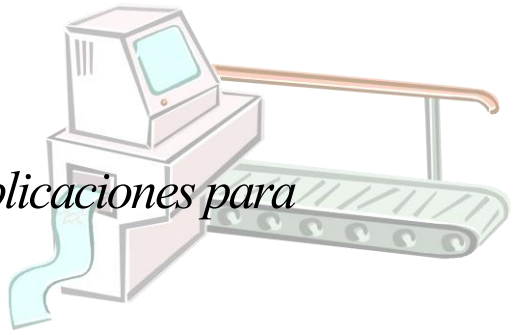
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Práctica 5

Sistemas Operativos de Tiempo Real

Laboratorio de Desarrollo de Aplicaciones para Sistemas Industriales



Sistemas Operativos de Tiempo Real

Introducción

En la presente práctica vamos a utilizar el RTOS llamado Keil RTX. Este es un RTOS libre y determinístico diseñado para dispositivos ARM y Cortex-M. En concreto, nosotros vamos a utilizar una versión de RTX que es compatible con el estándar CMSIS, con lo que podremos utilizar sus funciones, asegurando portabilidad entre distintos procesadores Cortex-M.

Creacion de un proyecto con el Sistema Operativo RTX en Keil uVision

Antes de crear un proyecto nuevo, se debe obtener el Sistema Operativo. Keil ofrece el sistema operativo a través del enlace https://www.keil.com/demo/eval/rtx.htm.

Una vez descargado e instalado en una carpeta (nosotros, en el desarrollo de la práctica vamos a referirnos a que la localización del sistema operativo se encuentra en la carpeta C:\Keil\cmsis_rtos_rtx_v4p70\), se crea un proyecto nuevo.

