LCD 显示中英文+ BMP 图片

作者	fire
E-Mail	firestm32@foxmail.com
QQ	313303034
博客	firestm32.blog.chinaunix.net
硬件平台	野火 STM32 开发板
库版本	ST3.0.0

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 卡中放入一张 bmp 图片作为 LCD 背景。并且调用截屏函数截取 LCD 背景并保存为 bmp 图片。

硬件连接：

MicroSD 卡

PC8-SDIO-D0	----DATA0
PC9-SDIO-D1	----DATA1
PC10-SDIO-D2	----DATA2
PC11-SDIO-D3	----CD/DATA3
PC12-SDIO-CLK	----CLK
PD2-SDIO-CMD	----CMD

TFT 数据线

PD14-FSMC-D0	----LCD-DB0
PD15-FSMC-D1	----LCD-DB1
PD0-FSMC-D2	----LCD-DB2
PD1-FSMC-D3	----LCD-DB3
PE7-FSMC-D4	----LCD-DB4
PE8-FSMC-D5	----LCD-DB5

PE9-FSMC-D6	----LCD-DB6
PE10-FSMC-D7	----LCD-DB7
PE11-FSMC-D8	----LCD-DB8
PE12-FSMC-D9	----LCD-DB9
PE13-FSMC-D10	----LCD-DB10
PE14-FSMC-D11	----LCD-DB11
PE15-FSMC-D12	----LCD-DB12
PD8-FSMC-D13	----LCD-DB13
PD9-FSMC-D14	----LCD-DB14
PD10-FSMC-D15	----LCD-DB15

TFT 控制信号线

PD4-FSMC-NOE	----LCD-RD
PD5-FSMC-NEW	----LCD-WR
PD7-FSMC-NE1	----LCD-CS
PD11-FSMC-A16	----LCD-DC
PE1-FSMC-NBL1	----LCD-RESET
PD13-FSMC-A18	----LCD-BLACK-LIGHT

