

**UNIVERSIDAD NACIONAL DE INGENIERIA (UNI)**

**AREA DE CONOCIMIENTO DE TECNOLOGIA DE LA INFORMACION Y COMUNICACIÓN**

**ARQUITECTURA DE MAQUINAS DE COMPUTADORAS**

**Integrantes: Raymar Atahualpa Cruz Orozco // 2018-0302U**

**Jamie Sicely Rodríguez Sanchez// 2021-0772I**

**Juridia Del Rosario Mayorga Salinas //2022-0365U**

**Jesly Isayana Moraga Pastran //2022-0485U**

**Docente: Juan Carlos Aguirre**

**Grupo: 4E1-COM-S**

**Fecha: 08/05/2025**

**ACTIVIDAD 9. INVESTIGUE LOS CONCEPTOS Y SU FUNCIONALIDAD DE LOS SIGUIENTES PUNTOS.**

**PROGRAMADORES DEL MICROCONTROLADOR**

**• Repertorio de Instrucciones.**

**• Instrucciones de transferencia de datos**

**• Instrucciones aritméticas y lógicas**

**• Control de flujo de programa**

**• Salto incondicional**

**• Salto condicionado**

**• Instrucciones de bits**

**PROGRAMADORES DEL MICROCONTROLADOR**

Los **programadores** son dispositivos o herramientas que permiten cargar el código (firmware) desde una computadora al microcontrolador. Sirven para transferir el programa escrito en un lenguaje de alto nivel (como C) o en ensamblador hacia la memoria del microcontrolador.

**Tipos de programadores:**

* **ICSP (In-Circuit Serial Programming):** Permite programar el microcontrolador sin sacarlo del circuito.
* **USBasp / AVRISP / PICkit / ST-Link:** Son ejemplos de programadores usados con familias de microcontroladores como AVR, PIC o STM32.

**REPERTORIO DE INSTRUCCIONES**

El **repertorio de instrucciones** (instruction set) es el conjunto de todas las instrucciones que un microcontrolador puede entender y ejecutar. Define el lenguaje interno del microcontrolador.

**Clasificación general:**

* Instrucciones de transferencia de datos
* Instrucciones aritméticas y lógicas
* Instrucciones de control de flujo
* Instrucciones de manipulación de bits

**INSTRUCCIONES DE TRANSFERENCIA DE DATOS**

Estas instrucciones permiten mover datos de un lugar a otro, ya sea entre registros, memoria o puertos.

**Ejemplos:**

* MOV: Mueve datos de una ubicación a otra.
* LDS, STS: Cargar/almacenar datos desde/hacia memoria.
* IN, OUT: Transferencia entre registros y puertos de E/S.

**INSTRUCCIONES ARITMÉTICAS Y LÓGICAS**

Permiten realizar operaciones matemáticas o de comparación.

**Aritméticas:**

* ADD: Suma
* SUB: Resta
* INC, DEC: Incremento/Decremento
* MUL, DIV: Multiplicación y división (según el microcontrolador)

**Lógicas:**

* AND, OR, XOR: Operaciones booleanas
* NOT o COM: Complemento lógico
* CLR: Limpia (pone en cero) un registro
* TST: Compara sin alterar registros

**CONTROL DE FLUJO DE PROGRAMA**

Estas instrucciones alteran la secuencia normal de ejecución del programa, permitiendo repetir secciones de código, saltar a otras o tomar decisiones.

**SALTO INCONDICIONAL**

Hace que el programa salte a una dirección específica sin importar ninguna condición.

**Ejemplo:**

* JMP: Salto directo a otra dirección.
* RJMP: Salto relativo (hacia adelante o atrás).

**SALTO CONDICIONADO**

Ejecuta el salto **solo si se cumple una condición**, como si un registro tiene cierto valor o si el resultado de una operación fue cero.

**Ejemplos:**

* BREQ: Salta si el resultado anterior fue igual a cero.
* BRNE: Salta si fue distinto.
* BRCS, BRCC: Saltos basados en el bit de acarreo.
* CP, CPI: Comparación de registros para condicionar el salto.

**INSTRUCCIONES DE BITS**

Manipulan bits individuales dentro de registros o puertos.

**Ejemplos:**

* SETB: Establece (pone en 1) un bit.
* CLR: Limpia un bit (pone en 0).
* CPL: Complementa un bit (invierte su valor).
* SBIC, SBIS: Salta si el bit está limpio/seteado.
* SBI, CBI: Establece o limpia un bit específico de un puerto de E/S.

