Trabajo Integrador

Aplicaciones de Sistemas Operativos en Tiempo Real

UTN FRA

Baza Victoria Josefina
Escuela Técnica N°7 de Quilmes
victoriajosefinabaza@gmail.com

Blasco Mauricio
Escuela Técnica N°7 de Quilmes
sirmauriciob@gmail.com

***Resumen*—Este informe describe el desarrollo de un sistema embebido multitarea basado en el microcontrolador NXP LPC845 y el sistema operativo FreeRTOS. Se integraron diversos periféricos para medir la luminosidad ambiente, compararla con un valor de referencia configurable por el usuario y accionar salidas visuales y sonoras en respuesta. La implementación incluyó sincronización mediante semáforos, colas y exclusión mutua, logrando una ejecución concurrente estable.**

I. Introducción

En el marco del curso de Sistemas Embebidos, dictado por el *prof.* Fabrizio Carlassara en la UTN, se desarrolló un trabajo integrador aplicando los conceptos aprendidos sobre **FreeRTOS** y el uso del microcontrolador **NXP LPC845 Breakout**. El proyecto consistió en el diseño e implementación de un sistema embebido de monitoreo y control de iluminación ambiente, integrando sensores, actuadores y una interfaz de usuario. El uso de un sistema operativo en tiempo real permitió explorar una arquitectura multitarea, con control preciso del CPU y de los recursos periféricos. FreeRTOS, por su parte, proporcionó las herramientas necesarias para la sincronización y comunicación entre tareas, asegurando concurrencia, respuesta rápida e independencia funcional. El sistema final incluyó un sensor de luz BH1750, un potenciómetro para regular el brillo de un LED, botones para definir un setpoint deseado y un display para visualizar información en tiempo real.

II. Desarrollo de contenidos

A. Desarrollo del proceso

Para estructurar el desarrollo del sistema embebido, se diseñaron y organizaron múltiples módulos del código fuente con fines de escalabilidad, legibilidad y mantenimiento; se creó un archivo denominado *labels.h*, destinado a centralizar la definición de etiquetas y alias que facilitan el acceso a los pines del microcontrolador, promoviendo una codificación más clara y eficiente.

Los archivos *wrappers.c* y *wrappers.h* encapsulan funciones del SDK, brindando una interfaz simplificada para la inicialización y control de periféricos, permitiendo una mayor portabilidad del código.

Los archivos *tareas.c* y *tareas.h* definen la estructura del sistema multitarea. En ellos se establecen los tamaños de pila, las prioridades y los prototipos de las funciones correspondientes a cada tarea. Además, se implementa la creación e inicialización de semáforos y colas requeridas para la sincronización y comunicación entre tareas. Este diseño modular favorece la claridad y facilita el escalado de funcionalidades bajo el sistema operativo FreeRTOS.

El archivo *main.c* contiene la configuración inicial del sistema, incluyendo la fijación del reloj a 30 MHz y la inicialización de la consola de depuración. Posteriormente, se crean las tareas mediante la función *xTaskCreate(),* asignando a cada una su función correspondiente, prioridad, tamaño de pila, nombre y puntero de control. Estas tareas gestionan distintos módulos funcionales del sistema, tales como sensores, display, control, consola y salidas. Finalmente, la función *vTaskStartScheduler()* da inicio al planificador de FreeRTOS, habilitando la ejecución concurrente de las tareas definidas.

