LSE001-2014: Sistemas Operativos en Tiempo Real

Dr. Casimiro Gómez González[1],Oscar Navarrete Aguilar[2]
Facultad de Electrónica, UPAEP
Correos: [1]casimiro.gomez@upaep.mx,[2]oscar.navarrete@upaep.edu.mx
Tel: 222 229 9428

Otoño 2014

Prólogo

En el presente reporte se realiza un estudio de los sistemas operativos en tiempo real con el objetivo de aplicarlos en las tarjetas embebidas con microcontroladores ARM. Se ha elegido la plataforma *CMSIS-RTOS RTX* de **Keil** y el microcontrolador *LPC4357*. Se describe la aplicación de los sistemas operativos en tiempo real para un satélite **CubeSat**. Este documento se inicio en el otoño 2014 y estará en actualización continua.

El autor
Casimiro Gómez González
Doctor en Ingeniería Mecatrónica

Índice general

Capítulo 1

Introducción

Los microprocesadores arribaron en 1970. Los sistemas operativos encontraron aplicación rápida en las sistemas basados en microprocesador. Para mediados de 1980 pocas de estas implementaciones usadas se pueden describir como sistemas operativos en tiempo real diseñados formalmente.Dos factores afectan la aceptación de los sistemas operativos en tiempo real, uno debido a los limites de la máquina, y los otros a la cultura de diseño alrededor de los microcontroladores. Los primeros microprocesador estaban bastante limitados en sus habilidades computacionales, velocidad de operación y capacidades de memoria. Tratar de establecer la estructura de un sistema operativo bajo estas bases es muy difícil. Sin embargo, la mayoría de estos sistemas operativos embebidos programados tienen poco o ningún relación con sistemas operativos [**?**].

1.1. Produciendo software de calidad

Si se desea diseñar software de alta calidad, este debe tener las siguientes características:

- Debe realizar su trabajo correctamente (**funcionalmente correcto**)
- Debe realizar su trabajo en el tiempo correcto (**temporalmente correcto**)
- Su comportamiento debe ser predecible
- Su comportamiento debe ser consistente

- El código no debe ser difícil de mantener (baja complejidad)
- Lo correcto del código puede ser analizado (análisis estático)
- El comportamiento del código puede ser analizado (análisis de cobertura)
- El comportamiento en tiempo de ejecución debe ser predecible
- Los requerimientos de memoria debe ser predecibles
- El código puede, si es necesario, ser validado de acuerdo con estándares relevantes.

1.2. Modelando Software

Existen diferentes etapas para producir un programa ejecutable, como se indica en la figura 2.1.

Primero el diseño del software, resulta en el diseño o modelo lógico. Esto es implementado como código fuente, el modelo físico. El código fuente es compilado, enlazado y construido a un código objeto, el modelo bajado (deployment model). Finalmente el código objeto es cargado en el procesador y despues ejecutado, el modelo de ejecución. El modelo del código nos proporciona un visión estática del software. En contraste con el modelo de ejecución, una combinación de código, datos y procesador, representa software en ejecución. Esto se define como un proceso de software, también conocida como una tarea en el mundo de los embebidos. En palabras simples, una tarea representa la ejecución de un programa secuencial simple. Por lo regular al modelo de ejecución también se le llama modelo de tareas.

1.3. La importancia del tiempo y el temporizado

En sistemas electrónicos analógicos todas las operaciones pueden, si se requiere, realizarse simultáneamente (concurrentemente), no solamente eso, sino que el procesamiento es hecho instantáneamente. Pero esto no es posible en sistemas basados en procesador ya que este es fundamentalmente discreto en sus operaciones:

- Un procesador puede hacer solamente una cosa a tiempo (una maquina secuencial) y
- Las operaciones toman tiempo - las cosas no suceden instantáneamente

Estos son dos factores que causan muchas angustias en nuestro trabajo de diseño. Y esta es la razón por lo cual si esta diseñando sin conocer tus necesidades de tiempo, debes estar preparado para sorpresas desagradables.

Con esto en mente, los primeros diseños tenían las siguientes características:

- Es ejecutado en un lazo continuo
- Termina su trabajo en cada lazo (Semantica de ejecutar hasta completar)
- Toma tiempo para completar su trabajo (Tiempo de ejecución de tareas- Te)
- Es repetido a intervalos regulares o periodos (operaciones periodicas - Tp)
- No hacer nada mientras esta retardado (tiempo desperdiciado Ts)
- Necesita un mecanismo de temporizado para controlar el periodo de tiempo

En este diseño en particular Tp y Td son requisitos del sistema, Te depende de nuestro código de solución y Ts es nuestro tiempo limite, como puede verse en la figura . Por ejemplo, suponiendo que Tp es $100ms$ y Te es $5ms$, esto nos da un valor de Ts de $95ms$. Esto quiere decir que el procesador esta ejecutando código por $5ms$ cada $100ms$, una utilización (U) de $5\,\%$.

Primero, en los sistemas embebidos se pretende que las tareas se ejecuten hasta ser completadas cada vez que sean activadas. Segundo, en los sitemas prácticos debe haber un tiempo de espera Ts. Tercero, por el hecho de que un código se pueda ejecutar dentro de un periodo de tiempo esto no significa que su rendimiento sea aceptable. El retardo de tiempo del procesamiento de entrada-salida puede causar problemas en el comportamiento de todo el sistema.

