Práctica 5 – Interrupciones

Jaime Arana
Manuel Ferrero
3ºA

Temporizador con interrupciones

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

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

// se encienda y apague 2 seg = 1 seg apagado + 1 seg encendido
#define T_PARPADEO 1000 // en ms

/* CÁLCULO DE PR2 */
// PR2 = (retardo / (divisor * 200 ns)) - 1 <= 65535
// PR2 = (1 ms / (1 * 200 ns)) - 1 = 4999
#define PERIODO_TIMER2 4999 // valor de PR2
#define PRIORIDAD_TIMER2 2
#define SUBPRIORIDAD_TIMER2 1
#define PRESCALER 0

int main(void) {
    int pulsador, puerto;
    uint32_t ticks_ant, ticks_act;

    // configurar entradas y salidas
    TRISB |= (1 << PIN_PULSADOR);
    TRISC &= ~(1 << PIN_LED0);
    TRISC &= ~(1 << PIN_LED3);

    // LED 0 y LED 3 empiezan apagados
    puerto = 0x0000;
    puerto |= (1 << PIN_LED0);
    puerto |= (1 << PIN_LED3);
    LATC |= puerto;

```

```
36      // llamamos a la función para inicializar Timer2
37      InicializarTimer2(PERIODO_TIMER2, PRIORIDAD_TIMER2, SUBPRIORIDAD_TIMER2, PRESCALER);
38
39      // activamos las interrupciones
40      INTCONbits.MVEC = 1;
41      asm(" ei");
42
43      ticks_ant = TicksDesdeArr();
44
45      while(1) {
46          ticks_act = TicksDesdeArr();
47          if((ticks_act - ticks_ant) > T_PARPADEO) {  // con la resta medimos el tiempo transcurrido
48              ticks_ant = ticks_act;
49              LATCINV = (1 << PIN_LED3); // invertimos el valor de LED
50          }
51
52          pulsador = (PORTB >> PIN_PULSADOR) & 1;
53
54          if(pulsador == 0) { // está pulsado
55              LATCCLR = (1 << PIN_LED0);  // LED = 0 --> encendido
56          } else {
57              LATCSET = (1 << PIN_LED0);  // LED = 1 --> apagado
58          }
59      }
60  }
```

Interrupción Timer2

```
1   #include <xc.h>
2   #include <stdio.h>
3   #include "Pic32Ini.h"
4
5   #define BITS_PRESCALER 4
6
7   void InicializarTimer2(int tiempo, int prioridad, int subprioridad, int valor_prescaler) {
8       T2CON = 0;  // se para el temporizador
9       TMR2 = 0;   //se pone la cuenta a 0
10      IFS0bits.T2IF = 0;  // borrar bit fin de cuenta (flag)
11      PR2 = tiempo;
12      IEC0bits.T2IE = 1;  // enable = 1
13      IPC2bits.T2IP = prioridad;  // prioridad = 2
14      IPC2bits.T2IS = subprioridad;  // subprioridad = 1
15
16      T2CON = 0x8000; // Timer2 --> ON = 1 + reloj interno seleccionado
17      T2CON |= (valor_prescaler << BITS_PRESCALER);   // añadimos el prescaler correspondiente
18  }
19
20  // variable contador que se irá iterando
21  static uint32_t ticks = 0;
22
```

```
23  __attribute__((vector(_TIMER_2_VECTOR), interrupt(IPL2SOFT), nomimps16))
24  void InterrupcionTimer2(void) {
25      // borrar flag para que no vuelve a entrar
26      // solo entrar cuando se produzca interrupción
27      IFS0bits.T2IF = 0;
28
29      // rutina de atención a la interrupcion
30      ticks++;
31  }
32
33  uint32_t TicksDesdeArr(void) {
34      uint32_t temp;  // variable para hacer copia atomica
35
36      asm(" di");  // inhabilitar interrupción
37      temp = ticks;
38      asm(" ei");  // habilitar
39
40      return temp;
41  }
```

Ejercicio

En este apartado se pide modificar el programa anterior y no utilizar las instrucciones LATCCLR, LATCSET y LATCINV. Estas instrucciones consiguen hacer modificar el registro en una instrucción de escritura, haciendo que esta sea atómica. Por lo tanto, se protege la escritura del registro frente a las interrupciones. En nuestro caso no hemos detectado ningún problema ya que ha dado la casualidad de que la interrupción no ha interrumpido el proceso de escritura del registro.

Solo se muestra la sección de código donde se producen los cambios.

