Cika Electrónica

Nota de Aplicación: CAN-007
Título: **Utilización de displays LCD gráficos (LC7981) con Rabbit 2000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	4/9/03	

Nos interiorizaremos ahora en el desarrollo de una interfaz para conectar un módulo LCD gráfico inteligente Powertip PG24064FRM-A, a un módulo Rabbit 2000. Se trata de un display de 240x64 pixels basado en chips controladores compatibles con el LC7981, de Sanyo. 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. 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

El LC7981 presenta una interfaz tipo Motorola (E, RS, R/W), como el HD44780 de los displays alfanuméricos, por lo que el hardware no nos presenta ninguna novedad.

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	----------	RS
PE.3	----------	RW
PE.0	----------	E
PE.1	----------	CS

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. La señal CS podría conectarse directamente a masa, a criterio del usuario. Dado que la señal E es activa en alto, no se nos presentan demasiados inconvenientes al momento del arranque.

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

El display dispone, además, de un pin de reset, el cual podemos controlar a voluntad o conectar al reset del circuito. Para el desarrollo de esta nota de aplicación, simplemente lo conectamos mediante un pull-up a la tensión de alimentación.

Software

Breve descripción del display gráfico

Estos displays tienen dos modos de funcionamiento: texto y gráfico, por lo que para superponer texto y gráficos deberemos hacer nuestra propia rutina de impresión de textos, como en la CAN-003. Para el caso del modo texto, el LC7981 dispone de un generador de caracteres y una ROM de caracteres de 5x7.

La estructura de memoria de pantalla es lineal, tanto en modo gráfico como en modo texto. En este último modo, el primer byte corresponde al caracter ubicado arriba a la izquierda, y el último byte corresponde al ubicado abajo a la derecha. En el modo gráfico, los pixels se agrupan horizontalmente en bytes, correspondiendo el primer byte de memoria a los primeros ocho pixels de la primera línea de arriba a la izquierda, y el último byte a los últimos ocho pixels de la última línea de abajo a la derecha. El bit menos significativo del primer byte de memoria corresponde al punto situado en la pantalla arriba a la izquierda, y el bit más significativo del último byte de memoria corresponde al punto situado en pantalla abajo a la derecha.

El direccionamiento del byte a leer o escribir en memoria se hace mediante comandos, especificando la dirección de memoria. Tiene además 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 pantalla.

Una característica interesante del display, es que posee un par de instrucciones para setear o resetear directamente un bit en memoria, lo que simplifica las rutinas de dibujo.

Algoritmos

Para direccionar un punto debemos traducir sus coordenadas a una dirección lineal, para ello, deberemos multiplicar la coordenada vertical *y* por la cantidad de bytes en sentido horizontal de la pantalla (30 bytes) y sumarle la coordenada horizontal *x* dividida por 8 (pixels por byte). El resto de dividir *x*/8 es el número de pixel dentro del byte: *address=30*y+x/8 ; bit=resto(x/8).*

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 el contador autoincrementado y la estructura de memoria; dada la estructura lineal, esto se reduce simplemente a enviar todos los bytes corridos. Si comparamos la estructura de memoria del display con la forma de guardar imágenes blanco y negro en formato XBM[1], veríamos que hay una correspondencia perfecta, pudiendo extraer directamente la información de dicho archivo, que no es más que una definición del tipo usado en C de un array de bytes. La única operación a realizar es que, si disponemos de un display del tipo STN-, como es el caso del desarrollo de esta nota de aplicación, deberemos invertir los colores previamente. También es posible procesar un archivo de tipo BMP[2], como se ha desarrollado en la CAN-005, no obstante, requiere un poco más de trabajo.

Para imprimir textos, calculamos simplemente la posición de memoria a partir de fila y columna de modo similar: *address=40*fila+columna*, para 40 caracteres por fila (matriz de caracteres de 6x8).

Desarrollo

Desarrollamos a continuación el software de base para manejo del display. Definiremos dos modos de pantalla: uno gráfico de 240x64 y uno de texto de 40x8, con caracteres de 6x8, usando el generador interno, por lo que se verán caracteres de 5x7 en una trama de 6x8.

