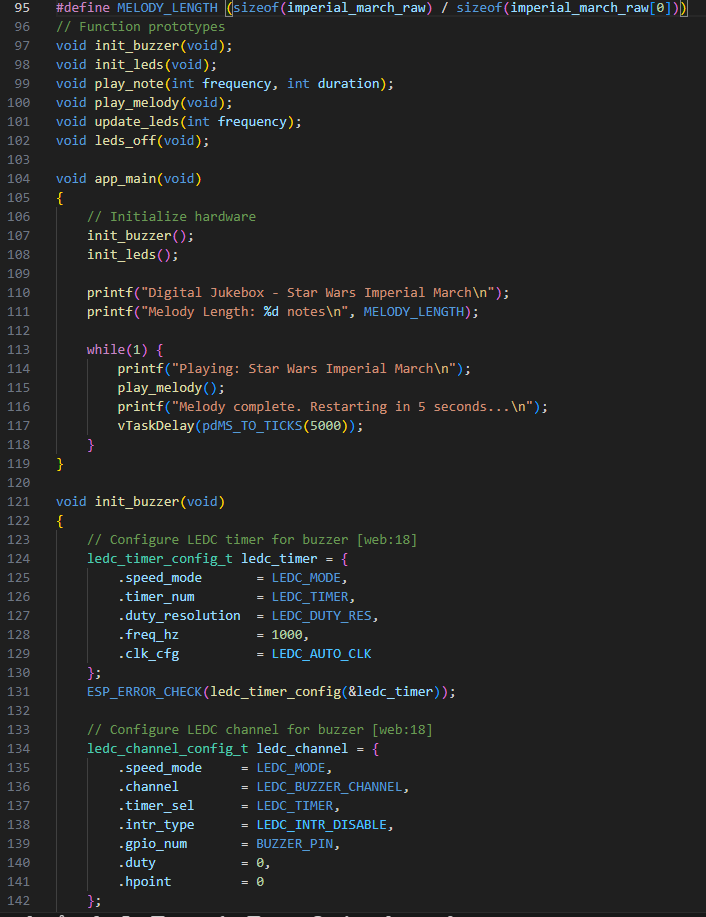
| **Component** | **ESP32 GPIO Pin** | **Other Connection** | **Purpose** |
| --- | --- | --- | --- |
| 🔊 **Passive Buzzer (+)** | **GPIO 5** | Buzzer (–) → GND | Melody / sound output |
| 🔴 **LED 1 (Low notes)** | **GPIO 4** | Cathode → 220Ω → GND | Indicates low-frequency notes |
| 🟡 **LED 2 (Mid notes)** | **GPIO 15** | Cathode → 220Ω → GND | Indicates mid-frequency notes |
| 🔵 **LED 3 (High notes)** | **GPIO 2** | Cathode → 220Ω → GND | Indicates high-frequency notes |
| ⏚ **Ground** | GND | Common ground | Mandatory for all |

