Práctica 2

Objetivo:

Implementar un módulo de software para trabajar con retardos no bloqueantes.

Punto 1

Implementar las funciones auxiliares necesarias para usar retardos no bloqueantes en un archivo fuente **main.c** con su correspondiente archivo de cabecera **main.h**.

En main.h se deben ubicar los prototipos de las siguientes funciones y las declaraciones

typedef uint32_t tick_t; // Qué biblioteca se debe incluir para que esto compile?

typedef bool bool_t; // Qué biblioteca se debe incluir para que esto compile?

```
typedef struct{
   tick_t startTime;
   tick_t duration;
   bool_t running;
} delay_t;

void delayInit( delay_t * delay, tick_t duration );
bool_t delayRead( delay_t * delay );
void delayWrite( delay_t * delay, tick_t duration );
```

En main.c se deben ubicar la implementación de todas las funciones:

Consideraciones para la implementación:

1. delayInit debe cargar el valor de duración del retardo en la estructura, en el campo correspondiente. No debe iniciar el conteo del retardo. Debe inicializar el flag *running* en `false´.
2. delayRead debe verificar el estado del flag *running*.
    - false, tomar marca de tiempo y cambiar *running* a ‘true’
    - true, hacer la cuenta para saber si el tiempo del retardo se cumplió o no:

      ‘marca de tiempo actual - marca de tiempo inicial es mayor o igual a duración del retardo’?
    - devolver un valor booleano que indique si el tiempo se cumplió o no.
    - Cuando el tiempo se cumple se debe cambiar el flag running a false.
3. delayWrite debe permitir cambiar el tiempo de duración de un delay existente

NOTA: para obtener una marca de tiempo se puede usar la función HAL_GetTick() que devuelve un valor que se incrementa cada 1 ms y que se puede usar como base de tiempo.

stm32f4xx_hal.c

```
303 /**
304   * @brief This function is called to increment  a global variable "uwTick"
305   *        used as application time base.
306   * @note In the default implementation, this variable is incremented each 1ms
307   *       in SysTick ISR.
308  * @note This function is declared as __weak to be overwritten in case of other
309   *      implementations in user file.
310   * @retval None
311   */
312 __weak void HAL_IncTick(void)
313 {
314   uwTick += uwTickFreq;
315 }
316
317 /**
318   * @brief Provides a tick value in millisecond.
319   * @note This function is declared as __weak to be overwritten in case of other
320   *       implementations in user file.
321   * @retval tick value
322   */
323 __weak uint32_t HAL_GetTick(void)
324 {
325   return uwTick;
326 }
```

Punto 2

Sobre el código desarrollado para el punto 1 y sobre el mismo proyecto, implementar un programa que utilice retardos no bloqueantes y haga parpadear el leds de la placa de desarrollo: 100 ms encendido, 100 ms apagado, en forma periódica.

Punto 3 [opcional]

Sobre el código desarrollado para el punto 2 y sobre el mismo proyecto, implementar un programa que haga parpadear el led de la placa de desarrollo en forma periódica con el siguiente patrón:

1. 5 veces con período 1 segundo y ciclo de trabajo 50%.
2. 5 veces con período 200 ms y ciclo de trabajo 50%.
3. 5 veces con período 100 ms y ciclo de trabajo 50%.

Utilizar un vector o arreglo para definir el patrón y cambiar los tiempos de parpadeo.

Para pensar luego de resolver el ejercicio:

- ¿Se pueden cambiar los tiempos de encendido de cada led fácilmente en un solo lugar del código o éstos están *hardcodeados*?

- ¿Qué bibliotecas estándar se debieron agregar para que el código compile? Si las funcionalidades crecieran, habría que pensar cuál sería el mejor lugar para incluir esas bibliotecas y algunos typedefs que se usan en el ejercicio.

- ¿Es adecuado el control de los parámetros pasados por el usuario que se hace en las funciones implementadas? ¿Se controla que sean valores válidos? ¿Se controla que estén dentro de los rangos correctos?

- ¿Cuán reutilizable es el código implementado?

- ¿Cuán sencillo resulta en su implementación cambiar el patrón de tiempos de parpadeo?