Seminario de Sistemas Embebidos
Práctica 7

Jorge Luis Madrid Gómez

14 de noviembre de 2020

UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías

Índice

Objetivo General

El objetivo de esta práctica es analizar el funcionamiento de las interrupciones en el uso de un microcontrolador y los registros necesarios, así como la implementación de interrupciones de alta y baja prioridad en la ejecución de un programa.

Marco Teórico

Interrupciones

Las interrupciones son recursos o mecanismos de los microcontroladores para responder a eventos, permitiendo suspender temporalmente el programa principal, para ejecutar una subrutina de servicio de interrupción ISR (*Interrupt Service Routines* por sus siglas en inglés); una vez terminada dicha subrutina, se reanuda la ejecución del programa principal [1].
Una de las situaciones en las que se generan interrupciones es cuando se necesita atender un monitoreo de manera continua, por lo que la interrupción puede ser de forma periódica, es decir por medio de una señal digital conectada a un pin específico del microcontrolador se puede atender tareas determinadas como adquisición de datos, monitoreo de sensores, cálculos numéricos, envío de comandos al sistema conectado, etc.
De forma general podemos clasificar las interrupciones como de baja o alta prioridad [2]; **una interrupción de alta prioridad detiene la ejecución del programa principal y de las interrupciones de baja prioridad mientras que una interrupción de baja prioridad no puede pausar una interrupción de alta prioridad**. El microcontrolador PIC18F4550 tiene múltiples fuentes de interrupción y a cada una se le puede asignar un nivel de prioridad. Las fuentes de interrupción tienen 4 bits para controlar la operación.

- Bit de habilitación. Habilita la interrupción con la cual se trabajará.
- Bit de indicador (bandera o *flag*). Es la bandera que indica que el evento de interrupción ocurrió.
- Bit de prioridad. Selecciona si la interrupción es de alta o baja prioridad.

- Bit de selección de flanco. La interrupción externa es activada al detectar un flanco de bajada (0) o un flanco de subida (1).

Para la habilitación de estos bits es necesario el uso de registros. Para esta práctica se utilizan interrupciones externas en el PIC18F4550 por lo que solo son necesarios los registros para este tipo de interrupciones. Estos registros son:

- RCON
- INTCON
- INTCON2
- INTCON3

Dependiendo de que bit en el micro usemos para la interrupción se utilizan los bits de estos registros.
Una vez determinada la configuración de las interrupciones se realizan las funciones que determinan que es lo que se ejecutará al realizarse la interrupción definida. Para ello se utiliza la siguiente sintaxis.

```
void __interrupt(priority) function_name(void){
  operation_1;
  operation_2;
  ...
  operation_n;
  flag_clean;
}
```

En el apartado de `priority` se especifica la prioridad; puede ser `low_priority` para baja prioridad o `high_priority` para alta prioridad. Al final de la función debe de limpiarse la bandera para garantizar que pueda suceder de nuevo la interrupción.

Materiales y métodos

Para el desarrollo de la práctica se necesitan los siguientes recursos.

- MPLAB con compilador XC8
- Proteus versión 8.++

Ya que la práctica solo requiere simulación y programación, no se requieren materiales físicos, por lo que en párrafos posteriores se mostrarán los elementos de la simulación necesarios. El microcontrolador usado en el curso será el **PIC18F4550**.
Se desarrollará un programa que utilice 2 interrupciones, una de alta prioridad y otra de baja. El programa principal debe esta en un bucle para observar el comportamiento de las interrupciones en el programa.
Se utiliza un programa que muestra un conteo de 0 a 9 en un bucle, además de mostrar la cantidad de interrupciones de alta y baja prioridad realizadas durante la ejecución de la simulación. Todos estos datos se mostrarán en una pantalla LCD para analizar el comportamiento durante la ejecución.

Simulación Proteus

Dentro del entorno de simulación de Proteus se realiza la construcción del circuito para la práctica, donde necesitamos los siguientes elementos electrónicos disponibles en las librerías que nos ofrece.

- El microcontrolador (PIC18F4550)
- Reloj Externo de Cuarzo a 8MHz (CRYSTAL)
- 2 capacitores de 22 pF (CAP)
- 3 Resistencias de 1 kΩ (RES)
- 3 Botones (BUTTON)
- Potenciometro de 1kΩ (POT-HG)
- Pantalla LCD de 16x2 (LM016L)

La construcción del circuito se muestra en la figura 1.

