Código de los programas realizados

A. Código en Wincupl

A.1. Conexión de un display LCD al bus de datos y direcciones

Archivo mipalpin2.pld

El código correspondiente a este archivo representa la implementación de un contador de 5 bits y una máquina de estados para la obtención de la señal XREADY.

```
Name       mipalpin2;
Partno     atf750c;
Date       03/9/05;
Revision   01;
Designer   Yad PLD;
Company    Atmel Corporation;
Assembly   U00;
Location   Sevilla;
Device     p22v10 ;

PIN     24  = Vdd;
PIN     1   = CLK;
PIN     2    = m0;
PIN     3    = m1;
PIN     4    = m2;
PIN     5    = m3;
PIN     6    = m4;
PIN     7    = !RES_ASINC;
PIN     8    = N_XWE;
PIN     9    = N_X0CS;
PIN     19   = reset;
PIN     18   = q0;
PIN     17   = q1;
PIN     16   = q2;
PIN     15   = q3;
PIN     23   = q4;
PIN     22   = qout;      /* Senal de cuenta y N_XREADY de salida*/
PIN     12   = Vss;
PIN     20   = XCS;
PIN     14   = XREADY;
PIN     21   = qsec;           /* Para implementar el otro bi de la mqestado*/


/* variables intermedias del contador*/
reset   = ( q0 $ m0 )# (q1 $ m1 )#( q2 $ m2 )#( q3 $ m3 )#( q4 $ m4 );

/* Ecuaciones intermedias de la maquina de estado*/

XCS = !(N_XWE # N_X0CS);
XREADY = !qout;

/* ecuaciones logicas del contador*/

q0.ar = RES_ASINC;
q1.ar = RES_ASINC;
q2.ar = RES_ASINC;
q3.ar = RES_ASINC;
q4.ar = RES_ASINC;
qout.ar = RES_ASINC;
qsec.ar = RES_ASINC;

q0.d = reset & (q0 $ qout);
q1.d = reset & (q1 $ (q0&qout));
q2.d = reset & (q2 $ (q1&q0&qout));
q3.d = reset & (q3 $ (q2&q1&q0&qout));
```

```
q4.d = reset & (q4 $ (q3&q2&q1&q0&qout));


/*Ecuaciones que describen el funcionamiento de la maquina de estado*/

qsec.d = !XCS # (qsec & !reset);
qout.d = (qout & reset) # (qsec & XCS & reset);
```

A.2. Conexión de un teclado al DSP

Archivo tecl2.pld

En este archivo se implementa el muestreo de las filas del teclado, así como la posterior codificación de las teclas recibidas.

```
Name     tecl2 ;
PartNo   00 ;
Date     23/06/05 ;
Revision 01 ;
Designer Engineer ;
Company  Atmel ;
Assembly None ;
Location  ;
Device   p22v10 ;


PIN     24  = Vdd;
PIN     1   = CLK;
PIN     2    = R_S;
PIN     5   = C1;
PIN     6   = C2;
PIN     7   = C3;
PIN     18  = q0;
PIN     17  = q1;
PIN     16  = q2;
PIN     15  = q3;
PIN     12  = Vss;
PIN     22  = q5;
PIN     21  = q6;
PIN     20  = q7;
PIN     19  = q8;
PIN     23  = hab;
PIN     14  = E;

/*Ecuaciones intermedias*/

hab = C1 # C2 # C3;
E = C1&C2 #  C1&C3 # C2&C3;

/*Ecuaciones de funcionamiento*/


q0.d = !R_S # q3;
q1.d = R_S & q0;
q2.d = R_S & q1;
q3.d = R_S & q2;


/*Identificacion de teclas */


q5.d= (q0 & C1 # q0 & C3 # q1 & C2 # q2 & C1 # q2 & C3 # q3 & C2 # E) &hab # q5 & !hab;


q6.d = (q0 & C2 # q0 & C3 # q1 & C3 # q2 & C1 # q3 & C1 # q3 & C2 # E) &hab # q6 & !hab;
```

```
q7.d = (q1 & C1 # q1 & C2 # q1 & C3 # q2 & C1 # q3 & C3 # E) &hab # q7 & !hab;

q8.d = (q2 & C2 # q2 & C3 #  q3 & C1 # q3 & C2 # q3 & C3 # E) &hab # q8 & !hab;
```

B. Código C

B.1. Conexión de un display LCD al bus de datos y direcciones

Archivo ppallcd2.

El presente código muestra la inicialización del DSP y la utilización de las distintas funciones que permiten escribir en el lcd.

```
#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   //DSP281x Examples Include File

extern unsigned int *LCD_CONTROL;
extern unsigned int *LCD_DATO;

void main(void)
{

/*
**Inicializar Sistema de Control:
**PLL, WatchDog,  habilitar Relojes de periféricos
**Esta función se encuentra en DSP281x_SysCtrl.c
*/
InitSysCtrl();

//Borrar interrupciones e inicializar el vector de la PIE
DINT;

/*
**Inicializar los registros de control de PIE a sus estados por
**defecto. El estado por defecto es todas las interrupciones de PIE
** deshabilitadas y flags borradas
**Esta función se encuentra en el archivo DSP281x_PieCtrl.c
*/
```

```
InitPieCtrl();

//Deshabilitar interrupciones de la CPU y sus correspondientes flags.
IER = 0x0000;
IFR = 0x0000;

/*Inicialización de la tabla del vector de la PIE con punteros a las
**rutinas de servicio de interrupción. Se encuentra en
**DSP281x_DefaultIsr.c y DSP281x_PieVect.c.
*/
InitPieVectTable();

//Configuración del XCLKOUT
XintfRegs.XINTCNF2.bit.XTIMCLK=1;
XintfRegs.XINTCNF2.bit.CLKMODE=0;
XintfRegs.XTIMING0.all=0xFFFFFFFF;


//Inicialización del LCD
LCD_Init();
delay_largo();

LCD_cursor_col (5);
LCD_DisplayCharacter ('A');

bucle_retrasos();

LCD_cursor_col (15);
LCD_DisplayCharacter ('E');

bucle_retrasos();

LCD_Cursor (2,2);
LCD_DisplayCharacter ('I');


LCD_DisplayString (1,7, "suerte");
LCD_DisplayString (2,5, "ey que haces");

bucle_retrasos();
```

```
LCD_Clear ();
LCD_DisplayRow (1, "otra frase nueva");

bucle_retrasos();

LCD_DisplayScreen ("ahora escribo toda la pantalla");

bucle_retrasos();

LCD_WipeOnLR("En un lugar de la Mancha de ...");

bucle_retrasos();

LCD_WipeOnRL("cuyo nombre no quiero acordarme.");

bucle_retrasos();

LCD_WipeOffLR();

bucle_retrasos();

LCD_WipeOnRL("y ahora borro en el otro sentido");

bucle_retrasos();

LCD_WipeOffRL();
LCD_DisplayString (2,7, "adios");
LCD_DisplayCharacter (4);


}
```

Archivo lcdrev2.c

Este archivo recoge todas las funciones desarrolladas para el uso del lcd, desde su inicialización, órdenes de escritura de datos y direcciones y diversas rutinas para escribir mensajes en la pantalla.

