UNIVERSIDAD DE GUANAJUATO
Campus Irapuato - Salamanca

DIVISION DE INGENIERIAS CAMPUS IRAPUATO-SALAMANCA
DICIS

Microprocesadores y Microcontroladores.

Reporte de proyecto final.

Dr. Gustavo Cerda Villafaña.

Castellano Ornelas Victor Manuel

Actividad: Reporte y descripción de código.

Fecha: 20/05/2021

Control de motor DC con pantalla LCD y sensado con monitoreo por medio de aplicación en Android OS mediante PIC18F45K50.

Microprocesadores y Microcontroladores

20/05/2021

Castellano Ornelas Victor Manuel

vm.castellanoornelas@ugto.mx

Objetivo.

En el siguiente reporte se presenta la parte teórica y de programación necesaria para el prototipo funcional del modelo de control que se elaboró con la ayuda de un microcontrolador PIC18F45K50. Se pretende mostrar, listar, y explicar los aspectos más importantes del proyecto.

Descripción.

Se trata de un sistema cuyo objetivo primordial es el de controlar y monitorear un pequeño motor CD de 5V. como resulta evidente, tanto el control como el monitoreo serán tareas procesadas por el microcontrolador, que para el sistema en cuestión es un PIC18F45K50.

I. Aplicación y comunicación.

La aplicación fue desarrollada en una metodología por bloques, mediante la aplicación MIT APP Inventor.

La finalidad de la aplicación es establecer el protocolo de comunicación RS232, que es un protocolo de comunicación asíncrona, el concepto de comunicación serial permite la transmisión/recepción de datos.

Se encarga de conectar el teléfono celular con el que se esté trabajando y el módulo BLUETOOTH HC-05, por medio de los registros BAUDCON1, SPBRG, TXSTA1, SPBRGH1, RCSTA1. Con lo cual se contará con una comunicación entre ambos dispositivos, para, por medio de la interfaz de la aplicación; encender o apagar el motor y controlar la temperatura será posible y de fácil acceso para el usuario promedio.

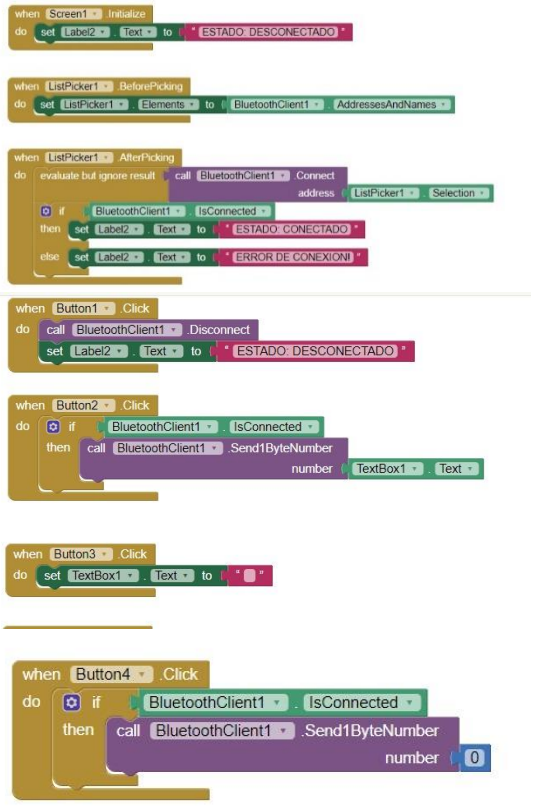

Imagen 1. Programación en bloques de la app para Android OS.

Imagen 2. Interfaz gráfica de la aplicación.

II. Control.

El primer componente para tomar en cuenta, sabiendo que se ha establecido la comunicación entre la aplicación de Android OS mostrada en el punto anterior y el microcontrolador (por medio del HC-05), es el motor en sí; lo que realmente pasa al modular la potencia del motor por medio de los valores que colocamos en nuestra aplicación es que se modula el ancho de pulso.