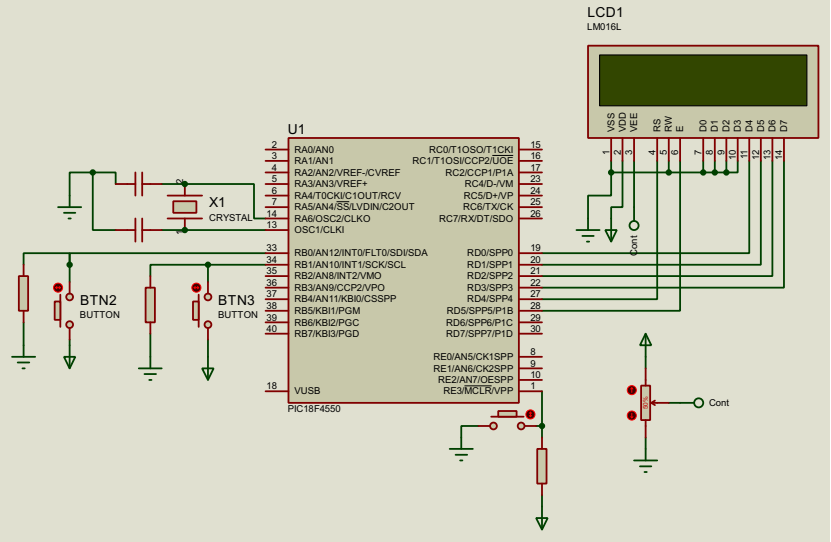


Figura 1: Simulación en Proteus

Se conecta la pantalla LCD en el puerto D del microcontrolador, como se ha hecho en prácticas anteriores. El bit B0 se utiliza para la interrupción de alta prioridad ya que el bit INT0 solamente puede ser de alta prioridad. El bit B1 será la interrupción externa INT1 de baja prioridad. Ambos se conectan a un botón con *pull down* ya que se configurarán con flancos de subida (1 lógico). Por último se conecta un reloj externo a 8 MHz y un botón de *reset*.

MPLAB

Dentro del entorno de programación de MPLAB se crea un proyecto configurado para nuestro microcontrolador y agregamos un `main.c` para programar el funcionamiento de la práctica.
Se agregan los fusibles habituales y se utilizan las librerías siguientes junto con el *header* `flex_lcd.h` para manejar la pantalla LCD.

```
#include <xc.h>
#include <stdio.h>
#define _XTAL_FREQ 8000000
#include "flex_lcd.h"
```

Se utiliza una cadena de caracteres para mostrar los datos en la pantalla LCD y 3 variables para contar el bucle de 0 a 9 y contar las interrupciones ocurridas.

```
//variables
unsigned char palabra[20];  //variable para palabras en LCD
unsigned int i, low=0, high=0; //contadores de interrupciones
```

En la función `main` configuramos el puerto D como de salida y los bits 0 y 1 del puerto B como entrada para detectar las interrupciones.

```
TRISD=0;  //Puerto D de salida (LCD)
LATD=0;
TRISBbits.TRISB0=1; //bit BO para High Interruption
TRISBbits.TRISB1=1; //bit B1 para Low Interruption
```

Se inicializa la pantalla LCD y seguido de eso se utilizan los registros `RCON` y `INTCON` para habilitar las interrupciones y los niveles de prioridad.

```
RCONbits.IPEN=1;  //Habilita niveles de prioridad
INTCONbits.GIE=1; //Habilita todas las interrupciones de alta
INTCONbits.GIEL=1;  //Habilita todas las interrupciones de baja
```

Ahora se configuran las interrupciones `INT0` con flanco de subida e `INT1` como de baja prioridad y con flanco de subida. La interrupción `INT0` es por defecto de alta prioridad.

```
//CONFIGURACION INT0
INTCONbits.INT0IE=1; //Habilita la interrupcion externa INT0
INTCON2bits.INTEDG0=1; //flanco de subida(1) flanco de bajada(0)
INTCONbits.INT0IF=0;     //Se limpia la bandera de INT0

//CONFIGURACION INT1
INTCON3bits.INT1IE=1; //Habilita la interrupcion externa INT1
INTCON2bits.INTEDG1=1; //flanco de subida(1) flanco de bajada(0)
INTCON3bits.INT1IP=0; //alta prioridad (1) baja prioridad (0)
INTCON3bits.INT1IF=0;    //Se limpia la bandera de INT1
```

Con esto hecho ya está listo el microcontrolador para detectar las interrupciones. Por último se realiza el programa principal que mostrará el conteo de 0 a 9 y el conteo de las interrupciones hechas.

