UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO FINAL DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DESARROLLO DE UNA UNIDAD DE ADQUISICIÓN Y MONITORIZACIÓN DE DATOS PARA UN VEHÍCULO

Programa

Autor: Aitor Andrés López

Director: Manuel Andrés Manzanares Brotons

Convocatoria: Mayo 2019

Índice

1. Programa principal

```
// Microcontrolador utilizado PIC18F4550

#include <18F4550.h>


// Convertidor ADC resolucion 10

#device adc=10


// xt para oscilador 4MHz, hs para osc >4MHz

#fuses xt,nowdt,nolvp,noprotect,nobrownout,put,cpudiv1,pll1


// oscilador de 4M

#use delay(clock =4M)


// Declaración Pines puerto serie RS232

#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7,bits=8)


// Declaración bus I2C

#use i2c(MASTER, sda=PIN_B0, scl=PIN_B1)


// Definición PUERTOS LCD - PUERTO D

#define lcd_rs_pin      pin_D0        // pin21

#define lcd_rw_pin      pin_D1        // pin22
```

```
#define lcd_enable_pin  pin_D2        // pin23

#define lcd_data4       pin_D4        // pin25

#define lcd_data5       pin_D5        // pin26

#define lcd_data6       pin_D6        // pin27

#define lcd_data7       pin_D7        // pin28


// Definición de PUERTOS A,B,C

#use standard_io (D)

#use standard_io (B)

#use standard_io (C)


// Librerias

#include <lcd.c>              // Pantalla LCD

#include <DS1307_18F4550.c>     // Reloj

#include <math.h>             // Operaciones matemáticas básicas

#include <TEMP_DS1621.c>        // Termperatura


// Escribir EEPROM

void write_ext_eeprom(long int address, BYTE data)

{

  //short int status;

  i2c_start();                //Inicializa la transmisión
```

```
  i2c_write(0xA0);              //Escribe la palabra de control (dirección 0h + 0 para escritura)
  i2c_write(address>>8);        //Parte alta de la dirección a escribir en la eeprom.
  i2c_write(address);           //Parte baja de la dirección a escribir en la eeprom.
  i2c_write(data);              //Dato a escribir
  i2c_stop();                   //Finalización de la transmisión.
  //i2c_start();                //Reinicio
  //status=i2c_write(0xa0);     //Lectura del bit ACK, para evitar escrituras incorrectas
}


// Leer EEPROM
BYTE read_ext_eeprom(long int address) {
  BYTE data;
  i2c_start();                  //Inicializa la transmisión
  i2c_write(0xA0);              //Escribe la palabra de control (dirección 0h + 0 para escritura)
  i2c_write(address>>8);        //Parte alta de la dirección a escribir en la eeprom.
  i2c_write(address);           //Parte baja de la dirección a escribir en la eeprom.
  i2c_start();                  //Reinicio
  i2c_write(0xa1);              //Escribe la palabra de control (dirección 0h + 1 para lectura)
  data=i2c_read(0);             //lectura del dato
  i2c_stop();                   //Finalización de la transmisión.