1.4. Manejando múltiples trabajos

Para manejar trabajos concurrentes independientes, lo cual significa que tenemos diferentes trabajos que en el mundo real pueden:

1. Ser atendidos a intervalos regulares - Funciones periódicas
2. Ser atendidos en tiempos aleatorios - Funciones asíncronas o aperiodicas
3. Ser procesados simultaneamente
4. Muy diferentes necesidades de temporizado

1.5. Usando Interrupciones como máquina de ejecución

Un **timer** se dispara, un interruptor se presiona, un dispositivo periférico necesita atención, son eventos típicos del mundo real en sistemas basados en computadoras. Hay, de hecho, solamente dos maneras de detectar estos eventos: Buscando estos eventos (polling) o enviando una señal directamente a el procesador (interrupciones de hardware).

A diferencia de los métodos anteriores cuando se usaba el polling dentro de una unidad secuencial de programa, ahora se buscan las interrupciones de hardware, una señal electrónica que es aplicada al procesador. Cuando la interrupción es generada se llama a una respuesta predeterminada (la cual es predefinida por el programador) para ejecutar un software específico. Así que la interrupción puede ser vista como una máquina de ejecución de una pieza particular de software.

Así que lo que tenemos es una ejecución aparentemente concurrente de un conjunto individual de procesos (tareas), llamado "cuasi-concurrencia". En otras palabras, usando interrupciones, podemos correr múltiples tareas simultáneamente: una forma de multitareas simple. De esta forma se puede concluir que:

1. Las interrupciones son mecanismos habilitados por llaves que permiten al diseño trabajar
2. Las interrupciones simplifican el diseño funcional del sistema

3. Las interrupciones simplifican el manejo de diferentes temporizados y requisitos de respuestas

4. El comportamiento actual en tiempo de ejecución no puede ser predicho o analizado estáticamente

Capítulo 2

Empezando a crear proyectos con μVision de Keil

En el presente capítulo se explica de forma detallada los pasos necesarios para compilar y cargar un proyecto usando el Software *μVision* y la plataforma *CMSIS-RTOS RTX*, ambos de **Keil**, así como del microcontrolador *LPC4357*

2.1. Instalando el software

El software se encuentra disponible en la página `https://www.keil.com/download/product/` seleccionando el producto **MDK-ARM v5**. Al terminar la descarga y la instalación, se abre la interfaz. A continuación es necesario instalar los paquetes necesarios para el microcontrolador. Para ello se abre el complemento llamado **Pack Installer** y se busca el microcontrolador que se encuentra en **NXP ->LPC4300 Series ->LPC435x ->LPC4357**. Del lado izquierdo aparecerán los paquetes necesarios para instalar. Se da click en los botones correspondientes. En la figura 2.1 se muestran estas opciones.

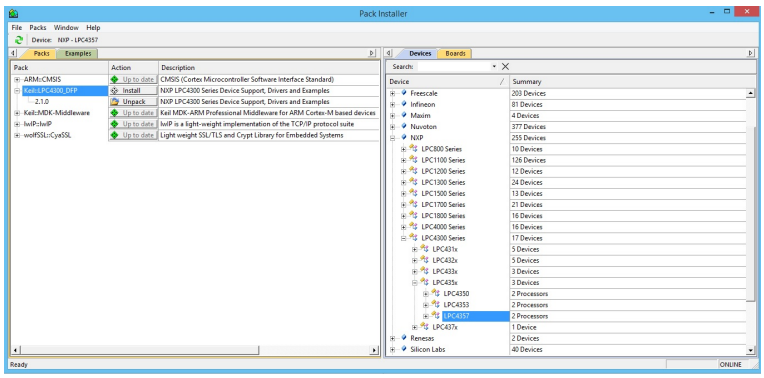

Figura 2.1: Instalación de paquetes del microcontrolador

2.2. Creación del proyecto

Para crear un nuevo proyecto en *µvision*, seguimos los siguientes pasos: **Barra de herramientas ->Project ->New µvision Project...**. En seguida se selecciona la carpeta en donde se quiere crear. Se recomienda tener una carpeta general y dentro de esta, exista una carpeta para cada proyecto debido al número de elementos creados por proyecto. Al seleccionar la carpeta y asingarle un nombre al proyecto, aparecerá una ventana como la de la figura 2.2. En seguida se selecciona el microcontrolador que se encuentra en **NXP ->LPC4300 Series ->LPC435x ->LPC4357 ->LPC4357: Cortex-M4** y se da click en **OK**.

