Brno University of Technology

Faculty of Information Technology

Microprocessors and Embedded Systems

RGB LED controller

Project documentation

Brno December 5, 2022 Maksim Tikhonov

1 Introduction

This project focuses on the implementation of remote LED controller. It allows you to control the animation (animation type and speed) that is played on three LEDs and one RGB LED via a web page. The project was implemented in both `ESP IDF` [1] (queues, tasks, timer, pin controls) and `Arduino` [2] (tcp communication, mDNS protocol) frameworks using `PlatformIO` extension [3] for `VSCode` IDE.

2 Circuit

Using all the elements provided for the projects, a circuit was constructed, which can be seen in the photo below.

Each of the three anodes of an RGB LED is connected to a pin responsible for the corresponding color, just as each anode of standard LEDs is connected to the pin responsible for particular led. All cathodes are grounded.

Pins	LEDs
GPIO_NUM_14	Red RGB LED color
GPIO_NUM_27	Blue RGB LED color
GPIO_NUM_16	Green RGB LED color
GPIO_NUM_17	Leftmost LED
GPIO_NUM_25	Middle LED
GPIO_NUM_26	Rightmost LED

Table 1: Used pins

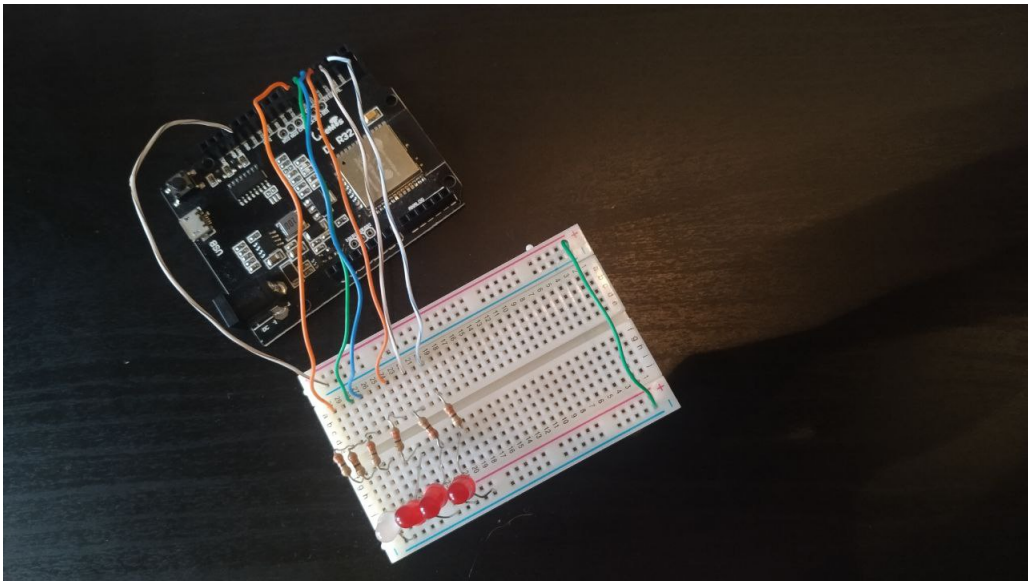

Figure 1: Photo of the circuit

3 Implementation

This section describes the project implementation details.

3.1 Initialization

The application initialization includes:

- Server initialization: connection to WiFi, start of mDNS responder, addition of tcp service to mDNS.
- GPIO configuration: sets pad as GPIO function for the required pins and sets them as output pins.
- Timer initialization: initialises timer with the `tick` function as a callback and 0.5 seconds period.
- Queue creation: creates a queue that is being used for inter-process communication.
- Semaphore creation: creates a semaphore that is used to block the client from stopping before the web page is sent.
- Tasks creation: creates two tasks, the first one, `client_task` is responsible for listening for any queries and sending the extracted information to the second one, `change_animation_state_task`, which handles animation changes (both speed and type) and sends a web page that corresponds to the current state of the application to a client.

3.2 Animations

Animations were implemented with a help of an internal **esp32** timer which is being initialized in function `set_and_start_tick_timer`. It has `tick` function as a callback, so each time the timer reaches a time limit, it calls an interrupt and calls it. This function controls what animation function will be called depending on `animation_type` global variable.

Each of four animation types (notated as **pump**, **worm**, **snake** and **wave**) has its own function that implements the animation by setting different pins depending on current value of internal counter, which is incremented at the end off each animation function's body.

3.3 Server communication

Client-Sever communication was implemented with a help of `Arduino` framework's libraries `ESPmDNS.h` and `WiFi.h`. The former was used to implement the support of **mDNS** protocol and the latter provides `WiFiServer` class. It's interface allows to easily control the flow of communication between client and server.

The processing of received messages was implemented with the help of two `FreeRTOS queues` – the first one listens to client, if some client was connected, parses the upcoming

request and sends the message to the the second one, which, based on received message, generates resulting web page and updates the current configuration.

Communication between the two tasks is also synchronized by a `semaphore`, which does not let the buffer to be sent to the client to overflow.

References

[1] Esp-idf documentation. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html.

[2] Arduino documentation. https://docs.espressif.com/projects/arduino-esp32/en/latest/getting_started.html.

[3] Platformio home page. https://platformio.org/.