

零死角玩转STM32

与野火同行 乐意惬意无边



原创教程，完全开源。



由浅入深，结合实操。



通俗易懂，详尽解读。



配套板子，全面玩转。



强强联合，不断更新。



野火团队 Wild Fire Team



0、友情提示

《零死角玩转 STM32》系列教程由初级篇、中级篇、高级篇、系统篇、四个部分组成，根据野火 STM32 开发板旧版教程升级而来，且经过重新深入编写，重新排版，更适合初学者，步步为营，从入门到精通，从裸奔到系统，让您零死角玩转 STM32。M3 的世界，与野火同行，乐意惬意无边。

另外，野火团队历时一年精心打造的《STM32 库开发实战指南》将于今年 10 月份由机械工业出版社出版，该书的排版更适于纸质书本阅读以及更有利于查阅资料。内容上会给你带来更多的惊喜。是一本学习 STM32 必备的工具书。敬请期待！



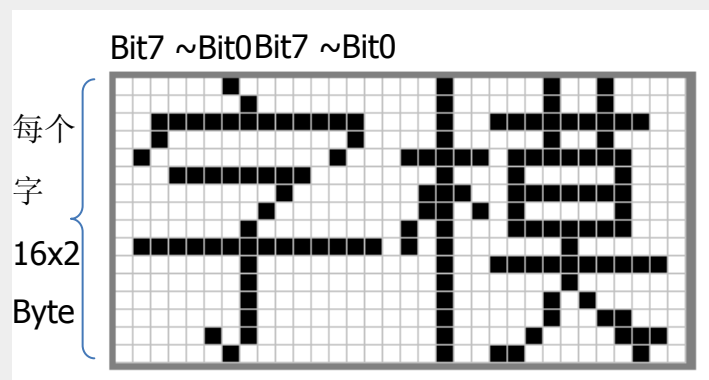
5、液晶显示（中、英、Pic）

5.1 实验简介

在《液晶触摸画板》中，我们已经成功地实现了驱动 LCD 和触摸屏，并制作了触摸画板小应用，但是若要显示文字或图片文件，则还需要利用文件系统，读取保存在 SD 卡中的字库文件、图片文件。

5.2 什么是字模

我们知道其实液晶屏就是一个由像素点组成的点阵，若要显示文字，则需要很多像素点的共同构成。见下错误！未找到引用源。，图中是两个由 16*16 的点阵显示的两个汉字。



如果我们规定：每个汉字都由这样 16*16 的点阵来显示，把笔迹经过的像素点以“1”表示，没有笔迹的点以“0”表示，每个像素点的状态以一个二进制位来记录，用 $16*16/8=32$ 个字节就可以把这个字记录下来。这 32 个字节数据就称为该文字的字模，还有其它常用字模是 24*24、32*32 的。16*16 的“字”的字模数据为：

```
1. /* 字 */
2. unsigned char code Bmp003[]=
3. {
4. /*-----
5. ; 源文件 / 文字 : 字
6. ; 宽*高(像素) : 16*16
7. ; 字模格式/大小 : 单色点阵液晶字模, 横向取模, 字节正序/32 字节
8. -----*/
9.
```

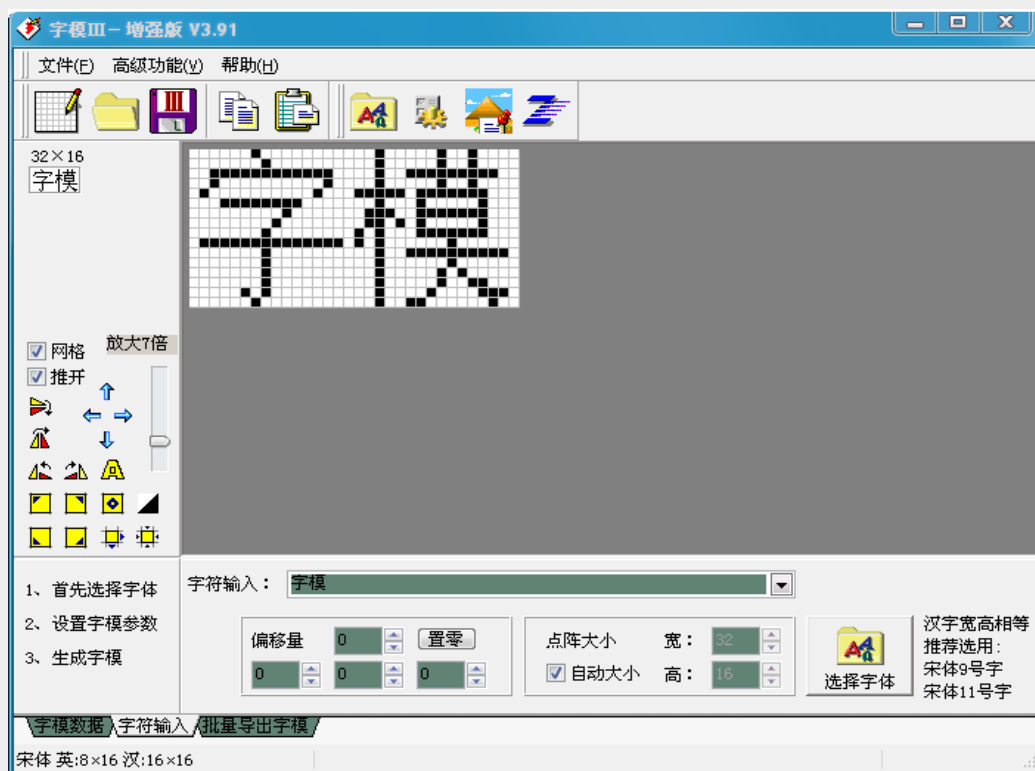
```
10. 0x02, 0x00, 0x01, 0x00, 0x3F, 0xFC, 0x20, 0x04, 0x40, 0x08, 0x1F, 0xE0, 0x00, 0x40,  
    0x00, 0x80,  
11. 0xFF, 0xFF, 0x7F, 0xFE, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00, 0x05, 0x00,  
    0x02, 0x00,  
12. };
```

在这样的字模中，以两个字节表示一行像素点，16行构成一个字模。如果使用 LCD 的画点函数，按位来扫描这些字模数据，把为 1 的位以黑色来显示 (也可以使用其它颜色)，即可把整个点阵还原出来，显示在液晶屏上。

5.3 制作字模

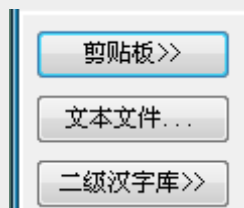
我们采用“字模 III-增强版 v3.91”  软件来制作中文字库，步骤如下：

1. 打开字模软件



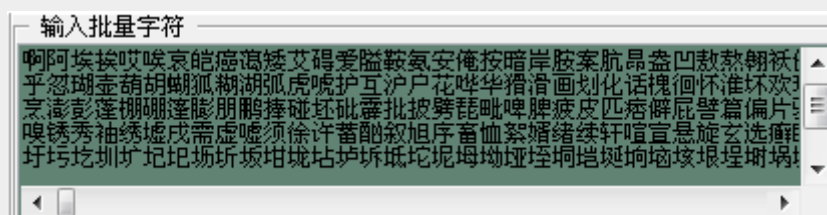
2. 点击“自动批量生成字库”按钮选项 .