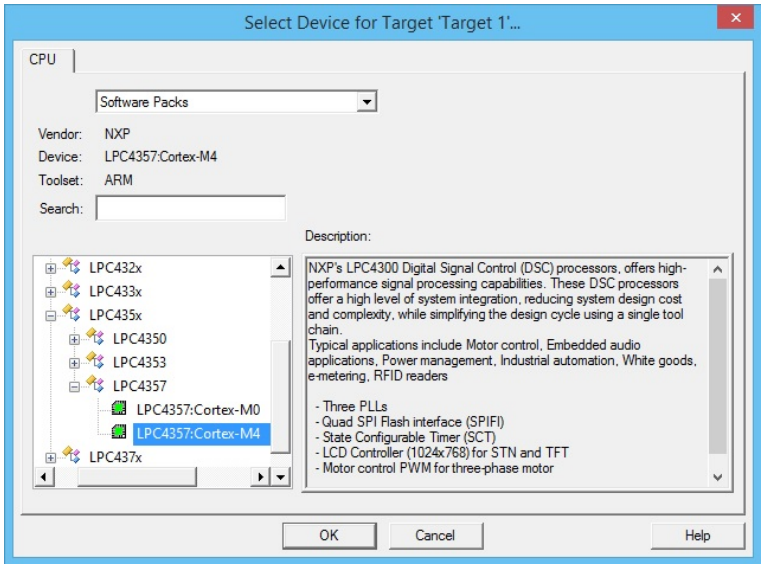

Figura 2.2: Selección del microcontrolador

Para finalizar esta parte, es necesario seleccionar los Componentes del Software esenciales para el proyecto. En este caso, se encenderán los leds de la tarjeta MCB4300 usando la plataforma CMSIS-RTOS, para ello se necesitarán de los siguientes elementos: **Board Support ->LED(API) ->LED** , **CMSIS ->CORE** , **CMSIS ->RTOS(API) ->Keil RTX** , **Device ->GPIO** , **SCU** , **Startup** . En la figura 2.3 se muestran los Componentes seleccionados. Al finalizar de seleccionar todos se da click en **OK**.

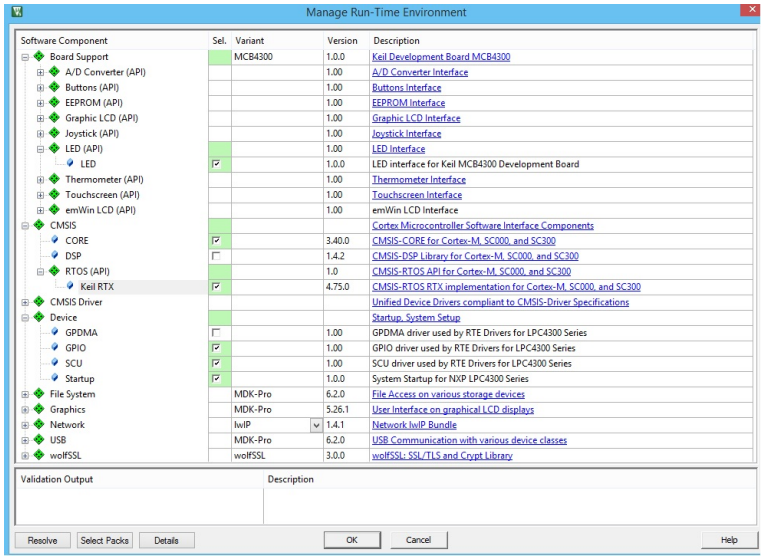

Figura 2.3: Componentes necesarios para proyecto

2.3. Configuraciones básicas

Se observa en la interfaz de µVision que ya se encuentran los componentes necesarios para el proyecto. El siguiente paso es definir un nombre para la carpeta objetivo y para el grupo de fuente. Para ello se da click en la carpeta que por defecto se llama **Target 1** y a continuación en **Manage Project Items...** En la pestaña de **Project Items** se encuentran tres ventanas. En la primera ventana llamada **Project Targets** se da click en el botón **New (Insert)** y se escribe el nuevo nombre del proyecto. Al terminar, se selecciona el anterior nombre y se elimina con el botón **Delete (Delete)**. Se repite el mismo procedimiento para el nombre del grupo. Todas estas configuraciones se muestran en la figura 2.4. Al finalizar se da click en **OK**.

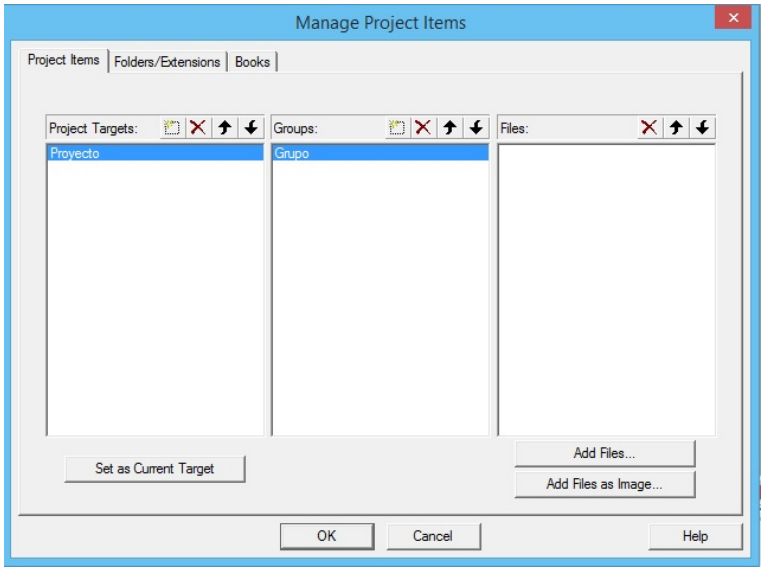

Figura 2.4: Configuración de objetos del proyecto

Se observa que ahora ya tienen otro nombre las carpetas del proyecto. Se prosigue a crear el archivo **main.c** para ello en la barra de herramientas se da click en **File** y en **New...**. En el **Anexo I** de este trabajo se encuentra un código de ejemplo que se puede ocupar para compilar y crear el primer proyecto, esto con el fin de enfocarse a la correcta realización del proyecto . Más adelante se explicarán a detalle las funciones y comandos necesarios de la plataforma CMSIS-RTOS.

