Nota de Aplicación: CAN-098
Título: **Utilización de displays LCD gráficos (HD61202) con Holtek ARM HT32F**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	22/02/12	port de CAN-003 y CAN-059, 060, 078

Nos interiorizaremos ahora en el desarrollo de una interfaz para conectar un módulo LCD gráfico inteligente a un microcontrolador de 32-bits de la familia HT32F de Holtek, basada en el core ARM Cortex-M3: el HT32F1253, con 32K de flash y 8KB de RAM. Como display 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.

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, deberemos tener en cuenta el controlador que posee el display. Dado que el micro es de 3,3V y posee algunos pines 5V-tolerant, podemos usar displays basados en KS0108 sin problemas, dado que éste acepta los niveles de tensión del HT32, pero si nuestro display posee un HD61202, deberemos intercalar algún traslador de nivel como por ejemplo 74HCT245, y controlar el sentido de habilitación de éste.

HT32F1253 LCD

PA.8 --------- D0
PA.9 --------- D1
PB.15 --------- D2
PA.11 --------- D3
PA.12 --------- D4
PA.13 --------- D5
PA.14 --------- D6
PA.15 --------- D7

PB.10 --------- I/D
PB.11 --------- R/W
PB.12 --------- E
PB.13 --------- CS1
PB.14 --------- CS2

Un último detalle: el pin PA.10 del HT32F1253 se comparte con la circuitería de selección de booteo y test del micro. Es recomendable no utilizarlo o hacerlo sólo como salida, por lo que en este caso lo dejamos sin utilizar.

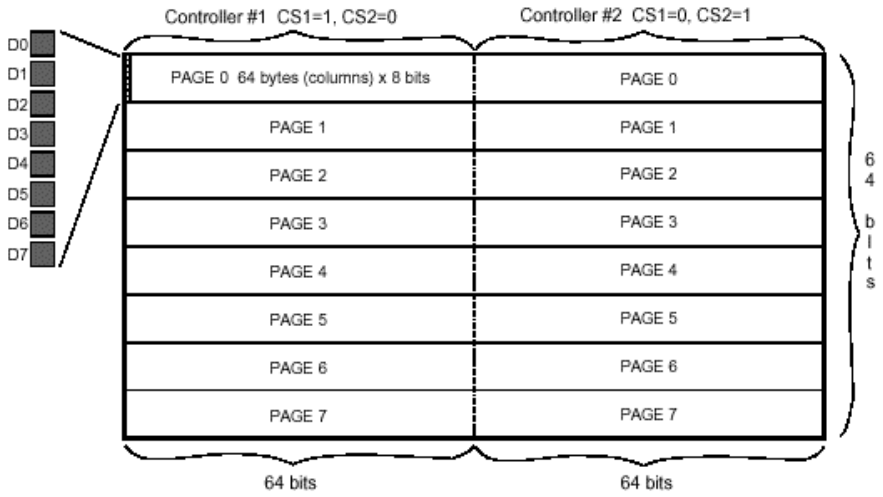
Software

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.

Tipografías

En esta nota de aplicación, utilizaremos tres tipografías: 5x7, 8x8, y 8x15. La tipografía de 7 puntos de alto cabe perfectamente dentro de los 8 bits de la página, manteniendo un punto de separación entre líneas; la de 8x8 deja por sí misma espacio entre líneas, y la de 8x15 ocupará dos "renglones", de modo de proveer un pixel de separación entre líneas. En sentido horizontal, haremos que un caracter tenga todos sus puntos dentro 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 mostrar textos en una matriz imaginaria de 20 caracteres por 8 líneas. Para las otras dos tipografías, como 64 es divisible por 8, tendremos 16 caracteres (8 en cada controlador) en sentido horizontal, y 8 ó 4 renglones respectivamente.

Para generar los sets de caracteres que se incluyen en esta nota de aplicación, tomamos tipografías de dominio público disponibles en Internet y las rotamos al formato de los controladores utilizados. Generalmente, los sets de caracteres se hallan definidos a un byte por línea horizontal, n líneas de arriba a abajo, caracter por caracter. La tarea a realizar consiste en transferir esa información, caracter por caracter, al formato del HD61202: un byte por línea vertical, n líneas de izquierda a derecha, caracter por caracter. En la CAN-041 dispone de software de ayuda

Bitmaps

Los bitmaps fueron generados según la información brindada en CAN-022.

Desarrollo

El listado del código en C se encuentra en el archivo adjunto. Desarrollamos a continuación las partes de bajo nivel que revisten especial consideración para este micro en particular. Los detalles pueden consultarse en las notas anteriores, de las que ésta resulta.