  return(data);
}
```

```
// Variables

  // Seleccion LCD
  char    LCD;              // Variables de funciones
  char    n_LCD = 9;        // Número de pantallas

  // Timer
  int8    cont          = 0;
  int8    traspaso      = 0;

  // Memoria Externa
  int16   valor         = 2;
  int16   Write_D       = 0;
  int16   Read_D        = 0;

  // 2 Reloj
  int     day,mth,year;     // Reloj
  int     dow,hour,min,sec; // Reloj

  // 3 Temperatura
  float   Dato_temp;        // Temperatura
```

```
//int     Dato_temp_int;      // Parte entera del float


// 4 LDR
int16   Lee_adc;
float   Dato_adc;
//int     Dato_adc_int;       // Parte entera del adc
float   Set_LDR  =  3;


// 5 Velocidades
int8    pulso          = 0;
int16   vueltas        = 0;
float   rpm,velocidad,km_h= 0;
float   radio          = 0.3; //en metros


// 6 Distancia
int16   N_vueltas      = 0;
float   D_total        = 0;
float   desarrollo     = 0;


// 7  Altitud
int16   Lee_adc2;
float   Dato_adc2;
```

```
  float    Altura;


  // 8  Carga batería
  int16    Lee_adc3;
  float    Dato_adc3;
  float    Carga;


  // 9 RS232
  int      Dato_leido;


//    Interrupción timer, cíclico, EEPROM + Hall
 #int_TIMER1
void temp1s (void)
{
// Captura cíclica giros de rueda
  if(cont == 20)             // 10 = 1 segundo
  {
    //output_high(pin_c1);
    cont = 0;
    vueltas = pulso;
    pulso = 0;
```

```
  }
  else output_low(pin_c1);

// Trapaso datos a la Memoria Externa
  if(traspaso == 50)          // 10 = 1 segundo
  {
    output_high(pin_c1);
    traspaso = 0;

    Write_ext_eeprom(Write_D, day);         // 1 Dia
    Write_D++;
    delay_ms(20);
    Write_ext_eeprom(Write_D, mth);         // 2 Mes
    Write_D++;
    delay_ms(20);
    Write_ext_eeprom(Write_D, year);        // 3 Año
    Write_D++;
    delay_ms(20);
    Write_ext_eeprom(Write_D, hour);        // 4 Hora
    Write_D++;
    delay_ms(20);
    Write_ext_eeprom(Write_D, min);         // 5 Minuto
```

```
  Write_D++;
  delay_ms(20);
  Write_ext_eeprom(Write_D, sec);         // 6 Segundo
  Write_D++;
  delay_ms(20);
  Write_ext_eeprom(Write_D, Dato_temp);    // 7 Temperatura
  Write_D++;
  delay_ms(20);
  Write_ext_eeprom(Write_D, D_total);      // 8 Distancia
  Write_D++;
  delay_ms(20);
  Write_ext_eeprom(Write_D, km_h);         // 9 km/h
  Write_D++;
  delay_ms(20);
  Write_ext_eeprom(Write_D, Altura);       // 10 Altura
  Write_D++;
  delay_ms(20);
}
else output_low(pin_c1);

set_timer1(53036);          // T = 100ms
cont++;
```

```
  traspaso++;
}

//   Interrupción para Bluetooth - Smartphone
#int_rda
void rda_isr()  {

  valor = getc();

  switch (valor)
  {
      case 'B':
      output_high(pin_c0);
      printf("B");
      break;

      case 'A':
      output_low(pin_c0);
      printf("A");
      break;

      case 'T':
```

```
output_high(pin_c2);

printf("T = %4.1f C",Dato_temp);

break;


case 'P':

output_high(pin_c2);

printf("LDR   =  %4.2f", Dato_adc);

break;


case 'R':

output_high(pin_c2);

printf("%02d:%02u:%02u",hour,min,sec);

break;


case 'V':

output_high(pin_c2);

printf("%4.2f",km_h);

break;


case 'D':

output_high(pin_c2);

printf("%3.3f km.",D_total);
```

```
    break;


    case 'C':
    output_high(pin_c2);
    printf("%4.1fm. ",Altura);
    break;


    case 'E':
    output_high(pin_c2);
    printf("%4.0f ",Carga);
    break;
}


if (valor==2)
