CIS 450/ECE 478: Operating Systems

Final Project

ESP32 Lighting and Voice Announcements

Gayathri Chava, Gavin Mitchell, & Ahmad Zayour

December 13th, 2024

Introduction

In our final project, multitasking and concurrency control are integrated to improve user interaction and expand the functionality of the Knob Panel Example for the ESP32-C3-LCDKit. The main goal is to have a speech announcement function that, whenever the brightness knob is turned, will verbally convey the current illumination level. In order to ensure that lighting modifications and aural feedback run simultaneously without interfering with one another, this project uses FreeRTOS to construct a distinct job for speech announcements. System responsiveness is maintained by securely accessing shared resources using synchronization techniques like mutexes and semaphores. By adding an additional feature of the developer's choosing, the project also promotes creativity and shows how multitasking concepts may be applied creatively in embedded systems. This hands-on activity improves the user experience of a real-world IoT device while also strengthening technical expertise.

System Architecture

The architecture for the upgraded lighting control panel makes use of a FreeRTOS-enabled multitasking framework, which enables simultaneous use of speech announcement and lighting control activities. The speech announcement work and the lighting control task are the two primary duties at the center of the architecture. Using the knob panel, the lighting control task tracks user input to modify brightness levels and triggers an event to alert the voice announcement task to any brightness changes. In order to provide a responsive user experience without interfering with lighting control functionality, the speech announcement job has been assigned to alert users of the current lighting level whenever modifications are performed.

The architecture uses event groups to control communication between the activities and synchronization mechanisms like mutexes to protect access to shared variables (such the lighting level) in addition to these fundamental tasks. The speech announcement task can be performed without interference thanks to the "xEventGroupWaitBits" and "xEventGroupSetBits" routines, which notify changes in the PWM value and preserve system fluidity and responsiveness. This structure is an excellent example of the efficient use of concurrent programming techniques in embedded systems since it not only maximizes efficiency through multitasking but also improves user involvement through instantaneous audio feedback. By adding more jobs and interactions while following the same concurrency control guidelines, the possibility of more bonus features like color temperature modification or automatic lighting schedules could enhance this architecture even more.