Se presentan únicamente los prototipos de las funciones en él desarrolladas y que pueden consultarse en el archivo palunion.c

```
/*
** Prototipos de las funciones
*/

static void LCD_WriteControl (unsigned int data);
static void LCD_WriteData (unsigned char data);
void LCD_Init (void);
void LCD_Clear (void);
void LCD_Home (void);
void LCD_DisplayCharacter (char a_char);
void LCD_Cursor (int row, int column);
void LCD_DisplayString (int row, int column, char *string);
void LCD_cursor_col (int col);
void LCD_DisplayRow (int row, char *string);
void LCD_DisplayScreen (char *ptr);
void LCD_WipeOnLR (char *ptr);
void LCD_WipeOnRL (char *ptr);
void LCD_WipeOffLR (void);
void LCD_WipeOffRL (void);
void LCD_CursorLeft (void);
void LCD_CursorRight (void);
void LCD_CursorOn (void);
void LCD_CursorOff (void);
void LCD_DisplayOff (void);
void LCD_DisplayOn (void);
void LCD_DefineChar (char address, const unsigned char *pattern);
void retraso(int pasos);
void bucle_retrasos(void);
void delay_largo(void);
```

B.2. Conexión de un display al DSP por medio del SPI

Archivo ppalspi.c

Se presenta un archivo principal muy similar al del apartado anterior, puesto que el objetivo final de esta aplicación es la misma, comunicar el DSP

con el lcd. Las variaciones se encuentran en las funciones de inicialización del módulo, ya que para mostrar su funcionamiento, se usan los mismos ejemplos.

```
#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   // DSP281x Examples Include File



// Prototipos de las funciones definidas en este archivo
void spi_xmit(unsigned int a);
void spi_fifo_init(void);
void spi_init(void);

void main(void)
{
// Inicialización del sistema de control, PLL, Watchdog, periféricos…
// Función perteneciente al archive DSP281x_SysCtrl.c
   InitSysCtrl();

// Pines del GPIO asociados al SPI
   EALLOW;
   GpioMuxRegs.GPFMUX.all=0x000F;  // GPIOs  asociados a pines SPI
   GpioMuxRegs.GPFDIR.bit.GPIOF4=1;   //Configurados como salidas
   GpioMuxRegs.GPFDIR.bit.GPIOF5=1;  // Serán Enable y R_S del lcd

   EDIS;

// Limpieza de interrupciones e inicialización de la PIE
   DINT;

// Función  perteneciente al archivo DSP281x_PieCtrl.c
   InitPieCtrl();

// Deshabilitación de interrupciones de la CPU y limpieza de banderas
   IER = 0x0000;
   IFR = 0x0000;

   InitPieVectTable();


   spi_fifo_init();       // Inicialización de la FIFO del Spi
```

```
   spi_init();            // inicialización del SPI

// Valores iniciales para las salidas de propósito general
      GpioDataRegs.GPFDAT.bit.GPIOF4=0;
      GpioDataRegs.GPFDAT.bit.GPIOF5=0;

retraso(500);

LCD_Init2();

bucle_retrasos();

LCD_cursor_col2 (5);
LCD_DisplayCharacter2 ('A');

bucle_retrasos();
LCD_cursor_col2 (15);
LCD_DisplayCharacter2 ('E');

bucle_retrasos();
LCD_Cursor2 (2,2);
LCD_DisplayCharacter2 ('I');

bucle_retrasos();
LCD_DisplayString2 (1,7, "suerte");

bucle_retrasos();
LCD_DisplayString2 (2,5, "ey que haces");

bucle_retrasos();
LCD_Clear2 ();

bucle_retrasos();
LCD_DisplayRow2 (1, "otra frase nueva");

bucle_retrasos();
LCD_DisplayScreen2 ("ahora escribo toda la pantalla");

bucle_retrasos();
LCD_WipeOnLR2("En un lugar de la Mancha de ...");
```

```
bucle_retrasos();
LCD_WipeOnRL2("cuyo nombre no quiero acordarme.");

bucle_retrasos();
LCD_WipeOffLR2();

bucle_retrasos();
LCD_WipeOnRL2("y ahora borro en el otro sentido");
bucle_retrasos();

LCD_WipeOffRL2();
LCD_DisplayString2 (2,7, "adios");
LCD_DisplayCharacter2 (4);

}


/*
** Spi_init: Inicialización del SPI, asignación de valores a registros
** de configuración.
*/

void spi_init()
{
      SpiaRegs.SPICCR.all =0x0007;  // Reset, flanco subida,16-bit
      SpiaRegs.SPICTL.all =0x000e;  // master mode, fase normal,
                              //talk habilitado y no SPI interrupcion
      SpiaRegs.SPIBRR =0x0000;
      SpiaRegs.SPICCR.all =0x0097;       // Reactivo SPI tras Reset
      SpiaRegs.SPIPRI.bit.FREE = 1;      // Breakpoints no afectan tx
}

/*
** Spi_xmit: transmisión de un dato por el spi
*/

void spi_xmit(unsigned int a)
{
    SpiaRegs.SPITXBUF=a;
}
```

```
/*
** Spi_fifo_init: Inicialización de la FIFO del SPI, asignación de
**valores a registros de configuración.
*/
void spi_fifo_init()

{
    SpiaRegs.SPIFFTX.all=0xE040;
    SpiaRegs.SPIFFRX.all=0x204f;
    SpiaRegs.SPIFFCT.all=0x0;
}
```

Archivo palspi.c

Las funciones pertenecientes a este archivo son, en su mayoría, las mismas que se encuentran en lcdrev2.c por razones evidentes. Difieren con respecto a ellas, las dos funciones de bajo nivel y, en consecuencia, todas las demás. Pueden consultarse en el archivo palunion.c

