CIS 450: Operating Systems Final Project
ESP32 Enhanced Lighting Control with Voice Feedback

December 13, 2024

Abraham Abdulkarim, Calli Best, Adam Winder

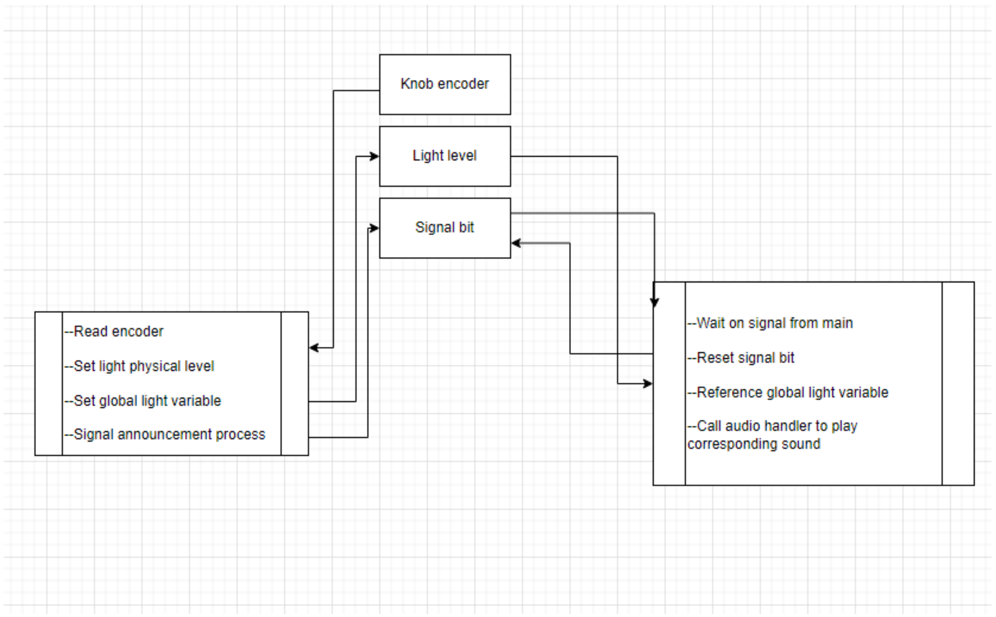
Introduction

With this project, we were tasked to expand on the Knob Panel Example given to our team and implement a new functionality. Currently, the software is being run as an ESP32-C3-LCDKit and is designed to allow users to interact with its lighting, washing machine, and thermostat features.

The feature we will be modifying is the lighting control feature, which allows users to turn the knob right and left, increasing and decreasing the brightness of the light being emitted from the board. If the knob is pressed in, the user is also able to change the color temperature of the light between warm and cool tones.

With this project, we will be implementing a voice announcement system that, with the adjustment of light brightness, audibly announces the new brightness percentage after it has been changed.

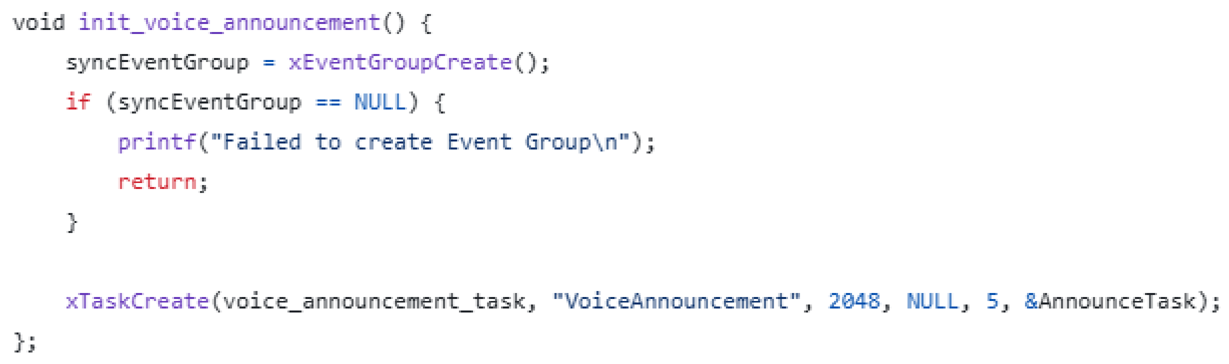
System Architecture


Knob encoder
Light level
Signal bit
--Read encoder
--Set light physical level
--Set global light variable
--Signal announcement process
--Wait on signal from main
--Reset signal bit
--Reference global light variable
--Call audio handler to play corresponding sound

The most relevant activity of the main process is the light_2color_layer_timer_cb function, which reads data from the knob encoder, triggering a change in the light level. When this happens, a global light level variable is updated, and a signal is sent which wakes up the previously started child process. As the child process wakes up, it resets the bit that signaled it in order to avoid continuously running. The child process checks the value of the light variable and passes the appropriate corresponding flag to the audio handler. Finally, it loops back to the top to wait for another signal from the main process.

