SISTEMAS BASADOS EN MICROPROCESADOR
PRÁCTICA 4. MULTITAREA COOPERATIVA

Grado en Ingeniería Electrónica Industrial (3er curso)
Curso 2025/2026

1 INTRODUCCIÓN

Una forma rudimentaria para realizar una multiprogramación es la multitarea cooperativa, tal como se ha descrito en teoría. Esta práctica se va a centrar en aprovechar parte de las subrutinas de las anteriores sesiones para desarrollar un programa que ejecute tres procesos independientes con una temporización precisa: una animación de leds, otra de los segmentos de la batería del LCD y una visualización de dígitos en el display de 7 segmentos.

1.1 Objetivos

- Establecer un sistema de temporización precisa.
- Abordar un sistema de multitarea cooperativa sencillo con tres procesos.

2 ESTUDIO TEÓRICO

Busque y estudie el *datasheet* del MSP430FR6989, la hoja de datos del *Launchpad*, así como la documentación del *Clock System* (`cs.pdf`) en la información de examen de la página web de la asignatura. En ella se recogen los registros de control del módulo de reloj del FR6989. Indique cuáles son las condiciones de arranque en el reset de los siguientes elementos:

- Velocidad del DCO: _____
- Fuente y velocidad del MCLK: _____
- Fuente y velocidad del SMCLK: _____
- Fuente y velocidad del ACLK: _____
- Estado del oscilador LFXT: _____
- Estado del oscilador HFXT: _____
- Estado del oscilador VLO: _____
- Tolerancia de la velocidad del DCO respecto de la nominal: _____

Busque en la hoja de datos del FR6989 en qué puertos están los terminales del oscilador LFXT:

- Anote aquí los puertos: _____
- Anote la función por defecto de los mismos: _____

Al arrancar el `ACLK` está conectado al oscilador de cristal de baja frecuencia, pero como el mismo está parado al estar los puertos de E/S bloqueados y no poderse conectar al cristal externo, el sistema lo reconecta automáticamente a una fuente de baja frecuencia que sí esté disponible. Indique cuál, qué velocidad tiene y en qué rango puede estar la misma:

- Fuente de ACLK: _____
- Frecuencia nominal: _____
- Rango: _____

Compárelos con las especificaciones del LFXT:

- Frecuencia nominal: _____
- Tiempo que tarda en arrancar: _____

Busque y estudie en la documentación de examen de la página web de la asignatura la documentación del Timer A (`TimerA.pdf`). En ella se recogen los registros de control de los

distintas instancias del módulo Timer A del MSP430FR6989. Estudie también los conceptos de multitarea cooperativa.

3 ESTUDIO PRÁCTICO

3.1 Preparando el proyecto

Copie el proyecto de la práctica anterior y llámelo P4.1. Se van a reaprovechar todos sus módulos, así como parte del programa principal en C.

3.2 Módulo de gestión del reloj `cs.asm`

El módulo de reloj recogerá aquellas funciones relacionadas con los relojes del MSP430. Haga un nuevo módulo para albergarlas. Para ello seleccione el proyecto en el *Project Explorer* y pulse sobre *File/New/Source File...*. Llame al fichero `cs.asm` y seleccione "*Ensamblador*" en "*Template*". No olvide hacer también el `cs.h` e incluya en él los prototipos de las funciones de `cs.asm`. Recuerde también incluir las etiquetas de las funciones accesibles desde el exterior en la directiva `.globals` y los prototipos en `cs.h`.

3.2.1 Inicializa el oscilador de baja frecuencia `csIniLFXT`

```
void csIniLFXT (void);
```

Vamos a arrancar el oscilador de cristal de baja frecuencia para que la frecuencia del mismo sea 32768Hz de forma precisa. Las tareas que realiza son:

- Configurar los pines de E/S para que tengan la función de los pines del oscilador.
- Habilitar el oscilador LFXT.
- Esperar hasta que el oscilador se estabilice[1]. Para ello debe borrar el flag `LFXTOFFG` de `CSCTL5` que indica error en el oscilador LFXT, así como el flag `OFIFG` del registro `SFRIFG1` que indica error en *un* oscilador. Hacer un bucle que repita este paso mientras `LFXTOFFG` siga activo.
- Recuerde que los registros del módulo de reloj están protegidos por contraseña. Desbloquéelos al principio y bloquéelos al final.