**Code:**

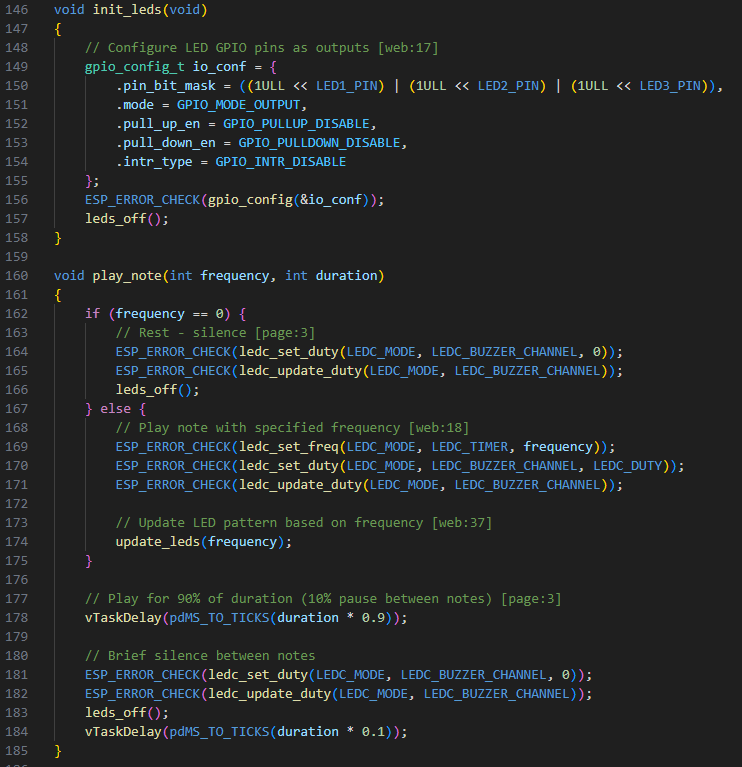


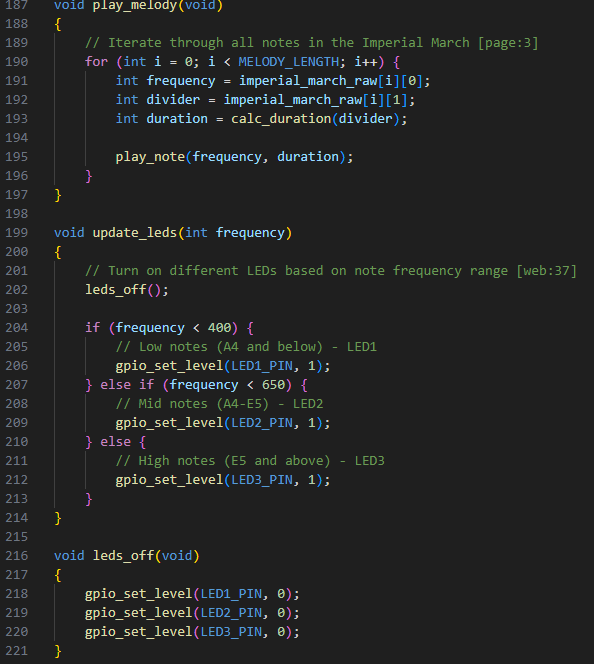




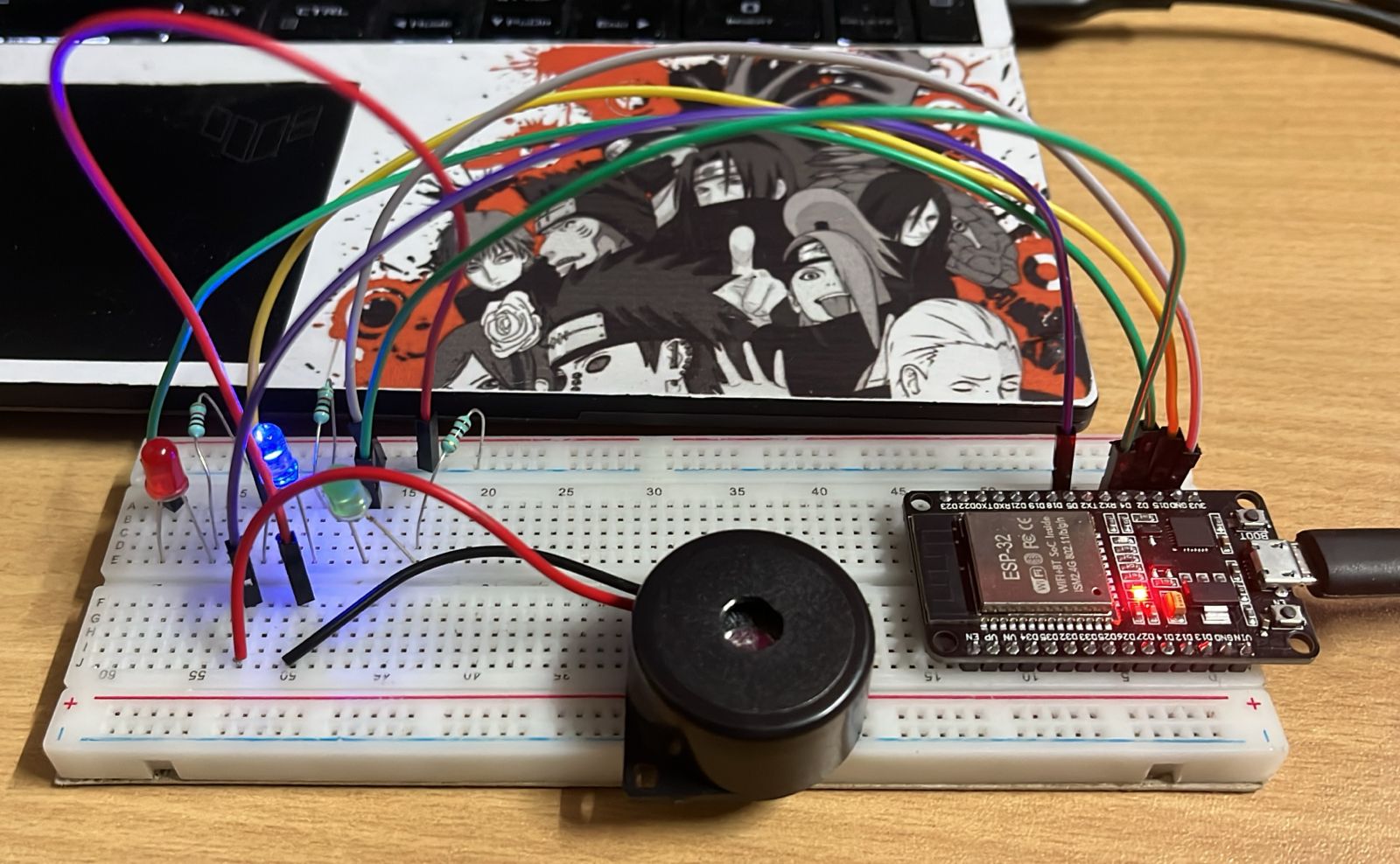
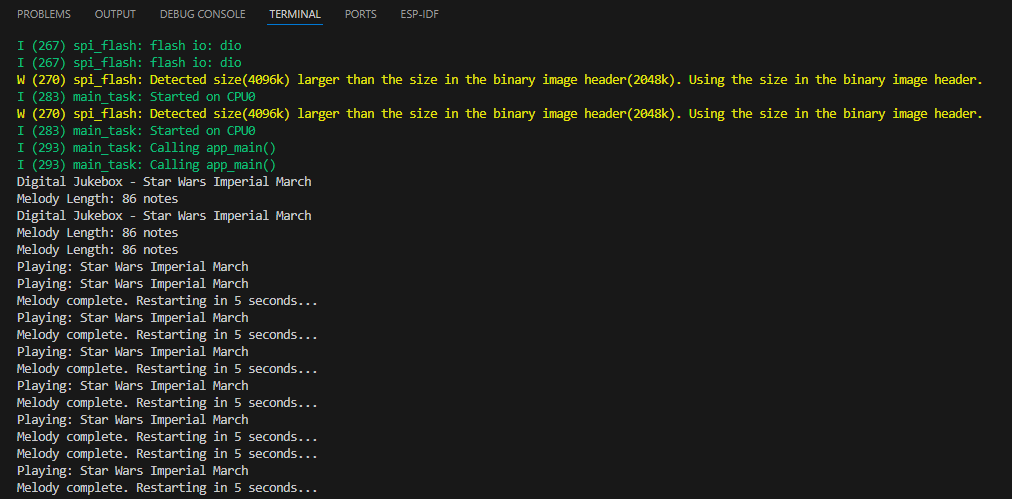








**Output:**



**Project-4: SOS Morse Code Beacon**

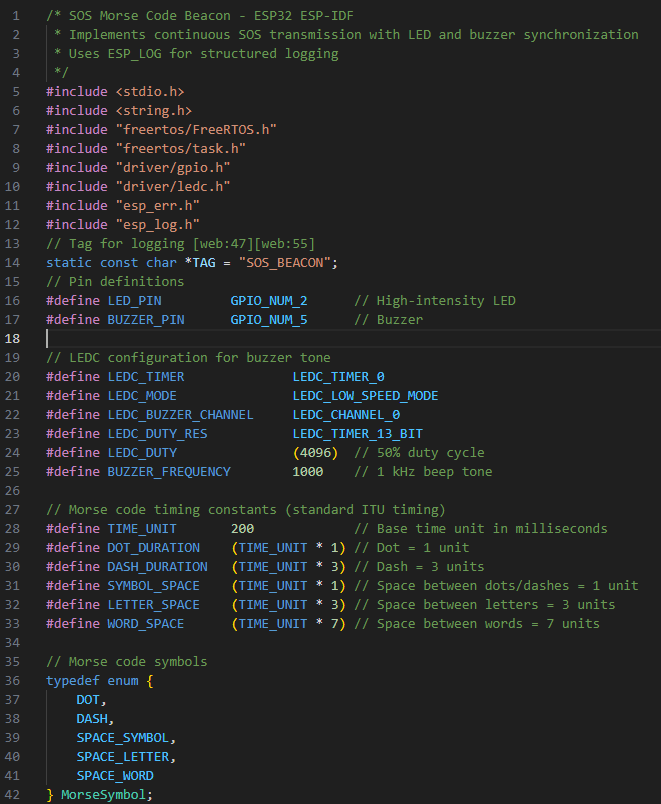
### ****Objective:**** To design an ESP32-based SOS Morse Code beacon that continuously transmits the pattern **(... --- ...)** using synchronized LED flashes and buzzer tones.