Realiza pequeños programas de un microcontrolador utilizando las plataformas MPLAB

1. Encender y apagar un LED con retardo Problema: Controlar el parpadeo de un LED conectado al pin RC0.

#include <xc.h>

// CONFIGURACIÓN DE FUSES

#pragma config FOSC = XT // Oscilador XT

#pragma config WDTE = OFF // Watchdog Timer desactivado

#pragma config PWRTE = ON // Power-up Timer activado

#pragma config BOREN = ON // Brown-out Reset activado

#pragma config LVP = OFF // Programación baja tensión desactivada

#pragma config CPD = OFF // Protección EEPROM desactivada

#pragma config WRT = OFF // Escritura en memoria desactivada

#pragma config CP = OFF // Protección de código desactivada

#define \_XTAL\_FREQ 4000000 // Frecuencia del oscilador (4 MHz)

void main(void) {

TRISCbits.TRISC0 = 0; // Configura RC0 como salida

PORTCbits.RC0 = 0; // Inicialmente en bajo (LED apagado)

while (1) {

PORTCbits.RC0 = 1; // Enciende el LED

\_\_delay\_ms(500); // Espera 500 ms

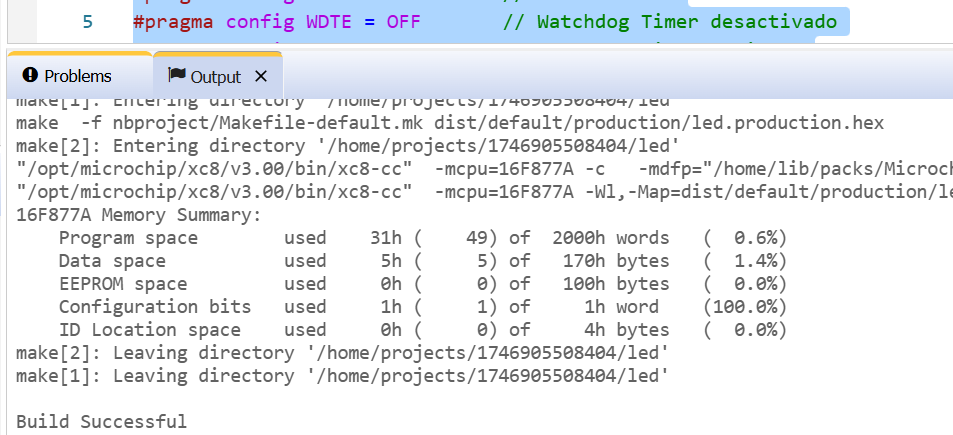
PORTCbits.RC0 = 0; // Apaga el LED

\_\_delay\_ms(500); // Espera 500 ms

}

}

Salida



1. Leer un botón y encender un LED Problema: Si se presiona un botón conectado a RA0, encender un LED en RC0.

#include <xc.h>

#define \_XTAL\_FREQ 4000000 // Frecuencia del oscilador (4 MHz)

void main(void) {

// Configurar RA0 como entrada (botón) y RC0 como salida (LED)

TRISA = 0x01; // RA0 como entrada (0b00000001), otros pines de PORTA como salida

TRISC = 0xFE; // RC0 como salida (0b11111110), otros pines de PORTC como entrada

// Inicializar el LED en apagado

PORTC = 0x00;

while (1) {

if (PORTAbits.RA0 == 0) { // Si el botón está presionado (asumido en lógica baja)

