Cika Electrónica

Nota de Aplicación: CAN-075
Título: **Manejo de displays LCD gráficos basados en T6963 con Holtek**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	16/03/07	

La presente nota de aplicación desarrolla un esquema de control para displays LCD gráficos de 128x64 puntos, basados en el controlador T6963C de Toshiba. La razón de la elección de este controlador es que el mismo incluye un generador de caracteres interno, liberándonos de dicha tarea y fundamentalmente del espacio de memoria asociado para almacenar los caracteres. El código desarrollado es assembler, pero se proveen ejemplos de uso del mismo tanto desde C como desde assembler. Aquellos lectores no interesados en el funcionamiento interno del código, pueden saltear dicha parte y acceder directamente a los ejemplos de uso. Para una explicación detallada de los requisitos a cumplir para utilizar C y assembler simultáneamente, se recomienda la lectura del tutorial CTU-010.

Breve descripción del display gráfico

La memoria del display 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 característica 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 auto-incrementado, auto-decrementado, 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 (16 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=16*y+x/8 ; bit=7-resto(x/8).*

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

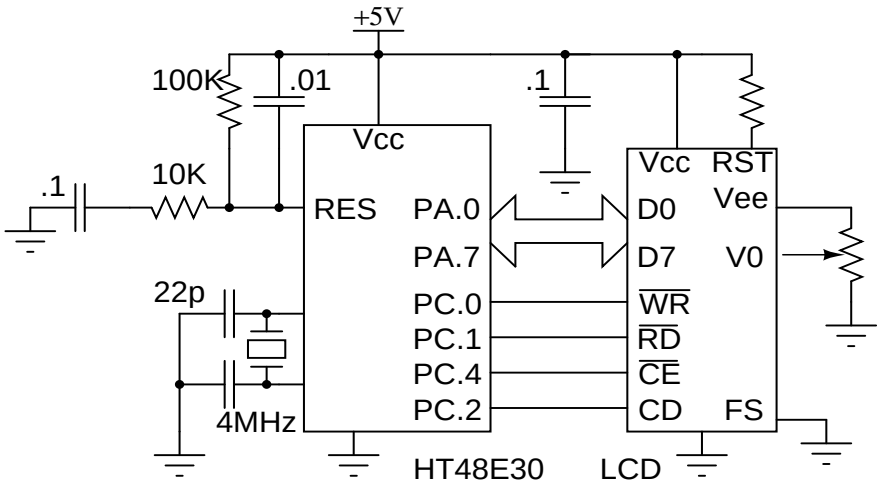
Para graficar, debemos tener en cuenta que la coordenada (0;0) se halla en el extremo superior izquierdo de la pantalla.

Para mostrar íconos, deberemos agrupar los datos de modo tal de poder enviarlos de forma que aproveche de manera eficiente el contador auto-incrementado 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-, 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. Un detalle importante a tener en cuenta es que este formato requiere que el ancho de la imagen sea múltiplo de 16 (número entero de words), caso contrario inserta bytes de padding. También es posible procesar un archivo de tipo BMP, como se ha desarrollado en otras notas de aplicación.

Para imprimir textos, calculamos simplemente la posición de memoria a partir de fila y columna de modo similar: *address=16*fila+columna*, para 16 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.

Hardware

Utilizaremos el port A completo para los datos, y algunos pines del port C para las señales de control. Como la gran mayoría de los microcontroladores, los ports de I/O son bidireccionales. Ambos puertos están presentes con todas las líneas necesarias desde 48E30 en adelante.



Para esta nota no hemos controlado el reset del display, algunas aplicaciones puede que lo requieran.