库文件 ：startup/start_stm32f10x_hd.c

CMSIS/core_cm3.c

CMSIS/system_stm32f10x.c

FWlib/stm32f10x_rcc.c

FWlib/misc.c

Fwlib/stm32f10x_systick.c

FWlib/stm32f10x_exti.c

FWlib/stm32f10x_gpio.c

FWlib/stm32f10x_sdio.c

FWlib/stm32f10x_dma.c

FWlib/stm32f10x_usart.c

FWlib/stm32f10x_fsmc.c

用户文件：USER/main.c

USER/stm32f10x_it.c

USER/systick.c

USER/usart1.c

USER/lcd.c

USER/ff.c

USER/sdcard.c

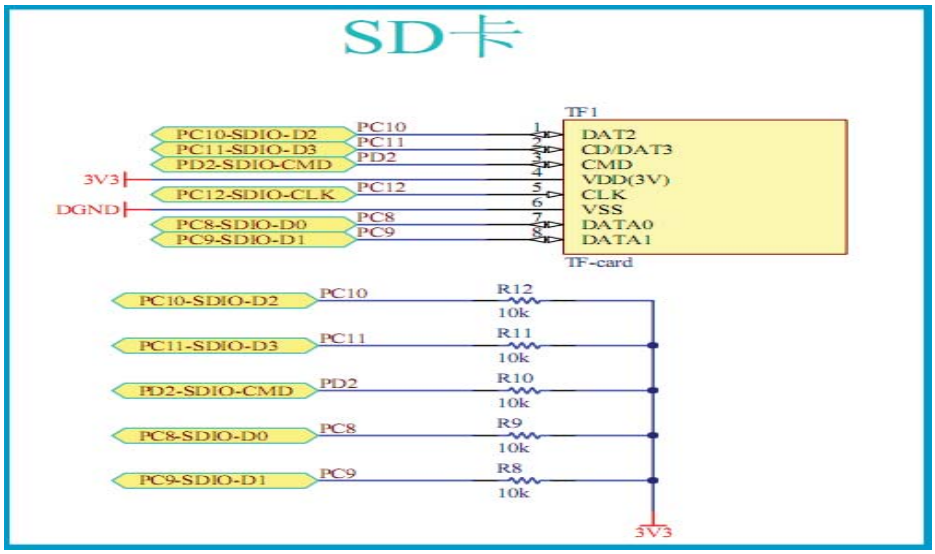
USER/diskio.c

USER/sd_fs_app.c

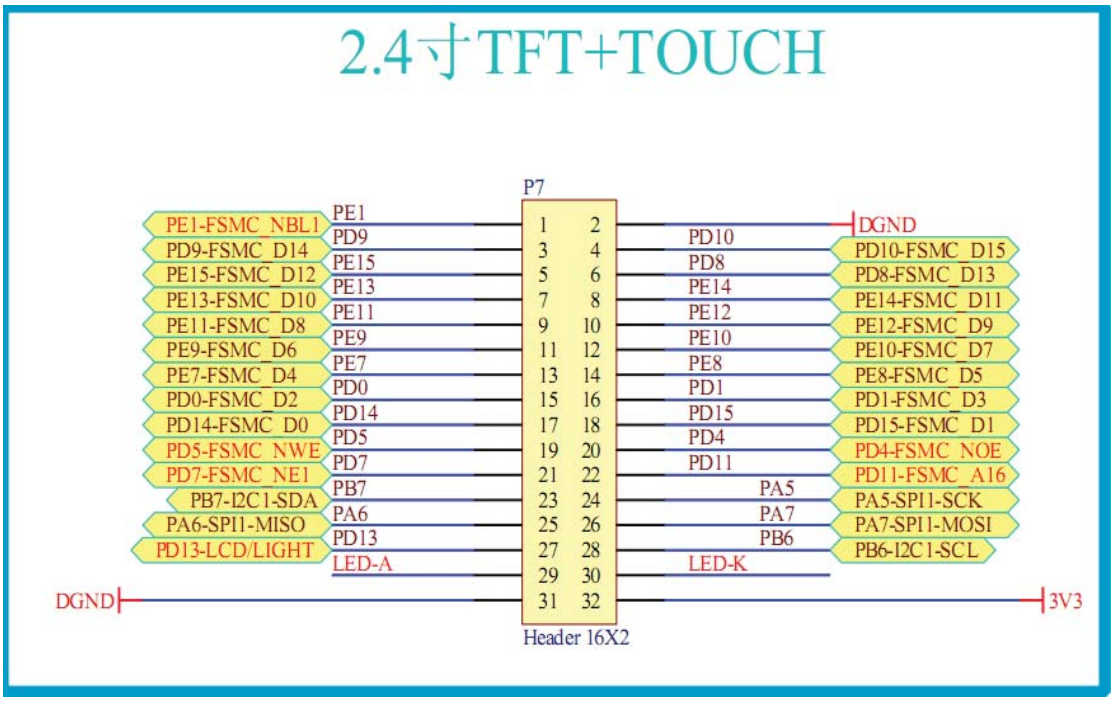
USER/Sd_bmp.c

野火 **STM32** 开发板 **LCD** 和 **SD** 卡 硬件连接图：

SD 卡接口连接如下



LCD 接口连接如下

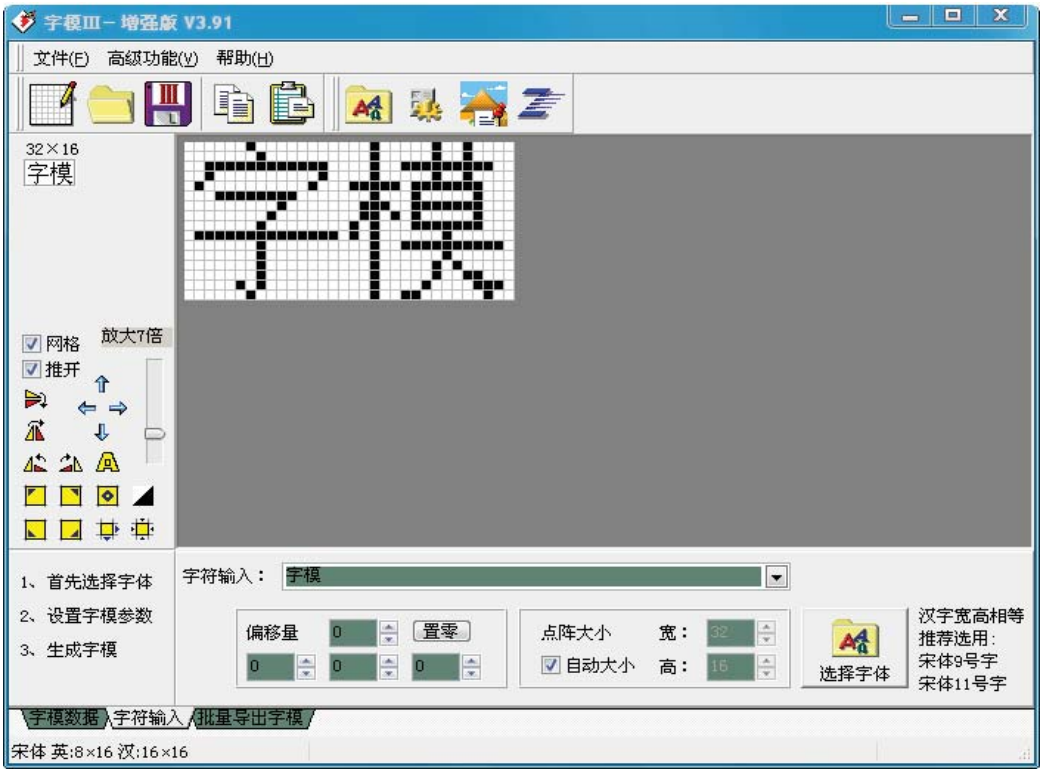


字库制作详细流程

我们采用“字模 III-增强版 v3.91” 