    {
    output_high(pin_c1);
    printf("2");
}


if (valor==1)
    {
    output_low(pin_c1);
```

```
      printf("1");
  }



  }

//   1. Principal       LCD = 0
void LCD_Principal(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    printf(lcd_putc, "TFG Aitor Andres");
    lcd_gotoxy(1,2);
    printf(lcd_putc, "   26/04/2019   ");
}

//   2. Reloj          LCD = 1
void Reloj(){
  ds1307_get_time(hour,min,sec);
  ds1307_get_date(day,mth,year,dow);
}
  void LCD_Reloj(){
    lcd_putc('\f');
```

```
    lcd_gotoxy(1,1);
    printf(lcd_putc, "    %02d:%02u:%02u    ",hour,min,sec);
    lcd_gotoxy(1,2);
    printf(lcd_putc, "    %02d/%02u/%02u    ",day,mth,year);
  }

//    3. Temperatura      LCD = 2
void Temperatura(){
  Dato_temp = read_full_temp(0x03);     // Lee temperatura DS1621

}
  void LCD_Temperatura(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    printf(lcd_putc,"  Temperatura");
    lcd_gotoxy(1,2);
    printf(lcd_putc,"    %4.1f C",Dato_temp);
}

//    4. LDR            LCD = 3
void LDR(){
  set_adc_channel(0);
```

```
Lee_adc    =  read_adc();
Dato_adc   =  (5.0*Lee_adc/1024.0);


if (LCD == 3)
{


  if (Dato_adc >= Set_LDR)
  {
  output_high(pin_c0);
  delay_ms(20);
  }


  if (input(PIN_B4) == 1)
  {
  output_high(pin_c0);
  Set_LDR = Set_LDR + 0.1;
  delay_ms(50);
  }
  else output_low(pin_c0);


  if (input(PIN_B3) == 1)
  {
```

```
    output_high(pin_c0);
    Set_LDR = Set_LDR - 0.1;
    delay_ms(50);
    }
  }
}
  void LCD_LDR(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    printf(lcd_putc, "Iluminacion LDR");
    lcd_gotoxy(1,2);
    printf(lcd_putc, "%4.1f  Set = %4.1f",Dato_adc,Set_LDR);
    }
//   5.Velocidad       LCD = 4
void Velocidades(){
    if (LCD == 4){
    if (input(PIN_B3) == 1){    //PIN 36
      output_high(pin_c0);
      pulso++;
      delay_ms(50);
    }
    }
```

```
    output_low(pin_c0);


    rpm        = vueltas*30;
    velocidad   = (rpm*2*pi*radio)/1000;
    km_h       = velocidad*60;
}
 void LCD_Velocidades(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    printf(lcd_putc,"Pul rpm  km/h");
    lcd_gotoxy(1,2);
    printf(lcd_putc,"\n%02u  %3.0f  %4.2f   \r",pulso,rpm,km_h);  //03Lu
    //delay_ms(50);
 }


//   6. Distancia        LCD = 5
void Distancia(){
    if (LCD == 5){
    if (input(PIN_B3) == 1){    //PIN 36
      output_high(pin_c0);
      N_vueltas++;
      delay_ms(50);
```

```
      }
    else  output_low(pin_c0);


    desarrollo = 2*pi*radio;
    D_total    = (desarrollo*N_vueltas)/1000;
    }
}
  void LCD_Distancia(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    printf(lcd_putc,"Distancia total");
    lcd_gotoxy(1,2);
    printf(lcd_putc,"\n   %3.3f km.     \r",D_total);
  }


//   7. Altitud         LCD = 6
void Altitud(){
  set_adc_channel(1);
  Lee_adc2   =   read_adc();
  Dato_adc2   =  (5.0*Lee_adc2/1024.0);


  Altura = (Dato_adc2*5750)/5;
```

```
}
  void LCD_Altitud(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    printf(lcd_putc,"    Altitud   ");
    lcd_gotoxy(1,2);
    printf(lcd_putc, "    %4.0fm.    ",Altura);
    //delay_ms(1000);

  }

//   8. Carga batería    LCD = 7
void Carga_bateria(){
  set_adc_channel(2);
  Lee_adc3   =   read_adc();
  Dato_adc3   =  (5.0*Lee_adc3/1024.0);

  Carga = (Dato_adc3*20);
}
  void LCD_Carga_bateria(){
    lcd_putc('\f');
    lcd_gotoxy(1,1);
```

