Universidad del Valle de Guatemala — Natalia de León Bercián

Electrónica Digital 2 — carne: 18193

enero 2021 — sección: 20

Laboratorio # 3

LCD

Link video

https://youtu.be/bjni2ST5BM4

Link Github

https://github.com/nataliadlb/LABS_REPOSITORIO.git

Pseudocodigo

```
/*
 * Pseudocodigo -- Laboratorio # 3
 * Author: Natalia de León Bercián
 * carné: 18193
 * Digital 2
 *
 * Created on 7 de febrero de 2021
 */


//******************************************************************************//
//IMPORTAR LIBRERIAS                                    //
//******************************************************************************//
#include <xc.h>
#include <stdint.h>
#include <pic16f887.h>
#include "Oscilador.h"
#include "LCD.h"
#include "Config_ADC.h"
```

```
//*****************************************************************************//
//CONFIGURACION BITS                                                           //
//*****************************************************************************//
// CONFIG1
#pragma config FOSC = XT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = OFF     // RE3/MCLR pin function select bit (RE3/MCLR pin function is digital input, MCLR internally tied to VDD)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF       // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = OFF      // Brown Out Reset Selection bits (BOR disabled)
#pragma config IESO = OFF        // Internal External Switchover bit (Internal/External Switchover mode is disabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)
#pragma config LVP = OFF        // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must be used for programming)

// CONFIG2
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)

//*****************************************************************************//
//DEFINE                                                                       //
//*****************************************************************************//
```

```
#define _XTAL_FREQ 8000000

#define RS PORTEbits.RE0
#define RW PORTEbits.RE1
#define EN PORTEbits.RE2
#define D0 PORTDbits.RD0
#define D1 PORTDbits.RD1
#define D2 PORTDbits.RD2
#define D3 PORTDbits.RD3
#define D4 PORTDbits.RD4
#define D5 PORTDbits.RD5
#define D6 PORTDbits.RD6
#define D7 PORTDbits.RD7

//*****************************************************************************//
//VARIABLES                                                                    //
//*****************************************************************************//
int ADC_VALOR_1;
int ADC_VALOR_2;
unsigned int a;
float S1_val;
float S2_val;
uint8_t S3_cont;
unsigned int x;

//*****************************************************************************//
//PROTOTIPOS DE FUNCIONES                                                      //
//*****************************************************************************//
void setup(void);
```

```
void Config_INTERRUPT(void);
float bin_to_float(uint8_t ADC_VAL); //funcion para convertir el valor de la
                        //conversion ADC en decimales
void USART_Init_transmission(void); // Config Trasmision //char y const long int
void USART_Init_reception(void);// Config recepcion de datos
void Trasmission(void);// funcion para constantemente mandar los valores ADC
void Receive(void); //funcion para constantemente recibir datos de la compu
//*****************************************************************************//
//INTERRUPCIONES                                        //
//*****************************************************************************//
void __interrupt() ISR(void) {

  // ---- Interrupción del ADC ----
  if (PIR1bits.ADIF) {
    PIR1bits.ADIF = 0;
    ADC_Config (0);
    __delay_ms(2); //Inicio de conversion ADC
    ADCON0bits.GO = 1;
    while (ADCON0bits.GO != 0) { //Mientras no se haya termindo una convers.
      ADC_VALOR_1 = ADC(ADRESL, ADRESH);

    }

    ADC_Config (1);
    __delay_ms(2); //Inicio de conversion ADC
    ADCON0bits.GO = 1;
    while (ADCON0bits.GO != 0) { //Mientras no se haya termindo una convers.
      ADC_VALOR_2 = ADC(ADRESL, ADRESH);
    }
```

```
      PORTB = ADC_VALOR_2;
   }

}

//*****************************************************************************//
//PROGRAMACION PRINCIPAL                                                       //
//*****************************************************************************//
void main(void) {
   setup(); //Configuracion de puertos de entrada y salida
   Config_INTERRUPT(); //Configuracion de la interrupcion del puerto B
   Lcd_Init();
   USART_Init_transmission();
   USART_Init_reception();

   //*****************************************************************************//
   //LOOP PRINCIPAL                                                               //
   //*****************************************************************************//
   while (1) {
      S1_val = bin_to_float(ADC_VALOR_1);
      S2_val = bin_to_float(ADC_VALOR_2);
      //nombres S1, S2 y S3
      Lcd_Clear();
      Lcd_Set_Cursor(1,2);
      Lcd_Write_String("S1:");
      Lcd_Set_Cursor(1,8);
      Lcd_Write_String("S2:");
      Lcd_Set_Cursor(1,13);
      Lcd_Write_String("S3:");
```

```
      //Valores de S1 y S2
      Lcd_Set_Cursor(2,1);
      Lcd_Write_Char(S1_val);
      Lcd_Set_Cursor(2,7);
      Lcd_Write_Char(S2_val);
      Lcd_Set_Cursor(2,13);
      Lcd_Write_Char(S3_cont);
      __delay_ms(2000);

   }
   return;
}

//*****************************************************************************//
//FUNCIONES                                                    //
//*****************************************************************************//
float bin_to_float(uint8_t ADC_VAL){
//convertir cada valor de la conversion ADC (cada POT) y parasrlo a decimal para
   //poder desplegarlo en la LCD
}

void Trasmission(void){
   //trasmitir los valores de la conversion ADC a la computadora
}

void Receive(void){
   //Recibir el valor del contador, cada vez que se presione + o -
}
```

```
//********************* CONFIGURACION PRINCIPAL ****************************//

void setup(void) { //Configuración de puertos de entrada y salida
   initOsc(0b00000111); //8MHz
   ANSEL = 0b00000011; //RA0 y RA1 como analogico
   ANSELH = 0;
   TRISA = 0b00000011; //potenciometros, como entrada
   PORTA = 0;
   PORTB = 0;
   PORTC = 0;
   TRISB = 0;
   TRISC = 0;
   TRISD = 0;
   PORTD = 0;
   TRISE = 0;
   PORTE = 0;
}

//**************** **** CONFIGURACION INTERRUPCIONES ************************//

void Config_INTERRUPT(void) {
   INTCON = 0b11000000;
   PIE1bits.ADIE = 1; // enables ADC interrupt
   PIR1bits.ADIF = 1;
}

//********************* CONFIGURACION COM SERIAL ****************************//
void USART_Init_transmission(void){
   BRGH = 1;
```

```
    TXEN = 1;
    SYNC = 0;
    SPEN = 1;
}

void USART_Init_reception(void){
    SPEN =1;
    CREN =1;
    SREN = 1;

}
```

