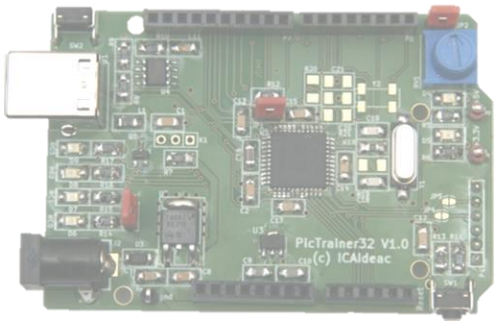
Microprocesadores

Práctica 5: Interrupciones

Francisco Martín Martínez



Jorge Calvar, Fernando Santana
1-3-2021

Contenido

Introducción

En esta práctica nos familiarizaremos con las interrupciones del microcontrolador.

2. Temporizador con interrupciones

Partiendo del programa que enciende el LED RC0 cuando se pulsa el pulsador RB5 realizado en la práctica 2, vamos a modificarlo para que el LED RC3 se encienda y se apague con un periodo de 2 s, en paralelo con el funcionamiento del pulsador y el LED RC0. Para esto vamos a utilizar las interrupciones que nos permiten realizar varias tareas a la vez de forma fácil. Las interrupciones son sucesos inesperados que interrumpen el flujo normal de ejecución del programa y se producen cuando un periférico externo necesita la atención de la CPU.

Vamos a configurar el temporizador 2 para que genere interrupciones cada segundo de prioridad 2 y subprioridad 0.

El prototipo de la rutina de atención a la interrupción que se encarga de encender y apagar el LED es el siguiente:

```
void __attribute__ (( vector ( _TIMER_2_VECTOR ) , interrupt ( IPL2SOFT ) , nomips16 ) )
InterrupcionTimer2 ( void )
```

```
/*
 * File:   main2.c
 * Author: Fernando  & Jorge
 *
 * Created on 05 March 2021, 14:20
 */
#include <xc.h>
#include <stdint.h>
#include "Pic32Ini.h"

#define PIN_PULSADOR 5
#define PIN_LED0 0
#define PIN_LED3 3


void InicializarTimer2(void) {

     // Se configura el timer para que termine la cuenta en 1 ms
     T2CON = 0;
     TMR2 = 0;
     PR2 = 19530; // (1*5E6/256)-1

     IPC2bits.T2IP = 2; // Prioridad 2
     IPC2bits.T2IS = 0; // Subprioridad 0
     IFS0bits.T2IF = 0;   // Se borra el flag de interrupción por si estaba
pendiente
     IEC0bits.T2IE = 1;  // Por último, se habilita su interrupción

     T2CON = 0x8070; // Se cambia el estado del led 3
}

int main(void) {

    int pulsador;

    ANSELC &= ~(1 << PIN_LED0);
    ANSELC &= ~(1 << PIN_LED3 | 1<<PIN_LED0);
```

```
    LATA = 0; // En el arranque dejamos todas las salidas a 0
    LATB = 0;
    LATC = 0xF; // Apago los LEDs de la placa

    TRISA = 0;
    TRISC = 0; //Todo output
    TRISB = 1 << PIN_PULSADOR; //PIN PULSADOR como entrada, resto salidas

    InicializarReloj();

    InicializarTimer2();

    // Una vez inicializados los periféricos, activamos las interrupciones
    INTCONbits.MVEC = 1; // Modo multivector
    asm(" ei"); // Interrupciones habilitadas

     while (1) {

         //Se lee estado del pulsador
         pulsador = (PORTB >> PIN_PULSADOR) & 1;

         asm(" di");
         if (pulsador == 0) {
             LATC &= ~(1<<PIN_LED0);
         } else {
             LATC |= 1<<PIN_LED0;
         }
         asm(" ei");
     }
    }

__attribute__((vector(_TIMER_2_VECTOR), interrupt(IPL2SOFT), nomips16))
void InterrupcionTimer2(void) {
    // Se borra el flag de interrupción para no volver a
    // entrar en esta rutina hasta que se produzca una nueva
    // interrupciÃ³n.
    IFS0bits.T2IF = 0;
    LATCINV = 1<<PIN_LED3;
}
```

Gracias a la utilización de interrupciones se facilita el uso de los temporizadores y la multitarea: pudiendo realizar múltiples procesos a la vez de manera sencilla como veremos en el ejercicio 3. Es importante desactivar las interrupciones cuando modificamos el mismo registro desde el *main* y la rutina interrupción. Otra forma de evitar problemas cuando hay datos compartidos sería utilizar las instrucciones SET, CLR, INV que se realizan en una sola instrucción.

2.1. Ejercicio

Como LATC es una variable compartida entre el programa principal y la interrupción si no desactivamos las interrupciones para la instrucción observamos como el parpadeo ya no es regular y hay veces que tarda más de lo debido en apagarse o encenderse. Esto se debe a que la interrupción ha saltado cuando LATC ya había sido cargado en un registro en el programa principal. Por tanto, ese valor cargado en el registro será el utilizado en el programa principal para guardar LATC y los cambios realizados por la interrupción son ignorados, por lo que el led no se apaga o no se enciende.

