Enhanced Lighting Control with Voice Feedback

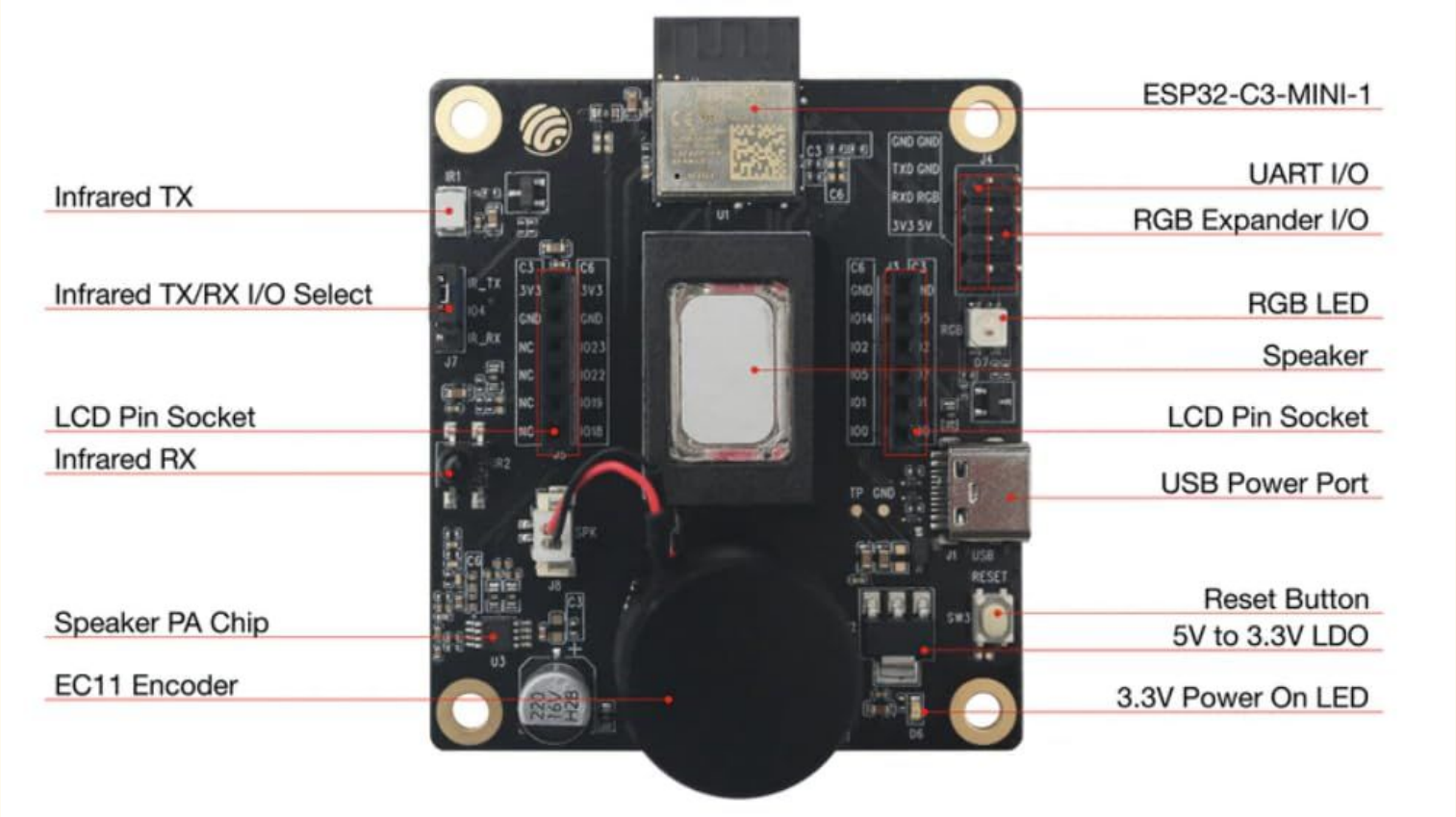
By: Murtatha Alzayadi, Mahdi Falouji, Ali Almuthafar

