# AN1602B ATK-OV5640 模块使用说明

# （照相机实验）

本应用文档（AN1602B，对应探索者 STM32F407 开发板）将教大家如何在 ALIENTEK 探索者 F407 开发板上使用 ATK-OV5640 五百万像素摄像头模块。

本文档分为如下几部分：

1， BMP&JPEG 编码简介

2， 硬件连接

3， 软件实现

4， 验证

## 1、BMP&JPEG 编码简介

本章，我们要实现的照相机，支持 BMP 图片格式的照片和 JPEG 图片格式的照片，这里简单介绍一下这两种图片格式的编码。

### 1.1 BMP 编码简介

BMP（全称 Bitmap）是 Window 操作系统中的标准图像文件格式，文件后缀名为“.bmp”，使用非常广。它采用位映射存储格式，除了图像深度可选以外，不采用其他任何压缩，因此，BMP 文件所占用的空间很大，但是没有失真。BMP 文件的图像深度可选 lbit、4bit、8bit、16bit、24bit 及 32bit。BMP 文件存储数据时，图像的扫描方式是按从左到右、从下到上的顺序。

典型的 BMP 图像文件由四部分组成：

1， 位图头文件数据结构，它包含 BMP 图像文件的类型、显示内容等信息；

2， 位图信息数据结构，它包含有 BMP 图像的宽、高、压缩方法，以及定义颜色等信息

1， 调色板，这个部分是可选的，有些位图需要调色板，有些位图，比如真彩色图（24 位的 BMP）就不需要调色板；

2， 位图数据，这部分的内容根据 BMP 位图使用的位数不同而不同，在 24 位图中直接使用 RGB，而其他的小于 24 位的使用调色板中颜色索引值。

关于 BMP 的详细介绍，请参考光盘的《BMP 图片文件详解.pdf》。

通过上面的了解，BMP 文件是由文件头、位图信息头、颜色信息和图形数据等四部分组成。我们先来了解下这几个部分。

1、BMP 文件头（14 字节）：BMP 文件头数据结构含有 BMP 文件的类型、文件大小和位图起始位置等信息。

```
//BMP 文件头
typedef __packed struct
{
    u16    bfType ;            //文件标志.只对'BM',用来识别 BMP 位图类型
```

```
    u32    bfSize ;            //文件大小,占四个字节
    u16    bfReserved1 ;       //保留
    u16    bfReserved2 ;       //保留
    u32    bfOffBits ;         //从文件开始到位图数据(bitmap data)开始之间的偏移量
}BITMAPFILEHEADER ;
```

2、位图信息头（40 字节）：BMP 位图信息头数据用于说明位图的尺寸等信息。

```
typedef __packed struct
{
    u32 biSize ;               //说明 BITMAPINFOHEADER 结构所需要的字数。
    long    biWidth ;          //说明图象的宽度，以象素为单位
    long    biHeight ;         //说明图象的高度，以象素为单位
    u16    biPlanes ;          //为目标设备说明位面数，其值将总是被设为 1
    u16    biBitCount ;        //说明比特数/象素，其值为 1、4、8、16、24、或 32
    u32 biCompression ;        //说明图象数据压缩的类型。其值可以是下述值之一：
    //BI_RGB：没有压缩；
    //BI_RLE8：每个象素 8 比特的 RLE 压缩编码，压缩格式由 2 字节组成
    //BI_RLE4：每个象素 4 比特的 RLE 压缩编码，压缩格式由 2 字节组成
    //BI_BITFIELDS：每个象素的比特由指定的掩码决定。
    u32 biSizeImage ;
    //说明图象的大小,以字节为单位。当用 BI_RGB 格式时,可设置为 0
    long    biXPelsPerMeter ;//说明水平分辨率，用象素/米表示
    long    biYPelsPerMeter ;//说明垂直分辨率，用象素/米表示
    u32 biClrUsed ;             //说明位图实际使用的彩色表中的颜色索引数
    u32 biClrImportant ;       //说明对图象显示有重要影响的颜色索引的数目，
                               //如果是 0，表示都重要。
}BITMAPINFOHEADER ;
```