PORTCbits.RC0 = 1; // Encender el LED

} else {

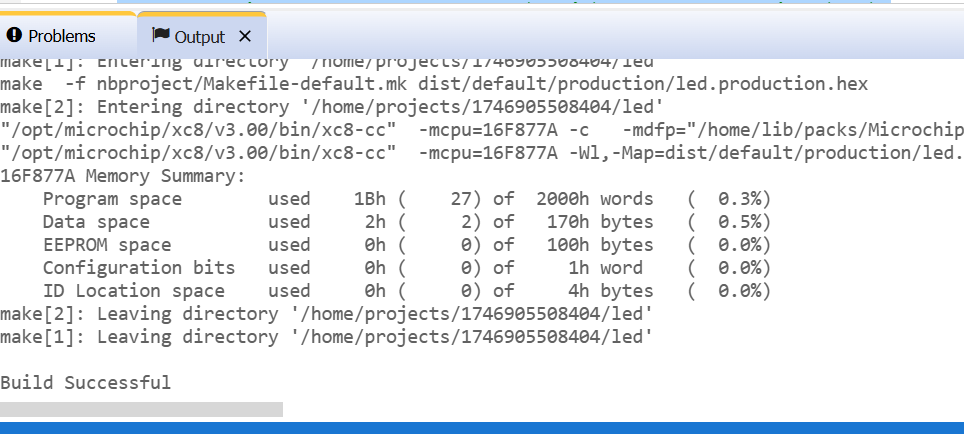
PORTCbits.RC0 = 0; // Apagar el LED

}

}

}

Salida



1. Medir voltaje con el ADC y encender LED si supera un umbral Problema: Medir un voltaje analógico en RA0. Si supera 2V, encender LED en RC0. (Suponiendo Vref = 5V y 10 bits de resolución)

#include <xc.h>

#define \_XTAL\_FREQ 4000000 // Frecuencia del oscilador (4 MHz)

#define ADC\_THRESHOLD 409 // Umbral para 2V (409 en 10 bits de resolución)

void ADC\_Init(void) {

// Configurar el ADC

//ADCON1 = 0x0E; // Configura RA0 como entrada analógica, el resto como digitales

//ADCON2 = 0xA0; // Justificado a la derecha, reloj Fosc/32

ADCON0 = 0x01; // Habilitar ADC y seleccionar el canal AN0 (RA0)

}

unsigned int ADC\_Read(void) {

// Iniciar la conversión ADC

ADCON0bits.GO\_nDONE = 1;

while (ADCON0bits.GO\_nDONE); // Esperar a que termine la conversión

return ((ADRESH << 8) | ADRESL); // Retornar el valor de 10 bits

}

void main(void) {

// Configurar RA0 como entrada (ADC) y RC0 como salida (LED)

TRISA = 0x01; // RA0 como entrada

TRISC = 0xFE; // RC0 como salida

PORTC = 0x00; // Apagar LED inicialmente

ADC\_Init(); // Inicializar el ADC

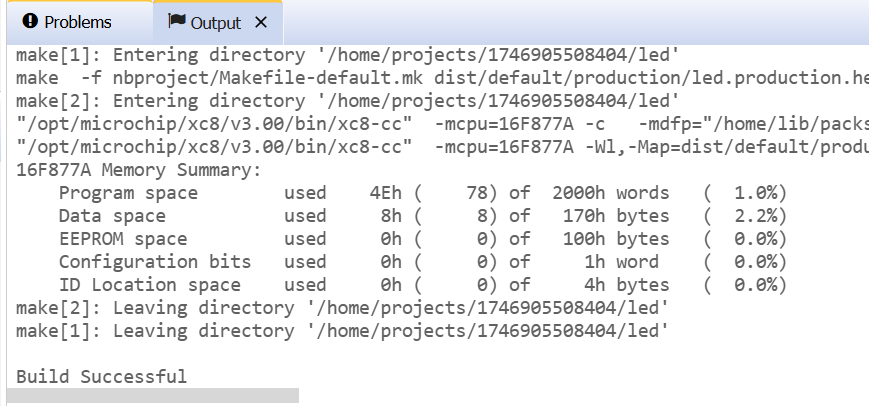
while (1) {

unsigned int adc\_value = ADC\_Read(); // Leer valor del ADC

if (adc\_value > ADC\_THRESHOLD) { // Si el valor supera 2V (409)

PORTCbits.RC0 = 1; // Encender el LED

} else {

PORTCbits.RC0 = 0; // Apagar el LED 

1. PWM para controlar velocidad de motor Problema: Generar una señal PWM con duty cycle del 50% usando CCP1 (RC2).

#include <xc.h>

#define \_XTAL\_FREQ 4000000 // Oscilador de 4MHz

// Configuración de fuses (ajústalos según tu entorno real)

#pragma config FOSC = HS

#pragma config WDTE = OFF

#pragma config PWRTE = ON

#pragma config BOREN = ON

#pragma config LVP = OFF

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

void PWM\_Init(void) {

TRISC2 = 0; // Salida PWM

PR2 = 249; // Periodo PWM ~2kHz con Fosc=4MHz y prescaler=4

CCP1CON = 0b00001100; // PWM mode

CCPR1L = 124; // 50% de duty: 124 = 50% de 249

CCP1CON |= 0b00000000; // DC1B1:DC1B0 = 00

T2CON = 0b00000101; // Prescaler 4, encenderá luego

// Interrupción Timer2

PIE1bits.TMR2IE = 1; // Habilitar interrupción TMR2

INTCONbits.PEIE = 1; // Habilitar interrupciones periféricas

INTCONbits.GIE = 1; // Habilitar interrupciones globales

T2CONbits.TMR2ON = 1; // Encender TMR2

}

void \_\_interrupt() ISR(void) {

if (PIR1bits.TMR2IF) {

PIR1bits.TMR2IF = 0; // Limpiar bandera

PORTDbits.RD0 = !PORTDbits.RD0; // Alternar LED

}

}

void main(void) {

TRISD0 = 0; // LED como salida

PORTD = 0;