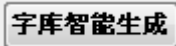
软件界面左下角将出现一下几个按钮选项：



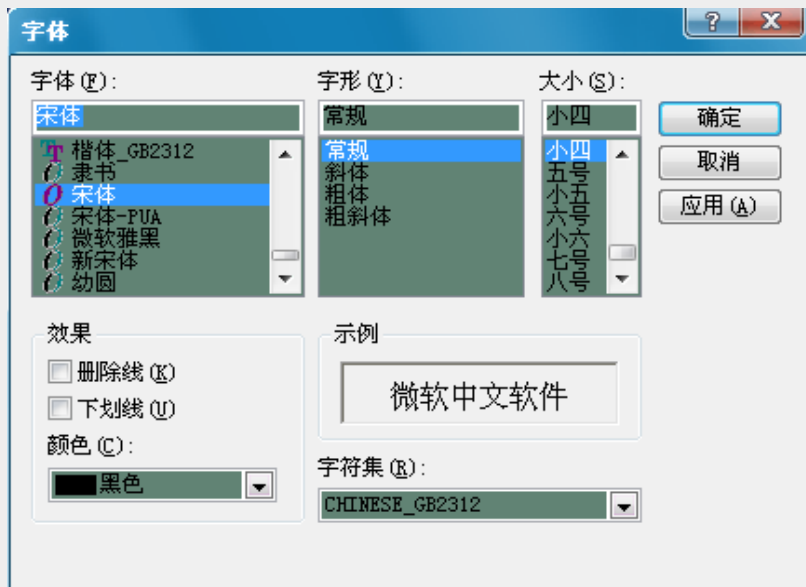
3. 点击选择“二级汉字库”按钮。

在“输入批量字符”框里面将会列出二级汉字的所有汉字，其中共收录了6768个汉字字符，非特殊情况下都能够满足大家的要求，如图：

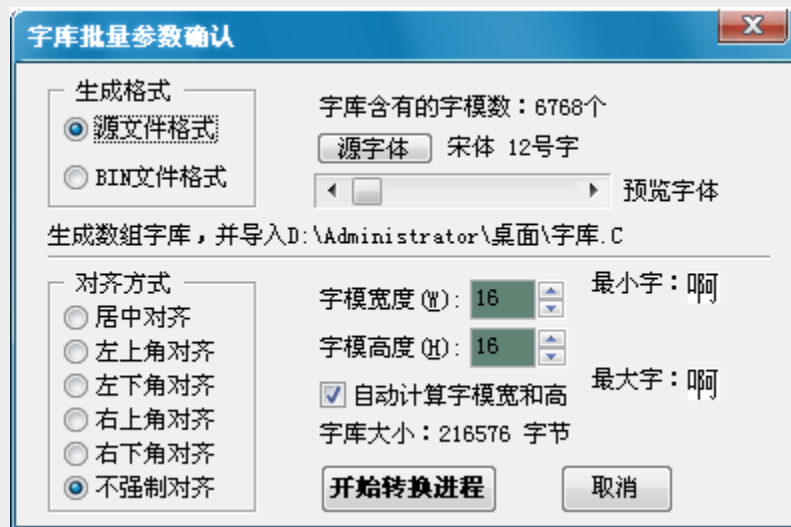


4. 点击“字库智能生成”按钮 ，弹出“字库批量参数确认”对话框。

我们在“源字体”选项里面做如下设置，需要注意的是大小问题，因为我们本次的设计目标是实现16*16的汉字，所以在此选择‘小四’字体。

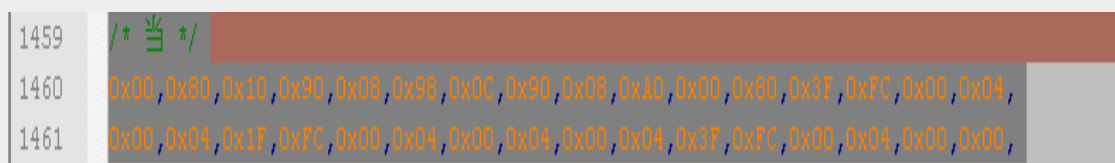


设置好之后如下：



5. 点击“开始转换进程”按钮 **开始转换进程**，就会在安装目录下或者你设置好的目录下生成.c 后缀的字库文件。
6. 对于 LCD 显示来说，只要能够在指定的位置描写制定颜色的点，那么就能够很好地根据汉字字模信息来描写汉字。在此，为了能够更好的清楚字模的取向和高低位的排列顺序，我们可以现先在 pc 测试我们刚才制作好的库文件。

在这里我们取“当”字符的数据来测试。

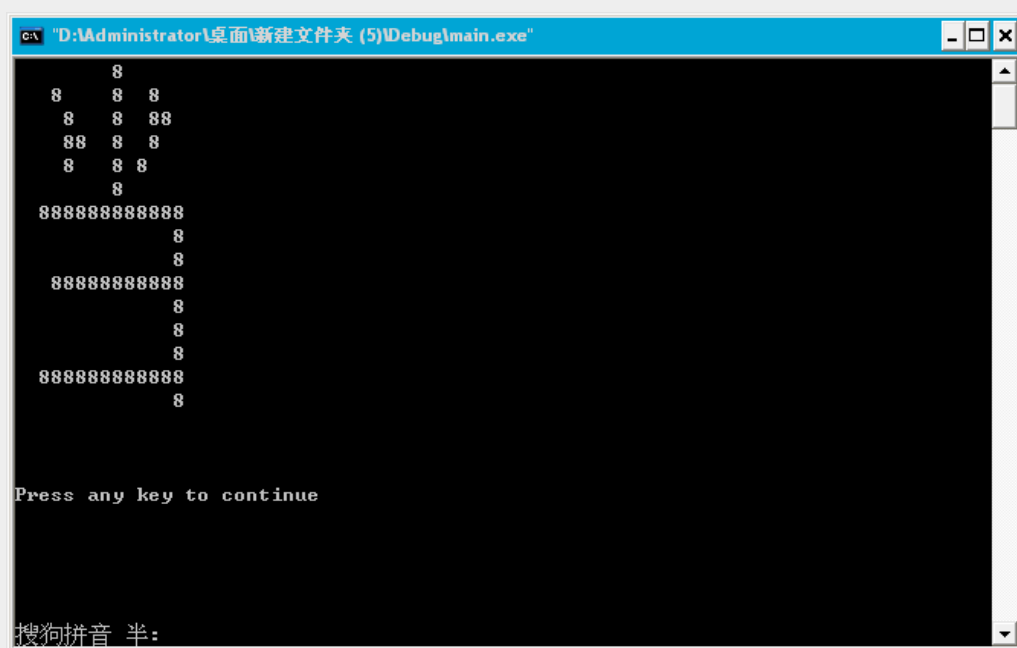


VC6.0 测试源码如下，该代码实现了把字模中为 1 的点都用数字“8”来表示：

```
1. #include <stdio.h>
2.
3. unsigned char cc[] =
4. { /* "当"字符 */
5. 0x00, 0x80, 0x10, 0x90, 0x08, 0x98, 0x0C, 0x90, 0x08, 0xA0, 0x00, 0x80, 0x3F, 0xFC,
6. 0x00, 0x04,
7. 0x00, 0x04, 0x1F, 0xFC, 0x00, 0x04, 0x00, 0x04, 0x00, 0x04, 0x3F, 0xFC, 0x00, 0x04,
8. 0x00, 0x00
9. };
10. void main()
11. {
12.     int i, j;
13.     unsigned char kk;
14.     for ( i=0; i<16; i++)
```

```
14.     {
15.         for(j=0; j<8; j++)
16.         {
17.             kk = cc[2*i] << j ;    //左移 j 位
18.
19.             if( kk & 0x80)    //如果最高位为 1
20.             {
21.                 printf("8");
22.             }
23.             else
24.             {
25.                 printf(" ");
26.             }
27.         }
28.
29.         for(j=0; j<8; j++)
30.         {
31.
32.             kk = cc[2*i+1] << j ;    //左移 j 位
33.
34.             if( kk & 0x80)           //如果最高位为 1
35.             {
36.                 printf("8");
37.             }
38.             else
39.             {
40.                 printf(" ");
41.             }
42.         }
43.
44.         printf("\n");
45.     }
46.     }
47.     printf("\n\n");
48.
49.
50.     }
51.
52.
```