3、颜色表：颜色表用于说明位图中的颜色，它有若干个表项，每一个表项是一个 RGBQUAD 类型的结构，定义一种颜色。

```
typedef __packed struct
{
    u8 rgbBlue ;          //指定蓝色强度
    u8 rgbGreen ;         //指定绿色强度
    u8 rgbRed ;           //指定红色强度
    u8 rgbReserved ;      //保留，设置为 0
}RGBQUAD ;
```

4、位图数据：位图数据记录了位图的每一个像素值，记录顺序是在扫描行内是从左到右，扫描行之间是从下到上。位图的一个像素值所占的字节数:

当 biBitCount=1 时，8 个像素占 1 个字节;

当 biBitCount=4 时，2 个像素占 1 个字节;

当 biBitCount=8 时，1 个像素占 1 个字节;

当 biBitCount=16 时，1 个像素占 2 个字节;

当 biBitCount=24 时，1 个像素占 3 个字节;

当 biBitCount=32 时，1 个像素占 4 个字节;

biBitCount=1 表示位图最多有两种颜色，缺省情况下是黑色和白色，你也可以自己定义

这两种颜色。图像信息头装调色板中将有两个调色板项，称为索引 0 和索引 1。图象数据阵列中的每一位表示一个像素。如果一个位是 0，显示时就使用索引 0 的 RGB 值，如果位是 1，则使用索引 1 的 RGB 值。

biBitCount=16 表示位图最多有 65536 种颜色。每个像素用 16 位（2 个字节）表示。这种格式叫作高彩色，或叫增强型 16 位色，或 64K 色。它的情况比较复杂，当 biCompression 成员的值是 BI_RGB 时，它没有调色板。16 位中，最低的 5 位表示蓝色分量，中间的 5 位表示绿色分量，高的 5 位表示红色分量，一共占用了 15 位，最高的一位保留，设为 0。这种格式也被称作 555 16 位位图。如果 biCompression 成员的值是 BI_BITFIELDS，那么情况就复杂了，首先是原来调色板的位置被三个 DWORD 变量占据，称为红、绿、蓝掩码。分别用于描述红、绿、蓝分量在 16 位中所占的位置。在 Windows 95（或 98）中，系统可接受两种格式的位域：555 和 565，在 555 格式下，红、绿、蓝的掩码分别是：0x7C00、0x03E0、0x001F，而在 565 格式下，它们则分别为：0xF800、0x07E0、0x001F。你在读取一个像素之后，可以分别用掩码“与”上像素值，从而提取出想要的颜色分量（当然还要再经过适当的左右移操作）。在 NT 系统中，则没有格式限制，只不过要求掩码之间不能有重叠。（注：这种格式的图像使用起来是比较麻烦的，不过因为它的显示效果接近于真彩，而图像数据又比真彩图像小的多，所以，它更多的被用于游戏软件）。

biBitCount=32 表示位图最多有 4294967296(2 的 32 次方)种颜色。这种位图的结构与 16 位位图结构非常类似，当 biCompression 成员的值是 BI_RGB 时，它也没有调色板，32 位中有 24 位用于存放 RGB 值，顺序是：最高位—保留，红 8 位、绿 8 位、蓝 8 位。这种格式也被成为 888 32 位图。如果 biCompression 成员的值是 BI_BITFIELDS 时，原来调色板的位置将被三个 DWORD 变量占据，成为红、绿、蓝掩码，分别用于描述红、绿、蓝分量在 32 位中所占的位置。在 Windows 95(or 98)中，系统只接受 888 格式，也就是说三个掩码的值将只能是：0xFF0000、0xFF00、0xFF。而在 NT 系统中，你只要注意使掩码之间不产生重叠就行。（注：这种图像格式比较规整，因为它是 DWORD 对齐的，所以在内存中进行图像处理时可进行汇编级的代码优化（简单））。

通过以上了解，我们对 BMP 有了一个比较深入的了解，本章，我们采用 16 位 BMP 编码（因为我们的 LCD 就是 16 位色的，而且 16 位 BMP 编码比 24 位 BMP 编码更省空间），故我们需要设置 biBitCount 的值为 16，这样得到新的位图信息（BITMAPINFO）结构体：

```
typedef __packed struct
{
    BITMAPFILEHEADER bmfHeader;
    BITMAPINFOHEADER bmiHeader;
    u32 RGB_MASK[3];            //调色板用于存放 RGB 掩码.
}BITMAPINFO;
```

