Cika Electrónica

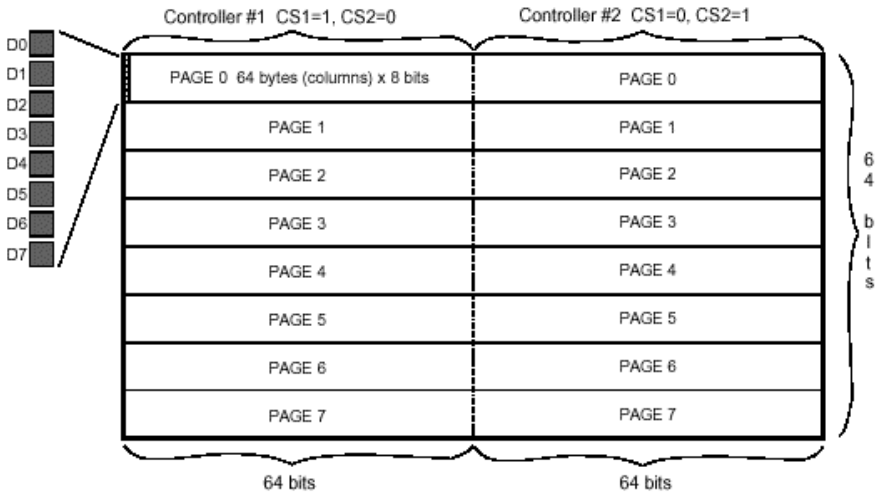
Nota de Aplicación: CAN-022
Título: **Conversión de bitmaps para LCD gráficos con HD61202**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	18/12/03	

Nos interiorizaremos ahora en el desarrollo de un algoritmo para convertir imágenes al formato utilizado por los chips controladores compatibles con el HD61202, de Hitachi, y su clon: el KS0108, de Samsung; presentes en los módulos Powertip PG12864, de 128x64 pixels.
Si bien hemos ya presentado la estructura de memoria de estos displays en la CAN-004, la repetimos aquí por claridad.

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, además, 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:



Para mostrar imágenes, 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 del controlador. Podemos optimizar el despliegue si restringimos las posiciones y altura de las imágenes a las páginas, es decir, las hacemos comenzar en una página y su altura es un múltiplo de ocho. De esta forma, la estructura en memoria de una imagen en este formato será:

- bytes correspondientes a la primera página de la imagen
- [bytes correspondientes a la segunda página de la imagen]...
- bytes correspondientes a la última página de la imagen

Si bien sería más eficiente mandar primero la información a un controlador y luego al otro, resulta menos complicado convertir las imágenes de un formato común a este formato, permitiendo además ubicar la imagen en cualquier posición en que quepa sin necesidad de cambiar su formato..

Debemos entonces desarrollar un algoritmo que nos permita tomar una imagen en algún formato conocido y convertirla a esta estructura. Partimos del formato XBM[1], el cual tiene una correspondencia en cuanto al orden de los bytes en el archivo y en la pantalla (de arriba a abajo y de izquierda a derecha) y el orden de los bits en cada byte (bit menos significativo a la izquierda). No obstante, este formato presenta una estructura lineal, con 8 pixels horizontales por byte.

Para convertir este formato al que necesitamos, deberemos tomar el equivalente a una página, es decir, ocho líneas de imagen, y rotarla 90º en sentido de las agujas del reloj, de modo que el bit menos significativo del primer byte de la primera línea no se mueva, el bit menos significativo del primer byte de la segunda línea corresponda al bit 1 del primer byte de la página, y así sucesivamente hasta completar el primer byte de la página. Luego, el bit menos significativo del segundo byte de la primera línea corresponderá al bit menos significativo del segundo byte de la página, y así sucesivamente hasta completar la página. Repetiremos esta tarea tantas veces como páginas tenga la imagen.

Un detalle a tener en cuenta, es que en nuestro formato, la imagen no debe necesariamente tener un ancho que sea múltiplo de ocho. Sin embargo, en el formato XBM, el archivo tendrá un número entero de bytes, y el ancho efectivo de la imagen será un múltiplo de ocho. Afortunadamente, el formato nos indica el ancho real de la imagen, de forma de procesar solamente los bits necesarios. En cuanto a la altura, este algoritmo necesita que la imagen tenga una altura de un número entero de páginas, es decir, un número de pixels que sea múltiplo de ocho.

