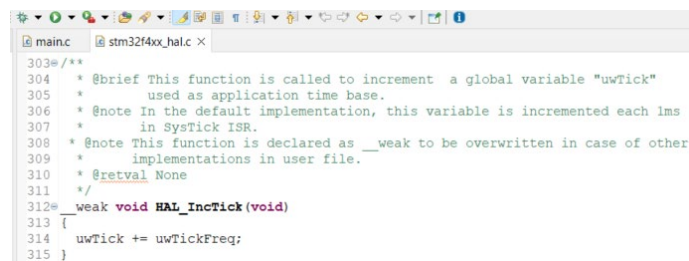


Funciones no bloqueantes

El SysTick:

El **SysTick** es un timer de 32 bits (implementado en el hardware de la arquitectura ARM) que se usa para temporizar eventos en nuestra app. Este Timer, en su configuración por defecto, está configurado para generar una interrupción una vez cada 1 milisegundo. En cada interrupción se incrementa una variable que almacena la cuenta en milisegundos. Podemos ver la función de la HAL: `HAL_IncTick()` que incrementa la cuenta del **SysTick** en la Figura 1.

Estas funciones se encuentran implementadas en el archivo de drivers de la HAL: `stm32f4xx_hal.c`. Cuando el **SysTick** genera la interrupción cada 1 milisegundo, llama a esta función. Esta función incrementa la variable `uwTick` que almacena en contenido de `uwTickFreq`. La variable `uwTickFreq` contiene la cuenta de cuánto tiempo pasó, en milisegundos, desde que se energizó el sistema. Determinada la frecuencia del **SysTick** que por defecto es de **1 Khz**, establece el tiempo del valor de cada cuenta para el **SysTick** que es de 1 milisegundo para la configuración por defecto.

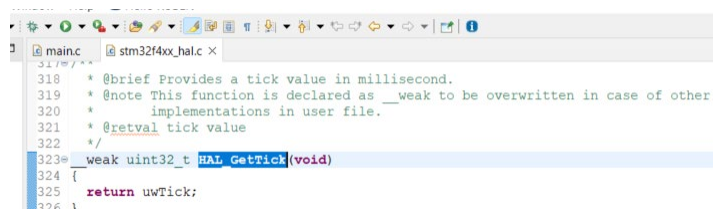


```
303= /**
304  * @brief This function is called to increment a global variable "uwTick"
305  *        used as application time base.
306  * @note In the default implementation, this variable is incremented each 1ms
307  *        in SysTick ISR.
308  * @note This function is declared as __weak to be 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 }
```

Figura 1

En resumen: el **SysTick** genera una interrupción cada 1 milisegundo que llama a la función la función `HAL_IncTick()` y esta incrementa la variable `uwTick`, en 1 milisegundo, cada vez que es llamada.

Para utilizar los valores del **SysTick** usamos la función `HAL_GetTick()`. Esta función retorna el valor de `uwTick`. Recuperando el valor de cuenta del **SysTick** en ese momento. Ver Figura 2.



```
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 }
```

Figura 2

Para desarrollar el tema de funciones no bloqueantes debemos analizar como esta implementada la función *HAL_Delay()* que es del tipo **No Bloqueante**. (Figura 3)

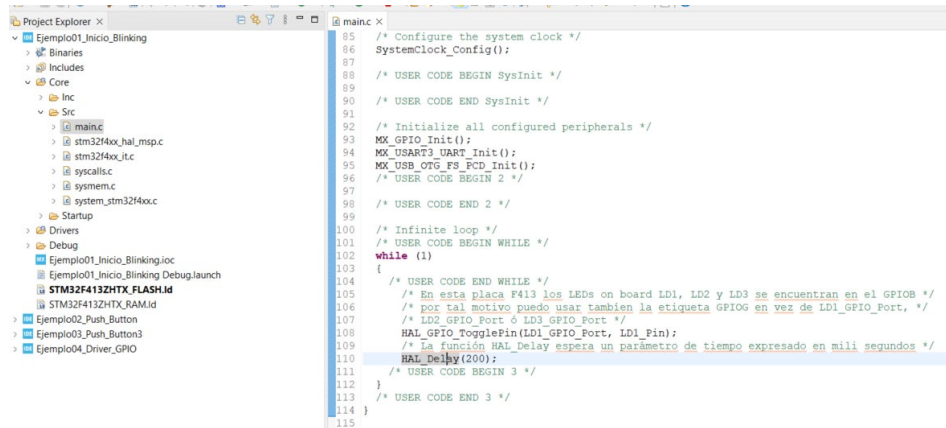


Figura 3

Hacemos click derecho sobre la función *HAL_Delay()* y seleccionamos *Open Declaration* para ver como esta implementada esta función. (Figura 4)

