Cika Electrónica

Nota de Aplicación: CAN-011
Título: **Utilización de displays LCD gráficos (T6963) con Rabbit 2000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	5/9/03	

Nos interiorizaremos ahora en el desarrollo de una interfaz para conectar un módulo LCD gráfico inteligente Powertip PG24064FRM-E, a un módulo Rabbit 2000. Se trata de un display de 240x64 pixels basado en chips controladores compatibles con el T6963, de Toshiba. 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 T6963 presenta una interfaz tipo Intel, es decir, con líneas de RD y WR separadas. Posee además la línea de selección, CE, y otra para determinar si lo que se escribe es un dato o un comando: C/D. Existe además otro pin que determina el formato de caracteres: 6x8 ú 8x8, llamado FS. A los fines de esta nota de aplicación, conectaremos este último pin a masa, forzando el modo 8x8.

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	----------	WR
PE.3	----------	RD
PE.0	----------	CE
PE.1	----------	C/D

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 CE podría conectarse directamente a masa, a criterio del usuario. El único inconveniente es una posible escritura no intencional al momento del arranque, problema que también podemos tener con este esquema, dado que los ports utilizados son entradas al momento de reset. Podrían incluírse sendos pull-ups si esta posibilidad resultara un inconveniente.

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 son versátiles, la memoria puede ser dividida en dos áreas: gráfica y de texto, las cuales pueden, a su vez, habilitarse y/o superponerse independientemente. Para el caso del modo texto, el T6963 dispone de un generador de caracteres y una ROM de caracteres de 5x7, aunque es posible utilizar una ROM externa o la misma RAM del display. Al momento de definir cada pantalla, definimos también en qué posición de memoria comienza y cómo se asigna la memoria. Una carcterística interesante es que el área gráfica puede funcionar como memoria de atributos, modificando al área de texto (parpadeo e invertido)

La estructura de memoria de pantalla es lineal, tanto para gráficos como para textos. En esta última modalidad, el primer byte corresponde al caracter ubicado arriba a la izquierda, y el último byte corresponde al ubicado abajo a la derecha. Para gráficos, los pixels se agrupan horizontalmente en bytes, correspondiendo el

primer byte de memoria a los primeros ocho pixels[1] 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 más significativo del primer byte de memoria corresponde al punto situado en la pantalla arriba a la izquierda, y el bit menos 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 que puede ser autoincrementado, autodecrementado, o estático; pudiendo apuntar a la dirección siguiente, previa, o no cambiar, luego de una lectura o escritura. Existe además un modo denominado "Auto", en el cual se envían todos los bytes de datos en bloque, sin su correspondiente comando de escritura asociado. 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. Otra característica, no tan interesante, es que según definamos el modo (pin FS) en 6x8 ú 8x8, los bytes en modo gráfico serán de 6 ú 8 pixels. Esto significa, que en el modo 6x8, el controlador ignora los bits 6 y 7 de los datos que se le escriben. Por este motivo, elegimos el modo 8x8 para el desarrollo de la nota de aplicación.

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. Dado que el MSB se halla a la izquierda, el pixel 0 corresponde al bit 7 y el pixel 7 al bit 0, es decir: *address=30*y+x/8 ; bit=7-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 "Sun raster"[2], veríamos que hay una correspondencia perfecta, pudiendo extraer directamente la información de dicho archivo. La única operación a realizar sobre la imagen 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. El formato utilizado corresponde al modo standard, sin compresión (RLE), y posee un header de 32 bytes que deberá removerse. También es posible procesar un archivo de tipo BMP[3], como se ha desarrollado en la CAN-005, dado que el formato de memoria es el mismo para ambos controladores.

Para imprimir textos, calculamos simplemente la posición de memoria a partir de fila y columna de modo similar: *address=30*fila+columna*, para 30 caracteres por fila (matriz de caracteres de 8x8). En el modo atributos, la pantalla gráfica funciona como memoria de atributos, de modo que tenemos dos pantallas de texto, escribiendo en una elegimos el caracter, y escribiendo en la otra seteamos los atributos. Cabe destacar que el set de caracteres no es ASCII, aunque los caracteres están en el mismo orden pero desplazados. Esto se resuelve restando 32 al código ASCII, dado que el set de caracteres comienza con el espacio.