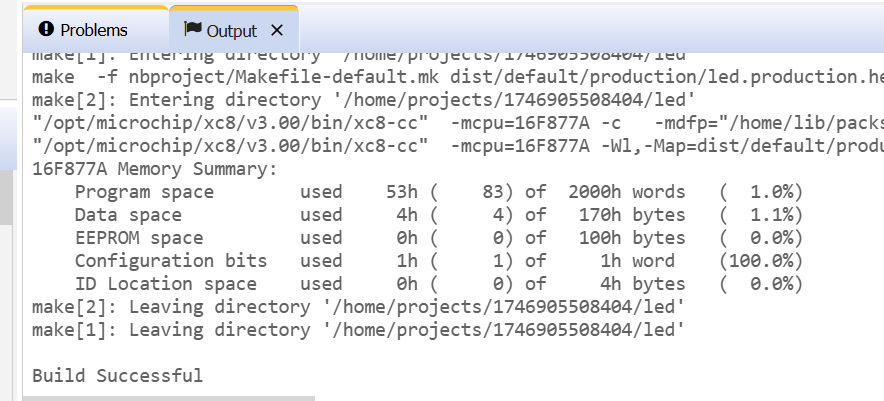
PWM\_Init();

while(1) {

// PWM e interrupciones funcionan solas

}

}



Ejemplo con Interrupciones: Botón que activa interrupción externa Problema: Cuando se presione un botón conectado a RB0 (INT), se enciende un LED en RC0 por 1 segundo usando una interrupción externa.

#include <xc.h>

#define \_XTAL\_FREQ 4000000 // Oscilador de 4MHz

// Configuración de fuses

#pragma config FOSC = HS // Oscilador externo

#pragma config WDTE = OFF // Watchdog Timer desactivado

#pragma config PWRTE = ON // Power-up Timer activado

#pragma config BOREN = ON // Brown-out Reset activado

#pragma config LVP = OFF // Low Voltage Programming desactivado

#pragma config CPD = OFF // Data EEPROM Memory Code Protection off

#pragma config WRT = OFF // Flash Program Memory Write Enable off

#pragma config CP = OFF // Code Protection off

void main(void) {

// Configuración de pines

TRISB0 = 1; // RB0 como entrada (botón)

TRISC0 = 0; // RC0 como salida (LED)

PORTC = 0;

// Configuración de interrupción externa

INTEDG = 1; // Interrupción en flanco de subida

INTF = 0; // Borrar bandera de interrupción externa

INTE = 1; // Habilitar interrupción externa

GIE = 1; // Habilitar interrupciones globales

while(1) {

// Loop principal vacío; espera interrupción

}

}

// Rutina de Interrupción

void \_\_interrupt() ISR(void) {

if (INTF) { // Verifica si la interrupción fue por RB0

PORTCbits.RC0 = 1; // Enciende LED

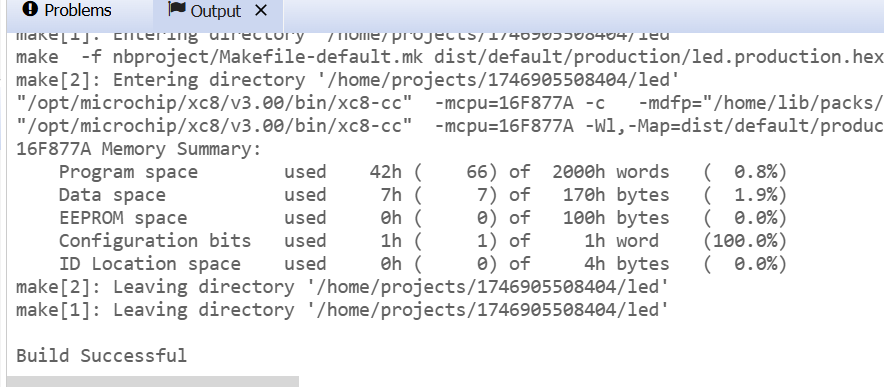
\_\_delay\_ms(1000); // Espera 1 segundo

