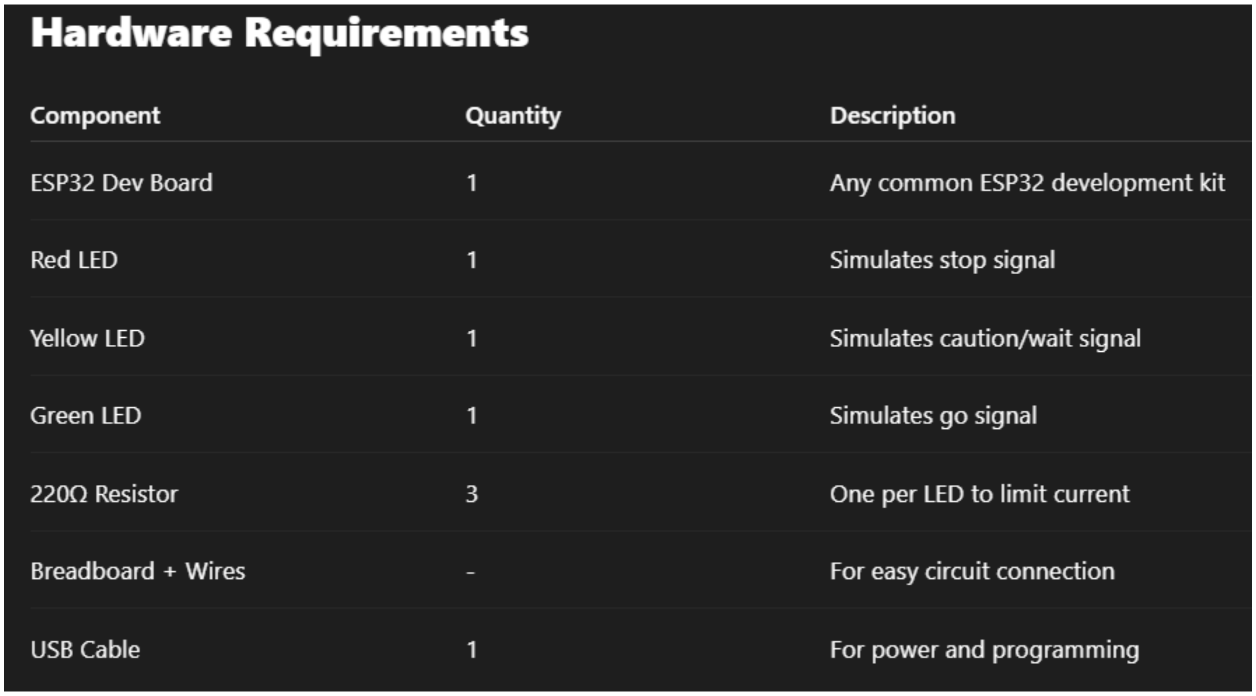
# ESP32 Traffic Light Simulation Using FreeRTOS (GPIO Control)

## Overview

This document explains a simple traffic light simulation using an ESP32 development board. The code controls three LEDs (red, yellow, and green) connected to different GPIO pins and toggles them in sequence using FreeRTOS task delays.