Desarrollo

Desarrollamos a continuación el software de base para manejo del display. Definiremos dos pantallas: una gráfica de 240x64 y una de texto de 30x8, con caracteres de 8x8, usando el generador interno, por lo que se verán caracteres de 5x7 en una trama de 8x8. Ambas pantallas estarán superpuestas (modo XOR) y podremos superponer texto y gráficos.

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.

1 En modo 6x8 los bytes son de 6 pixels, pero para esta nota utilizamos el modo 8x8.

2 Dicho formato es utilizado en ambiente X11, particularmente en máquinas de la firma Sun Microsystems. Un ejemplo de software gratis que lo utiliza es *Gimp* (GNU Image Manipulation Program).

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

Para aprovechar el modo “Auto”, y dado que éste utiliza otro flag de busy diferente, preparamos dos sets de rutinas: uno para modo normal y otro para modo “Auto”, con una 'A' en el nombre.

```
/* LCD control signals */
#define LCD_CD  1
#define LCD_CE  0
#define LCD_WR  4
#define LCD_RD  3

#asm
;Las funciones requiren un parametro:
;@sp+2= dato a escribir
;
LCD_WriteCmd::
        call LCD_Busy                ; Espera a que el controlador esté libre
        jr Write                     ; C/D vuelve seteado (comando)
LCD_WriteACmd::                      ; Modo Auto
        call LCD_ABusy               ; Espera a que el controlador esté libre
        jr Write                     ; C/D vuelve seteado (comando)

LCD_WriteData::
        call LCD_Busy                ; Espera a que el controlador esté libre
        jr WriteData
LCD_WriteAData::                     ; Modo Auto
        call LCD_ABusy               ; Espera a que el controlador esté libre
WriteData:
        ld hl,PEDRShadow             ; control port (shadow)
        ld de,PEDR                   ; control port
        res LCD_CD,(HL)              ; Baja C/D (data)
        ioi ldd                      ; ahora

Write:
        ld a,0x84                    ; PA0-7 = Outputs
        ioi ld (SPCR),a              ; escribe
        ld hl,(sp+2)                 ; recupera valor a escribir (parámetro) (LSB)
        ld a,l
        ioi ld (PADR),a              ; lo escribe al controlador
        ld hl,PEDRShadow             ; control port (shadow)
        ld de,PEDR                   ; control port
        res LCD_CE,(HL)              ; baja CE
        res LCD_WR,(HL)              ; baja WR
        ioi ldd                      ; ahora
        ld hl,PEDRShadow             ; control port (shadow)
        ld de,PEDR                   ; control port
        set LCD_WR,(HL)              ; sube WR
        set LCD_CE,(HL)              ; sube CE
        ioi ldd                      ; ahora
        ret

LCD_Busy:
        ld e',3                      ; STA0, STA1
        jr Busy
LCD_ABusy:                           ; Modo Auto
        ld e',8                      ; STA3
Busy:
        ld a,0x80                    ; PA0-7 = Inputs
        ioi ld (SPCR),a              ; escribe
        ld hl,PEDRShadow             ; control port (shadow)
        ld de,PEDR                   ; control port
        set LCD_CD,(HL)              ; sube C/D
        ioi ldd                      ; ahora
l1:
        ld hl,PEDRShadow             ; control port (shadow)
        ld de,PEDR                   ; control port
        res LCD_CE,(HL)              ; baja CE
        res LCD_RD,(HL)              ; baja RD
```

```
        ioi ldd                     ; ahora
        ioi ld a,(PADR)             ; lee al controlador
        ld hl,PEDRShadow            ; control port (shadow)
        ld de,PEDR                  ; control port
        set LCD_RD,(HL)             ; sube RD
        set LCD_CE,(HL)             ; sube CE
        ioi ldd                     ; ahora
        ex de',hl                   ; HL = DE' (L tiene los bits a chequear)
