Cika Electrónica

Nota de Aplicación: CAN-012
Título: **Utilización de displays LCD gráficos (SED1335) con Rabbit (bus)**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	03/10/03	

Desarrollamos una interfaz para conectar un módulo LCD gráfico inteligente Powertip PG320240FRS, a un módulo Rabbit 2000, pero esta vez al bus, dejando libres los ports de I/O para otra aplicación. Dado que ya hemos desarrollado software para este display, solamente comentaremos sobre la conexión al bus y su software asociado. Se recomienda al lector consultar la CAN-005 para una descripción completa de la aplicación.

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 no es necesario ningún tipo de glue-logic, hacemos una conexión directa entre el bus del Rabbit y el LCD, como puede apreciarse en la tabla a la derecha:

Rabbit	LCD
D0	D0
D1	D1
D2	D2
D3	D3
D4	D4
D5	D5
D6	D6
D7	D7
IORD	RD
IOWR	WR
A0	A0
PE.7	CS
RST	RST

Las señales RD y WR se toman de las correspondientes para operaciones en el espacio de I/O: IORD y IOWR. En muchos de los módulos, están señales están disponibles luego de un buffer, bajo el nombre BIORD y BIOWR. Lo mismo ocurre con el bus de datos y el de direcciones (BDx y BAx). La señal CS es generada utilizando la facilidad de IOSTROBE que provee Rabbit. Se asigna el espacio de I/O utilizado, se configura el pin, y éste funciona como CS para ese rango de direcciones. Elegimos utilizar el port PE.7, el cual responde al rango de direcciones 0xE000 a 0xFFFF, dentro de este rango, las direcciones pares acceden a un registro del controlador del display (A0=0) y las impares al otro (A0=1).

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

El display dispone, además, de un pin de reset, el cual podemos controlar a voluntad o conectar al reset del circuito, como hemos hecho.

Software

No ahondaremos en el display y su estructura interna, para tal información remitimos al lector a la CAN-005.

Una característica interesante de este 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.

Dada la gran cantidad de información a mover hacia el display para la presentación de pantallas, decidimos escribir la rutina básica de escritura en assembler.

```
#asm
;la función requiere dos parametros:
;@sp+2= address
;@sp+4= data
```

```
;
LCD_Write::

        ld hl,(sp+4)                ; dato (LSB)
        ld a,l
        ld hl,(sp+2)                ; address (LSB)
        ioe ld (HL),a               ; escribe
        ret

#endasm
```

El prefijo *ioe* es el que nos permite utilizar cualquier instrucción de acceso a memoria como instrucción de I/O, en este caso en un espacio de direccionamiento externo (bus). Esta es una de las mayores diferencias entre Rabbit y Z-80, en cuanto a set de instrucciones; descartando, claro está las funciones agregadas.

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.

Para configurar el controlador, utilizaremos entonces la dirección 0xE001 para el comando (impar, A0=1) y la dirección 0xE000 para los datos (par, A0=0)

```
void LCD_WriteStrCmd(unsigned char *cmd,int len)
{
        LCD_Write(0xE001,*cmd++);
        while(len--) {
                LCD_Write(0xE000,*cmd++);
        }
}

#define LCD_ReadData() RdPortE(0xE001)
```

Veamos ahora la función de inicialización, donde hemos omitido información redundante y nos concentramos solamente en lo pertinente al seteo del port E. La inicialización del chip en sí es igual a la hecha en la CAN-005.

```
void LCD_init ()
{
<se eliminaron partes para mayor claridad, consulte el software adjunto o la CAN-005>
// Usamos Port E bit 7 para el LCD_CS, configurado como I/O chip select con 3 wait-states
#define LCDSTROBE      0x80
#define LCDCSREGISTER  IB7CR
#define LCDCSSHADOW    IB7CRShadow
#define LCDCSCONFIG    0x88

        // Inicializamos el bit 7 del Port E como I/O pin normal
        WrPortI(PEFR, &PEFRShadow, (PEFRShadow|LCDSTROBE));

        // lo seteamos como salida
        WrPortI(PEDDR, &PEDDRShadow, (PEDDRShadow|LCDSTROBE));

        // Inicializamos el bit 7 del Port E para funcionar como chip select.
        WrPortI(LCDCSREGISTER, &LCDCSSHADOW, LCDCSCONFIG);

        // Seteamos el bit 7 del Port E controlado (clocked) por PCLK/2
        WrPortI(PECR, &PECRShadow, (PECRShadow & ~0xFF));
<se eliminaron partes para mayor claridad, consulte el software adjunto o la CAN-005>
}
```

Y esto es todo lo que necesitamos para tener nuestro display funcionando. El software que acompaña a esta nota incluye además las funciones desarrolladas en la CAN-005 y la demostración.