```
    printf(lcd_putc,"  Carga bateria ");
    lcd_gotoxy(1,2);
    printf(lcd_putc, "     %4.0f      ",Carga);
    delay_ms(100);


  }


//   9. RS232          LCD = 8
void RS232() {
   if (LCD == 8){


     // Transferencia datos de memoria a PC
     if (input(PIN_B3) == 1)         // PIN b3...36
     {
       output_high(pin_c2);
       printf("\n \r");                          // Línea en blanco
       printf("\nDescarga de ruta \r");                 // Texto de inicio
       printf("\n \r");                          // Línea en blanco
       printf("\nFecha: %02u/%02u/%02u \r" ,day,mth,year);    // Reloj
       printf("\nHora:  %02u:%02u:%02u \r" ,hour,min,sec);    // Reloj
       printf("\n \r");                          // Línea en blanco
```

```
  for ( Read_D = 0x0000 ; Read_D <= 0x0064 ; Read_D++)
     {
     Dato_leido = READ_EXT_EEPROM(Read_D);
     delay_ms(10);

     lcd_putc('\f');
     lcd_gotoxy(1,1);
     printf(lcd_putc, "Volcando        ");
     lcd_gotoxy(1,2);
     printf(lcd_putc, "          ruta");
     printf("\n %02u \r", Dato_leido);
     }
}
else output_low(pin_c2);

// Vaciado memoria
if (input(PIN_B4) == 1)         // PIN b4...37
{
  output_high(pin_c2);
  printf("\n \r");                          // Línea en blanco

  for ( Write_D = 0x0000 ; Write_D <= 0x0064 ; Write_D++) //0x7FFF
```

```
            {
              Write_ext_eeprom(Write_D, 0x0000);
              lcd_putc('\f');
              lcd_gotoxy(1,1);
              printf(lcd_putc, "Borrando       ");
              lcd_gotoxy(1,2);
              printf(lcd_putc, "       memoria");
              printf("\nBorrando memoria...\r");
            }
       }
       else output_low(pin_c2);
    }
}
  void LCD_RS232(){
       lcd_putc('\f');
       lcd_gotoxy(1,1);
       printf(lcd_putc, " Conexion RS232 ");
       lcd_gotoxy(1,2);
       printf(lcd_putc, "Carga X  Vacia X");
  }

// Refresco valores
```

```
void Refresco_valores(){
  Reloj();
  Temperatura();
  LDR();
  Distancia();
  Velocidades();
  Altitud();
  Carga_bateria();
  RS232();
}

// Visualización LCD  0...8 = n_LCD !
void Seleccion_LCD()
    {
  switch (LCD) {

      case 0:  // Pantalla 1
        LCD_Principal();
        break;

      case 1:  // Pantalla 2
        LCD_Reloj();
```

```
  break;


case 2:  // Pantalla 3
  LCD_Temperatura();
  break;


case 3:  // Pantalla 4
  LCD_LDR();
  break;


case 4:  // Pantalla 5
  LCD_Velocidades();
  break;


case 5:  // Pantalla 6
  LCD_Distancia();
  break;


case 6:  // Pantalla 7
  LCD_Altitud();
  break;
```

```
      case 7:  // Pantalla 8
        LCD_Carga_bateria();
        break;

      case 8:  // Pantalla 8
        LCD_RS232();
        break;
           }
    }

// Navegar por pantallas LCD
void Numero_LCD(){

  if (input(PIN_B7) == 1) {       // PIN b7...40
    LCD++;                    // Si pulsa aumenta valor LCD
    delay_ms(400);      }

  if (input(PIN_B6) == 1) {       // PIN b6...39
    LCD--;                   // Si pulsa disminuye valor LCD
    delay_ms(400);      }

  if (LCD > (n_LCD-1)) {          // Si 'LCD' supera el número de 'n_LCD'
```

```
    LCD = 0;          }


    }


// Programa principal
void main ()
{
    enable_interrupts(global);
    setup_adc_ports(AN0_to_AN2);
    setup_adc(ADC_CLOCK_INTERNAL);

    enable_interrupts(int_rda);          // Habilita interrupción Bluetoth

    setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
    set_timer1(53036);                   // Carga tiempo timer1
    enable_interrupts(INT_TIMER1);       // Habilita interrupccion timer1

// LED's OFF
    output_low(pin_c0);
    output_low(pin_c1);
    output_low(pin_c2);
```

```
// Inicialización
    lcd_init();                         // LCD
    lcd_putc("\f");                     // LCD
    init_temp(0x03);                    // Tª DS1621
    ds1307_init();                      // RTC DS1307
    //ds1307_set_date_time(26,04,19,1,16,45,00);   // Carga valores + 3V

while (true)
  {
  Refresco_valores();           // Actualiza valores
  Numero_LCD();                 // Selección pantallas 'LCD'
  Seleccion_LCD();              // Relación case 0...8 con Numero_LCD();
  delay_ms(100);
  }     // Fin while
}       // Fin main
```

1.1. Librería Termómetro digital DS1621

```
void init_temp(int address) {    //address=0x03(en el ejemplo)
  i2c_start();
  //Ver pagina 8 datasheet
  i2c_write(0x90|(address<<1)); //Genera primer byte (1001A2A1A0W) "1" lectura "0" escritura
  i2c_write(0xee);          //Inicia conversion
  i2c_stop();
}



float read_full_temp(int address) {
  float tura;
  BYTE datah;
  BYTE datal;
  int data;



  i2c_start();
  i2c_write(0x90|(address<<1)); //Genera primer byte (1001A2A1A0-W)
  i2c_write(0xaa);      //Lee temperatura
  i2c_start();
  i2c_write(0x91|(address<<1)); //Genera primer byte (1001A2A1A0-R)
```

```
  datah=i2c_read();      //Lectura parte alta
  datal=i2c_read(0);     //Lectura parte alta y NACK
  i2c_stop();


  data=datah;
  if (datal==128)
    tura=data+0.5;
  else
    tura=data;
  return(tura);
}
```

1.2. Librería Reloj RTC DS1307

```
#ifndef RTC_SDA

#define RTC_SDA  PIN_B0//B0

#define RTC_SCL  PIN_B1//B1

#endif


#use i2c(master, sda=RTC_SDA, scl=RTC_SCL)


#define DS1307_ALL_DISABLED        0b00000000 // All disabled

#define DS1307_OUT_ON_DISABLED_HIHG 0b10000000 // Out to Hight on Disable Out

#define DS1307_OUT_ENABLED         0b00010000 // Out Enabled

#define DS1307_OUT_1_HZ            0b00000000 // Freq. Out to 1 Hz

#define DS1307_OUT_4_KHZ           0b00000001 // Freq. Out to 4.096 Khz

#define DS1307_OUT_8_KHZ           0b00000010 // Freq. Out to 8.192 Khz

#define DS1307_OUT_32_KHZ          0b00000011 // Freq. Out to 32.768 Khz


#define Start_user_address_nvram   0x08

#define End_user_address_nvram     0x3f


//char
days_of_week[7][11]={"Lunes\0","Martes\0","Miércoles\0","Jueves\0","Viernes\0","Sábado\0","Do
mingo\0"};
```

```
byte ds1307_bin2bcd(byte binary_value);

byte ds1307_bcd2bin(byte bcd_value);


void ds1307_init(){ //int val


  byte seconds = 0;


#ifndef USE_INTERRUPTS

  disable_interrupts(global);

#endif


  i2c_start();

  i2c_write(0xD0);

  i2c_write(0x00);

  i2c_start();

  i2c_write(0xD1);

  seconds = ds1307_bcd2bin(i2c_read(0));

  i2c_stop();

  seconds &= 0x7F;


  delay_us(3);
```

```
  i2c_start();
  i2c_write(0xD0);
  i2c_write(0x00);
  i2c_write(ds1307_bin2bcd(seconds));
  i2c_start();
  i2c_write(0xD0);
  i2c_write(0x07);
  i2c_write(0x80);   //val
  i2c_stop();

#ifndef USE_INTERRUPTS
  enable_interrupts(global);
#endif

}

void ds1307_set_date_time(byte day, byte mth, byte year, byte dow, byte hr, byte min, byte sec){

#ifndef USE_INTERRUPTS
  disable_interrupts(global);
#endif
```

```
  sec &= 0x7F;
