PRÁCTICA DE PICs - ENCENDER UN LED

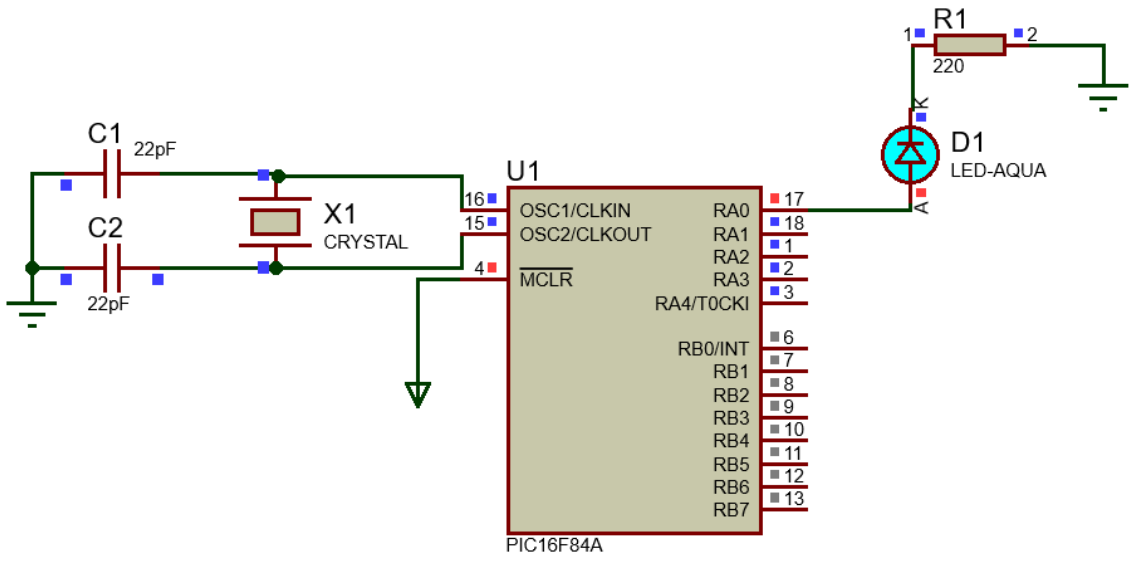
NIVEL: 1
DURACIÓN: 2 horas
OBJETIVO: Los estudiantes aprenderán a crear un sistema de riego automatizado.
MATERIALES

- PIC 16F84A
- 1 LED
- 1 resistencia 220
- 1 Oscilador de cristal (4MHz)
- 2 Capacitores cerámicos (22pF)
- 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

```c
/*
 * File:   newmain.c
 * Author: Andrés Corredor
 */
#include <xc.h>

#pragma config FOSC = XT // Configuración del oscilador (XT externo)
#pragma config WDTE = OFF // Desactivar el Watchdog Timer
#pragma config PWRTE = OFF// Desactivar el Power-up Timer
#pragma config CP = OFF // Desactivar la protección de código

#define _XTAL_FREQ 4000000// Definir la frecuencia del oscilador en 4 MHz

void main(void) {
    TRISA = 0x00; // Configurar el Puerto A como salida
    PORTA = 0x00;// Inicializar el Puerto A en 0 (todos lospinesapagados)

    while(1) { // Bucle infinito
        RA0 = 1; // Encender el LED en el pin RA0
        __delay_ms(500);// Retardo de 500 milisegundos
        RA0 = 0; // Apagar el LED en el pin RA0
        __delay_ms(500); // Retardo de 500 milisegundos
    }
}
```

Figura 1. Source Files (main)

Explicación

- La línea #include <xc.h> incluye las configuraciones y definiciones necesarias para programar el PIC con el compilador XC8.

- Las directivas #pragma config configuran algunos aspectos clave del microcontrolador:
- FOSC = XT: Configura el uso de un cristal externo como fuente de reloj del microcontrolador (en este caso, un cristal de 4 MHz).
- WDTE = OFF: Desactiva el Watchdog Timer para que el programa no se reinicie automáticamente.
- PWRTE = OFF: Desactiva el temporizador de encendido (Power-up Timer).
- CP = OFF: Desactiva la protección del código, lo que permite leer y escribir en la memoria de programa.
- La directiva #define _XTAL_FREQ 4000000 establece la frecuencia del oscilador en 4 MHz, lo que es esencial para que las funciones de retardo (__delay_ms()) funcionen correctamente.

- La instrucción TRISA = 0x00 configura el Puerto A como salida, lo que significa que los pines de RA0 a RA4 se usarán para enviar señales (encender o apagar el LED).