B.3. Conexión de un teclado al DSP

Archivo ppaltec.c

En este archivo aparece la inicialización del sistema y un sencillo bucle infinito en el que, cada vez que se recoge un dato, se almacena en una variable, de forma que, puede verse en simulación el resultado de las pruebas.

```
#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   //DSP281x Examples Include File


unsigned int *LCD_PUNTERO =(unsigned int *)0x4000;


void main(void)
{
unsigned int valor=0;
*LCD_PUNTERO=0;
```

```
/*
**Inicializar Sistema de Control:
**PLL, WatchDog,  habilitar Relojes de periféricos
**Esta función se encuentra en DSP281x_SysCtrl.c
*/
InitSysCtrl();

//Borrar todas las interrupciones e inicializar el vector de la PIE
DINT;

/*
**Inicializar los registros de control de PIE
**El estado por defecto es todas las interrupciones de PIE
**deshabilitadas y flags borradas
**Esta función se encuentra en el archivo DSP281x_PieCtrl.c
*/
InitPieCtrl();

//Deshabilitar interrupciones de la CPU y sus correspondientes flags.
IER = 0x0000;
IFR = 0x0000;

/*Inicialización de la tabla del vector de la PIE con punteros a las
**rutinas de interrupción de servicio. Se encuentra en
**DSP281x_DefaultIsr.c y DSP281x_PieVect.c.
*/
InitPieVectTable();

//Configuración del XCLKOUT
XintfRegs.XINTCNF2.bit.XTIMCLK=1;
XintfRegs.XINTCNF2.bit.CLKMODE=0;
XintfRegs.XTIMING0.all=0xFFFF3FFF;


while (1)
      {
      asm  (  "  NOP" );
      valor = *LCD_PUNTERO;
      }

}
```

B.4. Uso de los convertidores analógico/digital

Archivo Example_281xAdcSoc.c

El presente código permite la demostración del funcionamiento del ADC, ya que, la ejecución del mismo, permite obtener los valores resultantes de la conversión a digital de los valores correspondientes a una senoide introducida como entrada al DSP por el pin ADCINA3.

Para ello, se lleva a cabo la inicialización del módulo mediante la configuración de los registros oportunos, para conseguir la acción deseada (conversión del secuenciador 1 cada vez que se reciba una interrupción del timer 1 de EVA).

Se configura también una salida de propósito general para medir cuál es la frecuencia de conversión del DSP y comprobar de este modo que cumple lo esperado.

```
#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   // DSP281x Examples Include File

// Prototipos de las funciones usadas en este archivo
interrupt void adc_isr(void);

//Variables globales
unsigned int LoopCount;
unsigned int ConversionCount;
unsigned int Voltage1[1000];


main()
{

// Inicialización del sistema de control, PLL, Watchdog,  relojes de
//los periféricos. Función perteneciente al archivo DSP281x_SysCtrl.c
   InitSysCtrl();

// HSPCLK configurado como SYSCLKOUT / 6 (25Mhz para  SYSCLKOUT de
//150Mhz)
```

```
   EALLOW;
   SysCtrlRegs.HISPCP.all = 0x3;  // HSPCLK = SYSCLKOUT/6
   EDIS;

//Habilita una salida de propósito general para hacer pruebas
   EALLOW;
   GpioMuxRegs.GPFDIR.bit.GPIOF6=1;
   EDIS;

// Limpieza de todas las interrupciones e inicialización de la PIE
// Deshabilitar las interrupciones de la CPU
   DINT;

//Función perteneciente al archivo DSP281x_PieCtrl.c
   InitPieCtrl();

// Deshabilitación de interrupciones de CPU y limpieza de las banderas
   IER = 0x0000;
   IFR = 0x0000;

// Función perteneciente al archivo DSP281x_PieVect.c.
   InitPieVectTable();

// Mapeado de las rutinas de interrupción del archivo
   EALLOW;
   PieVectTable.ADCINT = &adc_isr;

   EDIS;
//Inicialización del ADC
  InitAdc();

// Habilitación de interrupciones ( ADCINT en PIE )
   PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
   IER |= M_INT1;      // Habilitación de la interrupción 1 de la CPU
   EINT;               // Habilitación de la interrupción global INTM
   ERTM;        // Habilitación de la interrupción de tiempo real DBGM

  //Valores iniciales
   GpioDataRegs.GPFDAT.bit.GPIOF6=0;
   LoopCount = 0;
   ConversionCount = 0;
```

```
// Configuración del ADC
AdcRegs.ADCMAXCONV.all = 0x0000;        //Una conversión en el SEQ1
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x3;   // ADCINA3 es la entrada
AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 1;   // Habilita EVASOC en el SEQ1
AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;   // Habilita la interrupción de
                                                  //SEQ1 todo EOS

// Configuración EVA
// Se asume reloj de EVA previamente habilitado con InitSysCtrl();
   EvaRegs.T1CMPR = 0x0080;           // Valor de comparación del T1
   EvaRegs.T1PR = 0x04E2;             // Valor del registro período
   EvaRegs.GPTCONA.bit.T1TOADC = 1;   // Se habilita EVASOC en EVA
   EvaRegs.T1CON.all = 0x1042;        // Se habilita comparación de T1
                                                  //(cuenta creciente)


// Se espera interrupción del ADC
   while(1)
   {
      LoopCount++;
   }

}
```

B.5. Unión de los distintos módulos desarrollados

Archivo ppallunion.c

Se presenta la fusión de los códigos de las aplicaciones ya descritas.

