

CIS 450 Final Project

Audio Integration

Version 1.0

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

Table of Contents

1. System Architecture	3
2. Concurrency Control Explanation	3
3. User Guide	4
4. Bonus Feature Implementation	4
5. Proof of Compilation	4
6. Code Screenshots	4
7. Github Link	4
8. Video Link	5

1. System Architecture

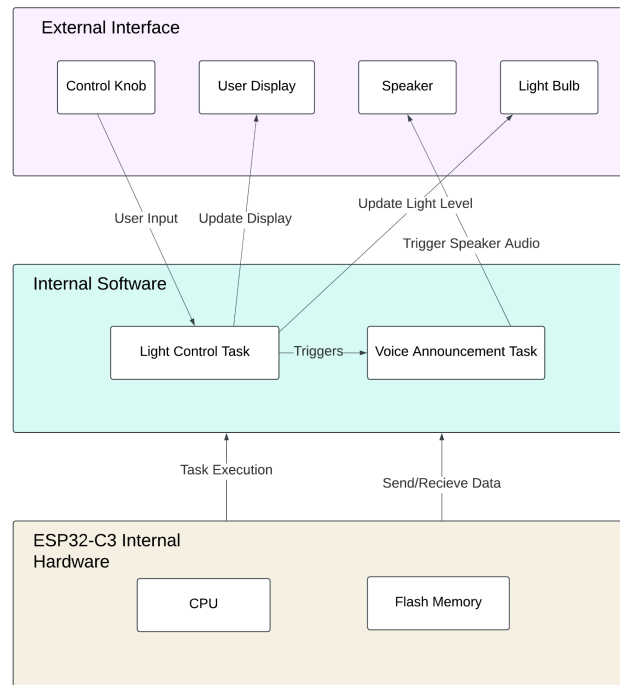
For this project, we used freeRTOS to implement a threading structure to allow the device to share resources and run tasks concurrently. We have four main aspects for our project architecture: the UI from the LCD screen, the Lighting control task, the voice announcement task, and the internal hardware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

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

The screenshot shows a VS Code editor with a C file named `ui_light_2color.c` open. The code is for an ESP-IDF project and includes a timer callback function `light_2color_layer_timer_cb` that sets lighting levels based on a PWM value. The code uses `xSemaphoreTake` and `xEventGroupSetBits` for thread synchronization. A comment indicates it's for sending audio events to a queue. The build output at the bottom shows a successful compilation with a table of memory usage:

	Address	Size	Start	End
Flash Data	2575024	30.7	5813552	8388576
.rodata	2574768	30.69		
.appdesc	256	0.0		
Flash Code	503796	6.01	7884780	8388576
.text	503796	6.01		
DRAM	101492	31.59	219804	321296
.text	69820	21.73		
.bss	22824	7.1		
.data	8516	2.65		
RTC SLOW	540	6.59	7652	8192
.rtc_reserved	24	0.29		
.force_fast	4	0.05		

Total image size: 3157160 bytes (.bin may be padded larger)

A "Build Successfully" message is displayed in the bottom right corner of the terminal panel.

6. Code Screenshots

App_audio.h

```
C app_audio.h 5 X
main > C app_audio.h > _
1
2
3  /*
4   * SPDX-FileCopyrightText: 2023 Espressif Systems (Shanghai) CO LTD
5   *
6   * SPDX-License-Identifier: CC0-1.0
7   */
8
9  #pragma once
10 #include "esp_err.h"
11 #include "freertos/freertos.h"
12 #include "freertos/semphr.h"
13 #include "freertos/event_groups.h"
14
15 typedef enum{
16     SOUND_TYPE_KNOB,
17     SOUND_TYPE_SNORE,
18     SOUND_TYPE_WASH_END_CN,
19     SOUND_TYPE_WASH_END_EN,
20     SOUND_TYPE_FACTORY,
21     //EDIT: Add sound for light control
22     SOUND_TYPE_LIGHT_OFF,
23     SOUND_TYPE_LIGHT_LEVEL_25,
24     SOUND_TYPE_LIGHT_LEVEL_50,
25     SOUND_TYPE_LIGHT_LEVEL_75,
26     SOUND_TYPE_LIGHT_LEVEL_100,
27 }PDM_SOUND_TYPE;
28
29 esp_err_t audio_force_quite(bool ret);
30
31 esp_err_t audio_handle_info(PDM_SOUND_TYPE voice);
32
33 esp_err_t audio_play_start();
34
35 extern EventGroupHandle_t lighting_event_group;
36 extern SemaphoreHandle_t light_config_mutex;
```

