**Laboratorio 6 – Tareas Periódicas en sistemas operativos**

Existen dos tipos de retardos los cuales son retardos de software y retardos de hardware. Los retardos de software son realizados como comúnmente conocemos, utilizando estructuras cíclicas como ‘for’ ‘while’ ‘do’ ‘goto’ y demás para realizar iteraciones sucecivas hasta detenerse por un conteo (expiración del temporizador de software). Los temporizadores de harware son módulos implementados en la arquitectura del silicón, es decir, el integrado tiene un módulo contador. La precisión y repititibilidad del contador dependen de que utiliza como origen de tiempo, si se utiliza un oscilador de 32kHz (figura inferior izquierda) será más impreciso que si se utiliza el cristal de la derecha o que el oscilador interno del microcontrolador.



**Una breve explicación de las funciones de temporización**

El laboratorio de hoy constara en generar tareas periódicas, las cuales son más precisas que la función osDelay() de Keil. La función inferior define una tarea periódica del sistema operativo en la cual necesita un nombre genérico y el callback (puntero a función) que representa la función a ejecutar.



Primeramente debemos declarar el identificador del temporizador



Seguido deberemos crear la función de temporización debajo





El primer argumento es el tipo de temporizador, el segundo el tipo de temporizador si es de un solo tiro (osTimerOnce) o periódico (osTimerPeriodic), el tercer argumento es una variable que se ingresa como argumento a la nueva función.

Finalmente ajustado el temporizador lo iniciamos por su identificador y luego de que tiempo arrancar la función



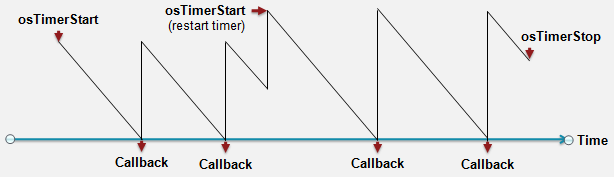
En recuento debemos realizar lo siguiente:

1 – osTimeDef() para definir el temporizador

2 – osTimerCreate() para crear el temporizador

3 – osTimerStart() osTimerStop() para arrancar y/o detener el temporizador

La imagen inferior muestra la función de manera gráfica de un temporizador

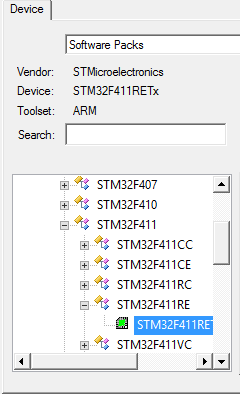


Creando el proyecto

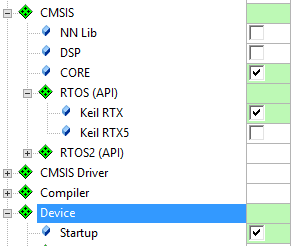
1 – Crear un folder con un nombre específico, por ejemplo, ‘TareasPeriódicas’

2 – En Keil cree el nuevo proyecto dentro de este folder

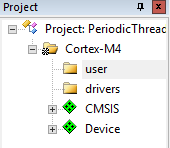
3 – Seleccione el dispositivo que posee en su tarjeta. En mi caso STM32F411RETX



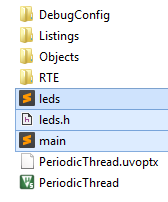
4 – En los board support packages seleccione CORE, RTX, startup.s como archivos iniciales



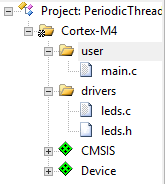
5 – Nombre y cree los siguientes grupos, solo para orden



6 – Copie los siguientes archivos enmarcados a su carpeta de proyecto



7 – Añada los siguientes archivos como sigue:



8 – Compile el programa. Este debe dar 0 errores. El código en main debe verse como sigue:

#include "leds.h"

#include "cmsis\_os.h" // ARM::CMSIS:RTOS:Keil RTX

#include "stm32f4xx.h" // Device header

int32\_t count1, count2,count3;

int main(void)

{

LEDs\_Init();

while(1){}

}

void red\_thread(void const \*argument)

{

while(1)

{

RED\_toggle();

osDelay(100);

}

}

void blue\_thread(void const \*argument)

{

while(1)

{

BLUE\_toggle();

osDelay(100);

}

}

void green\_thread(void const \*argument)

{

while(1)

{

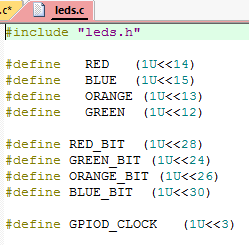
GREEN\_toggle();

osDelay(100);

}

}

9 – Modifique los bits de LEDs para que puedan salir físicamente, por ejemplo PORTA.BIT5, PORTA.BIT6, PORTA.BIT7, PORTA.BIT8.