En este archivo puede verse la función principal y las correspondientes a la inicialización de los módulos usados.

```
#define MYMASK 0x3F     //máscara para mostrar n°

#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   //DSP281x Examples Include File

extern unsigned int *LCD_CONTROL;
extern unsigned int *LCD_DATO;
```

```
extern unsigned int *LCD_PUNTERO;


//Prototipo de funciones asociadas al SPI
void spi_xmit(unsigned int a);
void spi_fifo_init(void);
void spi_init(void);

//Prototipo de función asociada al ADC
interrupt void adc_isr(void);
unsigned int LoopCount;
unsigned int ConversionCount;

#pragma DATA_SECTION(Voltage1,"misvariables")
#pragma DATA_SECTION(Voltage2,"misvariables")
#pragma DATA_SECTION(Voltage3,"misvariables")
#pragma DATA_SECTION(Voltage4,"misvariables")
#pragma DATA_SECTION(Voltage5,"misvariables")
#pragma DATA_SECTION(Voltage6,"misvariables")
#pragma DATA_SECTION(Voltage7,"misvariables")
#pragma DATA_SECTION(Voltage8,"misvariables")

unsigned int Voltage1[1000];
unsigned int Voltage2[1000];
unsigned int Voltage3[1000];
unsigned int Voltage4[1000];
unsigned int Voltage5[1000];
unsigned int Voltage6[1000];
unsigned int Voltage7[1000];
unsigned int Voltage8[1000];

void main(void)
{
int i=0;
unsigned int valor;
/*
**Inicializar Sistema de Control:
**PLL, WatchDog,  habilitar Relojes de periféricos
**Esta función se encuentra en DSP281x_SysCtrl.c
*/
```

```
InitSysCtrl();
 //En el módulo adc: HSPCLK = SYSCLKOUT/6
EALLOW;
   SysCtrlRegs.HISPCP.all = 0x3;
EDIS;
//Borrar todas las interrupciones e inicializar la PIE
DINT;

/*
**Inicializar registros de control de PIE a estados por defecto:
**todas las interrupciones de PIE deshabilitadas
**y flags borradas.
**Esta función pertenece al archivo DSP281x_PieCtrl.c
*/
InitPieCtrl();



EALLOW;
   GpioMuxRegs.GPFMUX.all=0x000F; //GPIOs como pines SPI
   GpioMuxRegs.GPFDIR.bit.GPIOF4=1;//GPIOs de salida
   GpioMuxRegs.GPFDIR.bit.GPIOF5=1;
   GpioMuxRegs.GPFDIR.bit.GPIOF6=1;
EDIS;

//Deshabilitar interrupciones de CPU y correspondientes flags.
IER = 0x0000;
IFR = 0x0000;

/*Inicialización de la tabla del vector de la PIE con
**punteros a las rutinas de interrupción de servicio. Se
**encuentra en DSP281x_DefaultIsr.c y DSP281x_PieVect.c.
*/
InitPieVectTable();

      spi_fifo_init();  // Inicialización de FIFO Spi
      spi_init();         // Inicialización SPI

//Asociación de la rutina de interrupción al vector de PIE
EALLOW;
   PieVectTable.ADCINT = &adc_isr;
EDIS;
```

```
//Configuración del XCLKOUT para conexión lcd-bus
XintfRegs.XINTCNF2.bit.XTIMCLK=1;
XintfRegs.XINTCNF2.bit.CLKMODE=0;
XintfRegs.XTIMING0.all=0xFFFFFFFF;


   InitAdc();  //Inicialización ADC


// Habilitación de la interrupción del ADC en PIE
   PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
   IER |= M_INT1; //Habilitación CPU Interrupt 1
   EINT;          //Habilitación Global interrupt INTM
   ERTM;          //Habilitación  Global realtime interrupt DBGM

      GpioDataRegs.GPFDAT.bit.GPIOF4=0;
      GpioDataRegs.GPFDAT.bit.GPIOF5=0;
      GpioDataRegs.GPFDAT.bit.GPIOF6=0;
      LoopCount = 0;
      ConversionCount = 0;

// Configuración ADC
   AdcRegs.ADCMAXCONV.all = 0x0007;       // 8 conversiones en SEQ1

   AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; //ADCINA0 es la 1ªconv.
   AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; //ADCINA1 es la 2ª conv.
   AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2;
   AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3;
   AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4;
   AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x5;
   AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x6;
   AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0x7;  //ADCINA7 es la 8ª conv.
   AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 1;  //EVASOC inicia el SEQ1
   AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; //Habilitación de interrup.
                                         //en SEQ1 (todo EOS)

// Configuración EVA
// EVA Clock habilitado en InitSysCtrl();
   EvaRegs.T1CMPR = 0x0080;               // Valor de comparación
   EvaRegs.T1PR = 0x04E2;              // Valor del registro periodo
```

```
   EvaRegs.GPTCONA.bit.T1TOADC = 1;    // Habilitación EVASOC en EVA
   EvaRegs.T1CON.all = 0x1042;         // Habilitación de comparación
                                          //timer 1 (cuenta creciente)

//Puesta en funcionamiento del módulo del spi