3.3 Módulo de gestión del SytemTimer `st.asm`

Se necesitará añadir un nuevo fichero en ensamblador al proyecto. Puede partir de un fichero vacío o copiar uno de los disponibles (`lcd.asm`, `pt.asm`). Llámelo `st.asm`. Si ha copiado uno anterior, borre el código y las etiquetas de la directiva `.global`. No olvide crear también el fichero de cabecera asociado `st.h` en el que incluirá los prototipos de las funciones de este módulo y que deberá incluir en cualquier fichero que use sus servicios.

Defina una variable que albergará el valor actual del tiempo del sistema en TICS de reloj y no la incluya en la lista de símbolos publicados (`.global`):

1. Cuando se activa un oscilador de cuarzo, éste no oscila de forma estable hasta pasado un tiempo. El FR6989 dispone de un circuito que monitoriza cada uno de los osciladores externos y activa un flag en caso de que el mismo esté en fallo (oscilación no estable). Hay un flag para el oscilador de baja frecuencia (`LFXTOFFG`), otro para el de alta frecuencia (`HFXTOFFG`) y otro que se activa cuando alguno ha fallado (`OFIFG`).

```
.bss    SystemTimer, 4
```

También se necesita una variable para almacenar el valor inicial del CCR0, que también es el periodo, ya que el timer se va a configurar en modo continuo:

```
.bss    stPeriodo, 2
```

Los prototipos de este módulo son:

3.3.1 Inicialización `stIni`

Inicializa el *system timer*. Para ello configura el TimerA2 en modo `CONTINUOUS` a 32768Hz (fuente ACLK y divisor 1). `periodo` es el valor a cargar en CCR0 que va a determinar el valor de la interrupción periódica. Dado que estamos en modo continuo, la subrutina deberá guardar localmente el `periodo` en `stPeriodo` para posteriores actualizaciones:

```
void stIni (uint16_t periodo);
```

3.3.2 Lectura del tiempo `stTime`

Devuelve el valor actual del *system timer*. Observe que la variable ocupa dos palabras y ,por tanto, la lectura no es atómica[2]. Para evitar condiciones de carrera[3] con la interrupción periódica, asegúrese de que las mismas no ocurren entre las dos lecturas:

```
uint32_t stTime (void);
```

3.3.3 Subrutina de servicio de interrupción `stA2ISR`

La subrutina de servicio de interrupción también se definirá en este módulo, pero no se publicará para que no sea accesible desde el exterior (es decir, no irá en `.global`). El cometido de la interrupción será incrementar la variable `SystemTimer`. También deberá actualizar el valor del CCR0 para que la próxima IRQ se produzca en tiempo y forma.

3.4 Programa principal en C

A modo de demostración, haga un programa que configure los módulos adecuadamente y gestione un *superbucle* con tres tareas cooperativas:

- Animación de la pantalla LCD. Misma funcionalidad que en la práctica 2, pero sustiyendo la temporización mediante un bucle con espera activa, por el nuevo método de tareas periódicas y mostrando su nombre y dos apellidos. Frecuencia de la tarea, 3 Hz.
- Animación de los segmentos de capacidad de batería del LCD (B1 a B6) de forma que inicialmente se encienda B1, pasado un tiempo, se apague B1 y se encienda B2 y así sucesivamente hasta B6. Una vez terminado el barrido de izquierda a derecha, se repetirá de derecha a izquierda. Después repita la animación al completo. Debe mantener encendidos en todo momento los segmentos del marco de la batería y el polo positivo. Frecuencia de la tarea: 8 Hz.
- Gestión de los leds de usuario del Launchpad:
  - El led1 se encenderá cuando se pulse el switch1. Además, cuando se suelte el pulsador, se alargará el encendido del mismo durante un segundo.

2. Una operación se llama atómica cuando no puede ser interrumpida. Una instrucción siempre es atómica.
3. Una condición de carrera (*race condition*) ocurre cuando dos o más procesos acceden un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada.

- El led2 se encenderá cuando se pulse el switch2. Además, cuando se suelte el pulsador, se alargará el encendido del led2 hasta igualar un tiempo mínimo de encendido de 3 segundos siempre que el switch2 se haya pulsado menos de ese tiempo.

3.4.1 Reduciendo el consumo

Una vez que se ha dado una pasada al superbucle, no hay nada que hacer hasta que se incremente el `SystemTimer`. Por ello, lo mejor es dormir la CPU y entrar en un modo de bajo consumo. Puesto que se requiere el ACLK activo para alimentar el TA2, no se puede ir más allá de LPM3. Modifique el superbucle para entrar en LPM3 al final del mismo. También tendrá que modificar la ISR del `TimerA2` para que el micro se despierte tras la ISR y pueda volver al superbucle para dar atención a las tareas cooperativas.