System Architecture & Concurrency Control

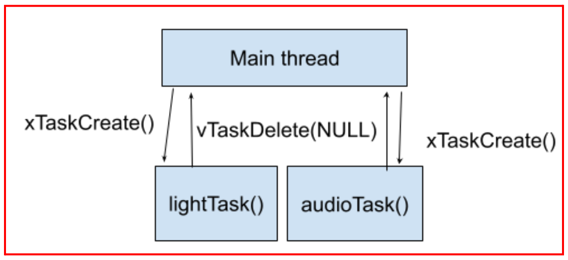
Our threaded knob_panel implementation makes use of the xTaskCreate() API provided by FreeRTOS to create tasks to run functions and return from them. Our implementation creates a total of 2 tasks when interacting with the light program on the board, one for each function of changing the light and playing a sound file.

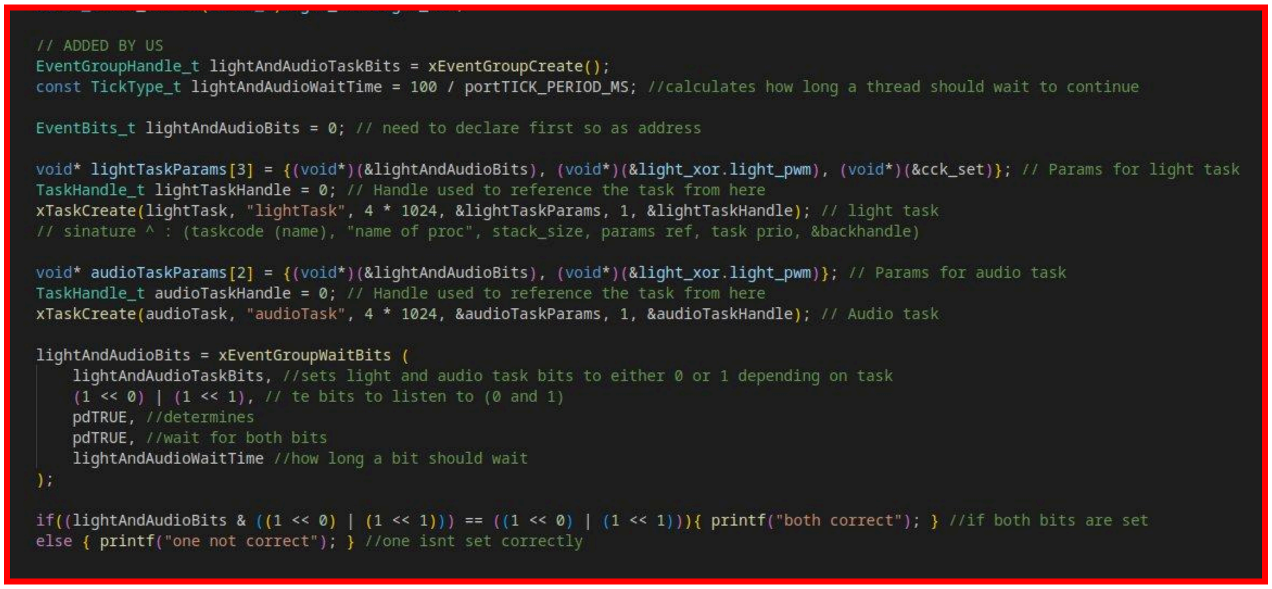
The two functions lightTask() and audioTask() are passed eventBits and lightLevel, with lightTask() also having the parameter cck_set. The arguments are given to the thread as an array of void* to avoid problems with mixing different types. Upon reaching the functions, this array is unpacked and each element is cast to the correct type. The way the functions work is by simply switching the lightLevel variable and changing the current light / sound according to its value. After doing so, the functions call for the deletion of the tasks instead of a return.

