Trabajo Integrador
Aplicaciones de Sistemas Operativos en Tiempo Real

Facundo Spagnoletta, Pablo Osores, Facundo Ledesma
Escuela de Educación Estatal Técnica N°7
Quilmes, Argentina
facundospagnoletta@impatrq.com
pabloezequielosores@impatrq.com
facundoezequielledesma@impatrq.com

***Abstract*— En este trabajo se desarrolló una aplicación en FreeRTOS sobre un microcontrolador LPC845. El sistema integra cinco tareas concurrentes que gestionan la lectura de un sensor de luz BH1750, la interacción mediante botones físicos y la visualización de datos en un display de 7 segmentos. Se controló un LED tricolor y un buzzer en función de la luminosidad y un sensor infrarrojo, empleando colas y semáforos para sincronización. Los resultados evidencian un manejo adecuado de tareas en tiempo real y un comportamiento estable para así tener un sistema operativo.**

I. INTRODUCCIÓN

Los sistemas operativos en tiempo real (RTOS) son esenciales en sistemas embebidos que requieren respuesta rápida y detallada para funciones y tareas específicas. FreeRTOS es una herramienta ampliamente utilizada para implementar multitarea en microcontroladores, permitiendo un control preciso de periféricos y procesos concurrentes.

El problema planteado consiste en gestionar de forma simultánea múltiples periféricos conectados a un LPC845, garantizando tiempos de respuesta estables. Este trabajo tiene como objetivo principal implementar una aplicación de multitasking que integre sensores, pulsadores, diferentes tipos de salida y una interfaz de usuario.

II. DESARROLLO DE CONTENIDOS

A. Objetivos Específicos

El desarrollo de este proyecto se centró en la consecución de los siguientes objetivos técnicos:

- Diseñar e implementar un sistema operativo bajo una arquitectura multitasking, utilizando el sistema FreeRTOS para gestionar la concurrencia en tiempo real a través de tareas dedicadas.
- Integrar y controlar un conjunto de periféricos desde el microcontrolador LPC845, abarcando:
    - Protocolos de Comunicación Serie: I²C para la interfaz con el sensor de luminosidad BH1750 y UART para el reporte de datos a una terminal.
    - Periféricos de Conversión y Temporización: El Conversor Analógico-Digital (ADC) para la lectura del potenciómetro y el SCTimer para la generación de señales de Modulación por Ancho de Pulso (PWM).
    - Manejo de Eventos Asíncronos: El periférico PINT para la captura de interrupciones externas generadas por pulsadores y sensores.
- Implementar un mecanismo de comunicación y sincronización inter-tareas (IPC) robusto y eficiente, empleando:
    - Colas (Queues) para el paso de datos de forma segura y desacoplada entre las tareas productoras y consumidoras.
    - Semáforos Binarios (Semaphores) para la sincronización entre las Rutinas de Servicio de Interrupción (ISRs) y las tareas de la aplicación, garantizando una respuesta de baja latencia a los eventos de hardware.

B. Metodología y Configuración del Sistema

El hardware utilizado incluyó un microcontrolador LPC845, un sensor de luminosidad BH1750, un display de 7 segmentos, un LED tricolor, un buzzer, un sensor infrarrojo y un potenciómetro RV22. El software se desarrolló en Visual Studio Code con la extensión de MCUXpresso IDE utilizando el SDK del LPC845_Breakout.

El sistema se programó mediante FreeRTOS, creando tareas independientes para cada función. Para la comunicación entre tareas se emplearon colas y semáforos , asegurando sincronización sin bloqueos prolongados.

La arquitectura de software se estructuró en dos capas principales:

1. Se desarrolló un conjunto de funciones wrapper para encapsular las llamadas de bajo nivel del SDK. Esto permitió que el código de la aplicación operara a un nivel lógico (wrapper_bh1750_read(), wrapper_pwm_update_rled()), entre otras, aumentando su legibilidad y portabilidad.
2. La lógica del sistema se distribuyó en tareas de FreeRTOS, cada una con una responsabilidad única (ej. task_bh1750 para medir la luz, task_display para la multiplexación del display). Este modelo garantiza que cada funcionalidad se ejecute en su propio hilo de control sin interferir con las demás.