App_Audio.c

```
C:\app_audio.c 9: X
main > C:\app_audio.c > _
1  /*
2   * SPDX-FileCopyrightText: 2023 Expressif Systems (Shanghai) CO LTD
3   *
4   * SPDX-License-Identifier: CC0-1.0
5   */
6
7  #include <stdio.h>
8  #include "freertos/FreeRTOS.h"
9  #include "freertos/queue.h"
10 #include "freertos/event_groups.h"
11 #include "freertos/task.h"
12 #include "esp_task_wdt.h"
13 #include "esp_check.h"
14 #include "esp_err.h"
15 #include "esp_log.h"
16 #include "esp_timer.h"
17 #include "esp_spiffs.h"
18 #include "esp_vfs.h"
19
20 #include "app_audio.h"
21 #include "audio_player.h"
22 #include "bsp/esp-bsp.h"
23
24 static const char *TAG = "app_audio";
25
26 static esp_codec_dev_handle_t play_dev_handle;
27
28 //ID11: Create lighting event group
29 EventGroupHandle_t lighting_event_group = NULL;
30
31 static esp_err_t bsp_audio_reconfig_clk(uint32_t rate, uint32_t bits_cfg, i2s_slot_mode_t ch);
32 static esp_err_t bsp_audio_write(void *audio_buffer, size_t len, size_t *bytes_written, uint32_t timeout_ms);
33 static void voice_announcement_task(void *param);
34
35
36
37 esp_err_t audio_force_quit(bool ret)
38 {
39     return audio_player_stop();
40 }
41
42 esp_err_t app_audio_write(void *audio_buffer, size_t len, size_t *bytes_written, uint32_t timeout_ms)
43 {
44     esp_err_t ret = ESP_OK;
45
46     if (bsp_audio_write(audio_buffer, len, bytes_written, 1000) != ESP_OK) {
47         ESP_LOGE(TAG, "write Task: i2s write failed");
48         ret = ESP_FAIL;
49     }
50
51     return ret;
52 }
```



```

51     return RET;
52 }
53
54 esp_err_t audio_handle_info(PDM_SOUND_TYPE voice)
55 {
56     char filepath[50];
57     esp_err_t RET = ESP_OK;
58
59     switch (voice) {
60     case SOUND_TYPE_KNOB:
61         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "knob_1ch.mp3");
62         break;
63     case SOUND_TYPE_SNORE:
64         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "snore_cute_1ch.mp3");
65         break;
66     case SOUND_TYPE_WASH_END_CN:
67         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "wash_end_cn_1ch.mp3");
68         break;
69     case SOUND_TYPE_WASH_END_EN:
70         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "wash_end_en_1ch.mp3");
71         break;
72     case SOUND_TYPE_FACTORY:
73         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "factory.mp3");
74         break;
75
76         //EDIT: Added sound for light control
77     case SOUND_TYPE_LIGHT_OFF:
78         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "0 Percent Brightness.mp3");
79         break;
80     case SOUND_TYPE_LIGHT_LEVEL_25:
81         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "25 percent brightness.mp3");
82         break;
83     case SOUND_TYPE_LIGHT_LEVEL_50:
84         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "50 percent brightness.mp3");
85         break;
86     case SOUND_TYPE_LIGHT_LEVEL_75:
87         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "75 percent brightness.mp3");
88         break;
89     case SOUND_TYPE_LIGHT_LEVEL_100:
90         sprintf(filepath, "%s/%s", CONFIG_ESP_SPIFFS_MOUNT_POINT, "100 percent brightne.mp3");
91         break;
92     }
93
94     FILE *fp = fopen(filepath, "r");
95     ESP_GOTO_ON_FALSE(fp, ESP_FAIL, err, TAG, "Failed open File:%s", filepath);
96
97     ESP_LOGI(TAG, "play: %s", filepath);
98     RET = audio_player_play(fp);
99
100 err:
101     return RET;
102 }
103
104 static esp_err_t app_mute_function(AUDIO_PLAYER_MUTE_SETTING setting)
105 {
106     return ESP_OK;
107 }
108
109 static void audio_callback(audio_player_ch_ctx_t *ctx)
110 {
111     switch (ctx->audio_event) {
112     case AUDIO_PLAYER_CALLBACK_EVENT_IDLE: /** Player is idle, not playing audio */
113         ESP_LOGI(TAG, "IDLE");
114         break;
115     case AUDIO_PLAYER_CALLBACK_EVENT_COMPLETED_PLAYING_NEXT:
116         ESP_LOGI(TAG, "NEXT");
117     }
118 }

