CIS 450 Final Project

Audio Integration

Version 1.0

Sukeina Ammar, Souad Omar, Firas Abueida, and Heather Buzle

Table of Contents

1. System Architecture

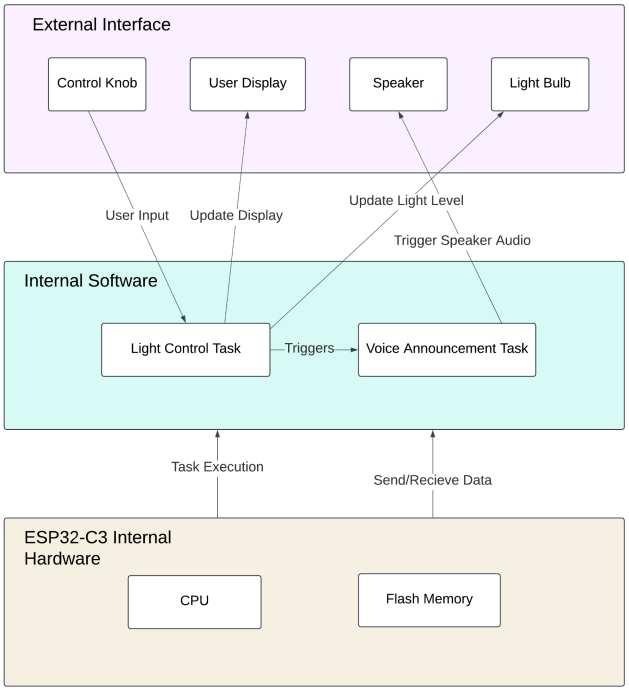
For this project, we used freeRTOS to implement a threading structure to allow the device to share resources and run tasks concurrently. We have four main aspects for our project architecture: the UI from the LCD screen, the Lighting control task, the voice announcement task, and the internal hardware of the ESP32-C3.

The User interface (UI) of this project is displayed on the LCD screen of the device and reacts based on the user's actions. The LCD display updates to show the current brightness level, ensuring that users receive immediate visual feedback as they adjust the knob.

The Lighting Control Task monitors the knob position to manage the LED brightness. This task runs in a loop, continuously reading the knob's position to determine the desired brightness level. Once the brightness level is updated, the new value is written to a shared variable. This task also signals the Voice Announcement Task using FreeRTOS's **xEventGroupSetBits** whenever a change in brightness occurs. This setup ensures that the Voice Announcement Task is promptly notified of any changes without delay.

The Voice Announcement Task, called by **xTaskCreate**, operates independently of the Lighting Control Task to provide real-time audio feedback. Upon receiving a signal using **xEventGroupWaitBits**, this task reads the current brightness level from the shared variable. It then constructs an announcement message and plays the corresponding audio via the connected speaker. This setup ensures that voice announcements are made immediately after any change in lighting.

Finally, the internal hardware of the ESP32-C3 is what allows us to have multiple tasks and functionalities. This microcontroller contains multiple features including knob panel, LEDs, and audio output components which allow it to do a variety of tasks. For this project, we focused on the LED, knob panel, and the audio output in order to have audio output. Overall, the design of both the software and hardware architecture allowed us to implement a concurrent control system, enabling the multitasking of audio and visual tasks.



2. Concurrency Control Explanation

Mutexes and semaphores are used to help make sure that when you are utilizing multiple threads, you don't have the threads completing their tasks at the same time. This helps prevent a race condition. Since they use a shared global memory you need to make sure that they can alternate between each thread. You would do this by using either mutexes or semaphores.

Mutexes help multiple threads complete their tasks not at the same time by using the lock and unlock functions. These functions are "lock()" and "unlock()". The important thing about mutexes is when one thread claims or calls the function lock(), then the other threads have to wait until the first thread calls unlock(). This is because only one thread can use the lock or unlock function at a time. They can't be used in multiple threads at the same time, even if they call it at the same time. The other threads have to wait until the first thread is done using those functions.

Semaphores help threads complete their task concurrently by using the functions wait(), and signal(). When a thread uses the wait() function, it decrements the integer, while the signal() function increments the integer. Semaphores can be used in multiple threads. It doesn't have to use both the wait() or signal() functions in one thread. One thread can call the wait() function and decrement, then a second thread can use the post() function to increment it. However, if the integer is already at 0, then, if one thread then calls for wait(), another thread would need to call post() to increment it to 1, so the other thread could then decrement back to 0.

In our project, we implemented mutexes and semaphores for the light configuration. When coding the light configuration part of the project, we made a mutex and then used “if statements” to make it equal to “NULL”. However, if the light configuration failed, it would provide a statement saying “Failed to create light_config_mutex”. The function xSemaphoreTake gives access to the mutex and then once its done completing the task, it then uses the function xSemaphoreGive. This function gives access to the other mutex variables for them to use. These functions, xEventGroupSetBits and xEventGroupWaitBits work in tandem with the semaphores functions.

