Nota de Aplicación: CAN-099
Título: **Utilización de displays LCD gráficos (SED1335) con Holtek ARM HT32F**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	22/02/12	port de CAN-005 y CAN-076

Nos interiorizaremos ahora en el desarrollo de una interfaz para conectar un módulo LCD gráfico inteligente de 320x240 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 utilizamos un Powertip PG320240, display basado en chips controladores compatibles con el SED1335, de S-MOS, y su clon de Epson.

Hardware

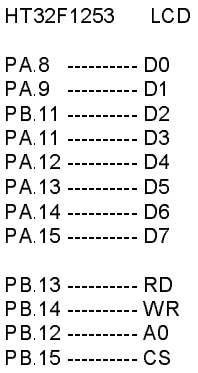
El SED1335 presenta una interfaz con dos posibles modos de trabajo: tipo Motorola (E, RS, R/W) o tipo Intel (RD, WR, A0). El PG320240FRS de Powertip lo utiliza en esta última modalidad.

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 5V-tolerant, podemos usar displays basados en S1D13305 y SED1335 sin problemas, dado que éstos aceptan los niveles de tensión del HT32F1253.

El circuito de contraste de este display es totalmente interno, el mismo puede ajustarse mediante un preset ubicado en la parte posterior del display.

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.

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

Breve descripción del display gráfico

Estos displays son sumamente versátiles, la memoria puede ser dividida en diferentes pantallas, las cuales, a su vez pueden definirse como gráficas o de texto, habilitarse y/o superponerse independientemente. Para el caso del modo texto, el SED1335 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.

La estructura de memoria de cada pantalla es lineal, tanto en modo gráfico como en modo textol. 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 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 el offset desde la primera dirección correspondiente a la pantalla que nos ocupa. 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 puede funcionar a una alta velocidad de acceso, simplemente da prioridad a la interfaz con el procesador e interrumpe el display. Si nuestra aplicación requiere que no haya

parpadeo (flicker) al momento de escritura, podemos primero chequear el flag de ocupado (busy) y realizar nuestra escritura en el momento que el controlador no accede a la RAM para refrescar el LCD. Si, por el contrario, toleramos el flicker o, como en nuestro caso, hacemos una escritura muy rápida que el mismo no se nota, podemos obviar un paso y escribir directamente sobre el controlador sin chequear el busy flag.

Algoritmos

Si bien el display tiene muchas formas de utilización, en esta nota de aplicación desarrollaremos algoritmos en base a las más simples.

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 (40 si utilizamos la totalidad de 320 pixels=40 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=40*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 los contadores autoincrementados 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 BMP, veríamos que son muy similares, por ejemplo: BMP va de abajo a arriba y el display de arriba a abajo, por lo que la imagen se ve espejada verticalmente; BMP usa el '0' para el negro y el '1' para el blanco, si el display es STN+, '1' corresponde a un punto negro. Además, BMP incluye un encabezado de 62 bytes.

Por consiguiente, para adaptar una imagen, debemos llevarla a la resolución deseada, espejarla verticalmente, invertir sus colores, salvarla en formato BMP y por último descartar los 62 bytes del comienzo con algún editor hexa. En el caso de íconos, si la cantidad de bytes por línea no es par, BMP agregará bytes de padding, y según el programa que utilicemos para salvar, puede que la inversión de colores no funcione, puesto que sólo invierte la paleta, que no utilizamos. Dado que luego generaremos un header file, es bueno tener una opción en el programa de conversión para invertir el contenido de los bytes.

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 8x8).

Para mostrar textos más grandes en el display, los armaremos sobre la pantalla gráfica. Para esto, cada tipografía se alojará en flash como una sucesión de bytes en el formato del display, es decir, de arriba hacia abajo. Si bien el código tal como está no soporta caracteres de más de 8 pixels, es fácil extenderla a 16 pixels tomando de a dos por fila. La rutina desarrollada ubica el inicio del caracter a imprimir en la flash, tomando luego la cantidad de bytes necesarios según la altura de la tipografía, y repitiendo el procedimiento para cada caracter del string.

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
                PB11            I/O     LCD D2

                PB.13   Out     LCD RD
                PB.14   Out     LCD WR
                PB.12   Out     LCD A0
                PB.15   Out     LCD CS
*/
```

```
/* LCD control signals */
#define LCD_RDb        13
#define LCD_A0b        12
#define LCD_WRb        14
#define LCD_CSb        15
#define LCD_D2b        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_RD  (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_RDb)))
#define LCD_A0  (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_A0b)))
#define LCD_WR  (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_WRb)))
#define LCD_CS  (*((__IO uint32_t *)BITBAND_PERI(&HT_GPIOB->DOUTR,LCD_CSb)))

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

void LCD_Write(unsigned char dat)
{
/* S1D13305: 0ns address setup, 120ns data setup, 120ns strobe width */

        HT_GPIOA->DOUTR = (dat<<8);          // escribe datos
        if(dat&(1<<2))                       // ídem D2b (PB.11)
                HT_GPIOB->DOUTR |= (1<<LCD_D2b);
        else
                HT_GPIOB->DOUTR &= ~(1<<LCD_D2b);
        HT_GPIOA->DIRCR |= 0xFB00;           // PA.8,9,11-15 salidas, pone datos en el bus
        HT_GPIOB->DIRCR |= (1<<LCD_D2b);     // LCD_D2 salida
        LCD_CS=0;                            // Baja CS
        LCD_WR=0;                            // Baja WR
        DILEI(1);
        LCD_WR=1;                            // Sube WR
        LCD_CS=1;                            // Sube CS
}

unsigned char LCD_Read()
{
/* S1D13305: 0ns address setup, 50ns data delay, 120ns strobe width */
unsigned char dat;

        HT_GPIOA->DIRCR &= ~0xFB00;          // PA.8,9,11-15 entradas
        HT_GPIOB->DIRCR &= ~(1<<LCD_D2b);    // LCD_D2 entrada
        LCD_CS=0;                            // Baja CS
        LCD_RD=0;                            // Baja RD
        dat=(unsigned char)(HT_GPIOA->DINR >> 8);    // lee datos
        if(HT_GPIOB->DINR & (1<<LCD_D2b))            // mapea LCD_D2 a D2
                dat |= (1<<2);
        else
                dat &= ~(1<<2);
        LCD_RD=1;                            // Sube RD
        LCD_CS=1;                            // Sube CS
        return(dat);
}

void LCD_setupIO()
{
        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_GPIOA->INER &= ~0x0400;            // inhibe entrada en PA.10
        HT_GPIOA->INER |= 0xFB00;             // habilita entradas en PA.8,9,11-15
        HT_GPIOB->INER |= (1<<LCD_D2b);       // habilita entrada en LCD_D2
        HT_GPIOB->DOUTR |= (1<<LCD_CSb)+(1<<LCD_WRb)+(1<<LCD_RDb);
        HT_GPIOB->DIRCR |= 0xF800;            // PB.11,12,13,14,15 salidas
        HT_GPIOA->DIRCR |= 0xFF00;            // PA.8-15 salidas
}
```

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.