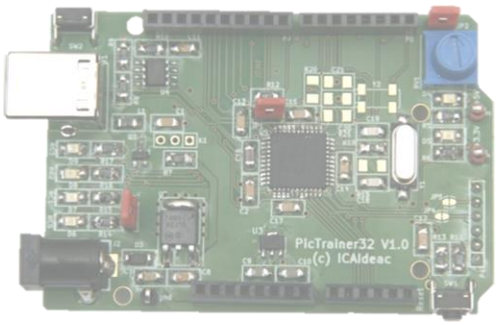
Microprocesadores

Práctica 3
Temporizadores

Francisco Martín Martínez



Jorge Calvar, Fernando Santana
5 de febrero de 2021

Contenido

Introducción

En esta práctica nos familiarizaremos con los temporizadores del PIC32MX230F064D. También aprenderemos a realizar programas que incluyan otros ficheros .c y a crear archivos de cabecera.

Generación de retardos

En este ejercicio haremos parpadear el LED RC0 con una frecuencia de 1Hz. Por tanto, ya que el periodo es de 1 segundo, estará medio segundo encendido y medio apagado.

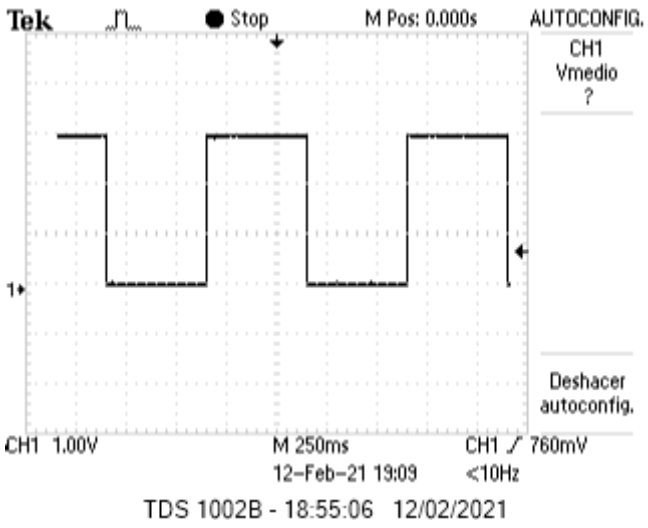
```
/*
 * File:   main2.c
 * Author: Jorge & Fernando
 *
 * Created on February 5, 2020
 */
#include <xc.h>
#include "Pic32Ini.h"
#define PIN_LED 0

int main(void) {

    InicializarReloj();

    ANSELC &= ~(1 << PIN_LED); // Pines del LED como digital

    LATA = 0; // En el arranque dejamos todas las salidas a 0
    LATB = 0;
    LATC = 0xF; // Excepto el LED, que es activo nivel bajo

    TRISA = 0; // Como salidas
    TRISB = 0;
    TRISC = 0;

    T2CON = 0; //Parar el temporizador
    TMR2 = 0; //Cuenta a 0
    IFS0bits.T2IF = 0; // Se borra el bit de fin de cuenta
    PR2 = 39061; //0.5/(64*200ns)-1=39061.5
    T2CON = 0x8060;

    while (1) {
      // Espera el fin del temporizador
        while (IFS0bits.T2IF == 0);
        IFS0bits.T2IF = 0;
      //Cada vez que llegamos a fin temp
        LATC ^= 1 << PIN_LED;//Inv LED
    }
}
```

Calculamos el preescalado con la siguiente fórmula

$$PR2 = \frac{t_{espera}}{div \cdot T_{PBCLK}} - 1$$

Ponemos T2CON en ON con preescalado 6 y reloj interno.

En la captura del osciloscopio podemos ver como el Led parpadea con frecuencia 1Hz encendiéndose 500ms=250ms*2 divisiones de la escala y apagándose otros 500ms.



Función para generar retardos

Ahora, para simplificar la creación de retardos, creamos una función con este propósito. Esta función convertirá en primer lugar su argumento (retardo en milisegundos) al contador teniendo en cuenta el reloj de la placa. La principal complejidad de esta primera parte es tener en cuenta el *prescaling*, que permite contar periodos mucho mayores.

```
/*
 * File:   retardo.h
 * Author: Jorge & Fernando
 *
 * Created on February 5, 2020
 */
int Retardo (uint16_t retardo_ms);
```

Primero creamos la cabecera con extensión .h donde únicamente se incluye el prototipo de la función.

