PRÁCTICA DE PICs - CONTADOR DE PULSOS CON BOTÓN

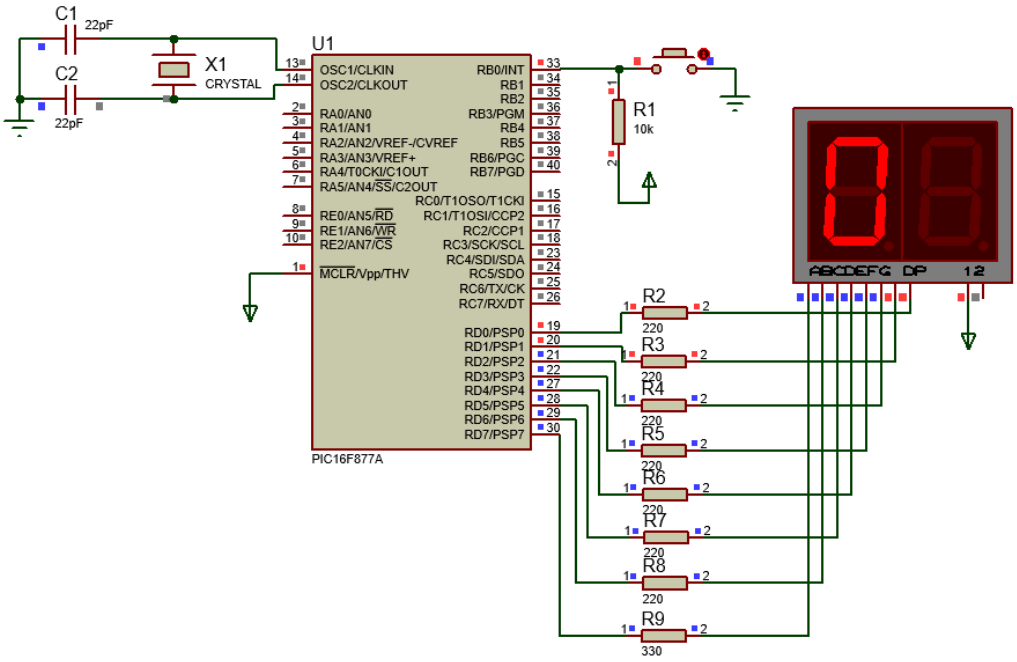
NIVEL: 1
DURACIÓN: 2 horas
OBJETIVO: Los estudiantes aprenderán a utilizar un botón para contar el número de pulsaciones y muestra el valor en un display de 7 segmentos.
MATERIALES

- PIC16F877A
- Botón pulsador
- Display de 7 segmentos (anodo o cátodo común)
- Resistencias para el botón y el display de 7 segmentos
- Cristal de reloj 4MHz
- Fuente de alimentación
- Cables y protoboard
- Proteus
- Mplab X IDE

1. DESARROLLO DE LA PRÁCTICA

1.1 SIMULACIÓN EN PROTEUS

1.1.1 MONTAJE



2.2 PROGRAMACIÓN EN MPLAB

```
#include <xc.h>

// Configuración del PIC
#pragma config FOSC = XT        // Oscilador XT (o configurarlo según tu cristal
#pragma config WDTE = OFF       // Watchdog Timer deshabilitado
#pragma config PWRTE = OFF      // Power-up Timer deshabilitado
#pragma config BOREN = ON       // Brown-out Reset activado
#pragma config LVP = OFF        // Low Voltage Programming deshabilitado
#pragma config CPD = OFF        // Código no protegido

#define _XTAL_FREQ 4000000      // Frecuencia del oscilador

unsigned int count = 0;

// Mapear los números al display de 7 segmentos (anodo común)
const char displayMap[] = {
    0b00000011,  // 0 (negado de 11111100)
    0b10011111,  // 1 (negado de 01100000)
    0b00100101,  // 2 (negado de 11011010)
    0b00001101,  // 3 (negado de 11110010)
    0b10011001,  // 4 (negado de 01100110)
    0b01001001,  // 5 (negado de 10110110)
    0b01000001,  // 6 (negado de 10111110)
    0b00011111,  // 7 (negado de 11100000)
    0b00000001,  // 8 (negado de 11111110)
    0b00001001   // 9 (negado de 11110110)
};
```

```
void display_number(unsigned int num) {
    PORTD = displayMap[num];
}

void __interrupt() ISR() {
    if (INTF) {  // Si hay interrupción externa
        count = (count + 1) % 10;  // Contador de 0 a 9
        INTF = 0;  // Limpiar bandera de interrupción
    }
}

void main(void) {
    // Configuración de puertos
    TRISB0 = 1;  // Pin RB0 como entrada (botón)
    TRISD = 0;   // Puerto D como salida (display de 7 segmentos)

    OPTION_REGbits.INTEDG = 1;  // Interrupción en el flanco ascendente
    INTE = 1;     // Habilitar interrupción externa
    GIE = 1;      // Habilitar interrupciones globales

    while (1) {
        display_number(count);  // Mostrar el valor en el display
    }
}
```