测试结果如下：



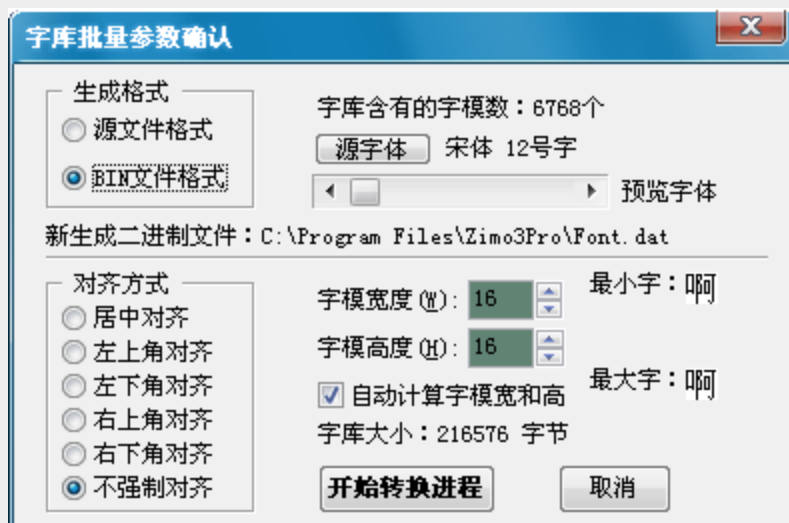
```
      8
8      8  8
  8      8  88
    88      8  8
      8      8  8
        8
88888888888888
          8
            8
      888888888888
          8
            8
              8
88888888888888
          8
```

Press any key to continue

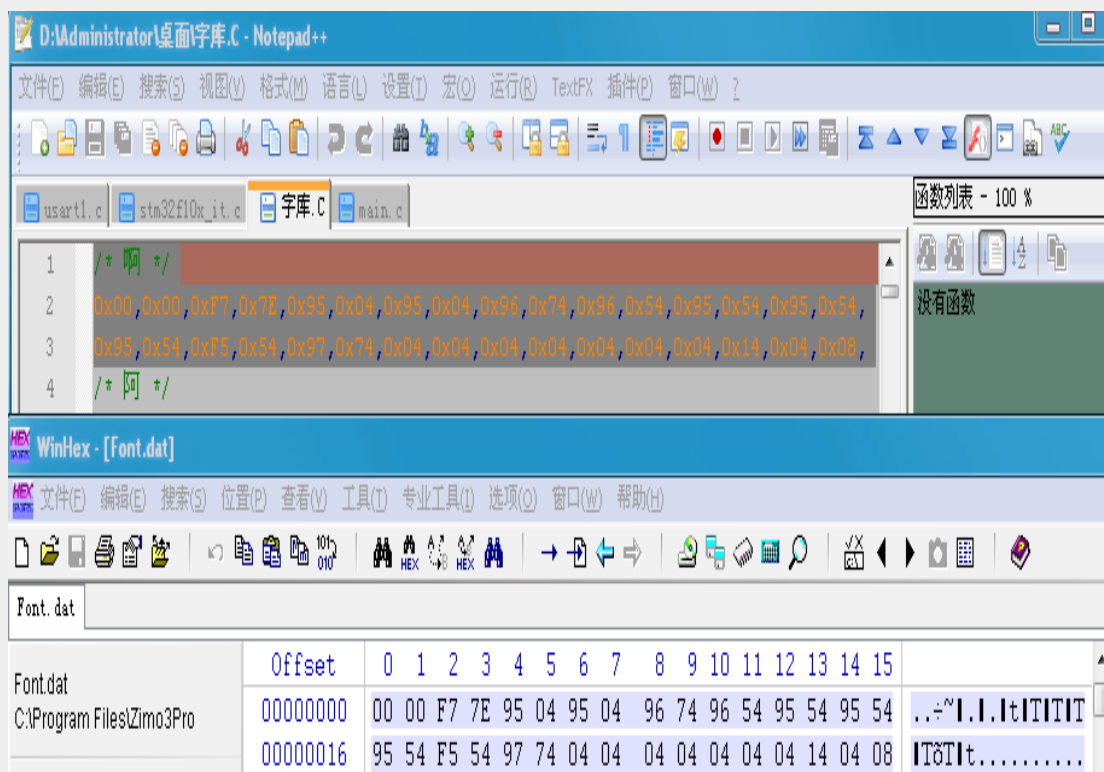
搜狗拼音 半:

看到以上的测试结果，相信大家对汉字的取模方向和高低位的排列顺序有了比较直观的了解。

7. 回到“字模 III-增强版 v3.91”软件，采用与之前同样的方式生成 bin 格式的字库文件(即“生成格式”选项设置为“bin 文件格式”)。



在软件安装目录下会生成 Font.dat 文件,我们用“WinHex”软件查看他的具体内容，与刚才制作的.c 字库的文件内容是一致的，对比如下：



将生成的汉字字库拷贝到 SD 卡根目录下并重命名为“HZLIB.bin”。把该文件保存到 SD 卡中，STM32 芯片通过文件系统读取文件即可获得字库的数据。

5.4. BMP 图片格式

使用 LCD 屏来显示 BMP 图片，就如同播放 MP3 文件一样，首先要了解其文件的格式。

BMP 文件格式，又称为 *位图*（Bitmap）或是 DIB(Device-Independent Device，设备无关位图)，是 Windows 系统中广泛使用的图像文件格式。BMP 文件保存了一幅图像中所有的像素。

BMP 格式可以保存单色位图、16 色或 256 色索引模式像素图、24 位真彩色图象，每种模式中单一像素点的大小分别为 1/8 字节，1/2 字节，1 字节和 32 字节。目前最常见的是 256 位色 BMP 和 24 位色 BMP。

BMP 文件格式还定义了像素保存的几种方法，包括不压缩、RLE 压缩等。

常见的 BMP 文件大多是不压缩的。

Windows 所使用的 BMP 文件，在开始处有一个文件头，大小为 54 字节。保存了包括文件格式标识、颜色数、图像大小、压缩方式等信息，因为我们仅讨论 24 位色不压缩的 BMP，所以文件头中的信息基本不需要注意，只有“大小”这一项对我们比较有用。图像的宽度和高度都是一个 32 位整数，在文件中的地址分别为 0x0012 和 0x0016。54 个字节以后，如果是 16 色或 256 色 BMP，则还有一个颜色表，但在 24 位色 BMP 文件则没有，我们这里不考虑。接下来就是实际的像素数据了。因此总的来说 BMP 图片的优点是简单。