El PWM es una técnica que consiste en variar el ancho de pulso de una señal de voltaje con el objetivo de controlar la cantidad de potencia administrada a los componentes o elementos electrónicos conectados; en este caso el motor.

III. Monitoreo.

Una pantalla LCD 16x2 mostrará la potencia que le hayamos indicado al motor por medio del proceso anterior, además de la temperatura a la cual se encuentre el campo de operación de mismo, esta información será entregada por un sensor de temperatura LM35 conectado a una entrada analógica configurada del PIC18F45K50, con lo cual, se podrá saber en todo momento y con certeza las características de operación del sistema.

IV. Rutinas de paro.

Para hacer más eficiente el sistema se han implementado un par de interrupciones para que el sistema se detenga y no vuelva a operar hasta que un operario reinicie el sistema. La interrupción de alta prioridad que aquí se programó básicamente se activa cuando la temperatura supera los 50°C o bien si se oprime el botón pulsable que se estableció como paro de emergencias.

A continuación, se muestra el diagrama general de interconexión de los componentes:

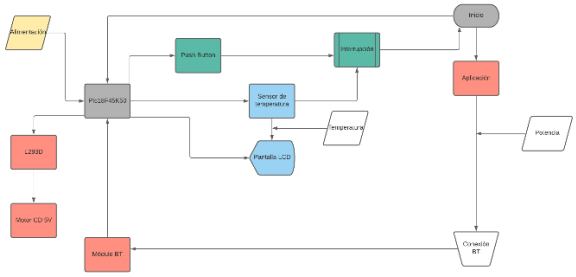

Diagrama 1. Conexiones modulares.

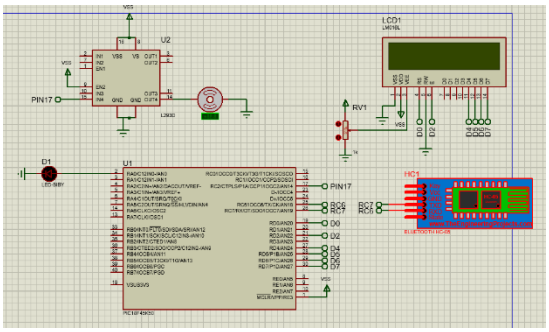

Imagen 3. Diagrama de conexiones del sistema.

Lista de componentes.

Los materiales necesarios para la realización del proyecto son:

- Plantilla de experimentos
- Microcontrolador PIC18F45K50
- Programador PIC-KIT3
- Pantalla LCD 16x2
- Teléfono celular Android OS
- Modulo Bluetooth HC-06
- Sensor de temperatura LM35
- 1 Push-button

- Driver L293D
- Motor de DC 5V
- Fuente de alimentación de 5V
- Capacitor 0.1 uF

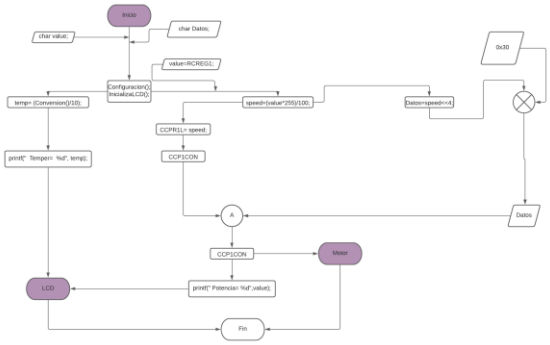

Diagrama 3. Funciones del sistema

Diagramas de flujo.

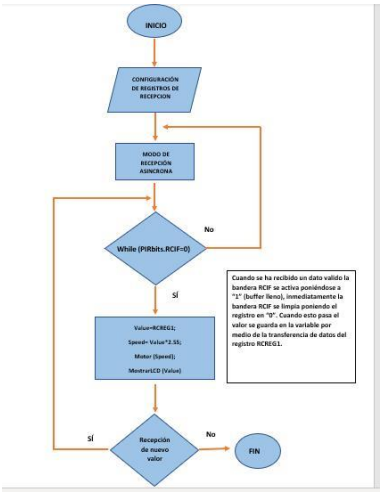

Diagrama 2. Comunicación del sistema.