3. User Guide

To use the lighting control panel, or LCP for short, follow these steps:

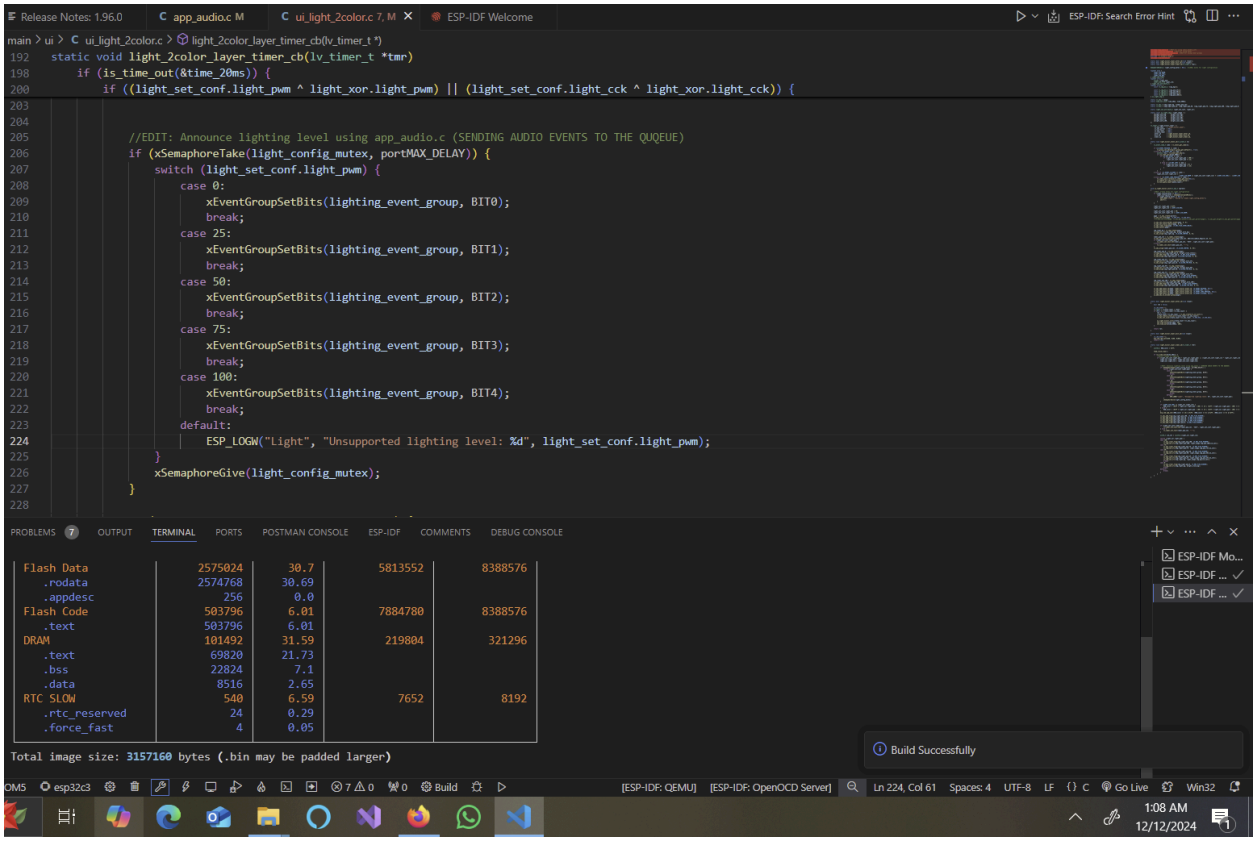
1) Plug the ESP32-C3 into the computer using a USB-C port.
2) If you haven’t already done so, build & flash the knob-example code into the device using the terminal
3) Wait for the device to boot, and then twist the knob to the right to select the LCP/ lighting panel.
4) Assuming that the USBC port is 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.

4. Bonus Feature Implementation

Our bonus feature includes our ESP32 program that demonstrates how to use FreeRTOS and the LVGL library to display a long sentence on the screen by splitting it into smaller, screen-friendly parts. The message is divided into an array of short strings, such as "Hello CIS 450," and "this is our ESP32," which are displayed sequentially, with each part shown for 3 seconds. The LVGL library manages the graphical display, while FreeRTOS handles multitasking, allowing the message to update without blocking other system functions. A FreeRTOS task (announcement_task) is created to iterate through the message parts, updating the on-screen label using the update_announcement_label() function and introducing delays between updates with vTaskDelay(). This non-blocking delay allows the FreeRTOS scheduler to handle other tasks concurrently. The program also initializes non-volatile storage (NVS) and audio playback, setting the stage for further enhancements like adding voice announcements. This approach highlights the efficiency of FreeRTOS in managing sequential updates for dynamic content on an ESP32.