Se guarda el código dando click en **File** y posteriormente en **Save As** en la misma carpeta del proyecto. El archivo se guarda con extensión **.c**. Ahora se da click derecho en la carpeta de grupo y se selecciona la opción **Add Existing Files to Group**. Se agrega el archivo que se acaba de guardar. Ahora ya aparecerá agregado en la lista debajo de la carpeta del grupo.

Ahora es necesario realizar un par de modificaciones más a las configuraciones que trae el software por defecto. Para ello se da click en a **Barra de Herramientas** , **Project** y **Options for Target**. Se abrirá una nueva ventana con varias pestañas. En la pestaña **User**, seleccionar en la opción **After Build/Rebuild** la casilla llamada **Run #1** y buscar la ruta en donde se tenga el ejecutable **ElfDwT.exe**. Generalmente se encuentra en **C: ->Keil_v5 ->ARM ->BIN**. Después de insertar la ruta del ejecutable, es necesario escribir la dirección base así: **!L BASEADDRESS(0x1A000000)** Esta

ventana se muestra en la figura 2.5. Al finalizar se da click en **OK**.

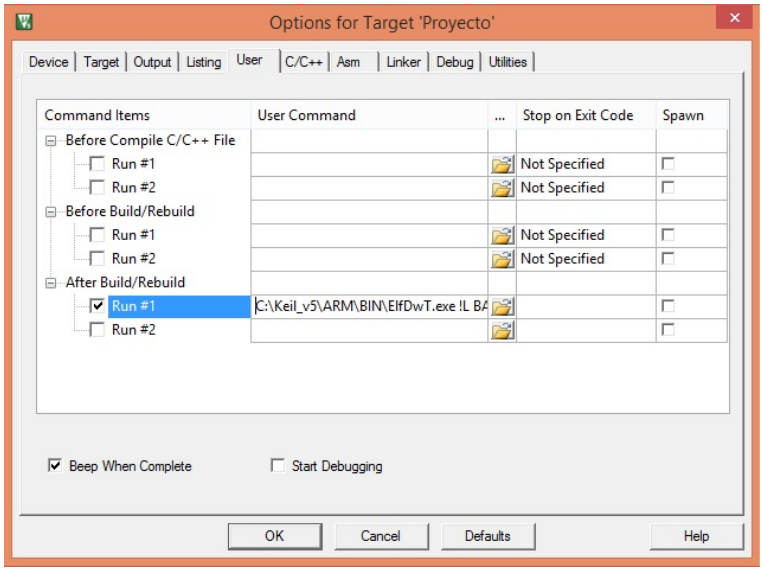

Figura 2.5: Ventana de opciones para el proyecto

Como última configuración, se da click nuevamente en **Barra de Herramientas** , **Project** y **Options for Target**. En la pestaña **Debug**, se da click en el botón **Settings** que se encuentra en la parte superior derecha de la ventana. En la pestaña **Flash Download** de la nueva ventana, en la sección de **Download Function** se activa la casilla llamada **Reset and Run**. Este paso se muestra en la figura 2.6. Al finalizar se da click en **OK** de ambas ventanas.

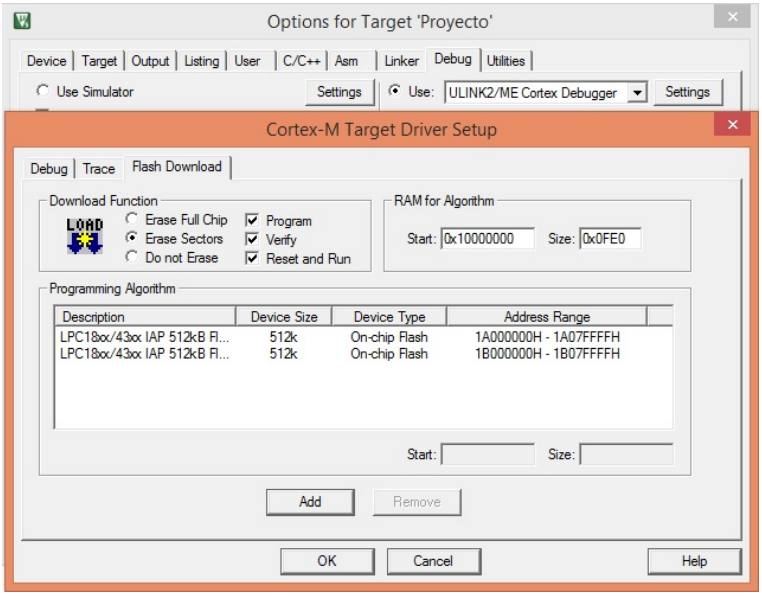

Figura 2.6: Ventana de opciones para cargar el proyecto

Ya teniendo todas las configuraciones, se puede proseguir a dar click en la **Barra de herramientas** , **Project** , **Rebuild All Target Files** . Se puede observar que todos los archivos: **main.c** , **LED_MCB4300.c** , **RTX_Conf_CM.c** , **GPIO_LPC43xx.c** , **SCU_LPC43xx.c** y **system_LPC43xx.c** tienen enlazados todas las librerías necesarias para cada uno. El último paso es dar click en **Barra de herramientas** , **Flash** , **Download**. Con esto ya se pueden crear proyectos más elaborados teniendo en cuenta estos sencillos pasos.