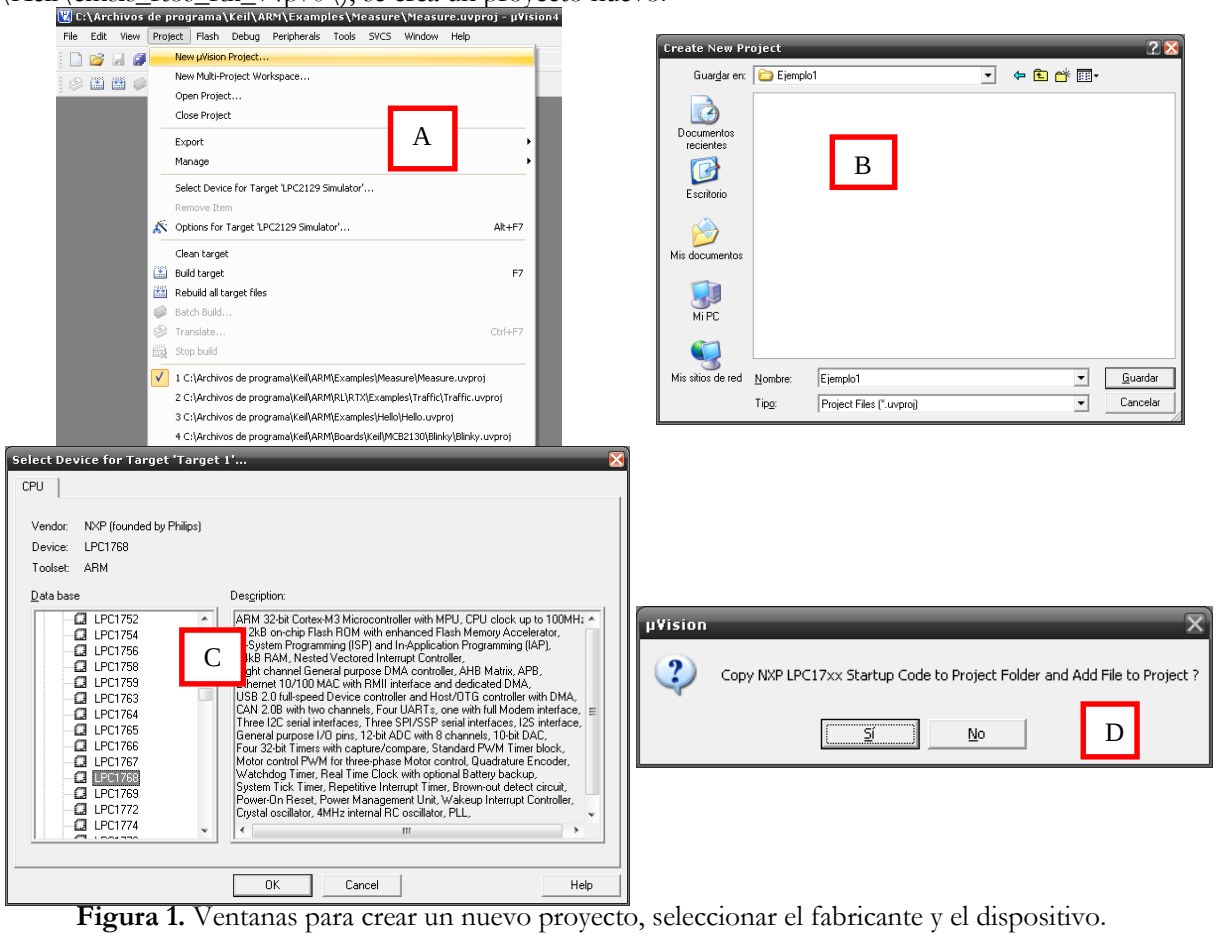


Figura 1. Ventanas para crear un nuevo proyecto, seleccionar el fabricante y el dispositivo.

Renombramos la entrada Target 1 como RTX-LPC1768 (haciendo click sobre ella, tras estar previamente seleccionada). Cambiamos también la entrada del árbol de proyecto *Source Group 1* como *StartUp*. En esta entrada de proyecto, además de contener el fichero *startup_LPC17xx.s*, incluiremos otro fichero llamado *system_LPC17xx.c* necesario para la configuración inicial del microcontrolador. Este fichero se encuentra en la ruta C:\Keil\ARM\Startup\NXP\LPC17xx\, pero es muy importante copiar este fichero desde este directorio hacia el mismo directorio en que se encuentre nuestro proyecto e incluirlo desde ahí, para así no correr el riesgo de modificar el original. Una vez copiado, lo incluiremos situándonos sobre la carpeta *StartUp*, pulsando el botón derecho y seleccionando la opción *Add Existing Files To Group*.

A continuación vamos a crear los grupos de objetos relacionados con el Sistema Operativo. Para eso, nos situamos sobre la entrada RTX-LPC1768 del árbol del proyecto, hacemos click con el botón derecho y seleccionamos *Add Group*…. Se creará una entrada llamada *New Group* que renombraremos como *RTX Configuration*. A continuación realizamos una copia del fichero RTX_Conf_CM.c, que se encuentra en la carpeta C:\Keil\cmsis_rtos_rtx_v4p70\Templates\ al directorio donde se encuentra nuestro proyecto, y lo incluimos dentro del grupo *RTX Configuration*. Este fichero contiene los parámetros configurables del RTOS. Se recomienda visualizar el contenido de este fichero con el objetivo de conocer las posibilidades de configuración que nos ofrece, o bien consultar su edición en el manual del Sistema Operativo.

Otro elemento relacionado con el Sistema Operativo es la librería RTX_CM3.lib, que se encuentra en el directorio C:\Keil\cmsis_rtos_rtx_v4p70\lib\ARM. Este fichero contiene el código compilado del Sistema Operativo, es decir, el código que se ejecutara cuando solicitemos algún servicio del RTOS. Para añadir esta librería crearemos un nuevo grupo llamado *RTX Library*.

Finalmente, se creará una entrada llamada *New Group* que renombraremos como *SourceFiles*. En este grupo incluiremos nuestros ficheros fuente.