软件来制作中文字库。

1 打开字模软件

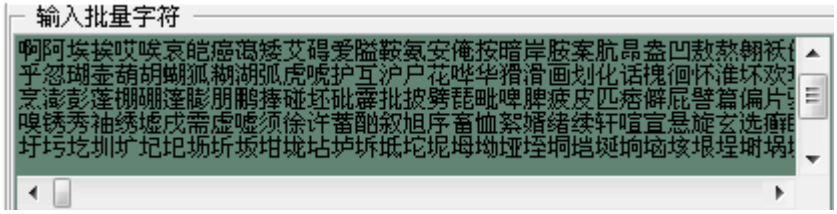


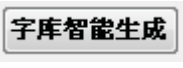
2 点击“自动批量生成字库”按钮选项  。

软件界面左下角将出现一下几个按钮选项：

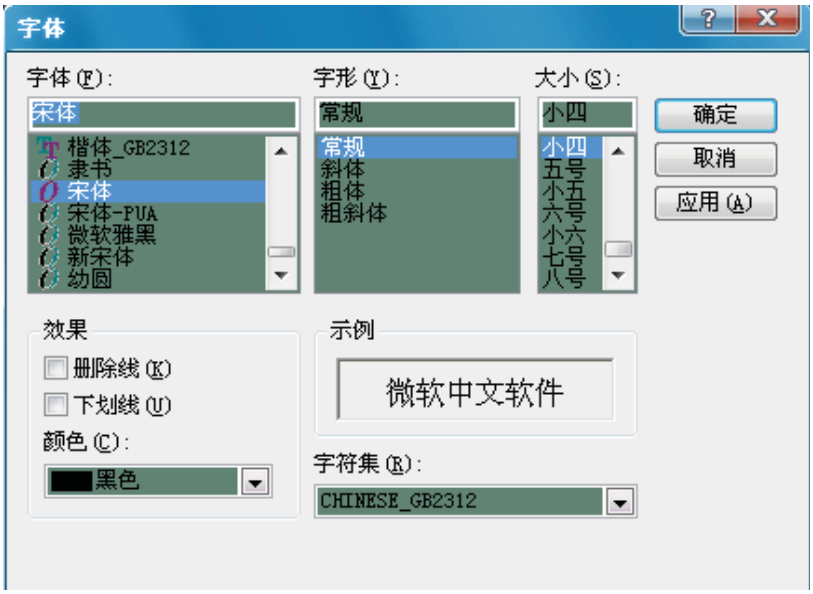
3 点击选择“二级汉字库”按钮。

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啦，如图：

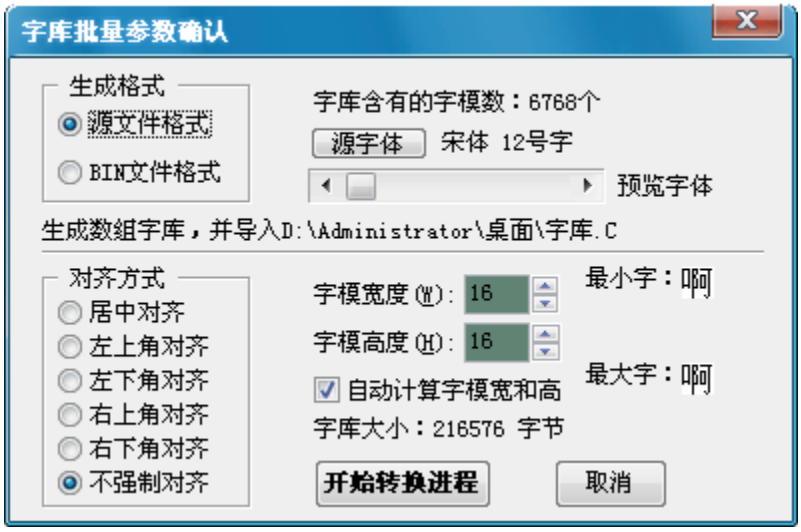


4 点击“字库智能生成”按钮  ，弹出“字库批量参数确认”对话框。

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 

.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。

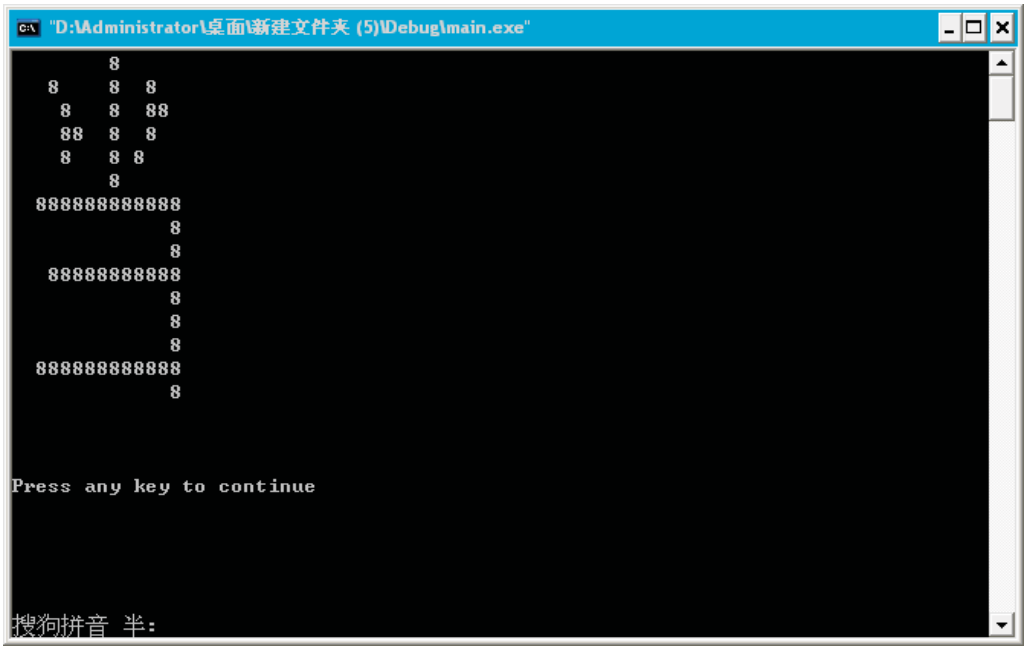
在这里我们取“当”字符的数据来测试。

```
1459  /* 当 */
1460  0x00,0x80,0x10,0x90,0x08,0x98,0x0C,0x90,0x08,0xA0,0x00,0x80,0x3F,0xFC,0x00,0x04,
1461  0x00,0x04,0x1F,0xFC,0x00,0x04,0x00,0x04,0x00,0x04,0x3F,0xFC,0x00,0x04,0x00,0x00,
```

VC6.0 测试源码如下：

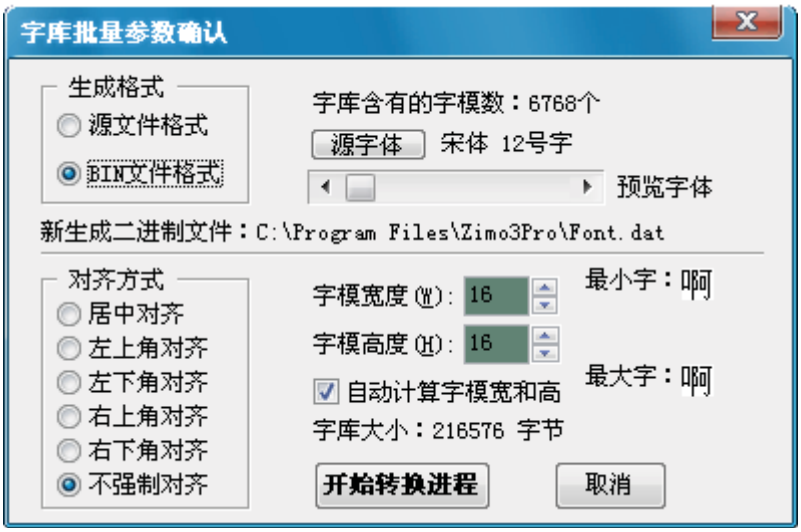
```
1.  #include <stdio.h>
2.
3.  unsigned char cc[] =
4.  {/*"当"字符*/
5.  0x00,0x80,0x10,0x90,0x08,0x98,0x0C,0x90,0x08,0xA0,0x00,0x80,0x3F,0xFC,
    0x00,0x04,
6.  0x00,0x04,0x1F,0xFC,0x00,0x04,0x00,0x04,0x00,0x04,0x3F,0xFC,0x00,0x04,
    0x00,0x00
7.  };
8.
9.  void main()
10. {
11.     int i,j;
12.     unsigned char kk;
13.     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.
45.         printf("\n");
46.
47.     }
48.     printf("\n\n");
49.
50.       }
51.
52.
```