其实就是颜色表由 3 个 RGB 掩码代替。最后，我们来看看将 LCD 的显存保存为 BMP 格式的图片文件的步骤：

**1）创建 BMP 位图信息，并初始化各个相关信息**

这里，我们要设置 BMP 图片的分辨率为 LCD 分辨率、BMP 图片的大小（整个 BMP 文件大小）、BMP 的像素位数（16 位）和掩码等信息。

**2）创建新 BMP 文件，写入 BMP 位图信息**

我们要保存 BMP，当然要存放在某个地方（文件），所以需要先创建文件，同时先保存 BMP 位图信息，之后才开始 BMP 数据的写入。

**3）保存位图数据。**

这里就比较简单了，只需要从 LCD 的 GRAM 里面读取各点的颜色值，依次写入第二步

创建的 BMP 文件即可。注意：保存顺序（即读 GRAM 顺序）是从左到右，从下到上。

**4）关闭文件。**

使用 FATFS，在文件创建之后，必须调用 f_close，文件才会真正体现在文件系统里面，否则是不会写入的！这个要特别注意，写完之后，一定要调用 f_close。

BMP 编码就介绍到这里。

## 1.2 JPEG 编码简介

JPEG（Joint Photographic Experts Group）是一个由 ISO 和 IEC 两个组织机构联合组成的一个专家组，负责制定静态的数字图像数据压缩编码标准，这个专家组开发的算法称为 JPEG 算法，并且成为国际上通用的标准，因此又称为 JPEG 标准。JPEG 是一个适用范围很广的静态图像数据压缩标准，既可用于灰度图像又可用于彩色图像。

JPEG 专家组开发了两种基本的压缩算法，一种是采用以离散余弦变换（Discrete Cosine Transform，DCT）为基础的有损压缩算法，另一种是采用以预测技术为基础的无损压缩算法。使用有损压缩算法时，在压缩比为 25:1 的情况下，压缩后还原得到的图像与原始图像相比较，非图像专家难于找出它们之间的区别，因此得到了广泛的应用。

JPEG 压缩是有损压缩，它利用了人的视角系统的特性，使用量化和无损压缩编码相结合来去掉视角的冗余信息和数据本身的冗余信息。

JPEG 压缩编码分为三个步骤：

1）使用正向离散余弦变换（Forward Discrete Cosine Transform，FDCT）把空间域表示的图变换成频率域表示的图。

2）使用加权函数对 DCT 系数进行量化，这个加权函数对于人的视觉系统是最佳的。

3）使用霍夫曼可变字长编码器对量化系数进行编码。

这里我们不详细介绍 JPEG 压缩的过程了，大家可以自行查找相关资料。我们本章要实现的 JPEG 拍照，并不需要自己压缩图像，因为我们使用的 ALIENTEK OV5640 摄像头模块，直接就可以输出压缩后的 JPEG 数据，我们完全不需要理会压缩过程，所以本章我们实现 JPEG 拍照的关键，在于准确接收 OV5640 摄像头模块发送过来的编码数据，然后将这些数据保存为.jpg 文件，就可以实现 JPEG 拍照了。

在《ATK-OV5640 摄像头模块使用说明（探索者摄像头实验）》，我们定义了一个很大的数组 jpeg_buf（124KB）来存储 JPEG 图像数据，但本章，我们要用到内存管理，其他地方也要用到一些数组，所以，肯定无法再定义这么大的数组了。并且这个数组不能使用外部 SRAM（实测：DCMI 接口使用 DMA 直接传输 JPEG 数据到外部 SRAM 会出现数据丢失，所以 DMA 接收 JPEG 数据只能用内部 SRAM），所以，我们本章将使用 DMA 的双缓冲机制来读取，DMA 双缓冲读取 JPEG 数据框图如图 1.2.1 所示：

