UTN Universidad Tecnológica Nacional
Facultad Regional Tucumán

Departamento Electrónica

Técnica Digitales II

Actividad de Formación
Practica 3

Tema: Funciones no bloqueantes, aplicación con SysTick en STM32CubeIDE, programación de microcontroladores.

Alumnos:

Angelillo Alessandro

Plaate Iván

Torres Juan

Hualampa Leonel Javier

Profesor:
Ing. Rubén Darío Mansilla

ATTP:
Ing. Lucas Abdala

Año: 2025

Funciones de retardo no bloqueantes: En esta activada hemos visto las funciones de retardo no bloqueantes, comprendimos su importancia y utilidad en el desarrollo de aplicaciones embebida. A diferencia de las funciones de retardo tradicionales, como la HAL_Delay(), que nos detiene las ejecución del programa durante el tiempo que le especificamos (Ej: 200ms) en este tiempo el micro se detiene hasta que pasen los 200ms, mientras que las funciones no bloqueantes permiten que el microcontrolador continué realizando otras tareas mientras se espera.

En qué se basan las funciones de retardo no bloqueantes: Las funciones de retardo no bloqueantes se basan en el uso de un temporizador (en nuestro caso, el SysTick) y la lectura continúa de su valor actual. En lugar de entrar en un bucle de espera, se registra el tiempo de inicio y periódicamente, se verifica si la diferencia entre el tiempo actual y el tiempo de inicio es mayor o igual al tiempo de retardo deseado. Si el tiempo transcurrido es suficiente, se realiza la acción correspondiente y se actualiza el tiempo de inicio para el próximo retardo.

El funcionamiento se puede resumir en los siguientes pasos:

1) **Inicialización:** Se configura un temporizador (SysTick) y se almacena el tiempo actual como punto de referencia inicial.
2) **Verificación:** En cada iteración del bucle principal, se compara el tiempo actual con el tiempo de referencia.
3) **Acción:** Si la diferencia entre el tiempo actual y el tiempo de referencia es mayor o igual al retardo deseado, se ejecuta la tarea correspondiente y se actualiza el tiempo de referencia.

Ventajas del uso de este tipo de funciones en una aplicación comercial:

El uso de funciones de retardo no bloqueantes ofrece varias ventajas significativas en una aplicación comercial:

- **Mejora de la capacidad de respuesta:** Al no bloquear la ejecución, el microcontrolador puede responder rápidamente a eventos externos, como interrupciones o entradas de datos entrada del usuario. Esto es crucial en aplicaciones en tiempo real donde la capacidad de respuesta es fundamental.

- **Mayor eficiencia:** Permite realizar múltiples tareas, optimizando el uso de los recursos del microcontrolador y mejorando la eficiencia general del sistema.
- **Escalabilidad:** Facilita la adición de nuevas funcionalidades a la aplicación sin comprometer el rendimiento, ya que las tareas se ejecutan de forma concurrente y no secuencial.
- **Mejor experiencia de usuario:** En aplicaciones interactivas, el uso de funciones no bloqueantes garantiza una experiencia de usuario más fluida y sin interrupciones, ya que la interfaz de usuario se mantiene receptiva en todo momento.
- **Optimización de recursos Ahorro de consumo:** Dado que se evita el uso de delay, el cual consume ciclos de reloj y mantiene ocupado al microcontrolador.

Aplicaciones desarrolladas usando funciones no bloqueantes:

App 1.1: Desarrolle una aplicaci´on que encienda y apague de manera secuencial los tres leds onboard de la placa de desarrollo. La secuencia debe encender 200 ms y apagar 200 ms cada led comenzando por el LED1 (Green), continuando con el LED2 (Blue) y luego el LED3 (Red) para volver a iniciar con el LED1. La aplicaci´on debe ser de car´acter general, de manera que pueda extenderse a una cantidad mayor de leds con m´ınimas modificaciones, por este motivo se sugiere que use un vector para el manejo de los Leds.

- **Link:** App1_1
- **Alumno:** Angelillo Alessandro
- **Observaciones:** La implementación no bloqueante con delayRead permitió controlar la secuencia de LEDs sin detener la ejecución del programa, lo que mejora la escalabilidad del código. Una dificultad fue organizar la transición entre LEDs utilizando el índice currentLed, ya que se requería reiniciar la secuencia al alcanzar el valor máximo (NUM_LEDS). Esto se resolvió aplicando una condición que reinicia el contador a cero. Una recomendación es centralizar las definiciones como NUM_LEDS y DELAY_MS en un archivo de cabecera común para evitar inconsistencias y facilitar la reutilización en futuros proyectos.