PORTCbits.RC0 = 0; // Apaga LED

INTF = 0; // Limpia la bandera de interrupción

}

}



2. Ejemplo con USART: Enviar datos por UART Problema: Enviar un texto por el puerto serial continuamente usando el módulo USART del PIC. Son ideales para comprender el uso de interrupciones y comunicación serial en sistemas embebidos

#include <xc.h>

#define \_XTAL\_FREQ 4000000 // Oscilador de 4 MHz

// Configuración de fuses

#pragma config FOSC = HS

#pragma config WDTE = OFF

#pragma config PWRTE = ON

#pragma config BOREN = ON

#pragma config LVP = OFF

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

// Función para inicializar el USART

void USART\_Init(unsigned int baud) {

unsigned int spbrg\_val = (\_XTAL\_FREQ / (64UL \* baud)) - 1;

SPBRG = spbrg\_val; // Velocidad de transmisión

BRGH = 0; // Modo baja velocidad

SYNC = 0; // Modo asincrónico

SPEN = 1; // Habilita el módulo serial

TXEN = 1; // Habilita transmisión

TRISC6 = 0; // TX como salida

}

// Función para enviar un carácter

void USART\_TxChar(char c) {

while (!TXIF); // Espera que el buffer esté vacío

TXREG = c;

}

// Función para enviar una cadena de texto

void USART\_SendString(const char \*str) {

while (\*str != '\0') {

USART\_TxChar(\*str);

str++;

}

}

void main(void) {

USART\_Init(9600); // Inicializa UART a 9600 baudios

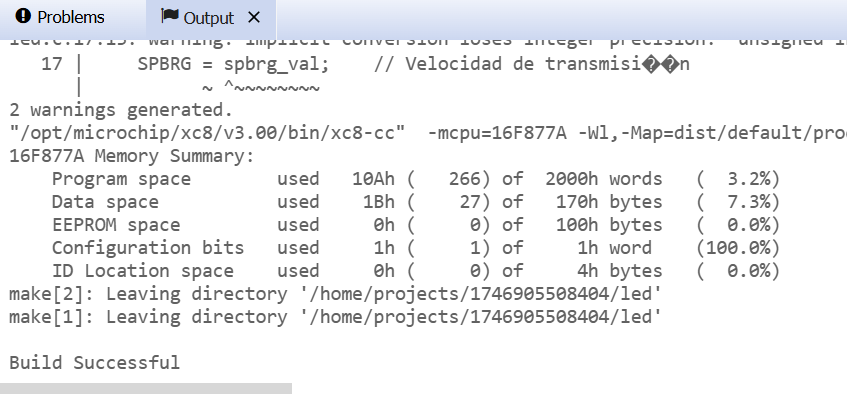
while (1) {

USART\_SendString("Hola desde PIC\r\n");

\_\_delay\_ms(1000);

}

}



**COMUNICACIÓN ENTRE EL MICROCONTROLADOR Y SENSORES O MEMORIAS EXTERNAS.**

Ejemplo con I2C: Comunicación maestro-esclavo (lectura desde EEPROM 24LC256)

Problema: El microcontrolador PIC actúa como maestro I2C y lee un byte desde la dirección 0x0000 de una EEPROM externa 24LC256, mostrando el valor en un LED binario conectado al PORTC. Requisitos:

• EEPROM I2C 24LC256

• SDA y SCL conectados a RC4 y RC3 (I2C del PIC16F877A)

• Pull-ups de 4.7kΩ en SDA/SCL

#include <xc.h>

#define \_XTAL\_FREQ 4000000

// CONFIG BITS

#pragma config FOSC = HS

#pragma config WDTE = OFF

#pragma config PWRTE = ON

#pragma config BOREN = ON

#pragma config LVP = OFF

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

#define EEPROM\_ADDRESS 0xA0 // Dirección base (control byte: 1010000X)

void I2C\_Init(void) {

SSPCON = 0b00101000; // Habilita I2C Master mode

SSPCON2 = 0;

SSPADD = ((\_XTAL\_FREQ / (4 \* 100000)) - 1); // 100kHz

SSPSTAT = 0;

TRISC3 = 1; // SCL como entrada

TRISC4 = 1; // SDA como entrada

}

void I2C\_Wait(void) {

while ((SSPCON2 & 0x1F) || (SSPSTAT & 0x04));