Otra ventaja de utilizar el formato XBM, es que éste es un array en C, con lo cual resulta muy fácil incluir la imagen en un programa que haga la conversión:

```
#define imagen_width 35
#define imagen_height 32
static unsigned char imagen_bits[] = {
   0x00, 0xe0, 0x0f, 0x00, 0x00, 0x00, 0x1c, 0x60, 0x00, 0x00, 0x00, 0x03,
   0x00, 0x01, 0x00, 0x80, 0x01, 0x00, 0x02, 0x00, 0x40, 0x00, 0x00, 0x00,
   0x00, 0x20, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x18,
   0x00, 0x00, 0x20, 0x00, 0x08, 0x30, 0x30, 0x00, 0x00, 0x04, 0x78, 0x78,
   0x00, 0x00, 0x04, 0x78, 0x78, 0x00, 0x00, 0x04, 0x78, 0x78, 0x00, 0x00,
   0x02, 0x78, 0x78, 0x00, 0x00, 0x02, 0x78, 0x78, 0x00, 0x00, 0x02, 0x30,
   0x30, 0x00, 0x00, 0x02, 0x02, 0x00, 0x01, 0x00, 0x02, 0x02, 0x00, 0x01,
   0x00, 0x02, 0x01, 0x00, 0x02, 0x00, 0xe2, 0x01, 0x00, 0x1e, 0x00, 0x02,
   0x02, 0x00, 0x01, 0x00, 0x04, 0x04, 0x80, 0x00, 0x00, 0x04, 0x0c, 0xc0,
   0x00, 0x00, 0x04, 0x38, 0x70, 0x00, 0x00, 0x08, 0xf0, 0x3f, 0x00, 0x00,
   0x18, 0xe0, 0x1f, 0x20, 0x00, 0x10, 0xc0, 0x0f, 0x00, 0x00, 0x20, 0x80,
   0x07, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x80, 0x01, 0x00, 0x02,
   0x00, 0x00, 0x03, 0x00, 0x01, 0x00, 0x00, 0x1c, 0x60, 0x00, 0x00, 0x00,
   0xe0, 0x0f, 0x00, 0x00 };
```

A continuación desarrollamos código en C para realizar la función correspondiente al algoritmo descripto, el cual ha sido compactado para relizar una sola operación dentro de un bloque iterativo, resolviendo el problema de forma rápida y simple, obteniendo la salida por standard output, de donde se puede incluír en nuestro código mediante copy/paste o redireccionando la salida a un archivo. En nuestro caso hemos utilizado como compilador *gcc* bajo Linux, sin embargo el código debería compilar sin mayores inconvenientes bajo cualquier otro compilador y plataforma, dado que no se han utilizado funciones no standard.

```
int process(unsigned char *image, int width, int height){
  int i,j,k,p,pwidth,page,pages,count;
  unsigned char *pagebuf;

    if((8*(pwidth=(width>>3)))!=width)
      pwidth+=1;

    if((8*(pages=(height>>3)))!=height)
      return(0);

    page=8*pwidth;

    printf("const static unsigned char imagen[]={\n\t");
```

1 El formato XBM es utilizado en ambiente X11, un ejemplo de software gratis que lo utiliza es *Gimp* (GNU Image Manipulation Program).

```
    for(p=0;p<pages;p++){
      pagebuf=(unsigned char *) calloc(page,1);
      for(i=0;i<8;i++)
        for(j=0;j<pwidth;j++)
          for(k=0;k<8;k++)
            pagebuf[j*8+k]|=(((image[j+i*pwidth+page*p]&(1<<k))>>k)<<i);
      count=0;
      for(i=0;i<width;i++){
        switch(count++){
        case 16:
          printf(",\n\t");
          count=1;
          break;
        default:
          printf(",");
        case 0:
          break;
        }
        printf("0x%03X",pagebuf[i]);
      }
      free(pagebuf);
    }
    puts("\n};");
    return(1);
}
```

Esta función, puede ser llamada desde un programa que tome los datos de la imagen incluida:

```
#include "imagen.xbm"

  if(!process(imagen_bits,imagen_width,imagen_height)){
    puts("imagen: altura incorrecta (múltiplo de 8)");
    exit(1);
  }
```

Este método es suficientemente rápido y práctico. Simplemente debemos llamar “imagen.xbm” al archivo al salvarlo en *Gimp*, aunque debemos recompilar el programa cada vez, o incluír todas las imágenes a convertir en el cuerpo del programa. Si esto resulta un inconveniente, el usuario puede desarrollar una interfaz para leer el formato directamente del disco o algún otro formato como BMP. No obstante, esto requiere de más trabajo, tanto antes de salvar la imagen como al procesar el archivo en disco. La intención de esta nota es mostrar un método rápido y simple para resolver la generación de bitmaps para los displays basados en HD61202.