Communication and concurrency for the tasks is handled by the EventGroupHandle_t variable lightAndAudioTaskBits, where the specific bits being used are stored in the EventBits_t variable lightAndAudioBits. When a task is created, it is passed a reference to these bits so it can set them when it is complete. Each task is responsible for a different bit (the 0 and 1 bits for light and audio, respectively), the main thread waits for BOTH of these bits to be set before continuing, for up to 100ms.

The management of the event bits is not the intended method, which would be to use xEventGroupSetBits() to set the bits, as this function ensures the operation is atomic. However, when we implemented it this way, we could compile with no warnings or errors, but the board would crash before the main menu. This is a strange symptom because our code only changes things in the light program, not the main menu or boot sequence. Our implementation uses a reference to the variable storing the bits, and simply assigns them directly. This does not ensure that the operation is atomic, so the two tasks may interfere with one another's assignments to the event_bits, causing a delay up to 100ms as the main thread waits for BOTH bits to be set.

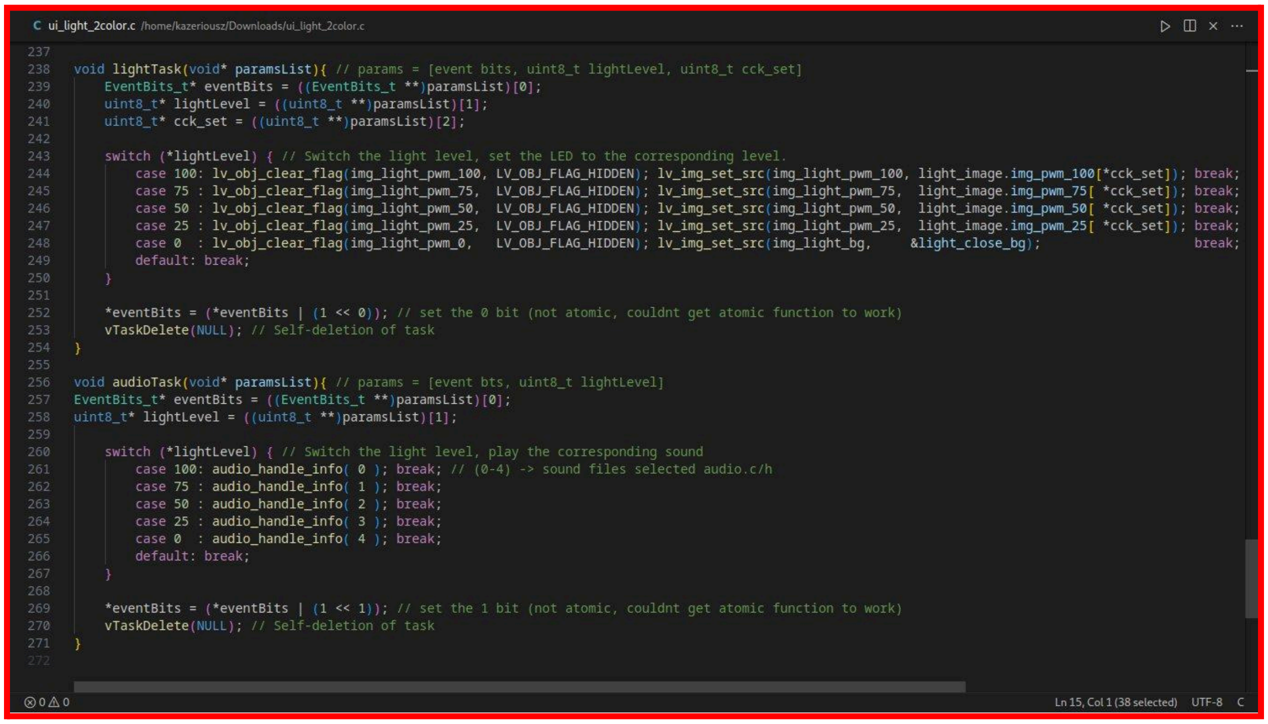
Our implementation does not have any additional features beyond changing the light level and playing a sound when turning the knob.





```c
// ADDED BY US
EventGroupHandle_t lightAndAudioTaskBits = xEventGroupCreate();
const TickType_t lightAndAudioWaitTime = 100 / portTICK_PERIOD_MS; //calculates how long a thread should wait to continue

EventBits_t lightAndAudioBits = 0; // need to declare first so as address

void* lightTaskParams[3] = {(void*)(&lightAndAudioBits), (void*)(&light_xor.light_pwm), (void*)(&cck_set)}; // Params for light task
TaskHandle_t lightTaskHandle = 0; // Handle used to reference the task from here
xTaskCreate(lightTask, "lightTask", 4 * 1024, &lightTaskParams, 1, &lightTaskHandle); // light task
// sinature ^ : (taskcode (name), "name of proc", stack_size, params ref, task prio, &backhandle)

void* audioTaskParams[2] = {(void*)(&lightAndAudioBits), (void*)(&light_xor.light_pwm)}; // Params for audio task
TaskHandle_t audioTaskHandle = 0; // Handle used to reference the task from here
xTaskCreate(audioTask, "audioTask", 4 * 1024, &audioTaskParams, 1, &audioTaskHandle); // Audio task

lightAndAudioBits = xEventGroupWaitBits (
    lightAndAudioTaskBits, //sets light and audio task bits to either 0 or 1 depending on task
    (1 << 0) | (1 << 1), // te bits to listen to (0 and 1)
    pdTRUE, //determines
    pdTRUE, //wait for both bits
    lightAndAudioWaitTime //how long a bit should wait
);

if((lightAndAudioBits & ((1 << 0) | (1 << 1))) == ((1 << 0) | (1 << 1))){ printf("both correct"); } //if both bits are set
else { printf("one not correct"); } //one isnt set correctly
```

