Cika Electrónica

Nota de Aplicación: CAN-003
Título: **Utilización de displays LCD gráficos (HD61202) 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 gráfico inteligente a un módulo Rabbit utilizando ports de I/O, como se haría con la mayoría de los microcontroladores. Utilizaremos un módulo Powertip PG12864, de 128x64 pixels, basado en chips controladores compatibles con el HD61202, de Hitachi, y su clon: el KS0108, de Samsung. 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, y de paso, demostrar sus capacidades gráficas. 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

Dado que el controlador utilizado en estos módulos sólo es capaz de direccionar 64x64 pixels, este tipo de displays utiliza dos controladores, uno para la parte izquierda y otro para la parte derecha del display. El hardware de conexión resulta casi igual al utilizado para los displays alfanuméricos, es decir, la clásica interfaz tipo 6800, pero con la introducción de tres nuevas señales: dos señales de selección del controlador (CS1 y CS2) y una de reset (RST). La señal RS de los displays alfanuméricos tiene el nombre I/D en los displays gráficos (Instruction/Data), pero su funcionamiento es el mismo.

Otra diferencia es el circuito de contraste; si bien los módulos alfanuméricos funcionan muy bien con una tensión fija cercana a los 0,6V, los módulos gráficos necesitan de una tensión de aproximadamente -8V. La misma puede obtenerse de la salida que estos módulos proveen (Vout ó Vee).

Para la interfaz con el micro 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
PE.4	----------	I/D
PE.5	----------	R/W
PE.0	----------	E
PE.1	----------	CS1
PE.7	----------	CS2

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

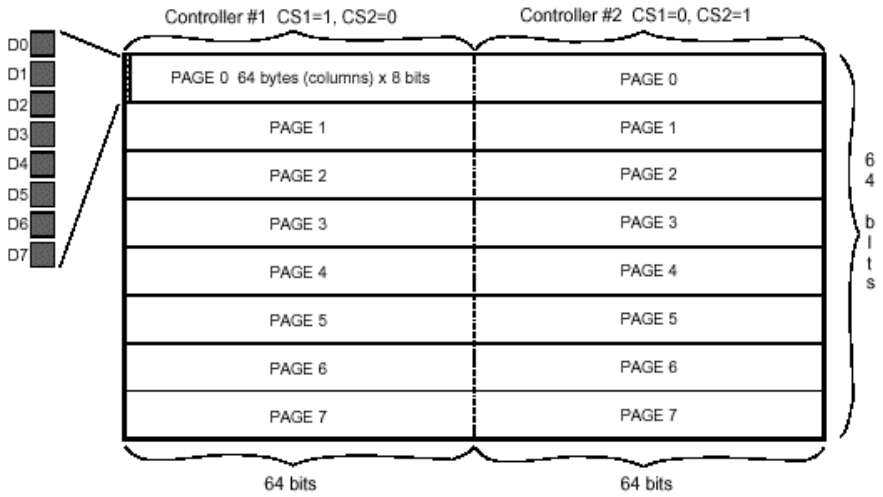
Software

Estructura y breve descripción del display gráfico

La estructura de memoria de estos módulos gráficos es algo caprichosa, la misma se halla agrupada en bytes en sentido vertical, divididos a su vez en páginas. Debido al hecho de que, a su vez, se necesitan dos controladores, la pantalla resulta dividida a la mitad, y cada mitad es atendida por un controlador.

El bit menos significativo del primer byte de memoria del primer controlador corresponde al punto situado en la pantalla arriba a la izquierda, y el bit más significativo del último byte de memoria del segundo controlador corresponde al punto situado en pantalla abajo a la derecha.

En el gráfico que figura a continuación puede apreciarse un esquema de esta estructura:



El direccionamiento del byte a leer o escribir en memoria se hace mediante comandos, como en los displays alfanuméricos, pero aquí tenemos dos direccionamientos: vertical y horizontal, y por lo tanto dos comandos :

1. setear dirección horizontal (de 0 a 63)
2. setear dirección vertical o número de página (de 0 a 7)

El direccionamiento horizontal tiene un contador autoincrementado, el cual apunta a la dirección siguiente luego de una lectura o escritura. Esto resulta óptimo para enviar los datos byte por byte hasta completar una página.

Algoritmos

Para direccionar un punto horizontalmente, basta con comparar su coordenada horizontal *x* con 64, y seleccionar uno u otro controlador. Una vez seleccionado el controlador correspondiente, la dirección horizontal dentro de ese controlador corresponde a la coordenada horizontal *x*, si *x*<64, o a *x*-64, si *x*>=64.

Para ubicar un punto verticalmente, deberemos dividir la coordenada vertical *y* por 8, siendo la parte entera del resultado el número de página y el resto el número de bit dentro del byte seleccionado.

Para graficar funciones, debemos tener en cuenta que la coordenada (0;0) se halla en el extremo superior izquierdo de la pantalla.

Para mostrar pantallas, deberemos agrupar los datos de modo tal de poder enviarlos de forma que aproveche de manera eficiente los contadores autoincrementados y la estructura de memoria:

- 64 bytes correspondientes a la primera página del controlador izquierdo (bit menos significativo corresponde a la parte superior)
- 64 bytes correspondientes a la primera página del controlador derecho (bit menos significativo corresponde a la parte superior)
- repetir para las 7 páginas siguientes, total de 1024 bytes.

Si bien sería más eficiente mandar primero la información a un controlador y luego al otro, resulta menos complicado convertir las pantallas de un formato común a este formato. El lector puede optar por la forma de implementación que más le agrade.

Para imprimir textos, podemos recurrir a un simple truco que nos permita simplificar el software: definimos líneas de texto o “renglones”, y nos limitamos a escribir dentro de ellas. Cada línea de texto corresponderá a una página de memoria del controlador, y si utilizamos una tipografía de 7 puntos de alto, la hacemos caber perfectamente dentro de los 8 bits de la página, manteniendo un punto de separación entre líneas. En sentido horizontal, haremos que un caracter tenga todos sus puntos de un mismo controlador. Si utilizamos una tipografía de 5x7, con un punto de separación entre caracteres, tendremos 6 puntos por caracter en sentido horizontal; dentro del espacio de un controlador (64 puntos), podremos dibujar 10 caracteres (60 puntos), y 4 pixels sin utilizar a ambos lados. Nuestro display gráfico puede entonces superponer textos en una matriz imaginaria de 20 caracteres por 8 líneas.

Para simplificar aún más, seteamos las direcciones al momento de escribir cada caracter; este esquema tal vez no sea el más eficiente, pero resulta extremadamente simple y es ideal para demostrar las capacidades de estos displays.

Desarrollo

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.

Las funciones de bajo nivel resultan muy similares a las desarrolladas para el display alfanumérico, dada la gran similitud entre controladores:

```
/* LCD control signals */
#define LCD_CS1 1
#define LCD_CS2 7
#define LCD_E   0
#define LCD_ID  4
#define LCD_RW  5

/* LCD status bits */
#define BUSY   7

/* Low level functions */

void LCD_SelSide(int side)
{
        if(side)       {
                BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_CS2 );         // Sube CS2
                BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_CS1 );         // Baja CS1
        }
        else    {
                BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_CS1 );         // Sube CS1
                BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_CS2 );         // Baja CS2
        }
}

void LCD_CalcPage(int y,int *page,int *row)
{
        *page=(y>>3);                 // page = y/8, parte entera
        *row=y-((*page)<<3);          // row = resto de la división anterior
}

void LCD_Write(int data)
{
        WrPortI ( PADR, &PADRShadow, data );         // escribe datos
        WrPortI ( SPCR, &SPCRShadow, 0x84 );         // PA0-7 = Outputs, datos en el bus
        BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_RW );  // Baja RW (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 );  // Sube RW (Read)
        WrPortI ( SPCR, &SPCRShadow, 0x80 );         // PA0-7 = Inputs, saca datos del bus
        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 );  // Baja I/D (Cmd)
```

```
        do {
                status=LCD_Read();                      // lee status (busy flag)
        } while(status&(1<<BUSY));                      // loop mientras busy=1
        return(status);
}

int LCD_ReadData()
{
        BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_RS );  // Sube I/D (Data)
        return(LCD_Read());
}

void LCD_WriteCmd(int cmd)
{
        LCD_Status();                                   // I/D=0 al volver de esta función
        LCD_Write(cmd);
}

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

Algunas funciones de soporte, para generar demoras

```
void MsDelay ( int iDelay )
{
        unsigned long ul0;
        ul0 = MS_TIMER;               // compara con valor actual del timer
        while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}

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

Funciones de inicialización y de soporte de alto nivel.