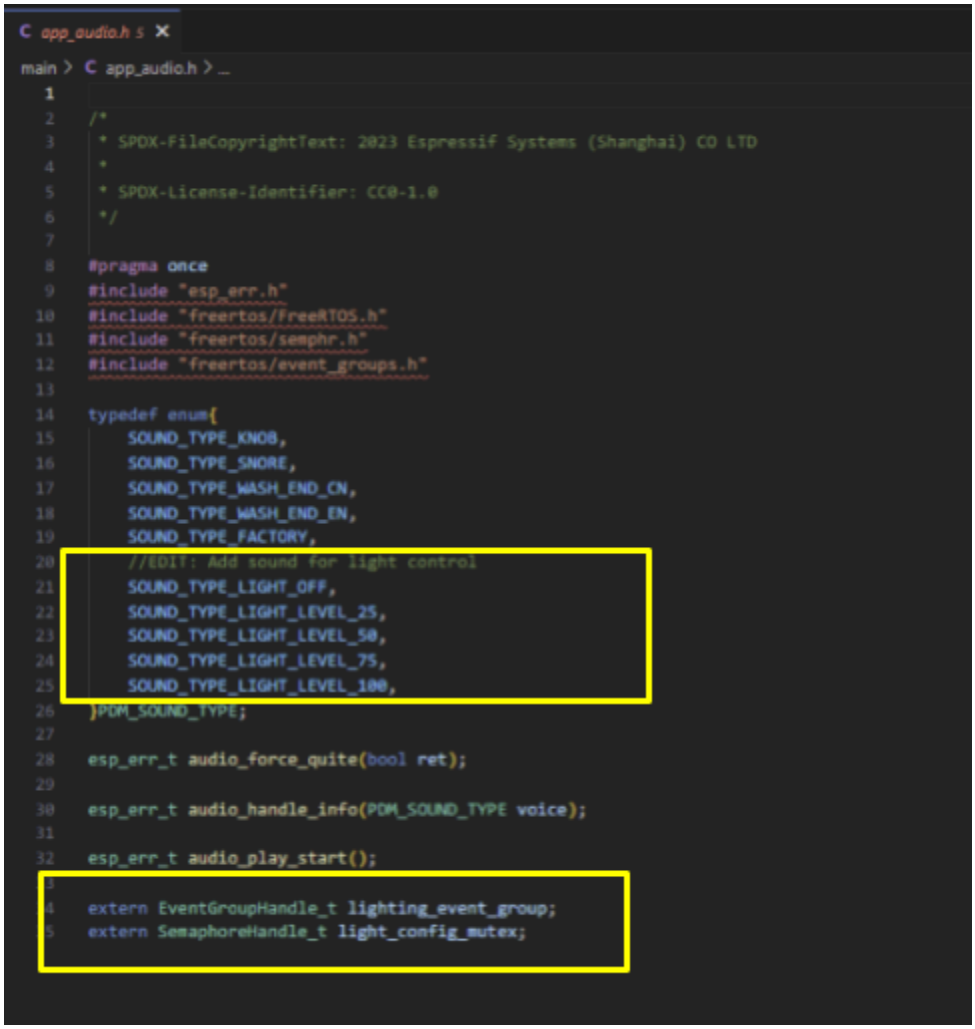
5. Proof of Compiliation

Program built successfully (using ESP-IDF VS code extension to build/run/flash)