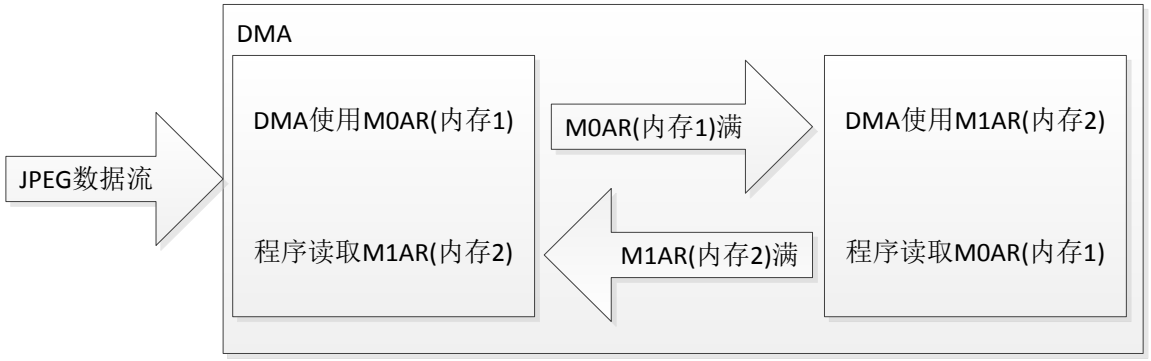


图 1.2.1 DMA 双缓冲读取 JPEG 数据原理框图

DMA 接收来自 OV5640 的 JPEG 数据流，首先使用 M0AR（内存 1）来存储，当 M0AR

满了以后，自动切换到 M1AR（内存 2），同时程序读取 M0AR（内存 1）的数据到外部 SRAM；当 M1AR 满了以后，又切回 M0AR，同时程序读取 M1AR（内存 2）的数据到外部 SRAM；依次循环（此时的数据处理，是通过 DMA 传输完成中断实现的，在中断里面处理），直到帧中断，结束一帧数据的采集，读取剩余数据到外部 SRAM，完成一次 JPEG 数据的采集。

这里，M0AR，M1AR 所指向的内存，必须是内部内存，不过由于采用了双缓冲机制，我们就不必定义一个很大的数组，一次性接收所有 JPEG 数据了，而是可以分批次接收，数组可以定义的比较小。

最后，将存储在外部 SRAM 的 jpeg 数据，保存为.jpg/.jpeg 存放在 SD 卡，就完成了一次 JPEG 拍照。

## 2、硬件连接

本章实验功能简介：开机的时候先检测字库，然后检测 SD 卡根目录是否存在 PHOTO 文件夹，如果不存在则创建，如果创建失败，则报错（提示拍照功能不可用）。在找到 SD 卡的 PHOTO 文件夹后，开始初始化 OV5640，在初始化成功之后，就一直在屏幕显示 OV5640 拍到的内容。按下 KEY2 可以手动单次自动对焦。当按下 KEY_UP 按键的时候，可以选择缩放，还是 1:1 显示，默认缩放。按下 KEY0，可以拍 bmp 图片照片（分辨率为：LCD 辨率）。按下 KEY1 可以拍 JPEG 图片照片（分辨率为 SXGA，即 1280*1024）。拍照保存成功之后，蜂鸣器会发出“滴”的一声，提示拍照成功。DS0 还是用于指示程序运行状态，DS1 用于提示 DCMI 帧中断。

所要用到的硬件资源如下：

1） 指示灯 DS0 和 DS1

2） KEY0、KEY1、KEY2 和 KEY_UP 按键

3） 蜂鸣器

4） 串口

5） LCD 模块

6） SD 卡

7） SPI FLASH

8） 摄像头模块

这几部分，在标准例程的实例中都介绍过了，我们在此就不介绍了。需要注意的是：SD 卡与 DCMI 接口有部分 IO 共用，所以他们不能同时使用，必须分时复用，本章，这部分共用 IO 我们只有在拍照保存的时候，才切换为 SD 卡使用，其他时间，都是被 DCMI 占用的。

## 3、软件实现

打开上一章的工程，由于本章要用到 OV5640、蜂鸣器、外部 SRAM 和定时器等外设，所以，先添加 dcmi.c、sccb.c、OV5640.c、beep.c、sram.c 和 timer.c 等文件到 HARDWARE 组下。