}

void I2C\_Start(void) {

I2C\_Wait();

SEN = 1;

}

void I2C\_RepeatedStart(void) {

I2C\_Wait();

RSEN = 1;

}

void I2C\_Stop(void) {

I2C\_Wait();

PEN = 1;

}

void I2C\_Write(unsigned char data) {

I2C\_Wait();

SSPBUF = data;

}

unsigned char I2C\_Read(unsigned char ack) {

unsigned char received;

I2C\_Wait();

RCEN = 1;

I2C\_Wait();

received = SSPBUF;

I2C\_Wait();

ACKDT = (ack) ? 0 : 1; // ACK = 0, NACK = 1

ACKEN = 1;

return received;

}

unsigned char EEPROM\_Read(unsigned int address) {

unsigned char data;

I2C\_Start();

I2C\_Write(EEPROM\_ADDRESS); // Dirección de escritura

I2C\_Write((address >> 8) & 0xFF); // Dirección alta

I2C\_Write(address & 0xFF); // Dirección baja

I2C\_RepeatedStart();

I2C\_Write(EEPROM\_ADDRESS | 1); // Dirección de lectura

data = I2C\_Read(0); // Leer sin ACK

I2C\_Stop();

return data;

}

void main(void) {

unsigned char eeprom\_data;

TRISC = 0x00; // PORTC como salida

PORTC = 0x00;

I2C\_Init();

while (1) {

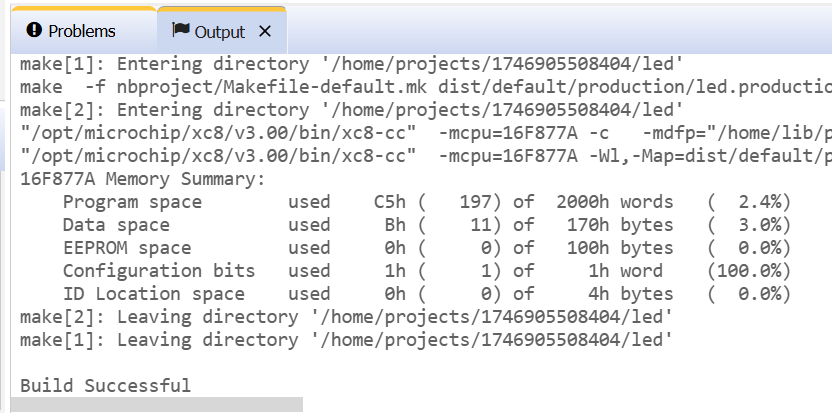
eeprom\_data = EEPROM\_Read(0x0000); // Leer desde dirección 0x0000

PORTC = eeprom\_data; // Mostrar en LEDs conectados a PORTC

\_\_delay\_ms(1000);

}

}



Mini Proyecto: Terminal Serial que consulta temperatura a un sensor I2C (LM75) al presionar un botón (interrupción externa) Objetivo: • Al presionar un botón (INT en RB0), se activa una interrupción externa. • El PIC lee la temperatura desde un sensor LM75 vía I2C. • Luego, envía la temperatura por UART a una terminal serial (como PuTTY)

#include <xc.h>

#include <stdio.h>

#include <stdint.h>

// CONFIGURACIÓN

#pragma config FOSC = HS // Oscilador HS

#pragma config WDTE = OFF // Watchdog Timer desactivado

#pragma config PWRTE = ON // Power-up Timer habilitado

#pragma config BOREN = ON // Brown-out Reset habilitado

#pragma config LVP = OFF // Programación en baja tensión deshabilitada

#pragma config CPD = OFF // Protección del EEPROM desactivada

#pragma config WRT = OFF // Escritura protegida desactivada

#pragma config CP = OFF // Protección de código desactivada

#define \_XTAL\_FREQ 4000000 // Oscilador de 4 MHz

#define LM75\_ADDRESS 0x90 // Dirección I2C del LM75 con bit R/W incluido

volatile uint8\_t flag\_leer\_temp = 0; // Bandera activada por la interrupción

// === FUNCIONES UART ===

void UART\_Init(void) {

TXSTA = 0x24; // Transmisión habilitada, modo asincrónico, 8 bits

RCSTA = 0x90; // Receptor habilitado, puerto serie habilitado

SPBRG = 25; // 9600 baudios con Fosc = 4MHz

TRISC6 = 0; // TX

TRISC7 = 1; // RX

}

void UART\_WriteChar(char c) {

while (!TXIF);