Project Objectives

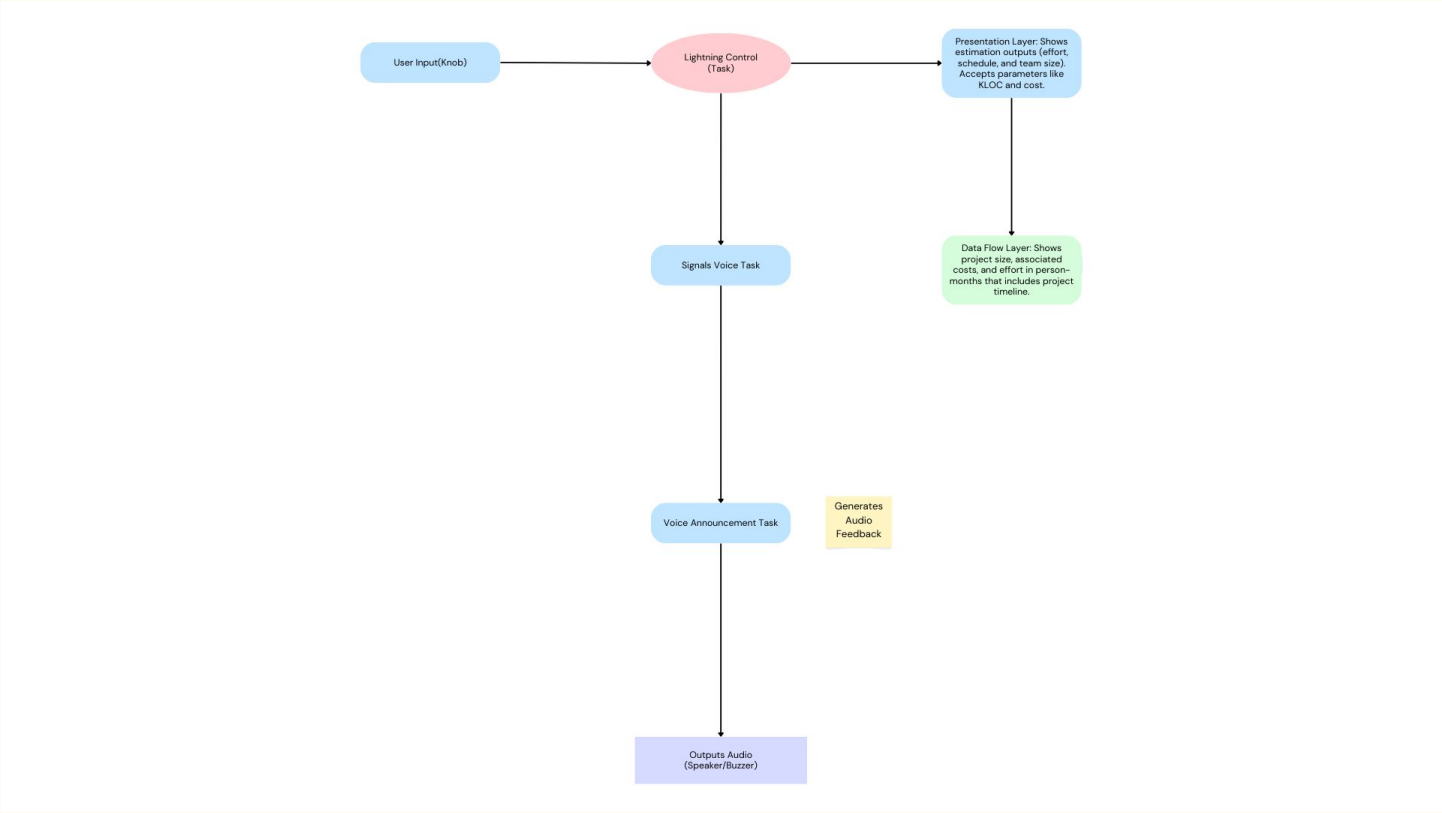
- Extend Existing Functionality: Enhance the "Lighting" case in the Knob Panel Example to include voice announcements.
- Implement Multitasking: Utilize FreeRTOS to manage concurrent tasks, ensuring that voice announcements run independently of the lighting control.
- Apply Concurrency Control: Use synchronization mechanisms such as mutexes and semaphores to manage access to shared resources, specifically the lighting level variable.
- Enhance User Interaction: Provide auditory feedback to users, improving the overall user experience.

Hardware Setup

Hardware: ESP32-C3-LCDKit, knobs, LEDs, speaker/buzzer.



System Architecture Diagram:



System Architecture Tasks:

Lighting Control Task

- **Functionality:**

  Adjust LED brightness based on knob input.

  Update the shared variable `brightness_level` to reflect the current brightness.

  Update the LCD to display the brightness percentage.

- **Tasks:**

  Read the mapped brightness level.

  Write the brightness level to a PWM signal for the LED.

  Protect shared resources (like `brightness_level`) with a mutex.

  Signal the Voice Announcement Task whenever the brightness level changes (using `xEventGroupSetBits`).

System Architecture Tasks cont.

- **Component utilized:** Knob (on ESP32-C3-LCDKit)
- **Tasks:**

  Monitor knob rotations using GPIO interrupts or analog input.