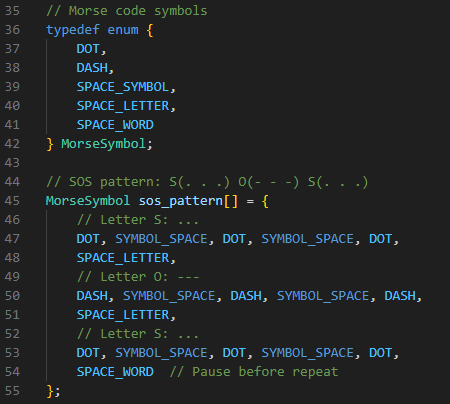
### ****Learning Outcome:**** The project enables understanding of **time-based signaling, Morse code encoding, and synchronized control of LED and buzzer outputs** using embedded C on ESP32.

**Circuit connection:**

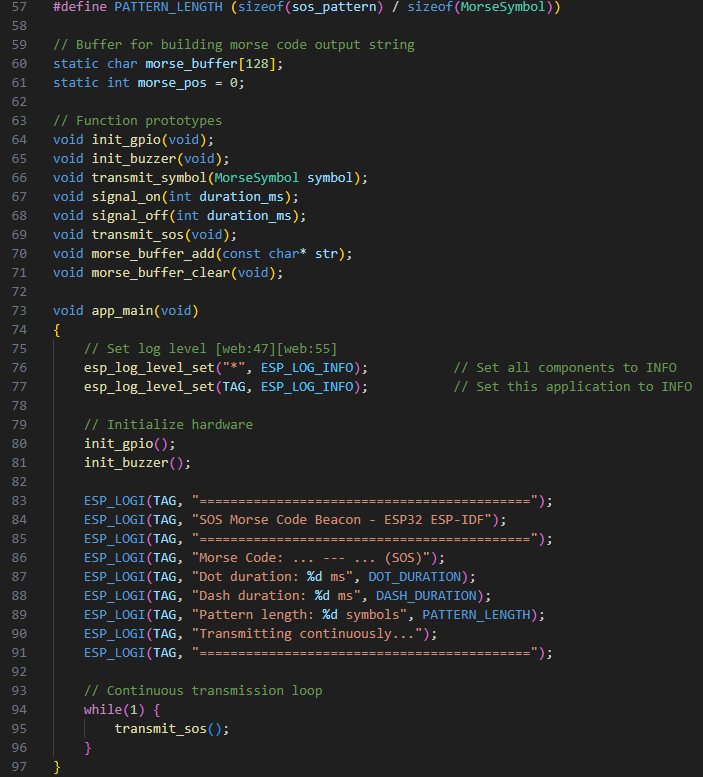
| **Component** | **ESP32 GPIO** | **Other Connection** |
| --- | --- | --- |
| LED (+) | GPIO 18 | LED – → 220Ω → GND |
| Buzzer (+) | GPIO 25 | Buzzer – → GND |
| GND | GND | Common ground |