TXREG = c;

}

void UART\_WriteString(const char\* str) {

while (\*str) {

UART\_WriteChar(\*str++);

}

}

// === FUNCIONES I2C ===

void I2C\_Init(void) {

SSPCON = 0x28; // I2C Master mode, clock = Fosc / (4 \* (SSPADD + 1))

SSPCON2 = 0x00;

SSPADD = 9; // 100kHz @ 4MHz Fosc

SSPSTAT = 0x00;

TRISC3 = 1; // SCL

TRISC4 = 1; // SDA

}

void I2C\_Wait(void) {

while ((SSPCON2 & 0x1F) || (SSPSTAT & 0x04));

}

void I2C\_Start(void) {

I2C\_Wait();

SEN = 1;

}

void I2C\_Stop(void) {

I2C\_Wait();

PEN = 1;

}

void I2C\_Write(uint8\_t data) {

I2C\_Wait();

SSPBUF = data;

}

uint8\_t I2C\_Read(uint8\_t ack) {

uint8\_t data;

I2C\_Wait();

RCEN = 1;

while (!BF);

data = SSPBUF;

I2C\_Wait();

ACKDT = (ack) ? 0 : 1;

ACKEN = 1;

return data;

}

// === INTERRUPCIÓN EXTERNA ===

void \_\_interrupt() ISR(void) {

if (INTF) {

flag\_leer\_temp = 1;

INTF = 0;

}

}

// === FUNCIÓN PARA LEER TEMPERATURA ===

int Leer\_Temperatura(void) {

int temp = 0;

I2C\_Start();

I2C\_Write(LM75\_ADDRESS); // Dirección + Write

I2C\_Write(0x00); // Registro de temperatura

I2C\_Start();

I2C\_Write(LM75\_ADDRESS | 0x01); // Dirección + Read

uint8\_t msb = I2C\_Read(1);

uint8\_t lsb = I2C\_Read(0);

I2C\_Stop();

// LM75 devuelve temperatura en MSB (formato complemento a 2)

if (msb & 0x80) {

temp = (int)(msb | 0xFF00); // negativo

} else {

temp = msb;

}

return temp;

}

// === MAIN ===

void main(void) {

TRISB0 = 1; // Botón como entrada

TRISC = 0x00; // PORTC como salida

PORTC = 0x00;

UART\_Init();

I2C\_Init();

// Configurar interrupción externa

INTE = 1;

INTEDG = 1; // Flanco de subida

GIE = 1;

UART\_WriteString("Sistema listo. Esperando boton...\r\n");

while (1) {

if (flag\_leer\_temp) {

flag\_leer\_temp = 0;

int temp = Leer\_Temperatura();

char buffer[32];

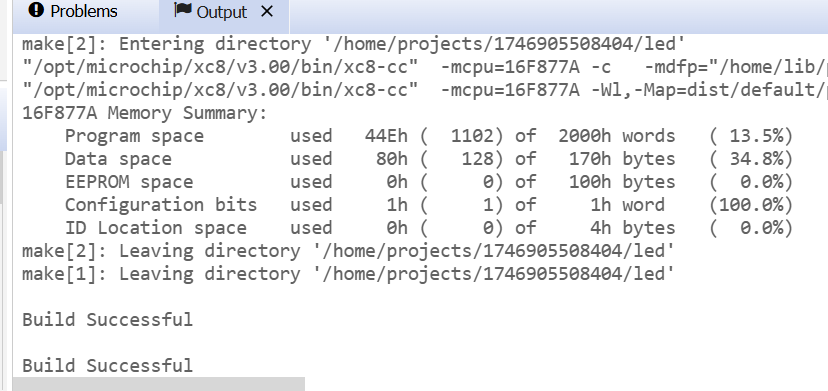
sprintf(buffer, "Temperatura: %d C\r\n", temp);

UART\_WriteString(buffer);

}

}

}



Requisitos: • Microcontrolador: PIC16F877A

• Sensor I2C: LM75 (dirección típica 0x90)

• Comunicación UART a 9600 baudios

• Botón en RB0 (interrupción externa)

• Pull-ups en SDA/SCL

**Resumen del Proyecto**

| **Elemento** | **Detalles** |
| --- | --- |
| **Microcontrolador** | PIC16F877A |
| **Sensor de temperatura** | LM75 (I2C, dirección 0x90 o 0x48 en 7 bits) |
| **Botón** | Conectado a **RB0**, activa **interrupción externa INT** |
| **UART** | Baud rate de **9600**, para comunicación serial con terminal como PuTTY |
| **I2C** | SDA → RC4, SCL → RC3, con resistencias pull-up de **4.7kΩ** |
| **Voltaje** | 5V tanto para el PIC como para el LM75 |