Capítulo 3

Usando el Archivo de Encabezado para Plantilla: cmsis_os.h

El archivo cmsis_os.h es un encabezado de plantilla que permite trabajar con todos los atributos de un Sistema Operativo en Tiempo Real, en este caso: el CMSIS-RTOS. Dicho archivo contiene la siguiente información [Keil]:

- Listas de comandos y acciones para definir hilos y otros objetos del núcleo del sistema operativo
- Definiciones de todas las funciones de la Interfaz de Programación de Aplicaciones (API) de CMSIS-RTOS
- Definiciones de estructuras para parámetros y tipos de retorno de información
- Valores de estado y proridades usados por las funciones de la API de CMSIS-RTOS

3.1. Funciones de CMSIS-RTOS

El Sistema Operativo en Tiempo Real de CMSIS maneja funciones que pueden ser llamados desde hilos y desde Rutinas de Servicio de Interrupciones (ISR en inglés). Estas funciones se muestran a continuación [Keil]:

- Manejo de hilos

- Activación de señales
- Manejo de rutinas mutuamente excluyentes
- Manejo de semáforos
- Manejo de almacenes de memoria
- Servicio de mensajes entre hilos
- Servicio de correos entre hilos

3.2. Convenciones de prefijos y manejo de hilos

Es importante mencionar que las definiciones que tengan el prefijo **os** se refieren a aquellas que tienen asignado un único espacio para las funciones del CMSIS-RTOS. Y aquellas que tengan asignado el prefijo **os_** no son usados por la API pero sí están referenciadas al archivo cmsis_rtos.h

La función de **Manejo de hilos** nos permite definir, crear y controlar las tareas de los hilos dentro del Sistema Operativo en Tiempo Real. La función **main** es una función de hilo especial que empieza al inicializar el sistema y tiene una prioridad inicial de *osPriorityNormal*

Cada hilo puede estar en uno de los cuatro siguientes estados:

- **ACTIVO:** Sólo un hilo a la vez puede estar en este estado
- **LISTO:** Aquél hilo que está a punto de pasar a ACTIVO está en este estado. Cuando un hilo está INACTIVO o EN ESPERA, aquél que esté en LISTO con la prioridad más alta pasa a estar ACTIVO.
- **EN ESPERA:** Aquél hilo que se encuentra esperando que un evento ocurra está en este estado.
- **INACTIVO:** Aquél hilo no creado o terminado está en este estado y no consumen recursos del sistema.

En la figura 3.1 se muestran los estados de los hilos, así como la transición entre estados.

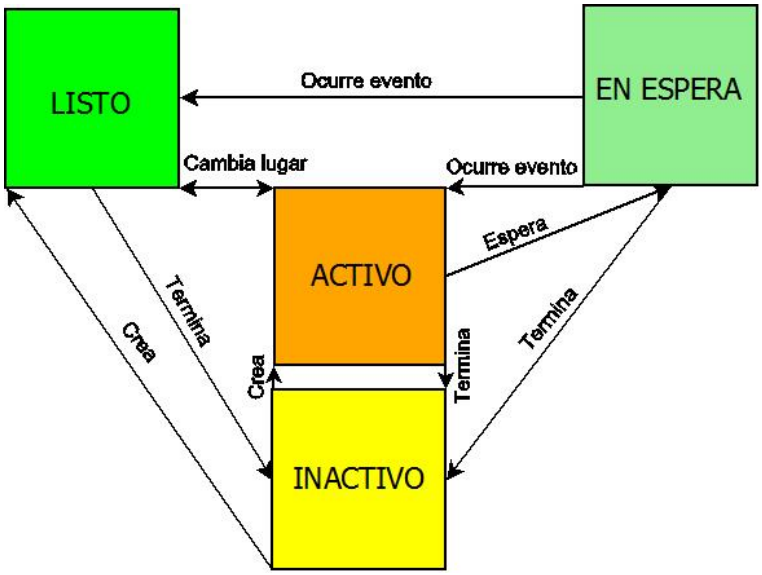

Figura 3.1: Estados y transiciones de hilos

Las funciones para poder utilizar los hilos de forma correcta están incluidos en la tabla 3.1.

Función	Descripción	Ejemplo
osThreadId	Define el nombre del hilo	osThreadId thread_faseA;
osThreadDef	Define atributos del hilo: nombre, prioridad inicial, número de instancias y tamaño para almacenar información	osThreadDef(faseA, osPriorityNormal, 1, 0);
osThreadCreate	Empieza el hilo y lo coloca en estado LISTO, además define su argumento inicial. Sí este hilo tiene mayor prioridad que el hilo actual ACTIVO, este pasa a ocupar su lugar	thread_faseA = osThreadCreate(osThread(faseA), NULL);
osThreadTerminate	Termina la tarea del hilo y lo remueve	osThreadTerminate (thread_faseA)
osThreadGetId	Obtiene el nombre del hilo ACTIVO	id = osThreadGetId ();
osThreadSetPriority	Cambia la prioridad del hilo	osThreadSetPriority (id, osPriorityBelowNormal);
osThreadGetPriority	Obtiene prioridad del hilo	priority = osThreadGetPriority (id)
osThreadYield	Pasa el control al siguiente hilo que está en estado LISTO	osThreadYield();