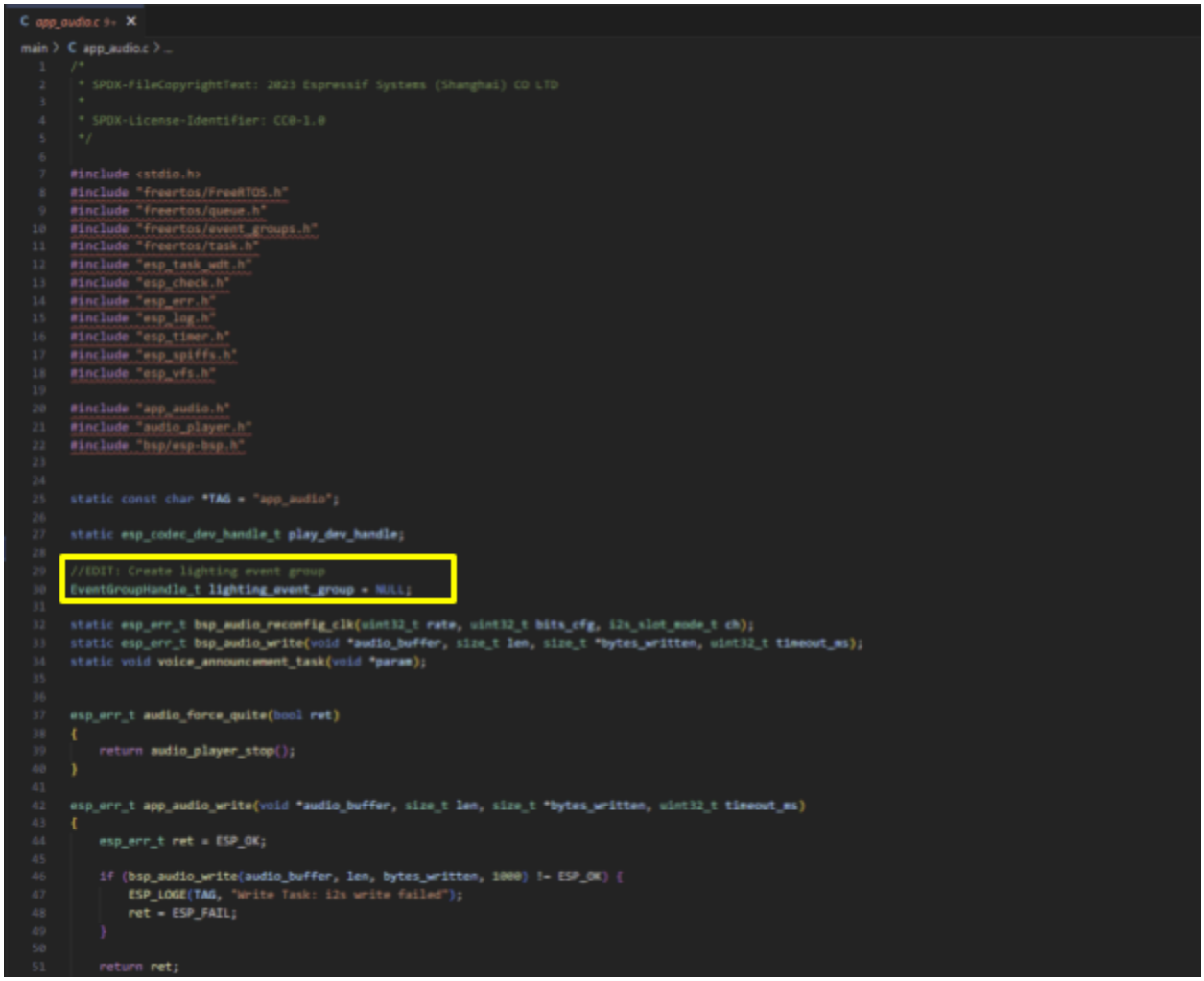
6. Code Screenshots

App_audio.h

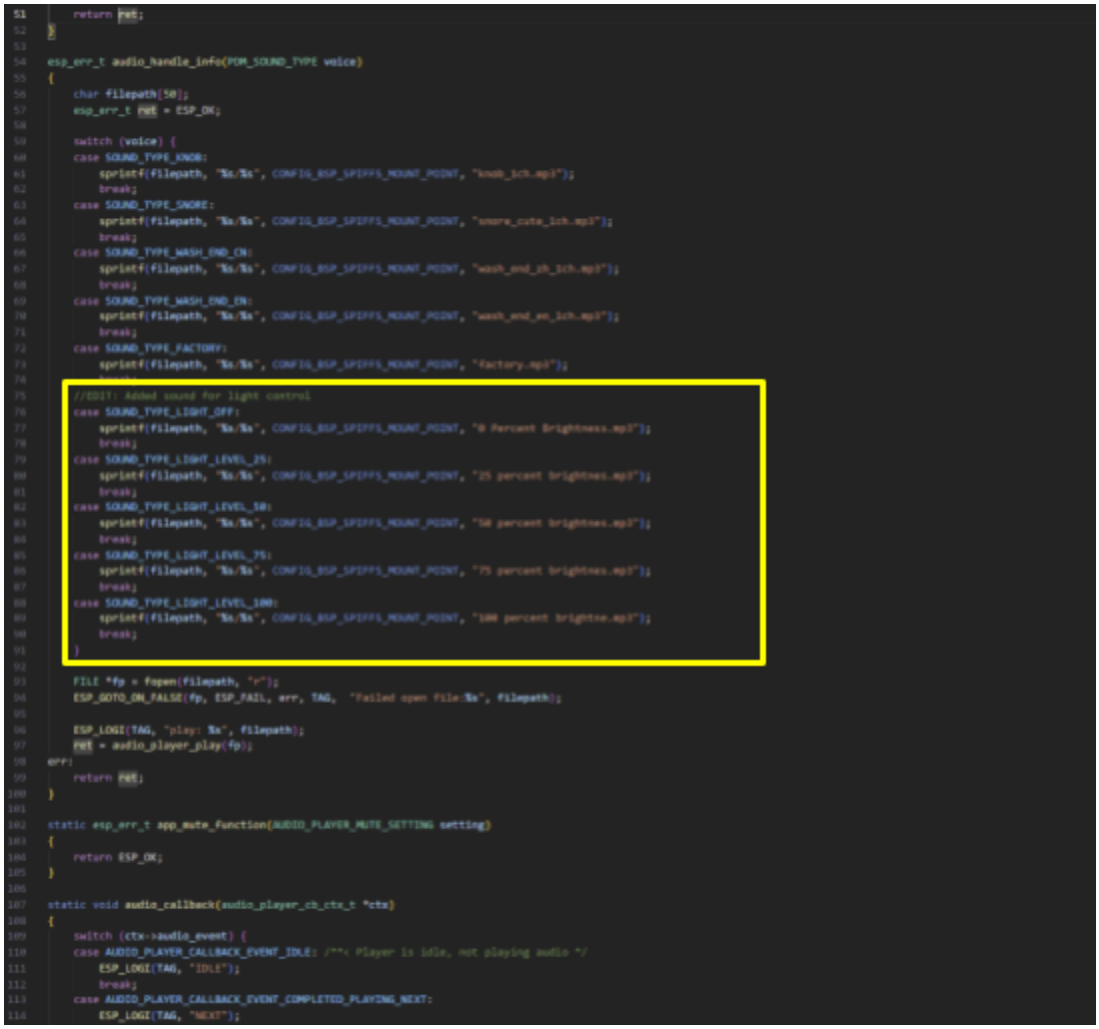


```c
/*
 * SPDX-FileCopyrightText: 2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: CC0-1.0
 */

#pragma once
#include "esp_err.h"
#include "freertos/FreeRTOS.h"
#include "freertos/semphr.h"
#include "freertos/event_groups.h"

typedef enum{
    SOUND_TYPE_KNOB,
    SOUND_TYPE_SNORE,
    SOUND_TYPE_WASH_END_CN,
    SOUND_TYPE_WASH_END_EN,
    SOUND_TYPE_FACTORY,
    //EDIT: Add sound for light control
    SOUND_TYPE_LIGHT_OFF,
    SOUND_TYPE_LIGHT_LEVEL_25,
    SOUND_TYPE_LIGHT_LEVEL_50,
    SOUND_TYPE_LIGHT_LEVEL_75,
    SOUND_TYPE_LIGHT_LEVEL_100,
}PDM_SOUND_TYPE;

esp_err_t audio_force_quite(bool ret);

esp_err_t audio_handle_info(PDM_SOUND_TYPE voice);

esp_err_t audio_play_start();

extern EventGroupHandle_t lighting_event_group;
extern SemaphoreHandle_t light_config_mutex;
```

