PRÁCTICA FINAL SDMI

Estación meteorológica

David Guerra, Roman Valls, Daniel Clemente. SDMI22E. Junio 2005.

Índice

PRÁCTICA FINAL SDMI....1
Enunciado....2
Diagrama de bloques de la placa....3
Esquema eléctrico....4
Explicación de la solución hardware propuesta....5
Diagrama de flujo del software....6
Cálculos Teóricos:....7
  Número de overflows por segundo:....7
  Conversiones entre unidades:....7
  Cálculo de medias:....7
  Coste de RSIs:....8
Ampliaciones:....8
  Visión general:....8
  Modos:....9
Explicación del código:....9
  main.c:....9
  timer2.c:....10
  ad.c:....10
  hsi.c:....10
  lcd.c:....10
  display.c:....10
Listado del software....11
  main.c....11
  timer2.c....13
  ad.c....15
  hsi.c....16
  lcd.c....18
  display.c....21
  interrupts.a96....27
  Lcd.c....28
  lcd.inc....31
Control de versiones concurrente:....35
  Breve introducción a Subversion:....35
Licencia:....36

Enunciado

La práctica consiste en usar el micro 80C196KD con una placa de expansión para implementar una estación meteorológica. Mediante un panel se mostrará:

- Temperatura ambiente (de -20 a 50 grados)
- Humedad (de 0 a 100 %)
- Velocidad del viento (km/h y nombre)
- Dirección del viento (ángulo y nombre)

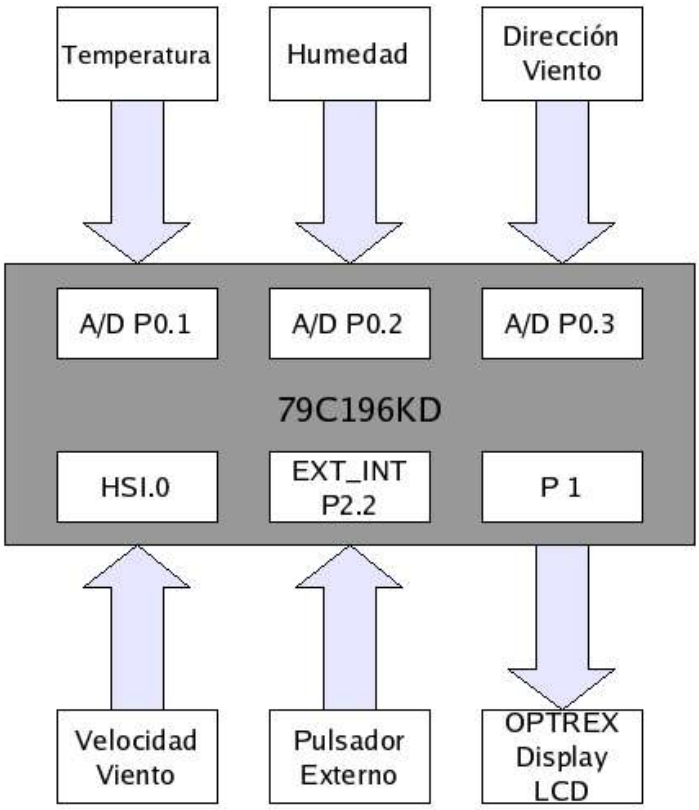
Los valores para V (velocidad) y D (dirección) se actualizan cada segundo, pero T (temperatura) y H (humedad) se muestran como la media durante los 10 últimos segundos (o minutos, según se configure).

Además está la función de Reloj, que cuenta en formato HH:MM:SS.

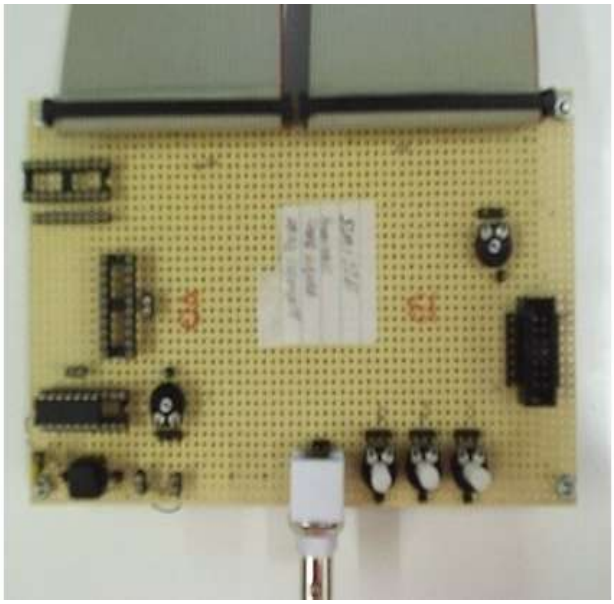
Para medir la temperatura, humedad y dirección usaremos el conversor AD, ya que las muestras tienen una variabilidad pequeña y por tanto se pueden medir correctamente.

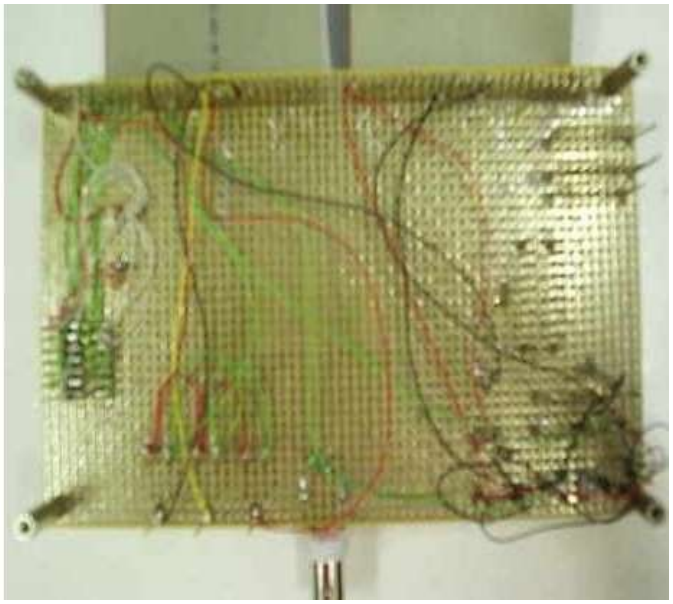
En cambio, para la velocidad usamos las HSI's porque queremos tratar valores discretos que nos llegan del generador de funciones.