Cuadro 3.1: Funciones básicas de hilos

3.3. Cambio de varias tareas por Algoritmo de Round-Robin

El algoritmo de Round-Robin se usa en sistemas interactivos de tiempo compartido, en donde cada tarea puede realizarse en un tiempo máximo definido, llamdo *cuantum*. Sí la tarea termina antes del tiempo máximo, deja el procesador por sí mismo. En caso contrario, sí ocupa todo el tiempo, el procesador expulsa el proceso. [Cobo]:

La ventaja de ocupar el algoritmo de Round-Robin es que permite que se ejecuten varios hilos de forma casi-paralela. Esto es debido a que el cuantum que se define generalmente es de unos cuantos milisegundos y aparenta que las tareas se ejecuten de forma simultanea aunque realmente sea debido a que el RTOS asigna pequeños pedazos de tiempo a cada hilo.

Cuando termina el tiempo de ejecución del hilo o cuando éste termina su tarea, el CMSIS-RTOS cambia al siguiente hilo que está en estado listo y tiene la misma prioridad que la que terminó.Sí no hay hilos listos con la misma prioridad, se reanuda la ejecución del mismo hilo.

El cuantum se puede modificar dentro de las configuraciones del código llamado **RTX_Conf_CM.c** que se generan al compilar correctamente el proyecto. Se encuentran en la lista **System Configuration ->Round-Robin Thread switching [ACTIVAR] ->Round-Robin Timeout [ticks]**. Esto se muestra en la figura 3.2

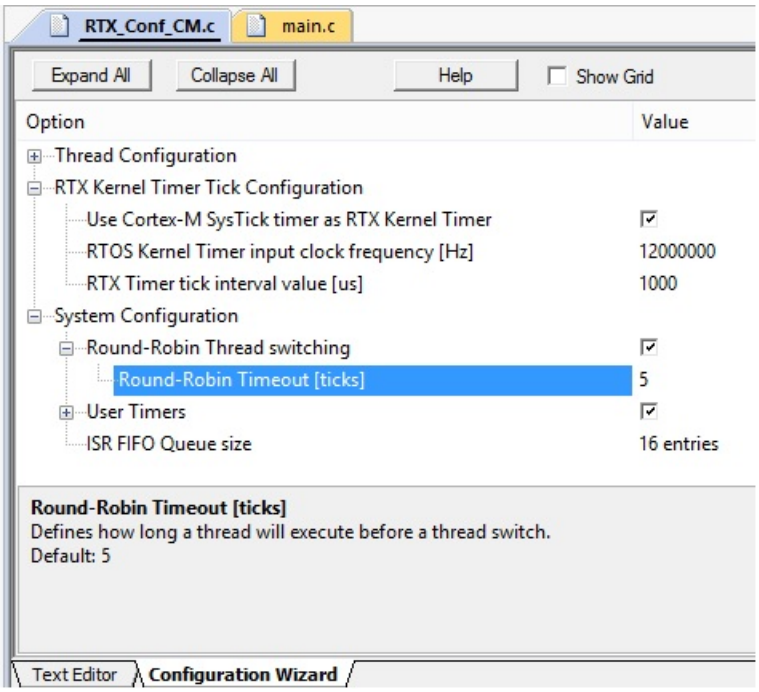

Figura 3.2: Configuraciones para el algoritmo de Round-Robin

En la figura 3.3 se observa un diagrama de un sistema mínimo usando el algoritmo de Round-Robin.

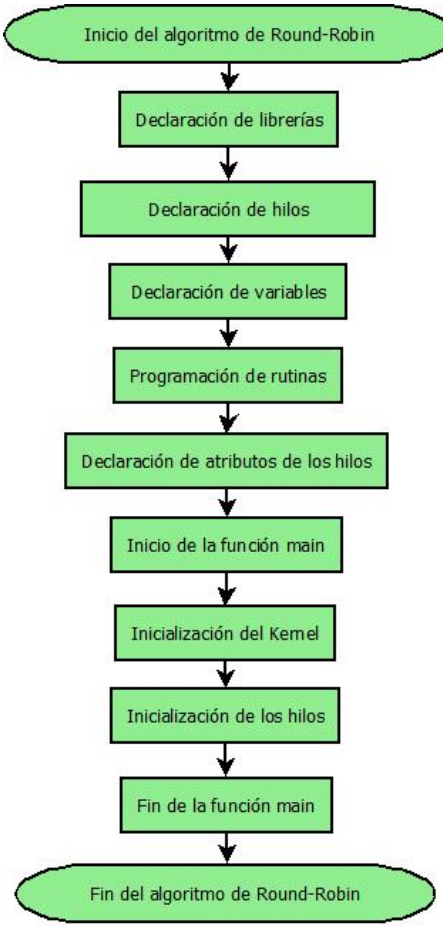

Figura 3.3: Diagrama de sistema mínimo usando algoritmo de Round-Robin

NOTA: El algoritmo de Round-Robin no se recomienda sí se quiere hacer programación con una estructura secuencial, debido al dificil control de ordenamiento de tareas. Se tiene el mismo problema sí dos o más hilos quieren acceder al mismo hardware: un hilo pondrá en espera al otro por lo que existirá un desfase de tiempos. Para estos casos, se recomienda utilizar otros algoritmos usando funciones de tipo mutuamente excluyentes o manejo de semáforos que se describen más adelante.