测试结果如下：

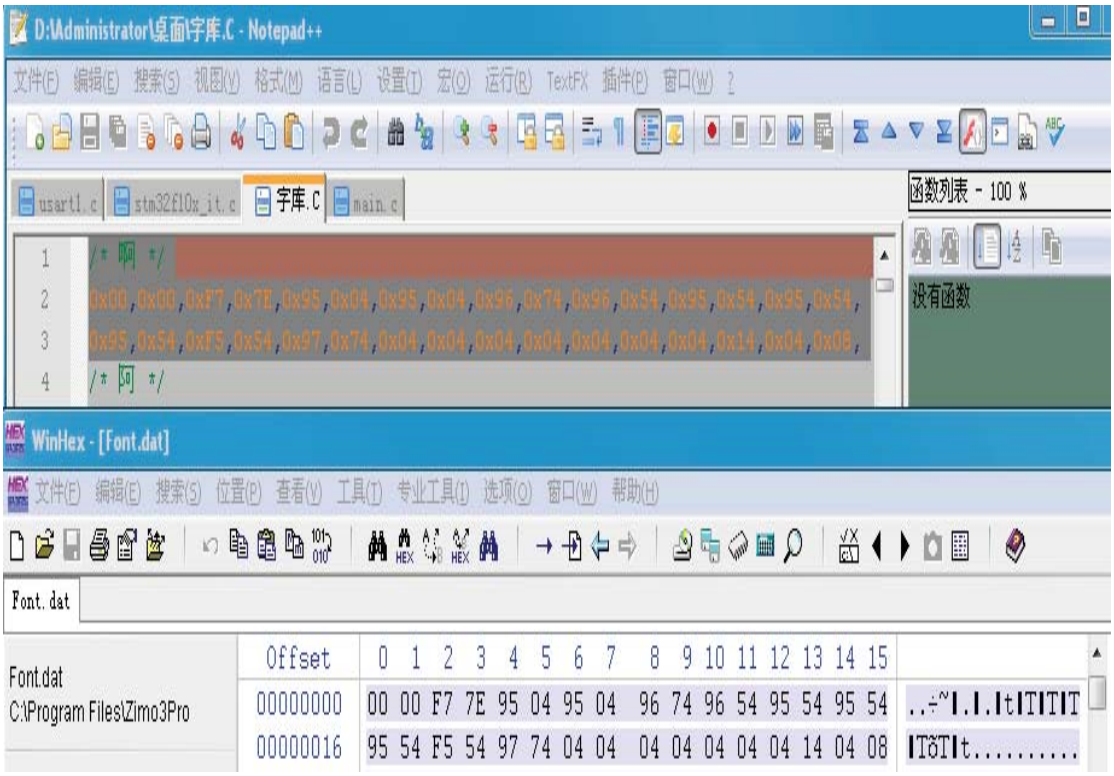


看到以上的测试结果，相信大家对汉字的取模方向和高低位的排列顺序有了比较直观的了解。

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”。

软件实现过程

先看看 **main** 函数先吧，源码如下：

```
1.  int main(void)
2.  {
3.      RCC_Configuration();                    //时钟配置
4.
5.      LCD_Init();                             //LCD 初始化
6.      USART1_Config();                        //串口 1 初始化
7.      sd_fs_init();                           //文件系统化
8.      Set_direction(0);                       //LCD AP 移动方向设置
9.      LCD_CLEAR(0,0,240,320);                 //清屏
10.     Lcd_show_bmp(0, 0,"/test.bmp");         //在 LCD 上显示 SD 卡上存放的
    test.bmp 图片
11.         //横屏显示
12.     LCD_ShowString(50,10,1,"6X12 ASCII");
13.     LCD_Show_8x16_String(50, 30, 1, "8X16 ASCII");
14.     PutChinese_strings21(50,50,"内存卡字库例程",0,1);
15.     PutChinese_strings11(50,70,"内存卡字库例程",0,0xffff);
16.
17.         //竖屏显示
18.    LCD_ShowString2(80,240,1,"6X12 ASCII");
19.    LCD_Show_8x16_String2(80,220, 1, "8X16 ASCII");
20.    PutChinese_strings22(80,200,"内存卡字库例程",0,1);
21.    PutChinese_strings12(80,180,"内存卡字库例程",0,0xffff);
```

```
22.
23.   PutChinese_strings11(50,120,"正在截图",0,0xffff);
24.   Screen_shot(0, 0, 320, 240, "/myScreen");  //截图操作
25.   PutChinese_strings11(50,120,"截图完成",0,0xffff);
26. }
```

Main 函数的工作就是在 LCD 上显示一些指定类型的字符，并完成显示 bmp 图像和截图功能。

这里先说汉字字符的显示，第 14 行：PutChinese_strings21(50,50,"内存卡字库例程",0,1);

该函数功能是显示汉子字符串，源码如下：

```
1.  /*********************************************************
2.   * 函数名: PutChinese_strings21
3.   * 描述   : 显示汉字字符串
4.   * 输入   : pos: 0~(319-16)
5.   *          Ypos: 0~(239-16)
6.   *          str: 中文字符串首址
7.   *          Color: 字符颜色
8.   *          mode: 0--文字背景色为白色
9.   *                1--文字悬浮
10.  * 输出   : 无
11.  * 举例   : PutChinese_strings2(200,100,"好人",0,0);
12.  * 注意     : 无
13.  *********************************************************/
14. void PutChinese_strings21(uint16_t Xpos,uint16_t Ypos,uint8_t *str,uin
    t16_t Color,u8 mode)  //横屏
15. {
16.
17.     uint16_t Tmp_x, Tmp_y;
18.     uint8_t *tmp_str=str;
19.     Tmp_x = Xpos;
20.     Tmp_y = Ypos;
21.
22.     while(*tmp_str != '\0')
23.     {
24.         PutChinese21(Tmp_x,Tmp_y,tmp_str,Color,mode); //显示该汉字
25.
26.         tmp_str += 2 ;
27.         Tmp_x += 16 ;
28.     }
29. }
```