然后，我们来看下 PICTURE 组下的 bmp.c 文件里面的 bmp 编码函数：bmp_encode，该函数代码如下：

```
//BMP 编码函数
//将当前 LCD 屏幕的指定区域截图,存为 16 位格式的 BMP 文件  RGB565 格式.
//保存为 rgb565 则需要掩码,需要利用原来的调色板位置增加掩码.这里我们增加了掩码.
//保存为 rgb555 格式则需要颜色转换,耗时间比较久,所以保存为 565 是最快速的办法.
//filename:存放路径
```

```
//x,y:在屏幕上的起始坐标
//mode:模式.0,仅创建新文件;1,如果存在文件,则覆盖该文件.如果没有,则创建新的文件.
//返回值:0,成功;其他,错误码.
u8 bmp_encode(u8 *filename,u16 x,u16 y,u16 width,u16 height,u8 mode)
{
    FIL* f_bmp; u8 res=0;
    u16 bmpheadsize;            //bmp 头大小
    BITMAPINFO hbmp;            //bmp 头
    u16 tx,ty;                  //图像尺寸
    u16 *databuf;               //数据缓存区地址
    u16 pixcnt;                 //像素计数器
    u16 bi4width;               //水平像素字节数
    if(width==0||height==0)return PIC_WINDOW_ERR;           //区域错误
    if((x+width-1)>lcddev.width)return PIC_WINDOW_ERR;      //区域错误
    if((y+height-1)>lcddev.height)return PIC_WINDOW_ERR;    //区域错误
#if BMP_USE_MALLOC == 1     //使用 malloc
    databuf=(u16*)pic_memalloc(1024);
    //开辟至少 bi4width 大小的字节的内存区域 ,对 240 宽的屏,480 个字节就够了.
    if(databuf==NULL)return PIC_MEM_ERR;        //内存申请失败.
    f_bmp=(FIL *)pic_memalloc(sizeof(FIL));     //开辟 FIL 字节的内存区域
    if(f_bmp==NULL){pic_memfree(databuf); return PIC_MEM_ERR; }//内存申请失败.
#else
    databuf=(u16*)bmpreadbuf;
    f_bmp=&f_bfile;
#endif
    bmpheadsize=sizeof(hbmp);               //得到 bmp 文件头的大小
    mymemset((u8*)&hbmp,0,sizeof(hbmp));//  申请到的内存置零.
    hbmp.bmiHeader.biSize=sizeof(BITMAPINFOHEADER);//信息头大小
    hbmp.bmiHeader.biWidth=width;       //bmp 的宽度
    hbmp.bmiHeader.biHeight=height;     //bmp 的高度
    hbmp.bmiHeader.biPlanes=1;          //恒为 1
    hbmp.bmiHeader.biBitCount=16;       //bmp 为 16 位色 bmp
    hbmp.bmiHeader.biCompression=BI_BITFIELDS;
    //每个象素的比特由指定的掩码决定。
    hbmp.bmiHeader.biSizeImage=hbmp.bmiHeader.biHeight*hbmp.bmiHeader.biWidth*
    hbmp.bmiHeader.biBitCount/8;//bmp 数据区大小
    hbmp.bmfHeader.bfType=((u16)'M'<<8)+'B';    //BM 格式标志
    hbmp.bmfHeader.bfSize=bmpheadsize+hbmp.bmiHeader.biSizeImage;
    //整个 bmp 的大小
    hbmp.bmfHeader.bfOffBits=bmpheadsize;       //到数据区的偏移
    hbmp.RGB_MASK[0]=0X00F800;                  //红色掩码
    hbmp.RGB_MASK[1]=0X0007E0;                  //绿色掩码
    hbmp.RGB_MASK[2]=0X00001F;                  //蓝色掩码
    if(mode==1)res=f_open(f_bmp,(const TCHAR*)filename,FA_READ|FA_WRITE);
```

```
        //尝试打开之前的文件
        if(mode==0||res==0x04)res=f_open(f_bmp,(const TCHAR*)filename,FA_WRITE|
        FA_CREATE_NEW);//模式 0,或者尝试打开失败,则创建新文件
        if((hbmp.bmiHeader.biWidth*2)%4)//水平像素(字节)不为 4 的倍数
        {
            bi4width=((hbmp.bmiHeader.biWidth*2)/4+1)*4;//实际像素,必须为 4 的倍数.
        }else bi4width=hbmp.bmiHeader.biWidth*2;            //刚好为 4 的倍数
        if(res==FR_OK)                                      //创建成功
        {
            res=f_write(f_bmp,(u8*)&hbmp,bmpheadsize,&bw);      //写入 BMP 首部
            for(ty=y+height-1;hbmp.bmiHeader.biHeight;ty--)
            {
                pixcnt=0;
                for(tx=x;pixcnt!=(bi4width/2);)
                {
                if(pixcnt<hbmp.bmiHeader.biWidth)databuf[pixcnt]=LCD_ReadPoint(tx,ty);
                //读取坐标点的值
                else databuf[pixcnt]=0Xffff;                    //补充白色的像素.
                pixcnt++; tx++;
                }
                hbmp.bmiHeader.biHeight--;
                res=f_write(f_bmp,(u8*)databuf,bi4width,&bw);   //写入数据
            }
            f_close(f_bmp);
        }
    #if BMP_USE_MALLOC == 1                                 //使用 malloc
        pic_memfree(databuf); pic_memfree(f_bmp);
    #endif
        return res;
    }
```