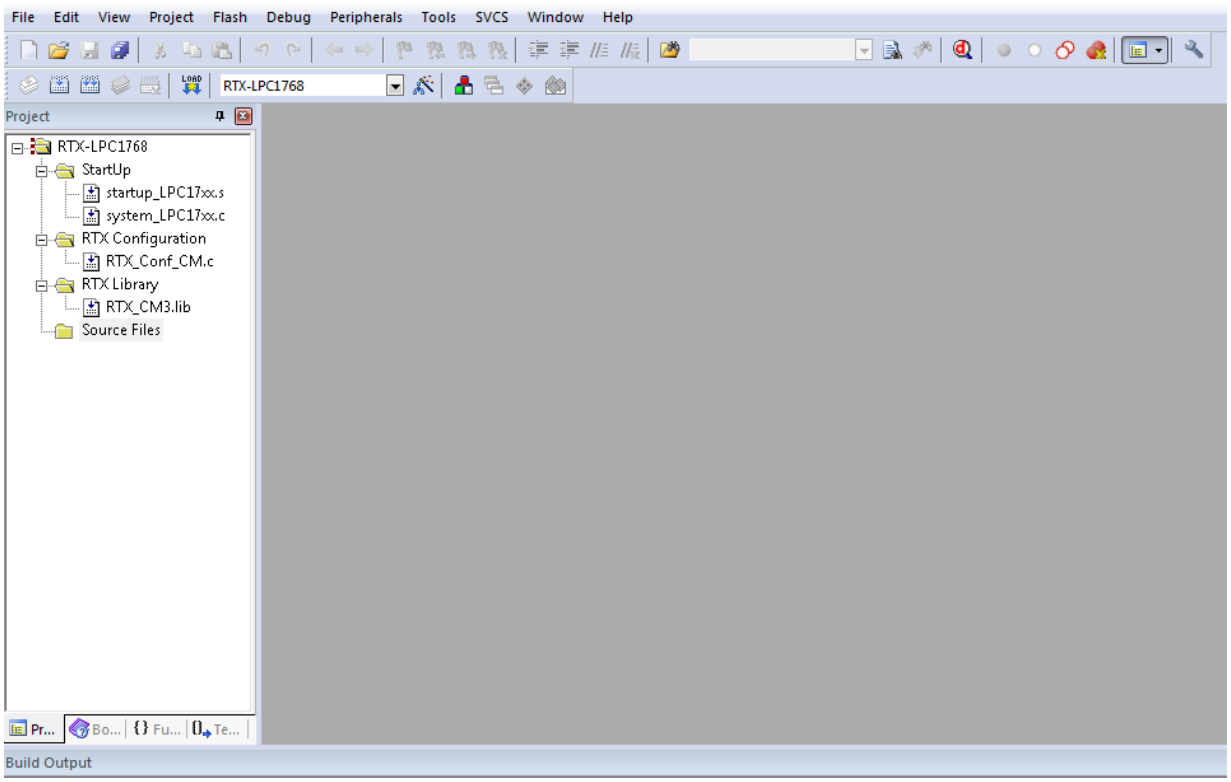


Figura 2. Configuración de los grupos del proyecto

Una vez creados los grupos, deberemos configurar el proyecto para indicarle que vamos a utilizar como Sistema Operativo el sistema RTX. Para eso, nos situamos sobre el nombre del proyecto en el árbol del proyecto y haciendo click con el botón derecho, pulsamos las Opciones del proyecto