Librería de 8 bits

```
/*
 * File: LCD.h
 *Se adaptó para 8 bits
 *
 * Comentarios
 * Se utilizó y se adaptaron las librerías de Ligo George
 * de la página www.electrosome.com
 * Enlace: https://electrosome.com/lcd-pic-mplab-xc8/
 */

#include "LCD.h"

//------ Función de inicio ------//
void Lcd_Init(){
  Lcd_Port(0x00);
    __delay_ms(20);
```

```
  Lcd_Cmd(0x030);
    __delay_ms(5);
  Lcd_Cmd(0x030);
    __delay_us(160);
  Lcd_Cmd(0x030);
    __delay_us(160);
    // o busy flag...?
  ////////////////////////////////////////////////
  Lcd_Cmd(0x030); // CREO
  Lcd_Cmd(0x008);
  Lcd_Cmd(0x001);
  Lcd_Cmd(0x00);
  Lcd_Cmd(0x006);
}

//------ Función para escoger el puerto ------//
void Lcd_Port(char a){
   PORTD = a;

}

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

```
//------ Función para limpiar el visualizador ------//
void Lcd_Clear(void){
        Lcd_Cmd(0);
        Lcd_Cmd(1);
}

//------ Función para posicionar el cursor ------//
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);
        }
}