Diagrama de bloques de la placa

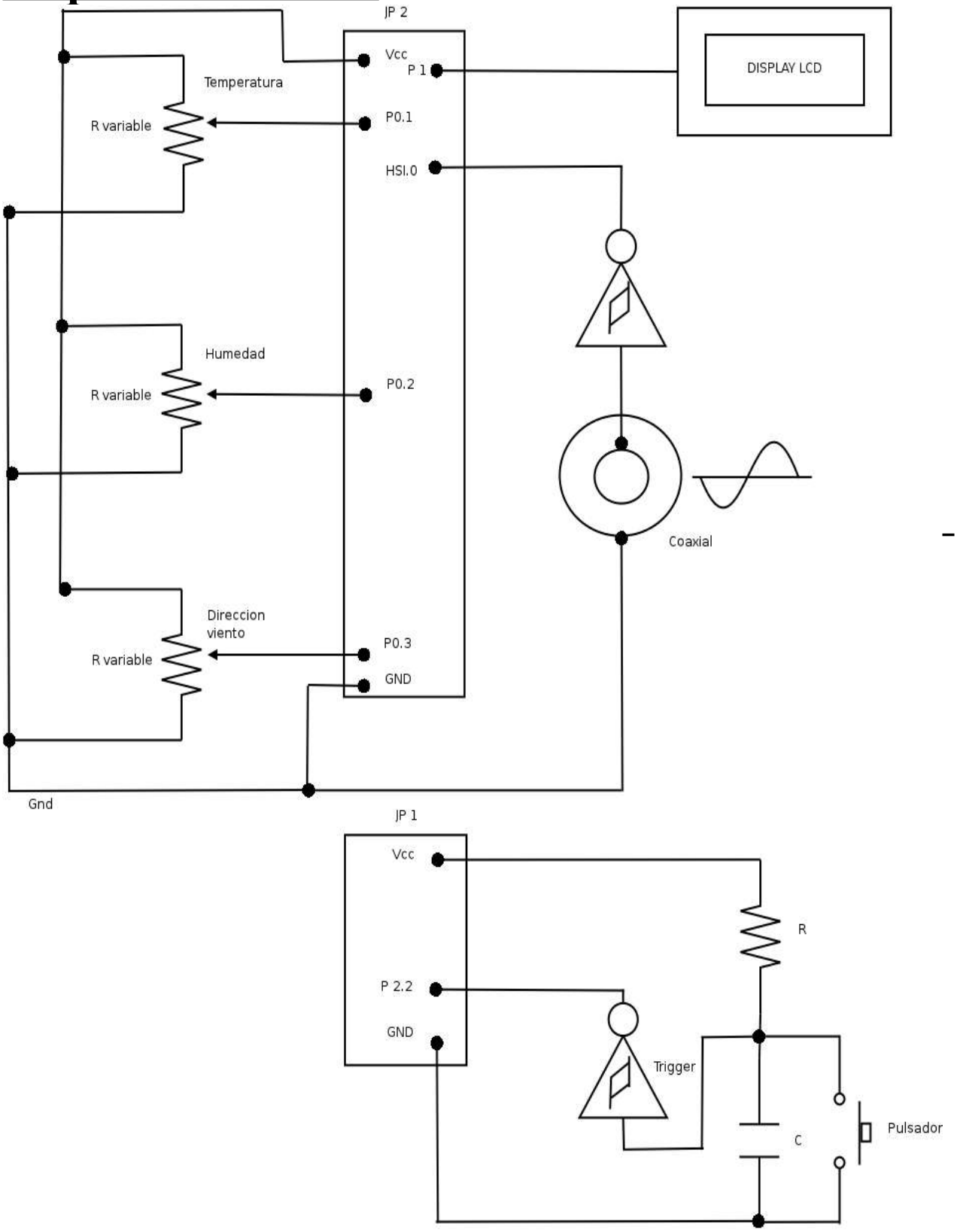


Las siguientes fotografías muestran el aspecto de la placa de ampliación una vez finalizadas todas las conexiones que hemos comentado:





Esquema eléctrico



Explicación de la solución hardware propuesta

En el montaje tratamos cinco entradas diferentes, tres de ellas son entradas A/D (temperatura, humedad y dirección del viento), una HSI (velocidad del viento), y por último una interrupción externa (pulsador); como interfaz de salida utilizamos un display LCD.

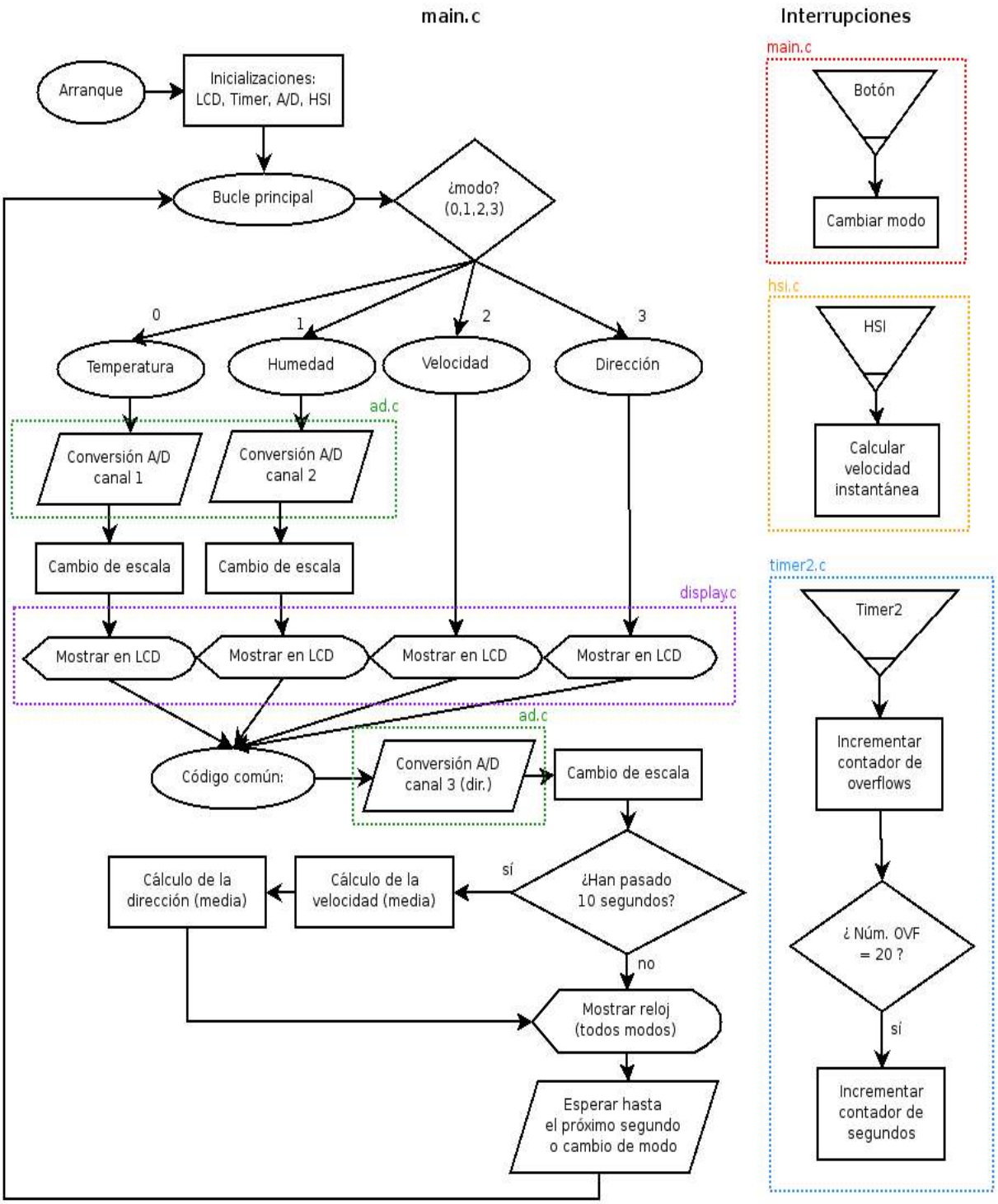
En el puerto P0 se encuentran las entradas analógicas; de las ocho entradas posibles, se han utilizado tres. Éstas han sido configuradas de la siguiente manera: P0.1 (patilla 29 JP2) para tomar muestras de temperatura, P0.2 (patilla 31 JP2) para muestras de humedad y P0.3 (patilla 35 JP2) para la dirección del viento. Cada vez que solicitamos una conversión, se ha de seleccionar, mediante el registro ad_command, la patilla del P0 que tiene la información que nos interesa. Como las entradas en realidad eran simuladas por potenciómetros, la Vcc y Gnd de estos se tomaron de las patillas 2 y 1 de JP2.