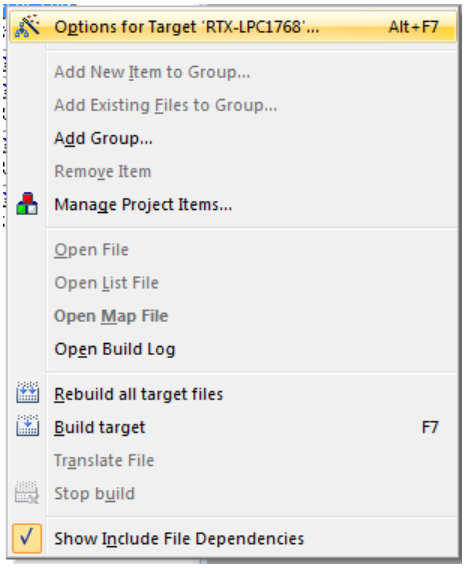


Figura 3. Menú de Opciones

En la pestaña Target, en la lista desplegable de Operating System, seleccionar la Opción RTX Kernel. Asimismo, se debe seleccionar la opción Use Microlib.

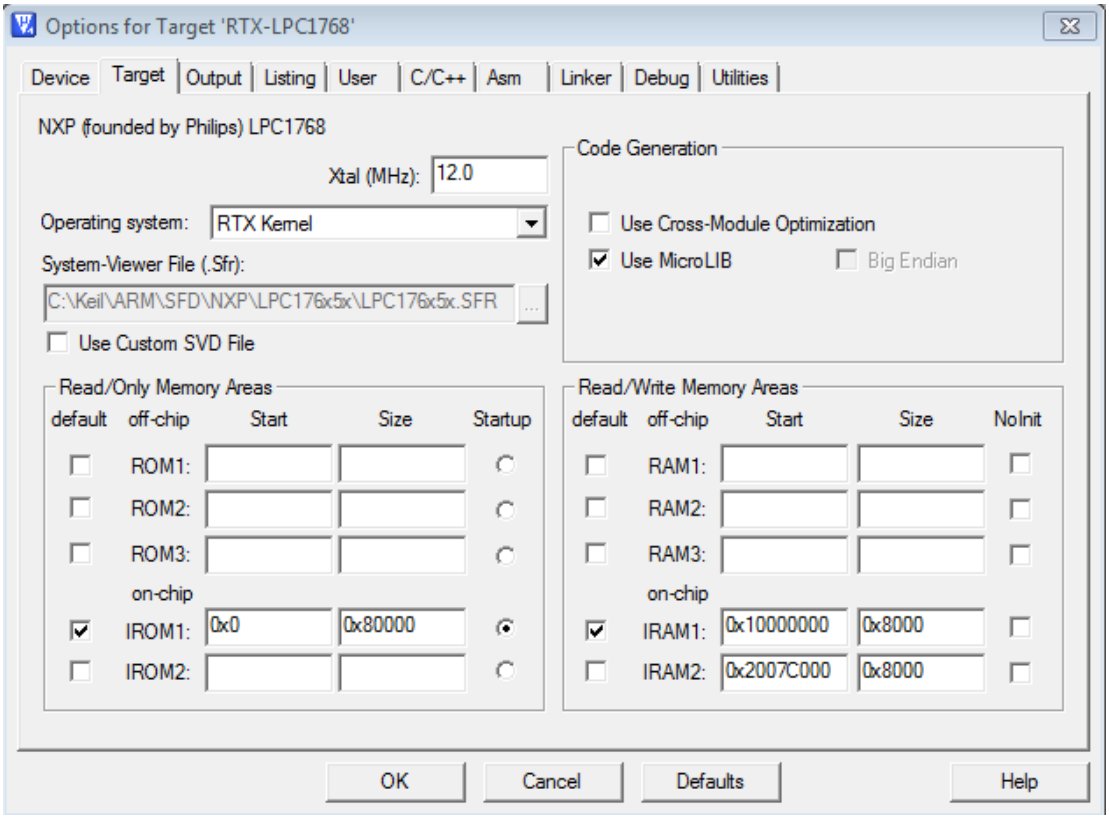


Figura 4. Configuracion de la pestaña Target

A continuación, se introduce en la pestaña C/C++, en el cuadro de dialogo Include PATHs, la ruta donde se encuentran los ficheros de inclusión d RTX (En nuestro caso, C:\Keil\cmsis_rtos_rtx_v4p70\INC)

