PRÁCTICA DE PICs - TERMÓMETRO DIGITAL

NIVEL: 1
DURACIÓN: 2 horas
OBJETIVO: Los estudiantes aprenderán a conecta un sensor de temperatura LM35 al PIC y mostrar la temperatura en grados Celsius en una pantalla LCD.

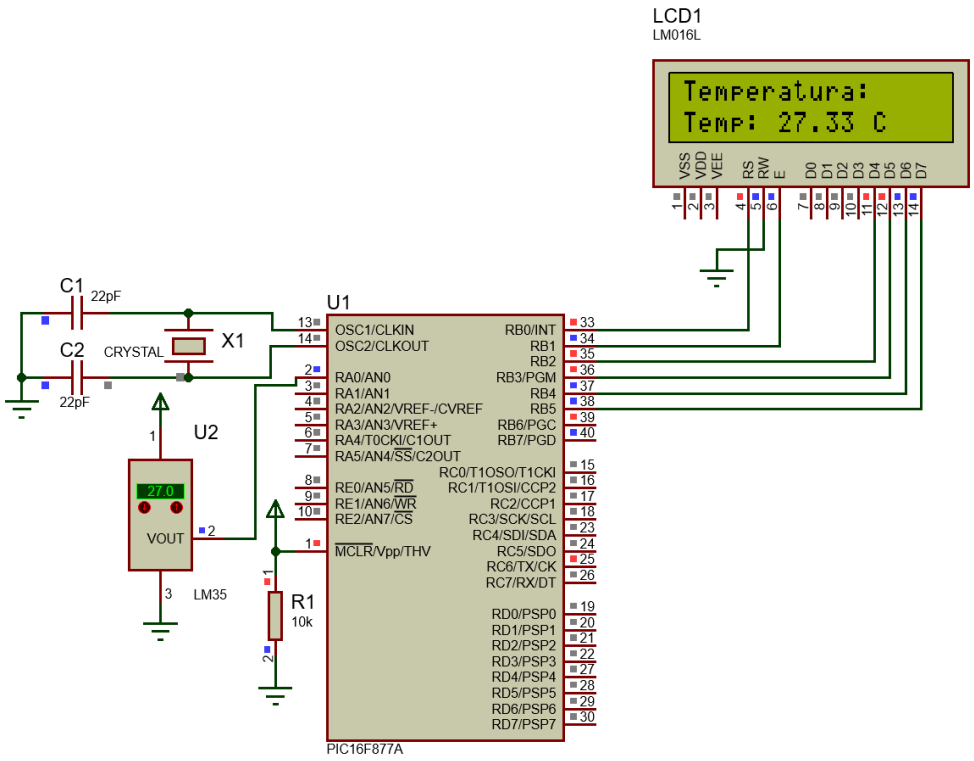
MATERIALES

- PIC16F877A
- Sensor de temperatura LM35
- Display LCD 16x2 con módulo PCF8574 (I2C)
- Resistores y potenciómetro para ajustar el contraste del LCD
- Conexiones y cables
- 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
#include "lcd.h"
#include <xc.h>
#include <stdio.h>  // Para sprintf

#define _XTAL_FREQ 4000000

// Prototipos de funciones
void ADC_Init();
int ADC_Read(int channel);
void USART_Init(long baud_rate);
void USART_Transmit(char data);
void USART_Send_String(const char *str);
void USART_Send_Float(float value);

void main(void) {
    int adc_value;
    float temperature;
    char buffer[16]; // Buffer para almacenar la temperatura en formato de caden

    // Inicializaciones
    ADC_Init();                 // Inicializa el ADC
    LCD_Init();                 // Inicializa el LCD
    USART_Init(9600);           // Inicializa USART con 9600 bps
    LCD_Set_Cursor(1, 1);
    LCD_String("Temperatura:");

    while (1) {
        // Lee el valor ADC del sensor LM35
        adc_value = ADC_Read(0);          // Lee el canal 0 (AN0)
        temperature = adc_value * 0.488;  // Convierte a grados Celsius

        // Muestra la temperatura en el LCD
        LCD_Set_Cursor(2, 1);
        sprintf(buffer, "Temp: %.2f C", temperature); // Convierte a string
        LCD_String(buffer);

        // Envía la temperatura al puerto serial
        USART_Send_String("Temperatura: ");
        USART_Send_Float(temperature);
```

```
            USART_Send_Float(temperature);
            USART_Send_String(" C\r\n");