5.4.1 BMP 图片分析

下面来用 WinHex 软件(跟 UltraEdit 软件功能类似)来分析一下 BMP 图像的文件内容：

测试图片为 123.bmp，先用 ACDSee 软件打开，查看图像，其图像内容见图 0-1。



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	42	4D	B6	01	00	00	00	00	00	00	36	00	00	00	28	00
00000016	00	00	0F	00	00	00	08	00	00	00	01	00	18	00	00	00
00000032	00	00	00	00	00	00	C4	0E	00	00	C4	0E	00	00	00	00
00000048	00	00	00	00	00	00	31	31	31	31	31	31	31	31	31	31

图 0-3 文件头部信息

图 0-3，阴影部分是文件头部信息。它可以分为两块：**BMP 文件头**和**位图信息头**。

0~1 字节

BMP 文件的 0 和 1 字节用于表示文件的类型。如果是位图文件类型，必须分别为 0x42 和 0x4D，0x424D='BM'。

3~14 字节

3 到 14 字节的意义可以用一个结构体来描述。如下：

```
1. typedef struct tagBITMAPFILEHEADER
2. {
3.     //attention: sizeof(DWORD)=4   sizeof(WORD)=2
4.     DWORD bfSize;           //文件大小
5.     WORD bfReserved1;      //保留字，不考虑
6.     WORD bfReserved2;      //保留字，同上
7.     DWORD bfOffBits;       //实际位图数据的偏移字节数，即前三个部分长度之和
8. } BITMAPFILEHEADER, tagBITMAPFILEHEADER;
```

14~53 字节

头部信息剩下的部分就是**位图信息头**，14 到 53 字节内容的意义如下：

```
1. typedef struct tagBITMAPINFOHEADER
2. {
3.     //attention: sizeof(DWORD)=4   sizeof(WORD)=2
4.     DWORD biSize;           //指定此结构体的长度，为 40
5.     LONG biWidth;           //位图宽，说明本图的宽度，以像素为单位
```



```
6. LONG biHeight;           //位图高，指明本图的高度，像素为单位
7. WORD biPlanes;           //平面数，为 1
8. WORD biBitCount;         //采用颜色位数，可以是 1，2，4，8，16，24 新的可以是 32
9. DWORD biCompression;     //压缩方式，可以是 0，1，2，其中 0 表示不压缩
10. DWORD biSizeImage;      //实际位图数据占用的字节数
11. LONG biXPelsPerMeter;    //X 方向分辨率
12. LONG biYPelsPerMeter;    //Y 方向分辨率
13. DWORD biClrUsed;         //使用的颜色数，如果为 0，则表示默认值(2^颜色位数)
14. DWORD biClrImportant;    //重要颜色数，如果为 0，则表示所有颜色都是重要的
15.
16. } BITMAPINFOHEADER, tagBITMAPINFOHEADER;
```

由上述分析与 WinHex 软件的分析内容结合得到该图片的信息如下：

文件大小:438

保留字:0

保留字:0

实际位图数据的偏移字节数:54

位图信息头:

结构体的长度:40

位图宽:15

位图高:8

biPlanes 平面数:1

biBitCount 采用颜色位数:24

压缩方式:0

biSizeImage 实际位图数据占用的字节数:0

X 方向分辨率:3780

Y 方向分辨率:3780

使用的颜色数:0

重要颜色数:0



2. 图像像素数据部分(如果是 24 位真彩色,则 54 字节之后就是像素部分):

00000048	00 00 00 00 00 00 31 31 31 31 31 31 31 31 31
00000064	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000080	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000096	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000112	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000128	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000144	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000160	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000176	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000192	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000208	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000224	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000240	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000256	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000272	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000288	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000304	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000320	31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000336	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000352	31 31 31 31 31 31 31 31 31 31 31 31 31 31 30 30
00000368	30 30 30 30 31 31 31 31 31 31 31 31 31 31 31
00000384	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
00000400	31 31 31 31 31 31 31 31 31 31 31 31 31 31 30 30
00000416	30 30 30 30 31 31 31 31 31 31 31 31 31 31 31
00000432	31 31 31 00 00 00

图 0-4 图像像素数据

有些读者可能已经发现, 在像素部分夹杂着一些值为 0 的数据信息, 如下图所示灰色部分所示:

00000096	31 31 31 00 00 00 31 31 31 31 31 31 31 31 31
----------	--

本例子中整张图片都是灰色的, 为什么会有 0 像素的出现呢?

对齐的数据, 更容易被操作系统或者硬件调入 cache, 使得 cache 的命中率提高, 从而提高访问效率。也就是基于性能上得考虑, 所以 Windows 规定一个扫描行所占的字节数必须是 4 的倍数(即以 long 为单位), 不足的以 0 填充。由前面位图信息头的分析可知, 位图宽为 15 个像素点, 由于是 24 位图, 每个像素点由 3 个字节构成。因此, 未补充字节前字节数为 15×3 等于 45 字节, 要到 4 的倍数, 必须向上取 4 的倍数即 48, 所以就有了不上 3 个字节 0 的结果。

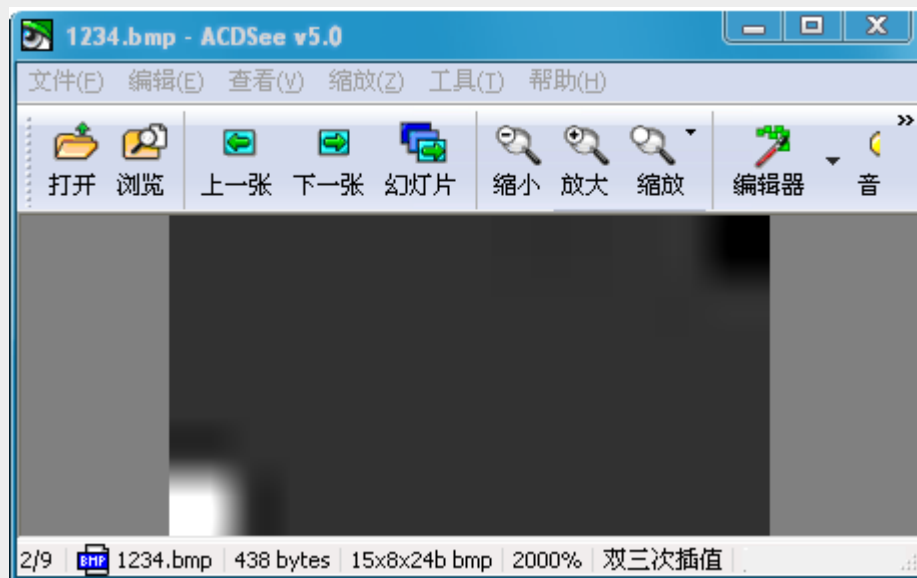
因此, 整个图像文件的大小为: $54 + (15 \times 3 + 3) \times 8$ 等于 438 字节, 和前面所得到的信息文件大小为 438 是一致的。



另外一点需要注意的是显示图像的顺序是 **由下到上, 由左到右**。即像素数据部分的第一像素数据是我们见到的图像的左下角的像素的数据; 而像素数据的最后一个有效数据是我们见到的图像的右上角的像素的数据。

下面我们修改图片来验证一下:

我们将左下角画上白色, 右上角画上黑色, **图片放大之后**如下



图片数据分析如下:

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	42	4D	B6	01	00	00	00	00	00	00	36	00	00	00	28	00
00000016	00	00	0F	00	00	00	08	00	00	00	01	00	18	00	00	00
00000032	00	00	80	01	00	00	C4	0E	00	00	C4	0E	00	00	00	00
00000048	00	00	00	00	00	00	FF	FF	FF	31	31	31	31	31	31	31
00000064	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000080	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000096	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31
00000112	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000128	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000144	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31
00000160	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000176	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000192	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31
00000208	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000224	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000240	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31
00000256	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000272	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000288	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31
00000304	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000320	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00000336	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31
00000352	31	31	31	31	31	31	31	31	31	31	31	31	31	31	30	30
00000368	30	30	30	30	31	31	31	31	31	31	31	31	31	00	00	00
00000384	00	00	00	00	00	00	31	31	31	31	31	31	31	31	31	31
00000400	31	31	31	31	31	31	31	31	31	31	31	31	31	31	30	30
00000416	30	30	30	30	31	31	31	31	31	31	31	31	31	00	00	00
00000432	00	00	00	00	00	00										

我们可以看到像素部分开始部分是白色像素(灰色部分)，对应我们图像的左下角：

```
00000048 | 00 00 00 00 00 00 FF FF FF 31 31 31 31 31 31 31
```

后尾部分是黑色像素（灰色部分）对应我们图像的右上角：

```
00000416 | 30 30 30 30 31 31 31 31 31 31 31 31 00 00 00
00000432 | 00 00 00 00 00 00
```

对 BMP 图像文件内容有了具体的了解之后就可以开始编写我们的应用程序，根据图片的头部信息，合理地读出其中的像素部分，把读出的像素点数据送到 LCD 屏，就可以显示出该图片了。



5.5 显示中英文及 BMP 图片实验

5.5.1 实验描述及工程文件清单

实验描述	使用软件制作自定义类型的字库，然之后将字库放入 SD 卡中，并且在 SD 卡中放入三张 BMP 图片。最后调用截屏函数截取 LCD 背景并保存为 BMP 图片。
硬件连接	本实验的硬件连接包括 MicroSD 卡控制信号、TFT 控制信号线及触摸屏 TSC2046 控制线，这些连接与前面章节的一样，可参照前面的电路图。
用到的库文件	startup/start_stm32f10x_hd.c CMSIS/core_cm3.c CMSIS/system_stm32f10x.c <i>FWlib/misc.c</i> <i>FWlib/stm32f10x_rcc.c</i> <i>FWlib/stm32f10x_systick.c</i> <i>FWlib/stm32f10x_exti.c</i> <i>FWlib/stm32f10x_gpio.c</i> <i>FWlib/stm32f10x_sdio.c</i> <i>FWlib/stm32f10x_dma.c</i> <i>FWlib/stm32f10x_usart.c</i> <i>FWlib/stm32f10x_fsmc.c</i>
用户编写的文件	USER/main.c USER/stm32f10x_it.c <i>USER/systick.c</i> <i>USER/usart1.c</i> <i>USER/lcd.c</i> <i>USER/ff.c</i>

	<i>USER/sdio_sdcard.c</i> <i>USER/lcd_botton.c</i> <i>USER/Sd_bmp.c</i> <i>USER/sd_fs_app.c</i>
文件系统文件	<i>ff9/diskio.c</i> <i>ff9/ff.c</i> <i>ff9/cc936.c</i>

5.5.2 配置工程环境

本实验需要制作字库，其文件名为：*HZLIB.bin*，三个 BMP 图片文件，文件名为：*pic1.bmp*、*pic2.bmp*、*pic3.bmp*，把这四个文件保存到 SD 卡中，再把该 SD 卡插入开发板的 SD 卡接口。

本实验中要把旧文件：*systick.c*、*usart1.c*、*lcd.c*、*ff.c*、*sdio_sdcard.c*、*lcd_botton.c*文件添加进新工程，新建 *Sd_bmp.c*、*sd_fs_app.c*文件，分别用于编写 BMP 文件相关的函数和字模获取函数。

5.5.3 main 文件

从 main 函数看起，它调用了很多函数，用于显示 BMP 图和沿各种方向排列的文字。

```
1. int main(void)
2. {
3.
4.     /* USART1 config */
5.     USART1_Config();
6.     SysTick_Init();
7.     LCD_Init();                               /* LCD 初始化*/
8.     sd_fs_init();
9.
10.    /*显示图像*/
11.    Lcd_show_bmp(0, 0, "/pic3.bmp");
12.    Lcd_show_bmp( 0,0, "/pic2.bmp");
13.    Lcd_show_bmp( 0,0, "/pic1.bmp");
14.
15.    /*横屏显示*/
16.    LCD_Str_O(20, 10, "LCD_DEMO",0);
17.    LCD_Str_CH(20,30, "阿莫论坛野火专区",0,0xffff);
18.    LCD_Str_CH_O(20,50, "阿莫论坛野火专区",0);
```



```
19. LCD_Num_6x12_O(20, 70, 65535, BLACK);
20. LCD_Str_6x12_O(20, 90, "LOVE STM32", BLACK);
21.
22. /*竖屏显示*/
23. LCD_Str_O_P(300, 10, "Runing", 0);
24. LCD_Str_CH_P(280, 10, "阿莫论坛野火专区欢迎你", 0xff, 0xffff);
25. LCD_Str_CH_O_P(260, 10, "阿莫论坛野火专区", 0);
26. LCD_Str_6x12_O_P(240, 10, "LOVE STM32", 0);
27. LCD_Str_ENCH_O_P(220, 10, "欢迎使用野火 stm32 开发板", 0);
28.
29. /*截图*/
30. LCD_Str_CH(20, 150, "正在截图", 0, 0xffff);
31. Screen_shot(0, 0, 240, 320, "/myScreen");
32. LCD_Str_CH(20, 150, "截图完成", 0, 0xffff);
33.
34.
35. while (1)
36. {
37. }
```

5.5.4 显示汉字

这里先说汉字字符的显示，第 17 行：LCD_Str_CH(20, 30, "阿莫论坛野火专区", 0, 0xffff);

该函数功能是显示汉字字符串，源码如下：

```
1. /*****
2. * 函数名: LCD_Str_CH
3. * 描述   : 在指定坐标处显示 16*16 大小的指定颜色汉字字符串
4. * 输入    : - x: 显示位置横向坐标
5. *           - y: 显示位置纵向坐标
6. *           - str: 显示的中文字符串
7. *           - Color: 字符颜色
8. *           - bkColor: 背景颜色
9. * 输出    : 无
10. * 举例   : LCD_Str_CH(0, 0, "阿莫论坛野火专区", 0, 0xffff);
11.             LCD_Str_CH(50, 100, "阿莫论坛野火专区", 0, 0xffff);
12.             LCD_Str_CH(320-16*8, 240-16, "阿莫论坛野火专区", 0, 0xffff);
13. * 注意   : 字符串显示方向为横向 已测试
14. *****/
15. void LCD_Str_CH(u16 x, u16 y, const u8 *str, u16 Color, u16 bkColor)
16. {
17.     Set_direction(0);
18.     while(*str != '\0')
19.     {
20.         if(x > (320-16))
21.         {
22.             /*换行*/
23.             x = 0;
24.             y += 16;
25.         }
26.         if(y > (240-16))
27.         {
28.             /*重新归零*/
29.             y = 0;
30.         }
31.         LCD_Str_CH(x, y, *str, Color, bkColor);
32.         str++;
33.     }
34. }
```

```
32.             x =0;
33.         }
34.         LCD_Char_CH(x,y,str,Color,bkColor);
35.         str += 2 ;
36.         x += 16 ;
37.     }
38. }
```