3.4. Manejo de señales

El manejo de señales permite controlar y esperar banderas de señales. Las funciones para poder utilizar estas señales de forma correcta están incluidos en la tabla 3.2.

Función	Descripción	Ejemplo
osFeature_Signals	Especifica el número de banderas máximas por hilo. Máximo pueden haber 31 banderas por hilo	#define osFeature_Signals 8
osSignalSet	Asigna una bandera a un hilo. Esta función se puede usar desde rutinas de interrupción	osSignalSet (hilo1, 0x0100);
osSignalWait	Suspende la ejecución del hilo ACTIVO hasta que todas las banderas están activadas. Después regresa o coloca al hilo EN ESPERA. Las banderas usadas como eventos se limpian automáticamente. En esta función se puede especificar cuánto tiempo el sistema espera a la bandera. Pueden ser en milisegundos o usando **osWaitForever** para que espere hasta que la señal se active	osSignalWait(0x0100, osWaitForever);
osSignalClear	Limpia las banderas especificadas de un hilo activo	osSignalClear (hilo1, 0x0100);

Cuadro 3.2: Funciones del manejo de señales

En la figura 3.4 se observa un diagrama de un sistema mínimo usando las funciones del manejo de señales

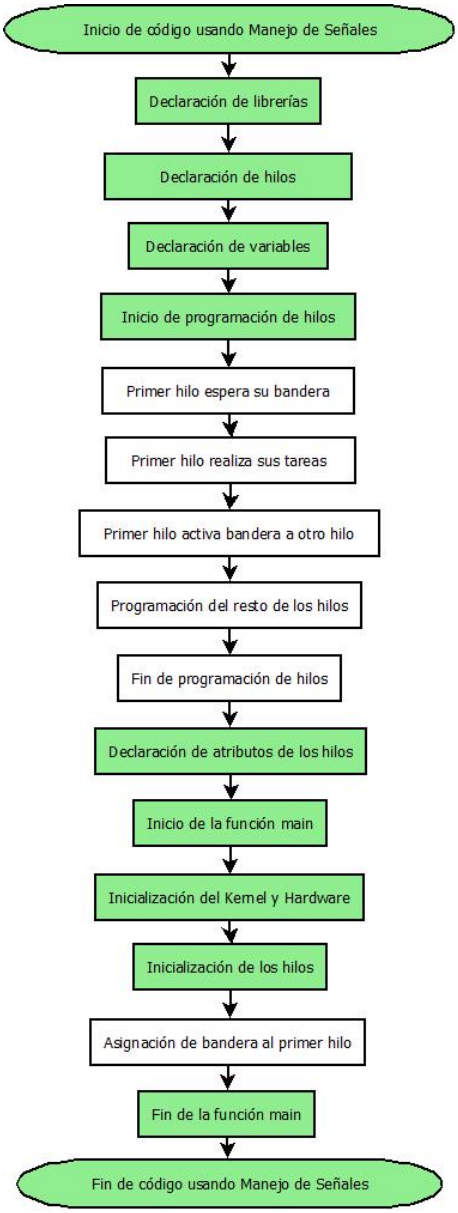

Figura 3.4: Diagrama de sistema mínimo usando manejo de señales

NOTA: El manejo de señales resulta útil cuando se quiere hacer programación secuencial y se requiere que un hilo se enlace directamente a otro al terminar. En la práctica, esto resulta complejo sí se tienen muchos hilos y

hay que enlazar un hilo con diferentes hilos dependiendo de las situaciones que se presenten. A continuación se describren otras funciones para esto.

3.5. Manejo de rutinas mutuamente excluyentes

El manejo de rutinas mutuamente excluyentes es usado para sincronizar la ejecución de hilos. Sobre todo se utiliza para proteger el acceso a un recurso compartido, por ejemplo: leds, memoria y demás periféricos. En la figura 3.5 se muestra el diagrama de estas funciones.

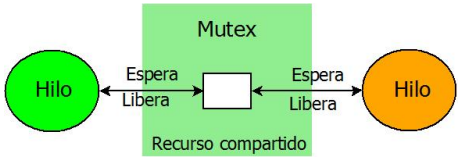

Figura 3.5: Rutinas mutuamente excluyentes usando la API de CMSIS-RTOS

Las funciones para poder utilizar el manejo de rutinas mutuamente excluyentes de forma correcta están incluidos en la tabla 3.3.

Función	Descripción	Ejemplo
osMutexDef	Define un objeto mutuamente excluyente	osMutexDef (Mutex1)
osMutexId	Asigna un nombre único al objeto mutuamente excluyente	osMutexId mutex_id
osMutexCreate	Crea e inicializa un objeto mutuamente excluyente	mutex_id = osMutexCreate (osMutex (Mutex1));
osMutexWait	Espera hasta que un objeto mutuamente excluyente está disponible. En esta función se puede especificar cuánto tiempo el sistema espera al mutex. Pueden ser en milisegundos o usando **osWaitForever** para que espere hasta que el mutex esté disponible	osMutexWait (mutex_id, 0);
osMutexRelease	Libera un mutex que fue obtenido por la función **osMutexWait**. Todos los demás hilos que estén esperando al mismo mutex ahora estarán en estado LISTO	osMutexRelease(mutex_id);
osMutexDelete	Borra un objeto mutex. La función libera el espacio en memoria que estaba ocupado por el objeto. Después de esta función, el nombre o identificador del mutex ya no se podrá ocupar hasta que se vuelva a crear el objeto usando la función **osMutexCreate**	osMutexDelete(mutex_id);

Cuadro 3.3: Funciones de rutinas mutuamente excluyentes

En la figura 3.6 se observa un diagrama de un sistema mínimo usando las funciones de rutinas mutuamente excluyentes.