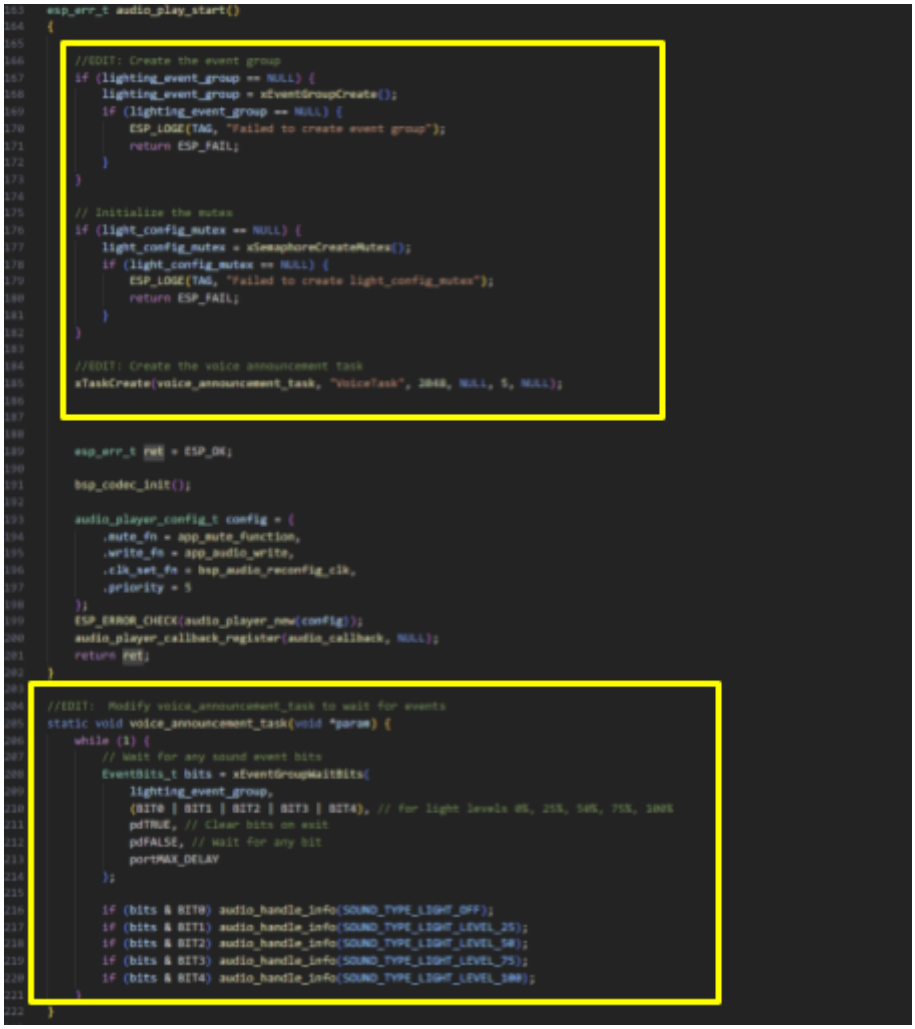
App_Audio.c



```c
/*
 * SPDX-FileCopyrightText: 2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: CC0-1.0
 */

#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/queue.h"
#include "freertos/event_groups.h"
#include "freertos/task.h"
#include "esp_task_wdt.h"
#include "esp_check.h"
#include "esp_err.h"
#include "esp_log.h"
#include "esp_timer.h"
#include "esp_spiffs.h"
#include "esp_vfs.h"

#include "app_audio.h"
#include "audio_player.h"
#include "bsp/esp-bsp.h"


static const char *TAG = "app_audio";

static esp_codec_dev_handle_t play_dev_handle;

//EDIT: Create lighting event group
EventGroupHandle_t lighting_event_group = NULL;

static esp_err_t bsp_audio_reconfig_clk(uint32_t rate, uint32_t bits_cfg, i2s_slot_mode_t ch);
static esp_err_t bsp_audio_write(void *audio_buffer, size_t len, size_t *bytes_written, uint32_t timeout_ms);
static void voice_announcement_task(void *param);


esp_err_t audio_force_quite(bool ret)
{
    return audio_player_stop();
}

esp_err_t app_audio_write(void *audio_buffer, size_t len, size_t *bytes_written, uint32_t timeout_ms)
{
    esp_err_t ret = ESP_OK;

    if (bsp_audio_write(audio_buffer, len, bytes_written, 1000) != ESP_OK) {
        ESP_LOGE(TAG, "Write Task: i2s write failed");
        ret = ESP_FAIL;
    }

    return ret;
```