bucle_retrasos();
LCD_Init2();

bucle_retrasos();

LCD_cursor_col2 (5);

LCD_DisplayCharacter2 ('A');

bucle_retrasos();

LCD_cursor_col2 (15);
LCD_DisplayCharacter2 ('E');

bucle_retrasos();

LCD_Cursor2 (2,2);
LCD_DisplayCharacter2 ('I');

bucle_retrasos();

LCD_DisplayString2 (1,7, "suerte");

bucle_retrasos();

LCD_DisplayString2 (2,5, "ey que haces");

bucle_retrasos();

LCD_Clear2 ();

bucle_retrasos();

LCD_DisplayRow2 (1, "otra frase nueva");
```

```
bucle_retrasos();


LCD_DisplayScreen2 ("ahora escribo toda la pantalla");

bucle_retrasos();

LCD_WipeOnLR2("En un lugar de la Mancha de ...");

bucle_retrasos();

LCD_WipeOnRL2("cuyo nombre no quiero acordarme.");

bucle_retrasos();


LCD_WipeOffLR2();

bucle_retrasos();

LCD_WipeOnRL2("y ahora borro en el otro sentido");

bucle_retrasos();

LCD_WipeOffRL2();
LCD_DisplayString2 (2,7, "adios");
LCD_DisplayCharacter2 (4);


//Puesta en funcionamiento del módulo adc
// Se espera interrupción del adc
   while(LoopCount<60000)
   {
      LoopCount++;
   }


//Puesta en funcionamiento del lcd conectado al bus
bucle_retrasos();

LCD_Init();
```

```
delay_largo();

LCD_cursor_col (3);
LCD_DisplayCharacter ('U');

bucle_retrasos();

LCD_cursor_col (11);
LCD_DisplayCharacter ('Y');

bucle_retrasos();

LCD_Cursor (2,7);
LCD_DisplayCharacter ('Z');

bucle_retrasos();

LCD_DisplayString (1,5, "boo!");
LCD_DisplayString (2,5, "amigos");

bucle_retrasos();

LCD_DisplayScreen ("400 aniversario de 'El Quijote'");

bucle_retrasos();

LCD_WipeOnLR("En un lugar de la Mancha de ...");

bucle_retrasos();

LCD_WipeOnRL("cuyo nombre no quiero acordarme.");

bucle_retrasos();

LCD_WipeOffLR();

bucle_retrasos();

LCD_WipeOnRL("y ahora borro en el otro sentido");

bucle_retrasos();
```

```
LCD_WipeOffRL();
LCD_cursor_col (21);
LCD_DisplayCharacter (0);
LCD_DisplayCharacter (1);
LCD_DisplayString (2,7, "sino");
LCD_DisplayCharacter (0);
LCD_DisplayCharacter (1);
bucle_retrasos();


//Recepción de teclas del teclado
while(1)
      {
      valor= *LCD_PUNTERO;
      valor= valor & MYMASK;
      switch(valor){
            case 0x3A: valor= 0x2A;break;
            case 0x3B: valor= 0x30;break;
            case 0x3C: valor= 0x23;break;
            default:break;
            }
      LCD_DisplayCharacter(valor);
      for (i=1; i<1000;i++)
            retraso(9000);
      }
}


/*
**Rutina de interrupción del ADC: guarda los valores convertidos en la
**tabla Voltage correspondiente y por cada ciclo de conversiones, se
**invierte la salida del pin GPIOF6 y se limpian las banderas y
**actualizan los campos necesarios para la próxima interrupción
*/
interrupt void  adc_isr(void)
{

  Voltage1[ConversionCount] = AdcRegs.ADCRESULT0 >>4;
  Voltage2[ConversionCount] = AdcRegs.ADCRESULT1 >>4;
  Voltage3[ConversionCount] = AdcRegs.ADCRESULT2 >>4;
```

```
  Voltage4[ConversionCount] = AdcRegs.ADCRESULT3 >>4;
  Voltage5[ConversionCount] = AdcRegs.ADCRESULT4 >>4;
  Voltage6[ConversionCount] = AdcRegs.ADCRESULT5 >>4;
  Voltage7[ConversionCount] = AdcRegs.ADCRESULT6 >>4;
  Voltage8[ConversionCount] = AdcRegs.ADCRESULT7 >>4;

  GpioDataRegs.GPFDAT.bit.GPIOF6=!(GpioDataRegs.GPFDAT.bit.GPIOF6);

  //Si se llena la tabla, volver a empezar
  if(ConversionCount == 999)
  {
     ConversionCount = 0;
  }
  else ConversionCount++;

  //Reinicialización del ADC para próxima secuencia
  AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;         // Reset SEQ1
  AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;       // Limpiar bit INT SEQ1
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Ack interrupción a PIE

  return;
}


/*
**spi_init: Función de inicialización del SPI,
**asignación de valores a registros de configuración.
*/
void spi_init()
{
      SpiaRegs.SPICCR.all =0x0007;// Reset, flanco subida,16-bit
      SpiaRegs.SPICTL.all =0x000e;// master mode, fase normal,
                                  //talk habilitado y no SPI interrupcion
      SpiaRegs.SPIBRR =0x0000;
      SpiaRegs.SPICCR.all =0x0097;  // Reactivo SPI tras Reset
      SpiaRegs.SPIPRI.bit.FREE = 1; // Breakpoints no afectan tx
}

/*
**Spi_xmit: transmisión de un dato por el spi
*/
```

```
void spi_xmit(unsigned int a)
{
    SpiaRegs.SPITXBUF=a;
}

/*
** Spi_fifo_init: Inicialización de la FIFO del SPI, asignación de
**valores a registros de configuración.
*/
void spi_fifo_init()

{
    SpiaRegs.SPIFFTX.all=0xE040;
    SpiaRegs.SPIFFRX.all=0x204f;
    SpiaRegs.SPIFFCT.all=0x0;
}
```