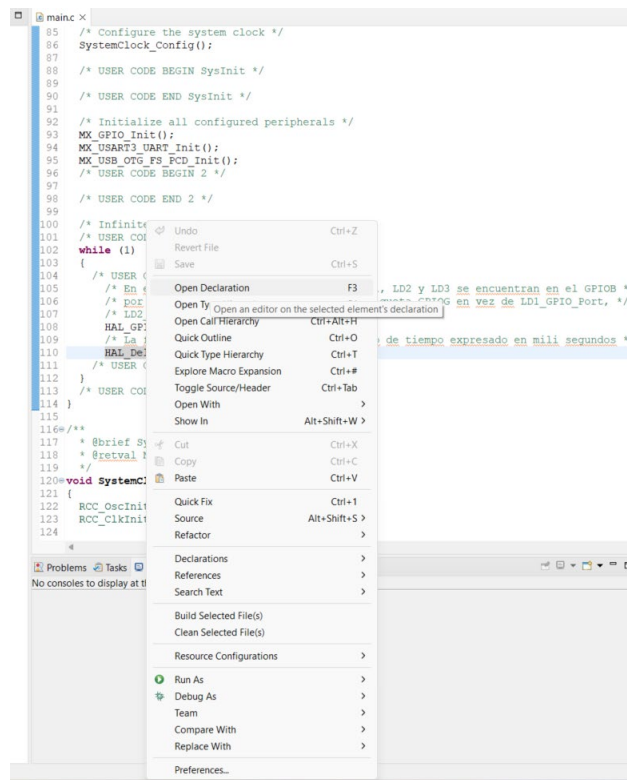
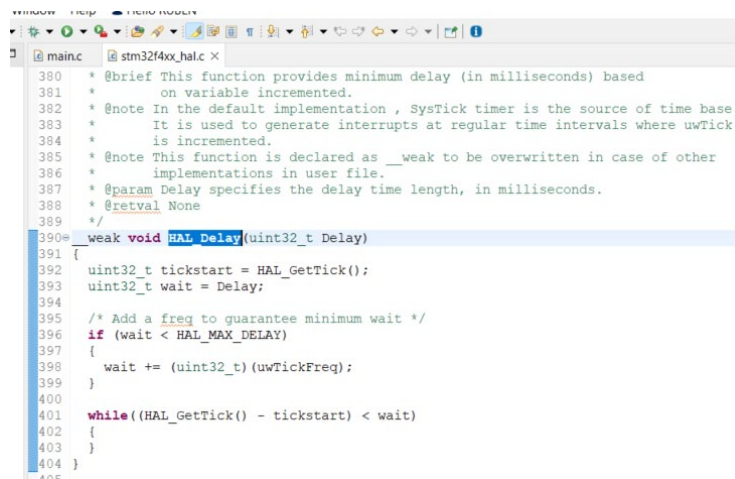


Figura 4

Vemos su implementación en la Figura 5:



```
380 * @brief This function provides minimum delay (in milliseconds) based
381 *       on variable incremented.
382 * @note In the default implementation , SysTick timer is the source of time base.
383 *       It is used to generate interrupts at regular time intervals where uwTick
384 *       is incremented.
385 * @note This function is declared as __weak to be overwritten in case of other
386 *       implementations in user file.
387 * @param Delay specifies the delay time length, in milliseconds.
388 * @retval None
389 */
390 __weak void HAL_Delay(uint32_t Delay)
391 {
392     uint32_t tickstart = HAL_GetTick();
393     uint32_t wait = Delay;
394
395     /* Add a freq to guarantee minimum wait */
396     if (wait < HAL_MAX_DELAY)
397     {
398         wait += (uint32_t)(uwTickFreq);
399     }
400
401     while((HAL_GetTick() - tickstart) < wait)
402     {
403     }
404 }
```

Figura 5

Analicemos como está implementada *HAL_Delay()* para ello veamos su desarrollo en la Figura 5.

Cuando se llama a *HAL_Delay()*, esta captura el valor actual del **SysTick** y lo almacena en la variable **tickstart**. Así tiene un valor de partida para el cálculo del tiempo transcurrido.

Luego guarda el tiempo, en milisegundos que desea contabilizar, en la variable **wait**, este valor lo recibe como parámetro la función *HAL_Delay()* y es el valor en milisegundos que se desea esperar.

Luego compara **wait** con **HAL_MAX_DELAY** (máximo valor que puede contar el **SysTick**) para evitar desbordes en la cuenta. Evalúa si **wait < HAL_MAX_DELAY**.

Si la condición anterior es verdadera, se suma **uwTickFreq** al valor de **wait**. El valor de **uwTickFreq** representa la frecuencia de los ticks del sistema. Al sumar esta cantidad, se asegura que el tiempo de espera mínimo sea al menos un "tick".

Luego entra en un lazo **while** del que no saldrá hasta que se cumpla la condición:

$$(\text{HAL_GetTick}() - \text{tickstart}) < \text{wait}$$

$$(\text{Valor actual del SysTick}) - (\text{valor de inicio de cuenta}) < (\text{tiempo de espera definido})$$

Para convertir esta función **bloqueante** en otra **no bloqueante** deberíamos modificar esta implementación sustituyendo el **while** por una condición **if** que se chequee cada vez que la app pase por esta nueva función.