该函数其实没做到什么工作，只是把字符串中的汉字一个一个提取出来并调用单字符显示函数 PutChinese21 显示出来，PutChinese21 函数的源码如下：

```
1.  /*********************************************************************
2.   * 函数名: PutChinese21
3.   * 描述   : 显示单个汉字字符串
4.   * 输入   : pos: 0~(319-16)
5.   *          Ypos: 0~(239-16)
6.   *          str: 中文字符串首址
7.   *          Color: 字符颜色
8.   *          mode: 0--文字背景色为白色
```

```
 *                1--文字悬浮
 * 输出  : 无
 * 举例  : PutChinese21(200,100,"好",0,0);
 * 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
 ***********************************************************************
****/
void PutChinese21(uint16_t Xpos,uint16_t Ypos,uint8_t *str,uint16_t Co
lor,u8 mode)
{
    uint8_t i,j;
    uint8_t buffer[32];                  //32 字节用于保存字模数据
    uint16_t tmp_char=0;
    Set_direction(0);
    GetGBKCode_from_sd(buffer,str);      // 从 sd 卡中取出字模数据

    for (i=0;i<16;i++)
    {
        tmp_char=buffer[i*2];
       tmp_char=(tmp_char<<8);
       tmp_char|=buffer[2*i+1];
        for (j=0;j<16;j++)
        {
            if ( (tmp_char >> 15-j) & 0x01 == 0x01)
            {
                LCD_Draw_ColorPoint(Ypos+i,Xpos+j,Color);
            }
            else
            {
                if ( mode == 0 )
                    LCD_Draw_ColorPoint(Ypos+i,Xpos+j,0xffff);  //背景
色显示白色
                else if ( mode == 1 )
                {
                    //不写入
                }
            }
        }
    }
}
```

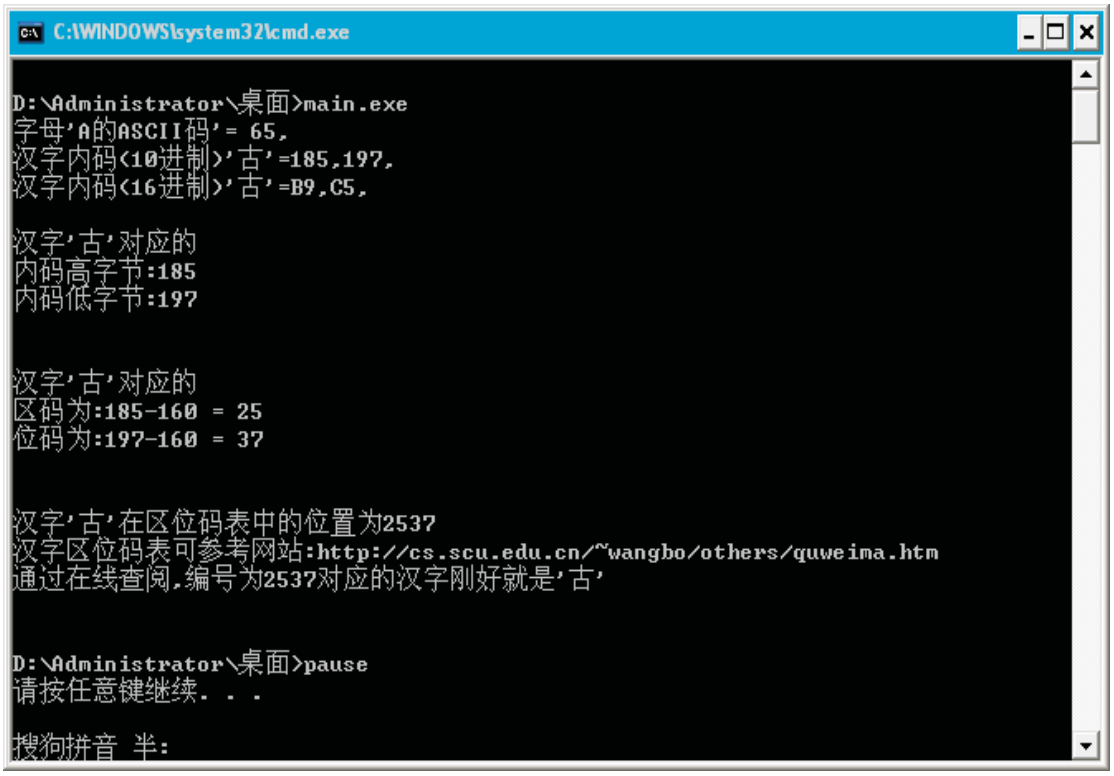
在 PutChinese21 这个函数中，首先从 sd 卡中读出我们需要显示在 LCD 上的指定汉字的字模数据，之后 22~43 行的代码就根据字模数据来描写，描写这一部分在这里就不再说啦，和前面 VC 测试部分思路是一样的。读者现在可能在想， 字库里面保存着大量的汉字字幕信息，我现在输入 GetGBKCode_from_sd(buffer,str)就能够拷贝出这个字符的字模数据，是怎样定位字模信息所在的位置的呢，换句话说，假如我现在要显示“吾”字，是怎样根据这个字来确定“吾”字符在字库中的保存位置的呢？其实这里面有一定的映射关系，那就是接下来要说的汉字“区码”和“位码”。

在汉字区位码表中，我们是如何定位我们指定汉字的编号的呢？接下来看看 vc6.0 测试源码：

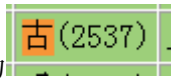
```
#include <stdio.h>
void main ()
{
    unsigned char * s , * e = "A" , * c = "古" ;
```

```
    unsigned char high_byte,lower_byte; //内码高字节，内码低字节
    printf ( "字母'%s 的 ASCII 码'=",e ) ;
    s = e ;

    while ( * s != 0 )                    //C 的字符串以 0 为结束符*
    {
        printf ( "%3d," , *s ) ;
        s ++ ;
    }
    printf ( "\n 汉字内码(10 进制)'%s'=",c ) ;

    s = c ;
    while ( *s != 0 )
    {
        printf ( "%3d," , * s );
        s ++ ;
    }

        printf ( "\n 汉字内码(16 进制)'%s'=",c ) ;

    s = c ;
    while ( *s != 0 )
    {
        printf ( "%0X," , * s );
        s ++ ;
    }


    s = c ;
    high_byte = *s;

    s ++ ;
    lower_byte = *s;

    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);
    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);

    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);
    printf("汉字区位码表可参考网站:http://cs.scu.edu.cn/~wangbo/others/quweima.htm\n");
    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);

}
```