Archivo palunion.c

En él se recoge el código correspondiente a las funciones que permiten mostrar el funcionamiento de los módulos vistos.

```
#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   // DSP281x Examples Include File


/*
** Asigna las direcciones HW al LCD.
*/
unsigned int *LCD_CONTROL=(unsigned int *)0x2000;
unsigned int *LCD_DATO=(unsigned int *)0x2001;
unsigned int *LCD_PUNTERO=(unsigned int *)0x4000;


/*
** Prototipos de las funciones
*/
static void LCD_WriteControl (unsigned int data);
static void LCD_WriteData (unsigned char data);
```

```
void LCD_Init (void);
void LCD_Clear (void);
void LCD_Home (void);
void LCD_DisplayCharacter (char a_char);
void LCD_Cursor (int row, int column);
void LCD_DisplayString (int row, int column, char *string);
void LCD_cursor_col (int col);
void LCD_DisplayRow (int row, char *string);
void LCD_DisplayScreen (char *ptr);
void LCD_WipeOnLR (char *ptr);
void LCD_WipeOnRL (char *ptr);
void LCD_WipeOffLR (void);
void LCD_WipeOffRL (void);
void LCD_CursorLeft (void);
void LCD_CursorRight (void);
void LCD_CursorOn (void);
void LCD_CursorOff (void);
void LCD_DisplayOff (void);
void LCD_DisplayOn (void);
void LCD_DefineChar (char address, const unsigned char *pattern);



static void LCD_WriteControl2 (unsigned int data);
static void LCD_WriteData2 (unsigned char data);
void LCD_Init2 (void);
void LCD_Clear2 (void);
void LCD_Home2 (void);
void LCD_DisplayCharacter2 (char a_char);
void LCD_Cursor2 (int row, int column);
void LCD_DisplayString2 (int row, int column, char *string);
void LCD_cursor_col2 (int col);
void LCD_DisplayRow2 (int row, char *string);
void LCD_DisplayScreen2 (char *ptr);
void LCD_WipeOnLR2 (char *ptr);
void LCD_WipeOnRL2 (char *ptr);
void LCD_WipeOffLR2 (void);
void LCD_WipeOffRL2 (void);
void LCD_CursorLeft2 (void);
void LCD_CursorRight2 (void);
void LCD_CursorOn2 (void);
```

```
void LCD_CursorOff2 (void);
void LCD_DisplayOff2 (void);
void LCD_DisplayOn2 (void);
void LCD_DefineChar2 (char address, const unsigned char *pattern);

void espera_39us(void);
void espera_43us(void);
void retraso(int pasos);
void bucle_retrasos(void);
void delay_largo(void);
void espera_almac(void);
void espera_lectura(void);
void delay_loop(void);

void error(void);

const unsigned char LCD_CHAR_UP_ARROW[] = {
      0x1f,       // ---11111
      0x1b,       // ---11011
      0x11,       // ---10001
      0x0a,       // ---01010
      0x1b,       // ---11011
      0x1b,       // ---11011
      0x1b,       // ---11011
      0x1f        // ---11111
};
const unsigned char LCD_CHAR_DOWN_ARROW[] = {
      0x1f,       // ---11111
      0x1b,       // ---11011
      0x1b,       // ---11011
      0x1b,       // ---11011
      0x0a,       // ---01010
      0x11,       // ---10001
      0x1b,       // ---11011
      0x1f        // ---11111
};
const unsigned char LCD_CHAR_TRADEMARK_T[] = {
      0x1f,       // ---11111
      0x04,       // ---00100
      0x04,       // ---00100
      0x04,       // ---00100
```

```
      0x00,       // ---00000
      0x00,       // ---00000
      0x00,       // ---00000
      0x00        // ---00000
};

const unsigned char LCD_CHAR_TRADEMARK_M[] = {
      0x11,       // ---10001
      0x1b,       // ---11011
      0x15,       // ---10101
      0x11,       // ---10001
      0x00,       // ---00000
      0x00,       // ---00000
      0x00,       // ---00000
      0x00        // ---00000
};
const unsigned char LCD_CARITA[] = {
      0x1f,     // ---11111
      0x00,       // ---00000
      0x0a,       // ---01010
      0x00,       // ---00000
      0x00,       // ---00000
      0x1f,       // ---11111
      0x0a,       // ---01010
      0x04        // ---00100
};


/*
**Funciones de manejo del display asociadas al spi
*/
/* -------- Rutinas de alto nivel del LCD asociado al spi---------- */
/*
**LCD_Init2: inicialización del lcd
*/
void LCD_Init2 (void)
{
      int i;
      asm( "  NOP");
      LCD_WriteControl2(0x38);
      LCD_WriteControl2(0x0E);
```

```
      LCD_WriteControl2(0x1);
      for(i=1;i<26;i++)
            {
            retraso(1000);
            }
      LCD_WriteControl2(0x06);
      asm( "  NOP");
// carga los caracteres definidos por el usuario en el LCD.

      LCD_DefineChar2 (0, LCD_CHAR_UP_ARROW);
      LCD_DefineChar2 (1, LCD_CHAR_DOWN_ARROW);
      LCD_DefineChar2 (2, LCD_CHAR_TRADEMARK_T);
      LCD_DefineChar2 (3, LCD_CHAR_TRADEMARK_M);
      LCD_DefineChar2 (4, LCD_CARITA);

}


/*
** LCD_Clear2: Limpia pantalla del LCD y coloca cursor al principio.
*/
void LCD_Clear2 (void)
{
      LCD_WriteControl2(0x01);
}


/*
** LCD_Home2: Sitúa el cursor del LCD en la fila1 y columna1.
*/
void LCD_Home2 (void)
{
      LCD_WriteControl2(0x02);
}


/*
** LCD_DisplayCharacter2: Escribe un carácter en la posición actual
**del cursor.
*/
void LCD_DisplayCharacter2 (char a_char)
```

```
{
      LCD_WriteData2 (a_char);
}


/*
** LCD_Cursor2: Sitúa el cursor en una fila y columna determinadas.
*/
void LCD_Cursor2 (int row, int column)
{
            LCD_WriteControl2 (0x80 +(row-1)*0x40+ column - 1);
}


/*
** LCD_DisplayString2: Escribe cadena a partir de posición del cursor
** especificada. La función que la llama debe comprobar la longitud de
** dicha cadena.
*/
void LCD_DisplayString2 (int row, int column, char *string)
{
      LCD_Cursor2 (row, column);
      while (*string)
            LCD_DisplayCharacter2 (*string++);
}


/*
** LCD_cursor_col2: Sitúa el cursor en la siguiente posición cambiando
** de fila si es necesario.
*/
void LCD_cursor_col2 (int col)
{
      if(col<17)
            {
            LCD_WriteControl2 (0x80 + col-1);
            }
      else
            {
            LCD_WriteControl2 (0x80 + 0x40+ col - 17);
            }
```

```
}


/*
** LCD_DisplayRow2: Escribe una línea en la fila especificada.
*/
void LCD_DisplayRow2 (int row, char *string)
{
      LCD_Cursor2 (row, 1);
      while (*string)
            LCD_DisplayCharacter2 (*string++);
}


/*
** LCD_DisplayScreen2: Escribe la pantalla completa.
*/
void LCD_DisplayScreen2 (char *ptr)
{
      LCD_DisplayRow2(1,ptr+ 0);
      LCD_DisplayRow2(2,ptr+16);

}


/*
** LCD_WipeOnLR2: Escribe una pantalla desplegándola de
**               izquierda a derecha.
*/
void LCD_WipeOnLR2 (char *ptr)