3. Interrupciones múltiples

En este apartado se van a configurar dos interrupciones a la vez para que el LED RC2 parpadee con un periodo de 1s, manteniendo el LED RC3 parpadeando con un periodo de 2s. Configuramos el temporizador 3 para que genere interrupciones cada 500 ms de prioridad 4, subprioridad 0.

Al funcionar dos interrupciones a la vez podrían aparecer problemas de incoherencia de datos en el manejo del LATC por lo que utilizamos instrucciones como LATINV que se realizan en una sola instrucción y además desactivamos las interrupciones cuando usamos LATC en el *main*.

```
/*
 * File:   main3.c
 * Author: Fernando  & Jorge
 *
 * Created on 05 March 2021, 14:25
 */
#include <xc.h>
#include <stdint.h>
#include "Pic32Ini.h"

#define PIN_PULSADOR 5
#define PIN_LED0 0
#define PIN_LED2 2
#define PIN_LED3 3

void InicializarTimer2(void) {

    // Se configura el timer para que termine la cuenta en 1 s
    T2CON = 0;
    TMR2 = 0;
    PR2 = 19530; // (1*5E6/256)-1

    IPC2bits.T2IP = 2;  // Prioridad 2
    IPC2bits.T2IS = 0;  // Subprioridad 0
    IFS0bits.T2IF = 0;  // Se borra el flag de interrupción por si estaba
pendiente
    IEC0bits.T2IE = 1;  // y por último se habilita su interrupción

    T2CON = 0x8070; // Se arranca el timer con prescalado 7=256
}

void InicializarTimer3(void) {

    // Se configura el timer para que termine la cuenta en 0.5s
    T3CON = 0;
    TMR3 = 0;
    PR3 = 39061;         // (1*5E6/64)-1=39061.5

    IPC3bits.T3IP = 4;  // Prioridad 4
    IPC3bits.T3IS = 0;  // Subprioridad 0
    IFS0bits.T3IF = 0;  // Se borra el flag de interrupción por si estaba
pendiente
    IEC0bits.T3IE = 1;  // y por último se habilita su interrupción

    T3CON = 0x8060;      // Se arranca el timer con prescalado 7=256
```

```
}

int main(void) {

    // Puertos digitales
    ANSELC &= ~(1 << PIN_LED0 | 1<<PIN_LED2 | 1<<PIN_LED3);

    LATA = 0; // En el arranque dejamos todas las salidas a 0
    LATB = 0;
    LATC = 0xF; // Apago los LEDs de la placa

    TRISA = 0; // Output.
    TRISC = 0;
    TRISB = 1<<PIN_PULSADOR; //PIN PULSADOR como entrada, resto salidas

    // Inicializamos los timer
    InicializarTimer2();
    InicializarTimer3();

    // Activamos las iterrupciones
    INTCONbits.MVEC = 1; // Modo multivector
    asm(" ei"); // Interrupciones habilitadas

    int pulsador = (PORTB >> PIN_PULSADOR) & 1;

    while (1) {
        //Se lee estado del pulsador
        asm("   di");
        pulsador = (PORTB >> PIN_PULSADOR) & 1;
        if (pulsador == 0)
            LATC &= ~(1<<PIN_LED0);
        else
            LATC |= 1<<PIN_LED0;
        asm("   ei");
    }
}
__attribute__((vector(_TIMER_3_VECTOR), interrupt(IPL4SOFT), nomips16))
void InterrupcionTimer3(void) {
    // Se borra el flag de interrupción para no volver a
    // entrar en esta rutina hasta que se produzca una nueva
    // interrupción.
    IFS0bits.T3IF = 0;
    LATCINV = 1<<PIN_LED2;
}
__attribute__((vector(_TIMER_2_VECTOR), interrupt(IPL2SOFT), nomips16))
void InterrupcionTimer2(void) {
    // Se borra el flag de interrupción para no volver a
    // entrar en esta rutina hasta que se produzca una nueva
    // interrupción.
    IFS0bits.T2IF = 0;
    LATCINV = 1<<PIN_LED3;
}
```

Como ya hemos comentado, las interrupciones facilitan realizar dos tareas simultáneas ya que solo hay que programar dos interrupciones (una para cada temporizador) de manera independiente.

4. Juego de velocidad

Finalmente vamos a realizar un programa que permita averiguar lo rápido que es un jugador pulsando repetidamente un pulsador. Partiendo del programa realizado en la sección 2, se creará una variable global compartida entre el programa principal y la interrupción. La interrupción la pondrá a cero cada vez que se ejecute y el programa principal la incrementará cada vez que se pulse el pulsador (necesitará detectar el flanco).

Además, si se detecta que la variable supera cinco pulsaciones, se encenderá el LED RC1, que permanecerá encendido hasta que se dé un reset al microcontrolador o transcurridos 4 segundos para volver a empezar el juego. El número de pulsaciones tras el cual se enciende RC1 se ha configurado usando la sentencia *define*, por lo que es posible modificarlo de manera sencilla para cambiar la velocidad del juego.