        and l                       ; resetea los demás bits
        cp l                        ; chequea flag(s) de busy
        jr nz,l1                    ; loop mientras no esté listo
        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.

También hicimos uso de dos nuevas instrucciones, que no existían en el Z-80: *LD E',n*, que nos permitió cargar directamente un valor en un registro del set alternativo, y *EX DE',HL*, que nos permitió recuperar ese valor, ya que sólo algunas operaciones están permitidas sobre registros del set alternativo directamente.

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,'\B10011011' );   // Señales de control inactivas
        WrPortI ( PEDDR,&PEDDRShadow,'\B10011011' ); // PE0,1,3,4,7 = output
        WrPortI ( PEFR, &PEFRShadow, 0 );            // PE: no I/O strobe
        MsDelay ( 1000 );                            // espera reset de LCD
        LCD_WriteData(0x0);
        LCD_WriteData(0x0);                          // 0x000
        LCD_WriteCmd(0x40);                          // Inicio de área de texto (TXT)
        LCD_WriteData(0x1e);                         // 30x8=240
        LCD_WriteData(0x0);
        LCD_WriteCmd(0x41);                          // Número de columnas
        LCD_WriteData(0x0);
        LCD_WriteData(0x10);                         // 0x1000
        LCD_WriteCmd(0x42);                          // Inicio de área gráfica/atributos (GFX)
        LCD_WriteData(0x1e);                         // 30x8=240
        LCD_WriteData(0x0);
        LCD_WriteCmd(0x43);                          // Número de columnas
        LCD_WriteCmd(0x81);                          // TXT XOR GFX, CG interno
        LCD_WriteCmd(0x9C);                          // TXT on, GFX on, no cursor, no blink
}

void LCD_address ( int address )
{
        LCD_WriteData(address&0xFF);                 // LO byte
        LCD_WriteData((address>>8)&0xFF);            // HI byte
        LCD_WriteCmd(0x24);                          // set address
}

void LCD_fill(int count,unsigned char pattern)
{
```

```
int i;
	LCD_WriteCmd(0xb0);				// Auto data write
	for(i=0;i<count;i++)
		LCD_WriteAData(pattern);		// llena la pantalla con 'pattern'
	LCD_WriteACmd(0xb2);				// reset Auto data write
}

void LCD_fillblack()
{
		LCD_address(0x1000);			// GFX screen
		LCD_fill(30*64,0);			// llena con 0's (negro en STN-)
}

void LCD_fillwhite()
{
		LCD_address(0x1000);			// GFX screen
		LCD_fill(30*64,0xFF);			// llena con FF's (blanco en STN-)
}

int LCD_print (char *ptr)
{
char c;
int i;

	i=0;
	LCD_WriteCmd(0xb0);				// Auto data write
	while (c=*ptr++){
		LCD_WriteAData (c-0x20);		// convierte ASCII a código controlador
		i++;
	}
	LCD_WriteACmd(0xb2);				// reset Auto data write
   return(i);					// devuelve número de caracteres
}

void LCD_printat (int cpl,unsigned int row, unsigned int col, char *ptr)
{
	LCD_address (cpl*row+col);			// set address
	LCD_print(ptr);
}

void LCD_printatAtt (int cpl,unsigned int row,unsigned int col,char *ptr,unsigned char attr)
{
int i;

	LCD_address (cpl*row+col);			// set address
	i=LCD_print(ptr);				// imprime, obtiene número de caracteres
	LCD_address (0x1000+cpl*row+col);		// set address de atributos
	LCD_WriteCmd(0xb0);				// Auto data write
	while (i--)					// para cada caracter
		LCD_WriteAData (attr);			// escribe atributos
	LCD_WriteACmd(0xb2);				// reset Auto data write
}

void LCD_CLS()
{
		LCD_address(0);				// TXT screen
		LCD_fill(30*64,0);			// llena con 0's (borra)
}

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

void LCD_plot (int x, int y)
{
int strip,bit;

	strip=x>>3;					// strip = x/8
```

```
        bit=7-x&'\B0111';                       // bit = resto (MSB a la izquierda)
        y=strip+30*y;
        LCD_address(0x1000+y);                  // direcciona strip en GFX screen
        LCD_WriteCmd(0xF8|bit);                 // set bit
}
```