```

```

163 esp_err_t audio_play_start()
164 {
165     //EDIT: Create the event group
166     if (lighting_event_group == NULL) {
167         lighting_event_group = xEventGroupCreate();
168         if (lighting_event_group == NULL) {
169             ESP_LOGE(TAG, "Failed to create event group");
170             return ESP_FAIL;
171         }
172     }
173
174     // Initialize the mutex
175     if (light_config_mutex == NULL) {
176         light_config_mutex = xSemaphoreCreateMutex();
177         if (light_config_mutex == NULL) {
178             ESP_LOGE(TAG, "Failed to create light_config_mutex");
179             return ESP_FAIL;
180         }
181     }
182
183     //EDIT: Create the voice announcement task
184     xTaskCreate(voice_announcement_task, "VoiceTask", 2048, NULL, 5, NULL);
185
186     esp_err_t ret = ESP_OK;
187
188     bsp_codec_init();
189
190     audio_player_config_t config = {
191         .mute_fn = app_mute_function,
192         .write_fn = app_audio_write,
193         .clk_set_fn = bsp_audio_reconfig_clk,
194         .priority = 5
195     };
196     ESP_ERROR_CHECK(audio_player_new(config));
197     audio_player_callback_register(audio_callback, NULL);
198     return ret;
199 }
200
201 //EDIT: Modify voice_announcement_task to wait for events
202 static void voice_announcement_task(void *param) {
203     while (1) {
204         // Wait for any sound event bits
205         EventBits_t bits = xEventGroupWaitBits(
206             lighting_event_group,
207             (BIT0 | BIT1 | BIT2 | BIT3 | BIT4), // for light levels 00, 25, 50, 75, 100
208             pdTRUE, // Clear bits on exit
209             pdFALSE, // Wait for any bit
210             portMAX_DELAY
211         );
212
213         if (bits & BIT0) audio_handle_info(SOUND_TYPE_LIGHT_OFF);
214         if (bits & BIT1) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_25);
215         if (bits & BIT2) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_50);
216         if (bits & BIT3) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_75);
217         if (bits & BIT4) audio_handle_info(SOUND_TYPE_LIGHT_LEVEL_100);
218     }
219 }