```
36      // llamamos a la función para inicializar Timer2
37      InicializarTimer2(PERIODO_TIMER2, PRIORIDAD_TIMER2, SUBPRIORIDAD_TIMER2, PRESCALER);
38
39      // activamos las interrupciones
40      INTCONbits.MVEC = 1;
41      asm(" ei");
42
43      ticks_ant = TicksDesdeArr();
44
45      while(1) {
46          ticks_act = TicksDesdeArr();
47          if((ticks_act - ticks_ant) > T_PARPADEO) {  // con la resta medimos el tiempo transcurrido
48              ticks_ant = ticks_act;
49              LATC ^= (1 << PIN_LED3); // invertimos el LED 0
50          }
51
52          pulsador = (PORTB >> PIN_PULSADOR) & 1;
53
54          if(pulsador == 0) { // está pulsado
55              LATC &= ~(1 << PIN_LED0); // ponemos el LED = 0 --> endencendido
56          } else {
57              LATC |= (1 << PIN_LED0); // ponemos el LED = 1 --> apagado
58          }
59      }
60  }
```

Interrupciones múltiples

```c
#include <xc.h>
#include <stdint.h>
#include "TemporizadorTimer2.h"    // Timer2
#include "TemporizadorTimer3.h"  // Timer3
#include "Pic32Ini.h"

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

#define T_PARPADEO_LED3 1000 // en ms
#define T_PARPADEO_LED2 500 // en ms

/* CÁLCULO DE PR2 */
// PR2 = (retardo / (divisor * 200 ns)) - 1 <= 65535
// PR2 = (1 ms / (1 * 200 ns)) - 1 = 4999
#define PERIODO_TIMER2 4999 // valor de PR2
#define PRIORIDAD_TIMER2 2
#define SUBPRIORIDAD_TIMER2 1
#define PRESCALER_TIMER2 0

/* CÁLCULO DE PR3 */
// PR3 = (retardo / (divisor * 200 ns)) - 1 <= 65535
// PR3 = (1 ms / (1 * 200 ns)) - 1 = 4999
#define PERIODO_TIMER3 4999 // valor de PR3
#define PRIORIDAD_TIMER3 4
#define SUBPRIORIDAD_TIMER3 0
#define PRESCALER_TIMER3 0

int main(void) {
    int pulsador, puerto;
    uint32_t ticks_ant, ticks_act, ticks_ant2, ticks_act2;

    // configurar entradas y salidas
    TRISB |= (1 << PIN_PULSADOR);
    TRISC &= ~(1 << PIN_LED0);
    TRISC &= ~(1 << PIN_LED2);
    TRISC &= ~(1 << PIN_LED3);

    // LEDs empiezan apagados
    puerto = 0x0000;
    puerto |= (1 << PIN_LED0);
    puerto |= (1 << PIN_LED2);
    puerto |= (1 << PIN_LED3);
    LATC |= puerto;

    // llamamos a las funciones para inicializar los timers
    InicializarTimer2(PERIODO_TIMER2, PRIORIDAD_TIMER2, SUBPRIORIDAD_TIMER2, PRESCALER_TIMER2);
    InicializarTimer3(PERIODO_TIMER3, PRIORIDAD_TIMER3, SUBPRIORIDAD_TIMER3, PRESCALER_TIMER3);

    INTCONbits.MVEC = 1; // ponemos CPU en modo multvector
    asm(" ei"); // activamos interrupciones