Código utilizado.

```
#pragma config FOSC = INTOSCIO // Oscillator Selection (Internal oscillator)
#pragma config WDTEN = OFF // Watchdog Timer Enable bits (WDT disabled in hardware (SWDTEN ignored))
#pragma config MCLRE = ON // Master Clear Reset Pin Enable (MCLR pin enabled; RE3 input disabled)
#pragma config LVP = OFF // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config ICPRT = OFF // Dedicated In-Circuit Debug/Programming Port Enable (ICPORT disabled)
#include <xc.h>
#include <stdio.h>
#define _XTAL_FREQ 1000000
```

Se establecen las condiciones necesarias paramétricas para la operación en el entorno de programación para el compilador xc8, así como la llamada de las librerías necesarias así como la frecuencia de operación del sistema.

```
void Configuracion(void)
{
//Entradas
TRISD=0;
TRISB=0xFF;
ANSELD=0;
ANSELB=0;
LATBbits.LB0 = 0;
//comunicacion bluetooth
    ANSELC=0;
    TRISC=0xC0;

    BAUDCON1 = 0x08;
    SPBRG1 = 25;
    TXSTA1 = 0x24;
    SPBRGH1 = 0x00;
    RCSTA1 = 0x90;
//sensor de temperatura
    ADCON0 = 0x05;
    ADCON1 = 0x08;
    ADCON2 = 0xAC;
    VREFCON0=0x90;//Activa el voltaje de referencia


//Motor
 T2CON=0x06; //Postscaler 1:1, prescaler 1:16
 CCP1CON=0x0C; // Selecciona modo PWM
 CCPR1L=0x0B; // Carga CCPR1L con un valor arbitrario 0000 1011

//Habilitación de Interrupcion
 INTCON=0xD0; // Habilita interrupcion externa RB0 // 1101 0000
 INTCON3=0xC8; // Habilita interrupción externa RB1, Alta prioridad INT1-INT2
 RCONbits.IPEN =1; // Habilita niveles de interrupción

}
```

Como resulta habitual, en esta icónica función se establecen los registros necesarios, como podemos observar es una función grande debido a la gran cantidad de registros que se utilizan en este proyecto, ya que prácticamente se abarcan todos los temas vistos en el curso de MyM.

La mayoría de los registros fueron llenados de acuerdo a la hoja de especificaciones del PIC18F45K50, y en los comentarios de las capturas de puede observar la división de cada uno de ellos.

```
void putcm(char data) {
    char Activa;
    Activa = data & 0xF0;
    LATD = Activa | 0x04;
    __delay_us(10);
    LATD = Activa;
    __delay_ms(1);
    Activa = data << 4;
    LATD = Activa | 0x04;
    __delay_us(10);
    LATD = Activa;
}
```

Una función prácticamente “predefinida” que será encargada de controlar la pantalla LCD para que realice diversas tareas tales como posicionar el cursor, imprimir, limpiar, etc. Por medio de comando espeficos.

```
void InicializaLCD(void)
{
__delay_ms(30);
putcm(0x02);    // Inicializa en modo 4 bits
__delay_ms(1);
//__delay_ms(1);
putcm(0x28);    // Inicializa en 2 líneas 5x7
__delay_ms(1);
//__delay_ms(30);
putcm(0x2C);
__delay_ms(1);
putcm(0x0C);
__delay_ms(1);
putcm(0x06);
__delay_ms(1);
putcm(0x80); //Posiciona el curson en 1,1
__delay_ms(1);
putcm(0x18);
__delay_ms(1);
}
```

Esta función hace operable la pantalla LCD para los fines propuestos, utiliza la función putcm() con sus diversas funciones para acomodar la pantalla; a 4 bits, establecer líneas y posicionar el cursos, básicamente consiste en una reiteración de la función putcm().

```
void __interrupt(high_priority) myHiIsr(void)
{
    int vel=0x00;
    char Datos=0;
   if((PORTB==0x01)||(temp>=50))
   {
       char A=5;

       while(A>0)
       {
       putcm(0x80);
       printf(" EMERGENCIA%s");
       __delay_ms(500);
       A -= 1;
        CCPR1L= vel;
        Datos=vel<<4;
        Datos=Datos&0x30; //Enmascara todos los bits menos 5:4
        CCP1CON = CCP1CON | Datos;
       }
   putcm(0x01);
   INTCONbits.INT0IF=0;
   }
}
```