            __delay_ms(1000);  // Actualiza cada segundo
        }
    }

    void ADC_Init() {
        ADCON0 = 0x41; // Habilita ADC y selecciona AN0
        ADCON1 = 0x80; // VDD y VSS como referencias
    }

    int ADC_Read(int channel) {
        ADCON0 &= 0xC5;                 // Limpia el canal
        ADCON0 |= channel << 3;         // Selecciona el canal
        __delay_ms(2);                  // Tiempo de adquisición
        GO_nDONE = 1;                   // Inicia la conversión ADC
        while (GO_nDONE);               // Espera a que la conversión termine
        return ((ADRESH << 8) + ADRESL); // Retorna el valor de 10 bits
    }

    void USART_Init(long baud_rate) {
        long spbrg_value;
        spbrg_value = (_XTAL_FREQ / (64 * baud_rate)) - 1;
        SPBRG = spbrg_value;            // Configura el baud rate
        TXSTA = 0x24;                   // Transmisión habilitada, modo asíncrono
        RCSTA = 0x90;                   // Habilita recepción y módulo serial
    }

    void USART_Transmit(char data) {
        while (!TXIF);                  // Espera a que el buffer esté listo
        TXREG = data;                   // Carga el dato en el registro de transmisió
    }

    void USART_Send_String(const char *str) {
        while (*str) {
            USART_Transmit(*str++);
        }
    }

    void USART_Send_Float(float value) {
        char buffer[10];
        sprintf(buffer, "%.2f", value); // Convierte el valor a string con 2 decimales
        USART_Send_String(buffer);
    }
```

Figura 1. Source Files (main.c)

```c
#include "lcd.h"
#define _XTAL_FREQ 4000000

void LCD_Enable() {
    LCD_EN = 1;
    __delay_ms(1);
    LCD_EN = 0;
}

void LCD_Command(unsigned char cmd) {
    LCD_RS = 0;                         // Modo comando
    LCD_D4 = (cmd >> 4) & 1;
    LCD_D5 = (cmd >> 5) & 1;
    LCD_D6 = (cmd >> 6) & 1;
    LCD_D7 = (cmd >> 7) & 1;
    LCD_Enable();
    LCD_D4 = cmd & 1;
    LCD_D5 = (cmd >> 1) & 1;
    LCD_D6 = (cmd >> 2) & 1;
    LCD_D7 = (cmd >> 3) & 1;
    LCD_Enable();
}

void LCD_Char(char data) {
    LCD_RS = 1;                         // Modo datos
    LCD_D4 = (data >> 4) & 1;
    LCD_D5 = (data >> 5) & 1;
    LCD_D6 = (data >> 6) & 1;
    LCD_D7 = (data >> 7) & 1;
    LCD_Enable();
    LCD_D4 = data & 1;
    LCD_D5 = (data >> 1) & 1;
    LCD_D6 = (data >> 2) & 1;
    LCD_D7 = (data >> 3) & 1;
    LCD_Enable();
}
```

```c
void LCD_Char(char data) {
    LCD_RS = 1;                         // Modo datos
    LCD_D4 = (data >> 4) & 1;
    LCD_D5 = (data >> 5) & 1;
    LCD_D6 = (data >> 6) & 1;
    LCD_D7 = (data >> 7) & 1;
    LCD_Enable();
    LCD_D4 = data & 1;
    LCD_D5 = (data >> 1) & 1;
    LCD_D6 = (data >> 2) & 1;
    LCD_D7 = (data >> 3) & 1;
    LCD_Enable();
}

void LCD_Init() {
    TRISB = 0x00;                       // Configura PORTB como salida
    __delay_ms(20);                     // Espera inicial
    LCD_Command(0x02);                  // Modo 4 bits
    LCD_Command(0x28);                  // LCD 2 líneas, 5x7 matriz
    LCD_Command(0x0C);                  // Enciende el display
    LCD_Command(0x06);                  // Modo de entrada
    LCD_Command(0x01);                  // Limpia el display
}

void LCD_Set_Cursor(unsigned char row, unsigned char col) {
    unsigned char pos = (row == 1) ? 0x80 + col - 1 : 0xC0 + col - 1;
    LCD_Command(pos);
}

void LCD_String(const char *str) {
    while (*str) {
        LCD_Char(*str++);
    }
}
```

Figura 2. Header Files (lcd.c)

```c
#ifndef LCD_H
#define LCD_H

#include <xc.h>

#define LCD_RS PORTBbits.RB0
#define LCD_EN PORTBbits.RB1
#define LCD_D4 PORTBbits.RB2
#define LCD_D5 PORTBbits.RB3
#define LCD_D6 PORTBbits.RB4
#define LCD_D7 PORTBbits.RB5

void LCD_Init();
void LCD_Command(unsigned char cmd);
void LCD_Char(char data);
void LCD_String(const char *str);
void LCD_Set_Cursor(unsigned char row, unsigned char col);

