Práctica 4. Interrupciones

(Duración: 1 sesión)

Introducción

En esta práctica nos familiarizaremos con las interrupciones del microcontrolador.

Objetivos

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

- 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.

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 solicitados en las secciones 2 y 3.

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_XX`, creada en la práctica 1.

Al igual que en la práctica 3, es necesario configurar la tarjeta para que funcione con el oscilador de cuarzo externo. Como recordará basta con incluir en el proyecto el archivo `Pic32Ini.c` disponible en Moodle.

Entregue capturas de osciloscopio en todos los apartados. Todos los programas › DEA › Osciloscopio › OpenChoice Desktop › OpenChoice Desktop.

2. Temporizador con interrupciones

Una de las ventajas de las interrupciones es que es fácil realizar dos tareas a la vez. Partiremos del programa que **enciende el LED `RC0` cuando se pulsa el pulsador `RB5`** realizado en la práctica 2, que, por si no se lo sabe ya de memoria de tanto usarlo, se incluye a continuación para mayor comodidad:

```c
#include <xc.h>
#define PIN_PULSADOR 5

int main(void)
{
  int pulsador;

  TRISC = XX; // A completar por el alumno (trabajo previo Práctica 2)
  LATC  = XX; // A completar por el alumno (trabajo previo Práctica 2)
  TRISB = XX; // A completar por el alumno (trabajo previo Práctica 2)

  while(1){
   // Se lee el estado del pulsador
   pulsador = (PORTB>>PIN_PULSADOR) & 1;
   if(pulsador == 0){
     LATC &= ~1;
   }else{
     LATC |= 1;
   }
  }
}
```

Una vez compilado el programa para ver que funciona correctamente, se modificará 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 ello, ha de configurar el **temporizador 2** para que genere interrupciones **cada segundo**. La prioridad de las interrupciones la pondremos a 2 y la subprioridad a 0. Una vez configurado el temporizador, se escribirá la rutina de atención a la interrupción, la cual se encargará de encender y apagar el LED. El prototipo de dicha rutina será el siguiente:

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

Tenga en cuenta que LATC es una variable compartida entre el programa principal y la interrupción, por lo que tendrá que tomar las medidas necesarias para evitar problemas de incoherencia de datos.

2.1. Ejercicio

Elimine momentáneamente las instrucciones que haya incluido para evitar la incoherencia de datos y vea si el programa presenta algún funcionamiento erróneo. Indique en la memoria de la práctica lo que haya observado.

3. Interrupciones múltiples

Modifique el programa anterior para que **el LED RC2 parpadee con un periodo de 1 s**, manteniendo el LED RC3 parpadeando con un periodo de 2 s. Para ello se configurará el **temporizador 3** para que genere interrupciones **cada 500 ms** y se le dará prioridad 4 y subprioridad 0 a su interrupción.

Tenga en cuenta que ahora funcionan las dos interrupciones a la vez, por lo que pueden aparecer problemas de incoherencia de datos en el manejo del LATC.

4. Juego de velocidad

Para terminar la práctica se va a realizar un programa que permita averiguar lo rápido que es un jugador pulsando repetidamente un pulsador. Para ello: se partirá 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,[1] se encenderá el **LED `RC1`**. Dicho LED permanecerá encendido hasta que se dé un reset al microcontrolador.

4.1. Apagado a los 4 segundos

Apague el led transcurridos 4 segundos para volver a empezar el juego.

4.2. Intercambio de variables compartidas con drivers

El apartado anterior podría hacerse con todas las funciones de inicialización e interrupciones en el main. Sin embargo, eso le privaría de saber cómo se haría si estas funciones estuvieran en un archivo llamado timer2.c que requiere el manejo de flags propios.

En este caso, es necesario que la variable del número de pulsaciones pertenezca al driver timer2.c para poder reiniciarse cada vez que salte la interrupción. Sin embargo, el main tiene que ser capaz de modificar su valor cada vez que se pulse. Existen varias formas de hacer esto, todas ellas, definiendo funciones en timer2.c/.h y protegiendo la variable compartida. Proponga una solución que funcione.

[1] Para poder cambiar fácilmente el número de pulsaciones para ganar el juego, es recomendable usar un `#define` para ello.