Para poder hacer el cálculo de la velocidad del viento, se simula mediante un generador de señal una entrada de pulsos (tics). Debido a que esta señal puede producir falsas lecturas, se utiliza también una puerta en histéresis (40106, patilla 3 entrada, patilla 4 salida), de esta forma nos aseguramos de que el ruido de la señal no provocará falsas HSI. La salida de la puerta en histéresis, al estar ya filtrada, se conecta a la entrada HSI.0 (patilla 12 JP2).

El montaje del pulsador (interrupción externa) consta de varios elementos: una resistencia y un condensador conectados en serie, un pulsador conectado en paralelo con el condensador y una puerta en histéresis (40106, patilla 1 entrada, patilla 2 salida); en este caso también es necesario filtrar la señal producida por el pulsador, ya que al conmutar, se producen una serie de flancos fantasma que podrían provocar la lectura de falsas pulsaciones. La entrada de la puerta en histéresis se conecta en el punto en el que se unen: pulsador, resistencia y condensador. La salida del trigger, al tener ya filtrada la pulsación, se conecta directamente a P2.2 (patilla 28 JP1); las Vcc y Gnd de la puerta en histéresis y del filtro formado por el condensador y la resistencia , se toman de las patillas 1 y 19 de JP1.

Para la salida de información utilizamos un display conectado al P1, cuyas Vcc y Gnd se han tomado de las patillas 2 y 1 de JP2. El display necesita además un potenciómetro para regular el contraste. Las Vcc y Gnd de este potenciómetro también se han tomado de las patillas 2 y 1 de JP2.

Diagrama de flujo del software



La primera tarea que realiza nuestro programa es la de llamar a las funciones encargadas de inicializar todos los dispositivos involucrados en el sistema (LCD, Timer, AD & HSI). Una vez establecidos los parámetros de cada dispositivo, se entra en el bucle principal del programa.

Mediante el botón se selecciona el modo activo (0, 1, 2, 3) en el gráfico. Dependiendo del modo selecionado, se lanzarán conversiones digitales o se guardarán interrupciones en de la hsi (con sus respectivos cambios de escala).

El tratamiento de medias se comentará más adelante en detalle, pero la idea básica es recoger valores cada segundo y mostrar el resultado de la media cada 10 segundos.

Paralelamente a las anteriores operaciones, se mantiene un contador mediante el timer2 que nos servirá para mostrar la hora. Más adelante también se detalla el cálculo teórico que nos permite decir que 20 overflows equivale a un segundo.

Cálculos Teóricos:

Número de overflows por segundo:

El reloj del 196KD va 20 Mhz. A razón de 1 state time = 2 ciclos, hay 10M st por segundo. Como hemos configurado el Timer2 en "modo normal", se incrementa una vez cada 8 st, o sea, 10M/8 = 10485760/8 = 1310720 incrementos por segundo. Como el Timer2 es de 16 bits, se desborda cada 2^16 incrementos, por tanto 1310720/(2^16) = 1310720/65536 = exactamente 20 veces por segundo. En total, cada 20 overflows del Timer2 representan un segundo; eso hace fácil contar el tiempo desde la propia RSI (cada 20 ejecuciones de la RSI, incrementar el número de segundos).

$$20\ MHz \cdot \frac{2^{20}\ Hz}{1\ MHz} \cdot \frac{1\ ciclo/s}{1\ Hz} \cdot \frac{1\ stt}{2\ ciclos} \cdot \frac{1\ incr}{8\ stt} \cdot \frac{1\ OVF}{2^{16}\ incr} = \frac{20\ OVF}{s}$$

Conversiones entre unidades:

Para poder mostrar el valor interno recogido por el conversor AD a un valor representativo, tenemos que usar factores de conversión simples procurando que ningún término de la operación se desborde (valor >65535). Estas operaciones simples se pueden observar en el código.

El cálculo más complicado es el de "velocidad de los tics HSI" a km/h al medir la velocidad del viento. Para que no se desborde la variable, se hace el cálculo en milisegundos, y teniendo en cuenta que el Timer puede haberse desbordado varias veces entre un tic y el siguiente. En el código están detalladas las conversiones necesarias.

Cálculo de medias:

La temperatura y la dirección del viento se muestrean cada segundo, pero el valor mostrado sólo se actualiza cada 10 segundos, con una media de los 10 últimos valores instantáneos.

Usamos la media aritmética normal: en una variable se van acumulando los 10 valores, y

cuando llega el décimo, dividimos entre 10. Con un entero (16 bits) para el sumatorio es suficiente, ya que la temperatura puede ser como mucho 50 grados (50*10=500) y la dirección del viento 360 grados (360*10=3600), por tanto el acumulador siempre valdrá menos de 65535.

Coste de RSIs:

Todas las RSI (botón, HSI, Timer2) inhiben las interrupciones, pero se ejecutan muy rápido. La del botón y la del reloj son triviales (unas pocas operaciones matemáticas), y la de HSI siguen siendo cálculos de tiempo constante, sin ningún bucle.

La de los HSI consiste en: 2 sumas/restas, 4 productos/divisiones, y varias asignaciones. Todo esto se tendrá que ejecutar antes de que llegue el siguiente tic por HSI para que los cálculos sean correctos. Si un tic llegase demasiado rápido, se perdería ya que las interrupciones están desactivadas mientras se ejecuta esta RSI. A velocidades razonables para un anemómetro, el código funciona bien.

Ampliaciones:

Aspecto de la placa y funcionamiento:

Visión general:

Modos:

Añadidos visuales: Hemos considerado oportuno añadir barras visuales (bloques negros que se aprecian en las fotos), para indicar el nivel de humedad y temperatura en estos dos modos.

Explicación del código:

main.c:

Este fichero contiene el programa principal (main). Lo primero que hace es inicializar los registros de los diferentes componentes de la placa que se usarán (ext_int, timer2, ad y hsi's).

Las funciones relativas a cada dispositivo se encuentran en ficheros separados para facilitar su mantenibilidad. Se separan dentro de cada uno de estos ficheros, las inicializaciones y el código específico que deben ejecutar para los propósitos de la práctica (inits_ad y conversio_ad, por ejemplo).

Cabe destacar la función del bucle principal que selecciona la acción a hacer de entre los diferentes modos, en función del valor de la variable 'modo' que es actualizada por el botón conectado a ext_int.

Además, también actualiza el reloj que se muestra en todos los modos, cada segundo (mediante la espera que realizamos con el while anidado). También se aprovecha para tomar muestras de las magnitudes que necesitan media (velocidad y dirección del viento).

timer2.c:

Obviando las inicializaciones del timer2 (inits_timer2()), la parte más relevante de este fichero es la RSI. Dicha RSI mantiene la cuenta de los overflows que se producen en el timer2, contando 1 segundo para cada 20 overflows (ver sección de cálculos teóricos para su explicación).

ad.c:

La temperatura, humedad y dirección usan 3 de los 8 canales del AD para realizar sus conversiones. Posteriormente se realiza un cambio de escala para adaptar el valor interno a la representación que queremos (grados centígrados, porcentaje y grados sexagesimales respectivamente).

hsi.c:

Para medir la velocidad del viento usamos las HSI's. Como el anemómetro puede girar muy poco a poco, lo que hacemos es contar el tiempo que ha pasado entre un tic HSI y el siguiente, y luego hacer una conversión (explicada en la sección de cálculos teóricos).

lcd.c:

Contiene las primitivas de control del display LCD, tal como se definieron en prácticas anteriores. Algunas de ellas han sido modificadas para evitar problemas al redibujar caracteres en el display y para hacerlas más genericas (siendo así usables en otras partes del proyecto).

display.c:

Contiene rutinas de alto nivel para el manejo del display LCD. Usa las funciones primitivas definidas en lcd.c. También están definidas funciones que muestran barras visuales para los modos humedad y temperatura, y los cálculos necesarios para saber qué texto hay que mostrar con los nombres de velocidades y direcciones del viento.