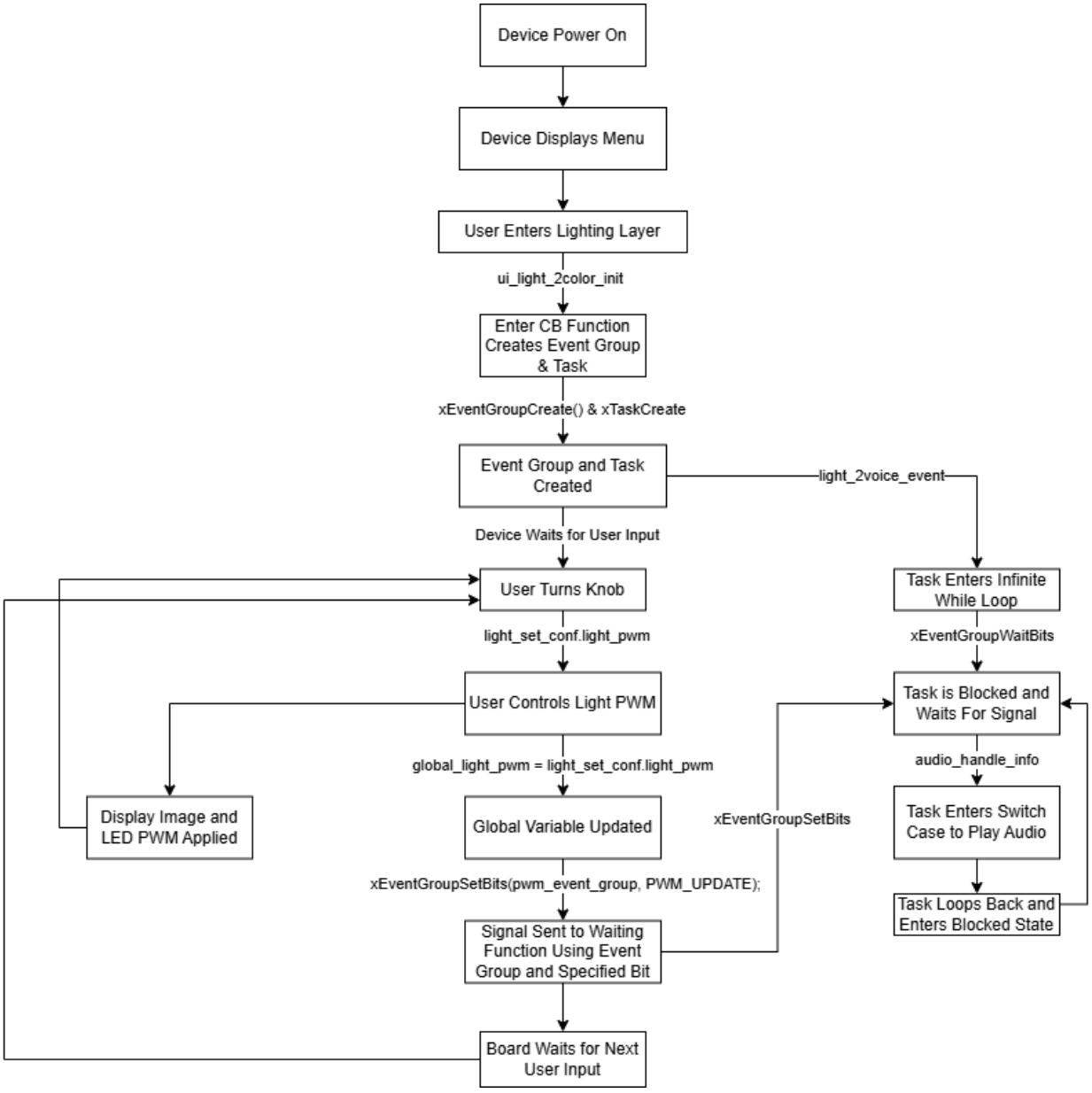

Figure 1: System Diagram

Concurrency Control Explanation

Concurrency Control Using Signals and Blocking

In this project, special attention was paid to managing concurrency to ensure proper coordination between light-intensity changes and audio feedback. As both processes require access to a shared global variable, adequate coordination procedures were necessary to avoid a race condition causing undesired behavior. We used the FreeRTOS xEventGroupWaitBits() and xEventGroupSetBits() functions as the backbone of our concurrency control. This approach enabled us to control both light intensity and its relevant sound properly. This event group (pwm_event_group) was used to signal tasks so that fog light intensity changes were registered and acted upon by the audio playback task at the most appropriate time.

Here's how it works:

- First we defined a global variable which was used to communicate the light PWM value between functions, defined a specific bit we would be setting and clearing to signal and block a function, and created an event group which would help with the signaling and blocking
- For light intensity, the system waits for the user to adjust the light PWM using the knob and the light_set_conf.light_pwm
- Once the light has been adjusted the current light PWM value is stored in the global variable global_light_pwm and then the xEventGroupSetBits(pwm_event_group, PWM_UPDATE) function is called
- The xEventGroupSetBits() function signals any xEventGroupWaitBits() function that is blocked by setting the specified bit which we defined as PWM_UPDATE inside our event group pwm_event_group
- The wait function which was blocking the rest of our code, in this project it was our switch case, then allows the subsequent code to run while turning itself back to the blocked state to wait for the next signal using pdTRUE as the third argument

- Once our switch case was unblocked it then reads our global global_light_pwm variable and plays the corresponding audio file

Task Management for Dedicated Operations

The audio playback logic was encapsulated within a dedicated FreeRTOS task (light_2voice_event). This task was responsible for:

1. Monitoring the event group for updates.
2. Playing the appropriate audio file based on the current value of global_light_pwm.

To avoid wasting resources, the task in question was created only for the duration of the active layer and was deleted as the layer being used was deactivated. This task management not only ensured that memory and computational resources were utilized effectively but also contained the effects of unused elements from executing in the system in the first instance.

Global Variable as a Communication Bridge

The global_light_pwm variable acted as the communication state between the lighting control logic and the audio system and was the perfect logical interface to synchronize both systems without a direct connection. Such a design-based architecture makes it simple to create and future-proof without complex changes.

Avoiding Race Conditions

Using concurrency control was essential for our project to avoid race conditions since we had two functions using a shared global variable. By blocking one function and making it wait until the other sends a signal after it has finished with the global variable we ensure that neither function is trying to access it at the same time which may cause unwanted behavior.

User Guide

To operate the lighting control panel, use the knob to adjust the LED brightness. The brightness can be increased by turning the knob clockwise and decreased by rotating it counterclockwise. The light intensity changes in 25% increments from 0% to 100%. The voice module will proclaim the brightness level at each increment, stating "Light level ___ percent" (for example, "Light level 25 percent"). You can quickly assess the illumination level without depending just on the visual display thanks to this audible feedback. Make sure the device is powered on before using the panel. Then, adjust the lighting to fit your surroundings and enjoy the smooth combination of visual and aural input.

Conclusion

In conclusion, this project effectively demonstrates the integration of multitasking and concurrency control in an embedded system, Utilizing FreeRTOS to enhance user interaction and functionality. By adding a speech announcement feature to the lighting control panel, the project successfully combines visual and auditory feedback to improve the user experience. The system's architecture, built around FreeRTOS tasks, event groups, and synchronization mechanisms like mutexes, ensures efficient coordination between lighting control and audio feedback, preventing race conditions and maintaining system responsiveness. This approach not only demonstrates the effectiveness of multitasking in embedded systems but also illustrates how thoughtful design and concurrency management can improve both the interactivity and performance of loT devices. This hands-on project provides valuable insights into real-world application development, laying a strong foundation for future improvements, such as new control features or more advanced user interactions.

Final Project Github Link:
https://github.com/gavmi/CIS-450-ECE-478-Final-Project

Final Project Demo/Code Walkthrough:
https://www.youtube.com/watch?v=OBWsgMTismU&t=16s