^Screenshot showing the creation of the event group via xEventGroupCreate(), tasks via xTaskCreate(), and error checking via comparing the bits to known values at the end



```c
void lightTask(void* paramsList){ // params = [event bits, uint8_t lightLevel, uint8_t cck_set]
    EventBits_t* eventBits = ((EventBits_t **)paramsList)[0];
    uint8_t* lightLevel = ((uint8_t **)paramsList)[1];
    uint8_t* cck_set = ((uint8_t **)paramsList)[2];

    switch (*lightLevel) { // Switch the light level, set the LED to the corresponding level.
        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]); break;
        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]); break;
        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]); break;
        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]); 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;
    }

    *eventBits = (*eventBits | (1 << 0)); // set the 0 bit (not atomic, couldnt get atomic function to work)
    vTaskDelete(NULL); // Self-deletion of task
}

void audioTask(void* paramsList){ // params = [event bts, uint8_t lightLevel]
EventBits_t* eventBits = ((EventBits_t **)paramsList)[0];
uint8_t* lightLevel = ((uint8_t **)paramsList)[1];

    switch (*lightLevel) { // Switch the light level, play the corresponding sound
        case 100: audio_handle_info( 0 ); break; // (0-4) -> sound files selected audio.c/h
        case 75 : audio_handle_info( 1 ); break;
        case 50 : audio_handle_info( 2 ); break;
        case 25 : audio_handle_info( 3 ); break;
        case 0  : audio_handle_info( 4 ); break;
        default: break;
    }

    *eventBits = (*eventBits | (1 << 1)); // set the 1 bit (not atomic, couldnt get atomic function to work)
    vTaskDelete(NULL); // Self-deletion of task
}
```

C ui_light_2color.c /home/kazeriousz/Downloads/ui_light_2color.c
⊗ 0 ⚠ 0
Ln 15, Col 1 (38 selected) UTF-8 C

^Screenshot showing the two functions used in tasks, lightTask() and audioTask(). The tasks use vTaskDelete(NULL) rather than a return.