Cika Electrónica

Nota de Aplicación: CAN-038
Título: **Generación de bitmaps color para controladores S1D13706**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	29/07/05	

Si bien las CAN-035 y CAN-036 describen el procedimiento para procesar imágenes para enviarlas fácilmente al controlador S1D13706, les presentamos un pequeño y sencillo programa de tipo Q&D[1] para resolver la tarea en 8bpp, fácilmente modificable para 4bpp.

Software

Para mostrar pantallas, deberemos agrupar los datos de modo tal de poder enviarlos de forma que aproveche de manera eficiente la estructura de memoria. Nada quita que guardemos el BMP en memoria y hagamos un programa para nuestro querido procesador que lo decodifique y envíe los datos al display, pero resulta que por lo general nuestro sistema tiene otras cosas más importantes que hacer, y resulta preferible minimizar el tiempo perdido y los preciosos bytes 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.

Desarrollo

El algoritmo que proponemos es simple, antes que eficiente, y lee un archivo en formato BMP, guardando dos archivos, uno conteniendo la paleta en formato BGR y otro con la información a mandar al display. Recordemos que el BMP debe ser en 256 bits, una utilidad sin costo para convertir estos archivos es *Gimp* (GNU Image Manipulation Program). En programas como éste, que carece de la opción de espejado vertical (vertical flip), deberemos realizar dos operaciones (luego de reducir la imagen a 256 colores): rotar 180º y espejar horizontalmente (horizontal flip).

```
typedef union {
  unsigned char array[4];
  long number;
} value;

value colors,width,height;

        fread(buffer,1,14+40,filein);               // saltea header, lee info
        memcpy(colors.array,&buffer[54-8],4);       // cantidad de colores
        memcpy(width.array,&buffer[14+4],4);        // ancho de imagen
        memcpy(height.array,&buffer[14+8],4);       // altura de imagen
        padding=width.number&3;                     // en BMP son múltiplos de 4 (padding)
        fread(buffer,1,colors.number*4,filein);     // lee paleta
        for(j=1;j<colors.number;j++){
              memmove(buffer+3*j,buffer+4*j,3);     // pack BGR0 -> BGR
        }
        while(j<256) {                              // pone a 0 (negro) colores no usados
                buffer[3*j]=0;
                buffer[3*j+1]=0;
                buffer[3*(j++)+2]=0;
        }
```

1 Quick & Dirty

```
        fwrite(buffer,1,3*256,fileout2);             // escribe paleta
        for(i=0;i<height.number;i++){
                fread(buffer,1,width.number+padding,filein);     // lee línea de pixels
        fwrite(buffer,1,width.number,fileout1);      // escribe, elimina padding (si lo hay)
```

Realizamos el procesamiento por líneas, para de alguna forma estructurar la operación y mejorar el acceso a archivos, simplemente.

El programa terminado, es el siguiente:

```
#include <stdio.h>
#include <string.h>

typedef union {
  unsigned char array[4];
  long number;
} value;

int main (int argc, char *argv[])
{

FILE *filein,*fileout1,*fileout2;
unsigned char *buffer,outimage[25],outpalette[25],r,g,b,rr,gg1,gg2,bb;
int i,j,padding;
value colors,width,height;

    if(argc!=3){
      fprintf(stderr,"usage: %s filein fileout\n",argv[0]);
      exit (-1);
    }

    if (!(filein=fopen(argv[1],"r"))) {
      fprintf(stderr,"%s: can not open %s\n",argv[0],argv[1]);
      exit (-1);
    }

    strncpy(outimage,argv[2],20);
    outimage[20]=0;
    strcat(outimage,".pbm");
    if (!(fileout1=fopen(outimage,"w"))) {
      fprintf(stderr,"%s: can not open %s\n",argv[0],outimage);
      exit (-1);
    }
    strncpy(outpalette,argv[2],20);
    outpalette[20]=0;
    strcat(outpalette,".pal");
    if (!(fileout2=fopen(outpalette,"w"))) {
      fprintf(stderr,"%s: can not open %s\n",argv[0],outpalette);
      exit (-1);
    }
    if(!(buffer=(char *)calloc(1024,1))){
      fprintf(stderr,"%s: not enough memory\n",argv[0]);
      exit (-1);
    }

        fread(buffer,1,14+40,filein);
        memcpy(colors.array,&buffer[54-8],4);
        if(colors.number>256){
                fprintf(stderr,"%s: too many colors %ld\n",argv[0],colors.number);
                exit(-1);
        }
        memcpy(width.array,&buffer[14+4],4);
        memcpy(height.array,&buffer[14+8],4);
        if(width.number>320 || height.number>320 ){
              fprintf(stderr,"%s: too big %ldx%ld\n",argv[0],width.number,height.number);
              exit(-1);
        }
        printf("%ld colors %ldx%ld\n",colors.number,width.number,height.number);
        padding=width.number&3;
        fread(buffer,1,colors.number*4,filein);
        for(j=1;j<colors.number;j++){
            memmove(buffer+3*j,buffer+4*j,3);
        }
```

```
        while(j<256) {
          buffer[3*j]=0;
          buffer[3*j+1]=0;
          buffer[3*(j++)+2]=0;
        }
        fwrite(buffer,1,3*256,fileout2);
        for(i=0;i<height.number;i++){
                fread(buffer,1,width.number+padding,filein);
                fwrite(buffer,1,width.number,fileout1);
        }
        free(buffer);
        fclose(filein);
        fclose(fileout1);
        fclose(fileout2);
}
```

Si bien hemos desarrollado esto sobre Linux y compilado con *gcc*, no debería haber inconvenientes en portarlo a cualquier otro sistema