Main

```
/*
 * File:   main4.c
 * Author: Fernando  & Jorge
 *
 * Created on 05 March 2021, 14:34
 */
#include <xc.h>
#include "Pic32Ini.h"
#include "temporizador.h"

#define PIN_LED0 0
#define PIN_LED1 1
#define PIN_LED3 3
#define PIN_PULSADOR 5
#define PULSACIONES 5

int main(void) {

    // Definimos los pins como digitales
    ANSELC &= ~(1<<PIN_LED0 | 1<<PIN_LED1 | 1<<PIN_LED3);
    ANSELB &= ~(1<<PIN_PULSADOR);

    // Definimos las salidas a 0 excepto el LED
    LATA = 0;
    LATB = 0;
    LATC = 0xF;

    // Definimos los puertos como salida excepto el pulsador
    TRISA = 0;
    TRISB = 1<<PIN_PULSADOR;
    TRISC = 0;

    InicializarReloj();

    // Inicializamos el reloj
    InicializarTimer2();

    // Iniciamos las interrupciones
    INTCONbits.MVEC = 1;
    asm("   ei");

    // Bucle
```

```
    int puls_act, puls_ant;
    puls_ant = (PORTB>>PIN_PULSADOR) & 1;
    while(1) {
        puls_act = (PORTB>>PIN_PULSADOR) & 1;

        //  Configuramos RC0 según el pulsador
        asm("   di");
        if(puls_act == 0)
            LATC &= ~(1<<PIN_LED0);
        else
            LATC |= 1<<PIN_LED0;
        asm("   ei");

        // Si hay flanco incrementamos la variable global
        if(puls_act>puls_ant) {
            setPulsaciones(getPulsaciones()+1);
        }

        // Si se llega al límite

        if(getPulsaciones() > PULSACIONES) {
            LATC &= ~(1<<PIN_LED1);
            setTicks(4);
        }
        puls_ant = puls_act;
    }
    return 0;
}
```

Se observa como funciona adecuadamente y al presionar el pulsador lo suficientemente rápido se acaba encendiendo el LED. También funciona correctamente el reinicio automático a los 4 segundos.

Driver

En este caso hemos separado en un driver el temporizador que gestiona las interrupciones. Para programas que aumentan en tamaño se recomienda separarlos en módulos lo más aislado posible. Primero escribimos la cabecera del módulo temporizador incluyendo los prototipos de sus funciones.

```
/*
 * File:   temporizador.h
 * Author: Fernando  & Jorge
 *
 * Created on 5 de marzo de 2021, 18:57
 */

#ifndef TEMPORIZADOR_H
#define TEMPORIZADOR_H

int getPulsaciones();
int getTicks();
void setPulsaciones(int _pulsaciones);
void setTicks(int _ticks);

void InicializarTimer2(void);

#endif  /* TEMPORIZADOR_H */
```

Para que la variable *ticks* no sea accesible a todo el programa y no se modifique por error, se crea una función *getter* llamada *getTicks()* que nos devuelve su valor. A continuación, se muestra el código de *temporizador.c*.

```
#include "xc.h"
#include "temporizador.h"

#define PIN_LED0 0
#define PIN_LED1 1
#define PIN_LED3 3
#define PIN_PULSADOR 5
#define PULSACIONES 5

static int pulsaciones = 0;
static int ticks = 0;

int getPulsaciones() { return pulsaciones; }
int getTicks() { return ticks; }
void setPulsaciones(int _pulsaciones) { pulsaciones = _pulsaciones; }
void setTicks(int _ticks) { ticks = _ticks; }

void InicializarTimer2(void) {

    T2CON = 0;
    TMR2 = 0;
    PR2 = 19530;         //1*5E6/256

    IPC2bits.T2IP = 2;   //Priority = 2
    IPC2bits.T2IS = 0;   //Subpriority = 2
    IFS0bits.T2IF = 0;   //Flag a cero
    IEC0bits.T2IE = 1;   //Enable interrupt

    T2CON = 0x8070;      //Inicializamos con prescaler = 256

}

__attribute__((vector(_TIMER_2_VECTOR), interrupt(IPL2SOFT),  nomips16))
void InterrupcionTimer2(void) {
    IFS0bits.T2IF = 0;  //Eliminamos el flag
    pulsaciones = 0;
    LATCINV = 1<<PIN_LED3;
    if (ticks!=0)
        ticks--;
    else
        LATCSET = 1<<PIN_LED1;
}
```

Al definir la variable *ticks* fuera de las funciones se trata de una variable global, pero la palabra *static* consigue que no sea accesible como global desde funciones fuera del módulo temporizador, así no se modificará por error en el *main* u otros módulos. Para acceder a ella usaremos *setTicks()* y *getTicks()*.

Conclusión

En esta práctica hemos aprendido a:

- Configurar los periféricos para generar interrupciones al microcontrolador.
- Escribir rutinas de tratamiento de interrupciones.
- Saber trabajar con variables compartidas entre interrupciones y programa principal evitando problemas de incoherencia de datos.