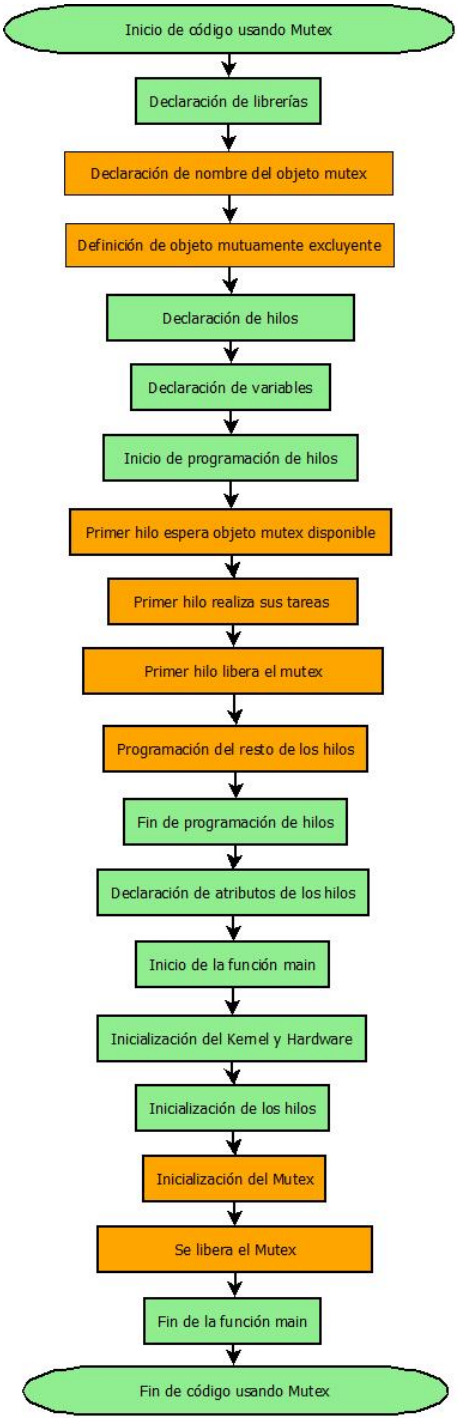

Figura 3.6: Diagrama de sistema mínimo usando rutinas mutuamente excluyentes

LSE001-2014
Elaboró: Dr. Casimiro Gómez González

NOTA: El manejo de rutinas mutuamente excluyentes es útil debido a que, a diferencia del manejo de señales, en estas se puede liberar el objeto mutuamente excluyente en una rutina y cualquier otra lo puede esperar, aún sin importar que sólo en algunas ocasiones esta se tiene que activar. Una de las desventajas de este método es que sí se quiere acceder a varios dispositivos en común, se vuelve complicado su manejo. Para ello se recomienda el manejo de semáforos que se describe a continuación.

Anexos

Anexos A

Anexo I: Código para crear un nuevo proyecto

```
#include "cmsis_os.h"
#include "LPC43xx.h"
#include "Board_LED.h"

/* Nombres de los hilos        */
osThreadId thread_faseA;
osThreadId thread_faseB;
osThreadId thread_reloj;

/* Primer hilo */
void faseA (void const *argument) {
for (;;) {
osSignalWait(0x0001, osWaitForever);
LED_SetOut       (0);
osSignalSet(thread_reloj, 0x0100);
osDelay(1000);
osSignalSet(thread_faseB, 0x0001);
//LED_Off(0);
}
}

/* Segundo hilo */
```

```
void faseB (void const *argument) {
for (;;) {
osSignalWait(0x0001, osWaitForever);
LED_SetOut       (254);
osSignalSet(thread_reloj, 0x0100);
osDelay(1000);
osSignalSet(thread_faseA, 0x0001);
//LED_Off(1);
}
}

/*Tercer hilo*/
void reloj (void  const *argument) {
for (;;) {
osSignalWait(0x0100, osWaitForever);
LED_On(7);
osDelay(500);
LED_Off(7);
osDelay(500);
}
}

osThreadDef(faseA, osPriorityNormal, 1, 0);
osThreadDef(faseB, osPriorityNormal, 1, 0);
osThreadDef(reloj,  osPriorityNormal, 1, 0);

/* Funcion main */
int main (void) {

LED_Initialize();
thread_faseA = osThreadCreate(osThread(faseA), NULL);
thread_faseB = osThreadCreate(osThread(faseB), NULL);
thread_reloj  = osThreadCreate(osThread(reloj),  NULL);
osSignalSet(thread_faseA, 0x0001);
}
```

Bibliografía

[Keil] ARM Ltd.(2014, Septiembre 24) *CMSIS-RTOS API: Generic RTOS Interface for Cortex-M processor-based devices* Recuperado de: http://www.keil.com

[Cobo] Cobo, Pablo Martínex «Sistemas operativos: teoría y práctica», *Díaz de Santos*, págs. 64–65, 1997.