Listado del software

main.c

```
#include <kd_sfrs.h>
#define NUM_MODOS 4 //Cuantos modos queremos ?

// Variables globales
int modo; // Modo actual

extern unsigned int timer2_seg; // Segundos

extern signed int tem_disp;
extern unsigned int hum_disp;
extern unsigned int vel_disp;
extern unsigned int dir_disp;
extern unsigned int vel_inst;

unsigned int vel_sum; // Sumatorio de velocidades, para hacer la media
unsigned int dir_sum; // Sumatoria de ángulos de dirección del viento

void main()
{

 int temp, temp1, temp2; // Temporales


// inits

  // Botón
  asm {di;}
  wsr=0;
  int_mask=int_mask|0x80;       // activamos la interrupcion (EXTINT)
  int_mask1=int_mask1&0xDF;     // Desactivamos INT13 (EXT_PIN=0)

  wsr=0;
  ioc1=ioc1 & 0xFD;                     //EXT_INT=0 para elegir p2.2
  asm {ei;}

  // Otros
  inits_lcd();
  inits_timer2();
  inits_ad();
  inits_hsi();

  modo=0;
  vel_sum=0;
  dir_sum=0;

//main loop
//   Este bucle se ejecuta cada segundo para tomar las muestras necesarias,
// hacer medias, y mostrarlas en pantalla.

while(1)
  {
```

```
switch (modo)
{

// Temperatura. A/D, sin media
case 0:
        temp=conversio_ad(1);
        // convertir de [0-255] a [-20 - 50]
        temp=((temp*70)/255)-20;
        tem_disp=temp;
        mostrar_tem();
        visual_tem();
        break;

// Humedad. A/D, sin media
case 1:
        temp=conversio_ad(2);
        // convertir de [0-255] a [0-100 %]
        temp=(temp*100)/255;
        hum_disp=temp;
        mostrar_hum();
        visual_hum();
        break;

// Velocidad viento. HSI, con nombres, con media
case 2:
        //   Ya tenemos vel_disp actualizada, ya que se hace en cada
        // iteración del bucle (cada segundo)
        mostrar_vel();
        break;

// Dirección viento. A/D, con nombres, con media
case 3:
        //   Ya tenemos dir_disp actualizada, ya que se hace en cada
        // iteración del bucle (cada segundo)
        mostrar_dir();
        break;

// Modo incorrecto
default:
        show_msg_lcd("Unimplemented mode code",2,1);
        break;
}


//   La velocidad (HSI) y dirección (A/D) del viento requieren media, y por tanto
// hay que mantenerla siempre actualizada, independientemente del modo en el que
// estemos.

temp=conversio_ad(3); // dirección del viento
// convertir de [0-255] a [0-360 grados]
// *(360/255). Para evitar desbordamientos,
temp=(temp*72)/51;

vel_sum+=vel_inst;    // Acumular
dir_sum+=temp;        // Acumular
if (timer2_seg%10==0) // Si han pasado 10 segundos
{
```

```
                vel_disp=vel_sum/10; // Actualizar el valor mostrado
                dir_disp=dir_sum/10;
                vel_sum=0;
                dir_sum=0;
        }

        // Mostrar reloj, común para todos los modos

        show_char_lcd(':',11,2);
        show_char_lcd(':',14,2);

        temp=timer2_seg%60; //segundos
        show_value_lcd(temp,2,'0',16,2);

        temp=timer2_seg/60; //minutos
        temp%=60;
        show_value_lcd(temp,2,'0',13,2);

        temp=timer2_seg/60;
        temp/=60;//horas
        temp%=24;//hora: de 0 a 23
        show_value_lcd(temp,2,'0',10,2);

        //   Esperar hasta que haya pasado un segundo o cambiemos de modo.
        //   Como tomamos muestras cada segundo, no habrá nada nuevo
        // que mostrar en pantalla hasta la siguiente muestra.
        temp1=timer2_seg; temp2=modo;
        while( temp1==timer2_seg && temp2==modo);

        // Si se ha cambiado de modo, borrar la pantalla
        if (temp2!=modo) CLRSCR_LCD();

        // Fin del bucle principal
  }

}


//RSI's
void RSIExtInt()
{
  asm{di;}
  modo=(modo+1)%NUM_MODOS;
  asm{ei;}
}
```

---

timer2.c

```
#include <kd_sfrs.h>

#define OVFS_POR_SEG 20

unsigned int timer2_seg;
```

```
unsigned int timer2_ovf;
unsigned int auxiliar;

extern unsigned int t2_ovf_hsi;

void inits_timer2 () {

        timer2_ovf=0;
        timer2_seg=0;


        /* Deshabilitem les interrupcions i les PTS */
        asm {
                dpts;
                di;
        }

        wsr=0x00;
        int_mask1 |= 0x10;      // T2OVF

        wsr=0x0f;
        auxiliar=ioc0;

        wsr=0x00;
        auxiliar &= 0xf7;
        ioc0=auxiliar;          // desactivo EXTRST del timer2

        wsr=0x0f;
        auxiliar=ioc2;
        wsr=0x00;
        auxiliar |= 0x02;
        ioc2=auxiliar;          // Configurem el TIMER2 per comptar UP en normal mode, al rang
                                // FFFFh-0000h.

        wsr=0x01;
        ioc3 = ioc3 | 0x01;     // Seleccionem el clock intern pel TIMER2

        wsr=0x00;
        timer2=0;               // Carreguem els valors inicials als comptadors


        asm { ei;}


}

void RSITimer2() {
// Incrementa el comptador de interrupcions del timer i avisa que s'ha produit una interrupcio
// i reprograma el timer2
        asm {di;}

        t2_ovf_hsi++; // Para poder contar bien el tiempo entre HSI y HSI

        timer2_ovf++;
        // contamos en overflows pero transformamos a segundos

        if (timer2_ovf==OVFS_POR_SEG) {
```

```
          ++timer2_seg;
          timer2_ovf=0;
        }
        // El timer2 sigue contando, automáticamente
        asm {ei;}
}
```

