UNIVERSIDAD PONTIFICIA ICAI ICADE COMILLAS MADRID

6 DE FEBRERO DE 2026

# INFORME PRÁCTICA 3
## TEMPORIZADORES

GUILLERMO FUENTES, IRENE MORALES

# Índice

# Introducción

En esta práctica de laboratorio, el objetivo principal es familiarizarse con el uso de los temporizadores del PIC32MX230F064D trabajando en modo polling (sin interrupciones), y aplicarlos a programas sencillos que interactúan con el hardware mediante la generación de retardos temporales.

Para ello, se parte de programas básicos donde el temporizador genera esperas fijas para controlar el parpadeo de un LED, y se evoluciona hacia una solución más reutilizable implementando una función de retardo en milisegundos. Finalmente, se aplica todo lo anterior a un caso más complejo: un LED que mantiene un periodo constante, pero con un tiempo de encendido variable. Como ampliación opcional, se combinan ambos comportamientos utilizando el periodo fijo de 20 ms como base temporal para generar, mediante un contador, una señal más lenta de 2 segundos, evitando así el uso de retardos largos adicionales.

# Código y Desarrollo Práctico

## Generación de Retardos

Para empezar con la práctica configuramos los puertos A, B y C para controlar el LED conectado al pin RC3, el cual parpadeará con una frecuencia de 0,5 Hz. Comenzamos configurando los pines para que todos actúen como digitales poniendo a 0 el valor del ANSELC. Después, configuramos que el estado inicial del LED sea apagado, ya que con LATC estamos poniendo a 1 el bit del LED cuando este es activo a nivel bajo.

Con el registro TRIS definimos cómo queremos que actúen los pines, es decir, si queremos que se comporten como entradas o como salidas, y como podemos ver, estamos poniendo como salidas todos los pines en los puertos A y B, mientras que el puerto C solamente ponemos como salidas los últimos 4 bits.

Una vez configurados los registros, nos dedicamos a programar el temporizador número 2, el cual es de tipo B. Su estructura es muy sencilla y es la siguiente. Primero paramos la cuenta y borramos esta misma T2CON = 0 ; TMR2 = 0. Borramos el flag IFS0bits.T2IF = 0. Y establecemos la cuenta a la que queremos llegar que en este caso es 19531, ya que al pedirnos que el LED parpadee a una frecuencia de 0,5 Hz, es decir cada 2 segundos decidimos que el LED debe estar encendido durante 1s y apagado durante otro segundo, además como es un número muy grande lo que hemos hecho para que la cuenta se pueda dar en un único temporizador es utilizar el prescaler de 256.

$$\frac{5 \times 10^6}{256} = 19531$$

Por último y terminar con la configuración del temporizador lo encendemos T2CON = 0x8070 indicando el prescaler que hemos utilizado e indicamos dentro del bucle que el temporizador espere a que la cuenta se haya completado y en ese momento se invierta el estado del LED.

```
#include <xc.h>
#include "Pic32Ini.h"

#define PIN_LED 3

int main(void){
    ANSELC = 0x0000;
    LATC = 0xFFFF;
    TRISC = 0xFFF0;
    TRISA = 0x0000;
    TRISB = 0x0000;

    T2CON = 0;
    TMR2 = 0;
    IFS0bits.T2IF = 0;
    //IFS0 &= ~(1 << 9);
    PR2 = 19531;
    T2CON = 0x8070;

    while (1) {
        while(IFS0bits.T2IF == 0);
        IFS0bits.T2IF = 0;
        LATC ^= (1<<PIN_LED);
    }
    return 0;
}
```

## Función para generar retardos

En este apartado nos centraremos en la programación de una función que nos permite generar retardos para que la cuenta se realice en un único temporizados, esta función lo que irá probando son los diferentes prescalers para que dicha cuenta quepa en el temporizador mencionado. Es decir, empezará comprobando si la cuenta cabe y en caso de no ser así aumentará el prescaler encontrando así el prescaler de menor tamaño para conseguir una mayor resolución.