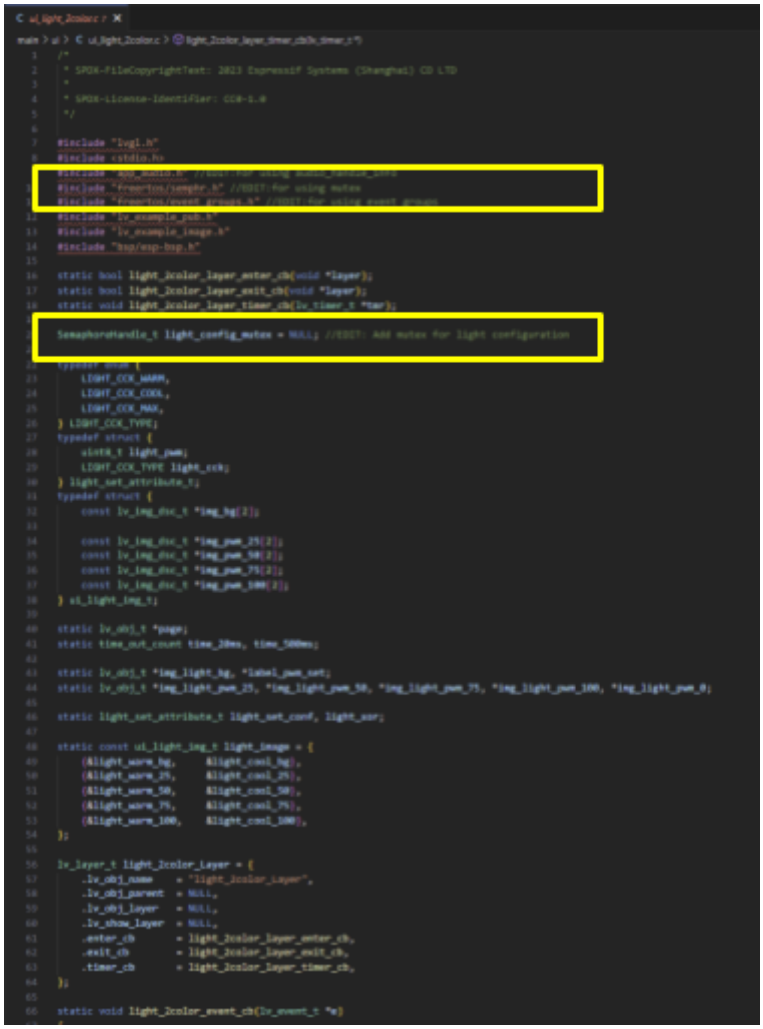


```
    return ret;
}

esp_err_t audio_handle_info(PDM_SOUND_TYPE voice)
{
    char filepath[50];
    esp_err_t ret = ESP_OK;

    switch (voice) {
    case SOUND_TYPE_KNOB:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "knob_1ch.mp3");
        break;
    case SOUND_TYPE_SNORE:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "snore_cute_1ch.mp3");
        break;
    case SOUND_TYPE_WASH_END_CN:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "wash_end_zh_1ch.mp3");
        break;
    case SOUND_TYPE_WASH_END_EN:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "wash_end_en_1ch.mp3");
        break;
    case SOUND_TYPE_FACTORY:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "factory.mp3");
        break;
    //EDIT: Added sound for light control
    case SOUND_TYPE_LIGHT_OFF:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "0 Percent Brightness.mp3");
        break;
    case SOUND_TYPE_LIGHT_LEVEL_25:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "25 percent brightnes.mp3");
        break;
    case SOUND_TYPE_LIGHT_LEVEL_50:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "50 percent brightnes.mp3");
        break;
    case SOUND_TYPE_LIGHT_LEVEL_75:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "75 percent brightnes.mp3");
        break;
    case SOUND_TYPE_LIGHT_LEVEL_100:
        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "100 percent brightne.mp3");
        break;
    }

    FILE *fp = fopen(filepath, "r");
    ESP_GOTO_ON_FALSE(fp, ESP_FAIL, err, TAG,  "Failed open File:%s", filepath);

    ESP_LOGI(TAG, "play: %s", filepath);
    ret = audio_player_play(fp);
err:
    return ret;
}

static esp_err_t app_mute_function(AUDIO_PLAYER_MUTE_SETTING setting)
{
    return ESP_OK;
}

static void audio_callback(audio_player_cb_ctx_t *ctx)
{
    switch (ctx->audio_event) {
    case AUDIO_PLAYER_CALLBACK_EVENT_IDLE: /**< Player is idle, not playing audio */
        ESP_LOGI(TAG, "IDLE");
        break;
    case AUDIO_PLAYER_CALLBACK_EVENT_COMPLETED_PLAYING_NEXT:
        ESP_LOGI(TAG, "NEXT");
```



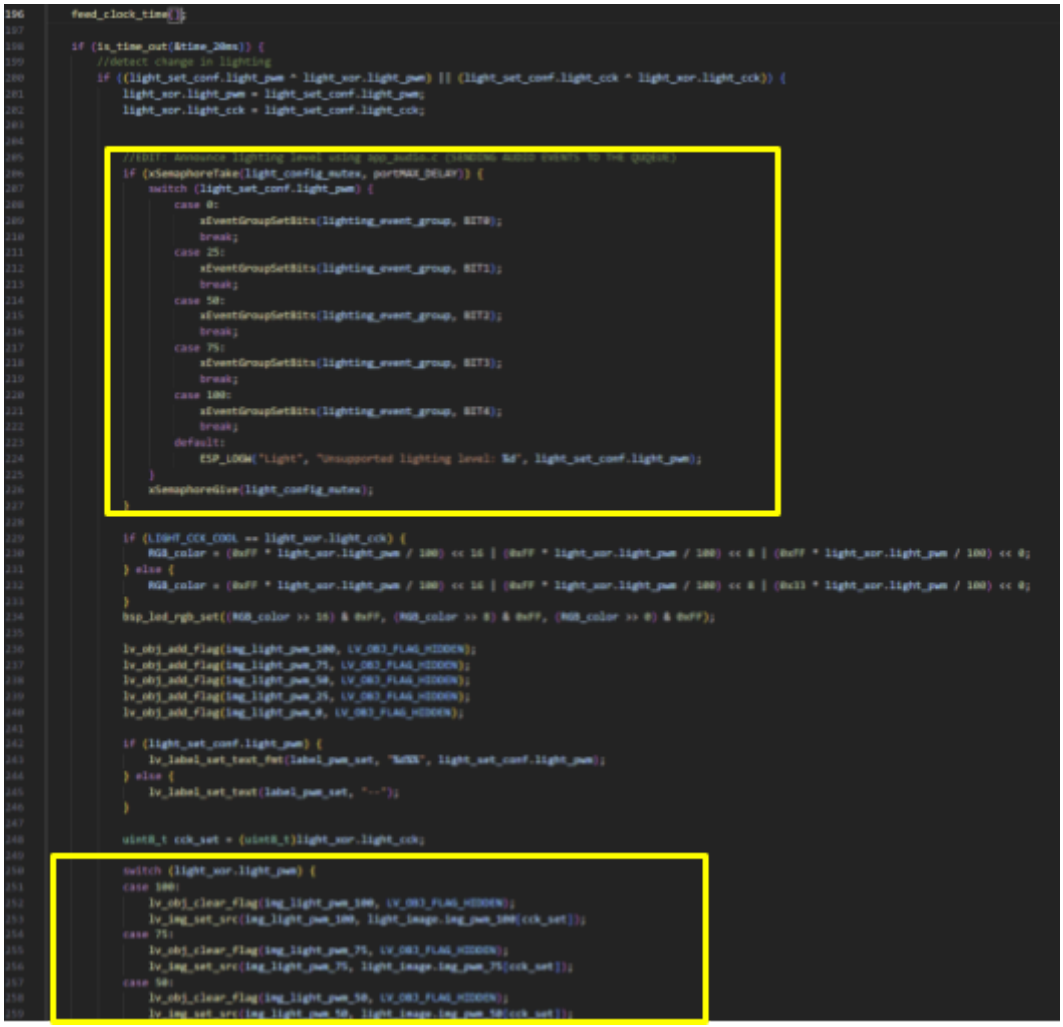
```
esp_err_t audio_play_start()
{
    //EDIT: Create the event group
    if (lighting_event_group == NULL) {
        lighting_event_group = xEventGroupCreate();
        if (lighting_event_group == NULL) {
            ESP_LOGE(TAG, "Failed to create event group");
            return ESP_FAIL;
        }
    }

    // Initialize the mutex
    if (light_config_mutex == NULL) {
        light_config_mutex = xSemaphoreCreateMutex();
        if (light_config_mutex == NULL) {
            ESP_LOGE(TAG, "Failed to create light_config_mutex");
            return ESP_FAIL;
        }
    }

    //EDIT: Create the voice announcement task
    xTaskCreate(voice_announcement_task, "VoiceTask", 2048, NULL, 5, NULL);


    esp_err_t ret = ESP_OK;

    bsp_codec_init();

    audio_player_config_t config = {
        .mute_fn = app_mute_function,
        .write_fn = app_audio_write,
        .clk_set_fn = bsp_audio_reconfig_clk,
        .priority = 5
    };
    ESP_ERROR_CHECK(audio_player_new(config));
    audio_player_callback_register(audio_callback, NULL);
    return ret;
}

//EDIT:  Modify voice_announcement_task 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 light levels 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);
    }
}
```