Dada la cantidad de información a mover hacia el display para la presentación de pantallas, sumado al hecho del constante chequeo del flag de busy, decidimos desarrollar toda la rutina de escritura en assembler. Dado que solamente nuestra rutina accede al port A, prescindimos de su correspondiente shadow register; no obstante, dado que el port E puede compartirse con otra tarea, creemos “buena práctica” el actualizar el shadow register. Esta actualización debe hacerse de forma atómica, para evitar que una interrupción altere el valor en la mitad de la operación.

```
/* LCD control signals */
#define LCD_CS  1
#define LCD_E          0
#define LCD_RS  4
#define LCD_RW  3

/* Low level functions */

#asm
;la función recibe dos parámetros:
;@sp+2= comando a escribir
;@sp+4= dato ó parámetro a escribir
;
LCD_WriteCmd::
        ld a,0x80                       ; PA0-7 = Inputs, puede leer
        ioi ld (SPCR),a                 ; ahora
        ld hl,PEDRShadow                ; apunta a control port (shadow)
        ld de,PEDR                      ; apunta a control port
        set LCD_RW,(HL)                 ; Sube RW
        set LCD_RS,(HL)                 ; Sube RS
        ioi ldd                         ; ahora
l1:
```

1 El formato XBM es utilizado en ambiente X11, un ejemplo de software gratis que lo utiliza es *Gimp* (GNU Image Manipulation Program).

2 No se ha incluído ese desarrollo en esta nota dado que a estas resoluciones existen bytes extra en el formato que dificultan la conversión.

```
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        set LCD_E,(HL)              ; Sube E
        ioi ldd                     ; ahora
        ioi ld a,(PADR)             ; lee status (A=PA0-7)
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        res LCD_E,(HL)              ; Baja E
        ioi ldd                     ; ahora
        rla                         ; Chequea busy (bit 7 al carry)
        jr c,l1                     ; loop mientras busy=1

        ld a,0x84                   ; PA0-7 = Outputs, puede escribir
        ioi ld (SPCR),a             ; ahora
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        res LCD_RW,(HL)             ; Baja RW
        ioi ldd                     ; ahora
        ld hl,(sp+2)                ; lee comando (parámetro 1) (LSB)
        call LCD_Write              ; escribe comando en display
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        res LCD_RS,(HL)             ; Baja RS
        ioi ldd                     ; ahora
        ld hl,(sp+4)                ; lee dato (parámetro 2) (LSB)
LCD_Write:
        ld a,l
        ioi ld (PADR),a             ; pone datos a escribir en display en port A
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        set LCD_E,(HL)              ; Sube E
        ioi ldd                     ; ahora
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        res LCD_E,(HL)              ; Baja E
        ioi ldd                     ; ahora
        ret

#endasm
```

El prefijo *ioi* es el que nos permite utilizar cualquier instrucción de acceso a memoria como instrucción de I/O. Esta es una de las mayores diferencias entre Rabbit y Z-80, en cuanto a set de instrucciones; descartando, claro está las funciones agregadas. Obsérvese como la operación atómica se realiza mediante la instrucción *ldd* (LoaD and Decrement), que copia hacia la posición apuntada por DE el contenido de la posición apuntada por HL, decrementando ambos punteros y el contador BC. El inconveniente es que más adelante debemos volver a cargar el mismo valor en los punteros (preferimos esto en vez de salvarlos en el stack porque es más rápido), o incrementarlos nuevamente (más rápido aún). Nos pareció menos confuso para el desarrollo de la nota volver a cargar el mismo valor, sacrificando eficiencia por claridad.

Dado que el display dispone de una función para setear un pixel (bit en memoria), no necesitamos funciones de lectura de memoria, al menos para el alcance de esta nota de aplicación.

