Cika Electrónica

Nota de Aplicación: CAN-033
Título: **Utilización de displays LCD color con controladores S1D13705 y Rabbit**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	15/06/05	

Modificamos levemente el desarrollo de la CAN-032, para trabajar en modo 8bpp, es decir 256 colores de una paleta de 4096

Algoritmos

Para ubicar un punto en pantalla, calculamos su posición en memoria sabiendo que alojamos un pixel por byte, es decir: $mem = x + 320 \cdot y$.

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 la estructura de memoria. Si comparamos la estructura de memoria del display con la forma de guardar imágenes en 256 colores 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. Además, BMP incluye un encabezado que contiene la paleta de colores.

Por consiguiente, para adaptar una imagen, debemos llevarla a la resolución deseada, reducirla a 256 colores, espejarla verticalmente, salvarla en formato BMP y por último descartar los 1078 bytes del comienzo. Entre los bytes a descartar tomaremos los bytes 54 a 1077, los cuales corresponden a la paleta en formato BGR0 (4 bytes), y la guardaremos como RGB. Dado que esto es algo más tedioso que para 16 colores, se recomienda escribir un pequeño programita que lo haga

Configuración del S1D13705

Los valores a setear en cada uno de los registros se obtienen de idéntica forma que en la CAN-032, con la salvedad de indicar 8bpp en la solapa *Preferences*. Exportamos de igual modo los datos en un archivo de tipo *C header file* (s1d13705.h), el cual podremos editar e incluir en el código para Rabbit

Software de bajo nivel

Nuevamente tenemos un bus de 16-bits simulado dentro de uno de 8-bits, con señalización similar a la utilizada por el 8086, con la complicación adicional que nuestro bus de direcciones es ahora de 17-bits, y los punteros de nuestro procesador son de 16-bits. Recibiremos entonces la dirección como un *long*, es decir, 32 bits, y alojaremos A0 a A15 en un par de registros y A16 en un registro (LSB). Obviamente, dado que ahora la cantidad de información es aún mayor, utilizamos assembler para las rutinas críticas.

```
/* VERSIONS FOR LONG ADDRESS */

#asm root
read13705:
       call addressbus              ; put addresses and BHE
       ld hl,PCDR                   ; apunta a port paralelo
       ioi res 2,(HL)               ; baja CS
       ld hl,PBDR                   ; apunta a port paralelo
       ioi res 7,(HL)               ; baja RD
rwait:
      ioi ld a,(hl)                 ; lee port paralelo
      rrca                          ; chequea bit 0 (WAIT)
      jr nc,rwait                   ; espera hasta que NOWAIT
      ioi ld a,(PADR)               ; lee dato
```

```
      ioi set 7,(HL)                ; sube RD
      ld hl,PCDR                    ; apunta a port paralelo
      ioi set 2,(HL)                ; sube CS
      ld h,0
      ld l,a
      ret

;sp+2: A0-A15
;sp+4: A16
;sp+6: data
write13705::
      call addressbus               ; put addresses and BHE
      ld hl,PCDR                    ; apunta a port paralelo
      ioi res 2,(HL)                ; baja CS
      ld hl,(sp+6)                  ; lee dato
      ld a,l
      ioi ld (PADR),a               ; pone dato
      ld a,0x84
      ioi ld (SPCR),a               ; A= salidas
      ld hl,PBDR                    ; apunta a port paralelo
      ioi res 6,(HL)                ; baja WE0
wwait:
      ioi ld a,(hl)                 ; lee port paralelo
      rrca                          ; chequea bit 0 (WAIT)
      jr nc,wwait                   ; espera hasta que NOWAIT
      ioi set 6,(HL)                ; sube WE0 (must be >1/BCLK after NOWAIT)
      ld hl,PCDR                    ; apunta a port paralelo
      ioi set 2,(HL)                ; sube CS (en shadow)
      ld a,0x80
      ioi ld (SPCR),a               ; A= entradas
      ret

;sp+2: dirección de retorno de la función que nos llama
;sp+4: A0-A15
;sp+6: A16
;sp+8: data
addressbus:
       ld hl,(sp+4)                 ; lee address
       ex de,hl                     ; en DE
       ld hl,(sp+6)                 ; lee address A16
       ld a,l                       ; en A
ab2:   ld hl,PEDR                   ; apunta a port
       ioi ld (hl),d                ; pone parte alta
       ld hl,PDDR                   ; apunta a port
       ioi ld (hl),e                ; pone parte baja
       ld hl,PCDR                   ; apunta a port paralelo
       ioi set 0,(hl)               ; BHE=1
       rrc e                        ; debo activar BHE ? (test LSB=1)
       jr nc,ok1                    ; no
       ioi res 0,(HL)               ; sí, BHE=0
ok1:
       ioi res 4,(HL)               ; A16=0
       rrca                         ; A16=1 ?
       jr nc,ok3                    ; no
       ioi set 4,(HL)               ; sí, A16=1
ok3:
       ret
```

Para poder observar algo, deberemos definir los colores, cargando la paleta. La rutina y la estructura de paleta son idénticas a las utilizadas en 4bpp, excepto que habrá 256 elementos RGB en la paleta.

A continuación, la inicialización del chip. Los valores los obtuvimos utilizando el software de configuración provisto por el fabricante, según comentáramos.

```
typedef unsigned char S1D_INDEX;
typedef unsigned char S1D_VALUE;
typedef struct
{
    S1D_INDEX Index;
    S1D_VALUE Value;
} S1D_REGS;
```

```
const static S1D_REGS aS1DRegs[] =
{
    { 0x00,             0x24 },       // Revision Code Register
    { 0x01,             0x3B },       // Mode Register 0 Register
    { 0x02,             0xD0 },       // Mode Register 1 Register
    { 0x03,             0x03 },       // Mode Register 2 Register (enable display)
    { 0x04,             0x27 },       // Horizontal Panel Size Register
    { 0x05,             0xEF },       // Vertical Panel Size Register (LSB)
    { 0x06,             0x00 },       // Vertical Panel Size Register (MSB)
    { 0x07,             0x1F },       // FPLINE Start Position Register
    { 0x08,             0x00 },       // Horizontal Non-Display Period Register
    { 0x09,             0x00 },       // FPFRAME Start Position Register
    { 0x0A,             0x0B },       // Vertical Non-Display Period Register
    { 0x0B,             0x00 },       // MOD Rate Register
    { 0x0C,             0x00 },       // Screen 1 Start Address Register (LSB)
    { 0x0D,             0x00 },       // Screen 1 Start Address Register (MSB)
    { 0x0E,             0x00 },       // Screen 2 Start Address Register (LSB)
    { 0x0F,             0x00 },       // Screen 2 Start Address Register (MSB)
    { 0x10,             0x00 },       // Screen Start Address Overflow Register
    { 0x11,             0x00 },       // Memory Address Offset Register
    { 0x12,             0xFF },       // Screen 1 Vertical Size Register (LSB)
    { 0x13,             0x03 },       // Screen 1 Vertical Size Register (MSB)
    { 0x18,             0x00 },       // GPIO Configuration Control Register
    { 0x19,             0x02 },       // GPIO Status/Control Register
    { 0x1A,             0x00 },       // Scratch Pad Register
    { 0x1B,             0x00 },       // SwivelView Mode Register
    { 0x1C,             0xA0 },       // Line Byte Count Register
};
#define S1DNUMREGS 25
#define S1DREGOFFSET 0x1FFE0
#define S1DMEMSIZE 76800

void init13705()
{
int i;

       for(i=1;i<S1DNUMREGS;i++)
              write13705((aS1DRegs[i].Index)+S1DREGOFFSET,aS1DRegs[i].Value);
}
```

Software

El resto del software lo escribimos mayormente en C, por comodidad y velocidad de desarrollo. Se trata de simples y comunes rutinas que no incluiremos aquí para no extender el texto, pero que el lector puede obtener del archivo adjunto con el software, o consultar en cualquiera de las otras notas de aplicación, dado que son muy similares.

Una excepción es el volcado de imágenes, para el cual escribimos una función especial en assembler. Dado el tamaño de una imagen ($320 \times 240 = 76800\, bytes$), necesitamos una función rápida que copie la imagen directamente desde xmem, leyendo por words y escribiendo de a dos bytes en el S1D13705. Básicamente, modificamos el código de *xmem2root()* y lo adaptamos a nuestros propósitos. En la CAN-029 encontrarán una rutina más simple, pero que hace uso de una función externa que resuelve el mapeo de dirección absoluta en 20-bits a valores de *XPC* y dirección lógica. Este ejemplo es una muestra de cómo suceden las cosas a nivel assembler, directamente.

```
;IY: A0-A15
;A: A16
;(IX+0): data left
;(IX+1): data right
writew13705::
      ld a',a                       ; save A16
      ld hl,iy                      ; obtiene A15-A0
      ex de,hl                      ; en DE
      call ab2                      ; pone addresses y BHE
      ld hl,PCDR                    ; apunta a port paralelo
      ioi res 2,(HL)                ; baja CS
      ld a,(IX+0)                   ; lee dato
      ioi ld (PADR),a               ; pone dato
      ld a,0x84
      ioi ld (SPCR),a               ; A= salidas
```

```
      ld hl,PBDR                    ; apunta a port paralelo
      ioi res 6,(HL)                ; baja WE0
wwait1:
      ioi ld a,(hl)                 ; lee port paralelo
      rrca                          ; chequea bit 0 (WAIT)
      jr nc,wwait1                  ; espera hasta que NOWAIT
      ioi set 6,(HL)                ; sube WE0 ( >1/BCLK después de NOWAIT)
      ld hl,PCDR                    ; apunta a port paralelo
      ioi set 2,(HL)                ; sube CS
      ex af,af'                     ; restore A16
      ld hl,iy                      ; lee address
      ld de,0x0001                  ; next byte
      add hl,de                     ; inc address
      adc a,d                       ; (adc a,0x00) propaga carry
      ex de,hl                      ; A15-A0 en DE
      ld a',a                       ; save A16
      call ab2                      ; address bus
      ld hl,PCDR                    ; apunta a port paralelo
      ioi res 2,(HL)                ; baja CS
      ld a,(IX+1)                   ; lee dato
      ioi ld (PADR),a               ; pone dato
      ld a,0x84
      ioi ld (SPCR),a               ; A= salidas
      ld hl,PBDR                    ; apunta a port paralelo
      ioi res 6,(HL)                ; baja WE0
wwait2:
      ioi ld a,(hl)                 ; lee port paralelo
      rrca                          ; chequea bit 0 (WAIT)
      jr nc,wwait2                  ; espera hasta que NOWAIT
      ioi set 6,(HL)                ; sube WE0 ( >1/BCLK después de NOWAIT)
      ld hl,PCDR                    ; apunta a port paralelo
      ioi set 2,(HL)                ; sube CS
      ld a,0x80
      ioi ld (SPCR),a               ; A= entradas
      ex af,af'                     ; restore A16
      ret

xdumpblk::
               ld     a,xpc                ; save the frame pointer, and xpc
               push   af
               push   ix

               ld     de,0xe000            ; de=lsw(dest)-0xe000
               ld     hl,(sp+10)
               or     a
               sbc    hl,de
               ex     de,hl
               ld     hl,(sp+11)           ; hl:e=(dest-0xe000)>>8
               jr     nc,.xcbf_1
               dec    h
.xcbf_1:
               ld     l,d

               add    hl,hl                ; h=(dest-0xe000)>>12
               add    hl,hl
               add    hl,hl
               add    hl,hl
               ld     a,h
               ld     xpc,a                ; xpc=(dest-0xe000)>>12

               ex     de,hl                ; calculate the offset in xpc window
               ld     a,h                  ;    ix=(dest&0x0fff)|0xe000
               and    0x0f
               or     0xe0
               ld     h,a
               ld     ix,hl

               ld     iy,(sp+6)            ; a:iy=dest, hl=length
               ld     hl,(sp+8)
               ld     a,l
               ld     hl,(sp+14)

               ld     c,h                  ; c:b=length
               ld     b,l
```

```
                ex      af,af'
                ld      a,l
                or      a
                jr      nz,.xcbf_2
                dec     c
.xcbf_2:
                ex      af,af'

xcbf_loop:
                call    writew13705             ; preserve BC,IX,IY,A
                ld      de,0x0002               ; 6 copying word by word
                add     ix,de                   ; 4
                add     iy,de                   ; 4
                adc     a,d                     ; 2, (adc a,0x00) propagate carry
                djnz    xcbf_loop               ; 5, critical path word
                dec     c
                jp      p,xcbf_loop

xcbf_done:
                pop     ix
                pop     af
                ld      xpc,a
                ret
#endasm

root useix void LCD_dump(long imgdata, triplet *paldata)
{

long dest;
unsigned int tocopy,len;


        len=S1DMEMSIZE/2;
        dest=0;
        imgdata+=sizeof(long);
        writepalette(paldata);
        while(len) {
                tocopy=2048-(int)(dest&2047);
                if(tocopy>len)
                        tocopy=len;
                xdumpblk(dest,imgdata,tocopy);
                dest+=(tocopy<<1);
                imgdata+=(tocopy<<1);
                len-=tocopy;
        }
}
```

Como punto importante, tengamos en cuenta al inicializar el módulo de setear correctamente los pines bidireccionales en el sentido en que los usamos, y todos en el estado inactivo. Inmediatamente después, inicializamos el chip y cargamos la paleta que vayamos a usar:

```
        WrPortI ( SPCR, &SPCRShadow, 0x80 );                // PA0-7 = Inputs
        WrPortI ( PEDDR,&PEDDRShadow,'\B11111111' );
        WrPortI ( PDDDR,&PDDDRShadow,'\B11111111' );
        WrPortI ( PBDR,&PBDRShadow,'\B11000000' );
        WrPortI ( PCDR,&PCDRShadow,'\B11111111' );
        init13705();
        writepalette(VGApalette);
```

Nota importante

Existe un inconveniente en el arranque, que es el hecho de que el Rabbit puede poner en estado activo varias de las señales de control, generando contenciones en el bus del S1D13705 y causando excesivo consumo de corriente. Esta situación se prolonga hasta tanto el Rabbit es correctamente inicializado y dependiendo del tipo de fuente de alimentación utilizada puede dificultar el arranque o tal vez causar daño a alguno de los chips. Se recomienda, en un entorno de producción, limitar estas condiciones. Una posibilidad es impedir que la señal $\overline{CS}$ se active hasta tanto los pines de I/O han sido correctamente inicializados, por ejemplo, mediante un buffer adicional con una constante RC en el pin de habilitación.