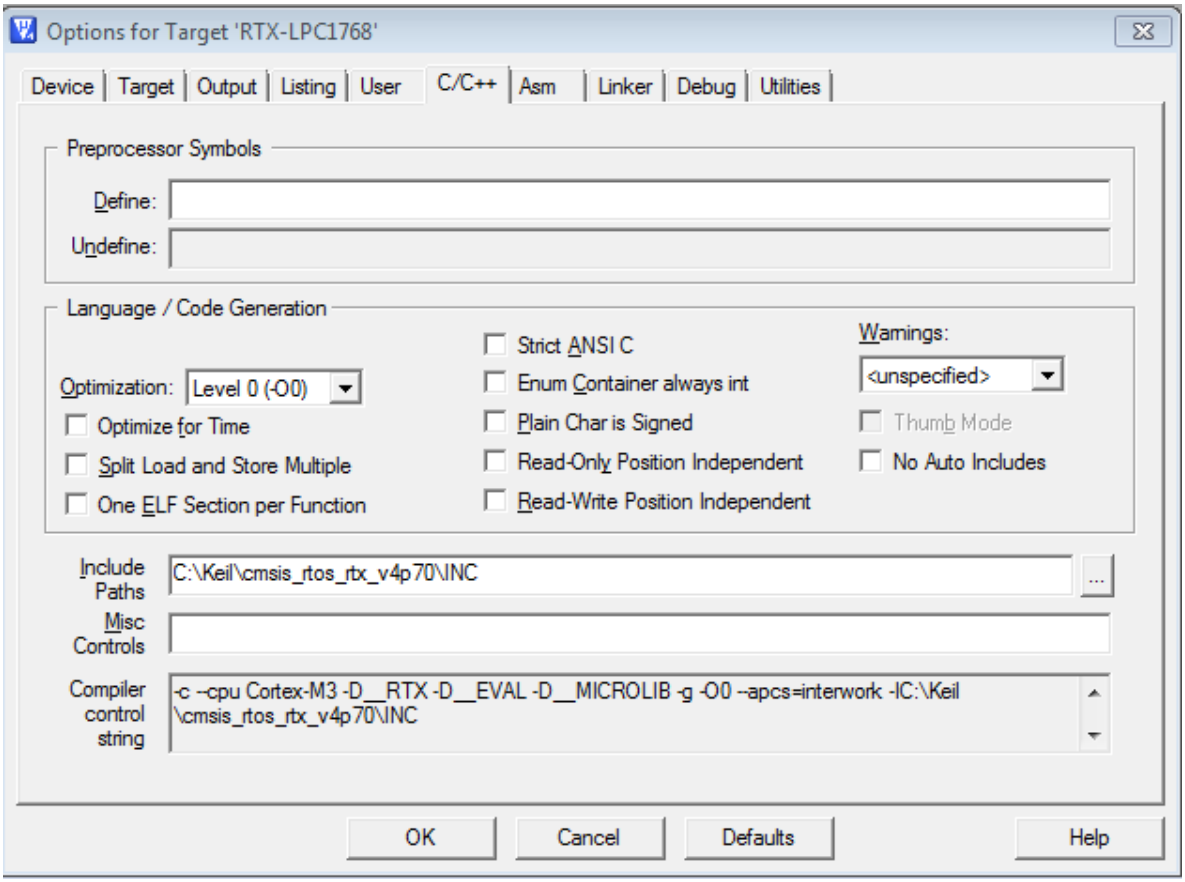


Figura 5. Configuracion de la pestaña C/C++

Con la selección de estas opciones, ya el proyecto se encuentra preparado para trabajar con el Sistema Operativo RTX. Según el resto de opciones que se quiera dotar al proyecto, se realiza la configuración pertinente.