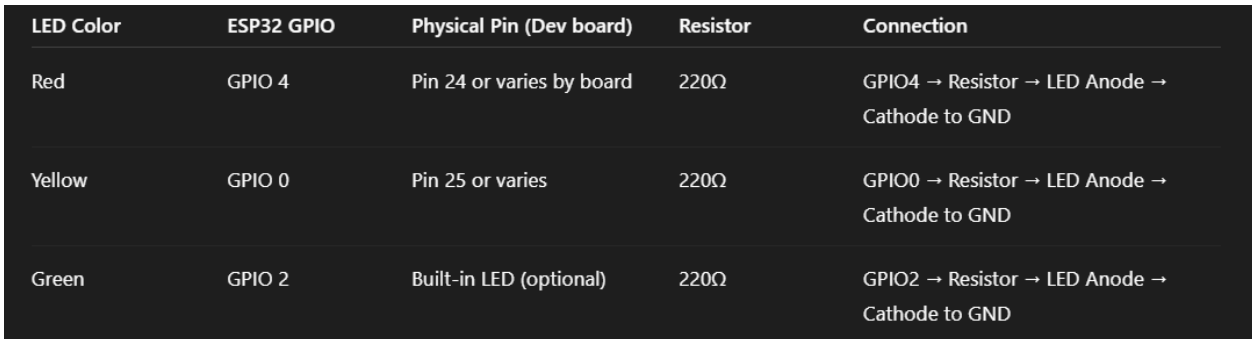


## Hardware Requirements

| Component | Quantity | Description |
|---|---|---|
| ESP32 Dev Board | 1 | Any common ESP32 development kit |
| Red LED | 1 | Simulates stop signal |
| Yellow LED | 1 | Simulates caution/wait signal |
| Green LED | 1 | Simulates go signal |
| 220Ω Resistor | 3 | One per LED to limit current |
| Breadboard + Wires | - | For easy circuit connection |
| USB Cable | 1 | For power and programming |

## Pin Configuration

Connect the LEDs as follows:



| LED Color | ESP32 GPIO | Physical Pin (Dev board) | Resistor | Connection |
|---|---|---|---|---|
| Red | GPIO 4 | Pin 24 or varies by board | 220Ω | GPIO4 → Resistor → LED Anode → Cathode to GND |
| Yellow | GPIO 0 | Pin 25 or varies | 220Ω | GPIO0 → Resistor → LED Anode → Cathode to GND |
| Green | GPIO 2 | Built-in LED (optional) | 220Ω | GPIO2 → Resistor → LED Anode → Cathode to GND |

⚠️ **Note:** Some ESP32 boards use GPIO 0 and GPIO 2 for boot functions. Avoid pressing the BOOT button during operation if using these pins.

**CODE**

```
#include<stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#define red GPIO_NUM_4

#define yellow GPIO_NUM_0

#define green GPIO_NUM_2



void app_main() {

    gpio_set_direction(red,GPIO_MODE_OUTPUT);

    gpio_set_direction(yellow,GPIO_MODE_OUTPUT);

    gpio_set_direction(green,GPIO_MODE_OUTPUT);



    while(1)

    {

        gpio_set_level(red,1);

        vTaskDelay(pdMS_TO_TICKS(500));

        gpio_set_level(red,0);

        vTaskDelay(pdMS_TO_TICKS(500));
        gpio_set_level(yellow,1);

        vTaskDelay(pdMS_TO_TICKS(500));

        gpio_set_level(yellow,0);

        vTaskDelay(pdMS_TO_TICKS(500));



        gpio_set_level(green,1);

        vTaskDelay(pdMS_TO_TICKS(500));

        gpio_set_level(green,0);

        vTaskDelay(pdMS_TO_TICKS(500));

    }



}
```

**Code Explanation**

```
#include<stdio.h>
```

- Includes the standard I/O library, allowing functions like `printf` (not used here but often included for debugging/logging).

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
```

- Includes **FreeRTOS** headers from ESP-IDF, enabling task management and time delays:
  - `FreeRTOS.h` contains kernel definitions.
  - `task.h` provides task APIs such as `vTaskDelay`.

```
#include "driver/gpio.h"
```

- Imports the **GPIO driver** to control general-purpose I/O pins.
- This allows setting pin direction (input/output) and pin voltage levels (high/low).

```
#define red GPIO_NUM_4
#define yellow GPIO_NUM_0
#define green GPIO_NUM_2
```

- Defines friendly names for the GPIO pins.
- `GPIO_NUM_X` are ESP-IDF macros representing specific GPIO numbers:
  - `red` is mapped to GPIO4
  - `yellow` is mapped to GPIO0
  - `green` is mapped to GPIO2

```
void app_main() {
```

- The **entry point** for ESP32 applications in ESP-IDF.
- Called automatically after system initialization.
- Replaces the traditional `main()` function in embedded C.

```
    gpio_set_direction(red,GPIO_MODE_OUTPUT);
    gpio_set_direction(yellow,GPIO_MODE_OUTPUT);
    gpio_set_direction(green,GPIO_MODE_OUTPUT);
```

- Configures GPIO4, GPIO0, and GPIO2 as **output** pins.
- This is necessary to control LEDs, which require voltage to be applied (written) to the pin.

**Main Control Loop**

```
    while(1)
```

- Infinite loop to keep the program running continuously.
- LEDs are toggled in sequence to simulate traffic light behavior.

```
        gpio_set_level(red,1);
```

- Turns **ON the red LED** by setting GPIO4 to **HIGH** (3.3V).