Antes de nada, queremos que el usuario de esta función nos indique el retardo que quiere realizar en milisegundos. Como hemos mencionado, lo primero que tenemos en la función es la inicialización de las variables t_PR a 0 y una lista de entero donde meteremos todos los posibles valores del prescaler, en este caso al utilizar el timer 2 que es de tipo B, tenemos los valores 1, 2, 4, 8, 16, 32, 64, 128 y 256. Una vez configurado el timer de la misma manera que hemos hecho anteriormente, comprobamos si el retardo es diferente a 0 o no, en caso de ser 0, no hay espera, es decir, mandamos la acción inmediatamente. De serlo, la función

entrará en un bucle que se iterará las veces que sea necesario para encontrar el prescaler idóneo. Una vez lo encuentre, realizaremos un simple cálculo para determinar en ticks la duración del retardo. Para finalizar con este apartado se configura PR2 con la cuenta que acabamos de sacar, encendemos el temporizador

```
#include <xc.h>
#include <stdint.h>
#include "Pic32Ini.h"

int Retardo(uint16_t retardo_ms) {

    int t_PR=0;
    int preescalados[] = {1, 2, 4, 8, 16, 32, 64, 256};
    int i = 0, tckps= -1;
    T2CON = 0;
    TMR2 = 0;
    IFS0bits.T2IF = 0;

    if (retardo_ms != 0) {

        for (i = 0; i < 8; i++) {
            t_PR = ((retardo_ms*5000)/ preescalados[i])-1;
            if  (t_PR <= 65535) {
                tckps = i;
                break;
            }
        }

        if (tckps==-1){
            return 1;
        }

    } else {
        return 0;
    }

    PR2 = t_PR;
    T2CON = 0x8000 | (tckps << 4);

    while (IFS0bits.T2IF == 0);
    IFS0bits.T2IF = 0;
    T2CON = 0x0000;
    return 0;
}
```

y esperamos a que el temporizador haya terminado la cuenta. En ese momento apagaremos el temporizador y terminaremos con la función.

## Generación de Retardos II

En este apartado implementaremos la función que hemos programado en el punto anterior en el programa del inicio del informe. Es un proceso muy simple, únicamente crearemos una variable el valor del retardo deseado en milisegundos, que como dijimos previamente es de 1s, es decir, 1000ms y lo añadiremos dentro del bucle while(1), una vez se ejecute nuestra función y haya terminado la cuenta deseada, el estado del LED se invertirá. Cabe recalcar que el programa funcionaría

de la misma manera, aunque no definamos la variable retardo_ms y directamente pusiésemos el valor en Retardo y quedaría tal que así Retardo(1000).

```
#include <xc.h>
#include "Pic32Ini.h"
#include "Retardo.h"

#define PIN_LED 3

int main(void){

    ANSELC = 0x0000;
    LATC = 0xFFFF;
    TRISC = 0xFFF0;
    TRISA = 0x0000;
    TRISB = 0x0000;

    uint16_t retardo_ms = 1000;

    while (1) {
        Retardo(retardo_ms);
        LATC ^= (1<<PIN_LED);
    }
    return 0;
}
```

## LED latiendo

En este apartado se desarrolla un programa que genera un efecto de “latido” en el LED conectado al pin RC0. El objetivo no es simplemente hacerlo parpadear, sino mantener un periodo fijo de 20 ms (50 Hz) mientras se modifica progresivamente el tiempo que permanece encendido dentro de cada periodo.

Una vez inicializado el sistema, el programa entra en el bucle infinito while(1), donde se ejecuta repetidamente el comportamiento temporal deseado. En cada iteración se genera un periodo completo de 20 ms dividido en dos fases: encendido (ON) y apagado (OFF). Para esto, se hace uso de la función Retardo(ms), implementada previamente mediante el temporizador en modo polling.

Durante la fase ON, el LED se activa y se llama a Retardo(on_ms), manteniéndolo encendido el número de milisegundos indicado por la variable on_ms. A continuación, en la fase OFF, el LED se apaga y se ejecuta Retardo(PERIODO_MS - on_ms), completando así el tiempo restante hasta llegar a los 20 ms totales. De esta manera, se asegura que la suma de ambos tiempos es siempre constante y el periodo no varía.