---

ad.c

```
#include <kd_sfrs.h>


void inits_ad()
{

        unsigned int regtemp;

        asm { di; }
        wsr=0;  //Aunque podemos usar cualquier ventana para int_mask/int_mask1
        int_mask=int_mask & 0xFD; // Deshabilitamos int del AD (realizaremos encuesta, no int)

/*  Modifiquem el registre IOC2, seleccionem les opcions del conversor.*/
        wsr=0xf; regtemp=ioc2;
        regtemp = regtemp & 0xF7; // AD_TIME_ENA=0 para modo compatible con 196KB
        wsr=0x0; ioc2=regtemp;

        wsr=0xf; regtemp=ad_command;
        regtemp = regtemp | 0x10; // 8 bits
        wsr=0x0; ad_command=regtemp;

/*  Finalment cal habilitar les interrupcions per poder servir-les. */
        asm { ei;}

}

unsigned int conversio_ad(char canal) {
/* Este bucle espera a que se haya acabado la interrupción que lee el contenido del
   registro de resultado del conversor AD (rutina RSIADDone). Una vez acabada se
   muestra el resultado por el array de leds de la placa placa */
        unsigned int regtemp;

        wsr=0x0; // Para leer ad_result
        while (ad_result&0x08); // Esperar a que haya acabado (encuesta)

        asm { di; }

        wsr=0xf; regtemp=ad_command;
        if (canal==1) { regtemp&=0xF9; regtemp|=1; }       // ...001
        else if (canal==2) { regtemp&=0xFA; regtemp|=2; }  // ...010
        else if (canal==3) { regtemp&=0xFB; regtemp|=3; }  // ...011
```

```
        regtemp = regtemp | 0x08; // GO=1
        wsr=0x0; ad_command=regtemp;

        asm { ei; }


        while (ad_result&0x08); // Esperar a que haya acabado (encuesta)

        asm { di; }

        regtemp=ad_result;
        regtemp=regtemp>>8;
        regtemp &= 0xFF;

        asm { ei; }

        return regtemp;
}
```

---

hsi.c

```
#include <kd_sfrs.h>

unsigned int instant_ant; // Momento en el que ha llegado la HSI anterior
unsigned int t2_ovf_hsi;  // Contador de overflows
unsigned int vel_inst;    // Velocidad instantánea, en km/h

// 2=2, K=0.15, PI=3.14. Per tant,
#define DOS_K_PI 1

static unsigned int auxiliar;

void inits_hsi()
{

        /* Deshabilitem les interrupcions i les PTS */
        asm {
                dpts;
                di;
        }

        wsr = 0x00;
        int_mask |= 0x04;       // HSIData Available, cal configurar IOC1.7

        wsr = 0x0f;
        auxiliar = ioc0;

        wsr = 0x00;
        auxiliar |= 0x01;
        ioc0 = auxiliar;        // activo hsi0
```

```
        wsr = 0x0f;
        auxiliar = ioc1;

        wsr = 0x00;
        auxiliar &= 0x7f;
        ioc1 = auxiliar;        // La Hsi Data Available (HSIINT) es generada quan
                                // es carrega el Holding Register
        wsr = 0x0f;
        auxiliar = hsi_mode;

        wsr = 0x00;
        auxiliar |= 0x01;       // cada transició positiva HSI.0 provocarà interrupció.
        hsi_mode = auxiliar;

        instant_ant=0;
        t2_ovf_hsi=0;
        vel_inst=0;

        asm { ei; }
}



void RSI_Hsi()
{
        unsigned int instant;  // Valor del Timer2 (en tics) cuando llega la HSI
        signed int ha_passat;  // 'signed' para contabilizar bien en caso de overflow
                               // Ej con 2 díg: sube de 45 a "15" (en realidad 115), num_ovf=1
                               // El total será 1*100 + (15-45) = 70
        extern int vel_disp;

        // Comptar els numero de tics que arriven
    asm { di; }
        // Leer hsi_time para cargar el siguiente valor
        wsr = 0x0;
        instant = hsi_time;

        ha_passat=instant-instant_ant;

        // Ara tenim el número de tics que han passats,
        // juntament amb el número d'overflows que hi ha hagut.
        // 20 OVFs del T2 == 1 segon.
        // Per tant, 50 ms per OVF.
        //
        // 20*2^16 = 1310720 tics = 1 segon
        // Per tant, 1310 tics per ms.
        //

        // mspertic = num_ovf*50+ha_passat/1310
        // Aquest és el nombre de mseg que ha trigat el tic: mseg/tic
        // El seu invers, tics/mseg, és la freqüència
        // En tics/seg, f=1000/mspertic
        // metrperseg=2*k*pi*f
        // kmperh=metrperseg*3600/1000
        //
        // En total queda: kmperh=3600/1000 * 2*k*pi * 1000/mspertic
        // Simplificat: kmperh= 3600/mspertic * 2*k*pi
```

```
        vel_inst=t2_ovf_hsi*50+ha_passat/1310; // mspertic
        vel_inst=3600/vel_inst; // ...
        vel_inst*=DOS_K_PI; // kmperhora


        //   En el bucle principal ya se cogerá esta velocidad instantánea (cada segundo),
        // y se hará la media cada 10.


        // Preparar-nos pel següent tic
        instant_ant=instant; // (encara que s'ignora el que ha trigat en executar-se la RSI)
        t2_ovf_hsi=0;
        asm { ei; }
}
```

---

lcd.c

```
#include <kd_sfrs.h>
#include "lcd.h"

/*****************************************************************************
;  Declaració de constants del display
;*****************************************************************************
;conexionat:
;       p1.4..p1.7 = D4..D7
;       p1.0 = RS
;       p1.2 = E
;       0 = R/W  siempre escribimos
*/

#define LCD_INS 0xfe
#define LCD_DAT 0x01
#define LCD_ENA 0x04

#define LCD_LINEA2 0xC0
#define LCD_LINEA1 0x80
#define LCD_CLRSCR 0x01                 /* RETARD DE 1.64 ms */

void WRITE_LCD_4BITS();

void INICIALITZA_LCD()
{
/*
;0      X000 0000B = 00H   ->              IOPORT1
;1      X000 0001B = 01H
;15     X000 1111B = 0FH
*/

  DELAY_10mS();
  DELAY_10mS();

  wsr=0;
```

```
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x20 & LCD_INS; /* interface de 4 bits */
  WRITE_LCD_4BITS();
  DELAY_10mS();

/*AHORA YA PODEMOS USAR LA FUNCION DE INSTRUCION LCD*/
/*FUNCTION SET- 4 bits dades, 2 lines, caracter font*/

  INSTRUCCIO_LCD(0x28);

  /*DISPLAY ON/OFF-DISPLAY ON, CURSOR OFF, CURSOR NO BLINK*/
  INSTRUCCIO_LCD(0x0C);

  INSTRUCCIO_LCD(0x06);  /*ENTRY MODE SET-INCREMENT CURSOR, CURSOR MOVEMENT*/

  INSTRUCCIO_LCD(0x01);  /*CLEAR DISPLAY*/
  DELAY_10mS();
  /*ESTE RETARDO DEBERIA DE SER DE 1.64mS*/
}

void WRITE_LCD_4BITS()
{
/*AQUESTA FUNCIO ES NOMES PER US INTERN DE LA LLIBRERIA DEL DISPLAY*/

  asm{NOP;NOP;}
  wsr=0;
  ioport1 = ioport1 ^ LCD_ENA; /*XORB IOPORT1,#LCD_ENA*/

  asm{
  NOP;NOP;NOP;NOP;NOP;
  NOP;NOP;NOP;NOP;NOP;
  }

  wsr=0;
  ioport1 = ioport1 ^ LCD_ENA; /*XORB IOPORT1,#LCD_ENA*/
  asm{NOP;}
}


void INSTRUCCIO_LCD(unsigned char inst)
{
  wsr=0;
  ioport1=0xF0 & inst & LCD_INS ;                /* RS=0 */
```

```
  WRITE_LCD_4BITS();
  inst<<=4;
  asm{
  NOP;NOP;
  NOP;NOP;NOP;
  }
  wsr=0;
  ioport1 =0xF0 & inst & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_40uS();
}

void DADA_LCD(unsigned char dato)
{
  wsr=0;
  ioport1=0xF0 & dato; /*parte alta primero*/
  ioport1=ioport1|LCD_DAT;   /*RS=1*/
  WRITE_LCD_4BITS();
  dato<<=4;
  asm{
  NOP;NOP;
  NOP;NOP;NOP;
  }
  wsr=0;
  ioport1=0x0F0 & dato;
  ioport1=ioport1|LCD_DAT;
  WRITE_LCD_4BITS();
  DELAY_40uS();
}

void CLRSCR_LCD()
{
  INSTRUCCIO_LCD(LCD_CLRSCR);
  DELAY_10mS();
}

void DELAY_10mS()
{
  unsigned int TIMER_LCD;

  TIMER_LCD=2000;
  while(TIMER_LCD!=0)
  {
     TIMER_LCD--;
  }
}

void DELAY_40uS()
{
  unsigned int TIMER_LCD;

  TIMER_LCD=40;
  while(TIMER_LCD!=0)
  {
   TIMER_LCD--;
  }
}
```

```
void GOTOXY(unsigned char x, unsigned char y)
/*   1<=x<=16  ,  1<=Y<=2 */
{
   unsigned char gotopos;
   gotopos=0x80 | ((y-1)*0x40) | (x-1);
   INSTRUCCIO_LCD(gotopos);
}
```