Software

Las rutinas básicas de control del display son las siguientes:

```
_send_cmd:
                call busy_chk
                mov a,send_cmd0
                jmp snd_cmd
_send_Acmd:
                call Abusy_chk
                mov a,send_Acmd0
snd_cmd:
```

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).

```
snd_:
                mov     lcm_data,a                      ; bus
                clr     lcm_ctrl.ce                     ; CE=0
                clr     lcm_ctrl.wr                     ; WR=0
                set     lcm_ctrl.wr                     ; WR=1
                set     lcm_ctrl.ce                     ; CE=1
                ret

busy_chk:
                mov a,3                                 ; STA0, STA1
                mov bsychk,A
                jmp busy
Abusy_chk:
                mov a,8                                 ; STA3
                mov bsychk,A
busy:           set lcm_data_ctrl                       ; inputs
                set lcm_ctrl.cd                         ; C/D=1 (cmd)
bsylp:          clr lcm_ctrl.ce                         ; CE=0
                clr lcm_ctrl.rd                         ; RD=0
                mov A,lcm_data
                set lcm_ctrl.rd                         ; RD=1
                set lcm_ctrl.ce                         ; CE=1
                and A,bsychk                            ; remueve bits no deseados
                xor A,bsychk                            ; busy ?
                snz Z
                jmp bsylp                               ; no, loop
                clr lcm_data_ctrl                       ; outputs
                ret

_send_data:
                call busy_chk
                mov a,send_data0
                jmp snd_dat
_send_Adata:
                call Abusy_chk
                mov a,send_Adata0
snd_dat:
                clr lcm_ctrl.cd                         ; C/D=0 (dat)
                jmp snd_
```

La inicialización del display se realiza mediante la rutina que sigue a continuación. En la misma, agrupamos los dos bytes de datos que cada comando requiere dentro de una palabra de ROM. El ancho de palabra de ROM de Holtek varía con la capacidad, para el HT48E30 es de 14-bits, lo cual es suficiente para almacenar los valores standard requeridos para la configuración.

```
ilcd            .section INPAGE 'CODE'
_init_LCD:
                mov a,OFFSET initlcd                    ; tabla "dato dato comando"
                mov TBLP,A                              ; puntero
                mov a,4                                 ; elementos de la tabla
                mov lctr,A
ill1:           tabrdc ACC                              ; lee dato1
                call snd_dat                            ; manda
                call busy_chk                           ; espera
                mov a,TBLH                              ; lee dato2 (6-bits)
                call snd_dat                            ; manda
                call busy_chk                           ; espera
                inc TBLP                                ; siguiente word
                tabrdc ACC                              ; lee comando
                call snd_cmd                            ; manda
                call busy_chk                           ; espera
                inc TBLP                                ; siguiente tríada
                sdz lctr
                jmp ill1
                mov a,081h                              ; TXT XOR GFX, CG interno
                call snd_cmd
                call busy_chk
                mov a,09Ch                              ; TXT on, GFX on, no cursor, no blink
                jmp snd_cmd

initlcd: dw 00000h,040h, 00010h,041h, 01000h,042h, 00010h,043h
; TXT home =0, TXT cols = 128, GFX home = 1000h, GFX cols=128
```

Para direccionar la posición de impresión en pantalla, utilizaremos la siguiente rutina:

```
_LCD_addr:
               call busy_chk
               mov a,LCD_addr0
               call snd_dat                    ; LSB
               call busy_chk
               mov a,LCD_addr0[1]
               call snd_dat                    ; MSB
               call busy_chk
               mov a,024h
               jmp snd_cmd                     ; CMD
```

Para llenar una parte de la pantalla con un patrón determinado, utilizaremos la siguiente rutina:

```
; 0: cols to fill
; 1: rows to fill
; 2: pattern
_LCD_fill:
               call busy_chk
               mov a,0B0h                      ; auto data
               call snd_cmd
               mov A,LCD_fill0
               mov lctr,A                      ; columnas
lfl1:          call Abusy_chk
               mov A,LCD_fill2                 ;
               call snd_dat                    ; patrón
               sdz lctr                        ; repite columnas
               jmp lfl1
               call Abusy_chk
               mov a,0B2h                      ; reset auto data
               call snd_cmd
               sdz LCD_fill1                   ; más filas ?
               jmp lfl2
               ret
lfl2:          call busy_chk                   ; siguiente fila
               mov a,010h
               addm A,LCD_addr0
               clr ACC
               adcm A,LCD_addr0[1]
               call _LCD_addr
               jmp _LCD_fill
```

