Sistema de Monitoreo de Iluminación Multitarea con FreeRTOS y LPC845

Integrantes: Agustín Azorin Barrenechea azorin.agustin231106@gmail.com

Eugenio Herrera eugenionico2006@gmail.com

Resumen

Este informe presenta el desarrollo de un sistema embebido multitarea basada en el microcontrolador NXP LPC845 y el sistema operativo en tiempo real FreeRTOS. El objetivo fue crear una solución capaz de medir la luminosidad del entorno, compararla con un valor de referencia definido por el usuario y activar respuestas visuales y sonoras en consecuencia. El sistema integra múltiples periféricos y tareas concurrentes que se comunican y sincronizan mediante semáforos, colas y mecanismos de exclusión mutua, logrando una ejecución estable y eficiente.

I. Introducción

En el marco del curso de Sistemas Embebidos dictado por el profesor Fabrizio Carlassara en la UTN, se llevó a cabo un proyecto integrador aplicando conceptos fundamentales del desarrollo sobre microcontroladores y sistemas operativos en tiempo real. La plataforma utilizada fue el NXP LPC845 Breakout, junto con FreeRTOS como entorno multitarea.

El sistema diseñado permite medir la luz ambiente mediante un sensor, visualizar información en tiempo real y controlar salidas según el valor de referencia configurado por el usuario. La arquitectura multitarea ofrecida por FreeRTOS posibilitó una gestión ordenada de los recursos del sistema, con tareas especializadas e independientes que interactúan entre sí mediante herramientas de sincronización propias del kernel.

II. Desarrollo del Sistema

A. Estructura del Código

El software fue organizado en módulos con el objetivo de mejorar la legibilidad, escalabilidad y mantenimiento del código. Se implementaron los siguientes archivos clave:

- **labels.h**: contiene definiciones de etiquetas y alias para facilitar el acceso a los pines del microcontrolador.
- **wrappers.c / wrappers.h**: encapsulan funciones del SDK, brindando una interfaz simplificada para el manejo de periféricos.
- **tareas.c / tareas.h**: definen las tareas del sistema, sus prioridades, tamaños de pila y los objetos de sincronización utilizados (semaphores, queues, mutexes).
- **main.c**: contiene la configuración principal, incluyendo la inicialización del sistema a 30 MHz, la configuración de periféricos y la creación de tareas mediante xTaskCreate (). Al finalizar, se inicia el planificador con vTaskStartScheduler ().

B. Tareas Definidas

El sistema fue dividido en tareas específicas, cada una con una responsabilidad concreta. Las principales son:

- **tsk_init**: configuración general del sistema
- **tsk_adc**: lectura y procesamiento del valor analógico proveniente del potenciómetro
- **tsk_BH1750**: adquisición de datos del sensor de luz
- **tsk_setpoint**: ajuste del valor de referencia (setpoint) mediante botones
- **tsk_display_write**: visualización de datos en el display
- **tsk_display_change**: gestión de entradas por botones
- **tsk_control**: comparación entre la luminosidad medida y el setpoint
- **tsk_buzzer**: activación del buzzer según condiciones
- **tsk_led_azul**: control del brillo del LED azul
- **tsk_leds_control**: gestión de múltiples LEDs
- **tsk_console_monitor**: envío de datos de monitoreo a la consola

III. Arquitectura del Sistema

El hardware utilizado se basa en el microcontrolador NXP LPC845 Breakout, que incorpora un núcleo ARM Cortex-M0+ a 30 MHz. Entre los recursos periféricos empleados se incluyen:

- **Sensor de luz BH1750** (comunicación I2C)

- **Display de 7 segmentos** (multiplexado)
- **Potenciómetro RV22** (lectura por ADC)
- **Botones físicos (USER, S1, S2)**
- **Sensor infrarrojo CNY70**
- **LED tricolor y LED azul** (controlados por PWM)
- **Buzzer activo**

El sistema fue desarrollado en MCUXpresso IDE utilizando el SDK oficial de NXP para el LPC845. La inicialización del reloj se realizó mediante la función BOARD_BootClockFRO30M.

IV. Funcionamiento General

El comportamiento del sistema responde a un flujo de trabajo distribuido entre tareas que operan de forma concurrente bajo FreeRTOS:

1. **Medición de luz ambiente**: La tarea tsk_BH1750 obtiene datos del sensor BH1750 de forma continua.
2. **Configuración del valor deseado**: El usuario define un valor de referencia mediante botones, gestionados por tsk_setpoint.
3. **Visualización**: Las tareas tsk_display_write y tsk_display_change manejan el contenido mostrado en el display según la selección del usuario.
4. **Control de salida de luz**: La intensidad del LED azul se ajusta con un potenciómetro, cuya lectura es interpretada por tsk_led_azul.
5. **Monitoreo**: La tarea tsk_console_monitor permite visualizar el estado del sistema en la consola de depuración.

V. Conclusiones

El desarrollo de este proyecto permitió poner en práctica de manera integral los conocimientos adquiridos durante el curso. Se logró una mejor comprensión del funcionamiento de FreeRTOS y su utilidad en la implementación de sistemas embebidos multitarea.

El trabajo en equipo fue fundamental para superar los desafíos encontrados durante el proceso, favoreciendo el aprendizaje colectivo y la resolución colaborativa de problemas. Esta experiencia consolidó nuestras habilidades en programación en C

aplicada a microcontroladores y nos brindó herramientas concretas para futuros desarrollos en sistemas de tiempo real.