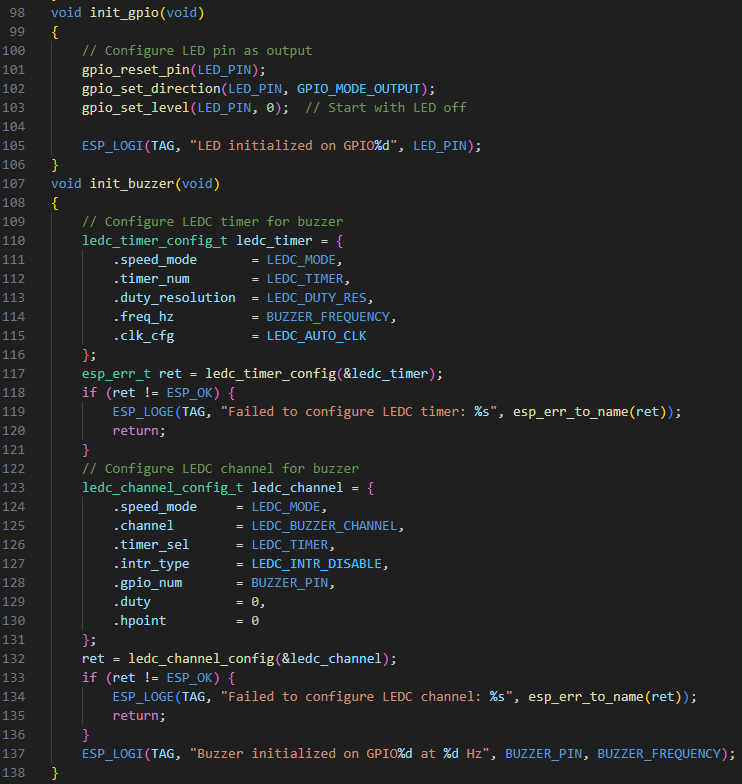
**Code:**

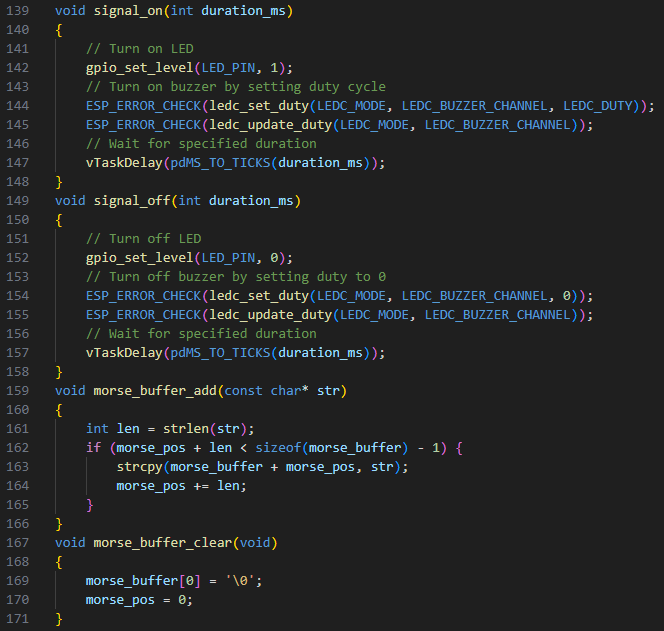




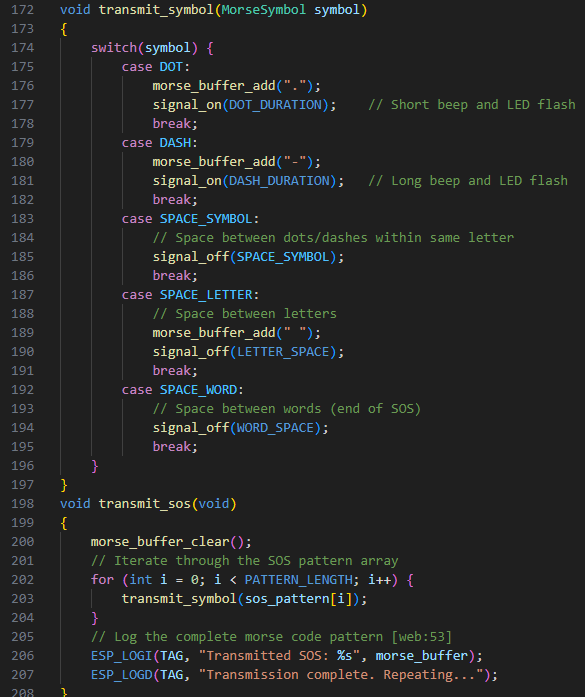




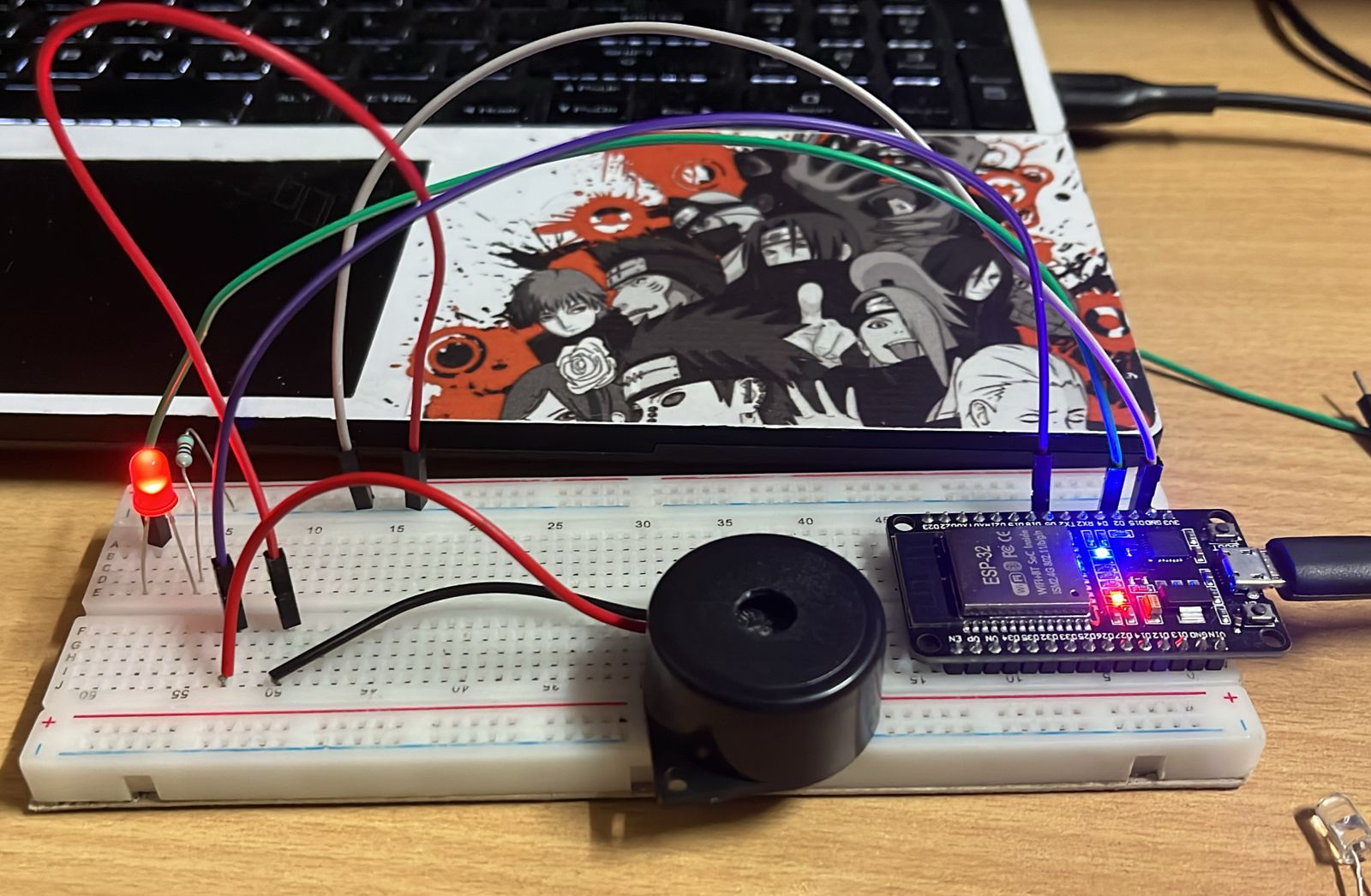


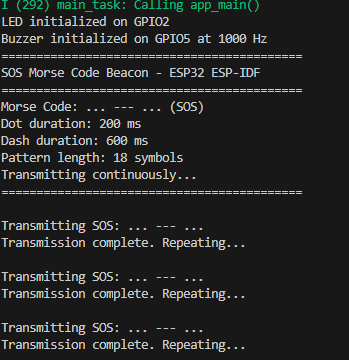






**Output:**







**Project-5: Ticking Time Bomb**

### ****Objective:**** To design an ESP32-based countdown timer that visually represents remaining time using LEDs and produces accelerating buzzer ticks followed by an explosion effect.

### L****earning Outcome:**** The project helps in understanding **array-based LED control, phase-based timing logic, variable delays, and synchronized audio-visual effects** using ESP32 and ESP-IDF.