#endif
```

Figura 2. Header Files (lcd.h)

Explicación

- main.c

- Declara las variables adc_value y temperature para almacenar el valor del ADC y la temperatura en grados Celsius, respectivamente.
- Declara buffer para formatear la temperatura como una cadena de texto.
- Inicializa el ADC, el LCD, y el puerto serial (USART) a 9600 baudios.
- Muestra el texto "Temperatura:" en la primera línea del LCD.
- Bucle principal: Lee el valor del sensor LM35 (canal 0 del ADC), convierte el valor leído a grados Celsius usando un factor de conversión basado en la sensibilidad del LM35 (10 mV/°C).
- Posiciona el cursor en la segunda línea del LCD y muestra la temperatura en el formato "Temp: XX.XX C" usando sprintf para formatear el valor en el buffer.
- Envía el texto "Temperatura: ", el valor de la temperatura y " C" al puerto serial, para ser visualizado en un terminal en la computadora.

- lcd.c

- Incluye lcd.h para utilizar las macros de control de pines y define la frecuencia del oscilador.
- LCD_Enable: Crea un pulso en el pin Enable (E) para que el LCD lea los datos o comandos enviados.
- LCD_Command: Envía un comando al LCD (ej., limpiar pantalla o configurar el cursor). Divide el comando en nibbles alto y bajo y los envía secuencialmente.

- LCD_Char: Envía un carácter al LCD usando el modo de 4 bits. Funciona de forma similar a LCD_Command, pero con RS en alto.
- LCD_Init: Configura el LCD en modo de 4 bits y establece varias opciones, como el tipo de matriz de caracteres, el modo de incremento automático y limpia la pantalla.
- LCD_Set_Cursor: Posiciona el cursor en la fila y columna especificada en el LCD.
- LCD_String: Envía una cadena de caracteres al LCD, mostrando un texto completo

- lcd.h

- Define la cabecera y evita múltiples inclusiones con #ifndef y #define.
- Define los pines del PIC utilizados para controlar los pines RS, EN, D4-D7 del LCD, facilitando el control del LCD en modo de 4 bits.
- Declara las funciones utilizadas en lcd.c para inicializar y controlar el LCD.

2. ANÁLISIS DE RESULTADOS

- ¿La lectura de temperatura en el LCD es precisa?

La precisión de la temperatura depende del factor de conversión usado y de la calibración del sensor LM35. Al usar el factor de 0.488 (aproximadamente 5V / 1024 / 0.01V), se obtienen lecturas precisas, siempre que el voltaje de referencia (VDD) del ADC sea estable.

- ¿El envío de datos al puerto serial funciona de manera correcta?

El puerto serial transmite los datos correctamente si el baud rate está configurado a 9600 bps, tanto en el PIC como en el programa de monitoreo en la computadora. Esto permite visualizar los datos en tiempo real en el terminal serial.

- ¿La frecuencia del oscilador afecta el funcionamiento?

Sí. La frecuencia del oscilador afecta los retardos y la velocidad de transmisión serial. Con _XTAL_FREQ definido en 4 MHz, los retardos y el baud rate están correctamente sincronizados para el funcionamiento del LCD y del puerto serial.

- ¿El LCD muestra los valores de temperatura sin problemas?

Sí, el LCD muestra los valores de temperatura en el formato adecuado ("Temp: XX.XX C") utilizando el modo de 4 bits. Esto permite una representación clara y estable en el LCD.

- ¿La comunicación entre el PIC y el FT232R es estable?

La comunicación con el FT232R es estable siempre que los pines de transmisión y recepción estén conectados correctamente (TX-RX y RX-TX). El FT232R funciona adecuadamente como adaptador de nivel para el puerto serial TTL, permitiendo la transmisión de datos a la computadora sin problemas.

- ¿La precisión de la temperatura es constante?

La precisión es constante si el sensor LM35 se alimenta de un voltaje estable y si el ambiente tiene cambios de temperatura lentos o moderados. Cambios bruscos pueden requerir un mayor número de muestras o promediado para estabilizar las lecturas.

3. CONCLUSIONES

a. El proyecto logra implementar un termómetro digital que utiliza el sensor LM35, con lecturas de temperatura mostradas en un LCD y transmitidas a una computadora a través de un puerto serial, lo cual permite tanto una visualización local como remota.

b. El proyecto hace uso de varias características del PIC16F877A, como el ADC para convertir la señal analógica del LM35, el módulo USART para la comunicación serial, y el control en modo de 4 bits del LCD. Esto demuestra el uso eficiente de los recursos de un microcontrolador en un proyecto práctico.

c. La conversión de la señal del LM35 a grados Celsius mediante el ADC ofrece una precisión adecuada para aplicaciones de monitoreo ambiental. Sin embargo, para aplicaciones que requieren mayor precisión, puede ser necesario implementar técnicas de calibración o usar un voltaje de referencia preciso.

d. La transmisión de la temperatura a través del puerto serial permite monitorear los datos en tiempo real en una computadora. Esto amplía las posibilidades del proyecto, permitiendo el registro de datos o su uso en aplicaciones de monitoreo remoto.

e. El proyecto es sensible a fluctuaciones en el voltaje de alimentación. Para obtener lecturas más estables, se podría utilizar un regulador de voltaje preciso para el sensor LM35.

f. La implementación actual usa __delay_ms(), lo cual puede limitar otras operaciones si se requiere un procesamiento más complejo. En proyectos futuros, se podría usar un temporizador para manejar el tiempo de muestreo sin bloquear otras tareas.

g. Este proyecto es una introducción a varios conceptos importantes en sistemas embebidos, como la conversión analógica-digital, el uso de un display LCD en modo de 4 bits, y la comunicación serial. Es

una práctica completa que abarca tanto el hardware como el software en el diseño de sistemas de medición y monitoreo.