//------ Función para mandar los caracteres al puerto ------//
void Lcd_Write_Char(char a){
   Lcd_Port(a);
   //maybeeeee....
```

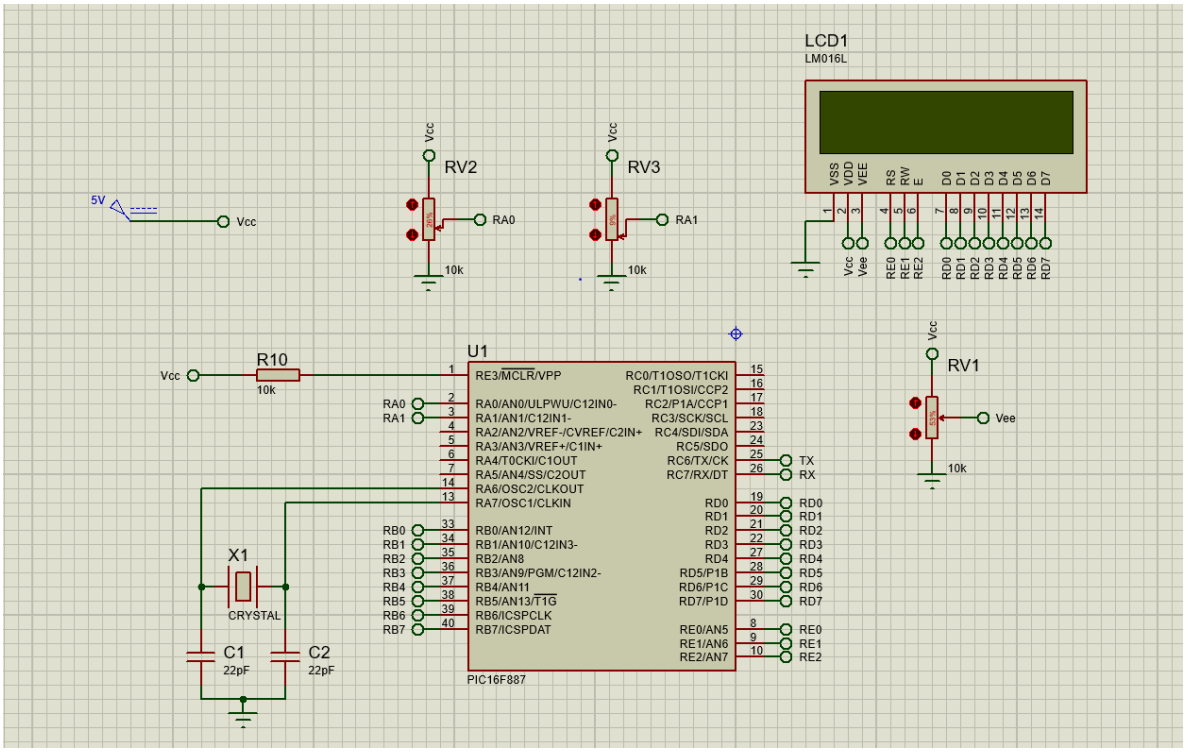
```
//   char temp,y;
//   temp = a & 0x0F;
//   y = a & 0xF0;
//   RS = 1;           // => RS = 1
//   Lcd_Port(y>>4);           //Data transfer
//   EN = 1;
//   __delay_us(40);
//   EN = 0;
//   Lcd_Port(temp);
//   EN = 1;
//   __delay_us(40);
//   EN = 0;
}

//------ Función para mandar strings al puerto ------//
void Lcd_Write_String(char *a){
        int i;
        for(i=0;a[i]!='\0';i++)
          Lcd_Write_Char(a[i]);
}

//------ Función para mover a la derecha ------//
void Lcd_Shift_Right(){
        Lcd_Cmd(0x01);
        Lcd_Cmd(0x0C);
}

//------ Función para mover a la izquierda ------//
```

```
void Lcd_Shift_Left(){
        Lcd_Cmd(0x01);
        Lcd_Cmd(0x08);
}
```



Código

```
/*
 * Código -- Laboratorio # 3
 * Author: Natalia de León Bercián
 * carné: 18193
 * Digital 2
 *
 * Created on 7 de febrero de 2021
 */
```

```
//*****************************************************************************
//Librerias
//*****************************************************************************
#include <xc.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <pic16f887.h>
#include "LCD.h"
#include "Config_ADC.h"
#include "USART.h"
#include "Oscilador.h"

// CONFIG1
#pragma config FOSC = HS          // Oscillator Selection bits (XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = OFF        // RE3/MCLR pin function select bit (RE3/MCLR pin function is MCLR)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF      // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = OFF      // Brown Out Reset Selection bits (BOR disabled)
#pragma config IESO = OFF         // Internal External Switchover bit (Internal/External Switchover mode is disabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)
#pragma config LVP = OFF        // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must be used for programming)
```

