PROYECTO LCD 2X16 PIC18F45K22 HC-06

JAVIER ANDRES TORRES PEREZ

atorresp@ecci.edu.co

DILAN VALENCIA RAMOS
Código:126857

ENLACE YOUTUBE - VIDEO MONTAJE

https://youtu.be/KD9uuSNN12k

- **Planteamiento del Problema**

En muchos proyectos de electrónica, es necesario medir y visualizar valores de voltaje de sensores o potenciómetros.
Sin embargo, lograr esta tarea requiere una implementación eficiente que pueda convertir valores analógicos en datos digitales y mostrarlos de una manera que los usuarios puedan entender. Además, es importante configurar un sistema que permita actuar en consecuencia, como encender o apagar los LED en función de los valores obtenidos.

Objetivo

El diseño e implementación del sistema se basa en un microcontrolador PIC capaz de medir voltaje analógico a través de su ADC, calcular el valor equivalente en voltios y mostrarlo en la pantalla LCD.

Además, hay una función de aviso visual mediante LED cuando la tensión supera el límite establecido.

- **Solución propuesta**

Se ha desarrollado un sistema que tiene las siguientes características:

Controlador PIC: Responsable de la conversión de ADC y señales de control a pantallas LCD y LED.
Pantalla LCD: muestra la tensión medida en tiempo real con una precisión de dos decimales.

LED de advertencia: Indica cuando el voltaje excede el límite de 4,2 V.
Medidor de voltaje: Proporciona voltaje variable para pruebas del sistema.

Circuito eléctrico: Asegúrese de que los componentes funcionen de manera estable.

- **CÓDIGO PROGRAMADO EN MPLAB Y SIMULADO EN PROTEUS**

```
#include "adc.h"

void ADC_Init(unsigned char p_ang)
{
   ADCON1 = p_ang;
   ADCON0 = 0x00;
```

```
    ADCON2 = 0x8F;
}

unsigned int ADC_Read(unsigned char ch)
{
    if(ch > 13){
        return 0;
    }else{
        ADCON0 = 0x00;
        ADCON0 = (ch << 2);
        ADCON0bits.ADON = 1;
        ADCON0bits.GO_DONE = 1;
        while(ADCON0bits.GO_DONE == 1);
        return ADRES;
    }
}
```

```
// CONFIG1H
#pragma config FOSC = XT        // Oscillator Selection bits (XT oscillator)
#pragma config PLLCFG = OFF     // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON    // Primary clock enable bit (Primary clock is always
enabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock
Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator
Switchover mode disabled)

// CONFIG2L
#pragma config PWRTEN = OFF     // Power-up Timer Enable bit (Power up timer disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset
enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 190       // Brown Out Reset Voltage bits (VBOR set to 1.90 V
nominal)

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is
always disabled. SWDTEN has no effect.)
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
```

```
#pragma config CCP2MX = PORTC1  // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = ON      // PORTB A/D Enable bit (PORTB<5:0> pins are configured as analog input channels on Reset)
#pragma config CCP3MX = PORTB5  // P3A/CCP3 Mux bit (P3A/CCP3 input/output is multiplexed with RB5)
#pragma config HFOFST = ON      // HFINTOSC Fast Start-up (HFINTOSC output and ready status are not delayed by the oscillator stable status)
#pragma config T3CMX = PORTC0   // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTD2   // ECCP2 B output mux bit (P2B is on RD2)
#pragma config MCLRE = EXTMCLR  // MCLR Pin Enable bit (MCLR pin enabled, RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = ON         // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if MCLRE is also 1)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection Block 0 (Block 0 (000800-001FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection Block 1 (Block 1 (002000-003FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection Block 2 (Block 2 (004000-005FFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection Block 3 (Block 3 (006000-007FFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection Block 0 (Block 0 (000800-001FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (002000-003FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection Block 2 (Block 2 (004000-005FFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection Block 3 (Block 3 (006000-007FFFh) not write-protected)

// CONFIG6H
```

```
#pragma config WRTC = OFF       // Configuration Register Write Protection bit
(Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block
(000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not
write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection Block 0 (Block 0
(000800-001FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection Block 1 (Block 1
(002000-003FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF      // Table Read Protection Block 2 (Block 2
(004000-005FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Table Read Protection Block 3 (Block 3
(006000-007FFFh) not protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block
(000000-0007FFh) not protected from table reads executed in other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.


#define _XTAL_FREQ 4000000
#include <xc.h>
#include <stdio.h>

#include "adc.h"                    // Libreria para el manejo del ADC
#include "lcd.h"                    // Libreria para el control de la pantalla LCD

char buffer[20];
int valor_adc;
float voltaje;

void main()
{
   ADC_Init(AN0_ANALOG);            // Inicializa el ADC
   Lcd_Init();                      // Inicializa la pantalla LCD
   TRISBbits.RB0 = 0;
   LATBbits.LB0 = 0;

   while(1)
   {
      valor_adc = ADC_Read(0);
      voltaje = (float)(valor_adc * 5.0) / 1023;
```

```
        Lcd_Set_Cursor(1,1);
        sprintf(buffer, "voltaje es: %0.2f", voltaje);
        Lcd_Write_String(buffer);


        if(voltaje > 4.2)
        {
           LATBbits.LB0 = 1;
        }
        else
        {
           LATBbits.LB0 = 0;
        }
        __delay_ms(150);




    }
}
```

```
#include "lcd.h"

void Lcd_Port(char a)
{
   (a & 1) ? (D4 = 1) : (D4 = 0);
   (a & 2) ? (D5 = 1) : (D5 = 0);
   (a & 4) ? (D6 = 1) : (D6 = 0);
   (a & 8) ? (D7 = 1) : (D7 = 0);
}

void Lcd_Cmd(char a)
{
   RS = 0;
   Lcd_Port(a);
   EN = 1;
   __delay_ms(4);
   EN = 0;
}

void Lcd_Clear(void)
{
   Lcd_Cmd(0);
```