Para imprimir un ícono, utilizaremos la siguiente rutina:

```
; 0: cols
; 1: rows
; 2: icon
_LCD_icon:
               mov A,LCD_icon2                 ; obtiene dirección del ícono
               call seticon
               mov TBLP,A                      ; puntero
lil5:          call busy_chk
               mov a,0B0h                      ; auto data
               call snd_cmd
               mov A,LCD_icon0
               mov lctr,A                      ; columnas
lil1:          call Abusy_chk
               mov A,LCD_icon2
               call geticon                    ; obtiene dato en A
               call snd_dat
               sdz lctr                        ; más columnas ?
               jmp lil1                        ; sí
lil4:          call Abusy_chk                  ; no
               mov a,0B2h                      ; reset auto data
               call snd_cmd
               sdz LCD_icon1                   ; más filas ?
               jmp lil2
               ret
lil2:          call busy_chk                   ; siguiente fila
               mov a,010h
               addm A,LCD_addr0
               clr ACC
```

```
                adcm A,LCD_addr0[1]
                call _LCD_addr
                jmp lil5

si              .section INPAGE 'CODE'
seticon:
                addm a,PCL                          ; obtiene dirección del ícono
                ret a,OFFSET icon0d
                ret a,OFFSET icon1d
                ;

gi              .section INPAGE 'CODE'
geticon:
                addm a,PCL                          ; devuelve dato
                jmp icon0
                jmp icon1
```

Los íconos los definimos de la siguiente forma, debido a que la instrucción TABRDC debe estar en la misma página que la data que lee (lee de la página corriente):

```
icon0s  .section INPAGE 'CODE'
icon0:          tabrdc ACC
                inc TBLP
                ret
icon0d: ; 48x33, 6 cols, 33 rows
        DW      000h,000h,000h,008h,000h,000h,000h,000h,000h,01Ch,000h,000h
        ...
icon1s  .section INPAGE 'CODE'
icon1:          tabrdc ACC
                inc TBLP
                ret
icon1d: ; 64x25, 8 cols, 25 rows
        DW      001h,0FEh,078h,001h,0FCh,000h,000h,000h,001h,0F4h,07Eh,003h
        ...
```

Para encender un punto determinado, utilizaremos la siguiente rutina:

```
; 0: x
; 1: y
_LCD_plot:
                mov a,LCD_plot0                     ; x
                and a,07h
                mov lctr,a
                mov a,7
                subm a,lctr                         ; 7-x%8 en lctr
                clr C
                rrc LCD_plot0
                clr C
                rrc LCD_plot0
                clr C
                rrc LCD_plot0                       ; x/8 en LCD_plot0
                clr LCD_addr0[1]
                clr C
                rlca LCD_plot1
                mov LCD_addr0,a
                rlc LCD_addr0[1]
                rlc LCD_addr0
                rlc LCD_addr0[1]
                rlc LCD_addr0
                rlc LCD_addr0[1]
                rlc LCD_addr0
                rlc LCD_addr0[1]                    ; y*16 en LCD_addr
                mov a,LCD_plot0
                addm A,LCD_addr0                    ; + x/8
                mov A,010h
                adcm A,LCD_addr0[1]                 ; + 1000h
                call _LCD_addr                      ; setea dirección del punto
                call busy_chk
                mov A,lctr
                or a,0F8h                           ; comando "set bit"
                jmp snd_cmd
```