测试结果如下：



打开汉字区位码表在线查询网站：http://cs.scu.edu.cn/~wangbo/others/quweima.htm

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。下面就来看看 PutChinese21 函数中调用到的函数 GetGBKCode_from_sd 的源码：

```
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_REA
    D);
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. }
```

pos = ((High8bit-0xb0)*94+Low8bit-0xa1)*2*16 这条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 SD 卡字库制作和实现的具体流程。

实现 SD 卡 BMP 图像的读取与保存

BMP 文件格式，又称为 Bitmap（位图）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 文件保存了一幅图象中所有的像素。这种文件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，每种模式种单一像素的大小分别为 1/8 字节，1/2 字节，1 字节和 3 字节。目前最常见的是 256 位色 BMP 和 24 位色 BMP。这种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 24 位色 BMP 没有这个，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。

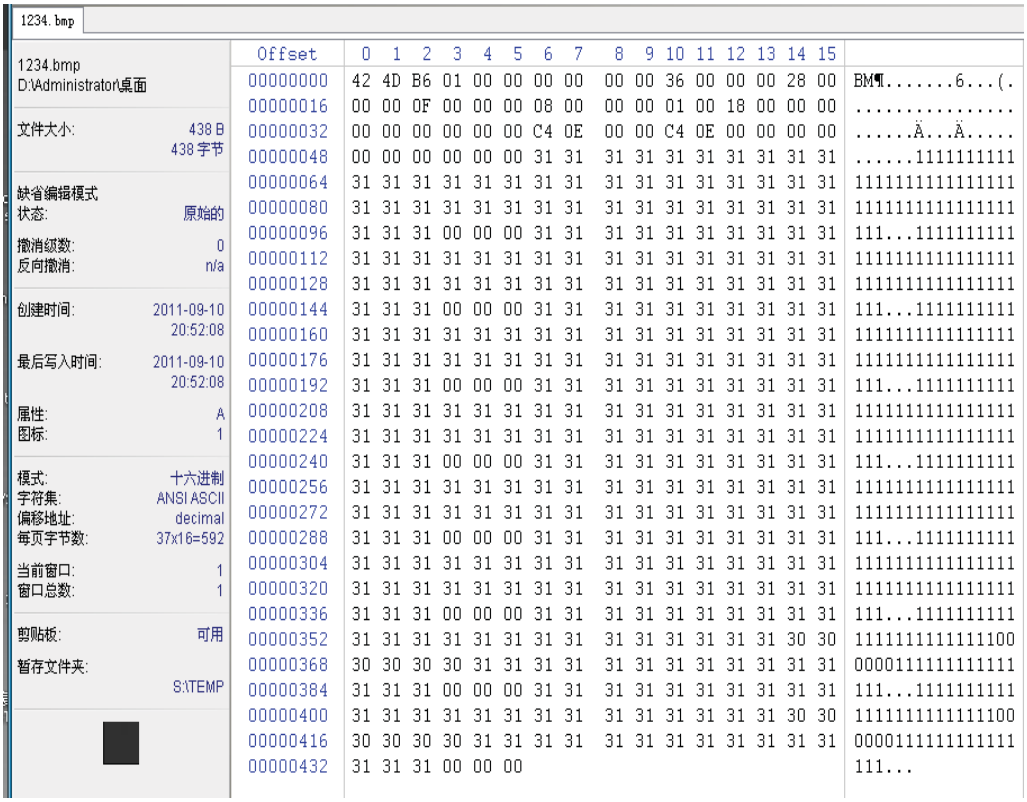
图片分析：

下面来用 WinHex 软件来分析一下 bmp 图像的文件内容：

测试图片 123.bmp，用 ACDSee 软件打开，其图像内容为

可知，这幅图像的大小是 15*8，是 24 位真彩色 bmp 图像。

图片数据为：



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	BM¶.....6...(.
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	......Ä...Ä.....
00000048	00	00	00	00	00	00	31	31	31	31	31	31	31	31	31	31	1111111111
00000064	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000080	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000096	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000112	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000128	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000144	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000160	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000176	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000192	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000208	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000224	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000240	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000256	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000272	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000288	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000304	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000320	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111
00000336	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000352	31	31	31	31	31	31	31	31	31	31	31	31	31	31	30	30	1111111111111100
00000368	30	30	30	30	31	31	31	31	31	31	31	31	31	31	31	31	0000111111111111
00000384	31	31	31	00	00	00	31	31	31	31	31	31	31	31	31	31	111...1111111111
00000400	31	31	31	31	31	31	31	31	31	31	31	31	31	31	30	30	1111111111111100
00000416	30	30	30	30	31	31	31	31	31	31	31	31	31	31	31	31	0000111111111111
00000432	31	31	31	00	00	00											111...

1 文件头部信息部分(前面 54 字节)：

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

如上图，阴影部分就是文件头部信息。

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 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 31 31 31
00000384  31 31 31 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 31 31 31
00000432  31 31 31 00 00 00
```

3 对文件头部信息部分的解析：

这里可以分为两块：BMP 文件头和位图信息头

从 0 开始：

0 和 1 字节 是文件的类型，如果是位图文件类型，必须分别为 0x42 和 0x4D 。

接下来的部分可以用一个结构体来描述。

3 到 14 字节的意义如下：

```
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;
```

头部信息剩下的部分就是位图信息头信息：

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
6.  LONG biHeight;          //位图高
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

4 文件像素部分的分析

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，如下图灰色部分所示：