{

      int i=0;
      int j=0;
      while (*(ptr+i))
            i++;
      j=i;
      if(j<17)
            {
                  for (i=0; i<j; i++)
```

```
                        {
                        LCD_Cursor2(1,i+1);
                        LCD_DisplayCharacter2(*(ptr+i));
                        }
            }
      else
            {
            for (i=0; i<(j-16); i++)
                  {
                        LCD_Cursor2(1,i+1);
                        LCD_DisplayCharacter2(*(ptr+i));
                        LCD_Cursor2(2,i+1);
                        LCD_DisplayCharacter2(*(ptr+16+i));
                  }
            while(i<17)
                  {
                        LCD_Cursor2(1,i+1);
                        LCD_DisplayCharacter2(*(ptr+i));
                        i++;
                  }

            }
}


/*
** LCD_WipeOnLR2: Escribe una pantalla completa desplegándola de
**               derecha a izquierda.
*/
void LCD_WipeOnRL2 (char *ptr)
{
      int j;
      char i;
      for (i=16; i>0; i--) {
            LCD_Cursor2(1,i);
            LCD_DisplayCharacter2(*(ptr+ 0+i-1));
            LCD_Cursor2(2,i);
            LCD_DisplayCharacter2(*(ptr+16+i-1));
            for(j=0; j<200; j++)
                  retraso(1000);
            }
```

```
}


/*
** LCD_WipeOffLR2: Borra la pantalla de izqda a derecha.
*/
void LCD_WipeOffLR2 (void)
{
      int j;
      char i;
      for (i=1; i<17; i++) {
            #define BLOCK 0xff
            LCD_Cursor2(1,i);
            LCD_DisplayCharacter2(BLOCK);
            LCD_Cursor2(2,i);
            LCD_DisplayCharacter2(BLOCK);
            for(j=0; j<200; j++)
                  retraso(1000);

      }
}


/*
** LCD_WipeOffRL2: Borra la pantalla de derecha a izqda.
*/
void LCD_WipeOffRL2 (void)
{
      int j;
      char i;
      for (i=16; i>0; i--) {
            #define BLOCK 0xff
            LCD_Cursor2(1,i);
            LCD_DisplayCharacter2(BLOCK);
            LCD_Cursor2(2,i);
            LCD_DisplayCharacter2(BLOCK);
            for(j=0; j<200; j++)
                  retraso(1000);

      }
}
```

```
/*
** LCD_CursorLeft2: Mueve el cursor un caracter a la izquierda.
*/
void LCD_CursorLeft2 (void)
{
      LCD_WriteControl2 (0x10);
}


/*
** LCD_CursorRight2: Mueve el cursor un caracter a la derecha.
*/
void LCD_CursorRight2 (void)
{
      LCD_WriteControl2 (0x14);
}


/*
** LCD_CursorOn2: Enciende el cursor.
*/
void LCD_CursorOn2 (void)
{
      LCD_WriteControl2 (0x0f);
}


/*
** LCD_CursorOff2: Apaga el cursor.
*/
void LCD_CursorOff2 (void)
{
      LCD_WriteControl2 (0x0c);
}


/*
** LCD_DisplayOff2: Apaga el LCD.
*/
```

```
void LCD_DisplayOff2 (void)
{
      LCD_WriteControl2(0x08);
}


/*
** LCD_DisplayOn2: Enciende el LCD.
*/
void LCD_DisplayOn2 (void)
{
      LCD_WriteControl2(0x0c);
}


/*
** LCD_DefineCharacter2: Define el patrón de puntos para caracteres
** definidos por el usuario.
**
**  entradas: address = dirección del carácter (0x00-0x07)
**            pattern = puntero a array de 8-byte con patrón.
*/
void LCD_DefineChar2 (char address, const unsigned char *pattern)
{
      char i;
      LCD_WriteControl2(0x40 + (address << 3));

      for (i=0; i<8; i++)
      {
            LCD_WriteData2(*pattern++);

      }
}


/* -------- Rutinas de bajo nivel del LCD asociado al spi---------- */

/*
** LCD_WriteControl2: Escribe una instrucción de control para el LCD.
*/
static void LCD_WriteControl2 ( unsigned int data)
```

```
{
      unsigned int justif=0;
      unsigned int rdata;
      justif=data<<8; //justificación del dato a la izqda
      spi_xmit(justif);
      while(SpiaRegs.SPIFFRX.bit.RXFFST !=1) { }
      rdata = SpiaRegs.SPIRXBUF;
     if(rdata != data)  //Comparación dato rxdo y txdo
      error();

    GpioDataRegs.GPFDAT.bit.GPIOF4=0; //R_S=0 para control
    espera_almac();
      GpioDataRegs.GPFDAT.bit.GPIOF5=1; //Activación Enable lcd
    espera_lectura();
      GpioDataRegs.GPFDAT.bit.GPIOF5=0; //Desactivación Enable lcd
      espera_39us();

}

/*
** LCD_WriteData2: Escribe un byte de datos para el LCD.
*/
static void LCD_WriteData2 (unsigned char data)
{
      unsigned int justif=0;
      unsigned int rdata;
      justif=data<<8; //justificación del dato a izqda.
      spi_xmit(justif);
      while(SpiaRegs.SPIFFRX.bit.RXFFST !=1) { }
      rdata = SpiaRegs.SPIRXBUF;
     if(rdata != data) //Comparación dato txdo y rxdo
      error();

      GpioDataRegs.GPFDAT.bit.GPIOF4=1; //R_S=1 para dato
      espera_almac();
      GpioDataRegs.GPFDAT.bit.GPIOF5=1; //Activación Enable lcd
      espera_lectura();
      GpioDataRegs.GPFDAT.bit.GPIOF5=0; //Desactivación Enable lcd
      espera_43us();

}
```

```
/* --------Rutinas de alto nivel del lcd asociado al bus-------- */

/*
** LCD_Init: Inicializa el LCD.
*/
void LCD_Init (void)
{
      int i;

      *LCD_CONTROL=0x38;
      retraso(3000);
      *LCD_CONTROL=0x0E;
      retraso(3000);
      *LCD_CONTROL=0x1;
      for(i=1;i<100;i++)
            {
            retraso(1000);
            }
      *LCD_CONTROL=0x06;
      retraso(1000);

// carga los caracteres definidos por el usuario en el LCD.

      LCD_DefineChar (0, LCD_CHAR_UP_ARROW);
      LCD_DefineChar (1, LCD_CHAR_DOWN_ARROW);
      LCD_DefineChar (2, LCD_CHAR_TRADEMARK_T);
      LCD_DefineChar (3, LCD_CHAR_TRADEMARK_M);
      LCD_DefineChar (4, LCD_CARITA);

}


/*
** LCD_Clear: Limpia pantalla del LCD y coloca el cursor al principio.
*/
void LCD_Clear (void)
{
      LCD_WriteControl(0x01);
```

```
}


/*
** LCD_Home: Sitúa el cursor del LCD en la fila1 y columna1.
*/
void LCD_Home (void)
{
      LCD_WriteControl(0x02);
}


/*
** LCD_DisplayCharacter: Escribe carácter en posición del cursor.
*/
void LCD_DisplayCharacter (char a_char)
{
      LCD_WriteData (a_char);
}


/*
** LCD_Cursor: Sitúa el cursor en una fila y columna determinadas.
*/
void LCD_Cursor (int row, int column)
{
      switch (row) {
            case 1:     LCD_WriteControl (0x80 + column - 1); break;
            case 2: LCD_WriteControl (0x80 +0x40+ column - 1); break;
            default: break;
      }
}


/*
** LCD_DisplayString: Escribe cadena a partir de posición del cursor
** especificada. La función que la llama debe comprobar la longitud de
** dicha cadena.
*/