该函数其实没做到什么工作，对超出屏幕范围的显示坐标进行换行处理，并把字符串中的汉字一个一个提取出来调用单字符显示函数 `LCD_Char_CH()` 显示出来，`LCD_Char_CH()` 函数的源码如下：

```
1.  /*****
2.  * 函数名: LCD_Char_CH
3.  * 描述   : 显示单个汉字字符
4.  * 输入    :    x: 0~(319-16)
5.  *          y: 0~(239-16)
6.  *          str: 中文字符串首址
7.  *          Color: 字符颜色
8.  *          bkColor: 背景颜色
9.  * 输出    : 无
10. * 举例   :    LCD_Char_CH(200,100,"好",0,0);
11. * 注意   : 如果输入大于1的汉字字符串，显示将会截断，只显示最前面一个汉字
12. *****/
13. void LCD_Char_CH(u16 x,u16 y,const u8 *str,u16 Color,u16 bkColor)
14. {
15.
16. #ifndef NO_CHNISEST_DISPLAY           /*如果汉字显示功能没有关闭*/
17.     u8 i,j;
18.     u8 buffer[32];
19.     u16 tmp_char=0;
20.
21.
22.     GetGBKCode_from_sd(buffer,str); /* 取字模数据 */
23.
24.     for (i=0;i<16;i++)
25.     {
26.         tmp_char=buffer[i*2];
27.         tmp_char=(tmp_char<<8);
28.         tmp_char|=buffer[2*i+1];
29.         for (j=0;j<16;j++)
30.         {
31.             if ( (tmp_char >> 15-j) & 0x01 == 0x01)
32.             {
33.                 LCD_ColorPoint(x+j,y+i,Color);
34.             }
35.             else
36.             {
37.                 LCD_ColorPoint(x+j,y+i,bkColor);
38.             }
39.         }
40.     }
41.
42. #endif
43. }
```

函数中的条件编译 `#ifndef NO_CHNISEST_DISPLAY`，是用于开关汉字显示功能的，若定义了 `NO_CHNISEST_DISPLAY`，则本函数为空，关闭了显示汉字的功能。

在 `LCD_Char_CH()` 这个函数中，首先调用 `GetGBKCode_from_sd()` 从 SD 卡中读出我们需要显示在 LCD 上的指定汉字的字模数据。

接着在 22~40 行的代码就根据字模数据来描写，把字模中为 1 的数据位，在 LCD 屏中的像素点中使用画点函数 `LCD_ColorPoint()` 显示特定的颜色。思路和前面 VC 测试部分用数字“8”来显示“当”字是一样的。

5.5.4.1 查找字模

读者现在可能在想，字库里面保存着大量的汉字字幕信息，现在输入 `GetGBKCode_from_sd(buffer, str)` 就能够拷贝出这个字符的字模数据，是怎样定位字模信息所在的位置的呢？换句话说，假如现在要显示“吾”字，是怎样根据这个字来确定“吾”字符在字库中的保存位置的呢？其实这里面有一定的映射关系，那就是接下来要说的汉字“区码”和“位码”。

在前面生成的 `HZLIB.bin` 文件，实际是按国标 GB2312 生成的二级汉字库。在国标 GB2312—80 中规定，所有的国标汉字及符号在字库中的存储形式是：分配在一个 94 行、94 列的阵列中，阵列的每一行称为一个“区”，共有 01 区到 94 区；每一列称为一个“位”，共有 01 位到 94 位，阵列中的每一个汉字和符号所在的区号和位号组合在一起形成的四个阿拉伯数字就是它们的“区位码”。区位码的前两位是它的区号，后两位是它的位号。*我们生成的汉字库就是这样按区位码排列的阵列，通过区位码，就能查找出该字的字模。*

*汉字的机内码*是指在计算机中表示一个汉字的编码。为避免与 ASCII 码混淆。用机内码的两个字节表示一个汉字，这两个字节分别称为高位字节和低位字节。

高位字节 = 区码 + 20H + 80H(或区码 + A0H)

低位字节 = 位码 + 20H + 80H(或位码 + A0H)

因此，我们就可以通过汉字的机内码，运算得出汉字在字库中的区位码，由区位码查找出该汉字的字模。



下面以 vc6.0 的测试源码来说明机内码、区位码的关系：

```
1. #include <stdio.h>
2. void main ()
3. {
4.     unsigned char * s , * e = "A" , * c = "古" ;
5.     unsigned char high_byte,lower_byte; //内码高字节, 内码低字节
6.     printf ( "字母%s 的 ASCII 码'=",e ) ;
7.     s = e ;
8.
9.     while ( * s != 0 )                //c 的字符串以 0 为结束符*
10.    {
11.        printf ( "%3d," , *s ) ;
12.        s ++ ;
13.    }
14.    printf ( "\n 汉字内码(10 进制) '%s'=",c ) ;
15.
16.    s = c ;
17.    while ( *s != 0 )
18.    {
19.        printf ( "%3d," , * s );
20.        s ++ ;
21.    }
22.
23.    printf ( "\n 汉字内码(16 进制) '%s'=",c ) ;
24.
25.    s = c ;
26.    while ( *s != 0 )
27.    {
28.        printf ( "%0X," , * s );
29.        s ++ ;
30.    }
31.
32.
33.    s = c ;
34.    high_byte = *s;
35.
36.    s ++ ;
37.    lower_byte = *s;
38.
39.    printf("\n\n 汉字'%s'对应的\n 内码高字节:%d\n 内码低字
    节:%d\n",c,high_byte,lower_byte);
40.    printf("\n\n 汉字'%s'对应的\n 区码为:%d-160 = %d\n 位码为:%d-
    160 = %d\n",c,high_byte,high_byte-160,lower_byte,lower_byte-160);
41.
42.    printf("\n\n 汉字'%s'在区位码表中的位置为%d%d\n",c,high_byte-
    160,lower_byte-160);
43.    printf("汉字区位码表可参考网
    站:http://cs.scu.edu.cn/~wangbo/others/quweima.htm\n");
44.    printf("通过在线查阅,编号为%d%d 对应的汉字刚好就是'%s'\n\n",high_byte-
    160,lower_byte-160,c);
45.
46. }
```

测试结果如下：



```
C:\WINDOWS\system32\cmd.exe

D:\Administrator\桌面>main.exe
字母'A的ASCII码' = 65,
汉字内码<10进制>'古' = 185,197,
汉字内码<16进制>'古' = B9,C5,

汉字'古'对应的
内码高字节:185
内码低字节:197


汉字'古'对应的
区码为:185-160 = 25
位码为:197-160 = 37

汉字'古'在区位码表中的位置为2537
汉字区位码表可参考网站:http://cs.scu.edu.cn/~wangbo/others/quweima.htm
通过在线查阅,编号为2537对应的汉字刚好就是'古'

D:\Administrator\桌面>pause
请按任意键继续. . .

搜狗拼音 半:
```

打开汉字区位码表在线查询网站: <http://www.jscj.com/index/gb2312.php>

查询“古”汉字的区位码刚好如计算所得, 为 .