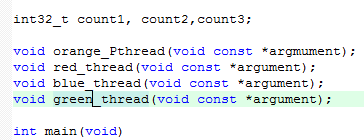
10 – Ahora crearemos una tarea periódica, primero creamos la tarea que se llamará orange\_Pthread y las tareas red\_thread, blue\_thread, Green\_thread para tenero orden. Recuerde que primero siempre declaramos y luego definimos (ya las definiciones están listadas):

void orange\_Pthread(void const \*argument);

void red\_thread(void const \*argument);

void blue\_thread(void const \*argument);

void green\_thread(void const \*argument);



11 – Continuamos con el primer paso de tareas periódicas de un sistema operativo que es definir el temporizador y definir la función

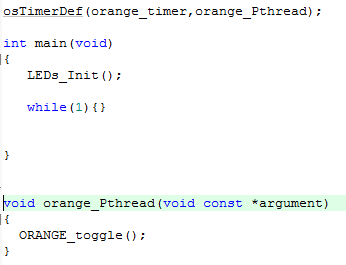
osTimerDef(orange\_timer,orange\_Pthread);

void orange\_Pthread(void const \*argument)

{

ORANGE\_toggle();

}



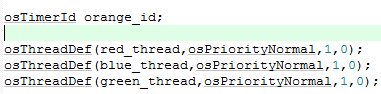
12 – Seguido de osTimerDef necesitamos el id de la tarea, además de definir las tareas

osTimerId orange\_id;

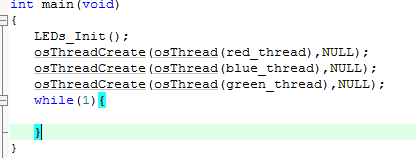
osThreadDef(red\_thread,osPriorityNormal,1,0);

osThreadDef(blue\_thread,osPriorityNormal,1,0);

osThreadDef(green\_thread,osPriorityNormal,1,0);

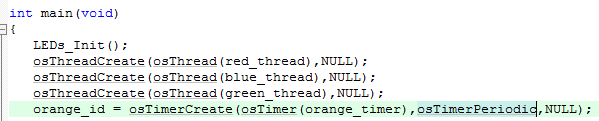


13 – osThreadCreate será utilizada dentro del main para crear las tareas



14 – Adicionalmente creamos la tarea de temporización. La crearemos como un temporizado de un solo tiro como sigue:

orange\_id = osTimerCreate(osTimer(orange\_timer),osTimerOnce,NULL);



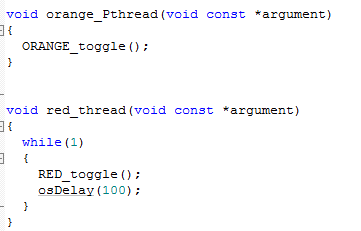
15 – Finalmente inicializamos el temporizador con el id adquirido y ajustamos su temporizado.

osTimerStart(orange\_id,1000);



16 – Ajuste el ST-Link y el cristal, esto es importante pues si no, no generará la acción de tiempo, en este caso 1000 ms.

17 – Algo que no se ha explicado y que debe notar, es que las tareas periódicas o temporizadas no usan while(1) para ejecutarse, esa es una de las diferencias de las tareas comunes y las temporizadas



18 – Finalmente compile y descarge, observe que posee 3 leds ejecutándose con retardo de software vs uno ejeutandose con retardo de hardware

19 – Si desea generar un temporizador que se ejecuta solamente una vez puede probar lo siguiente: osTimerCreate(osTimer(orange\_timer),osTimerOnce,NULL);

**Siguiente 50% del Laboratorio**

Hellen

Generar el mismo código temporizado para 3 LEDs

LED1 = Temporizador de un solo tiro de 2000 ms

LED2 = Temporizador periódico de 3000ms

LED3 = Temporizador periódico de 5000ms

Diego

Generar el mismo código temporizado para 3 LEDs

LED1 = Temporizador de un solo tiro de 500 ms

LED2 = Temporizador periódico de 1500ms

LED3 = Temporizador periódico de 10000ms

Laura

Generar el mismo código temporizado para 3 LEDs

LED1 = Temporizador de un solo tiro de 800 ms

LED2 = Temporizador periódico de 4000 ms

LED3 = Temporizador periódico de 8000 ms