IOT – Sistemas Embebidos

Laboratorio 3

Integrantes del grupo:

- Nicolás Raposo
- Martín Da Rosa
- Rafael Durán

Docentes: Nicolás Calarco y Pablo Alonso

12/6/2025

Índice

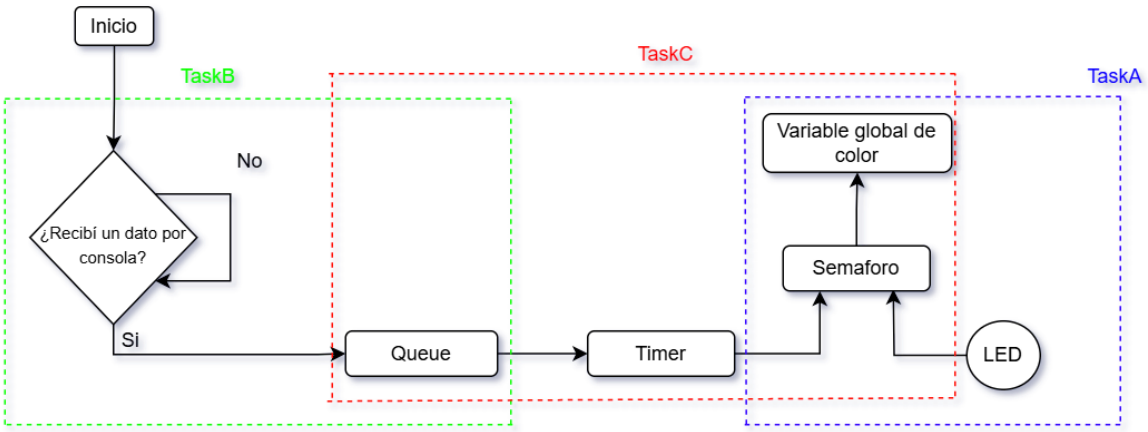
Introducción

En este laboratorio se creará un programa utilizando FreeRTOS que permite modificar el color de un LED que parpadea de forma constante escribiendo por consola un color de una lista determinada y un tiempo en que el comando se hará efectivo. Por ejemplo, si se introduce en la consola “rojo 10, verde 20”, a los 10 segundos el led empezará a parpadear en rojo y a los 20 segundos cambiará a verde.

Diseño

Diagrama de Flujo

Antes de empezar con la programación, se realizó el siguiente diagrama de flujo para organizar el trabajo y ver como las tasks interactúan entre sí con las diferentes herramientas de FreeRTOS.



En primer lugar, la TaskB se encarga de revisar constantemente la consola para detectar el color y el tiempo que el usuario indique, en caso de recibir algo lo envía a una queue.

La TaskC se encarga de estar mirando la queue, cuando la TaskB reciba el dato del usuario y lo coloque en la queue, la TaskC lo verá, tomando los datos y creando un timer con el tiempo indicado. Cuando este timer termine, la tarea tomará un semáforo mutex para acceder a una variable compartida de color que contiene los valores de RGB y los actualizará según el usuario haya elegido devolviendo después el semáforo.

Por último, la TaskA toma el mutex y accede de a la variable global antes mencionada para cambiar el color usado en el parpadeo del LED.

Pseudocódigo

Aparte del diagrama de flujo, se escribieron las tasks en pseudocódigo:

Pseudocodigo Task A:

```
Task_A()
{
    while(true){
        semaforo = obtener_semaforo();

        color_led = obtener_color();
        prender_led(color_led);

        liberar_semaforo(semaforo);
    }
}
```

Pseudocodigo Task B:

```
Task_B()
{
    queue = obtener_queue();

    while(nuevo_evento = obtener_desde_uart()) {

        r,g,b = mapear_color(nuevo_evento.color)

        queue.send({
            r: r,
            g: g,
            b: b,
            tiempo: nuevo_evento.tiempo
        });

}
```

Pseudocodigo Task C:

```
Task_C()
{
    queue = obtener_queue();

    while( (color_led,tiempo) = queue.recieve() ){

        esperar(tiempo).luego(

            semaforo = obtener_semaforo()
            setear_color(color_led)
            liberar_semaforo()
        )

    }

}
```

Implementación

TaskA

Para implementar la TaskA se siguió la lógica antes mostrada, sin embargo, como se puede ver en el repositorio correspondiente, fue necesario crear una función que haga la conexión entre las variables locales de la tarea con las variables global del main.c mediante punteros. Esta función es `task_a_set_shared_resources(color, semáforo, LED)`, donde color, semáforo y LED son las 3 variables que hay que conectar con main. Posteriormente, como fue explicado en el pseudocódigo, se toma el semáforo y se hace parpadear al LED, en caso de no poder tomar el semáforo se apaga el LED y se imprime un mensaje en consola para poder detectar el problema.

