# Lighting Audio Integration

Souad Omar, Firas Abueida, Sukeina Ammar, Heather Buzle

# Using threading for Voice Announcements

- **Lighting Detection and Configuration**
  - *light_2color_layer_timer_cb* detects brightness changes in the lighting system and sets the appropriate configuration for the UI and LEDs.
- **Event Group Synchronization**
  - Utilizes *xEventGroupSetBits* to signal specific brightness levels (e.g., 25%, 50%) to other tasks.
- **Concurrency and Mutex**
  - Ensures thread-safe access to shared lighting configuration using *xSemaphoreTake* and *xSemaphoreGive*.
- **Audio Feedback Integration**
  - A dedicated voice announcement task, created using *xTaskCreate*, waits for brightness events and triggers audio announcements
- **Efficient Event Handling**
  - Waits for brightness events using *xEventGroupWaitBits*, clearing bits after processing to synchronize tasks.

# Code Walkthrough

```c
//voice_announcement_task Function to wait for events
static void voice_announcement_task(void *param) {
    while (1) {
        // Wait for any sound event bits
        EventBits_t bits = xEventGroupWaitBits(
            lighting_event_group,
            (BIT0 | BIT1 | BIT2 | BIT3 | BIT4), // for each light level 0%, 25%, 50%, 75%, 100%
            pdTRUE, // Clear bits on exit
            pdFALSE, // Wait for any bit
            portMAX_DELAY
        );

        if (bits & BIT0) audio_handle_info(SOUND_TYPE_LIGHT_OFF);
        if (bits & BIT1) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_25);
        if (bits & BIT2) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_50);
        if (bits & BIT3) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_75);
        if (bits & BIT4) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_100);
    }
```

The task continuously waits for specific brightness events (e.g., 25%, 50%) using *xEventGroupWaitBits*, which synchronizes tasks and clears the event bits after processing

When a brightness event is detected, it triggers corresponding audio feedback by calling *audio_handle_info* with the appropriate sound type.

# Code Walkthrough 2

```
SemaphoreHandle_t light_config_mutex = NULL; //GLOBAL mutex for light con

if (xSemaphoreTake(light_config_mutex, portMAX_DELAY)) {
    switch (light_set_conf.light_pwm) {
        case 0:
            xEventGroupSetBits(lighting_event_group, BIT0);
            break;
        case 25:
            xEventGroupSetBits(lighting_event_group, BIT1);
            break;
        case 50:
            xEventGroupSetBits(lighting_event_group, BIT2);
            break;
        case 75:
            xEventGroupSetBits(lighting_event_group, BIT3);
            break;
        case 100:
            xEventGroupSetBits(lighting_event_group, BIT4);
            break;
        default:
            ESP_LOGW("Light", "Unsupported lighting level: %d", light_set_conf.light_pwm);
    }
    xSemaphoreGive(light_config_mutex);
}
```

## The file: ui_light_2color.c

- The *xSemaphoreTake* ensures exclusive access to the lighting mutex, preventing race conditions when multiple tasks attempt to modify the shared *light_set_conf* variable.
- Depending on the brightness level, specific bits in the event group are set using *xEventGroupSetBits*. This signals the voice announcement task to respond accordingly.
- After the updates and signaling are complete, the mutex is released using *xSemaphoreGive*, allowing other tasks to safely access the shared resource.