Proveeremos ahora un par de rutinas para imprimir strings. Debido a que la arquitectura de Holtek es Harvard, dispondremos de una rutina para textos dinámicamente generados, es decir, en RAM, y otra para textos fijos, en ROM. Los strings deben ubicarse a continuación de la subrutina, debido a la utilización de la instrucción TABRDC, que lee una tabla en la misma página de flash. Si el espacio no alcanza, deberá modificarse el código para realizar un call en lugar de esta instrucción, como hemos hecho con los íconos.

```
_send_string_RAM:
               mov A,send_string_RAM0
               mov MP0,A
               call busy_chk
               mov a,0B0h                          ; auto data
               call snd_cmd
ssloop:        call Abusy_chk
               mov A,IAR0
               sz ACC
               jmp ssl1
               call Abusy_chk
               mov a,0B2h                          ; reset auto data
               jmp snd_cmd
ssl1:          sub A,020h                          ; ASCII -> T6963
               call snd_dat
               inc MP0
               jmp ssloop

ssrom   .section INPAGE 'CODE'
_send_string_ROM:
               call busy_chk
               mov A,0B0h                          ; auto data
               call snd_cmd
               mov A,send_string_ROM0
               call setstring
               mov TBLP,A
ss2loop:
               call Abusy_chk
               tabrdc ACC
               sz ACC                              ; fin de string ?
               jmp ss2l1                           ; no, print
               mov A,0B2h                          ; auto data
               jmp snd_cmd
ss2l1:         sub A,020h                          ; ASCII -> T6963
               call snd_dat
               inc TBLP                            ; siguiente
               jmp ss2loop

; strings
txt1:   dw     67,105,107,97,32,69,108,101,99,116,114,111,110,105,99,97,0
txt2:   dw     72,111,108,116,101,107,32,76,67,68,32,100,101,109,111,0

ss             .section INPAGE 'CODE'
setstring:
               addm a,PCL
               ret a,OFFSET txt1
               ret a,OFFSET txt2
               ;
```

Utilización desde C

Incluimos un header file, lcd.h, de modo que el compilador pueda chequear la sintaxis y el usuario conozca las funciones que debe llamar.

Inicialización

Para inicializar el LCD, realizamos lo siguiente:

```
               _pa=0;
               _pac=0x00;                          // PA = salidas
               _pc=0x17;                           // (CS,RD,WR,CD=1)
               _pcc=0;                             // PC = salidas
               _delay(65535);
```

```
INIT_LCD();
```

Direccionamiento de la posición de impresión
Para direccionar la posición de impresión en pantalla, llamaremos a la rutina correspondiente:

```
LCD_ADDR(0);                        // inicio del área de texto
```

Borrado de pantalla e impresión de patrones
Para borrar la pantalla lo que hacemos es llenarla con blancos. Para ello direccionamos la pantalla correspondiente, texto o gráficos, y llamamos a la rutina de impresión de patrones:

```
LCD_ADDR(0);                        // área de texto
LCD_FILL(16,8,0);                   // borra área de 16 columnas x 8 filas
LCD_ADDR(0x1000);                   // área gráfica
LCD_FILL(16,64,0);                  // borra área de 16x8=128 ancho x 64 alto
```

Los parámetros que recibe la función *LCD_fill()* son: el ancho en bytes o caracteres, el alto en líneas (para texto 1 a 8, para gráficos 1 a 64), y el patrón a imprimir.

Envío de comandos al display
Podemos enviar comandos al display llamando a la rutina correspondiente:

```
SEND_CMD(0x84);                     // modo atributos, CG interno
```

Strings en RAM
Para imprimir un texto en RAM, llamamos a la rutina correspondiente con la dirección del texto como parámetro:

```
SEND_STRING_RAM(buffer);
```

Strings en ROM
El ejemplo siguiente muestra la forma de imprimir un string en ROM. El string corresponde a los definidos en assembler en la sección anterior, y deberá cargarse en dicho archivo (o modificarlo para que los incluya.desde otro).

```
SEND_STRING_ROM(0);
```

Si bien el archivo provisto tiene los strings ahí mismo, es posible incluirlos desde otro, generado por una simple rutina que toma los strings de un archivo de texto y los convierte a las cadenas en decimal o hexa necesarias.