```
00000096 31 31 31 00 00 00 31 31  31 31 31 31 31 31 31 31
```

整张图片都是灰色的，为什么会有 0 像素的出现呢？

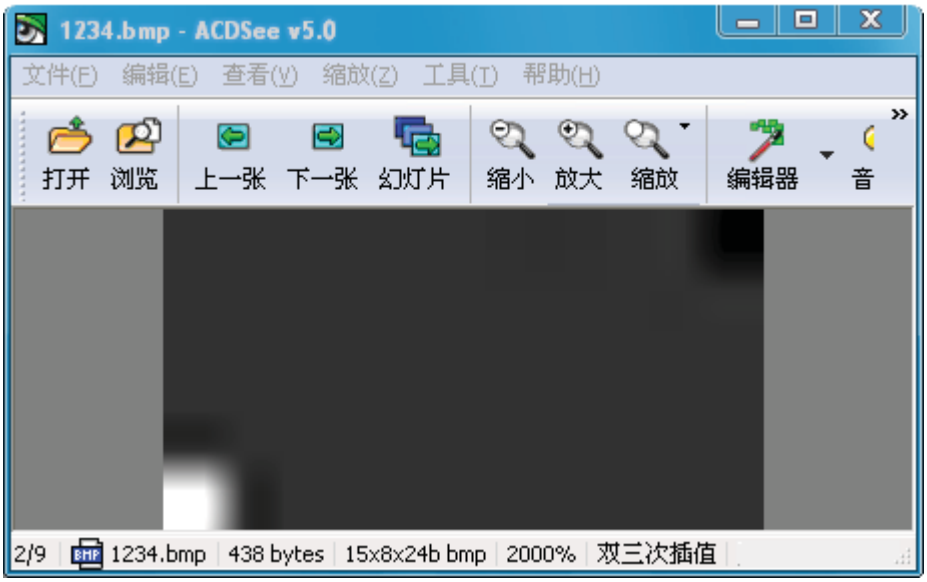
由于对齐的数据,操作系统或者硬件更容易把它调入 cache,使得 cache 的命中率提高，从而提高访问效率。也就是基于性能上得考虑，所以 Windows 规定一个扫描行所占的字节数必须是 4 的倍数(即以 long 为单位),不足的以 0 填充。上面分析可知，位图宽:15，　因此，未补充字节前字节数为 15*3 等于 45 字节，要到 4 的倍数，必须向上取 4 的倍数即 48，所以就有了不上 3 个字节 0 的结果。

至此，整个图像文件的大小为：54+ （15*3 + 3）*8　等于 438 字节，和前面所得到的信息文件大小: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。

下面我们修改一下图片来验证一下：

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)：

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

对应我们图像的左下角。

后尾部分是黑色像素（灰色部分）：

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

对应我们图像的右上角。

对 BMP 图像文件内容有了具体的了解之后我们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（由于 pc 机上，24 位真彩色使用比较广，在此只分析 24 位为真彩色）。

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，其源码如下：

```
/*****************************************************
* 函数名: Lcd_show_bmp
* 描述   : LCD 显示 RGB888 位图图片
* 输入   : x                        --显示纵坐标(0-239)
*          y                        --显示横坐标(0-319)
*          pic_name                 --图片名称
* 输出   : 无
* 举例   : Lcd_show_bmp(0, 0,"/test.bmp");
* 注意   : 图片为 24 为真彩色位图图片
                        图片宽度不能超过 320
                        图片在 LCD 上的粘贴范围为:纵向:  [x,x+图像高度]    横向 [Y, Y+图像宽度]
                        当图片为 320*240 时--建议 X 输入 0   y 输入 0
 *****************************************************/
BYTE pColorData[960];
FATFS bmpfs[2];
FIL bmpfsrc, bmpfdst;
FRESULT bmpres;
void Lcd_show_bmp(unsigned short int x, unsigned short int y,unsigned char *pic_name)
{
    int k;
    int i;
    int j;
    int width;
    int height;
    int l_width;
    BYTE red,green,blue;
```

```
27.     BITMAPFILEHEADER bitHead;
28.     BITMAPINFOHEADER bitInfoHead;
29.     WORD fileType;
30.     unsigned int read_num;
31.     unsigned char tmp_name[20];
32.     sprintf((char*)tmp_name,"0:%s",pic_name);
33.     f_mount(0, &bmpfs[0]);
34.
35.     bmpres = f_open( &bmpfsrc , (char *)tmp_name, FA_OPEN_EXISTING | F
    A_READ);    //打开 bmp 文件
36.
37.     if(bmpres == FR_OK)
38.     {
39.         USART1_printf(USART1,"Open file success\r\n");
40.
41.         //读取位图文件类型信息
42.         f_read(&bmpfsrc,&fileType,sizeof(WORD),&read_num);//读取 1、2 字
    节(文件类型)
43.
44.         if(fileType != 0x4d42)    //0x4d42 是 bmp 图像类型标志
45.         {
46.             USART1_printf(USART1,"file is not .bmp file!\r\n");
47.             return;
48.         }
49.         else
50.         {
51.             USART1_printf(USART1,"Ok this is .bmp file\r\n");
52.         }
53.
54.            //读取位图文件头信息结构(该结构已经在前面描述过啦)
55.         f_read(&bmpfsrc,&bitHead,sizeof(tagBITMAPFILEHEADER),&read_num
    );
56.
57.         showBmpHead(&bitHead);
58.         USART1_printf(USART1,"\r\n");
59.
60.         //读取位图信息头信息(该结构已经在前面描述过啦)
61.         f_read(&bmpfsrc,&bitInfoHead,sizeof(BITMAPINFOHEADER),&read_nu
    m);
62.
63.         showBmpInforHead(&bitInfoHead);
64.         USART1_printf(USART1,"\r\n");
65.     }
66.
67.     else
68.     {
69.         USART1_printf(USART1,"file open fail!\r\n");
70.         return;
71.     }
72.
73.
74.     width = bitInfoHead.biWidth;
75.     height = bitInfoHead.biHeight;
76.
77.     l_width = WIDTHBYTES(width* bitInfoHead.biBitCount);
        //计算位图的实际宽度并确保它为 32 的倍数(4 字节倍数)
78.
79.     if(l_width>960)
80.     {
81.         USART1_printf(USART1,"\nSORRY, PIC IS TOO BIG (<=320)\n");
82.         return;
83.     }
84.
85.     if(bitInfoHead.biBitCount>=24)
86.     {
87.
```