# Difficulties of this project:

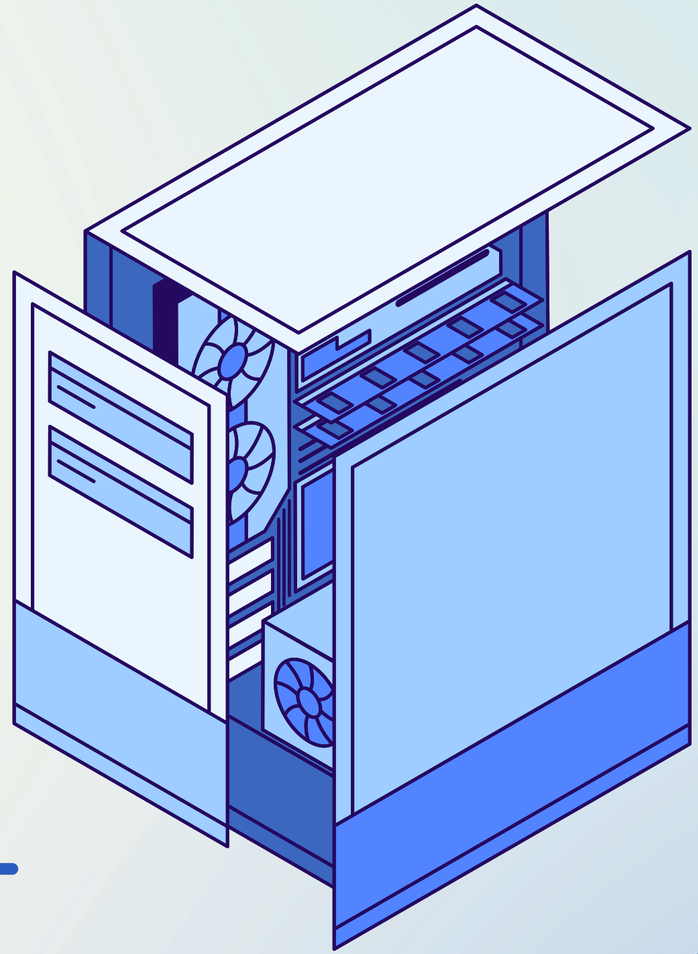
1. Debugging was complex due to unclear error messages.
2. Only one person could use the device at a time, limiting collaboration.
3. Understanding and integrating the provided code was challenging.
4. Memory constraints on the ESP32 required careful optimization.

# The lessons learned during this project:

1. We learned how to put the audio snippets into code by changing them into an mp3.
2. We learned how to use threading with voice implementation.
3. We learned how to use an ESP-32 device and how to make adjustments to it.

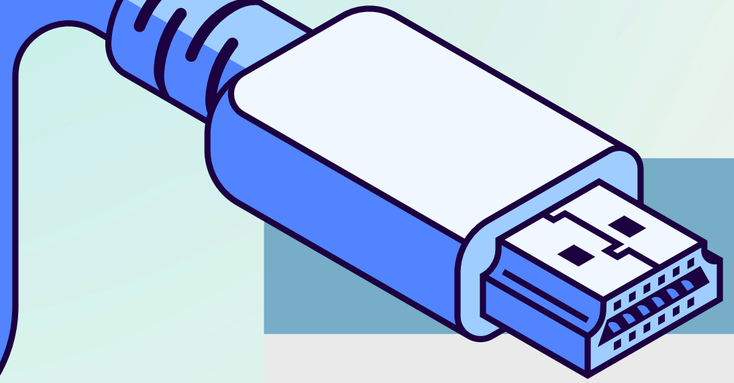
# Demo:

1. Plug the ESP32-C3 into the computer using a USB-C port.
2. If you haven't already done so, build & flash the knob-example code into the device using the terminal
3. Wait for the device to boot, and then twist the knob to the right to select the LCP/ lighting panel.
4. Assuming that the USBC port is on the right, twist the knob to the left to increase brightness, and twist the knob to the right to decrease brightness. You will hear an audio announcement when adjusting accordingly.
5. Press the knob to change the color/warmth of the light.



# Bonus Feature:

- Our ESP32 program uses FreeRTOS and LVGL to display a long sentence in smaller parts, updating the screen every 3 seconds without blocking other functions.
- A FreeRTOS task handles updates efficiently, supporting features like voice announcements.
- The ESP32 device does display the sentence: ''Hello CIS 450, this is our ESP 32 Threading Project, created by Souad, Sukeina, Heather, & Firas :).''



# Thank you!