```

Ui_light_2color.c

```

C:\light_decoder > K
main > a > C:\light_decoder > @light_decoder_layer_timer_callback_timer_t
1
2
3 * SPDX-FileCopyrightText: 2023 Expressif Systems (Shanghai) CO LTD
4
5 * SPDX-License-Identifier: CC0-1.0
6
7 #include "log.h"
8 #include "stdio.h"
9
10 #ifndef __GNUC__
11 #include "freertos/semphr.h" //NOT for using mutex
12 #include "freertos/event_groups.h" //NOT for using event groups
13 #include "lv_example_obj.h"
14 #include "lv_example_image.h"
15 #include "esp_log.h"
16
17 static bool light_decoder_layer_enter_cb(void *layer);
18 static bool light_decoder_layer_exit_cb(void *layer);
19 static void light_decoder_layer_timer_cb(void *timer, uint32_t *time);
20
21 SemaphoreHandle_t light_config_mutex = NULL; //NOT: Add mutex for light configuration
22
23 #define LIGHT_COX_MAX 100
24 #define LIGHT_COX_MIN 0
25 #define LIGHT_COX_STEP 1
26 #define LIGHT_COX_TYPE 1
27 typedef struct {
28     uint8_t light_pwm;
29     LIGHT_COX_TYPE light_cox;
30 } light_set_attribute_t;
31 typedef struct {
32     const lv_img_desc_t *img_bg;
33
34     const lv_img_desc_t *img_pwm_25;
35     const lv_img_desc_t *img_pwm_50;
36     const lv_img_desc_t *img_pwm_75;
37     const lv_img_desc_t *img_pwm_100;
38 } ui_light_img_t;
39
40 static lv_obj_t *page;
41 static time_t count_time_30ms, time_500ms;
42
43 static lv_obj_t *img_light_bg, *label_pwm_set;
44 static lv_obj_t *img_light_pwm_25, *img_light_pwm_50, *img_light_pwm_75, *img_light_pwm_100, *img_light_pwm_0;
45
46 static light_set_attribute_t light_set_conf, light_set;
47
48 static const ui_light_img_t light_image = {
49     (img_light_bg, img_cox_bg),
50     (img_light_pwm_25, img_cox_25),
51     (img_light_pwm_50, img_cox_50),
52     (img_light_pwm_75, img_cox_75),
53     (img_light_pwm_100, img_cox_100),
54 };
55
56 lv_layer_t light_decoder_layer = {
57     .lv_obj_name = "light_decoder_layer",
58     .lv_obj_parent = NULL,
59     .lv_obj_layer = NULL,
60     .lv_obj_theme = NULL,
61     .enter_cb = light_decoder_layer_enter_cb,
62     .exit_cb = light_decoder_layer_exit_cb,
63     .timer_cb = light_decoder_layer_timer_cb,
64 };
65
66 static void light_decoder_event_cb(lv_event_t *e)
67

```

Final Project Document

```

66 static void light_encoder_event_cb(lv_event_t *e)
67 {
68     lv_event_code_t code = lv_event_get_code(e);
69
70     if (LV_EVENT_FOCUSED == code) {
71         lv_group_set_editing(lv_group_get_default(), true);
72     } else if (LV_EVENT_KEY == code) {
73         uint32_t key = lv_event_get_key(e);
74         if (lx_time_get(time_300ms)) {
75             if (LV_KEY_RIGHT == key) {
76                 if (light_set_conf.light_pwm < 100) {
77                     light_set_conf.light_pwm += 10;
78                 }
79             } else if (LV_KEY_LEFT == key) {
80                 if (light_set_conf.light_pwm > 40) {
81                     light_set_conf.light_pwm -= 10;
82                 }
83             }
84         }
85     } else if (LV_EVENT_CLICKED == code) {
86         light_set_conf.light_cek = !
87             (LIGHT_CCK_MAX == light_set_conf.light_cek) ? (LIGHT_CCK_MIN) : (LIGHT_CCK_MAX);
88     } else if (LV_EVENT_LONG_PRESSED == code) {
89         lv_indev_wait_release(lv_indev_get_next(NULL));
90         ui_remove_all_objs_from_encoder_group();
91         lv_font_goto_layer(home_layer);
92     }
93 }
94
95 void ui_light_encoder_init(lv_obj_t *parent)
96 {
97     //TODO: Create mutex for light configuration
98     if (light_config_mutex == NULL) {
99         light_config_mutex = xSemaphoreCreateMutex();
100     }
101     if (light_config_mutex == NULL) {
102         ESP_LOGE("light", "Failed to create light_config_mutex");
103         abort();
104     }
105
106     light_set_conf.light_pwm = 0;
107     light_set_conf.light_cek = LIGHT_CCK_MAX;
108
109     light_set_conf.light_pwm = 50;
110     light_set_conf.light_cek = LIGHT_CCK_MAX;
111
112     page = lv_obj_create(parent);
113     lv_obj_set_size(page, LV_HOR_RES, LV_VER_RES);
114     //lv_obj_set_size(page, lv_obj_get_width(lv_obj_get_parent(page)), lv_obj_get_height(lv_obj_get_parent(page)));
115
116     lv_obj_set_style_border_width(page, 0, 0);
117     lv_obj_set_style_radius(page, 0, 0);
118     lv_obj_clear_flag(page, LV_OBJ_FLAG_SCROLLABLE);
119     lv_obj_center(page);
120
121     img_light_bg = lv_img_create(page);
122     lv_img_set_src(img_light_bg, &light_worm_bg);
123     lv_obj_align(img_light_bg, LV_ALIGN_CENTER, 0, 0);
124
125     label_pwm_set = lv_label_create(page);
126     lv_obj_set_style_text_font(label_pwm_set, &helvetica_regular_24, 0);
127     if (light_set_conf.light_pwm) {
128         lv_label_set_text(label_pwm_set, "LED", light_set_conf.light_pwm);
129     } else {
130         lv_label_set_text(label_pwm_set, "-");
131     }