---

display.c

```
#include <kd_sfrs.h>
#include <lcd.h>
#include <string.h>

#define LCD_INS 0xfe
#define LCD_DAT 0x01
#define LCD_ENA 0x04

#define LCD_LINEA2 0xC0
#define LCD_LINEA1 0x80
#define LCD_CLRSCR 0x01                 /* RETARD DE 1.64 ms */


// Las variables con los valores correctos, listos para ser mostrados en pantalla
signed int tem_disp;
unsigned int hum_disp;
unsigned int vel_disp;
unsigned int dir_disp;

void WRITE_LCD_4BITS();

void INICIALITZA_LCD()
{
/*
;0      X000 0000B = 00H   ->              IOPORT1
;1      X000 0001B = 01H
;15     X000 1111B = 0FH
*/

  DELAY_10mS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();
```

```
  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x20 & LCD_INS; /* interface de 4 bits */
  WRITE_LCD_4BITS();
  DELAY_10mS();

/*AHORA YA PODEMOS USAR LA FUNCION DE INSTRUCION LCD*/
/*FUNCTION SET- 4 bits dades, 2 lines, caracter font*/

  INSTRUCCIO_LCD(0x28);

  /*DISPLAY ON/OFF-DISPLAY ON, CURSOR OFF, CURSOR NO BLINK*/
  INSTRUCCIO_LCD(0x0C);

  INSTRUCCIO_LCD(0x06);  /*ENTRY MODE SET-INCREMENT CURSOR, CURSOR MOVEMENT*/

  INSTRUCCIO_LCD(0x01);  /*CLEAR DISPLAY*/
  DELAY_10mS();
  /*ESTE RETARDO DEBERIA DE SER DE 1.64mS*/
}

void WRITE_LCD_4BITS()
{
/*AQUESTA FUNCIO ES NOMES PER US INTERN DE LA LLIBRERIA DEL DISPLAY*/

  asm{NOP;NOP;}
  wsr=0;
  ioport1 = ioport1 ^ LCD_ENA; /*XORB IOPORT1,#LCD_ENA*/

  asm{
  NOP;NOP;NOP;NOP;NOP;
  NOP;NOP;NOP;NOP;NOP;
  }

  wsr=0;
  ioport1 = ioport1 ^ LCD_ENA; /*XORB IOPORT1,#LCD_ENA*/
  asm{NOP;}
}


void INSTRUCCIO_LCD(unsigned char inst)
{
  wsr=0;
  ioport1=0xF0 & inst & LCD_INS ;                /* RS=0 */
  WRITE_LCD_4BITS();
  inst<<=4;
  asm{
  NOP;NOP;
  NOP;NOP;NOP;
  }
  wsr=0;
  ioport1 =0xF0 & inst & LCD_INS;
  WRITE_LCD_4BITS();
```

```
  DELAY_40uS();
}

void DADA_LCD(unsigned char dato)
{
  wsr=0;
  ioport1=0xF0 & dato; /*parte alta primero*/
  ioport1=ioport1|LCD_DAT;   /*RS=1*/
  WRITE_LCD_4BITS();
  dato<<=4;
  asm{
  NOP;NOP;
  NOP;NOP;NOP;
  }
  wsr=0;
  ioport1=0x0F0 & dato;
  ioport1=ioport1|LCD_DAT;
  WRITE_LCD_4BITS();
  DELAY_40uS();
}

void CLRSCR_LCD()
{
  INSTRUCCIO_LCD(LCD_CLRSCR);
  DELAY_10mS();
}

void DELAY_10mS()
{
  unsigned int TIMER_LCD;

  TIMER_LCD=2000;
  while(TIMER_LCD!=0)
  {
     TIMER_LCD--;
  }
}

void DELAY_40uS()
{
  unsigned int TIMER_LCD;

  TIMER_LCD=40;
  while(TIMER_LCD!=0)
  {
   TIMER_LCD--;
  }
}

void GOTOXY(unsigned char x, unsigned char y)
/*   1<=x<=16  ,  1<=Y<=2 */
{
   unsigned char gotopos;
   gotopos=0x80 | ((y-1)*0x40) | (x-1);
   INSTRUCCIO_LCD(gotopos);
}
```

```
void show_value_lcd(unsigned int wValor, int lon, char rell, int x, int y)
{

          unsigned char xifra;
          unsigned int valor;
          int i=0;
          valor=wValor;
          while (valor > 0 || wValor==0) {
                GOTOXY(x-i,y);
                xifra=valor%10+'0';
                valor=valor/10;
                DADA_LCD(xifra);
                wValor=1; //para no volver a entrar en el bucle
                i++;
          }

          while (i<lon) {
                GOTOXY(x-i,y);
                DADA_LCD(rell);
                i++;
          }
}

void show_char_lcd(unsigned int wValor, int x, int y)
{

                GOTOXY(x,y);
                DADA_LCD(wValor);
}

void inits_lcd()
{
        INICIALITZA_LCD();
        CLRSCR_LCD();
}

void show_msg_lcd(char missatge[], int x, int y)
{
        unsigned int i=0;

        GOTOXY(x,y);

        while (i<strlen(missatge)) {
                DADA_LCD(missatge[i]);
                i++;
        }
}



/**Funciones para mostrar valores de los modos**/

void mostrar_tem()
{
        signed int tem_abs;
        char signo;
        tem_abs=tem_disp;
```

```
        // La temperatura puede ser negativa
        if (tem_abs<0) {
                signo='-'; // '-'
                tem_abs=-tem_abs;
        } else {
                // El ' ' sobreescribirá al '-' si pasamos de tem. negativa a positiva
                signo=' '; // ' '
        }

        show_msg_lcd("temperatura",1,1);
        show_char_lcd(signo,12,1);
        show_value_lcd(tem_abs,2,' ',14,1);
        show_char_lcd('C',16,1);
        show_char_lcd(0xDF,15,1);
}

void visual_tem()
{
        signed int i=-2;
        signed int conv=(tem_disp)/10;

        GOTOXY(1,2);
        while(i<0) {
          DADA_LCD((conv<=i)?0xDB:' ' );
          i++;
        }
        while(i<5) {
          DADA_LCD( (conv>=i)?0xFF:' ' );
          i++;
        }


}

void mostrar_hum()
{
        show_msg_lcd("humedad",1,1);
        show_value_lcd(hum_disp,3,' ',15,1);
        show_char_lcd('%',16,1);
}

void visual_hum()
{
        int i=0;
        int conv=(7*hum_disp)/100;

        GOTOXY(1,2);
        while(i<conv) {
          DADA_LCD(0xFF);
          i++;
        }

        while(i<7) {
          DADA_LCD(' ');
          i++;
        }
```

```
}

void mostrar_vel()
{
char* velocitats[] = {
 "calm            ",
 "light air       ",
 "light breeze    ",
 "gentle breeze   ",
 "moderate breeze ",
 "fresh breeze    ",
 "strong breeze   ",
 "near gale       ",
 "gale            ",
 "strong gale     ",
 "storm           ",
 "violent storm   ",
 "hurricane       "
};

        unsigned int i;

        show_value_lcd(vel_disp,3,' ',3,2);
        show_msg_lcd("km/h",4,2);

        // vel_disp en km/h
        i=0;
        if (vel_disp>1) i=1;
        if (vel_disp>6) i=2;
        if (vel_disp>11) i=3;
        if (vel_disp>19) i=4;
        if (vel_disp>29) i=5;
        if (vel_disp>39) i=6;
        if (vel_disp>50) i=7;
        if (vel_disp>61) i=8;
        if (vel_disp>74) i=9;
        if (vel_disp>87) i=10;
        if (vel_disp>101) i=11;
        if (vel_disp>118) i=12;

        show_msg_lcd(velocitats[i],1,1);


}

void mostrar_dir()
{
char* direccions[] = {
  "tramunt.",
  "gregal  ",
  "llevant ",
  "xaloc   ",
  "migjorn ",
  "llebeig ",
  "ponent  ",
  "mestral "
```

```
};

        signed int i=0;

        show_msg_lcd("direccion",1,1);
        show_value_lcd(dir_disp,3,' ',15,1);
        show_char_lcd(0xDF,16,1);

        // Transformación para sacar el índice del vector:
        i=dir_disp+22; // Hacer que 0 grados sea donde empieza el viento norte
        i=i/45;        // Clasificar en uno de los 8 vientos
        i=i%8;         // Para conseguir rango [0-7]

        show_msg_lcd(direccions[i],1,2);

}
```