Como observación extra, se vio que colocando menos de 1800 de stack en esta tarea se producía stackoverflow, dado que el uso de memoria en este laboratorio no es un problema se asignó 3072 para evitar cualquier error.

TaskB

La TaskB debe recibir por consola alguno de las entradas de la lista siguiente, donde x es el tiempo en milisegundos en que el cambio de color será efectivo. Hay que presionar la tecla ENTER luego de escribir alguno de los siguientes para que la entrada sea procesada:

- `red x`
- `green x`
- `blue x`
- `yellow x`
- `violet x`
- `white x`

Asimismo se pueden encadenar varios colores de la siguiente forma: “`red 1000, green 5000, white 10000`” esto hará que al segundo de tocar la tecla ENTER el LED cambie a rojo, a los 5 segundos se pondrá verde y a los 10 segundos en naranja. En caso de escribir un negativo o flotante se mostrará un mensaje por consola y se pedirá que se escriba de vuelta el color y el tiempo, lo mismo sucede si se coloca una cola de valores donde uno es inválido.

TaskC

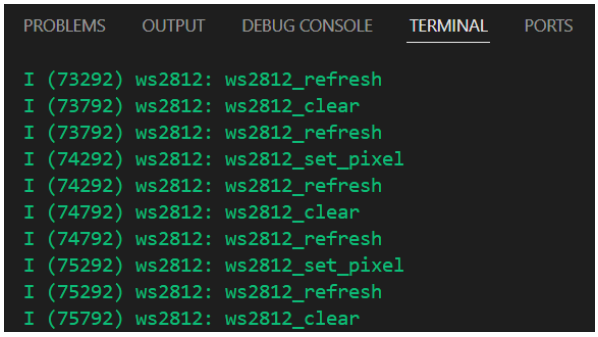
En primer lugar, la TaskC tiene función similar a la de la TaskA en este caso llamada `task_C_set_shared_resources(color, semáforo)` que hace la conexión con el color y el semáforo globales. Aparte de esto, igual que en el pseudocódigo, se mira la queue constantemente, cuando se recibe algo, se guarda espacio en memoria para la variable del color que contiene los valores RGB y se crea un timer con el tiempo indicado por la queue. Cuando este timer termine, llamará a la función `timer_callback(timer)` donde se toma al semáforo mutex y se actualiza el color global, posteriormente se devuelve el mutex y se libera la memoria.

Main

Dado que se pidió que las tareas anteriores se implementen como librerías, en el archivo main solamente se inicializan las variables globales como el color, el LED y también se crean la queue y el semáforo. Por último se llaman a las funciones de configuración de recursos compartidos mencionadas para hacer la conexión y se inician las tres tareas.

Consideraciones

Como se mencionó en la implementación, el formato para introducir el color y tiempo es: “[Color] [Tiempo], [Color] [Tiempo], [Color] [Tiempo]...” Esto debe introducirse por la consola misma de VScode por donde se flashea el código a la placa. Sin embargo, dado que la librería usada para prender el led hace un print por esa consola cada vez que el led prende, puede resultar un poco confuso el uso, puesto que al escribir por ejemplo: “white 10”, la consola estará siendo constantemente actualizada con los prints del encendido del LED. Esto podría evitarse utilizando alguna contra consola como PuTTY. Los prints pueden verse en la imagen de abajo:



Resultados y Conclusiones

En conclusión, se logró la correcta comunicación entre tasks usando las queues, timers y mutex solicitados, es decir, el LED se prende en el momento y con el color indicados. También se integraron correctamente las tareas como librerías externas conectadas al main.c, A pesar del pequeño inconveniente de la sobreescritura constante en consola por parte del LED, el programa cumple con los objetivos planteados por los docentes en la letra del laboratorio.

Por último, algo que destacamos con la arquitectura orientada a queues y mensajes, es que al coordinar la estructura del mensaje que produciría y consumiría cada tarea, pudimos desarrollar de manera independiente y paralela las tres tareas mientras seguíamos la convención previamente definida.