```

```
129     } else {
131         |
132         lv_obj_align(label_pwm_set, LV_ALIGN_CENTER, 0, 65);
133
134         img_light_pwm_0 = lv_img_create(page);
135         lv_img_set_src(img_light_pwm_0, &light_close_status);
136         lv_obj_add_flag(img_light_pwm_0, LV_OBJ_FLAG_HIDDEN);
137         lv_obj_align(img_light_pwm_0, LV_ALIGN_TOP_MID, 0, 0);
138
139         img_light_pwm_25 = lv_img_create(page);
140         lv_img_set_src(img_light_pwm_25, &light_warm_25);
141         lv_obj_align(img_light_pwm_25, LV_ALIGN_TOP_MID, 0, 0);
142
143         img_light_pwm_50 = lv_img_create(page);
144         lv_img_set_src(img_light_pwm_50, &light_warm_50);
145         lv_obj_align(img_light_pwm_50, LV_ALIGN_TOP_MID, 0, 0);
146
147         img_light_pwm_75 = lv_img_create(page);
148         lv_img_set_src(img_light_pwm_75, &light_warm_75);
149         lv_obj_add_flag(img_light_pwm_75, LV_OBJ_FLAG_HIDDEN);
150         lv_obj_align(img_light_pwm_75, LV_ALIGN_TOP_MID, 0, 0);
151
152         img_light_pwm_100 = lv_img_create(page);
153         lv_img_set_src(img_light_pwm_100, &light_warm_100);
154         lv_obj_add_flag(img_light_pwm_100, LV_OBJ_FLAG_HIDDEN);
155         lv_obj_align(img_light_pwm_100, LV_ALIGN_TOP_MID, 0, 0);
156
157         lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_FOCUSED, NULL);
158         lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_KEY, NULL);
159         lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_LONG_PRESSED, NULL);
160         lv_obj_add_event_cb(page, light_2color_event_cb, LV_EVENT_CLICKED, NULL);
161         ui_add_obj_to_encoder_group(page);
162     }
163
164
165 static bool light_2color_layer_enter_cb(void *layer)
166 {
167     bool ret = false;
168
169     LV_LOG_USER("");
170     lv_layer_t *create_layer = layer;
171     if (NULL == create_layer->lv_obj_layer) {
172         ret = true;
173         create_layer->lv_obj_layer = lv_obj_create(lv_scr_act());
174         lv_obj_remove_style_all(create_layer->lv_obj_layer);
175         lv_obj_set_size(create_layer->lv_obj_layer, LV_HOR_RES, LV_VER_RES);
176
177         ui_light_2color_init(create_layer->lv_obj_layer);
178         set_time_out(&time_20ms, 20);
179         set_time_out(&time_500ms, 200);
180     }
181
182     return ret;
183 }
184
185 static bool light_2color_layer_exit_cb(void *layer)
186 {
187     LV_LOG_USER("");
188     bsp_led_rgb_set(0x00, 0x00, 0x00);
189     return true;
190 }
191
192 static void light_2color_layer_timer_cb(lv_timer_t *tmr)
193 {
194     uint32_t RGB_color = 0xFF;
```

```

196 feed_clock_time();
197
198 if (ix_time_out(&time_30ms)) {
199     //detect change in lighting
200     if ((light_set_conf.light_pwm * light_xor.light_pwm) || (light_set_conf.light_cck * light_xor.light_cck)) {
201         light_xor.light_pwm = light_set_conf.light_pwm;
202         light_xor.light_cck = light_set_conf.light_cck;
203     }
204
205     //!!! resource lighting level using esp8266's (arduino) gpio pins to the gpio pins
206     if (xSemaphoreTake(light_config_mutex, portMAX_DELAY)) {