  hr &= 0x3F;

  i2c_start();
  i2c_write(0xD0);
  i2c_write(0x00);
  i2c_write(ds1307_bin2bcd(sec));
  i2c_write(ds1307_bin2bcd(min));
  i2c_write(ds1307_bin2bcd(hr));
  i2c_write(ds1307_bin2bcd(dow));
  i2c_write(ds1307_bin2bcd(day));
  i2c_write(ds1307_bin2bcd(mth));
  i2c_write(ds1307_bin2bcd(year));
  i2c_stop();

#ifndef USE_INTERRUPTS
   enable_interrupts(global);
#endif

}

void ds1307_get_date(byte &day, byte &mth, byte &year, byte &dow){
```

```
#ifndef USE_INTERRUPTS
  disable_interrupts(global);
#endif

 i2c_start();
 i2c_write(0xD0);
 i2c_write(0x03);
 i2c_start();
 i2c_write(0xD1);
 dow  = ds1307_bcd2bin(i2c_read() & 0x7f);
 day  = ds1307_bcd2bin(i2c_read() & 0x3f);
 mth  = ds1307_bcd2bin(i2c_read() & 0x1f);
 year = ds1307_bcd2bin(i2c_read(0));
 i2c_stop();

#ifndef USE_INTERRUPTS
  enable_interrupts(global);
#endif

}
```

```
void ds1307_get_time(byte &hr, byte &min, byte &sec){


#ifndef USE_INTERRUPTS
  disable_interrupts(global);
#endif


 i2c_start();
 i2c_write(0xD0);
 i2c_write(0x00);
 i2c_start();
 i2c_write(0xD1);
 sec = ds1307_bcd2bin(i2c_read() & 0x7f);
 min = ds1307_bcd2bin(i2c_read() & 0x7f);
 hr  = ds1307_bcd2bin(i2c_read(0) & 0x3f);
 i2c_stop();


#ifndef USE_INTERRUPTS
  enable_interrupts(global);
#endif


}
```

```
char ds1307_read_nvram_byte(char addr){


  char retval;


#ifndef USE_INTERRUPTS
  disable_interrupts(global);
#endif


  i2c_start();
  i2c_write(0xD0);
  i2c_write(addr);


  i2c_start();
  i2c_write(0xD1);
  retval = i2c_read(0);
  i2c_stop();


  return(retval);


#ifndef USE_INTERRUPTS
  enable_interrupts(global);
#endif
```

```
}

void ds1307_write_nvram_byte(char addr, char value){

#ifndef USE_INTERRUPTS
  disable_interrupts(global);
#endif

  i2c_start();
  i2c_write(0xD0);
  i2c_write(addr);
  i2c_write(value);
  i2c_stop();

#ifndef USE_INTERRUPTS
  enable_interrupts(global);
#endif

}

void ds1307_get_day_of_week(char* ptr){
```

```
  byte lday;
  byte lmonth;
  byte lyr;
  byte ldow;
  ds1307_get_date(lday,lmonth,lyr,ldow);
  //sprintf(ptr,"%s",days_of_week[ldow]);
}

byte ds1307_bin2bcd(byte binary_value){

 byte temp;
 byte retval;

 temp = binary_value;
 retval = 0;
 while(true){
  if(temp >= 10){
   temp -= 10;
   retval += 0x10;
  }else{
   retval += temp;
```

```
    break;
   }
  }
  return(retval);
}

byte ds1307_bcd2bin(byte bcd_value){

  byte temp;

  temp = bcd_value;
  temp >>= 1;
  temp &= 0x78;
  return(temp + (temp >> 2) + (bcd_value & 0x0f));
}
```

1.3. Librería LCD 2x16

```
typedef struct
{                    // This structure is overlayed
  BOOLEAN enable;        // on to an I/O port to gain
  BOOLEAN rs;            // access to the LCD pins.
  BOOLEAN rw;            // The bits are allocated from
  BOOLEAN unused;        // low order up.  ENABLE will
  int     data : 4;      // be LSB pin of that port.
 #if defined(__PCD__)    // The port used will be LCD_DATA_PORT.
  int    reserved: 8;
 #endif
} LCD_PIN_MAP;


// this is to improve compatability with previous LCD drivers that accepted
// a define labeled 'use_portb_lcd' that configured the LCD onto port B.
#if ((defined(use_portb_lcd)) && (use_portb_lcd==TRUE))
 #define LCD_DATA_PORT getenv("SFR:PORTB")
#endif


#if defined(__PCB__)
  // these definitions only need to be modified for baseline PICs.
  // all other PICs use LCD_PIN_MAP or individual LCD_xxx pin definitions.