```
        vTaskDelay(pdMS_TO_TICKS(500));
```

- Delays for **500 milliseconds** using FreeRTOS. LED remains ON during this time.

```
        gpio_set_level(red,0);
        vTaskDelay(pdMS_TO_TICKS(500));
```

- Turns **OFF the red LED** by setting GPIO4 to **LOW** (0V).
- Waits another 500 ms before switching to the next LED.

```
        gpio_set_level(yellow,1);
        vTaskDelay(pdMS_TO_TICKS(500));
        gpio_set_level(yellow,0);
        vTaskDelay(pdMS_TO_TICKS(500));
```

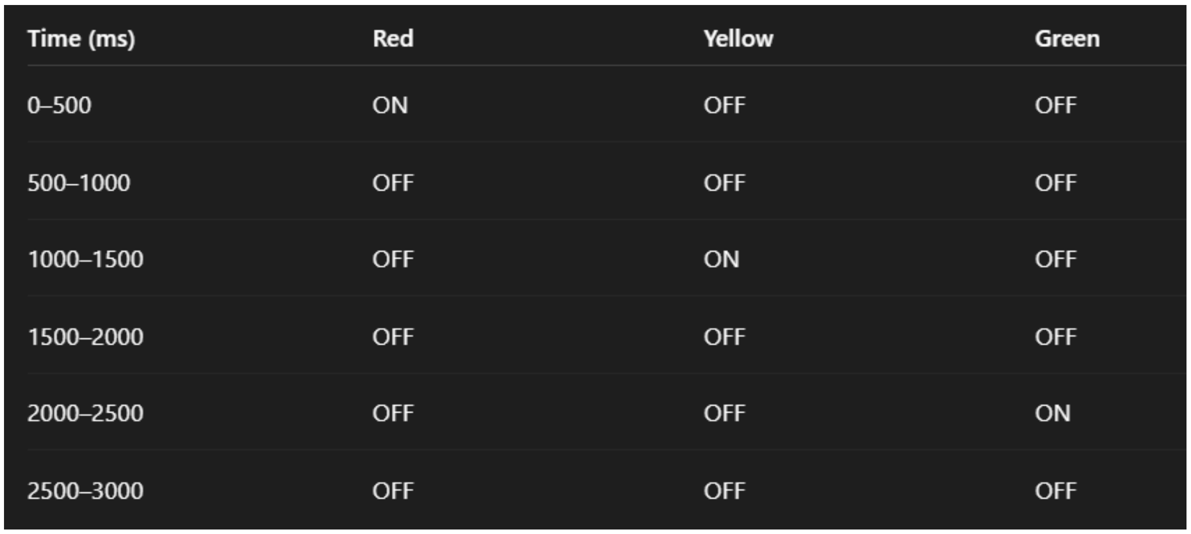
- Similar pattern:
    - **Yellow LED ON**
    - Delay 500ms
    - **Yellow LED OFF**
    - Delay 500ms

```
        gpio_set_level(green,1);
        vTaskDelay(pdMS_TO_TICKS(500));
        gpio_set_level(green,0);
        vTaskDelay(pdMS_TO_TICKS(500));
```

- Finally:
    - **Green LED ON**
    - Delay 500ms
    - **Green LED OFF**
    - Delay 500ms

**Sequence Summary**

The LEDs are toggled **sequentially**, creating a repeating pattern like a traffic signal:



| Time (ms) | Red | Yellow | Green |
|---|---|---|---|
| 0–500 | ON | OFF | OFF |
| 500–1000 | OFF | OFF | OFF |
| 1000–1500 | OFF | ON | OFF |
| 1500–2000 | OFF | OFF | OFF |
| 2000–2500 | OFF | OFF | ON |
| 2500–3000 | OFF | OFF | OFF |

**System Timing Diagram**

```
Time (ms):    0     500   1000  1500  2000  2500  3000 ...
Red LED:     ON     OFF
Yellow LED:        ON     OFF
Green LED:                 ON     OFF
```

Each LED is ON for 500ms, then OFF for 500ms before the next one activates.

## Conclusion

This ESP32 program demonstrates a basic traffic light simulation using FreeRTOS and GPIO control. Key takeaways:

- Demonstrates FreeRTOS task delay via `vTaskDelay()`.
- Teaches GPIO pin configuration and control.
- Useful as a learning exercise for embedded systems programming with ESP-IDF.