上面的测试结果说明了每一个汉字的内码具体作用。回到本实验工程中获取字模函数 `GetGBKCode_from_sd()` 中, 它的具体定义如下:

```
1.  /*****
2.  * 函数名: GetGBKCode_from_sd
3.  * 描述   : 从sd卡上的字库文件中拷贝指定汉字的字模数据
4.  * 输入   : pBuffer---数据保存地址
5.  *         c--汉字字符低字节码
6.  * 输出   :      0                (成功)
7.  *         -1                 (失败)
8.  *****/
9.
10. int GetGBKCode_from_sd(unsigned char* pBuffer, unsigned char * c)
11. {
12.     unsigned char High8bit, Low8bit;
13.     unsigned int pos;
14.     High8bit = *c;
15.     Low8bit = *(c+1);
16.
17.     pos = ((High8bit-0xb0)*94+Low8bit-0xa1)*2*16;
18.     f_mount(0, &myfs[0]);
19.     myres = f_open(&myfsrc, "0:/HZLIB.bin", FA_OPEN_EXISTING | FA_READ);
20.
21.     if (myres == FR_OK)
22.     {
23.         f_lseek(&myfsrc, pos); //制定读取位置
```

```
24.         myres = f_read( &myfsrc, pBuffer, 32, &mybr ); //16*16 大小的汉
           字其字模占用 16*2 个字节
25.         f_close(&myfsrc);
26.
27.         return 0;
28.     }
29.
30.     else
31.         return -1;
32.
33. }
```

第 17 行中的: $pos = ((High8bit-0xb0)*94+Low8bit-0xa1)*2*16$ 语句就是根据约定的映射关系由汉字内码求得该汉字字模在字库中的存放起始位置。之后就到指定的位置去拷贝字模数据就可以了。

以上就是利用 SD 卡字库实现 LCD 显示汉字的具体流程。对于 ASCII 码(包括英文字符)的显示, 原理实际上也是一样的, 由于 ASCII 码占用空间较少, 所以我们直接把它的字模数据以数组的形式存储在代码中, 这些数组在文件 *ascii.h* 和 *asc_font.h* 中定义。

5.6 实现 SD 卡 BMP 图像的读取与保存

5.6.1 显示 BMP 图

再回到前面的 main 函数, 其中调用了一个 BMP 图片显示函数 *Lcd_show_bmp()*, 其定义如下:

```
1.  /*****
2.  * 函数名: Lcd_show_bmp
3.  * 描述   : LCD 显示 RGB888 位图图片
4.  * 输入   : x                --显示横坐标 (0-319)
5.             y                --显示纵坐标 (0-239)
6.  *          pic_name         --图片名称
7.  * 输出   : 无
8.  * 举例   : Lcd_show_bmp(0, 0, "/test.bmp");
9.  * 注意   : 图片为 24 为真彩色位图图片
10.             图片宽度不能超过 320
11.             图片在 LCD 上的粘贴范围:纵向: [x, x+图像高度] 横
           向 [Y, Y+图像宽度]
12.             当图片为 320*240 时--建议 x 输入 0   y 输入 0
13. *****/
14. void Lcd_show_bmp(unsigned short int x, unsigned short int y, unsigned
    char *pic_name)
15. {
16.     int i, j, k;
17.     int width, height, l_width;
18.
19.     BYTE red, green, blue;
```



```
20.     BITMAPFILEHEADER bitHead;
21.     BITMAPINFOHEADER bitInfoHead;
22.     WORD fileType;
23.
24.     unsigned int read_num;
25.     unsigned char tmp_name[20];
26.     sprintf((char*)tmp_name, "0:%s", pic_name);
27.     f_mount(0, &bmpfs[0]);
28.
29.     BMP_DEBUG_PRINTF("file mount ok \r\n"); //使用串口输出调试信息
30.
31.     bmpres = f_open(&bmpfsrc, (char *)tmp_name, FA_OPEN_EXISTING | F
A_READ);
32.     Set_direction(0);
33.
34.     if(bmpres == FR_OK)
35.     {
36.         BMP_DEBUG_PRINTF("Open file success\r\n");
37.
38.         //读取位图文件头信息
39.         f_read(&bmpfsrc, &fileType, sizeof(WORD), &read_num);
40.
41.         if(fileType != 0x4d42)
42.         {
43.             BMP_DEBUG_PRINTF("file is not .bmp file!\r\n");
44.             return;
45.         }
46.         else
47.         {
48.             BMP_DEBUG_PRINTF("Ok this is .bmp file\r\n");
49.         }
50.
51.         f_read(&bmpfsrc, &bitHead, sizeof(tagBITMAPFILEHEADER), &read_num
);
52.
53.         showBmpHead(&bitHead);
54.         BMP_DEBUG_PRINTF("\r\n");
55.
56.         //读取位图信息头信息
57.         f_read(&bmpfsrc, &bitInfoHead, sizeof(BITMAPINFOHEADER), &read_nu
m);
58.         showBmpInforHead(&bitInfoHead);
59.         BMP_DEBUG_PRINTF("\r\n");
60.     }
61.     else
62.     {
63.         BMP_DEBUG_PRINTF("file open fail!\r\n");
64.         return;
65.     }
66.
67.     width = bitInfoHead.biWidth;
68.     height = bitInfoHead.biHeight;
69.
70.     l_width = WIDTHBYTES(width* bitInfoHead.biBitCount); //计算
位图的实际宽度并确保它为 32 的倍数
71.
72.     if(l_width>960)
73.     {
74.         BMP_DEBUG_PRINTF("\nSORRY, PIC IS TOO BIG (<=320)\n");
75.         return;
76.     }
77.
78.     if(bitInfoHead.biBitCount>=24)
79.     {
80.
81.         bmp_lcd(x, 240-y-height, width, height); //LCD 参数相关设置
82.
```




```
83.         for(i=0;i<height+1; i++)
84.         {
85.
86.             for(j=0; j<l_width; j++)    //将一行数据全部读入
87.             {
88.
89.                 f_read(&bmpfsrc,pColorData+j,1,&read_num);
90.             }
91.
92.             for(j=0;j<width;j++)//一行有效信息
93.             {
94.                 k = j*3;
95.
96.                 //一行中第k个像素的起点
97.                 red = pColorData[k+2];
98.                 green = pColorData[k+1];
99.                 blue = pColorData[k];
100.                 LCD_WR_Data(RGB24TORGB16(red,green,blue));    //写入
101.                 LCD-GRAM
102.             }
103.         }
104.         bmp_lcd_reset();    //lcd 扫描方向复原
105.     }
106.     else
107.     {
108.         BMP_DEBUG_PRINTF("SORRY, THIS PIC IS NOT A 24BITS REAL COLOR");
109.         return ;
110.     }
111.     f_close(&bmpfsrc);
112. }
```

该函数的主要工作流程是：读取头部信息确定宽度和高度并确定每一行后面具体需要读出的字节数(保证是 4 字节的倍数)----->读取一行像素点并显示-->读取下一行并显示-->直至读完所有行。

另外一点就是：**RGB24TORGB16**是个宏定义，因为图像数据是 RGB888 即 24 为真彩色，而我们的 LCD 是 RGB565 即 16 位色度的，所以我们需要按比例将 24 为真彩色压缩为 16 位。宏定义如下：

```
1. #define RGB24TORGB16(R,G,B) (((unsigned short int) (((R)>>3)<<11)|((G)>>2)<<5)| ((B)>>3)))
```