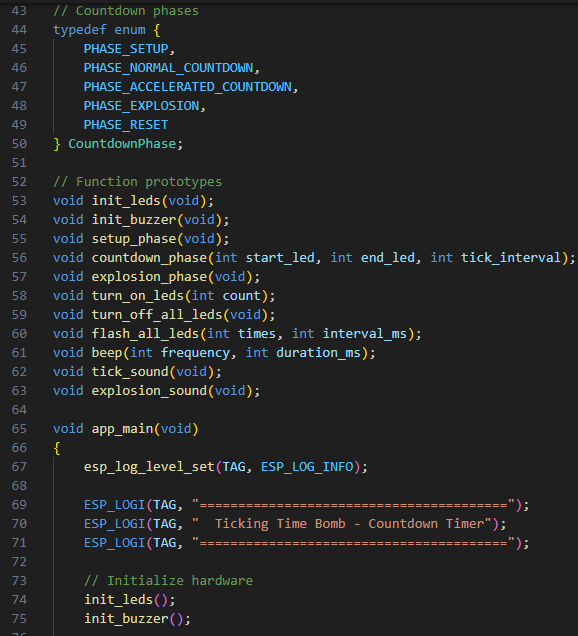
**Circuit connections:**

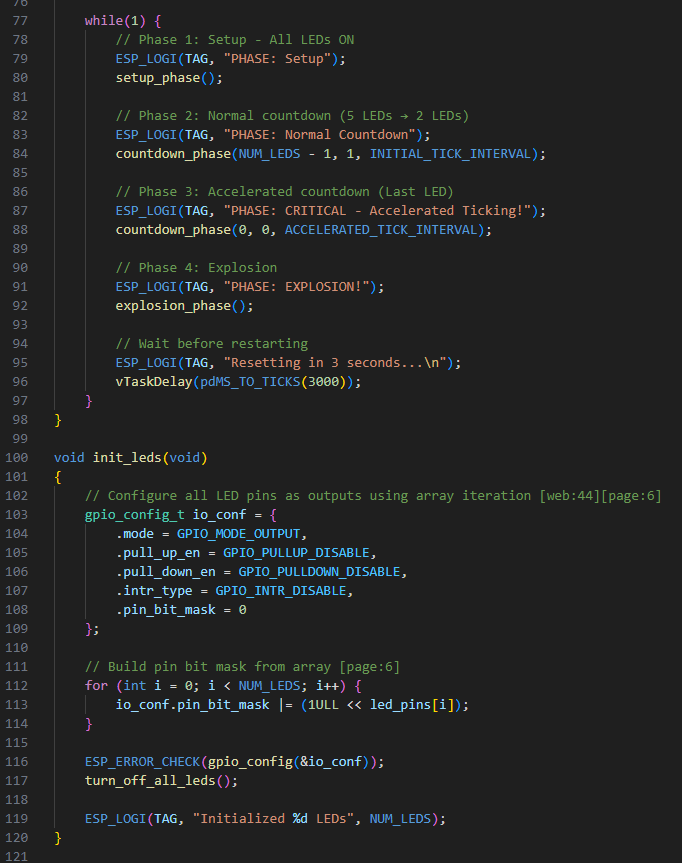
| **Component** | **ESP32 GPIO Pin** | **Other Connection** | **Description** |
| --- | --- | --- | --- |
| LED 1 | GPIO 2 | Cathode → 220Ω → GND | Countdown LED (1st) |
| LED 2 | GPIO 4 | Cathode → 220Ω → GND | Countdown LED (2nd) |
| LED 3 | GPIO 15 | Cathode → 220Ω → GND | Countdown LED (3rd) |
| LED 4 | GPIO 18 | Cathode → 220Ω → GND | Countdown LED (4th) |
| LED 5 | GPIO 19 | Cathode → 220Ω → GND | Countdown LED (5th) |
| 🔊 Passive Buzzer (+) | GPIO 5 | Buzzer (–) → GND | Tick & explosion sound |
| Ground | GND | Common ground | Mandatory |