Además, necesitamos crear la función con extensión .c donde incluimos el fichero.h y describimos la función

```
/*
 * File:   main5.c
 * Author: Jorge & Fernando
 *
 * Created on February 5, 2020
 */

#include <xc.h>
#include "retardo.h"

int Retardo (uint16_t retardo_ms){
    //CALCULO SI SE PUEDO GENERAR RETARDO CON 16 BITS
    int periodo;
    int i;

    //si el retardo es 0 o superior al que podemos generar con el maximo preesc
    //return 1 y acaba la función
    if(retardo_ms==0)
        return 1;
    if(retardo_ms>=3355.5)
        return 1;

    //bucle para probar con los distintos preescalados
    for(i=1; i<8 ; i++){
        if (i==7)
            i=8;
        //el preescalado 7 es en realidad 2^8=256

        periodo = (5000*retardo_ms/(1 << i)) -1; //(ret/200ns*2^TCKPS)-1
        if(periodo<=65535){
            break; //Puedo realizar retardo, salgo del bucle
        }
    }
    if(i==8)
        i = 7;
    //recordamos que aunque i=8 para poder conseguir preesc 256
    //en realidad es el preescalado 7


    // GENERO UN RETARDO
```

```
    T2CON=0; //Parar el temporizador
    TMR2=0; //Cuenta a 0
    IFS0bits.T2IF = 0; // Se borra el bit de fin de cuenta
    PR2=periodo;

    T2CONbits.TCKPS=i; //Ponemos preescaler
    T2CONbits.ON= 1; //ENCENDEMOS

    while(IFS0bits.T2IF==0); //Espera fin de la cuenta

    T2CONbits.ON=0; //APAGAMOS
    return 0;
}
```

Primero se comprueba que se va a poder generar el retardo y si es posible se pasa a un bucle donde se prueban los distintos preescalados empezando por el mínimo. Un preescalado no es suficiente y tenemos que pasar a uno mayor cuando el PR es superior a 2^16-1=65535. Importante tener en cuenta que el preescalado 7 es en realidad 2^8. Finalmente TCKPS a i que es la variable del bucle que representa el preescalado y generamos el retardo con el PR calculado

Generación de retardos II

En este apartado simplemente crearemos un programa con el mismo objetivo que Generación de retardos I pero utilizando la función desarrollada en el apartado anterior.

```
/*
 * File:   main4.c
 * Author: Jorge & Fernando
 *
 * Created on February 5, 2020
 */

#include <xc.h>
#include "Pic32Ini.h"
#include "retardo.h"
#define PIN_LED 0

int main(void) {

    InicializarReloj();

    // Pines del LED como digital
    ANSELC &= ~(1 << PIN_LED);
    LATA = 0; // En el arranque dejamos todas las salidas a 0
    LATB = 0;
    LATC = 0xF; //apagados al empezar

     // Como salidas
    TRISA = 0;
    TRISB = 0;
    TRISC = 0;
    while(1){
        Retardo(500); //0.5s  para que la frecuencia de parpadeo = 1Hz
        LATC ^= 1 << PIN_LED; // Se invierte el LED
    }
```

El programa es igual que el apartado Generación de retardos, pero empleando la función retardo con 500ms para que parpadee con la frecuencia de 1Hz requerida

LED latiendo

Por último, creamos un programa que permita el parpadeo con un periodo de 50Hz pero con unas características especiales: en la primera iteración el LED permanece apagado todo el tiempo, en la segunda está encendido durante 1ms y apagado durante el resto del tiempo. El tiempo de encendido se va incrementando en tramos de 1ms hasta llegar a 20ms, cuando comienza a descender. La iteración completa se repite indefinidamente.