Practica Propuesta

Se propone la realización de la práctica con la misma funcionalidad de la practica anterior, pero haciendo que la gestión de cada diodo la realice un hilo distinto. De esta forma, se crearán dos hilos, utilizando las funciones que nos ofrece nuestro sistema Operativo, y uno de ellos se encargará del parpadeo del diodo LD2 y el otro hilo del parpadeo del diodo LD3. Asímismo, la velocidad de parpadeo se puede variar usando los pulsadores KEY1 y KEY2.

Realización de la Práctica

Realícese un proyecto que realice las mismas funciones que se obtenían en la práctica 4, pero usando un hilo para la gestión de cada diodo, mediante el uso del Sistema Operativo RTX.

Una vez conocidos los requisitos que se plantean en la práctica, vamos a construir paso a paso funciones que nos permitan llegar hasta la aplicación que resuelva el problema. Es importante destacar, que las aplicaciones

desarrolladas se ejecutaran en modo usuario, es decir, en modo no privilegiado, por lo que habrá que configurar el parámetro OS_RUNPRIV del fichero RTX_Conf_CM.c a 0.

Paso 1. Configuración de los hilos del programa

Realícese un proyecto, siguiendo los pasos descritos, que cree una estructura de dos hilos (a los que llamaremos *gestionLD2 y gestionLD3*).

```
#include "lpc17XX.h"

#include <os_cmsis.h>

osThreadId hilo1_id;
osThreadId hilo2_id;

void gestionLD2 (void const *argument);
void gestionLD3 (void const *argument);

osThreadDef(gestionLD2, osPriorityNormal, 1, 0);    // 1: el número de hilos de ese tipo. 0: Tamaño de la Pila→ tamaño estandar
osThreadDef(gestionLD3, osPriorityNormal, 1, 0);

//Configuro el hilo que gestiona el led LD2
void gestionLD2(void const *argument){

while(1){
      …
     }
}


//Configuro el hilo que gestiona el led LD3
void gestionLD3(void const *argument){

while(1){
        …
        }
}
int main(){
…
// Crear productor y consumidor
hilo1_id = osThreadCreate(osThread(gestionLD2), NULL);
hilo2_id = osThreadCreate(osThread(gestionLD3), NULL);
osSignalWait(0x01,osWaitForever);
}
```

Complete el proyecto anterior y situando *breakpoints* en el código de las funciones *gestionLD2* y *gestionLD3*, como el sistema Operativo reparte el uso de la CPU entre ambos hilos, consiguiendo que se ejecute alternativamente código de ambas funciones

Paso 2. Inclusión de la funcionalidad de las IRQ´s.

Añádase al proyecto que se obtiene del paso anterior, la funcionalidad desarrollada en la práctica 4 respecto a la gestión de interrupciones (cambio de velocidad de parpadeo de los diodos LD2 y LD3). Para tal fin, téngase en cuenta que en todo momento el código desarrollado se debe ejecutar en modo no privilegiado, por lo que será necesario la configuración de llamadas al sistema.

```
#include "lpc17XX.h"

#include <os_cmsis.h>

osThreadId hilo1_id;
osThreadId hilo2_id;
```

```
void gestionLD2 (void const *argument);
void gestionLD3 (void const *argument);

osThreadDef(gestionLD2, osPriorityNormal, 1, 0);    // 1: el número de hilos de ese tipo. 0: Tamaño de la Pila→ tamaño estandar
osThreadDef(gestionLD3, osPriorityNormal, 1, 0);

void EINT1_IRQHandler(){
…
}

void EINT2_IRQHandler(){
…
}


//Configuro el hilo que gestiona el led LD2
void gestionLD2(void const *argument){

while(1){
      …
     }
}


//Configuro el hilo que gestiona el led LD3
void gestionLD3(void const *argument){

while(1){
        …
        }
}
int main(){
…
configIRQ();
// Crear productor y consumidor
hilo1_id = osThreadCreate(osThread(gestionLD2), NULL);
hilo2_id = osThreadCreate(osThread(gestionLD3), NULL);
osSignalWait(0x01,osWaitForever);
}
```