Para mostrar pantallas, podemos, como en la CAN-007, convertirlas a un formato compilable. Afortunadamente, existe un formato binario que coincide con la distribución de memoria, y Dynamic C provee una función que permite, al momento de compilación, leer un archivo y ubicarlo para ocupar un área de memoria en xmem; asociando esa posición de memoria física con un puntero extendido que hace referencia a esa posición. El área apuntada corresponde a una estructura del tipo:

puntero (long) -> longitud de los datos ('n', long)
datos propiamente dichos (n bytes)

De esta forma, resulta muy fácil incluir las imágenes al programa:

```
#ximport "rabbit.im2" rabbit
```

El único inconveniente de trabajar de esta forma, es que acceder a la memoria por su dirección física resulta algo engorroso, por lo que es preferible copiarla al área base y transferirla de allí al display. Dynamic C trae funciones para hacer ésto. Para utilizar poca memoria, partimos la imagen en bloques, movemos estos bloques de una a otra área y copiamos al display. En este caso, serían 8 bloques de 240 bytes c/u.

```
#define IMGCHUNK 240
#define CHUNKS 8

void LCD_dump(long imgdata)
{
int x,y;
unsigned char buffer[IMGCHUNK];

        LCD_address(0x1000);                    // GFX screen
        LCD_WriteCmd(0xb0);                     // Auto data write
        for(y=0;y<CHUNKS;y++){                  // lee bloques de xmem
                xmem2root(&buffer,imgdata+IMGCHUNK*y+sizeof(long),IMGCHUNK);
                for(x=0;x<IMGCHUNK;x++)
                        LCD_WriteAData(buffer[x]);    // escribe bloque en controlador
        }
        LCD_WriteACmd(0xb2);                    // reset Auto data write
}
```

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, mostramos un ejemplo del “modo atributos” y finalmente mostramos una pantalla gráfica.

```
/* MAIN PROGRAM */

main()
{
int i;

        LCD_init();
        while(1){

                LCD_CLS();                      // borra TXT screen
                LCD_fillblack();                // pinta de negro
                MsDelay ( 3000 );               // espera 3 segs
                LCD_fillwhite();                // white screen
                MsDelay ( 3000 );               // espera 3 segs

                LCD_printat(30,4,3,"Cika Electronica S.R.L.");  // texto negro fondo blanco
                MsDelay ( 3000 );               // espera 3 segs
```

```
		LCD_fillblack();                    // invierte imagen (blanco, fondo negro)
		MsDelay ( 3000 );                   // espera 3 segs
		LCD_CLS();                          // borra TXT screen

		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))));
		LCD_printat(30,0,0,"sin(x)");
		MsDelay ( 3000 );                   // espera 3 segs
		LCD_CLS();                          // borra TXT screen
		LCD_fillblack();                    // borra GFX screen
		LCD_WriteCmd(0x84);                 // modo atributeos, CG interno

		LCD_printatAtt(30,1,8,"MODO ATRIBUTOS",0);
		LCD_printatAtt(30,3,8,"Texto Normal",0);
		LCD_printatAtt(30,4,7,"Texto Invertido",5);
		LCD_printatAtt(30,5,3,"Texto Normal Parpadeando",8);
		LCD_printatAtt(30,6,1,"Texto Invertido Parpadeando",13);
		MsDelay ( 6000 );                   // espera 6 segs
		LCD_CLS();                          // borra TXT screen
		LCD_WriteCmd(0x81);                 // TXT XOR GFX, CG interno

		LCD_dump(rabbit);                   // muestra logo Rabbit
		MsDelay ( 6000 );                   // espera 6 segs
	}
}
```