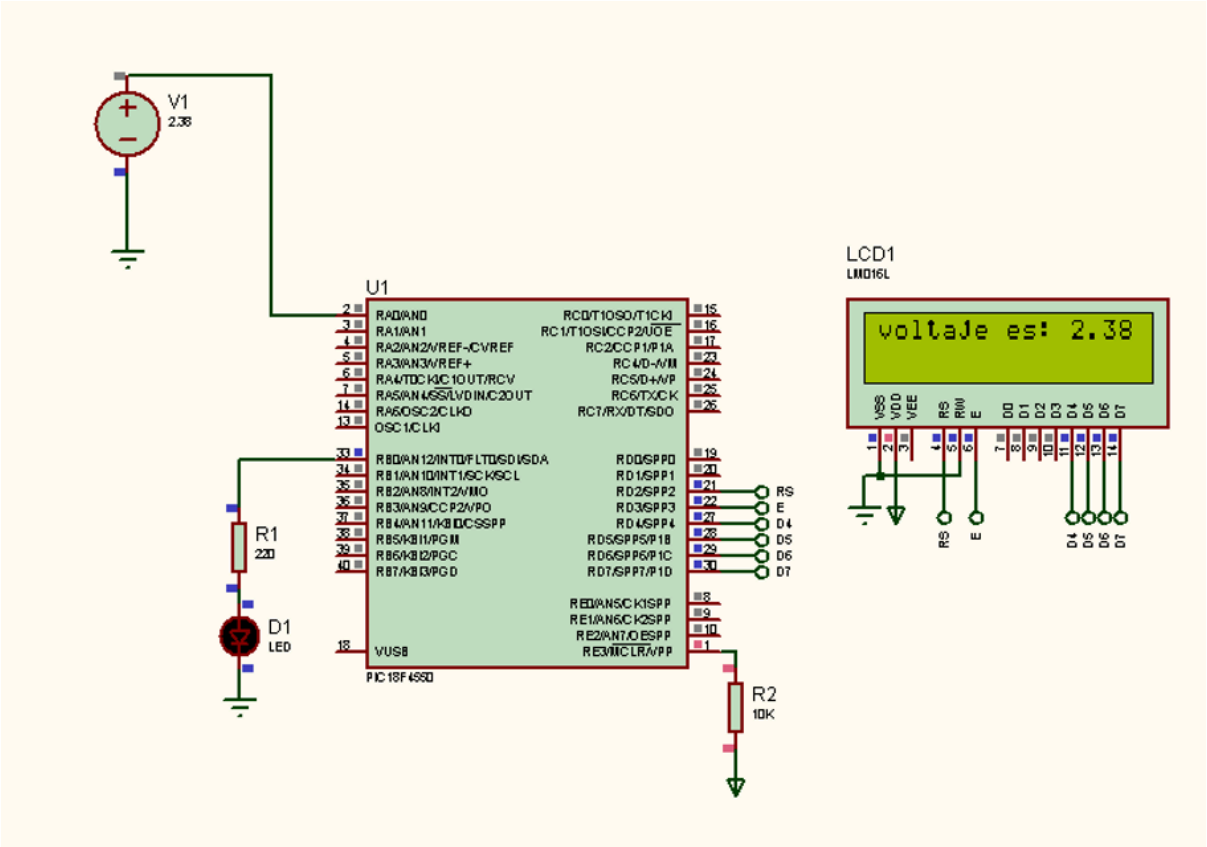
```
    Lcd_Cmd(1);
}

void Lcd_Set_Cursor(char a, char b)
{
    char temp,z,y;
    if(a == 1)
    {
        temp = 0x80 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
    else if(a == 2)
    {
        temp = 0xC0 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
    else if(a == 3)
    {
        temp = 0x94 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
    else if(a == 4)
    {
        temp = 0xD4 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
}

void Lcd_Init(void)
{
RS_DIR = 0;
EN_DIR = 0;
D4_DIR = 0;
D5_DIR = 0;
D6_DIR = 0;
D7_DIR = 0;
```

```
    Lcd_Port(0x00);
    __delay_ms(20);
    Lcd_Cmd(0x03);
    __delay_ms(5);
    Lcd_Cmd(0x03);
    __delay_ms(11);
    Lcd_Cmd(0x03);
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x08);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x0C);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x06);
    Lcd_Clear();
}

void Lcd_Write_Char(char a)
{
    char temp,y;
    temp = a&0x0F;
    y = a&0xF0;
    RS = 1;
    Lcd_Port(y>>4);
    EN = 1;
    __delay_us(40);
    EN = 0;
    Lcd_Port(temp);
    EN = 1;
    __delay_us(40);
    EN = 0;
}

void Lcd_Write_String(const char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
        Lcd_Write_Char(a[i]);
}

void Lcd_Shift_Right(void)
{
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x0C);
}

void Lcd_Shift_Left(void)
{
```

```
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x08);