void LCD_DisplayString (int row, int column, char *string)
{
```

```
      LCD_Cursor (row, column);
      while (*string)
            LCD_DisplayCharacter (*string++);
}


/*
** LCD_cursor_col: Sitúa el cursor en la siguiente posición cambiando
** de fila si es necesario.
*/
void LCD_cursor_col (int col)
{
      if(col<17)
            {
            LCD_WriteControl (0x80 + col-1);
            }
      else
            {
            LCD_WriteControl (0x80 + 0x40+ col - 17);
            }

}


/*
** LCD_DisplayRow: Escribe una línea en la fila especificada.
*/
void LCD_DisplayRow (int row, char *string)
{
      LCD_Cursor (row, 1);
      while (*string)
            LCD_DisplayCharacter (*string++);
}


/*
** LCD_DisplayScreen: Escribe la pantalla completa.
*/
void LCD_DisplayScreen (char *ptr)
{
      LCD_DisplayRow(1,ptr+ 0);
```

```
      LCD_DisplayRow(2,ptr+16);

}


/*
** LCD_WipeOnLR: Escribe una pantalla desplegándola de
**               izquierda a derecha.
*/
void LCD_WipeOnLR (char *ptr)
{

      int i=0;
      int j=0;
      while (*(ptr+i))
            i++;
      j=i;
      if(j<17)
            {
                  for (i=0; i<j; i++)
                        {
                        LCD_Cursor(1,i+1);
                        LCD_DisplayCharacter(*(ptr+i));
                        }
            }
      else
            {
            for (i=0; i<(j-16); i++)
                  {
                        LCD_Cursor(1,i+1);
                        LCD_DisplayCharacter(*(ptr+i));
                        LCD_Cursor(2,i+1);
                        LCD_DisplayCharacter(*(ptr+16+i));
                  }
            while(i<17)
                  {
                        LCD_Cursor(1,i+1);
                        LCD_DisplayCharacter(*(ptr+i));
                        i++;
                  }
```

```
            }
}


/*
** LCD_WipeOnLR: Escribe una pantalla completa desplegándola de
**               derecha a izquierda.
*/
void LCD_WipeOnRL (char *ptr)
{
      int j;
      char i;
      for (i=16; i>0; i--) {
            LCD_Cursor(1,i);
            LCD_DisplayCharacter(*(ptr+ 0+i-1));
            LCD_Cursor(2,i);
            LCD_DisplayCharacter(*(ptr+16+i-1));
            for(j=0; j<200; j++)
                  retraso(1000);
            }
}


/*
** LCD_WipeOffLR: Borra la pantalla de izqda a derecha.
*/
void LCD_WipeOffLR (void)
{
      int j;
      char i;
      for (i=1; i<17; i++) {
            #define BLOCK 0xff
            LCD_Cursor(1,i);
            LCD_DisplayCharacter(BLOCK);
            LCD_Cursor(2,i);
            LCD_DisplayCharacter(BLOCK);
            for(j=0; j<200; j++)
                  retraso(1000);

      }
}
```

```
/*
** LCD_WipeOffRL: Borra la pantalla de derecha a izqda.
*/
void LCD_WipeOffRL (void)
{
      int j;
      char i;
      for (i=16; i>0; i--) {
            #define BLOCK 0xff
            LCD_Cursor(1,i);
            LCD_DisplayCharacter(BLOCK);
            LCD_Cursor(2,i);
            LCD_DisplayCharacter(BLOCK);
            for(j=0; j<200; j++)
                  retraso(1000);

      }
}


/*
** LCD_CursorLeft: Mueve el cursor un caracter a la izquierda.
*/
void LCD_CursorLeft (void)
{
      LCD_WriteControl (0x10);
}


/*
** LCD_CursorRight: Mueve el cursor un caracter a la derecha.
*/
void LCD_CursorRight (void)
{
      LCD_WriteControl (0x14);
}


/*
```

```
** LCD_CursorOn: Enciende el cursor.
*/
void LCD_CursorOn (void)
{
      LCD_WriteControl (0x0f);
}


/*
** LCD_CursorOff: Apaga el cursor.
*/
void LCD_CursorOff (void)
{
      LCD_WriteControl (0x0c);
}


/*
** LCD_DisplayOff: Apaga el LCD.
*/
void LCD_DisplayOff (void)
{
      LCD_WriteControl(0x08);
}


/*
** LCD_DisplayOn: Enciende el LCD.
*/
void LCD_DisplayOn (void)
{
      LCD_WriteControl(0x0c);
}


/*
** LCD_DefineCharacter: Define el patrón de puntos para caracteres
** definidos por el usuario.
**
**  entradas: address = dirección del carácter (0x00-0x07)
**            pattern = puntero a array de 8-byte con el patrón.
```

```
*/
void LCD_DefineChar (char address, const unsigned char *pattern)
{
      char i;

      LCD_WriteControl(0x40 + (address << 3));
      delay_largo();
      for (i=0; i<8; i++)
      {
            LCD_WriteData(*pattern++);
            delay_largo();
      }
}




/* -----Rutinas de bajo nivel del LCD asociado al bus------- */


/*
** LCD_WriteControl: Escribe una instrucción de control para el LCD.
*/

static void LCD_WriteControl (unsigned int data)
{
      *LCD_CONTROL=data;
      retraso(1000);
}

/*
** LCD_WriteData: Escribe un byte de datos para el LCD.
*/

static void LCD_WriteData (unsigned char data)
{
      *LCD_DATO=data;
      retraso(1000);
}
```

```
/*
**Introducimos retrasos para que le de tiempo a reaccionar al micro
**del LCD.
*/
void retraso(int pasos)
 {
      short i;
      for (i=0;i<pasos;i++){ asm (  "NOP"); }
}


void bucle_retrasos()
{

      int i;
            for(i=1;i<2600;i++)
            {
            retraso(9000);
            }

}


void delay_largo()
{
    long      i;
    for (i = 0; i < 100; i++) {}
}

/*
**Espera_almac: Introduce estados de espera tras la actualización de
**la señal R_S.
*/

void espera_almac()      //Objetivo:esperar 1us a 150Mhz
{
      asm ("      NOP");

}

/*
```

```
**Espera_lectura: Introduce estados de espera para la lectura del
**dato por parte del lcd, tras la activación del enable
*/
void espera_lectura()      //Objetivo:esperar 230ns a 150Mhz
{
      int i=0;
      for (i=0; i<4; i++);
            asm ("     NOP");

}


/*
** Delay_loop: Introduce estados de espera adicionales
*/
void delay_loop()
{
    long      i;
    for (i = 0; i < 1000; i++) {}
}

/*
**Funciones para introducir retrasos adicionales
*/
void espera_39us(void)
{
int ii;
for(ii=0;ii<650;ii++) asm ("  NOP");
}

void espera_43us(void)
{
int ii;
for(ii=0;ii<720;ii++) asm ("  NOP");

}
/*
**error: detiene la ejecución del programa si el dato
**recibido no coincide con el transmitido
*/
void error(void)
```

```
{
    asm("     ESTOP0");
    for (;;);
}
```

Nota: La elección de los valores de las iteraciones en los bucles de espera es fruto de las pruebas realizadas, en las que se comprobó que, para tiempos menores, el funcionamiento no era adecuado.