La implementación de CMSIS de Holtek, al momento de escribir esta nota, no incluye macros para poder acceder a los pines de I/O bit a bit. Sin embargo, esto es muy simple de resolver mediante algunas macros que aplican la definición standard del área de bit banding y el álgebra correspondiente.

```
/*      Hardware definitions for graphic LCD displays

                PA.8,9,11-15   I/O     LCD D0,1,3-7
                PB15                   I/O     LCD D2

                PB.12   Out     LCD E
                PB.10   Out     LCD ID
                PB.11   Out     LCD R/W_

                PB.13   Out     LCD CS1
                PB.14   Out     LCD CS2
*/

/* LCD control signals */
#define LCD_CS1b        13
#define LCD_CS2b        14
#define LCD_Eb          12
#define LCD_IDb         10
#define LCD_RWb         11

// Bit-band alias = Bit-band base + (byte offset * 32) + (bit number * 4)
/* Bit-Band for Device Specific Peripheral Registers */
#define BITBAND_PERI(addr, bitnum) (HT_PERIPH_BB_BASE + (((uint32_t)(addr) - HT_PERIPH_BASE)
<< 5) + ((uint32_t)(bitnum) << 2))

#define LCD_CS1 (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_CS1b)))
#define LCD_CS2 (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_CS2b)))
#define LCD_E   (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_Eb)))
```

```
#define LCD_ID  (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_IDb)))
#define LCD_RW  (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_RWb)))

/* LCD status bits */
#define LCD_BUSY      7

/* Demora para el timing del controlador. A 72 MHz -> ~100ns/cuenta + call + ret (no toma en
cuenta wait-states en flash (2) */
void DILEI(int x){volatile int mydelay;mydelay=(x);while(mydelay--);}

void LCD_Write(unsigned char dat)
{
  /* HD61202U: 150ns address setup, 200ns data setup, 450ns E width */
        HT_GPIOA->DOUTR = (dat<<8);           // escribe datos
        if(dat&(1<<2))
                HT_GPIOB->DOUTR |= (1<<15);   // ídem PB.15
        else
                HT_GPIOB->DOUTR &= ~(1<<15);
        HT_GPIOA->DIRCR |= 0xFB00;            // PA.8,9,11-15 salidas, pone datos en el bus
        HT_GPIOB->DIRCR |= (1<<15);           // PB-15 salida
        LCD_RW=0;                             // Baja RW (Write)
        DILEI(1);
        LCD_E=1;                              // Sube E
        DILEI(3);
        LCD_E=0;                              // Baja E
        DILEI(2);
}

unsigned char LCD_Read()
{
  /* HD61202U: 150ns address setup, 450ns E width, 320ns data delay */
unsigned char dat;

        LCD_RW=1;                             // Sube RW (Read)
        HT_GPIOA->DIRCR &= ~0xFB00;          // PA.8,9,11-15 entradas
        HT_GPIOB->DIRCR &= ~(1<<15);         // PB15 entrada
        DILEI(1);
        LCD_E=1;                              // Sube E
        DILEI(2);
        dat=(unsigned char)(HT_GPIOA->DINR >> 8);    // lee datos
        if(HT_GPIOB->DINR & (1<<15))                 // mapea PB15 a D2
                dat |= (1<<2);
        else
                dat &= ~(1<<2);
        DILEI(1);
        LCD_E=0;                              // Baja E
        DILEI(2);
        return(dat);
}

void LCD_setupI0 ()
{
        HT_CKCU->APBCCR0 |= (1<<17)+(1<<16)+(1<<14); /* 17=PBEN, 16=PAEN, 14=AFIOEN
                                              habilita APB clocks en GPIOA, GPIOB, y AFIO */
        HT_AFIO->GPACFGR = 0x54000000;        // PA15,14,13 -> AF1 (chau SWD, no debugging)
        HT_AFIO->GPBCFGR = 0x00100000;        // PB10 -> AF1
        HT_GPIOA->INER &= ~0x0400;            // inhibe entrada en PA.10
        HT_GPIOA->INER |= 0xFB00;             // habilita entradas en PA.8,9,11-15
        HT_GPIOB->INER |= 0x8000;             // habilita entrada en PB.15
        HT_GPIOA->DIRCR |= 0xFF00;            // PA.8-15 salidas
        HT_GPIOB->DIRCR |= 0xFC00;            // PB.10,11,12,13,14,15 salidas
        LCD_E=0;
}
```

Los nombres de los pines de I/O corresponden a CMSIS v2.0. Nótese que `HT_GPIOA->DIRCR` apunta al registro de control del port A, en el cual configuramos los pines como salida o como entrada. Por defecto, los clocks hacia cada periférico se encuentran inhabilitados, por lo que antes de utilizar un puerto de I/O debemos habilitarle el clock correspondiente en el bus de periféricos (APB), lo cual realizamos con el registro `HT_CKCU->APBCCR0`. El registro INER habilita la circuitería de entrada, si el pin es entrada o salida se controla mediante DIRCR.