B. Listado de tareas:

A continuación, se enlistan las tareas declaradas en el archivo main.c, junto con su función principal dentro del sistema:

Tarea	Función principal
tsk_init	Inicialización general del sistema
tsk_adc	Lectura y procesamiento del valor del ADC
tsk_BH1750	Adquisición de datos desde el sensor de luz
tsk_setpoint	Ajuste del valor de luminosidad (por el usuario)
tsk_display_write	Escritura de valores en el display
tsk_display_change	Manejo de entradas de los botones
tsk_control	Comparación entre el valor leído y el setpoint
tsk_buzzer	Control del buzzer
tsk_led_azul	Control del LED azul
tsk_leds_control	Control de LEDs múltiples
tsk_console_monitor	Supervisión de datos a través de la consola

- ***tsk_init*:** Inicialización general del sistema
- ***tsk_adc*:** Lectura y procesamiento del valor del ADC
- ***tsk_BH1750*:** Adquisición de datos desde el sensor de luz
- ***tsk_setpoint*:** Ajuste del valor de luminosidad (por el usuario)
- ***tsk_display_write*:** Escritura de valores en el display
- ***tsk_display_change*:** Manejo de entradas de los

botones
- ***tsk_control*:** Comparación entre el valor leído y el setpoint
- ***tsk_buzzer*:** Control del buzzer
- ***tsk_led_azul*:** Control del LED azul
- ***tsk_leds_control*:** Control de LEDs múltiples
- ***tsk_console_monitor*:** Supervisión de datos a través de la consola serie

Cada tarea se declara con un propósito funcional definido, permitiendo una ejecución concurrente eficiente.

C. *Arquitectura del Sistema*

El sistema fue implementado sobre un microcontrolador NXP LPC845 Breakout. Este chip ARM Cortex-M0+ opera a 30 MHz y cuenta con periféricos como ADC, SCTimer/PWM, UART, I2C, PINT para interrupciones externas, y múltiples GPIOs. La arquitectura física incluye:

- Sensor de luminosidad BH1750 (*conectado por I2C*)
- Display de 7 segmentos (*múltiplexado*)
- Potenciómetro RV22 (*conectado al ADC*)
- Botones USER, S1 y S2
- Sensor infrarrojo CNY70
- LED tricolor y LED azul (*controlados por PWM*)
- Buzzer activo

El sistema cuenta con inicialización temprana del reloj a 30 MHz (*BOARD_BootClockFRO30M*). El código fue desarrollado en MCUXpresso IDE utilizando el SDK oficial del LPC845.

III. Flujo de trabajo

El sistema implementado sigue un flujo de trabajo secuencial y multitarea, coordinado mediante el sistema operativo en tiempo real FreeRTOS. Se detalla el funcionamiento general de las tareas involucradas:

1. La tarea *tsk_BH1750* se encarga de medir continuamente la intensidad de luz ambiente utilizando el sensor digital BH1750.
2. El valor de referencia o setpoint es ajustable por el usuario mediante botones físicos, a través de la tarea *tsk_setpoint*.
3. La visualización en el display de 7 segmentos alterna entre el valor medido y el setpoint, en función del estado de un botón de selección, gestionado por las tareas *tsk_display_change* y *tsk_display_write*.
4. La intensidad del LED azul se regula mediante un potenciómetro, cuya lectura analógica es procesada por la tarea *tsk_led_azul*.
5. Por último, la tarea *tsk_console_monitor* registra y muestra en la consola de depuración los datos relevantes del sistema, permitiendo el monitoreo en tiempo real.

IV. Conclusiones

La realización y desarrollo de este trabajo integrador nos permitió consolidar y profundizar los conocimientos adquiridos en el curso de Sistemas Embebidos. A través de este proyecto, logramos un mejor dominio del lenguaje C aplicado a microcontroladores y sistemas en tiempo real.

Si bien la experiencia presentó desafíos, ninguno fue insuperable. Trabajando en equipo, pudimos coordinarnos eficazmente y apoyarnos mutuamente, recurriendo a la consulta con nuestros pares cuando surgieron dudas. Esta colaboración resultó clave para superar obstáculos y avanzar en el desarrollo del sistema.

En definitiva, esta experiencia nos dejó un aprendizaje significativo para la implementación de tareas multitarea con FreeRTOS y reforzó nuestras capacidades para afrontar proyectos futuros con herramientas similares.

Referencias

[1] Todas las obtenidas del repositorio propio de GitHub.