}

void Lcd_Blink(void)
{
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x0F);
}

void Lcd_NoBlink(void)
{
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x0C);
}

#ifdef USE_CGRAM_LCD
void Lcd_CGRAM_CreateChar(char add, const char* chardata)
{
    switch(add)
    {
        case 0:
            for(char i=0; i<=7; i++)
                Lcd_Write_Char(chardata[i]);
            break;
        case 1:
            for(char i=8; i<=15; i++)
                Lcd_Write_Char(chardata[i-8]);
            break;
        case 2:
            for(char i=16; i<=23; i++)
                Lcd_Write_Char(chardata[i-16]);
            break;
        case 3:
            for(char i=24; i<=31; i++)
                Lcd_Write_Char(chardata[i-24]);
            break;
        case 4:
            for(char i=32; i<=39; i++)
                Lcd_Write_Char(chardata[i-32]);
            break;
        case 5:
            for(char i=40; i<=47; i++)
                Lcd_Write_Char(chardata[i-40]);
            break;
        case 6:
            for(char i=48; i<=55; i++)
                Lcd_Write_Char(chardata[i-48]);
```

```
            break;
        case 7:
            for(char i=56; i<=63; i++)
                Lcd_Write_Char(chardata[i-56]);
            break;
    }
}

void Lcd_CGRAM_Init(void)
{
    Lcd_Cmd(0x04);
    Lcd_Cmd(0x00);
}

void Lcd_CGRAM_Close(void)
{
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x02);
}
#endif
```

SIMULACION EN PROTEUS



Montaje en físico