C. Implementación de las Tareas en FreeRTOS

1. Tarea de Lectura de Luminosidad: Lee valores del BH1750 y los convierte en porcentaje de 0% a 100% (30000 lux = 100%).
2. Tarea de Manejo de Setpoint: Incrementa o decrementa el setpoint al presionar S1 o S2.
3. Tarea de Display: Alterna entre mostrar luminosidad o setpoint con el botón USER; activa el punto decimal en modo setpoint.
4. Tarea de Control de LED Tricolor: Ajusta el brillo del LED rojo o azul proporcionalmente a la diferencia entre luminosidad y setpoint.
5. Tarea de Buzzer y Sensor IR: Activa el buzzer cuando se detecta obstrucción en el sensor IR y regula intensidad con RV22.
6. Tarea de Consola: Muestra cada segundo: tiempo transcurrido, luminosidad, setpoint y brillo del LED.

III. RESULTADOS Y VALIDACIÓN EXPERIMENTAL

La validación del sistema se realizó mediante pruebas funcionales sobre el hardware implementado, observando el comportamiento del sistema en respuesta a los diferentes estímulos de entrada. Los resultados obtenidos confirmaron el cumplimiento de todos los requisitos funcionales y de rendimiento.

- Rendimiento en Tiempo Real y Concurrencia: Se verificó que todas las tareas se ejecutaron de forma concurrente sin interferencias perceptibles. La multiplexación del display de 7 segmentos (task_display) y el control de intensidad del LED RGB mediante PWM (task_tricolor, task_Led_Azul_rv22) operaron de forma fluida y estable, incluso durante la recepción de eventos asíncronos, lo que valida la correcta gestión de los contextos y prioridades por parte del planificador de FreeRTOS.
- Respuesta a Eventos Asíncronos: La respuesta a la pulsación del botón USER y a la activación del sensor infrarrojo fue inmediata y determinista. Al utilizar un modelo basado en interrupciones y semáforos, se eliminó la latencia asociada al sondeo (polling), demostrando la eficacia de este patrón de diseño para sistemas reactivos.
- Validación del Mecanismo de Exclusión Mutua: La estrategia de coordinación para el control del LED RGB fue uno de los puntos críticos a validar. Se comprobó experimentalmente que al superar el umbral del potenciómetro, la tarea task_Led_Azul_rv22 suspendía correctamente a task_tricolor, tomando control exclusivo del LED azul sin conflictos. Al bajar del umbral, la reanudación de task_tricolor devolvía el control al modo automático de forma limpia y sin condiciones de carrera. Este resultado valida la estrategia de suspensión/reanudación como un método eficaz para la exclusión mutua en este contexto.
- Precisión del Control y Reporte: El sistema de control de la luminosidad operó según lo esperado, ajustando la intensidad del LED rojo o azul de forma proporcional a la diferencia entre la medición y el setpoint. Simultáneamente, la task_monitoreo, operando a baja prioridad, envió los reportes de estado a la consola UART con la periodicidad programada de un segundo, sin impactar negativamente el rendimiento de las tareas de control de mayor prioridad.

TABLA I

Resultados Principales del Sistema

	Apariencia (en Time New Roman ó Times)		
Componente	**Condición**	**Respuesta esperada**	**Respuesta observada**
LED rojo	Lux > Setpoint	**Brillo proporcional**	Correcta
LED azul	Lux < Setpoint	**Brillo proporcional**	Correcta
Buzzer	Sensor IR tapado	**Activado inmediato**	Correcta
Display	Botón USER	**Alterna luminosidad/setpoint**	Correcta
Pulsador S1	Posición cerrado	**Subir setpoint**	Correcta
Pulsador S2	Posición cerrado	**Bajar setpoint**	Correcta

IV. CONCLUSIONES

El desarrollo permitió comprobar la viabilidad de implementar múltiples tareas concurrentes en FreeRTOS con un LPC845. Se cumplieron todos los objetivos propuestos, logrando sincronización confiable entre tareas y una correcta interacción con los periféricos. A su vez, pudimos desarrollar habilidades sobre el uso de FreeRTOS y sus complementos, como las Queues, Semaphores y las tareas.