    ticks_ant = TicksDesdeArrTimer2();
    ticks_ant2 = TicksDesdeArrTimer3();

```

```
58          while(1) {
59              // para el LED3 se utiliza el TIMER2
60              ticks_act = TicksDesdeArrTimer2();
61              if( (ticks_act - ticks_ant) > T_PARPADEO_LED3){ // con la resta medimos el tiempo transcurrido
62                    ticks_ant = ticks_act;
63                    LATCINV = (1 << PIN_LED3);
64              }
65
66              // para el LED2 se utiliza TIMER3
67              ticks_act2 = TicksDesdeArrTimer3();
68              if( (ticks_act2 - ticks_ant2) > T_PARPADEO_LED2){
69                    ticks_ant2 = ticks_act2;
70                    LATCINV = (1 << PIN_LED2);
71              }
72
73              pulsador = (PORTB >> PIN_PULSADOR) & 1;
74
75              if(pulsador == 0) { // está pulsado
76                  LATCCLR = (1 << PIN_LED0);  // LED = 0 --> encendido
77              } else {
78                  LATCSET = (1 << PIN_LED0);  // LED = 1 --> apagado
79              }
80          }
81      }
```

Interrupción Timer2

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

#define BITS_PRESCALER 4

void InicializarTimer2(int tiempo, int prioridad, int subprioridad, int valor_prescaler) {
    T2CON = 0;
    TMR2 = 0;
    IFS0bits.T2IF = 0;
    PR2 = tiempo;
    IEC0bits.T2IE = 1;  // enable = 1
    IPC2bits.T2IP = prioridad;  // prioridad = 2
    IPC2bits.T2IS = subprioridad;  // subprioridad = 1

    T2CON = 0x8000; // Timer2 --> ON = 1 + reloj interno seleccionado
    T2CON |= (valor_prescaler << BITS_PRESCALER);    // añadimos el prescaler correspondiente
}

// variable contador que se irá iterando
static uint32_t ticks = 0;

__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;

    // rutina de atención a la interrupcion
    ticks++;
}

uint32_t TicksDesdeArrTimer2(void) {
    uint32_t temp;  // variable para hacer copia atomica

    asm(" di");  // inhabilitar interrupción
    temp = ticks;
    asm(" ei");  // habilitar

    return temp;
}
```

Interrupción Timer3

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

#define BITS_PRESCALER 4

void InicializarTimer3(int tiempo, int prioridad, int subprioridad, int valor_prescaler) {
    T3CON = 0;
    TMR3 = 0;
    IFS0bits.T3IF = 0;
    PR3 = tiempo;
    IEC0bits.T3IE = 1;  // enable = 1
    IPC3bits.T3IP = prioridad;  // prioridad = 4
    IPC3bits.T3IS = subprioridad;  // subprioridad = 0

    T3CON = 0x8000; // Timer3 --> ON = 1 + reloj interno seleccionado
    T3CON |= (valor_prescaler << BITS_PRESCALER);   // añadimos el prescaler correspondiente

}

// variable contador que se irá iterando
static uint32_t ticks = 0;
__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;

    // rutina de atención a la interrupcion
    ticks++;
}

uint32_t TicksDesdeArrTimer3(void) {
    uint32_t temp;  // variable para hacer copia atomica

    asm(" di");  // inhabilitar interrupción
    temp = ticks;
    asm(" ei");  // habilitar

    return temp;
}
```

Juego de velocidad

Primero se adjunta el código sin la sección opcional del ejercicio.

```c
#include <xc.h>
#include <stdint.h>
#include "TemporizadorSeccion4Simple.h"
#include "Pic32Ini.h"

#define PIN_PULSADOR 5
#define PIN_LED 1

#define MAX_PULSACIONES 5

/* CÁLCULO DE PR2 */
// PR2 = (retardo / (divisor * 200 ns)) - 1 <= 65535
// PR2 = (1 seg / (256 * 200 ns)) - 1 = 19531
#define PERIODO_TIMER2 19531 // le damos al jugador 1 seg para pulsar x5 veces el pulsador
#define PRIORIDAD_TIMER2 2
#define SUBPRIORIDAD_TIMER2 1
#define PRESCALER 256

int main(void) {
    int puls_ant, puls_act;

    // configurar entradas y salidas
    TRISB |= (1 << PIN_PULSADOR);
    TRISC &= ~(1 << PIN_LED);

    LATCSET = (1 << PIN_LED);  // poner a 1 --> apagado

    // llamamos a la función para inicializar Timer2
    InicializarTimer2(PERIODO_TIMER2, PRIORIDAD_TIMER2, SUBPRIORIDAD_TIMER2, PRESCALER);

    INTCONbits.MVEC = 1; // ponemos la CPU en modo multivector
    asm(" ei"); // activamos las interrupciones

    puls_ant = (PORTB >> PIN_PULSADOR) & 1;

    while(1) {
        puls_act = (PORTB >> PIN_PULSADOR) & 1;

        if((puls_ant != puls_act) && (puls_act == 0)) { // detectar flanco bajada pulsador
            asm( "di");  // desactivamos interupciones
            num_pulsaciones++;
            asm(" ei");  // activamos interrupciones