**Conexiones Electrónicas**

| **Componente** | **PIN PIC16F877A** | **Observaciones** |
| --- | --- | --- |
| **LM75 SDA** | RC4 (pin 23) | Línea de datos I2C (requiere pull-up de 4.7kΩ a Vcc) |
| **LM75 SCL** | RC3 (pin 18) | Línea de reloj I2C (pull-up de 4.7kΩ a Vcc) |
| **Botón** | RB0 (pin 33) | Activará interrupción externa (usa resistencia de pull-down o pull-up) |
| **TX (PIC)** | RC6 (pin 25) | Transmisión serial hacia PC (a través de conversor RS232 o USB-TTL) |
| **GND / Vcc** | GND y Vcc del PIC y LM75 conectados entre sí | 5V en común |

**Configuración de UART**:

// === FUNCIONES UART ===

void UART\_Init(void) {

TXSTA = 0x24; // Transmisión habilitada, modo asincrónico, 8 bits

RCSTA = 0x90; // Receptor habilitado, puerto serie habilitado

SPBRG = 25; // 9600 baudios con Fosc = 4MHz

TRISC6 = 0; // TX

TRISC7 = 1; // RX

}

void UART\_WriteChar(char c) {

while (!TXIF);

TXREG = c;

}

void UART\_WriteString(const char\* str) {

while (\*str) {

UART\_WriteChar(\*str++);

}

}

**Manejo de I2C y lectura del LM75:**

// === FUNCIONES I2C ===

void I2C\_Init(void) {

SSPCON = 0x28; // I2C Master mode, clock = Fosc / (4 \* (SSPADD + 1))

SSPCON2 = 0x00;

SSPADD = 9; // 100kHz @ 4MHz Fosc

SSPSTAT = 0x00;

TRISC3 = 1; // SCL

TRISC4 = 1; // SDA

}

void I2C\_Wait(void) {

while ((SSPCON2 & 0x1F) || (SSPSTAT & 0x04));

}

void I2C\_Start(void) {

I2C\_Wait();

SEN = 1;

}

void I2C\_Stop(void) {

I2C\_Wait();

PEN = 1;

}

void I2C\_Write(uint8\_t data) {

I2C\_Wait();

SSPBUF = data;

}

uint8\_t I2C\_Read(uint8\_t ack) {

uint8\_t data;

I2C\_Wait();

RCEN = 1;

while (!BF);

data = SSPBUF;

I2C\_Wait();

ACKDT = (ack) ? 0 : 1;

ACKEN = 1;

Return

**Interrupción Externa**:

// === INTERRUPCIÓN EXTERNA ===

void \_\_interrupt() ISR(void) {

if (INTF) {

flag\_leer\_temp = 1; // Activamos la bandera cuando se presiona el botón

INTF = 0; // Limpiamos la bandera de la interrupción

}

}

**Interrupción Externa**:

// === INTERRUPCIÓN EXTERNA ===

void \_\_interrupt() ISR(void) {

if (INTF) {

flag\_leer\_temp = 1; // Activamos la bandera cuando se presiona el botón

INTF = 0; // Limpiamos la bandera de la interrupción

}

}

**Main**:

// === MAIN ===

void main(void) {

TRISB0 = 1; // Configuramos RB0 como entrada (botón)

TRISC = 0x00; // Configuramos PORTC como salida

PORTC = 0x00; // Inicializamos PORTC en 0 (todos los LEDs apagados)

UART\_Init(); // Inicializamos UART

I2C\_Init(); // Inicializamos I2C

// Configuración de la interrupción externa

INTE = 1; // Habilitamos la interrupción externa

INTEDG = 1; // Flanco de subida (cuando el botón se presiona)

GIE = 1; // Habilitamos las interrupciones globales

UART\_WriteString("Sistema listo. Esperando botón...\r\n");

while (1) {

if (flag\_leer\_temp) {

flag\_leer\_temp = 0; // Limpiamos la bandera

int temp = Leer\_Temperatura(); // Leemos la temperatura del LM75

char buffer[32];

sprintf(buffer, "Temperatura: %d C\r\n", temp);

UART\_WriteString(buffer); // Enviamos la temperatura por UART

}

}

}