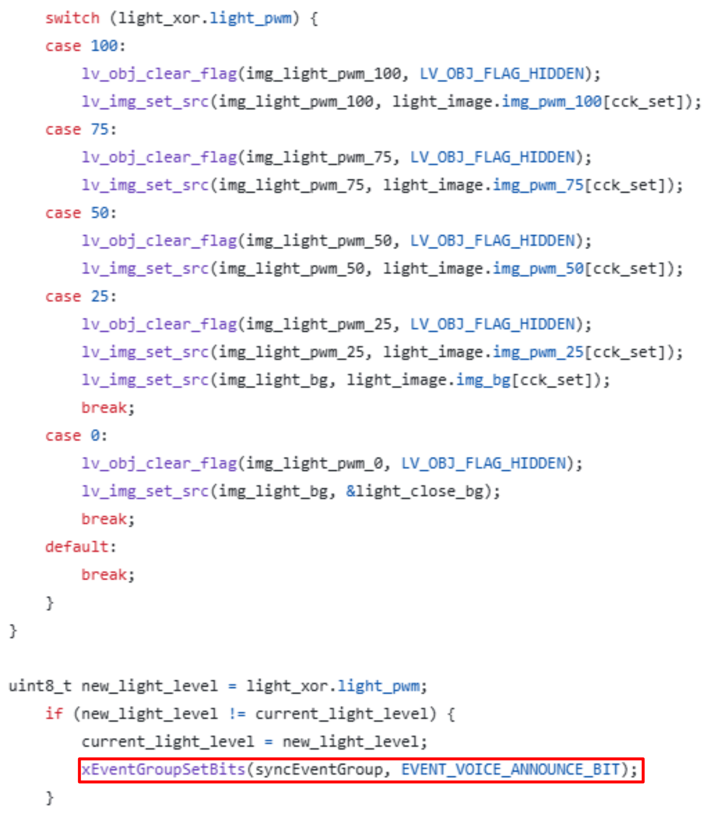
Concurrency Control

In order to ensure the concurrent activities of different tasks, we had to utilize mutex locks provided by the FreeRTOS library. With this, we were able to create processes for the system to run and an event group to manage the separate concurrently running events utilizing the xEventGroupCreate and xTaskCreate functions.

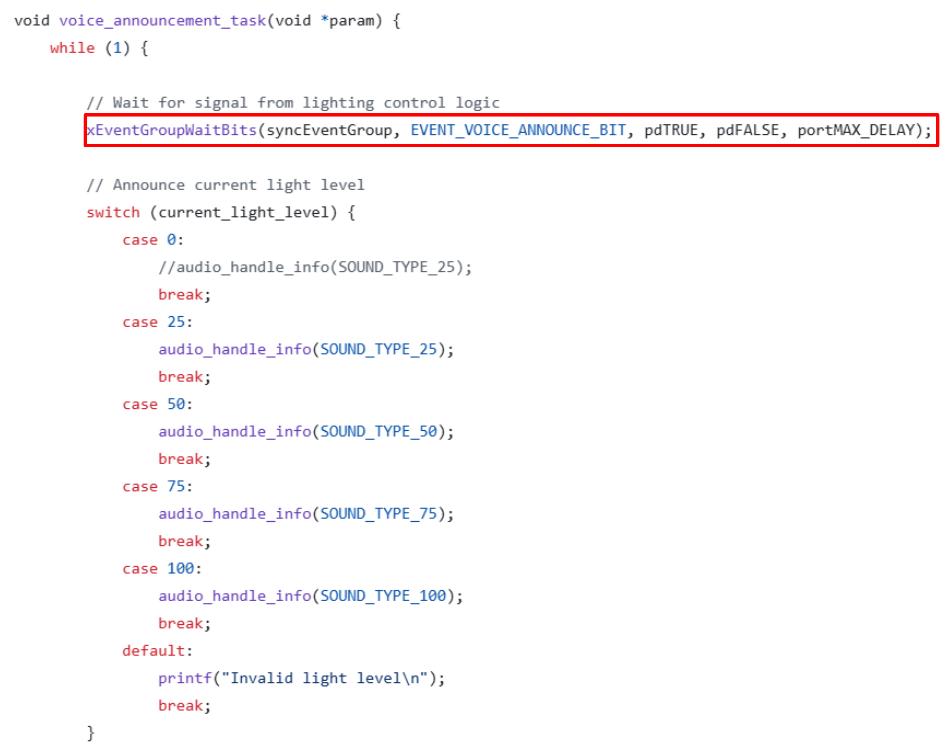


```
void init_voice_announcement() {
    syncEventGroup = xEventGroupCreate();
    if (syncEventGroup == NULL) {
        printf("Failed to create Event Group\n");
        return;
    }

    xTaskCreate(voice_announcement_task, "VoiceAnnouncement", 2048, NULL, 5, &AnnounceTask);
};
```