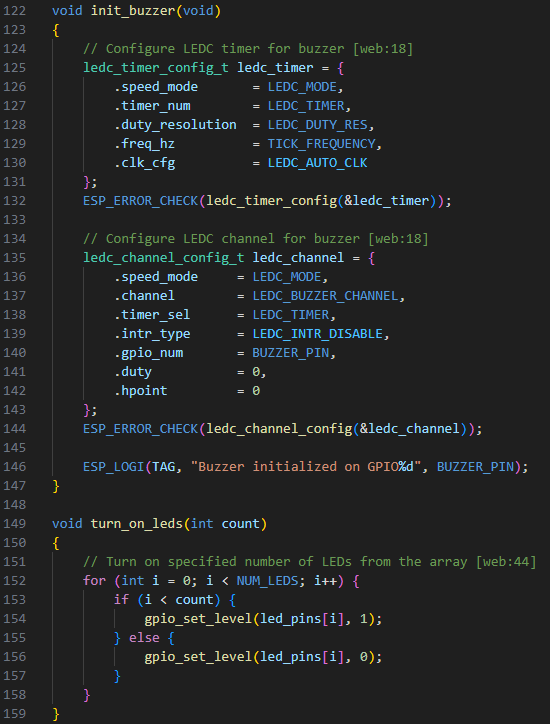
**Code:**





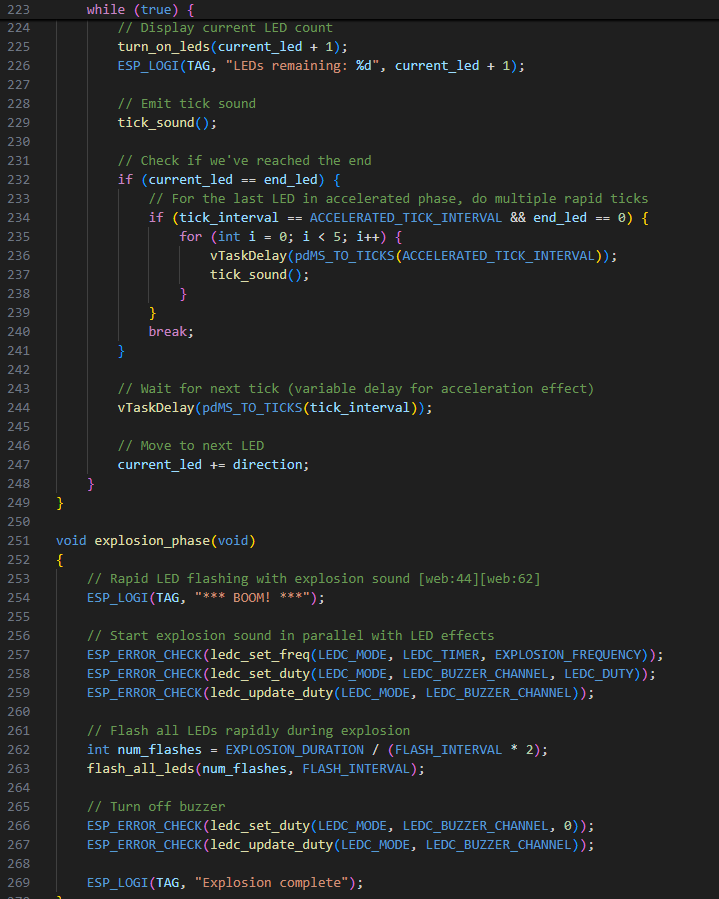




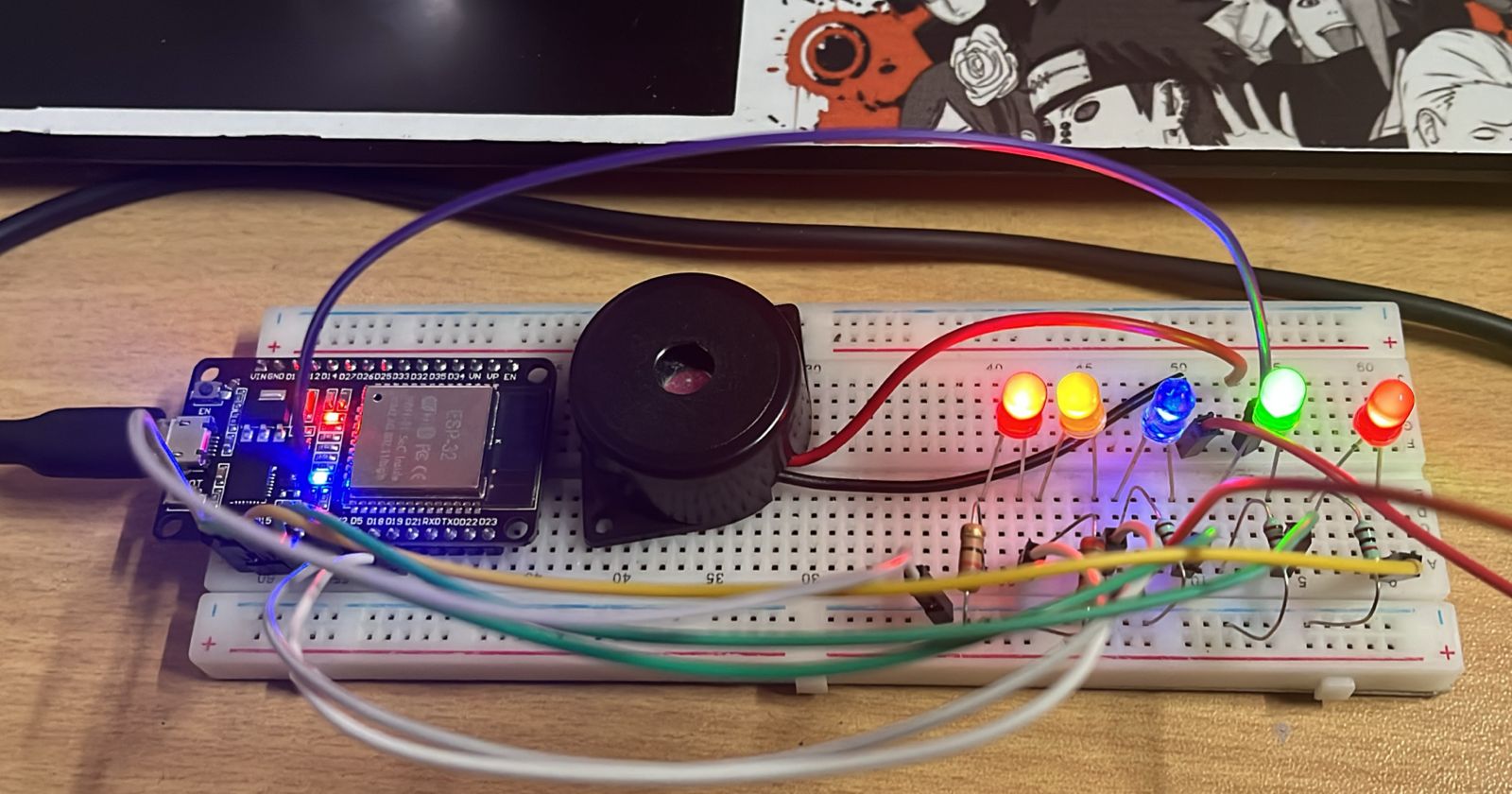
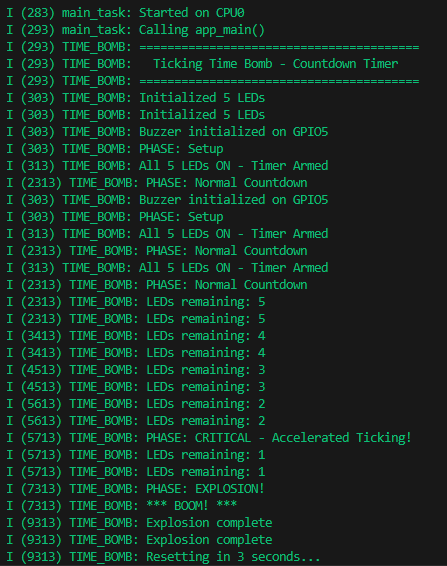