---

interrupts.a96

```
;
; This is an example STARTUP file.
; @(#)cstart.a96        1.5
;
STARTUP MODULE  CMAIN

        RSEG
SP      EQU     018H:WORD

      cseg at 0C000H
;  ------------------------
; Interrupt service routine addresses to be used in RISM EPROM.
; Note:
;  Of all these interrupt vectors, only the SERIAL, NMI and TRAP vectors are
;  required for operation of the RISM.  The other vectors are provided as
;  fixed entry points for routines which may be loaded into RAM in the
;  diagnostic mode.
;  In the diagnostic mode memory at the interrupt vectors is mapped to EPROM
;  so it is not possible to write into the vector table.

        EXTRN   main:NULL
        EXTRN   RSIExtInt:NULL
        EXTRN   RSITimer2:NULL
        EXTRN   RSI_Hsi:NULL

cstart:
        PUBLIC  cstart

        DI
        LD      SP,#100H        ; pila al final variables
        EI
```

```
; Some applications might want to do a CALL to main,
; and also have exit() calls.
; In that case enable the following:

        LCALL   main

_exit:
        PUBLIC  _exit
exit:
        PUBLIC  exit
_cstop:
        PUBLIC  _cstop          ; public for debuggers to set a breakpoint
        BR      _cstop                  ; keep on looping

EXTINT:
        PUSHA
        LCALL   RSIExtInt
        POPA
        RET

HSIINT:
  PUSHA
  LCALL   RSI_Hsi
  POPA
  RET

TIMER2_OVF:
  PUSHA
  LCALL   RSITimer2
  POPA
  RET

;; Button (INT7)
cseg at 0d0e0h
   br EXTINT

;; Interrupcio de "Data available" del HSI (INT2)
cseg at 0d040h
   br HSIINT

;; Interrupció de "Timer 2 Overflow" (INT12)
cseg at 0d180h
   br TIMER2_OVF

END
```

Lcd.c

```
#include <kd_sfrs.h>
#include "lcd.h"

/*****************************************************************************
```

```
;  Declaració de constants del display
;*********************************************************************************
;conexionat:
;       p1.4..p1.7 = D4..D7
;       p1.0 = RS
;       p1.2 = E
;       0 = R/W  siempre escribimos
*/

#define LCD_INS 0xfe
#define LCD_DAT 0x01
#define LCD_ENA 0x04

#define LCD_LINEA2 0xC0
#define LCD_LINEA1 0x80
#define LCD_CLRSCR 0x01                 /* RETARD DE 1.64 ms */

void WRITE_LCD_4BITS();

void INICIALITZA_LCD()
{
/*
;0      X000 0000B = 00H   ->              IOPORT1
;1      X000 0001B = 01H
;15     X000 1111B = 0FH
*/

  DELAY_10mS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x30 & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_10mS();

  wsr=0;
  ioport1=0x20 & LCD_INS; /* interface de 4 bits */
  WRITE_LCD_4BITS();
  DELAY_10mS();

/*AHORA YA PODEMOS USAR LA FUNCION DE INSTRUCION LCD*/
/*FUNCTION SET- 4 bits dades, 2 lines, caracter font*/

  INSTRUCCIO_LCD(0x28);

  /*DISPLAY ON/OFF-DISPLAY ON, CURSOR OFF, CURSOR NO BLINK*/
  INSTRUCCIO_LCD(0x0C);
```

```
  INSTRUCCIO_LCD(0x06);  /*ENTRY MODE SET-INCREMENT CURSOR, CURSOR MOVEMENT*/

  INSTRUCCIO_LCD(0x01);  /*CLEAR DISPLAY*/
  DELAY_10mS();
  /*ESTE RETARDO DEBERIA DE SER DE 1.64mS*/
}

void WRITE_LCD_4BITS()
{
/*AQUESTA FUNCIO ES NOMES PER US INTERN DE LA LLIBRERIA DEL DISPLAY*/

  asm{NOP;NOP;}
  wsr=0;
  ioport1 = ioport1 ^ LCD_ENA; /*XORB IOPORT1,#LCD_ENA*/

  asm{
  NOP;NOP;NOP;NOP;NOP;
  NOP;NOP;NOP;NOP;NOP;
  }

  wsr=0;
  ioport1 = ioport1 ^ LCD_ENA; /*XORB IOPORT1,#LCD_ENA*/
  asm{NOP;}
}


void INSTRUCCIO_LCD(unsigned char inst)
{
  wsr=0;
  ioport1=0xF0 & inst & LCD_INS ;                 /* RS=0 */
  WRITE_LCD_4BITS();
  inst<<=4;
  asm{
  NOP;NOP;
  NOP;NOP;NOP;
  }
  wsr=0;
  ioport1 =0xF0 & inst & LCD_INS;
  WRITE_LCD_4BITS();
  DELAY_40uS();
}

void DADA_LCD(unsigned char dato)
{
  wsr=0;
  ioport1=0xF0 & dato; /*parte alta primero*/
  ioport1=ioport1|LCD_DAT;   /*RS=1*/
  WRITE_LCD_4BITS();
  dato<<=4;
  asm{
  NOP;NOP;
  NOP;NOP;NOP;
  }
  wsr=0;
  ioport1=0x0F0 & dato;
  ioport1=ioport1|LCD_DAT;
```

```
  WRITE_LCD_4BITS();
  DELAY_40uS();
}

void CLRSCR_LCD()
{
  INSTRUCCIO_LCD(LCD_CLRSCR);
  DELAY_10mS();
}

void DELAY_10mS()
{
  unsigned int TIMER_LCD;

  TIMER_LCD=2000;
  while(TIMER_LCD!=0)
  {
     TIMER_LCD--;
  }
}

void DELAY_40uS()
{
  unsigned int TIMER_LCD;

  TIMER_LCD=40;
  while(TIMER_LCD!=0)
  {
   TIMER_LCD--;
  }
}

void GOTOXY(unsigned char x, unsigned char y)
/*   1<=x<=16  ,  1<=Y<=2 */
{
   unsigned char gotopos;
   gotopos=0x80 | ((y-1)*0x40) | (x-1);
   INSTRUCCIO_LCD(gotopos);
}
```