La inicialización del módulo LCD se realiza por hardware, mediante el pin RST. El estado de reset puede comprobarse mediante los bits de status; no obstante, por simpleza, asumimos que el controlador se ha reseteado exitosamente.

```
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 de LCD
        LCD_SelSide(1);                              // Controlador 1
        LCD_WriteCmd ( '\B00111111' );               // Display on
        LCD_SelSide(0);                              // Controlador 2
        LCD_WriteCmd ( '\B00111111' );               // Display on
}

/* High level functions */

void LCD_home ( void )
{
        LCD_SelSide(1);                              // lado derecho
```

```
        LCD_WriteCmd ('\B01000000');                // address = 0
        LCD_WriteCmd ('\B11000000');                // display empieza en address 0
        LCD_SelSide(0);                             // lado izquierdo
        LCD_WriteCmd ('\B01000000');                // address = 0
        LCD_WriteCmd ('\B11000000');                // display empieza en address 0
}

void LCD_fill(unsigned char pattern)
{
int i,page;
        LCD_home();                                 // address 0,0
        for(page=0;page<8;page++) {
                LCD_SelSide(0);                     // Controlador izquierdo
                LCD_WriteCmd (0xB8+page);           // Número de página vertical
                for(i=0;i<64;i++)
                        LCD_WriteData(pattern);     // Llena página con datos
                LCD_SelSide(1);                     // Controlador derecho
                LCD_WriteCmd (0xB8+page);           // Número de página vertical
                for(i=0;i<64;i++)
                        LCD_WriteData(pattern);     // Llena página con datos
        }
}

#define LCD_clear() LCD_fill(0)

void LCD_plot (int x, int y)
{
int page,row,data;
        LCD_SelSide(x>63);                          // 0-63 => side=0; 63-127 => side=1
        x=(x>63)?(x-64):x;                          // corrige x si >63
        LCD_WriteCmd (0x40+x);                      // address = x o x-64
        LCD_CalcPage(y,&page,&row);                 // calcula página y fila en página
        LCD_WriteCmd (0xB8+page);                   // setea número de página
        LCD_ReadData();                             // luego de setear dirección hay que
                                                    //  hacer una lectura ficticia
        data=LCD_ReadData();                        // lee datos en esa dirección
        data|=(1<<row);                             // setea el punto solicitado (bit)
        LCD_WriteCmd (0x40+x);                      // vuelve a setear la dirección
                                                    // (leer incrementa el puntero)
        LCD_WriteData(data);
}

void LCD_dump (unsigned char *imgdata)
{
int i,page;

        LCD_home();                                         // address 0,0
        for(page=0;page<8;page++) {
                LCD_SelSide(0);                             // izquierdo
                LCD_WriteCmd (0xB8+page);                   // página 0
                for(i=0;i<64;i++)
                        LCD_WriteData(*(imgdata++));        // lado izquierdo por página
                LCD_SelSide(1);                             // derecho
                LCD_WriteCmd (0xB8+page);                   // página 0
                for(i=0;i<64;i++)
                        LCD_WriteData(*(imgdata++));        // lado derecho por página
        }
}

void LCD_putchr ( char chr )
{
const static char font5x7[] = {
<eliminada por cuestiones de espacio, 5 bytes por cada caracter, en formato de pantalla>
};
int i;
        for(i=0;i<5;i++)
```

```
                LCD_WriteData(font5x7[5*(chr-0x20)+i]);      // dibuja caracter
        LCD_WriteData(0);                                    // separador (línea en blanco)
}

void LCD_printat (unsigned int row, unsigned int col, char *ptr)
{
int x;
        do {
                x=(col>9)?6*(col-10):6*col;          // corrige x si col>9
                if(col>9)                            // 0-9 => side=0; 10>20 => side=1
                        LCD_SelSide(1);              // lado derecho, comienza normal
                else {
                        LCD_SelSide(0);              // lado izquierdo
                        x+=4;                        // 4 pixels corrido a la derecha
                }
                LCD_WriteCmd (0xB8+row);             // fila = página
                LCD_WriteCmd (0x40+x);               // address = 6*col ó 6*col-10
                LCD_putchr (*ptr++);                 // escribe caracter
                col++;                               // siguiente columna
        }       while (*ptr);                        // toda la cadena
}
```

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 gráfico inteligente. Para observar si todos los pixels funcionan correctamente, pintamos la pantalla de negro y luego la blanqueamos. Luego graficamos una función seno y finalmente mostramos una pantalla.

```
/* MAIN PROGRAM */

main()
{
int i,page,pepe;

const static unsigned char image[]= {
<eliminada por cuestiones de espacio, 1024 bytes (128x8) en formato de pantalla>
};
        LCD_init();
        while(1){
                LCD_clear();                                 // pantalla blanca
                MsDelay ( 3000 );                            // espera 3 segs
                LCD_fill(0xFF);                              // pantalla negra
                MsDelay ( 3000 );
                LCD_clear();

                for(i=0;i<128;i++)
                        LCD_plot(i,32);                      // eje horizontal
                for(i=0;i<64;i++)
                        LCD_plot(64,i);                      // eje vertical
                for(i=-64;i<63;i++)
                        LCD_plot(64+i,32-(int)(32*(sin(i*3.14159/64))));  // función sen(x)
                LCD_printat(0,0,"sin(x)");                   // texto
                MsDelay ( 3000 );
                LCD_dump(image);                             // imagen
                MsDelay ( 3000 );
        }
}
```