With these tasks and event groups created, we can then manage the way processes run using xEventGroupSetBits and xEventGroupWaitBits. These two functions control access to the mutex lock. Within the original lighting control code, after the adjustment to the LED's brightness has been made, the lock is given to the voice announcement task process. Given the light brightness level, in a similar fashion to the light adjustment, the appropriate voice announcement is made.



```
    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;
    }
}

uint8_t new_light_level = light_xor.light_pwm;
    if (new_light_level != current_light_level) {
        current_light_level = new_light_level;
        xEventGroupSetBits(syncEventGroup, EVENT_VOICE_ANNOUNCE_BIT);
    }
```



```
void voice_announcement_task(void *param) {
    while (1) {

        // Wait for signal from lighting control logic
        xEventGroupWaitBits(syncEventGroup, EVENT_VOICE_ANNOUNCE_BIT, pdTRUE, pdFALSE, portMAX_DELAY);

        // Announce current light level
        switch (current_light_level) {
            case 0:
                //audio_handle_info(SOUND_TYPE_25);
                break;
            case 25:
                audio_handle_info(SOUND_TYPE_25);
                break;
            case 50:
                audio_handle_info(SOUND_TYPE_50);
                break;
            case 75:
                audio_handle_info(SOUND_TYPE_75);
                break;
            case 100:
                audio_handle_info(SOUND_TYPE_100);
                break;
            default:
                printf("Invalid light level\n");
                break;
        }
```

In this case, it is important that each of these processes is able to access the lock on their own in order to avoid any potential race conditions as there is no guarantee that the threads running each of the tasks will execute the code in the order they are intended to. For example, if the thread running the light brightness tasks runs slower than the voice announcement thread, the voice announcement can be made before the light level is adjusted and announce the incorrect percentage. But, with the mutex lock accessed first by the light adjustment, we can guarantee that the processes are run in the order by which they are intended.

User Guide

Starting off, the kit will not be able to run without downloading the esp-idf environment, specifically the 5.1.5 version, the steps are as follows:

1. Go to https://docs.espressif.com/projects/esp-idf/en/v5.1.5/esp32/get-started/index.html#installation
2. Under the manual installation heading, go to the link that is best for your set up, either windows, mac, linux.
3. Follow the prompts in order to get the environment downloaded correctly.
4. Download the link repository to build and flash to the lcdkit panel
    a. https://github.com/agabdul/esp-dev-kits-master/tree/12a6ce8c363039635670efbf5b2204d7f5192ccd/esp32-c3-lcdkit/examples/knob_panel

Build

1. Enter *idf.py build* into your command prompt to build your code
2. When successfully built, move to flashing

Flash

1. To flash the code to your kit, enter *ify.py -p [port of kit] flash*
    a. This will send your selected code to the lcdkit

Using the lcd panel

1. Now that you have built and flashed the code, you can use the panel.
2. Turn the knob to the lighting and push the knob in. This will select the lighting option of the kit.
3. The light will turn on. To control the lighting percentage, turn the knob left or right, respectively, to increase or decrease the brightness. As you are doing so, the kit will automatically announce the current percentage, as well as, show the percentage on the screen

You have now successfully created an esp-idf environment, built and flashed respective code, and learned how to use the lighting on the lcd kit panel!