```
// CONFIG2
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)

//******************************************************************************
//Variables
//******************************************************************************
#define _XTAL_FREQ 8000000

#define RS PORTEbits.RE0
#define RW PORTEbits.RE1
#define EN PORTEbits.RE2
#define D0 PORTDbits.RD0
#define D1 PORTDbits.RD1
#define D2 PORTDbits.RD2
#define D3 PORTDbits.RD3
#define D4 PORTDbits.RD4
#define D5 PORTDbits.RD5
#define D6 PORTDbits.RD6
#define D7 PORTDbits.RD7

//******************************************************************************//
//VARIABLES                                                   //
//******************************************************************************//

float S1_val = 0.0;
float S2_val = 0.0;
char data_total[20];
```

```
uint8_t cont;
char data_recive;

//*****************************************************************************
//Prototipos de funciones
//*****************************************************************************

void setup(void);
void ADC_channel1(void);
void ADC_channel2(void);
void ADC_to_string(void);
void Show_val_LCD(void);


//*****************************************************************************
//Interrupciones
//*****************************************************************************

void __interrupt() ISR(void) {
   if(PIR1bits.RCIF == 1){
      data_recive = RCREG; //Recibe los datos que manda la terminal
      if (data_recive == '+'){ //auementa
         cont++;
         PORTB = cont;
      }
      else if (data_recive == '-'){ //decrementa
         cont--;
         PORTB = cont;
      }
```

```
        data_recive = 0;
        }
}

//*****************************************************************************
//Ciclo Principal
//*****************************************************************************

void main(void) {
    setup();
    TRISD = 0x00;
    Lcd_Init();
    Lcd_Clear();



    while (1) {
        ADC_channel1(); //conversion ADC de un pot
        __delay_ms(1);
        ADC_channel2(); //conversion ADC del otro pot


        Write_USART_String("S1   S2   S3 \n"); //enviar los datos del pic a la compu
        ADC_to_string();
        //sprintf(data_total, "%1.2fV   %1.1fV     %d", S2_val, S1_val, cont); //convertir los valores de voltaje y el contador a un string para que los lea bien la compu
        Write_USART_String(data_total); //enviar el string con los valores a la pc
        Write_USART(13);//13 y 10 la secuencia es para dar un salto de linea
        Write_USART(10);


        Show_val_LCD();
```

```
        __delay_ms(500);


    }
}

//*****************************************************************************
//Funciones
//*****************************************************************************
void ADC_to_string(void){
    sprintf(data_total, "%1.2fV %1.2fV  %d", S2_val, S1_val, cont);
}

void Show_val_LCD(void){
    //Valores de S1 y S2
    Lcd_Clear();
    Lcd_Set_Cursor(1,2); //nombres S1, S2 y S3
    Lcd_Write_String("S1:   S2:   S3:");
    Lcd_Set_Cursor(2,1);
    Lcd_Write_String(data_total);
}

void ADC_channel1(void) {
    ADC_Config (0); //channel 0
    __delay_us(40);
    ADCON0bits.GO = 1; //Inicia la conversión
    while (ADCON0bits.GO != 0) { //Mientras no se termine la conversion
        S1_val = ((ADRESH * 5.0) / 255);


    }
```

```
}

void ADC_channel2(void) {
   ADC_Config (1); //channel 1
   __delay_us(40); //Para conversion
   ADCON0bits.GO = 1; //Inicia la conversión ADC
   while (ADCON0bits.GO != 0) { //Mientras no se termine la conversion
      S2_val = ((ADRESH * 5.0) / 255);

   }
}

void setup(void) {
   initOsc(7); //8MHz
   ANSEL = 0b00000011; //RA0 y RA1 como analogico
   ANSELH = 0;
   TRISA = 0b00000011; //potenciometros, como entrada
   TRISB = 0;
   TRISCbits.TRISC6 = 0;
   TRISCbits.TRISC7 = 1;
   TRISD = 0;
   TRISE = 0;
   PORTA = 0;
   PORTB = 0;
   PORTC = 0;
   PORTD = 0;
   PORTE = 0;
   USART_Init_BaudRate();
   USART_Init();
```

```
    USART_INTERRUPT();


}
```