```

```
/*                          EN, RS,   RW,   UNUSED,  DATA  */
 const LCD_PIN_MAP LCD_OUTPUT_MAP =  {0,  0,    0,    0,      0};
 const LCD_PIN_MAP LCD_INPUT_MAP =   {0,  0,    0,    0,      0xF};
#endif


//////////////////// END CONFIGURATION //////////////////////////////////


#ifndef LCD_ENABLE_PIN
  #define lcd_output_enable(x) lcdlat.enable=x
  #define lcd_enable_tris()   lcdtris.enable=0
#else
  #define lcd_output_enable(x) output_bit(LCD_ENABLE_PIN, x)
  #define lcd_enable_tris()  output_drive(LCD_ENABLE_PIN)
#endif


#ifndef LCD_RS_PIN
  #define lcd_output_rs(x) lcdlat.rs=x
  #define lcd_rs_tris()   lcdtris.rs=0
#else
  #define lcd_output_rs(x) output_bit(LCD_RS_PIN, x)
  #define lcd_rs_tris()  output_drive(LCD_RS_PIN)
#endif
```

```
#ifndef LCD_RW_PIN
  #define lcd_output_rw(x) lcdlat.rw=x
  #define lcd_rw_tris()   lcdtris.rw=0
#else
  #define lcd_output_rw(x) output_bit(LCD_RW_PIN, x)
  #define lcd_rw_tris()  output_drive(LCD_RW_PIN)
#endif

// original version of this library incorrectly labeled LCD_DATA0 as LCD_DATA4,
// LCD_DATA1 as LCD_DATA5, and so on.  this block of code makes the driver
// compatible with any code written for the original library
#if (defined(LCD_DATA0) && defined(LCD_DATA1) && defined(LCD_DATA2) && defined(LCD_DATA3) && !defined(LCD_DATA4) && !defined(LCD_DATA5) && !defined(LCD_DATA6) && !defined(LCD_DATA7))
  #define  LCD_DATA4    LCD_DATA0
  #define  LCD_DATA5    LCD_DATA1
  #define  LCD_DATA6    LCD_DATA2
  #define  LCD_DATA7    LCD_DATA3
#endif

#ifndef LCD_DATA4
#ifndef LCD_DATA_PORT
```

```
  #if defined(__PCB__)
    #define LCD_DATA_PORT     0x06    //portb
    #define set_tris_lcd(x)   set_tris_b(x)
  #else
   #if defined(PIN_D0)
    #define LCD_DATA_PORT     getenv("SFR:PORTD")    //portd
   #else
    #define LCD_DATA_PORT     getenv("SFR:PORTB")    //portb
   #endif
  #endif
#endif

#if defined(__PCB__)
  LCD_PIN_MAP lcd, lcdlat;
  #byte lcd = LCD_DATA_PORT
  #byte lcdlat = LCD_DATA_PORT
#elif defined(__PCM__)
  LCD_PIN_MAP lcd, lcdlat, lcdtris;
  #byte lcd = LCD_DATA_PORT
  #byte lcdlat = LCD_DATA_PORT
  #byte lcdtris = LCD_DATA_PORT+0x80
#elif defined(__PCH__)
```

```
  LCD_PIN_MAP lcd, lcdlat, lcdtris;
  #byte lcd = LCD_DATA_PORT
  #byte lcdlat = LCD_DATA_PORT+9
  #byte lcdtris = LCD_DATA_PORT+0x12
#elif defined(__PCD__)
  LCD_PIN_MAP lcd, lcdlat, lcdtris;
  #word lcd = LCD_DATA_PORT
  #word lcdlat = LCD_DATA_PORT+2
  #word lcdtris = LCD_DATA_PORT-0x02
#endif
#endif  //LCD_DATA4 not defined

#ifndef LCD_TYPE
  #define LCD_TYPE 2          // 0=5x7, 1=5x10, 2=2 lines
#endif

#ifndef LCD_LINE_TWO
  #define LCD_LINE_TWO 0x40   // LCD RAM address for the second line
#endif

BYTE const LCD_INIT_STRING[4] = {0x20 | (LCD_TYPE << 2), 0xc, 1, 6};
                    // These bytes need to be sent to the LCD
```

```
                    // to start it up.