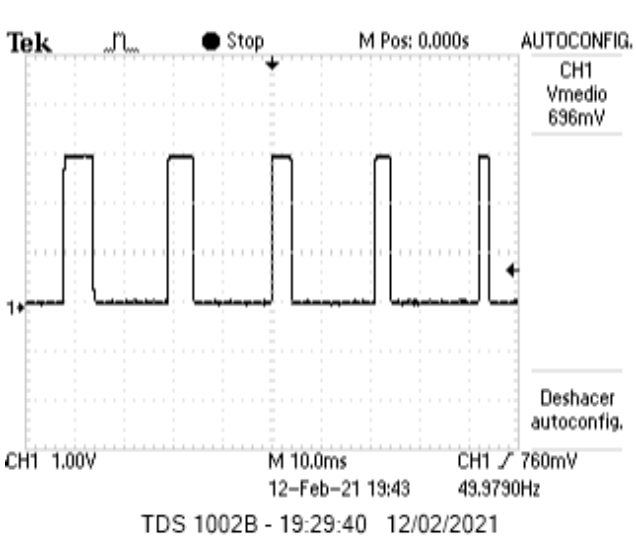
```
/*
 * File:   main5.c
 * Author: Jorge & Fernando
 *
 * Created on February 5, 2020
 */

#include <xc.h>
#include "Pic32Ini.h"
#include "retardo.h"
#define PIN_LED 0

int main(void) {

    InicializarReloj();

    // Pin del LED como digital
    ANSELC &= ~(1 << PIN_LED);

    // Establecemos todos los pines como salidas
    TRISA = 0;
    TRISB = 0;
    TRISC = 0;

    // Salidas a 0
    LATA = 0;
    LATB = 0;
    LATC = 0xF; //apagados


    int i = 0;
    int cambio = 1;

    while(1){

        LATC ^= 1 << PIN_LED; // ENCENDEMOS
        Retardo(i);
        LATC ^= 1 << PIN_LED; //APAGAMOS
        Retardo(20-i);      //esperamos 20ms-el ciclo que llevemos

        i += cambio;    //sumamos 1 en la ida, -1 en la vuelta

        if (i == 20 || i == 0)  //al final de los 20 ciclos para el cambio
            cambio *= -1;       //al finalizar la ida, 1*-1= -1 para la vuelta
                                //al finalizar la vuelta -1*-1=1 para la ida
    }
}
```

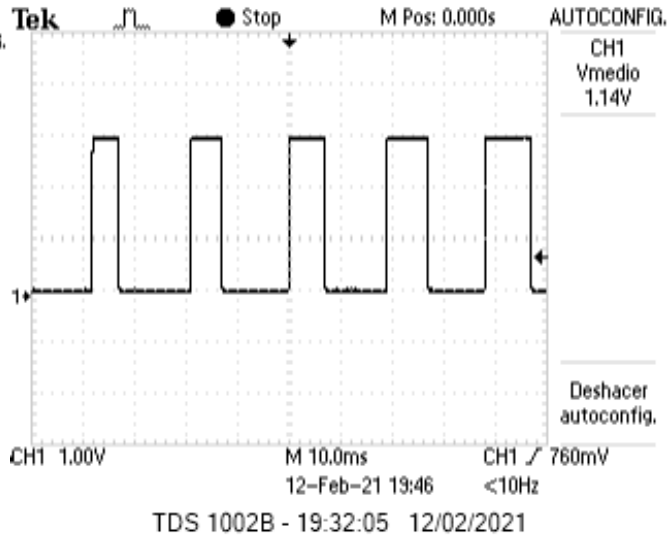
Podemos volver a utilizar la función retardo. En la primera iteración comienza apagado más tiempo y cada vez el tiempo de encendido es mayor. Cada iteración dura 20ms. Empezando 20:off 0:on, 19:off 1:on, 18:off:2on …

Por tanto, vemos utilizaremos i y 20-i como tiempos de espera. Al finalizar el primer ciclo completo comienza la vuelta en el led empieza 20ms encendido y cada vez esta menos tiempo encendido y más apagado.

Los cambios al final del ciclo se han realizado con un if cuando i==20 o i==0 donde pasamos de sumar a restar 1 a i para comenzar la vuelta y volvemos a sumar cuando finaliza la vuelta y queremos volver a comenzar la ida.

A continuación, observamos los dos ciclos completos que se van a ir repitiendo.





En la primera imagen comienza más tiempo apagado y cada vez está menos tiempo apagado y más encendido (ida)

En la segunda imagen va aumentando el tiempo apagado y disminuyendo el tiempo encendido (vuelta)

Conclusión

En esta práctica hemos aprendido a:

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