**Output:**



**Project-6: Automated Traffic Light System (Blind-Friendly Variant)**

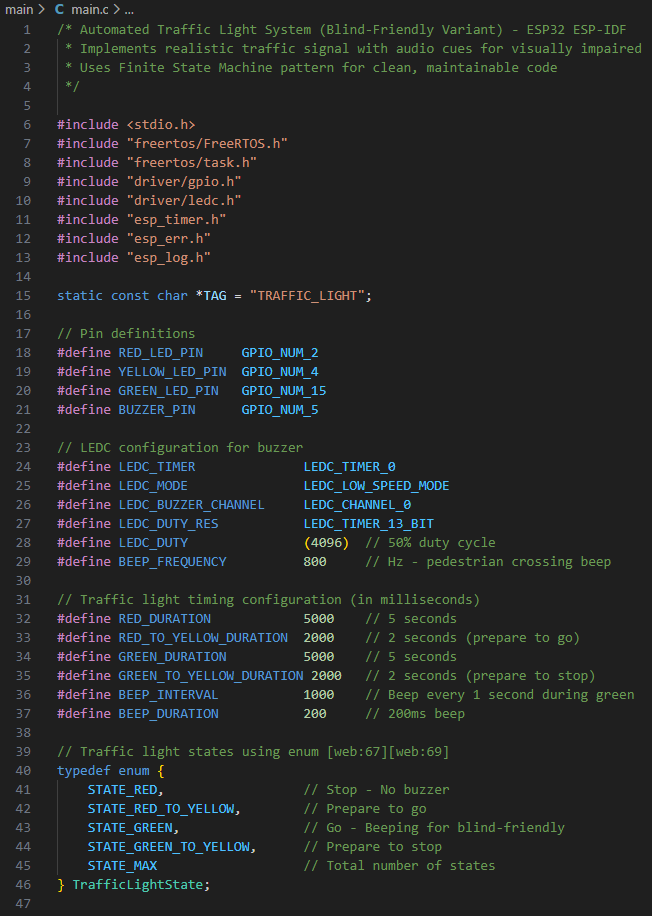
**Objective:** To design an ESP32-based automated traffic light system that controls red, yellow, and green signals with audio cues to assist visually impaired pedestrians.

**Learning outcomes:** The project enables understanding of finite state machine design, time-based control, and synchronized LED–buzzer operation in embedded systems using ESP32.

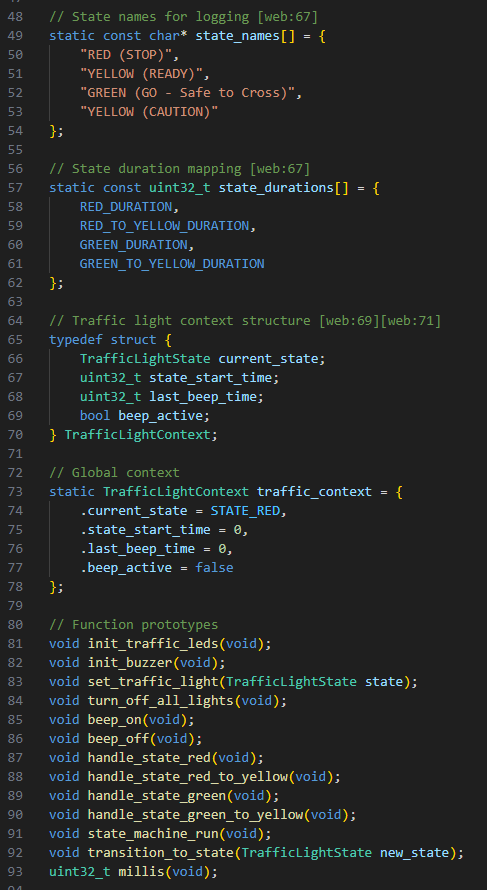
**Circuit Connections:**

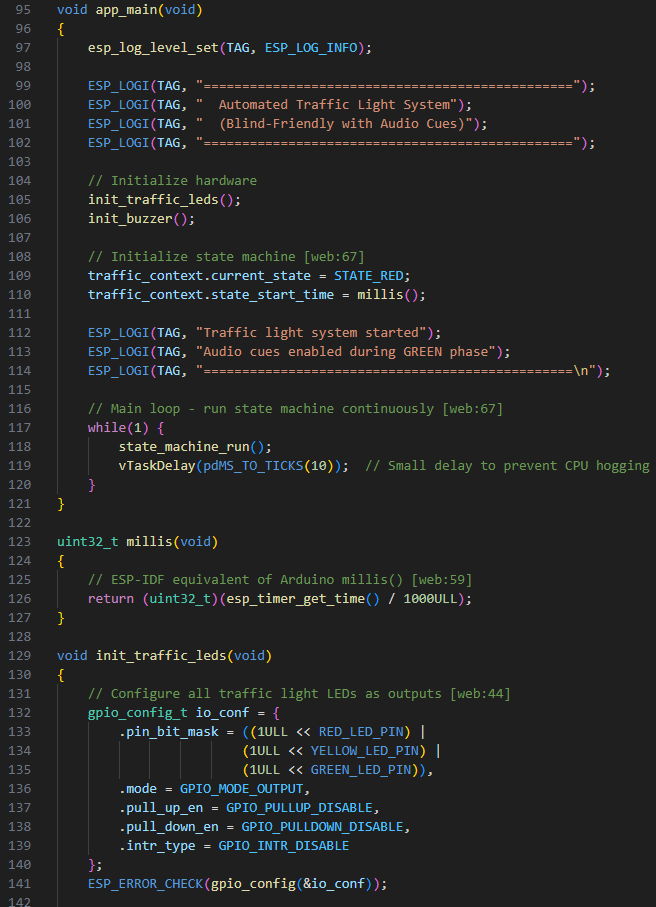
| **Component** | **ESP32 GPIO Pin** | **Other Connection** | **Description** |
| --- | --- | --- | --- |
| 🔴 **Red LED** | GPIO **2** | Cathode → 220Ω → GND | STOP signal |
| 🟡 **Yellow LED** | GPIO **4** | Cathode → 220Ω → GND | READY / CAUTION |
| 🟢 **Green LED** | GPIO **15** | Cathode → 220Ω → GND | GO signal |
| 🔊 **Passive Buzzer (+)** | GPIO **5** | Buzzer (–) → GND | Audio cue for blind pedestrians |
| ⏚ **Ground** | GND | Common ground | Mandatory |