Esta interrupción de alta prioridad se activará cuando se presione el botón pulsable del pin 1 del puerto B (paro de emergencia) o cuando la temperatura supere los 50°C (suponiendo que es un rango de operación), desplegará un mensaje de “Emergencia” en la pantalla por un momento e inmediatamente dejará de operar el motor, pues se le envía un 0 de potencia.

```
int Conversion(void) {

ADCON0bits.GO = 1;
while (ADCON0bits.GO);
return  ADRESL + ADRESH*256; // Retorna los 10 bits como int justificado a la derecha }
}
```

Realiza la conversión de los valores analógicos captados por el sensor de temperatura a binario, con 10 bits de resolución.

```
void main(void)
{
char Datos = 0;
char value;
int auxt;

Configuracion();
InicializaLCD();
__delay_ms(10);

while (1)
{
//putcm(0x01);
temp= (Conversion()/10);
putcm(0x80);
printf("  Temper=  %d", temp);
while(PIR1bits.RC1IF==0);
{
    value=RCREG1;
    speed=(value*255)/100;


    CCPR1L= speed;
    Datos=speed<<4;
    Datos=Datos&0x30; //Enmascara todos los bits menos 5:4
    CCP1CON = CCP1CON | Datos;
}
    putcm(0xC2);
    printf(" Potencia= %d",value);
__delay_ms(1000);
putcm(0x01);
}
return;
}
```

Se trata de la función más compleja del proyecto, la función principal donde se opera finalmente el sistema.

Primeramente, se inicializan variables auxiliares necesarias para operaciones posteriores y se mandan llamar funciones anteriores necesarias.

Lo primero en operarse es la temperatura, cuyo valor análogo obtenido será digitalizado y pasado por el factor de conversión de acuerdo con los cálculos de la resolución del sensor LM35 (1/10), para posteriormente ser impreso, esto se lleva a cabo dentro del bucle infinito que es la operación maestra del proceso.

Posteriormente se tiene otro bucle, con el que se opera la velocidad del motor donde la variable Value es el valor que el usuario ingresa en la aplicación, donde el porcentaje PWM se establece con la siguiente relación (255) /100.

Finalmente sale de los ciclos, tiene un retardo, y termina el programa.

Conclusiones.

Sin duda se trató de un proyecto tanto retador como enriquecedor en muchos aspectos, pues el conocimiento adquirido al realizar este proyecto fue basto al abarcarse varios temas y componentes vistos en el curso, además de reafirmarse conocimiento de otras áreas como la electrónica y la programación.

Hablando de manera más puntual, la implementación e integración sobre todo del sensor de temperatura teniendo un motor fue un verdadero reto, así como el haber trabajado de manera remota y modular, lo cual sin duda complicó en demasía la integración del código, y ni hablar de la implementación del prototipo físico, que acabó en mermas impresionantes de material y mucho tiempo probando e implementando.

En la parte técnica, haber descifrado los registros para cada función y componente a utilizar fue de verdad interesante, pues escapa un poco de los paradigmas tradicionales de programación que conozco, con lo cual era bastante interesante ver como por un registro mal configurado se podía dañar por completo el performance del programa.

En genera considero que se cumplió el objetivo, pues fue enriquecedor en la parte técnica, fortaleció la resiliencia y tuve un acercamiento al trabajo remoto y en equipo, además de lo satisfactorio que resulta programar algo y tener la posibilidad de verlo funcionando físicamente.

Vídeo del funcionamiento del prototipo:

https://drive.google.com/file/d/1ZqIzvWk7A6MnpSzI-WfYUGkOIHAiB0iM/view?usp=sharing