  Translate knob position into brightness levels

  - Levels included 25, 50, 75, and 100

  Signal the Lighting Control Task when the brightness level changes.

Project Objective: Voice Implementation

Objective:

- Capture user input through the knob to adjust the LED brightness

Implementation:

- The knob’s rotation was read using GPIO interrupts or an analog input pin.
- The rotation was mapped to a range (e.g., 0 to 100%) representing the LED brightness level.
- The system monitored changes in the knob's position and signaled the Lighting Control Task whenever a change was detected.

Voice Announcement Implementation:

```
esp_err_t audio_handle_info(PDM_SOUND_TYPE voice)

{

  char filepath[30];

    esp_err_t ret = ESP_OK;

    switch (voice) {

    case SOUND_TYPE_100:

        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "100_percent.mp3");

        break;

    case SOUND_TYPE_75:

        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "75_percent.mp3");

        break;

    case SOUND_TYPE_50:

        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "50_percent.mp3");

        break;

    case SOUND_TYPE_25:

        sprintf(filepath, "%s/%s", CONFIG_BSP_SPIFFS_MOUNT_POINT, "25_percent.mp3");

        break;

    case SOUND_TYPE_KNOB:
```

Voice Announcement Implementation cont.

```
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;
    default:
        ESP_LOGE(TAG, "Invalid sound type");
        return ESP_ERR_INVALID_ARG;
    }
```

Project Objective: Lighting Control Task

Objective:

- Adjust the brightness of the LED and update the brightness level variable.

Implementation:

- The task continuously monitored the `brightness_level` variable for updates.
- Adjusted the LED brightness using PWM (Pulse Width Modulation) based on the brightness level.
- Updated the LCD display with the new brightness percentage.
- Signaled the Voice Announcement Task using `xEventGroupSetBits` when never the brightness level changed.

Lighting Code Implementation:

```
void set_led_brightness(int brightness) {
    // Adjust brightness for two-color light
    ledc_set_duty(LEDC_LOW_SPEED_MODE, LEDC_CHANNEL_0, brightness);
    ledc_update_duty(LEDC_LOW_SPEED_MODE, LEDC_CHANNEL_0);
}
```

Project Objective: Voice announcement:

Objective:

- Provide auditory feedback announcing the brightness level.

Implementation:

- A separate FreeRTOS task was created to handle voice feedback (`xTaskCreate`).
- The task waited for a signal from the Lighting Control Task using `xEventGroupWaitBits`.
- Once signaled, it retrieved the `brightness_level` variable in a thread-safe manner (using a mutex).
- Audio playback was implemented using pre-recorded voice clips or generated text-to-speech to announce the brightness level.
- Played the corresponding audio file (e.g., "Brightness is 50%") through a connected speaker or buzzer.

Voice Announcement Tasks:

Functionality:

- Provide auditory feedback by announcing the brightness level when it changes.
- Run concurrently with the Lighting Control Task without blocking it.

Tasks:

- Wait for a signal (`xEventGroupWaitBits`) from the Lighting Control Task.
- Safely read the `brightness_level` variable (using a mutex).
- Generate voice announcements for the speaker or buzzer.
- Handle audio playback (e.g., using pre-recorded audio or text-to-speech).

Voice Announcements Implementation

```
d voice_announcement_task(void *arg) {
  while (1) {
    xEventGroupWaitBits(event_group, LIGHT_ADJUST_EVENT, pdTRUE, pdFALSE, portMAX_DELAY);
    uint8_t level;
    xSemaphoreTake(lighting_mutex, portMAX_DELAY);
    level = light_set_conf.light_pwm;
    xSemaphoreGive(lighting_mutex);

    ESP_LOGI(TAG, "Announcing brightness level: %d%%", level);
    switch (level) {
      case 100: audio_handle_info(SOUND_TYPE_100); break;
      case 75:  audio_handle_info(SOUND_TYPE_75); break;
      case 50:  audio_handle_info(SOUND_TYPE_50); break;
      case 25:  audio_handle_info(SOUND_TYPE_25); break;
      default:  ESP_LOGW(TAG, "No audio mapped for brightness level: %d%%", level);
    }
  }
```

Challenges Faced:

User Handling Issue: Ensure that input was processed in real time without lag.

Resolution: Reliable and correct input mechanism that allows users to interact with to control brightness.

Lighting Control issues: Ensure input was processed in the correct manner while maintaining

Avoiding performance bottlenecks caused by frequent knob adjustments.

Ensured smooth brightness transitions by controlling signals effectively.

Memory Leak:

Flash issues: Static performances were brought on the screen, and failed to visualize correct image originally

Ensured smooth brightness transitions by controlling signals effectively

Github:
https://github.com/alzayadi97/Knob_Panel

Thank You