Ui_light_2color.c



```
static void light_2color_event_cb(lv_event_t *e)
{
    lv_event_code_t code = lv_event_get_code(e);

    if (LV_EVENT_FOCUSED == code) {
        lv_group_set_editing(lv_group_get_default(), true);
    } else if (LV_EVENT_KEY == code) {
        uint32_t key = lv_event_get_key(e);
        if (is_time_out(&time_500ms)) {
            if (LV_KEY_RIGHT == key) {
                if (light_set_conf.light_pwm < 100) {
                    light_set_conf.light_pwm += 25;
                }
            } else if (LV_KEY_LEFT == key) {
                if (light_set_conf.light_pwm > 0) {
                    light_set_conf.light_pwm -= 25;
                }
            }
        }
    } else if (LV_EVENT_CLICKED == code) {
        light_set_conf.light_cck = \
                                   (LIGHT_CCK_WARM == light_set_conf.light_cck) ? (LIGHT_CCK_COOL) : (LIGHT_CCK_WARM);
    } else if (LV_EVENT_LONG_PRESSED == code) {
        lv_indev_wait_release(lv_indev_get_next(NULL));
        ui_remove_all_objs_from_encoder_group();
        lv_func_goto_layer(&menu_layer);
    }
}

void ui_light_2color_init(lv_obj_t *parent)
{
    //EDIT: Create mutex for light configuration
    if (light_config_mutex == NULL) {
        light_config_mutex = xSemaphoreCreateMutex();
        if (light_config_mutex == NULL) {
            ESP_LOGE("Light", "Failed to create light_config_mutex");
            abort();
        }
    }

    light_scr.light_pwm = 0xFF;
    light_scr.light_cck = LIGHT_CCK_MAX;

    light_set_conf.light_pwm = 50;
    light_set_conf.light_cck = LIGHT_CCK_WARM;

    page = lv_obj_create(parent);
    lv_obj_set_size(page, LV_HOR_RES, LV_VER_RES);
    //lv_obj_set_size(page, lv_obj_get_width(lv_obj_get_parent(page)), lv_obj_get_height(lv_obj_get_parent(page)));

    lv_obj_set_style_border_width(page, 0, 0);
    lv_obj_set_style_radius(page, 0, 0);
    lv_obj_clear_flag(page, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_center(page);

    img_light_bg = lv_img_create(page);
    lv_img_set_src(img_light_bg, &light_warm_bg);
    lv_obj_align(img_light_bg, LV_ALIGN_CENTER, 0, 0);

    label_pwm_set = lv_label_create(page);
    lv_obj_set_style_text_font(label_pwm_set, &HelveticaNeue_Regular_24, 0);
    if (light_set_conf.light_pwm) {
        lv_label_set_text_fmt(label_pwm_set, "%d%%", light_set_conf.light_pwm);
    } else {
        lv_label_set_text(label_pwm_set, "--");
    }
```

```
    } else {
    }
    lv_obj_align(label_pwm_set, LV_ALIGN_CENTER, 0, 65);

    img_light_pwm_0 = lv_img_create(page);
    lv_img_set_src(img_light_pwm_0, &light_close_status);
    lv_obj_add_flag(img_light_pwm_0, LV_OBJ_FLAG_HIDDEN);
    lv_obj_align(img_light_pwm_0, LV_ALIGN_TOP_MID, 0, 0);

    img_light_pwm_25 = lv_img_create(page);
    lv_img_set_src(img_light_pwm_25, &light_warm_25);
    lv_obj_align(img_light_pwm_25, LV_ALIGN_TOP_MID, 0, 0);

    img_light_pwm_50 = lv_img_create(page);
    lv_img_set_src(img_light_pwm_50, &light_warm_50);
    lv_obj_align(img_light_pwm_50, LV_ALIGN_TOP_MID, 0, 0);

    img_light_pwm_75 = lv_img_create(page);
    lv_img_set_src(img_light_pwm_75, &light_warm_75);
    lv_obj_add_flag(img_light_pwm_75, LV_OBJ_FLAG_HIDDEN);
    lv_obj_align(img_light_pwm_75, LV_ALIGN_TOP_MID, 0, 0);

    img_light_pwm_100 = lv_img_create(page);
    lv_img_set_src(img_light_pwm_100, &light_warm_100);
    lv_obj_add_flag(img_light_pwm_100, LV_OBJ_FLAG_HIDDEN);
    lv_obj_align(img_light_pwm_100, LV_ALIGN_TOP_MID, 0, 0);

    lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_FOCUSED, NULL);
    lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_KEY, NULL);
    lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_LONG_PRESSED, NULL);
    lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_CLICKED, NULL);
    ui_add_obj_to_encoder_group(page);
}


static bool light_2color_layer_enter_cb(void *layer)
{
    bool ret = false;

    LV_LOG_USER("");
    lv_layer_t *create_layer = layer;
    if (NULL == create_layer->lv_obj_layer) {
        ret = true;
        create_layer->lv_obj_layer = lv_obj_create(lv_scr_act());
        lv_obj_remove_style_all(create_layer->lv_obj_layer);
        lv_obj_set_size(create_layer->lv_obj_layer, LV_HOR_RES, LV_VER_RES);