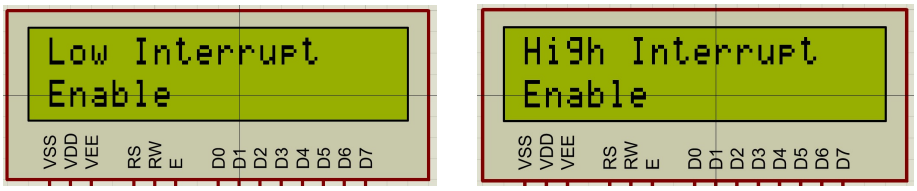
```
//Programa principal
while(1){
    for(i=0;i<9;i++){
        sprintf(palabra,"High: %d",high);
        Lcd_Out2(1,0,palabra);
        sprintf(palabra,"Low: %d Cont: %d",low,i);
        Lcd_Out2(2,0,palabra);
        __delay_ms(500);
        Lcd_Cmd(LCD_CLEAR);
    }
}
```

Ahora se realizan las funciones de las interrupciones para determinar que es lo que hará el microcontrolador la detectarlas. Se hacen 2 funciones, una para la alta prioridad (interrupción `INT0`) y otra para la baja prioridad (interrupción `INT1`). En ambos casos mostrará un mensaje en la pantalla LCD con su respectiva prioridad además de aumentar su respectivo contador en 1.

```
//Interrupcion de alta prioridad
void __interrupt() ISR_INT0(void){
    if(PORTBbits.RB0==1){
        Lcd_Cmd(LCD_CLEAR);
        Lcd_Out(1,0,"High Interrupt");
        Lcd_Out(2,0,"Enable");
        high=high+1;
    }while(PORTBbits.RB0==1);
    Lcd_Cmd(LCD_CLEAR);
    INTCONbits.INT0IF=0;
}
//Interrupcion de baja prioridad
void __interrupt(low_priority) ISR_INT1(void){
    if(PORTBbits.RB1==1){
        Lcd_Cmd(LCD_CLEAR);
        Lcd_Out(1,0,"Low Interrupt");
        Lcd_Out(2,0,"Enable");
        low=low+1;
    }while(PORTBbits.RB1==1);
    Lcd_Cmd(LCD_CLEAR);
    INTCON3bits.INT1IF=0;
}
```

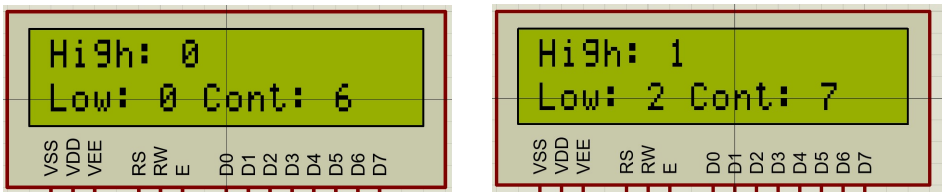
Resultados

A continuación se muestran los resultados al ejecutar las interrupciones presionando los botones correspondientes de cada una.



(a) Interrupción de baja prioridad (b) Interrupción de alta prioridad

Figura 2: Ejecución de la rutina de las interrupciones



(a) Main antes de las interrupciones (b) Main después de las interrupciones

Figura 3: Programa principal

Link del vídeo: https://cutt.ly/Dg9ZbGl

Conclusión

En la figura 3 se observa que antes de las interrupciones los contadores no tenían valor de 0 mientras que el bucle de 0 a 9 no se detiene. Al ejecutarse 2 interrupciones de baja prioridad y una de alta los contadores obtienen sus valores correspondientes y el contador del programa principal continua justo donde se quedó.
En la figura 2 se observa la rutina que se ejecuta al detectar la interrupción, mostrando un mensaje en la pantalla LCD de la prioridad correspondiente. Al terminar la interrupción se borra este mensaje y continua el programa principal.
En los resultados de las figuras no es apreciable pero al intentar ejecutar una interrupción de baja prioridad siendo ejecutada una interrupción de alta, la primera (baja prioridad) no muestra ningún mensaje en pantalla ya que no puede interrumpir la segunda (alta prioridad).
Las interrupciones en los microcontroladores son una gran herramienta para responder a eventos que pueden ser esperados en la ejecución de programa pero que no pueden ser detectados en un tiempo o periodo específico y el uso de condicionales en diversos puntos del programa pueden consumir demasiados recursos computacionales o ser ineficientes. Para obtener las ventajas de las interrupciones es necesario tener una buena comprensión de los registros a utilizar y los distintos tipos de interrupciones para saber en que conceptos pueden ser utilizadas y en cuales no.

Referencias

[1] E. M. Pérez, *Microcontroladores PIC: sistema integrado para el autoaprendizaje*. Marcombo, 2007.

[2] M. D. D. Sheet, "The pic18f4550 microcontroller," *Microchip Technology Inc., Arizona, USA*, 2009.