El resto de las funciones se ha escrito en C, dado que con el incremento de velocidad logrado es suficiente para el módulo utilizado y las prestaciones esperadas de una nota de aplicación.

```
void LCD_init ()
{
        WrPortI ( PEDR,&PEDRShadow,0x00 );                  // Outputs=0
        WrPortI ( PEDDR,&PEDDRShadow,'\B10011011' );        // PE0,1,3,4,7 = output
        WrPortI ( PEFR, &PEFRShadow, 0 );                   // PE: no I/O strobe
        MsDelay ( 1000 );                                   // espera reset LCD

        BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_CS );         // Baja CS
}
```

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

Rutinas de soporte de alto nivel, sumamente simples

```
/* High level functions */

void LCD_set5x7()
{
	LCD_WriteCmd ( 1,'\B01110101' );          // caracteres de 5x7 (matriz de 6x8)
	LCD_WriteCmd ( 2,39 );                    // 40 caracteres (240/6)
	LCD_WriteCmd ( 3,63);                // 1/64 duty cycle (hoja de datos del display)
	LCD_WriteCmd ( 4,7);                      // cursor en línea 8
	LCD_WriteCmd ( 0,'\B00110000' );          // Display on, master, texto, sin cursor
}

void LCD_setgfx()
{
	LCD_WriteCmd ( 1,'\B01110111' );          // matriz de 8x8
	LCD_WriteCmd ( 2,29 );                    // 30 strips (bytes) (240/8)
	LCD_WriteCmd ( 3,63);                     // 1/64 duty cycle
	LCD_WriteCmd ( 0,'\B00110010' );          // Display on, master, gráficos
}

void LCD_address ( int address )
{
	LCD_WriteCmd(10,address&0xFF);            // LO byte
	LCD_WriteCmd(11,(address>>8)&0xFF);       // HI byte
}

void LCD_home ( void )
{
	LCD_WriteCmd (8,0);                       // display comienza en address 0
	LCD_WriteCmd (9,0);
	LCD_address (0);                          // address 0
}

void LCD_fill(int count,unsigned char pattern)
{
int i;
	LCD_home();                               // address 0
	for(i=0;i<count;i++)
		LCD_WriteCmd(0x0C,pattern);       // llena todo
}


void LCD_printat (unsigned int cpl,unsigned int row, unsigned int col, char *ptr)
{
	LCD_address (cpl*row+col);                // cursor address
	while (*ptr)
		LCD_WriteCmd (0x0C,*ptr++);
}


/* Plot image data, 8 pixels per byte, MSB right */

void LCD_plot (int x, int y)
{
int strip,bit;
	strip=x>>3;                               // strip = x/8
	bit=x&'\B0111';                           // bit = resto
```

```
        y=strip+30*y;                                   // address = 30*y+strip
        LCD_address(y);                                 // direcciona strip en pantalla
        LCD_WriteCmd(0x0F,bit);                         // setea bit
}

void LCD_dump(unsigned char *imgdata)
{
int x;
                LCD_home();
                for(x=0;x<1920;x++)
                        LCD_WriteCmd(0x0C,imgdata[x]);        // manda datos al display
}
```

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;

const static unsigned char rabbit[] = {
<eliminada por cuestiones de espacio, 1920 bytes (30x64) en formato de pantalla>
};


        LCD_init();
        while(1){

                LCD_setgfx();
                LCD_fill(30*64,0);                      // borra (llena con 0's)
                MsDelay ( 3000 );                       // espera 3 seg
                LCD_fill(30*64,0xFF);                   // pinta (llena con FF's)
                MsDelay ( 3000 );                       // espera 3 seg

                LCD_set5x7();
                LCD_fill(40*8,' ');                     // borra texto (llena con espacios)
                LCD_printat(40,4,8,"Cika Electronica S.R.L.");
                MsDelay ( 3000 );                       // espera 3 seg

                LCD_setgfx();                           // modo gráfico
                LCD_fill(30*64,0);                      // borra
                for(i=0;i<240;i++)
                        LCD_plot(i,32);                 // eje x
                for(i=0;i<64;i++)
                        LCD_plot(120,i);                // eje y
                for(i=-120;i<120;i++)
                        LCD_plot(120+i,32-(int)(32*(sin(i*3.14159/120)))); // grafica función
                MsDelay ( 3000 );                       // espera 3 seg
                LCD_dump(rabbit);                       // muestra logo Rabbit
                MsDelay ( 6000 );                       // espera 3 seg
        }
}
```