Si deseamos manejar los strings desde C, el tema es algo más complicado. Debido a que la arquitectura de Holtek es Harvard, no es simple pasar un puntero a una constante como parámetro. El compilador C que el fabricante provee no lo soporta, por lo que deberemos definir al string como un array y tratarlo como tal. Una posibilidad es escribir una pequeña rutina para cada string, que compilada ocupa unos pocos bytes:

```
const char str00[16]= "Cika Electronica";
const char str01[15]= "Holtek LCD demo";

SEND_CMD(0xB0);                     // auto data
for(i=0;i<16;i++)
       SEND_ADATA(str00[i]-0x20);
SEND_ACMD(0xB2);                    // reset auto data
LCD_ADDR(16*7);                     // línea 7
SEND_CMD(0xB0);                     // auto data
for(i=0;i<15;i++)
       SEND_ADATA(str01[i]-0x20);
SEND_ACMD(0xB2);                    // reset auto data
```

Otra alternativa es armar una tabla de strings y pasar a una subrutina el número de string, el inconveniente es que todos los strings de la tabla ocupan el mismo espacio, es decir, deben tener el mismo largo:

```
const char strs[2][16]= {
"Cika Electronica",
"Holtek LCD demo "
```

```
};

void ss(unsigned int string, unsigned int len)
{
unsigned int i;
               SEND_CMD(0xB0);                      // auto data
               for(i=0;i<len;i++)
                       SEND_ADATA(strs[string][i]-0x20);
               SEND_ACMD(0xB2);                     // reset auto data
}
```

Dicha subrutina la llamamos de la siguiente forma:

```
       ss(0,16);
       LCD_ADDR(16*3);
       ss(1,16);
```

Atributos de texto

Para enviar atributos de texto, podemos utilizar la rutina de impresión de patrones, dado que genralmente aplicamos el mismo atributo a un conjunto de caracteres:

```
               SEND_CMD(0x84);                      // modo atributos, CG interno
               LCD_ADDR(16*3);
               ss(2,15);                            // imprime string en línea 3
               LCD_ADDR(16*3+0x1000);               // zona de atributos de línea 3
               LCD_FILL(15,1,5);                    // atributo: negativo
               LCD_ADDR(16*4);
               ss(3,16);                            // 4
               LCD_ADDR(16*4+0x1000);
               LCD_FILL(16,1,8);                    // atributo: parpadea
               LCD_ADDR(16*5);
               ss(4,16);                            // 5
               LCD_ADDR(16*5+0x1000);
               LCD_FILL(16,1,13);                   // atributo: parpadea + negativo
```

Íconos

Para imprimir un ícono, llamamos a la rutina correspondiente con el número de ícono como parámetro. Los íconos se definen en el archivo assembler que tiene las rutinas de soporte; puede hacerse que éste incluya a otro archivo, en el cual se alojan los íconos, generados por una simple rutina[3] que convierte los bytes a las cadenas en decimal o hexa necesarias.

```
               LCD_ADDR(0x1000);                    // arriba a la izquierda
               LCD_ICON(6,33,0);                    // 6x8=48 ancho x 33 alto, ícono 0
               LCD_ADDR(0x1208);                    // abajo a la derecha
               LCD_ICON(8,25,1);                    // 8x8=64 ancho x 25 alto, ícono 1
```

Los parámetros que recibe dicha función son: ancho en bytes, alto en líneas, y número de ícono

Para borrar el ícono, simplemente imprimimos un patrón en blanco de iguales dimensiones en la misma dirección:

```
               LCD_ADDR (0x1105);
               LCD_ICON(6,33,0);                    // muestra
               LCD_ADDR (0x1105);
               LCD_FILL(6,33,0);                    // borra
```

Líneas y puntos

Para imprimir líneas o puntos, utilizamos la función *LCD_PLOT()*:

```
               for(i=0;i<128;i++)
                       LCD_PLOT(i,32);
               for(i=0;i<64;i++)
                       LCD_PLOT(64,i);
```

Los parámetros que recibe dicha función son las coordenadas del punto, x e y respectivamente.
Para líneas en diagonal, se deberá recurrir a un algoritmo como el de Bresenham, disponible en otras notas de aplicación

3 Un ejemplo de esto es *bin2c*, disponible en la Internet, con menores modificaciones

Utilización desde assembler

No incluiremos en esta nota ejemplos de utilización desde assembler, la operatoria es realmente simple y corresponde al llamado de cada una de las rutinas correspondientes, colocando los parámetros en las variables correspondientes, como se describe en el tutorial CTU-010 y se ejemplifica en CAN-074