---

lcd.inc

```
;*******************************************************************************
;  Declaració de constants del display
;*******************************************************************************
;conexionat:
;       p1.4..p1.7 = D4..D7
;       p1.0 = RS
;       p1.2 = E
;       0 = R/W
LCD_INS SET 11111110B
LCD_DAT SET 00000001B
LCD_ENA SET 00000100B
```

```
LCD_LINEA2 SET 0C0H
LCD_LINEA1 SET 080H
LCD_CLRSCR SET 001H                     ;RETARDO DE 1.64 mS

;*********************************************************************************************
INICIALITZA_LCD:;*****************************************************************************
;*********************************************************************************************

;0      X000 0000B = 00H                ->              IOPORT1
;1      X000 0001B = 01H
;15     X000 1111B = 0FH


  PUSH WSR
  LDB WSR,#00h

  CALL DELAY_10mS
  CALL DELAY_10mS

  LDB IOPORT1,#30H AND LCD_INS
  CALL WRITE_LCD_4BITS
  CALL DELAY_10mS

  LDB IOPORT1,#30H AND LCD_INS
  CALL WRITE_LCD_4BITS
  CALL DELAY_10mS

  LDB IOPORT1,#30H AND LCD_INS
  CALL WRITE_LCD_4BITS
  CALL DELAY_10mS

  LDB IOPORT1,#20H AND LCD_INS
  CALL WRITE_LCD_4BITS
  CALL DELAY_10mS

;AHORA YA PODEMOS USAR LA FUNCION DE INSTRUCION LCD
  LDB bAux,#28H                         ;FUNCTION SET- 4 bits dades, 2 lines, caracter font
  CALL INSTRUCCIO_LCD

  LDB bAux,#0FH                         ;DISPLAY ON/OFF-DISPLAY ON, CURSON ON, CURSOR BLINK
  CALL INSTRUCCIO_LCD

  LDB bAux,#06H                         ;ENTRY MODE SET-INCREMENT CURSOR, CURSOR MOVEMENT
  CALL INSTRUCCIO_LCD

  LDB bAux,#01H                         ;CLEAR DISPLAY-
  CALL INSTRUCCIO_LCD
  CALL DELAY_10mS                       ;ESTE RETARDO DEBERIA DE SER DE 1.64mS

  POP WSR
  RET

;*********************************************************************************************
WRITE_LCD_4BITS:;*****************************************************************************
;*********************************************************************************************
;AQUESTA FUNCIO ES NOMES PER US INTERN DE LA LLIBRERIA DEL DISPLAY
```

```
  NOP
  NOP
  XORB IOPORT1,#LCD_ENA
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  NOP
  XORB IOPORT1,#LCD_ENA
  NOP
  RET

;*********************************************************************************************
INSTRUCCIO_LCD:;*****************************************************************************
;*********************************************************************************************
  PUSH WSR
  LDB WSR,#00H
  LDB IOPORT1,#0F0H
  ANDB IOPORT1,bAux
  ANDB IOPORT1,#LCD_INS                 ;RS=0
  CALL WRITE_LCD_4BITS

  SHL bAux,#4H
  NOP
  NOP
  NOP
  NOP
  NOP

  LDB IOPORT1,#0F0H
  ANDB IOPORT1,bAux
  ANDB IOPORT1,#LCD_INS
  CALL WRITE_LCD_4BITS
  CALL DELAY_40uS
  POP WSR
  RET

;*********************************************************************************************
DADA_LCD:;**********************************************************************************
;*********************************************************************************************
  PUSH WSR
  LDB WSR,#00H
  LDB IOPORT1,#0F0H
  ANDB IOPORT1,bAux                   ;parte alta primero
  ORB IOPORT1,#LCD_DAT                ;RS=1
  CALL WRITE_LCD_4BITS

  SHL bAux,#4H
  NOP
  NOP
  NOP
  NOP
  NOP
```

```
  LDB IOPORT1,#0F0H
  ANDB IOPORT1,bAux
  ORB IOPORT1,#LCD_DAT
  CALL WRITE_LCD_4BITS
  CALL DELAY_40uS
  POP WSR
  RET


;*********************************************************************************************
CLRSCR_LCD:;*********************************************************************************
;*********************************************************************************************
  LDB bAux,#LCD_CLRSCR
  CALL INSTRUCCIO_LCD
  CALL DELAY_10mS
  RET

;*********************************************************************************************
DELAY_10mS:;*********************************************************************************
;*********************************************************************************************
  LD TIMER_LCD,#2000D
DELAY_10mS_NEXT:
  DEC TIMER_LCD
  CMP TIMER_LCD,0
  JNE DELAY_10mS_NEXT
  RET


;*********************************************************************************************
DELAY_40uS:;*********************************************************************************
;*********************************************************************************************
  LD TIMER_LCD,#40D
DELAY_40uS_NEXT:
  DEC TIMER_LCD
  CMP TIMER_LCD,0
  JNE DELAY_10mS_NEXT
  RET


;*********************************************************************************************
_GOTOXY MACRO X,Y;***************************************************************************
;*********************************************************************************************
;   1&lt;=x&lt;=16  ,  1&lt;=Y&lt;=2
   LDB bAux,#080H OR ((Y-1)*040H) OR (X-1)
   CALL INSTRUCCIO_LCD
   endm
```

Control de versiones concurrente:

Para coordinar y versionar el código de la práctica hemos usado un repositorio de software llamado SubVersion (http://subversion.tigris.org). Esta herramienta nos ha facilitado mucho el mantenimiento del código entre las tres personas del grupo de trabajo.

Breve introducción a Subversion:

Un sistema de control de versions (Version Control System) és una aplicació que s'utilitza per a mantenir un conjunt d'arxius en un servidor central (anomenat repositori) i permetre l'accés a ells de forma concurrent des de diversos clients.

SVN (SubVersioN), com ja us podeu imaginar, és un sistema de control de versions.

- És lliure.
- És bastant fàcil d'usar.
- És més que suficient pel que necessitarem, ja que no farem ús de branques, merges, changesets, etc.

Les funcions principals (més usades) d'aquest tipus de programes són:

- Facilitat per obtenir sempre els arxius més recents.
- Mescla automàtica de canvis, en cas que dos desenvolupadors estiguin modificant el mateix arxiu.
- Possibilitat de tornar a versions més antigues de qualsevol arxiu.
- Emmagatzenament de tots els canvis amb explicacions de cadascun.

En altres paraules, a nosaltres ens pot facilitar la feina perquè no haurem de preocupar-nos de quí te els arxius més recents, de comprovar quins canvis hi ha entre els fitxers de cadascú, etc. És a dir, tots els arxius finals de la pràctica estaran guardats en una única màquina, a la qual podrem accedir des de qualsevol punt per a baixar-nos el nostre codi o per a fer modificacions.

Licencia: