Cika Electrónica

Nota de Aplicación: CAN-008
Título: **Display de textos en LCD gráficos (HD61202) con Rabbit 2000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	29/7/03	

Complementamos el desarrollo de la CAN-003 ampliando la forma de mostrar textos en módulos LCD gráficos inteligentes Powertip PG12864, con Rabbit 2000. Se recomienda al lector el estudio de la CAN-003 para mayor información sobre la estructura de memoria de los displays basados en HD61202 y/o KS0108 y su utilización para aplicaciones puramente gráficas.

Hardware

El hardware de conexión tiene una pequeña diferencia respecto de la CAN-003, dado que preferimos mantener el esquema de la CAN-005.

Para la interfaz con el micro no es necesario ningún tipo de glue-logic, hacemos una conexión directa entre los ports del Rabbit y el LCD, al igual que con la gran mayoría de los microcontroladores, como puede apreciarse en la tabla a la derecha:

Rabbit		LCD
PA.0	----------	D0
PA.1	----------	D1
PA.2	----------	D2
PA.3	----------	D3
PA.4	----------	D4
PA.5	----------	D5
PA.6	----------	D6
PA.7	----------	D7
PE.4	----------	I/D
PE.3	----------	R/W
PE.0	----------	E
PE.1	----------	CS1
PE.7	----------	CS2

El port A, hace las veces de bus de datos, mientras que los ports libres del port E generarán, por software, las señales de control.

Software

Recordemos que la estructura de memoria de estos módulos gráficos se halla agrupada en bytes en sentido vertical, divididos a su vez en páginas. Debido al hecho de que 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.

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. Para el desarrollo de 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", proveyendo 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 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

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.

Desarrollo

Elegimos, para el desarrollo de esta nota, realizar las funciones de lectura y escritura en assembler. Si bien no es necesario para el manejo de textos simples, resulta de gran ayuda cuando se muestran ventanas con sombra, recuadros, o gran cantidad de detalles.

```
/* LCD control signals */
#define LCD_CS1        1
#define LCD_CS2        7
#define LCD_E          0
#define LCD_ID         4
#define LCD_RW         3

/* LCD status bits */
#define BUSY   7


/* Low level functions */

#asm

;no recibe parámetros
;retorna un resultado (HL)
;
LCD_Status::
       ld hl,PEDRShadow             ; apunta a control port (shadow)
       ld de,PEDR                   ; apunta a control port
       res LCD_ID,(HL)              ; Baja I/D
       ioi ldd                      ; ahora
l1:
       call LCD_Read                ; Lee status
       bit 7,l                      ; Chequea busy
       jr nz,l1                     ; loop si busy=1
       ret

;no recibe parámetros
;retorna un resultado (HL)
;
LCD_ReadData::
       ld hl,PEDRShadow             ; apunta a control port (shadow)
       ld de,PEDR                   ; apunta a control port
       set LCD_ID,(HL)              ; Sube I/D
       ioi ldd                      ; ahora

;no recibe parámetros
;retorna un resultado (HL)
;
LCD_Read:
       ld a,0x80                    ; PA0-7 = Inputs, puede leer
       ioi ld (SPCR),a              ; ahora
       ld hl,PEDRShadow             ; apunta a control port (shadow)
       ld de,PEDR                   ; apunta a control port
       set LCD_RW,(HL)              ; Sube RW
       ioi ldd                      ; ahora
       ld hl,PEDRShadow             ; apunta a control port (shadow)
       ld de,PEDR                   ; apunta a control port
       set LCD_E,(HL)               ; Sube E
       ioi ldd                      ; ahora
       ioi ld a,(PADR)              ; Lee status
       ld hl,PEDRShadow             ; apunta a control port (shadow)
       ld de,PEDR                   ; apunta a control port
       res LCD_E,(HL)               ; Baja E
       ioi ldd                      ; ahora
```

```
        ld l,a                      ; retorna resultado
        ld h,0
        ret


;un parámetro:
;@sp+2= dato a escribir
;
LCD_WriteData::
        call LCD_Status             ; Chequea busy (vuelve con I/D bajo)
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        set LCD_ID,(HL)             ; Sube I/D
        ioi ldd                     ; ahora
        jr LCD_Write

;un parámetro:
;@sp+2= dato a escribir
;
LCD_WriteCmd::
        call LCD_Status             ; Chequea busy (vuelve con I/D bajo)
LCD_Write:
        ld a,0x84                   ; PA0-7 = Outputs, puede escribir
        ioi ld (SPCR),a             ; ahora
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        res LCD_RW,(HL)             ; Baja RW
        ioi ldd                     ; ahora
        ld hl,(sp+2)                ; obtiene valor a escribir (LSB)
        ld a,l
        ioi ld (PADR),a             ; escribe
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        set LCD_E,(HL)              ; Sube E
        ioi ldd                     ; ahora
        ld hl,PEDRShadow            ; apunta a control port (shadow)
        ld de,PEDR                  ; apunta a control port
        res LCD_E,(HL)              ; Baja E
        ioi ldd                     ; ahora
        ret

#endasm



void LCD_SelSide(int side)
{
        if(side)       {
                BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_CS2 );        // Sube CS2
                BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_CS1 );        // Baja CS1
        }
        else    {
                BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_CS1 );        // Sube CS1
                BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_CS2 );        // Baja CS2
        }
}

void LCD_CalcPage(int y,int *page,int *row)
{
        *page=(y>>3);               // page = y/8, parte entera
        *row=y-((*page)<<3);        // row = resto de la división anterior
}
```

Algunas funciones de soporte, para generar demoras

```
void MsDelay ( int iDelay )
{
	unsigned long ul0;
	ul0 = MS_TIMER;		// compara con valor actual del timer
	while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}
```

Funciones de inicialización y de soporte de alto nivel.

La inicialización del módulo LCD se realiza por hardware, mediante el pin RST. El estado de reset puede comprobarse mediante los bits de status; no obstante, por simpleza, asumimos que el controlador se ha reseteado exitosamente.

```
void LCD_init ()
{
	WrPortI ( PEDR,&PEDRShadow,0x00 );		// Outputs=0
	WrPortI ( PEDDR,&PEDDRShadow,'\B10110011' ); // PE0,1,4,5,7 = output
	WrPortI ( PEFR, &PEFRShadow, 0 );		// PE: no I/O strobe
	MsDelay ( 1000 );				// espera reset de LCD
	LCD_SelSide(1);					// Controlador 1
	LCD_WriteCmd ( '\B00111111' );			// Display on
	LCD_SelSide(0);					// Controlador 2
	LCD_WriteCmd ( '\B00111111' );			// Display on
}

/* High level functions */

void LCD_home ( void )
{
	LCD_SelSide(1);					// lado derecho
	LCD_WriteCmd ('\B01000000');			// address = 0
	LCD_WriteCmd ('\B11000000');			// display empieza en address 0
	LCD_SelSide(0);					// lado izquierdo
	LCD_WriteCmd ('\B01000000');			// address = 0
	LCD_WriteCmd ('\B11000000');			// display empieza en address 0
}

void LCD_fill(unsigned char pattern)
{
int i,page;
	LCD_home();					// address 0
	for(page=0;page<8;page++) {
		LCD_SelSide(0);				// Controlador izquierdo
		LCD_WriteCmd (0xB8+page);		// Número de página vertical
		for(i=0;i<64;i++)
			LCD_WriteData(pattern);		// Llena página con datos
		LCD_SelSide(1);				// Controlador derecho
		LCD_WriteCmd (0xB8+page);		// Número de página vertical
		for(i=0;i<64;i++)
			LCD_WriteData(pattern);		// Llena página con datos
	}
}

#define LCD_clear() LCD_fill(0)

void LCD_plot (int x, int y)
{
int page,row,data;
	LCD_SelSide(x>63);				// 0-63 => side=0; 63-127 => side=1
	x=(x>63)?(x-64):x;				// corrige x si >63
	LCD_WriteCmd (0x40+x);				// address = x ó x-64
	LCD_CalcPage(y,&page,&row);			// calcula página y fila en página
	LCD_WriteCmd (0xB8+page);			// setea número de página
```

```
        LCD_ReadData();                             // luego de setear dirección hay que
                                                    //  hacer una lectura ficticia
        data=LCD_ReadData();                        // lee datos en esa dirección
        data|=(1<<row);                             // setea el punto solicitado (bit)
        LCD_WriteCmd (0x40+x);                      // vuelve a setear la dirección
                                                    // (leer incrementa el puntero)
        LCD_WriteData(data);
}

void LCD_dump (unsigned char *imgdata)
{
int i,page;

        LCD_home();                                         // address 0,0
        for(page=0;page<8;page++) {
                LCD_SelSide(0);                             // izquierdo
                LCD_WriteCmd (0xB8+page);                   // página 0
                for(i=0;i<64;i++)
                        LCD_WriteData(*(imgdata++));        // lado izquierdo por página
                LCD_SelSide(1);                             // derecho
                LCD_WriteCmd (0xB8+page);                   // página 0
                for(i=0;i<64;i++)
                        LCD_WriteData(*(imgdata++));        // lado derecho por página
        }
}
```

Agregamos ahora algunas funciones para graficar líneas y rectangulos, que utilizaremos para generar recuadros y subrayados. Los algoritmos distan de ser óptimos y se centran en la claridad.

```
// Bresenham Line Algorithm

void LCD_line(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2)
{
int x,y,dx,dy,incx,incy,balance;

        if (x2 >= x1)
        {
                dx = x2 - x1;
                incx = 1;
        }
        else
        {
                dx = x1 - x2;
                incx = -1;
        }

        if (y2 >= y1)
        {
                dy = y2 - y1;
                incy = 1;
        }
        else
        {
                dy = y1 - y2;
                incy = -1;
        }

        x = x1;
        y = y1;

        if (dx >= dy)
        {
                dy <<= 1;
                balance = dy - dx;
                dx <<= 1;
```

```
                while (x != x2)
                {
                        LCD_plot(x, y);
                        if (balance >= 0)
                        {
                                y += incy;
                                balance -= dx;
                        }
                        balance += dy;
                        x += incx;
                } LCD_plot(x, y);
        }
        else
        {
                dx <<= 1;
                balance = dx - dy;
                dy <<= 1;

                while (y != y2)
                {
                        LCD_plot(x, y);
                        if (balance >= 0)
                        {
                                x += incx;
                                balance -= dy;
                        }
                        balance += dx;
                        y += incy;
                } LCD_plot(x, y);
        }
}

void LCD_rectangle(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2)
{
       LCD_line(x1,y1,x2,y1);
       LCD_line(x2,y1,x2,y2);
       LCD_line(x2,y2,x1,y2);
       LCD_line(x1,y2,x1,y1);
}
```

Definimos los sets de caracteres.

```
const static unsigned char font5x7[] = {
<eliminada por cuestiones de espacio, 5 bytes por cada caracter, en formato de pantalla>
};
const static unsigned char font8x8[] = {
<eliminada por cuestiones de espacio, 8 bytes por cada caracter, en formato de pantalla>
};
const static unsigned char font8x15[] = {
<eliminada por cuestiones de espacio, 16 bytes por cada caracter, 8 de arriba y 8 de abajo,
en formato de pantalla>
};
```

Desarrollamos las rutinas de impresión de textos. Como agrupamos los bytes en el sentido en que se muestran en pantalla, simplemente debemos tomar byte a byte y mostrarlo en pantalla.

```
void LCD_putchr6 ( char chr )                           // usa 5x7
{
int i;
       for(i=0;i<5;i++)
              LCD_WriteData(font5x7[5*chr+i]);          // caracter
       LCD_WriteData(0);                                // separador
```

```
}

void LCD_putchr ( char chr )                                    // usa 8x8
{
int i;
        for(i=0;i<8;i++)
                LCD_WriteData(font8x8[8*chr+i]);                // caracter
}

void LCD_putchrB ( char chr , int row, int col)                 // usa 8x15
{
int i;

        for(i=0;i<8;i++)
                LCD_WriteData(font8x15[16*chr+i]);              // parte superior (8 bytes)
        LCD_WriteCmd (0xB9+row);                                // renglón siguiente
        LCD_WriteCmd (0x40+col);                                // columna de inicio
        for(;i<16;i++)
                LCD_WriteData(font8x15[16*chr+i]);              // parte inferior
}



void LCD_print6at (unsigned int row, unsigned int col, char *ptr)  // 20x8
{
int x;
        do {
                if(col>19){                             // pasa a renglón siguiente si excede
                        col=0;
                        row++;
                }
                x=(col>9)?6*(col-10):6*col;             // corrige x si col>9
                if(col>9)                               // 0-9 => side=0; 10>20 => side=1
                        LCD_SelSide(1);                 // lado derecho
                else {
                        LCD_SelSide(0);                 // lado izquierdo
                        x+=4;                           // 4 pixels corrido a la derecha
                }
                LCD_WriteCmd (0xB8+row);                // fila = página
                LCD_WriteCmd (0x40+x);                  // address = 6*col ó 6*(col-10)
                LCD_putchr (*ptr++);                    // escribe caracter
                col++;                                  // siguiente columna
        }       while (*ptr);                           // toda la cadena
}

void LCD_printat (unsigned int row, unsigned int col, char format, char *ptr)     // 16x8ó4
{
int x;
        do {
                if(col>15){                             // pasa a renglón siguiente si excede
                        col=0;
                        row++;
                        if(format=='B')                 // 8x15 usa dos renglones
                                row++;
                }
                x=(col>7)?8*(col-8):8*col;              // corrige x si col>9
                if(col>7)                               // 0-9 => side 0; 10>20 => side 1
                        LCD_SelSide(1);                 // derecho
                else {
                        LCD_SelSide(0);                 // izquierdo
                }
                LCD_WriteCmd (0xB8+row);                // fila=página
                LCD_WriteCmd (0x40+x);                  // address = 8*col ú 8*(col-10)
                switch(format){
                        case 'B':                       // 8x15
                                LCD_putchrB (*ptr++,row,x);
```

```
                        break;
                default:                        // 8x8
                        LCD_putchr (*ptr++);
                        break;
                }
                col++;
        }       while (*ptr);
}
```

El siguiente programa muestra las distintas opciones de texto que hemos desarrollado, y finalmente despliega una pantalla simulando el funcionamiento de un caudalímetro, utilizando 8x15 para resaltar el valor importante, 8x8 para los demás números, y 5x7 para los textos.

```
/* MAIN PROGRAM */

main()
{
int i;
char numero[6];
float num,prom,vcc,sum;

        LCD_init();

        while(1){
                LCD_clear();
                LCD_print6at(0,0,"Texto en 6x8, 20 cpl");
                LCD_printat(2,0,0,"Texto 8x8, 16cpl");
                LCD_printat(4,0,'B',"Texto 8x16,16cpl");
                MsDelay (5000);
                LCD_clear();
                LCD_print6at(1,0,"El texto en 6x8 es  ideal para mostrar  dialogos cortos");
                LCD_rectangle(0,0,120,40);
                MsDelay (5000);
                LCD_clear();
                LCD_printat(1,0,0,"El texto en 8x8 resalta valores de variables:");
                LCD_print6at(7,0,"Corriente:        mA");
                LCD_printat(7,9,0,"12,3");
                MsDelay (5000);
                LCD_clear();
                LCD_printat(1,0,'B',"El formato 8x16 resulta ideal ennumeros");
                MsDelay (5000);
                LCD_clear();
                LCD_print6at(0,4,"Caudalimetro");
                LCD_line(26,8,98,8);
                LCD_print6at(2,0,"Caudal:        m3/hr");
                LCD_print6at(5,0," Qprom:        m3/hr");
                LCD_print6at(7,0,"  Alim:        V");
                num=110.0;                                      // “caudal”
                vcc=5.0;                                        // “alimentación”
                sum=0;                                          // cálculo del promedio
                for(i=1;i<41;i++) {
                        num+=(rand()-0.5)/10;                   // simula variaciones de Q
                        sum+=num;                               // calcula promedio
                        prom=sum/i;
                        sprintf(numero,"%.1f",num);
                        LCD_printat(2,6,'B',numero);
                        sprintf(numero,"%.1f",prom);
                        LCD_printat(5,6,0,numero);
                        vcc+=(rand()-0.5)/100;                  // simula variaciones de Vcc
                        sprintf(numero,"%.2f",vcc);
                        LCD_printat(7,7,0,numero);
                        MsDelay (500);
                }
        }
```

```
}
```