

# Práctica 3. Temporizadores

(Duración: 1 sesión)

## Introducción

En esta práctica se usará la tarjeta PicTrainer32 para familiarizarse con los temporizadores del PIC32MX230F064D.

## Objetivos

Al concluir la práctica, el alumno deberá ser capaz de:

- Manejar los temporizadores del PIC32MX230F064D en modo *polling*.
- Realizar programas sencillos que interactúen con el *hardware* en los que intervengan retrasos.
- Depurar programas para encontrar sus fallos.

## Trabajo previo

Antes de ir al laboratorio ha de:

- Leer detenidamente la práctica, anotando las dudas que le surjan para preguntárselas al profesor.
- Escribir los programas y funciones solicitados en las secciones 2, 3, 4 y 6.

## 1. Crear un proyecto

En primer lugar ha de crear un proyecto para esta práctica, siguiendo los pasos indicados en el apartado 2 de la primera práctica. No olvide trabajar en la carpeta `D:\Micros\Grupo\P3`, creada en la práctica 1.

Para que funcione la tarjeta con el oscilador de cuarzo externo es necesario configurar una serie de bits en los registros de configuración del microcontrolador. Como la configuración del oscilador es algo compleja y no se ha visto en clase aún, en Moodle encontrará el archivo `Pic32Ini.c` con su correspondiente `.h` el que se hace dicha configuración usando una serie de sentencias `pragma` del compilador. Para que el microcontrolador funcione a 40 MHz y el bus de periféricos a 5 MHz basta con que descargue el archivo `Pic32Ini.c` en la carpeta del proyecto y lo incluya en dicho proyecto.

Si observa el archivo `Pic32Ini.c` verá que se incluye una función denominada `InicializarReloj`. Dicha función se ha incluido por si se quiere cambiar la configuración del oscilador durante la ejecución del programa por lo que en nuestro caso **no es necesario llamarla**.

## 2. Generación de retardos

Tomando como base los ejemplos de los apuntes, escriba un programa para que el LED conectado a RC3 parpadee con una frecuencia de 0,5 Hz. El retardo necesario se generará usando el temporizador 2. El resto de los LEDs de la placa permanecerán apagados.

### 3. Función para generar retardos

Escriba una función para generar un retardo usando el temporizador 2. Dicha función tendrá el siguiente prototipo:

```
int Retardo(uint16_t retardo_ms);
```

en donde `retardo_ms` es el retardo deseado en milisegundos.

Si el retardo es 0, la función sale inmediatamente. En caso contrario, la función ha de obtener en primer lugar el valor del divisor de reloj necesario para que la cuenta pueda realizarse con 16 bits y a continuación configurar el temporizador para esperar dicho tiempo. Antes de salir de la función el temporizador deberá desactivarse para ahorrar energía.

Por último, la función devolverá un 0 si se ha podido generar el retardo o un 1 si el retardo solicitado es demasiado grande.

Se valorará el hecho de situar esta función en otro archivo, denominado `Retardo.c` con su consiguiente archivo cabecera (`Retardo.h`) que incluya el prototipo de la función `Retardo`.

### 4. Generación de retardos II

Modifique el programa de la sección 2 para que el retardo necesario se genere usando la función escrita en la sección anterior.

### 5. LED latiendo

Realice un programa para que el LED RC0 parpadee con una frecuencia de 50 Hz, pero con un tiempo de encendido variable. Como el periodo es de 20 ms, en la primera iteración del programa el LED permanecerá apagado todo el tiempo, en la segunda estará encendido durante un milisegundo y apagado el resto del tiempo y así sucesivamente hasta que el LED permanezca los 20 ms encendido. A partir de entonces la secuencia se invertirá, estando en la siguiente iteración 19 ms encendido y 1 ms apagado, en la siguiente 18 ms encendido y 2 ms apagado, y así sucesivamente hasta que el LED se apague completamente, momento en el que se volverá a repetir la secuencia. **Evite usar bucles innecesarios dentro del `while(1)`.**

### 6. LED latiendo + Generación de retardos II (opcional)

Combine ambos apartados usando el periodo de la señal (20ms) para contar el tiempo necesario (que es superior) y generar la señal del apartado 4.