App 1.2: Desarrolle una aplicaci´on que utilice el pulsador onboard de la placa de desarrollo para alternar entre dos secuencias diferentes. La aplicaci´on inicia con la secuencia de la App 1.1 y, cuando se presione el pulsador, dicha secuencia debe invertirse y continuar, de manera que, cada vez que se presione el pulsador la secuencia actual se invierta. La aplicaci´on debe ser de

car´acter general, por lo que aplica la misma recomendaci´on para el punto anterior.

- **Link:** App_3_2
- **Alumno: Hualampa Leonel**
- **Observaciones:** La integración del pulsador con lógica de anti-rebote no bloqueante permitió controlar el cambio de dirección en la secuencia de LEDs de manera fluida, sin afectar la ejecución del resto del programa. El principal desafío fue manejar correctamente el estado anterior del botón y el temporizador de debounce, ya que cualquier error en esta lógica podía provocar múltiples cambios de dirección con una sola pulsación. Esto se resolvió combinando una variable de estado (debounceActive) con un delay_t dedicado al anti-rebote. Una recomendación es abstraer la gestión del pulsador en un módulo independiente, de forma que el main.c quede más limpio y la lógica pueda reutilizarse en otros proyectos.

App 1.3: Desarrolle una aplicaci´on que utilice el pulsador onboard de la placa de desarrollo para alternar entre cuatro secuencias diferentes. La app iniciar´a con la secuencia 1 hasta que se presione el pulsador y pase a la secuencia 2, luego de presionar de nuevo el pulsador pasar´a a la secuencia 3 y as´ı, sucesivamente, hasta la secuencia 4 para volver a comenzar con la secuencia 1. Descripci´on de las secuencias: Secuencia 1: ´ıdem App 1.1 con una alternancia de 150 ms. Secuencia 2: hace parpadear los tres leds simult´aneamente con una alternancia de 300 ms. Secuencia 3: hace parpadear el LED1 con una alternancia de 100 ms, el LED2 con una alternancia de 300 ms y el led3 con una alternancia de 600 ms. Secuencia 4: Hace parpadear simult´aneamente LED1 y LED3, mientras que LED2 lo har´a de manera inversa, con una alternancia de 150 ms. Mientras LED1 y LED3 est´an encendidos, LED2 estar´a apagado y luego a la inversa.

- **Link:** App_3_3
- **Alumno:** Plaate Ivan
- **Observaciones:** La implementación de cuatro secuencias distintas de LEDs con delay_t no bloqueantes permitió un control flexible y modular del comportamiento, sin interrumpir la ejecución del programa. El principal desafío fue organizar múltiples temporizadores en paralelo (especialmente en la secuencia 3, donde cada LED tiene un periodo diferente) y mantener el código legible dentro del switch. Esto se resolvió inicializando y manejando cada delay_t de manera independiente y utilizando arreglos para los puertos y pines, lo que evitó redundancia. Una recomendación es encapsular cada secuencia en funciones separadas, de modo que el main.c quede más limpio y sea más sencillo agregar o modificar secuencias en el futuro.

App 1.4: Desarrolle una aplicaci´on que haga parpadear simult´aneamente los 3 leds onboard de la placa y que use el pulsador onboard para cambiar la frecuencia de parpadeo de manera secuencial entre 4 frecuencias predefinidas por los siguientes tiempos de alternancia: Tiempo 1: 100 ms. Tiempo 2: 250 ms. Tiempo 3: 500 ms. Tiempo 4: 1000 ms. La aplicaci´on inicia con la alternancia determinada por el tiempo 1, al presionar el pulsador pasa al tiempo 2 y as´ısucesivamente hasta llegar al tiempo 4, para reiniciar con el tiempo 1

- **Link:** App_3_4
- **Alumno:** Torres Juan
- **Observaciones:** La implementación permitió controlar el parpadeo de los tres LEDs con diferentes frecuencias de manera sencilla, utilizando un arreglo de tiempos (tiempos[]) y un índice que se actualiza mediante la pulsación de un botón. El principal desafío fue manejar el anti-rebote y la lectura del pulsador, ya que se optó por una solución bloqueante con HAL_Delay y un bucle de espera que detiene la ejecución del programa hasta que el botón se suelte. Esto resolvió el problema de múltiples detecciones, pero limita la capacidad de expandir el sistema con otras tareas concurrentes. Una recomendación es migrar hacia un manejo no bloqueante con temporizadores y máquinas de estado, lo que haría el código más escalable y apto para aplicaciones de mayor complejidad.

Link del repositorio grupal:

https://github.com/Alessandro-Angelillo/Grupo_5_TDII_2025