Cika Electrónica

Nota de Aplicación: CAN-002
Título: **Utilización de displays LCD alfanuméricos con Rabbit 2000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	24/6/03	

Nos interiorizaremos ahora en el desarrollo de una interfaz para conectar un módulo LCD alfanumérico inteligente a un módulo Rabbit utilizando ports de I/O, como se haría con la mayoría de los microcontroladores. Analizaremos más tarde el software de control y un simple programa demostración, que sirve para comprobar el correcto funcionamiento de los módulos LCD que tengamos en stock. A fin de probar la mayor parte posible del hardware, la interfaz será de 8 bits y realizará lectura y escritura del controlador LCD.

Hardware

No es necesario ningún tipo de glue-logic, hacemos una conexión directa entre los ports del Rabbit y el LCD, al igual que con la gran mayoría de los microcontroladores, como puede apreciarse en la tabla a la derecha:

Rabbit	LCD
PA.0	D0
PA.1	D1
PA.2	D2
PA.3	D3
PA.4	D4
PA.5	D5
PA.6	D6
PA.7	D7
PA.8	D8
PE.4	RS
PE.5	R/W
PE.0	E

El port A, como dijimos, hace las veces de bus de datos, mientras que los ports libres del port E generarán, por software, las señales de control.

Software

Desarrollamos a continuación el software de base para manejo del display, elegimos el lenguaje C para mostrar lo fácil que resulta trabajar con Dynamic C. Si el usuario así lo desea, puede utilizar assembler.

El control de I/O se realiza mediante funciones predefinidas en la biblioteca de funciones de soporte que Dynamic C trae. Sin entrar en demasiado detalle, las presentaremos a continuación:

Escritura de I/O ports internos del procesador:

```
WrPortI ( port_address, shadow_register, data )
```

Set y reset de bits particulares en I/O ports internos del procesador:

```
BitWrPortI ( port_address, shadow_register, data, bit)
```

Lectura de ports de I/O internos del procesador:

```
data = RdPortI ( port_address )
```

La incorporación de shadow registers, facilita la integración de las diversas bibliotecas de funciones y su funcionamiento concurrente, dado que Dynamic C incluye, aún en la versión que acompaña a los kits de desarrollo, primitivas de soporte de multitarea cooperativo.
Veamos las funciones de bajo nivel:

```
/* LCD control signals */
#define LCD_E   0
#define LCD_RS  4
#define LCD_RW  5

/* LCD status bits */
#define BUSY    7


/* Low level functions */
```

```
void LCD_Write(int data)
{
       WrPortI ( PADR, &PADRShadow, data );         // escribe el dato
       WrPortI ( SPCR, &SPCRShadow, 0x84 );         // PA0-7 = Outputs, datos en el bus
       BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_RW );  // RW = 0 (Write)
       BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_E );   // Sube E
       BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_E );   // Baja E
}

int LCD_Read()
{
int data;
       BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_RW );  // RW=1 (Read)
       WrPortI ( SPCR, &SPCRShadow, 0x80 );         // PA0-7 = Inputs, bus=3state
       BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_E );   // Sube E
       data=RdPortI ( PADR );                              // lee datos
       BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_E );   // Baja E
       return(data);
}

int LCD_Status()
{
int status;
       BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_RS );  // RS=0 (Cmd)
       do {
              status=LCD_Read();                    // lee status: busy_flag+address
       } while(status&(1<<BUSY));                   // loop mientras busy=1
       return(status);
}

int LCD_ReadData()
{
       BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_RS );  // RS=1 (Data)
       return(LCD_Read());
}

void LCD_WriteCmd(int cmd)
{
       LCD_Status();                                // vuelve con RS=0
       LCD_Write(cmd);
}

void LCD_WriteData(int data)
{
       LCD_Status();
       BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_RS );  // RS=1 (Data)
       LCD_Write(data);
}
```

Algunas funciones de soporte, para generar demoras

```
void MsDelay ( int iDelay )
{      unsigned long ul0;
       ul0 = MS_TIMER;              // get current timer value
       while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}

void UsDelay ( int iDelay )
{      int i;
       iDelay /= 11;         // this is experimental, for a RCM2100
       for ( i=0; i<iDelay; i++ );
}
```