**Code:**

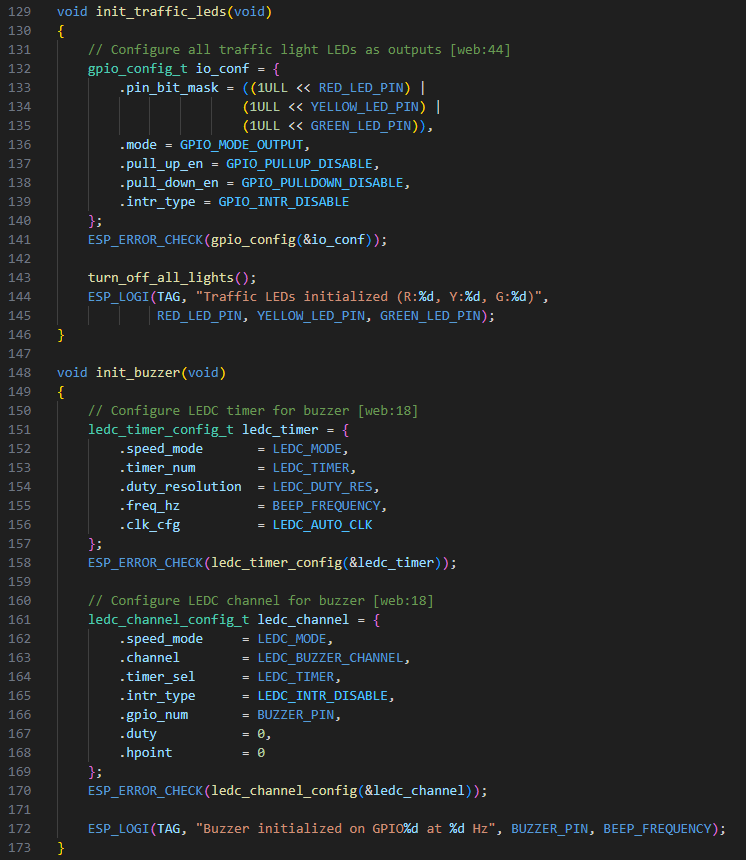


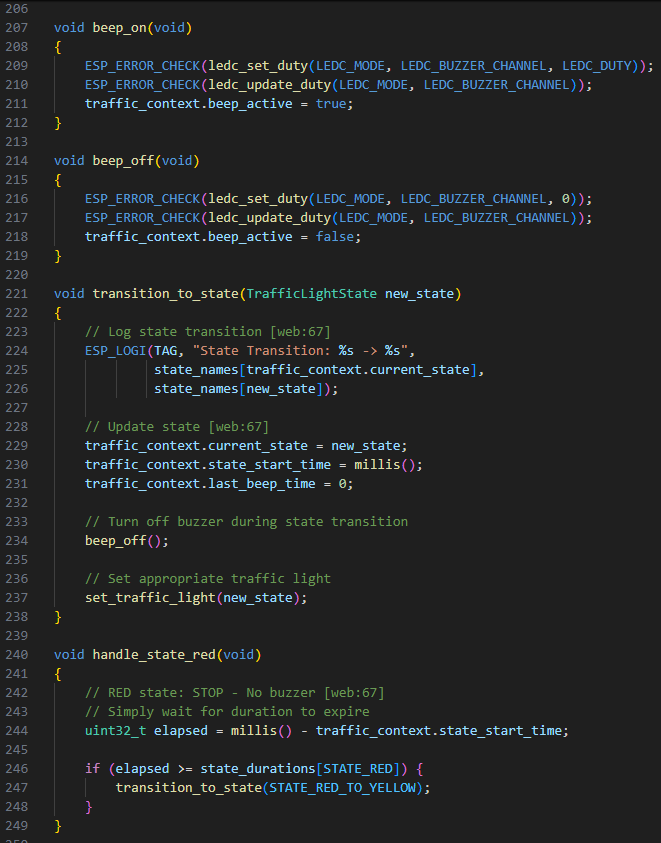


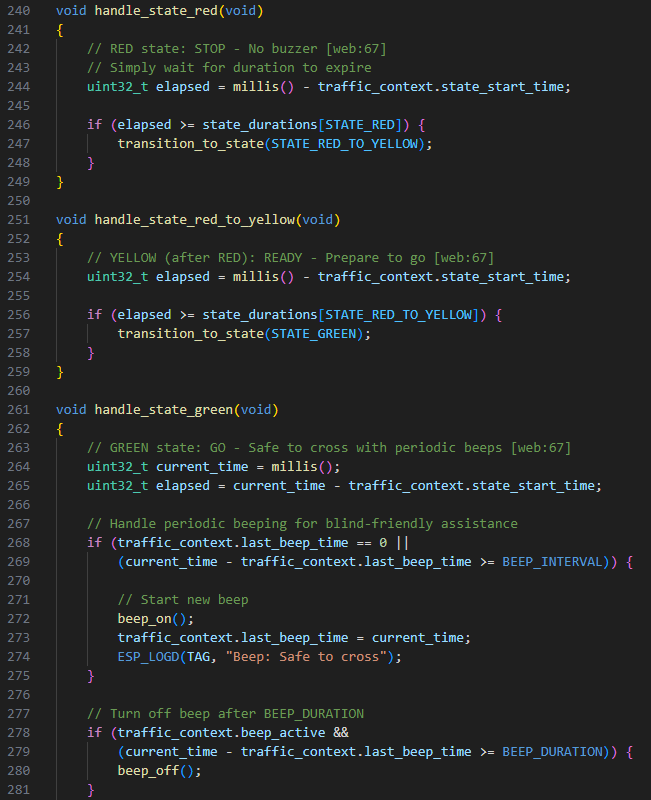




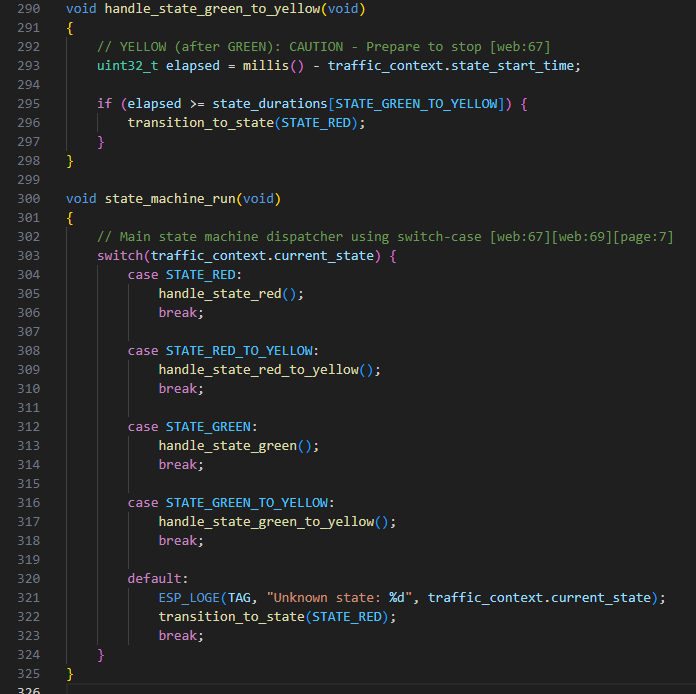












**Output:**