5.6.2 LCD 截图功能

为了实现截图功能，我们可以根据用户截屏范围的宽和高来构造 BMP 文件的信息头，并且根据位图宽与四字节对齐的关系来补充需要的 0 大小字节。在 main 函数中，我们调用了 **Screen_shot()**这个函数来截图。保存的图片是 24 位的真彩色。**Screen_shot()**的源码如下：



```
1.  /*****
2.  * 函数名: Screen_shot
3.  * 描述   : 截取 LCD 指定位置 指定宽高的像素 保存为 24 位真彩色 bmp 格式图片
4.  * 输入    :      x                      ---水平位置
5.  *                      y                      ---竖直位置
6.  *                      Width                  ---水平宽度
7.  *                      Height                ---竖直高度
8.  *                      filename              ---文件名
9.  * 输出    :      0                      ---成功
10. *          -1                      ---失败
11. *          8                      ---文件已存在
12. * 举例    : Screen_shot(0, 0, 320, 240, "/myScreen");-----全屏截图
13. * 注意    : x 范围[0,319] y 范围[0,239] Width 范围[0,320-x] Height 范围
14. *                      如果文件已存在,将直接返回
15. *****/

16. int Screen_shot(unsigned short int x, unsigned short int y, unsigned s
    hort int Width, unsigned short int Height, unsigned char *filename)
17. {
18.     unsigned char header[54] =
19.     {
20.         0x42, 0x4d, 0, 0, 0, 0,
21.         0, 0, 0, 0, 54, 0,
22.         0, 0, 40, 0, 0, 0,
23.         0, 0, 0, 0, 0, 0,
24.         0, 0, 1, 0, 24, 0,
25.         0, 0, 0, 0, 0, 0,
26.         0, 0, 0, 0, 0, 0,
27.         0, 0, 0, 0, 0, 0,
28.         0, 0, 0, 0, 0, 0,
29.         0, 0, 0
30.     };
31.     int i;
32.     int j;
33.     long file_size;
34.     long width;
35.     long height;
36.     unsigned short int tmp_rgb;
37.     unsigned char r,g,b;
38.     unsigned char tmp_name[30];
39.     unsigned int mybw;
40.     char kk[4]={0,0,0,0};
41.
42.
43.     // if(!(Width%4))
44.     //     file_size = (long)Width * (long)Height * 3 + 54;
45.     //else
46.     file_size = (long)Width * (long)Height * 3 + Height*(Width%4) + 54
;         //宽*高 +补充的字节 + 头部信息
47.
48.     header[2] = (unsigned char)(file_size & 0x000000ff);
49.     header[3] = (file_size >> 8) & 0x000000ff;
50.     header[4] = (file_size >> 16) & 0x000000ff;
51.     header[5] = (file_size >> 24) & 0x000000ff;
52.
53.
54.     width=Width;
55.     header[18] = width & 0x000000ff;
56.     header[19] = (width >> 8) & 0x000000ff;
57.     header[20] = (width >> 16) & 0x000000ff;
58.     header[21] = (width >> 24) & 0x000000ff;
59.
60.     height = Height;
61.     header[22] = height & 0x000000ff;
62.     header[23] = (height >> 8) & 0x000000ff;
```





```
63.     header[24] = (height >> 16) &0x000000ff;
64.     header[25] = (height >> 24) &0x000000ff;
65.
66.     sprintf((char*)tmp_name, "0:%s.bmp", filename);
67.     f_mount(0, &bmpfs[0]);
68.
69.
70.     bmpres = f_open( &bmpfsrc , (char*)tmp_name, FA_CREATE_NEW | FA_WRITE);
71.
72.     f_close(&bmpfsrc); //新建文件之后要先关闭再打开才能写入
73.     bmpres = f_open( &bmpfsrc , (char*)tmp_name, FA_OPEN_EXISTING | FA_WRITE);
74.     if ( bmpres == FR_OK )
75.     {
76.         bmpres = f_write(&bmpfsrc, header, sizeof(unsigned char)*54, &mybw);
77.         for(i=0;i<Height;i++) //高
78.         {
79.             if(!(Width%4))
80.             {
81.                 for(j=0;j<Width;j++) //宽
82.                 {
83.                     #ifdef HX8347
84.                         tmp_rgb = bmp4(j+y, Height-i+x);
85.                     #else
86.                         tmp_rgb = bmp4(Height-
87.                             i+x, j+y);
88.                     #endif
89.
90.                     r = GETR_FROM_RGB16(tmp_rgb);
91.                     g = GETG_FROM_RGB16(tmp_rgb);
92.                     b = GETB_FROM_RGB16(tmp_rgb);
93.
94.                     bmpres = f_write(&bmpfsrc, &b, sizeof(unsigned char), &mybw);
95.                     bmpres = f_write(&bmpfsrc, &g, sizeof(unsigned char), &mybw);
96.                     bmpres = f_write(&bmpfsrc, &r, sizeof(unsigned char), &mybw);
97.                 }
98.             }
99.         }
100.        else
101.        {
102.            for(j=0;j<Width;j++)
103.            {
104.                #ifdef HX8347
105.                    tmp_rgb = bmp4(j+y, Height-i+x);
106.                #else
107.                    tmp_rgb = bmp4(Height-
108.                        i+x, j+y);
109.                #endif
110.
111.                r = GETR_FROM_RGB16(tmp_rgb);
112.                g = GETG_FROM_RGB16(tmp_rgb);
113.                b = GETB_FROM_RGB16(tmp_rgb);
114.
115.                bmpres = f_write(&bmpfsrc, &b, sizeof(unsigned char), &mybw);
116.                bmpres = f_write(&bmpfsrc, &g, sizeof(unsigned char), &mybw);
117.                bmpres = f_write(&bmpfsrc, &r, sizeof(unsigned char), &mybw);
118.
```



```
119.         }
120.
121.         bmpres = f_write(&bmpfsrc, kk, sizeof(unsigned c
    har) * (Width%4), &mybw);
122.
123.
124.         }
125.     }
126.
127.     f_close(&bmpfsrc);
128.     return 0;
129. }
130. else if ( bmpres == FR_EXIST ) //如果文件已经存在
131. {
132.     f_close(&bmpfsrc);
133.     return FR_EXIST; //8
134. }
135.
136. else
137. { f_close(&bmpfsrc);
138.   return -1;
139. }
140. }
```

该函数中，调用了 *bmp4()* 这个函数，该函数返回 LCD 上指定位置的像素信息。*GETG_FROM_RGB16* (绿色)、*GETB_FROM_RGB16* (蓝色) 和 *GETR_FROM_RGB16* (红色) 都是宏定义，将 RGB565 即 16 位色度抽取其中的 RGB 数据并分别将其线性映射为 8 位数据即映射为 RGB888 真彩色。宏定义的内容如下：

```
1. #define GETR_FROM_RGB16(RGB565) ((unsigned char)(( (unsigned short i
    nt ) RGB565) >>11)<<3)) //返回 8 位 R
2. #define GETG_FROM_RGB16(RGB565) ((unsigned char)(( (unsigned short i
    nt ) (RGB565 & 0x7ff)) >>5)<<2)) //返回 8 位 G
3. #define GETB_FROM_RGB16(RGB565) ((unsigned char)(( (unsigned short i
    nt ) (RGB565 & 0x1f))<<3)) //返回 8 位 B
```

利用 *Screen_shot()* 函数，就实现了把屏幕的当前显示的图像转换为 BMP 文件的功能。

5.7 实验现象

把文件 *HZLIB.bin*、*pic1.bmp*、*pic2.bmp*、*pic3.bmp*，保存到 SD 卡中，再把该 SD 卡插入开发板的 SD 卡接口（也可直接把工程下的 SD 字库备份文件夹下的内容复制到 SD 卡的根目录），然后将野火 STM32 开发板供电

(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，接上液晶屏，将编译好的程序下载到开发板。

程序运行之后截图保存为“myScreen.bmp”如下：



(^ ^ 截图保存图片功能+摄像头模块 就构成了一个完整的照相机啦！！)