La inicialización y algunas rutinas de soporte de alto nivel. A los fines prácticos, la inicialización del módulo LCD asume que el módulo se ha reseteado al aplicarse la alimentación.

```
void LCD_init ()
```

```
{
        WrPortI ( PEDR,&PEDRShadow,0x00 );           // Outputs=0
        WrPortI ( PEDDR,&PEDDRShadow,'\B10110011' ); // PE0,1,4,5,7=output
        WrPortI ( PEFR, &PEFRShadow, 0 );            // PE: no I/O strobe
        MsDelay ( 1000 );                            // espera reset LCD
        LCD_WriteCmd ( '\B00111000' );               // 8 bit, 2 lineas, caracteres 5x7
        LCD_WriteCmd ( '\B00000110' );               // inc address, desplaza cursor
        LCD_WriteCmd ( '\B00001100' );               // sin cursor, ni blink
}

/* High level functions */

void LCD_show ( char *ptr )
{
        while (*ptr)
                LCD_WriteData (*ptr++);
}

void LCD_at (unsigned int row, unsigned int col)
{
        const static char acPos[4] = { 0x80,0xC0,0x94,0xD4 };

                LCD_WriteCmd( acPos[row]+col );
}


void LCD_clear ( void )
{
        LCD_at(0,0);                                 //address 0
        LCD_WriteCmd (0x10);                         // home
        LCD_WriteCmd (0x01);                         // borra el display
}

LCD_defchar(int charadd, int *pattern)
{
int i;
        LCD_WriteCmd((charadd<<3)^0x40);             // CGRAM address
        for(i=0;i<8;i++)
                LCD_WriteData(pattern[i]);
}
```

El siguiente fragmento de código es un simple ejemplo de la utilización de Dynamic C para escribir un corto y simple programa de control que nos permita corroborar el funcionamiento de un módulo LCD alfanumérico inteligente. Colocamos primero un texto corto y simple., hacemos un desplazamiento (scroll) hacia la izquierda y luego a la derecha, y por último, aprovechando la capacidad de definir caracteres, hacemos una sencilla animación que nos permite observar si todos los"pixels" funcionan correctamente.

```
/* MAIN PROGRAM */

main()
{
int i;

const static int p1[]={0,0,0,0,0,0,0,0};                    // blanco
const static int p[][]={
        {0,0,0,0,0,0,0,0x1f},                        // 1 línea negra
        {0,0,0,0,0,0,0x1f,0x1f},
        {0,0,0,0,0,0x1f,0x1f,0x1f},
        {0,0,0,0,0x1f,0x1f,0x1f,0x1f},
        {0,0,0,0x1f,0x1f,0x1f,0x1f,0x1f},
        {0,0,0x1f,0x1f,0x1f,0x1f,0x1f,0x1f},
        {0,0x1f,0x1f,0x1f,0x1f,0x1f,0x1f,0x1f},
```

```
        {0x1f,0x1f,0x1f,0x1f,0x1f,0x1f,0x1f,0x1f}    // negro
};

        LCD_init();
        LCD_clear();
        LCD_show("test LCD");                       // línea 1
        LCD_at(1,0);
        LCD_show("- Cika -");                       // línea 2
        MsDelay(1000);
        for(i=0;i<8;i++){
                LCD_WriteCmd('\B00011000');         // mueve a izquierda
                MsDelay(300);
        }
        for(i=0;i<8;i++){
                LCD_WriteCmd('\B00011100');         // mueve a derecha
                MsDelay(300);
        }
        MsDelay(1000);
        LCD_defchar(0,p1);                          // borra char 0
        LCD_at(0,0);
        for(i=0;i<80;i++)
                LCD_WriteData(0);                   // llena DDRAM con char 0
        while(1){
                LCD_defchar(0,p1);
                MsDelay(200);
                for(i=0;i<8;i++){
                        LCD_defchar(0,p[i]);        // animación arriba
                        MsDelay(200);
                }
                MsDelay(2000);
                for(i--;i>=0;i--){
                        LCD_defchar(0,p[i]);        // animación abajo
                        MsDelay(200);
                }
                LCD_defchar(0,p1);                  // borra char 0
                MsDelay(200);
        }
}
```