207         switch (light_set_conf.light_pwm) {
208             case 0:
209                 xEventGroupSetBits(lighting_event_group, BIT0);
210                 break;
211             case 25:
212                 xEventGroupSetBits(lighting_event_group, BIT1);
213                 break;
214             case 50:
215                 xEventGroupSetBits(lighting_event_group, BIT2);
216                 break;
217             case 75:
218                 xEventGroupSetBits(lighting_event_group, BIT3);
219                 break;
220             case 100:
221                 xEventGroupSetBits(lighting_event_group, BIT4);
222                 break;
223             default:
224                 ESP_LOGW("light", "Unsupported lighting level: %d", light_set_conf.light_pwm);
225         }
226         xSemaphoreGive(light_config_mutex);
227     }
228
229     if ((light_cck_000 == light_xor.light_cck) {
230         RGB_color = (0xFF * light_xor.light_pwm / 100) << 16 | (0xFF * light_xor.light_pwm / 100) << 8 | (0xFF * light_xor.light_pwm / 100) << 0;
231     } else {
232         RGB_color = (0xFF * light_xor.light_pwm / 100) << 16 | (0xFF * light_xor.light_pwm / 100) << 8 | (0xFF * light_xor.light_pwm / 100) << 0;
233     }
234     bsp_led_rgb_set((RGB_color >> 16) & 0xFF, (RGB_color >> 8) & 0xFF, (RGB_color >> 0) & 0xFF);
235
236     lv_obj_add_flag(img_light_pwm_100, LV_OBJ_FLAG_HIDDEN);
237     lv_obj_add_flag(img_light_pwm_75, LV_OBJ_FLAG_HIDDEN);
238     lv_obj_add_flag(img_light_pwm_50, LV_OBJ_FLAG_HIDDEN);
239     lv_obj_add_flag(img_light_pwm_25, LV_OBJ_FLAG_HIDDEN);
240     lv_obj_add_flag(img_light_pwm_0, LV_OBJ_FLAG_HIDDEN);
241
242     if (light_set_conf.light_pwm) {
243         lv_label_set_text(label_pwm_set, "100%");
244     } else {
245         lv_label_set_text(label_pwm_set, "...");
246     }
247
248     uint8_t cck_set = (uint8_t)light_xor.light_cck;
249
250     switch (light_xor.light_pwm) {
251         case 100:
252             lv_obj_clear_flag(img_light_pwm_100, LV_OBJ_FLAG_HIDDEN);
253             lv_img_set_src(img_light_pwm_100, light_image.img_pwm_100[cck_set]);
254         case 75:
255             lv_obj_clear_flag(img_light_pwm_75, LV_OBJ_FLAG_HIDDEN);
256             lv_img_set_src(img_light_pwm_75, light_image.img_pwm_75[cck_set]);
257         case 50:
258             lv_obj_clear_flag(img_light_pwm_50, LV_OBJ_FLAG_HIDDEN);
259             lv_img_set_src(img_light_pwm_50, light_image.img_pwm_50[cck_set]);
260
261         case 25:
262             lv_obj_clear_flag(img_light_pwm_25, LV_OBJ_FLAG_HIDDEN);
263             lv_img_set_src(img_light_pwm_25, light_image.img_pwm_25[cck_set]);
264             lv_img_set_src(img_light_bg, light_image.img_bg[cck_set]);
265             break;
266         case 0:
267             lv_obj_clear_flag(img_light_pwm_0, LV_OBJ_FLAG_HIDDEN);
268             lv_img_set_src(img_light_bg, &light_close_bg);
269             break;
270         default:
271             break;
272     }
273 }
274 }

```

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>