Figura 1. Source Files (main)

Explicación

- Array displayMap[]:
  - Este arreglo contiene el mapeo de bits para controlar el display de 7 segmentos con lógica negada (para cátodo común).
  - Cada valor en displayMap[] corresponde a un número del 0 al 9. Los bits están organizados para encender o apagar los segmentos del display, según el número que se desee mostrar.
- Función display_number():
  - Esta función toma el número que se quiere mostrar y lo convierte en el valor adecuado para encender los segmentos correspondientes del display. Luego, asigna ese valor a PORTD, que controla el display de 7 segmentos.
- Rutina de Interrupción ISR():
  - Esta función se ejecuta cada vez que se presiona el botón, gracias a la configuración de interrupciones. El botón está conectado a RB0, y al presionarlo, el número mostrado en el display incrementa en 1 (de 0 a 9).
  - Se utiliza una interrupción externa en RB0 para detectar la pulsación del botón y ejecutar la lógica sin estar constantemente revisando el estado del botón (esto mejora la eficiencia del código).
- Función Principal main():
  - La función principal configura los pines:
    TRISB0 = 1: Configura el pin RB0 como entrada (para el botón).
    TRISD = 0: Configura el puerto D como salida (para el display).
  - También habilita las interrupciones globales y la interrupción externa en RB0.
  - Luego, entra en un bucle infinito, donde continuamente se llama a la función display_number(count) para mostrar el valor actual del contador en el display de 7 segmentos.

2. ANÁLISIS DE RESULTADOS

- ¿Cómo incrementa el número en el display cuando se presiona el botón?

Cada vez que se presiona el botón conectado a RB0, el programa detecta la interrupción y la rutina de interrupción incrementa el valor del contador. Este valor se envía a la función display_number() para mostrar el nuevo número en el display de 7 segmentos.

- ¿Qué pasaría si no se utilizara una resistencia pull-up en el botón?

Si no se utiliza una resistencia pull-up, el pin RB0 podría estar flotante, lo que significa que podría captar ruido eléctrico y detectar pulsaciones falsas del botón, haciendo que el contador se incremente sin haber presionado el botón.

- ¿Por qué es necesario utilizar interrupciones en lugar de un bucle continuo para detectar el botón?

El uso de interrupciones permite que el microcontrolador realice otras tareas o se mantenga en un estado eficiente, sin tener que estar revisando continuamente el estado del botón. Esto mejora el rendimiento del sistema, ya que la interrupción solo ocurre cuando el botón es presionado, lo que ahorra recursos.

- ¿Qué pasaría si el display tuviera una lógica directa en lugar de lógica negada?

Si el display fuera de ánodo común (con lógica directa), los valores en displayMap[] deberían ser invertidos. En un display de ánodo común, un valor 0 enciende el segmento, mientras que en un display de cátodo común (con lógica negada), el valor 1 enciende el segmento.

- ¿Por qué el contador se reinicia a 0 después de llegar a 9?

El contador está configurado para incrementar desde 0 hasta 9, y luego volver a 0 mediante la operación (count + 1) % 10. La operación % es el operador de módulo que asegura que el valor del contador se mantenga dentro del rango de 0 a 9.

3. CONCLUSIONES

a. Al utilizar interrupciones externas, el sistema responde de manera eficiente a la pulsación del botón sin necesidad de revisar continuamente el estado de entrada. Esto hace que el sistema sea más eficiente y confiable.

b. Es fundamental entender la lógica del display (ánodo común o cátodo común) para mapear correctamente los bits que controlan los segmentos. En este caso, como el display tiene una lógica negada (cátodo común), fue necesario invertir los bits para que el número se muestre correctamente.

c. Este proyecto puede ampliarse para controlar más dígitos (utilizando técnicas de multiplexado) o agregar más funciones, como la posibilidad de restablecer el contador a través de otro botón o agregar un temporizador que controle el incremento automático.

d. El uso de resistencias pull-up asegura que el botón funcione correctamente, evitando lecturas incorrectas debido a fluctuaciones en el voltaje o ruido en la línea de entrada.

Ing. Andrés Corredor