El efecto de “latido” se consigue modificando progresivamente el valor de on_ms. Esta variable aumenta desde 0 hasta 20 ms y, una vez alcanzado el máximo, comienza a disminuir hasta volver a 0. Para controlar este comportamiento se utiliza una variable de dirección que indica si el tiempo de encendido debe

incrementarse o decrementarse en cada ciclo. Cuando se alcanza uno de los extremos (0 o 20 ms), se invierte el sentido de la variación. Lo que se observa en la

```c
#include <xc.h>
#include "Retardo.h"
#include "Pic32Ini.h"

#define PIN_LED 0
#define PERIODO_MS 20

int main(void) {

    ANSELC = 0x0000;
    LATC = 0xFFFF;
    TRISC = 0xFFF0;
    TRISA = 0x0000;
    TRISB = 0x0000;

    uint16_t on_ms = 0;
    int dir = 1;

    while (1) {

        // ON
        LATC &= ~(1 << PIN_LED); //pongo 0 en el LED
        Retardo(on_ms); //espero 0s en 1ª iteracion 1ms en la 2ª...

        // OFF
        LATC |= (1 << PIN_LED); //pongo 1 en el LED
        Retardo(PERIODO_MS - on_ms);

        // actualizar on_ms para el efecto "latido"
        if (on_ms == PERIODO_MS) {
            dir = -1;
        }
        else if (on_ms == 0){
            dir = +1;
        }
        on_ms = (uint16_t)((int)on_ms + dir);
    }
    return 0;
}
```

placa es como el LED pasa gradualmente de estar casi apagado a casi completamente encendido y viceversa, generando ese efecto visual.

## LED latiendo + Generación de Retardos II

En este apartado se amplía el programa anterior combinando el efecto de “latido” del LED conectado al pin RC0 con un segundo comportamiento temporal: el parpadeo del LED conectado al pin RC3 con un periodo de 2 segundos (0,5 Hz). La característica principal de esta implementación es que no se emplea un retardo independiente de 2 s, sino que se utiliza como base de tiempo el periodo fijo de 20 ms ya generado para el efecto de latido.

Al igual que en el apartado anterior, el programa mantiene un periodo constante de 20 ms dentro del bucle infinito while(1), dividido en dos fases: encendido (ON) y apagado (OFF) del LED RC0.

La novedad introducida consiste en utilizar cada periodo de 20 ms como una unidad básica de tiempo. Para ello se incorpora una variable ticks que se incrementa una vez en cada iteración completa del bucle, es decir, cada 20 ms. Cuando el contador alcanza el valor 100, se han acumulado:

$$100 * 20 \text{ ms} = 2000 \text{ ms} = 2 \text{ s}$$

En ese momento se invierte el estado del LED RC3, y el contador se reinicia a cero. Así se genera un parpadeo con periodo de 2 segundos utilizando como referencia temporal la señal de 20 ms ya existente.

En la placa se observa simultáneamente el efecto de latido suave en RC0 y el parpadeo lento en RC3, ambos sincronizados con la misma base temporal de 20 ms.

```c
#include <xc.h>
#include "Retardo.h"
#include "Pic32Ini.h"

#define PIN_RC0 0
#define PERIODO_MS 20
#define PIN_RC3 3

int main(void) {

    ANSELC = 0x0000;
    LATC = 0xFFFF;
    TRISC = 0xFFF0;
    TRISA = 0x0000;
    TRISB = 0x0000;

    uint16_t on_ms = 0;
    uint16_t ticks = 0;
    int dir = 1;

    while (1) {

        // RC0 ON
        LATC &= ~(1 << PIN_RC0);
        Retardo(on_ms);

        // RC0 OFF
        LATC |= (1 << PIN_RC0);
        Retardo(PERIODO_MS - on_ms);

        // latido
        if (on_ms == PERIODO_MS) {
            dir = -1;
        }
        else if (on_ms == 0){
            dir = +1;
        }
        on_ms = (uint16_t)((int)on_ms + dir);

        // parpadeo RC3 de 2s con base 20ms
        ticks++;
        if (ticks==100){ //100*20ms=2s
            LATC ^= (1 << PIN_RC3);
            ticks = 0;
        }
    }
    return 0;
}
```

# Conclusión

En esta práctica se ha trabajado el uso del temporizador Timer2 en modo polling para generar retardos temporales y controlar el comportamiento de distintos LEDs. Se ha comprendido cómo configurar el temporizador, seleccionar el prescaler adecuado y relacionar la frecuencia del sistema con el tiempo real obtenido.

La implementación de la función Retardo(ms) ha permitido reutilizar el temporizador de forma flexible, facilitando la generación de diferentes comportamientos temporales sin reconfigurar constantemente el hardware. Finalmente, se ha aplicado una base de tiempo fija de 20 ms para construir señales más lentas mediante un contador, demostrando cómo a partir de un periodo constante pueden generarse tiempos mayores de forma estructurada.

En conjunto, la práctica ha servido para consolidar los fundamentos de temporización y preparar el terreno para el trabajo posterior con interrupciones.