        ui_light_2color_init(create_layer->lv_obj_layer);
        set_time_out(&time_20ms, 20);
        set_time_out(&time_500ms, 200);
    }

    return ret;
}

static bool light_2color_layer_exit_cb(void *layer)
{
    LV_LOG_USER("");
    bsp_led_rgb_set(0x00, 0x00, 0x00);
    return true;
}

static void light_2color_layer_timer_cb(lv_timer_t *tmr)
{
    uint32_t RGB_color = 0xFF;
```



```
    feed_clock_time();

    if (is_time_out(&time_20ms)) {
        //detect change in lighting
        if ((light_set_conf.light_pwm ^ light_xor.light_pwm) || (light_set_conf.light_cck ^ light_xor.light_cck)) {
            light_xor.light_pwm = light_set_conf.light_pwm;
            light_xor.light_cck = light_set_conf.light_cck;


            //EDIT: Announce lighting level using app_audio.c (SENDING AUDIO EVENTS TO THE QUEUE)
            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);
            }

            if (LIGHT_CCK_COOL == light_xor.light_cck) {
                RGB_color = (0xFF * light_xor.light_pwm / 100) << 16 | (0xFF * light_xor.light_pwm / 100) << 8 | (0xFF * light_xor.light_pwm / 100) << 0;
            } else {
                RGB_color = (0xFF * light_xor.light_pwm / 100) << 16 | (0xFF * light_xor.light_pwm / 100) << 8 | (0x33 * light_xor.light_pwm / 100) << 0;
            }
            bsp_led_rgb_set((RGB_color >> 16) & 0xFF, (RGB_color >> 8) & 0xFF, (RGB_color >> 0) & 0xFF);

            lv_obj_add_flag(img_light_pwm_100, LV_OBJ_FLAG_HIDDEN);
            lv_obj_add_flag(img_light_pwm_75, LV_OBJ_FLAG_HIDDEN);
            lv_obj_add_flag(img_light_pwm_50, LV_OBJ_FLAG_HIDDEN);
            lv_obj_add_flag(img_light_pwm_25, LV_OBJ_FLAG_HIDDEN);
            lv_obj_add_flag(img_light_pwm_0, LV_OBJ_FLAG_HIDDEN);

            if (light_set_conf.light_pwm) {
                lv_label_set_text_fmt(label_pwm_set, "%d%%", light_set_conf.light_pwm);
            } else {
                lv_label_set_text(label_pwm_set, "--");
            }

            uint8_t cck_set = (uint8_t)light_xor.light_cck;

            switch (light_xor.light_pwm) {
            case 100:
                lv_obj_clear_flag(img_light_pwm_100, LV_OBJ_FLAG_HIDDEN);
                lv_img_set_src(img_light_pwm_100, light_image.img_pwm_100[cck_set]);
            case 75:
                lv_obj_clear_flag(img_light_pwm_75, LV_OBJ_FLAG_HIDDEN);
                lv_img_set_src(img_light_pwm_75, light_image.img_pwm_75[cck_set]);
            case 50:
                lv_obj_clear_flag(img_light_pwm_50, LV_OBJ_FLAG_HIDDEN);
                lv_img_set_src(img_light_pwm_50, light_image.img_pwm_50[cck_set]);
```

```
                case 25:
                    lv_obj_clear_flag(img_light_pwm_25, LV_OBJ_FLAG_HIDDEN);
                    lv_img_set_src(img_light_pwm_25, light_image.img_pwm_25[cck_set]);
                    lv_img_set_src(img_light_bg, light_image.img_bg[cck_set]);
                    break;
                case 0:
                    lv_obj_clear_flag(img_light_pwm_0, LV_OBJ_FLAG_HIDDEN);
                    lv_img_set_src(img_light_bg, &light_close_bg);
                    break;
                default:
                    break;
                }
            }
        }
    }
```

Bonus Feature Located in App_main.c:

```
// Task to display the custom message and trigger the voice announcement
void announcement_task(void* pvParameters) {
    // Split the message into smaller parts
    const char* messages[] = {
        "Hello CIS 450,",
        "this is our ESP32",
        "Threading Project,",
        "created by Souad,",
        "Sukeina, Heather,",
        "and Firas :)"
    };
    const int num_messages = sizeof(messages) / sizeof(messages[0]); // Number of message parts

    while (1) {
        for (int i = 0; i < num_messages; i++) {
            update_announcement_label(messages[i]); // Update the label with the current part
            vTaskDelay(pdMS_TO_TICKS(3000));       // Wait 3 seconds before showing the next part
        }
    }
}
```

7. **Github Link**
   https://github.com/susu0mar/CIS450-Lighting-Audio_Integration-Final.git

8. **Video Link**

   Demo starts at around 4 minute mark in the video!
   https://youtu.be/6VCtub6oggU