```
88.         bmp(x,y,width, height);                 //LCD 参数相关设置
89.
90.         for(i=0;i<height; i++)
91.         {
92.
93.             for(j=0; j<l_width; j++)            //将一行数据全部读入
94.             {
95.
96.                 f_read(&bmpfsrc,pColorData+j,1,&read_num);
97.             }
98.
99.
100.                for(j=0;j<width;j++)                //一行有效信息
101.                {
102.                    k = j*3;                        //一行中第 k 个像素的起
    点
103.                    red = pColorData[k+2];
104.                    green = pColorData[k+1];
105.                    blue =  pColorData[k];
106.                    MY_LCD_WR_Data(RGB24TORGB16(red,green,blue));
        //写入 LCD-GRAM
107.                }
108.
109.
110.            }
111.
112.            bmp3();                         //lcd 扫描方向复原
113.
114.        }
115.
116.        else
117.        {
118.
119.            USART1_printf(USART1,"SORRY, THIS PIC IS NOT A 24BITS R
    EAL COLOR");
120.            return ;
121.        }
122.
123.        f_close(&bmpfsrc);
124.
125.    }
```

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)))
```

完成 LCD 截图功能。

我们可以根据用户的长和高来构造我们前面的信息头，并且根据长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：

```
/*************************************************************
 * 函数名: Screen_shot
 * 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
 * 输入   :     x                                    ---水平位置
 *              y                                    ---竖直位置
 *              Width                                ---水平宽度
 *              Height                               ---竖直高度
 *              filename                             ---文件名
 * 输出   :      0                                   ---成功
 *              -1                                   ---失败
 *              8                                    ---文件已存在
 * 举例   : Screen_shot(0, 0, 320, 240, "/myScreen");全屏截图
 * 注意   : x 范围[0,319]  y 范围[0,239]  Width 范围[0,320-x]  Height 范围[0,240-y]
 *                   如果文件已存在,将直接返回
 *************************************************************/
int Screen_shot(unsigned short int x, unsigned short int y, unsigned short int Width, unsigned short int Height, unsigned char *filename)
{   //直接构造前面 54 字节的头部信息
    unsigned char header[54] =
    {
        0x42, 0x4d, 0, 0, 0, 0,
        0, 0, 0, 0, 54, 0,
        0, 0, 40,0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 24, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0,
        0, 0, 0, 0, 0,
        0, 0, 0, 0, 0,
        0, 0, 0
    };
    int i;
    int j;
    long file_size;
    long width;
    long height;
    unsigned short int tmp_rgb;
    unsigned char r,g,b;
    unsigned char tmp_name[30];
    unsigned int mybw;
    char kk[4]={0,0,0,0};


        file_size = (long)Width * (long)Height * 3 + Height*(Width%4) + 54;     //宽*高 +补充的字节 + 头部信息

    header[2] = (unsigned char)(file_size &0x000000ff);
    header[3] = (file_size >> 8) & 0x000000ff;
    header[4] = (file_size >> 16) & 0x000000ff;
    header[5] = (file_size >> 24) & 0x000000ff;


    width=Width;
```

```
52.     header[18] = width & 0x000000ff;
53.     header[19] = (width >> 8) &0x000000ff;
54.     header[20] = (width >> 16) &0x000000ff;
55.     header[21] = (width >> 24) &0x000000ff;
56.
57.     height = Height;
58.     header[22] = height &0x000000ff;
59.     header[23] = (height >> 8) &0x000000ff;
60.     header[24] = (height >> 16) &0x000000ff;
61.     header[25] = (height >> 24) &0x000000ff;
62.
63.     sprintf((char*)tmp_name,"0:%s.bmp",filename);
64.     f_mount(0, &bmpfs[0]);
65.     bmpres = f_open( &bmpfsrc , (char*)tmp_name, FA_CREATE_NEW | FA_WR
    ITE);
66.
67.     if ( bmpres == FR_OK )
68.     {
69.         bmpres = f_write(&bmpfsrc, header,sizeof(unsigned char)*54, &m
    ybw);
70.         for(i=0;i<Height;i++)                          //高
71.         {
72.             if(!(Width%4))
73.             {
74.                 for(j=0;j<Width;j++)     //宽
75.                 {
76.
77.                     tmp_rgb = bmp4(Height-i+x,j+y);//获取该位置 LCD 的像
    素值
78.
79.                     r =  GETR_FROM_RGB16(tmp_rgb);
80.                     g =  GETG_FROM_RGB16(tmp_rgb);
81.                     b =  GETB_FROM_RGB16(tmp_rgb);
82.
83.                     bmpres = f_write(&bmpfsrc, &b,sizeof(unsigned char
    ), &mybw);
84.                     bmpres = f_write(&bmpfsrc, &g,sizeof(unsigned char
    ), &mybw);
85.                     bmpres = f_write(&bmpfsrc, &r,sizeof(unsigned char
    ), &mybw);
86.
87.                 }
88.
89.             }
90.             else
91.             {
92.                 for(j=0;j<Width;j++)
93.                 {
94.
95.                     tmp_rgb =bmp4(Height-i+x,j+y); //获取该位置 LCD 的像
    素值
96.
97.                     r =  GETR_FROM_RGB16(tmp_rgb);
98.                     g =  GETG_FROM_RGB16(tmp_rgb);
99.                     b =  GETB_FROM_RGB16(tmp_rgb);
100.
101.                         bmpres = f_write(&bmpfsrc, &b,sizeof(unsign
    ed char), &mybw);
102.                         bmpres = f_write(&bmpfsrc, &g,sizeof(unsign
    ed char), &mybw);
103.                         bmpres = f_write(&bmpfsrc, &r,sizeof(unsign
    ed char), &mybw);
104.
105.
106.                     }
107.
```

```
                        bmpres = f_write(&bmpfsrc, kk,sizeof(unsigned char)*(Width%4), &mybw);


                    }
                }

                f_close(&bmpfsrc);
                return 0;
            }
            else if ( bmpres == FR_EXIST )   //如果文件已经存在
            {
                return FR_EXIST;                            //8
            }

            else
            {
                return -1;
            }

        }
```

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、GETB_FROM_RGB16 和 GETR_FROM_RGB16 都是宏定义，将 RGB565 即 16 位色度抽取出其中的 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：

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

实验现象如下：

放入 sd 卡的背景图：“test.bmp”如下：

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！！)

实验讲解完毕，野火祝大家学习愉快^_^……