            if(num_pulsaciones >= MAX_PULSACIONES) {
                LATCCLR = (1 << PIN_LED);  // poner a 0 --> encendido
            }
        }
        puls_ant = puls_act;
    }
}
```

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

#define BITS_PRESCALER 4

void InicializarTimer2(int tiempo, int prioridad, int subprioridad, int valor_prescaler) {
    T2CON = 0;  // se para el temporizador
    TMR2 = 0;   //se pone la cuenta a 0
    IFS0bits.T2IF = 0;  // borrar bit fin de cuenta (flag)
    PR2 = tiempo;
    IEC0bits.T2IE = 1;  // enable = 1
    IPC2bits.T2IP = prioridad;  // prioridad = 2
    IPC2bits.T2IS = subprioridad;  // subprioridad = 1

    T2CON = 0x8000; // Timer2 --> ON = 1 + reloj interno seleccionado
    T2CON |= (valor_prescaler << BITS_PRESCALER);   // añadimos el prescaler correspondiente
}

// variable global compartida que cuenta las pulsaciones
static uint16_t num_pulsaciones = 0;

__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;

    // rutina de atención interrupción
    // se ponen a 0 las pulsaciones al saltar la interrupción
    num_pulsaciones = 0;
}
```

Código con la opción de esperar 4 segundos y apagar el LED

```c
#include <xc.h>
#include <stdint.h>
#include "TemporizadorSeccion4.h"
#include "Pic32Ini.h"

#define PIN_PULSADOR 5
#define PIN_LED 1

#define MAX_PULSACIONES 5

/* CÁLCULO DE PR2 */
// PR2 = (retardo / (divisor * 200 ns)) - 1 <= 65535
// PR2 = (1 seg / (256 * 200 ns)) - 1 = 19531
#define PERIODO_TIMER2 19531 // le damos al jugador 1 seg para pulsar x5 veces el pulsador
#define PRIORIDAD_TIMER2 2
#define SUBPRIORIDAD_TIMER2 1
#define PRESCALER 256

#define RETARDO 4

extern uint16_t fin_partida;

int main(void) {
    int puls_ant, puls_act;

    // configurar entradas y salidas
    TRISB |= (1 << PIN_PULSADOR);
    TRISC &= ~(1 << PIN_LED);

    LATCSET = (1 << PIN_LED);  // poner a 1 --> apagado

    // llamamos a la función para inicializar Timer2
    InicializarTimer2(PERIODO_TIMER2, PRIORIDAD_TIMER2, SUBPRIORIDAD_TIMER2, PRESCALER);

    INTCONbits.MVEC = 1; // ponemos la CPU en modo multivector
    asm(" ei"); // activamos las interrupciones

    puls_ant = (PORTB >> PIN_PULSADOR) & 1;

    while(1) {
        puls_act = (PORTB >> PIN_PULSADOR) & 1;

        if((puls_ant != puls_act) && (puls_act == 0)) { // detectar flanco bajada pulsador
            asm( "di");  // desactivamos interupciones
            num_pulsaciones++;
            asm(" ei");  // activamos interrupciones

            if(num_pulsaciones >= MAX_PULSACIONES) {
                fin_partida = 1;
                LATCCLR = (1 << PIN_LED);  // poner a 0 --> encendido
            }
        }
        puls_ant = puls_act;
    }
}
```

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

#define BITS_PRESCALER 4
#define PIN_LED 1

void InicializarTimer2(int tiempo, int prioridad, int subprioridad, int valor_prescaler) {
    T2CON = 0;  // se para el temporizador
    TMR2 = 0;   //se pone la cuenta a 0
    IFS0bits.T2IF = 0;  // borrar bit fin de cuenta (flag)
    PR2 = tiempo;
    IEC0bits.T2IE = 1;  // enable = 1
    IPC2bits.T2IP = prioridad;  // prioridad = 2
    IPC2bits.T2IS = subprioridad;  // subprioridad = 1

    T2CON = 0x8000; // Timer2 --> ON = 1 + reloj interno seleccionado
    T2CON |= (valor_prescaler << BITS_PRESCALER);   // añadimos el prescaler correspondiente
}

// variable global compartida que cuenta las pulsaciones
static uint16_t num_pulsaciones = 0;
uint16_t fin_partida = 0;

__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.
    static int contador = 0;
    IFS0bits.T2IF = 0;

    // rutina de atención interrupción
    // se ponen a 0 las pulsaciones al saltar la interrupción
    if (fin_partida == 0) {
        num_pulsaciones = 0;
    } else {
        contador++;
        if (contador == 4) {
            fin_partida = 0;
            LATCSET = (1 << PIN_LED);
        }
    }
}
```