- PORTA = 0x00 asegura que todos los pines del Puerto A estén inicialmente en bajo (0V).
- El while(1) crea un bucle infinito que ejecuta continuamente las acciones dentro de las llaves {}.
- RA0 = 1: Pone en alto el pin RA0, lo que enciende el LED.
- __delay_ms(500): Introduce un retardo de 500 milisegundos (medio segundo).
- RA0 = 0: Pone en bajo el pin RA0, apagando el LED.
- Otro retardo de 500 milisegundos asegura que el LED permanezca apagado durante medio segundo antes de volver a encenderse.

2. ANÁLISIS DE RESULTADOS

- ¿Qué comportamiento tiene el LED durante la ejecución del programa?

El LED conectado al pin RA0 parpadea de forma intermitente, encendiéndose durante 500 milisegundos y apagándose durante otros 500 milisegundos, repitiendo este ciclo indefinidamente.

- ¿Cómo se garantiza que el LED parpadee indefinidamente?

El uso de un bucle infinito con while(1) garantiza que el código se ejecute sin detenerse. Esto es común en sistemas embebidos donde se requiere que el microcontrolador repita acciones de forma continua mientras esté alimentado.

- ¿Qué tan preciso es el temporizador?

La precisión del temporizador depende de la frecuencia del oscilador externo, en este caso, un cristal de 4 MHz, que proporciona una señal de reloj estable. El temporizador controlado por __delay_ms(500) se ajusta con esta frecuencia, lo que da lugar a retardos bastante precisos de 500 milisegundos.

- ¿Qué sucede si no se usa el cristal de cuarzo?

Si no se utiliza un cristal de cuarzo, la señal de reloj sería menos precisa o inexistente, dependiendo de la configuración del microcontrolador. El PIC16F84A no tiene un oscilador interno, por lo que es necesario un cristal para que el programa funcione correctamente y el LED parpadee con los tiempos correctos.

- ¿Es posible ajustar la velocidad de parpadeo del LED?

Sí, la velocidad de parpadeo del LED puede ajustarse modificando los valores en la función __delay_ms(). Por ejemplo, cambiar __delay_ms(500) a __delay_ms(1000) haría que el LED se mantuviera encendido y apagado durante 1 segundo, ralentizando el parpadeo.

- ¿Qué controla si un pin está en estado de salida o entrada?

El registro TRISA controla si los pines del Puerto A son entradas o salidas. En este código, TRISA = 0x00 configura todos los pines del Puerto A como salidas, lo que permite controlar el estado del LED conectando el pin RA0.

- ¿Qué importancia tiene el bucle infinito en sistemas embebidos?

El bucle infinito es esencial en sistemas embebidos porque garantiza que el microcontrolador ejecute continuamente las tareas, como monitorear hardware o controlar dispositivos, sin detenerse. Sin este bucle, el programa podría finalizar y el microcontrolador dejaría de operar.

- ¿Cómo se implementa el control del LED en el código?

El control del LED se realiza utilizando el pin RA0 del Puerto A. Cuando RA0 se pone en estado alto (RA0 = 1), el LED se enciende. Cuando RA0 se pone en estado bajo (RA0 = 0), el LED se apaga.

- ¿Qué ventaja tiene el uso de retardos en este proyecto?

Los retardos (__delay_ms()) permiten establecer el tiempo durante el cual el LED permanece encendido y apagado, creando el efecto de parpadeo. Sin retardos, el LED cambiaría de estado tan rápido que el ojo humano no podría percibir el parpadeo.

- ¿Qué sucede si se cambia la frecuencia del cristal?

Si se cambia la frecuencia del cristal, los retardos también cambiarían proporcionalmente. Por ejemplo, si se usa un cristal de 8 MHz en lugar de 4 MHz, el microcontrolador ejecutaría las instrucciones más rápido y los retardos serían más cortos, lo que haría que el LED parpadeara más rápidamente.

3. CONCLUSIONES

a. El uso de while(1) es esencial para asegurar que el microcontrolador siga ejecutando tareas sin detenerse. Esto es especialmente útil en sistemas embebidos donde las operaciones deben repetirse indefinidamente, como el parpadeo de un LED, la lectura de sensores, o el control de dispositivos.

b. El PIC16F84A es versátil en cuanto a su configuración mediante los registros como TRISA y PORTA, permitiendo controlar fácilmente las entradas y salidas. El uso del cristal de 4 MHz asegura una buena precisión en la temporización de los eventos.

c. Programar un PIC en C utilizando el compilador XC8 es relativamente sencillo y permite una buena legibilidad del código. Las directivas de preprocesador y las funciones de retardo permiten controlar el hardware de manera eficiente.

d. Este tipo de código es solo el inicio. Se podría ampliar para controlar más LEDs, interactuar con botones, o incluso implementar interrupciones para mejorar la eficiencia.

e. Este tipo de aplicaciones es muy útil para proyectos donde se requiere monitorear el estado de un dispositivo o señalizar visualmente el estado del sistema. Es un ejemplo claro de cómo controlar hardware externo con microcontroladores de manera sencilla.