BYTE lcd_read_nibble(void);


BYTE lcd_read_byte(void)

{

  BYTE low,high;


 #if defined(__PCB__)

  set_tris_lcd(LCD_INPUT_MAP);

 #else

 #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))

  output_float(LCD_DATA4);

  output_float(LCD_DATA5);

  output_float(LCD_DATA6);

  output_float(LCD_DATA7);

 #else

  lcdtris.data = 0xF;

 #endif

 #endif


  lcd_output_rw(1);
```

```
  delay_cycles(1);
  lcd_output_enable(1);
  delay_cycles(1);
  high = lcd_read_nibble();

  lcd_output_enable(0);
  delay_cycles(1);
  lcd_output_enable(1);
  delay_us(1);
  low = lcd_read_nibble();

  lcd_output_enable(0);

 #if defined(__PCB__)
  set_tris_lcd(LCD_INPUT_MAP);
 #else
 #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
  output_drive(LCD_DATA4);
  output_drive(LCD_DATA5);
  output_drive(LCD_DATA6);
  output_drive(LCD_DATA7);
 #else
```

```
  lcdtris.data = 0x0;
 #endif
#endif

  return( (high<<4) | low);
}

BYTE lcd_read_nibble(void)
{
 #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
  BYTE n = 0x00;

  /* Read the data port */
  n |= input(LCD_DATA4);
  n |= input(LCD_DATA5) << 1;
  n |= input(LCD_DATA6) << 2;
  n |= input(LCD_DATA7) << 3;

  return(n);
 #else
  return(lcd.data);
 #endif
```

```
}


void lcd_send_nibble(BYTE n)

{

 #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) && defined(LCD_DATA7))

  /* Write to the data port */

  output_bit(LCD_DATA4, bit_test(n, 0));

  output_bit(LCD_DATA5, bit_test(n, 1));

  output_bit(LCD_DATA6, bit_test(n, 2));

  output_bit(LCD_DATA7, bit_test(n, 3));

 #else

  lcdlat.data = n;

 #endif


  delay_cycles(1);

  lcd_output_enable(1);

  delay_us(2);

  lcd_output_enable(0);

}


void lcd_send_byte(BYTE address, BYTE n)

{
```

```
  lcd_output_rs(0);
  while ( bit_test(lcd_read_byte(),7) ) ;
  lcd_output_rs(address);
  delay_cycles(1);
  lcd_output_rw(0);
  delay_cycles(1);
  lcd_output_enable(0);
  lcd_send_nibble(n >> 4);
  lcd_send_nibble(n & 0xf);
}

void lcd_init(void)
{
  BYTE i;

 #if defined(__PCB__)
  set_tris_lcd(LCD_OUTPUT_MAP);
 #else
 #if (defined(LCD_DATA4) && defined(LCD_DATA5) && defined(LCD_DATA6) &&
defined(LCD_DATA7))
  output_drive(LCD_DATA4);
  output_drive(LCD_DATA5);
  output_drive(LCD_DATA6);
```

```
 output_drive(LCD_DATA7);
#else
 lcdtris.data = 0x0;
#endif
 lcd_enable_tris();
 lcd_rs_tris();
 lcd_rw_tris();
#endif


 lcd_output_rs(0);
 lcd_output_rw(0);
 lcd_output_enable(0);


 delay_ms(15);
 for(i=1;i<=3;++i)
 {
    lcd_send_nibble(3);
    delay_ms(5);
 }


 lcd_send_nibble(2);
 for(i=0;i<=3;++i)
```

```
    lcd_send_byte(0,LCD_INIT_STRING[i]);
}

void lcd_gotoxy(BYTE x, BYTE y)
{
  BYTE address;

  if(y!=1)
    address=LCD_LINE_TWO;
  else
    address=0;

  address+=x-1;
  lcd_send_byte(0,0x80|address);
}

void lcd_putc(char c)
{
  switch (c)
  {
    case '\f'   :  lcd_send_byte(0,1);
                   delay_ms(2);
```

```
                break;

    case '\n'   : lcd_gotoxy(1,2);       break;

    case '\b'   : lcd_send_byte(0,0x10);  break;

    default     : lcd_send_byte(1,c);     break;
  }
}

char lcd_getc(BYTE x, BYTE y)
{
  char value;

  lcd_gotoxy(x,y);
  while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
  lcd_output_rs(1);
  value = lcd_read_byte();
  lcd_output_rs(0);

  return(value);
}
```