该函数实现了对 LCD 屏幕的任意指定区域进行截屏保存，用到的方法就是 47.1.1 节我们所介绍的方法，该函数实现了将 LCD 任意指定区域的内容，保存个为 16 位 BMP 格式，存放在指定位置（由 filename 决定）。注意，代码中的 BMP_USE_MALLOC 是在 bmp.h 定义的一个宏，用于设置是否使用 malloc，本章我们选择使用 malloc。

在 jpeg 拍照的时候，我们使用了双缓冲机制，且用到了 DMA 传输完成中断，这里我们需要修改 dcmi.c 里面的 DCMI_DMA_Init 函数，并添加 DMA 传输完成中断服务函数，代码如下：

```
    //DCMI DMA 配置
    //mem0addr:存储器地址 0   将要存储摄像头数据的内存地址(也可以是外设地址)
    //mem1addr:存储器地址 1   当只使用 mem0addr 的时候,该值必须为 0
    //memsize:存储器长度      0~65535
    //memblen:存储器位宽      0,8 位,1,16 位,2,32 位
    //meminc:存储器增长方式,0,不增长;1,增长
    void DCMI_DMA_Init(u32 mem0addr,u32 mem1addr,u16 memsize,u8 memblen,u8
```

```
meminc)
    {
        RCC->AHB1ENR|=1<<22;                //DMA2 时钟使能
        while(DMA2_Stream1->CR&0X01);       //等待 DMA2_Stream1 可配置
        DMA2->LIFCR|=0X3D<<6*1;             //清空通道 1 上所有中断标志
        DMA2_Stream1->FCR=0X0000021;        //设置为默认值
        DMA2_Stream1->PAR=(u32)&DCMI->DR;//外设地址为:DCMI->DR
        DMA2_Stream1->M0AR=mem0addr;        //mem0addr 作为目标地址 0
        DMA2_Stream1->M1AR=mem1addr;        //mem1addr 作为目标地址 1
        DMA2_Stream1->NDTR=memsize;         //传输长度为 memsize
        DMA2_Stream1->CR=0;                 //先全部复位 CR 寄存器值
        DMA2_Stream1->CR|=0<<6;             //外设到存储器模式
        DMA2_Stream1->CR|=1<<8;             //循环模式
        DMA2_Stream1->CR|=0<<9;             //外设非增量模式
        DMA2_Stream1->CR|=meminc<<10;       //存储器增量模式
        DMA2_Stream1->CR|=2<<11;            //外设数据长度:32 位
        DMA2_Stream1->CR|=memblen<<13;      //存储器位宽,8/16/32bit
        DMA2_Stream1->CR|=2<<16;            //高优先级
        DMA2_Stream1->CR|=0<<21;            //外设突发单次传输
        DMA2_Stream1->CR|=0<<23;            //存储器突发单次传输
        DMA2_Stream1->CR|=1<<25;            //通道 1 DCMI 通道
        if(mem1addr)                        //双缓冲的时候,才需要开启
        {
            DMA2_Stream1->CR|=1<<18;                    //双缓冲模式
            DMA2_Stream1->CR|=1<<4;                     //开启传输完成中断
            MY_NVIC_Init(0,0,DMA2_Stream1_IRQn,2);      //抢占 1，子优先级 3，组 2
        }
    }
    void (*dcmi_rx_callback)(void);                     //DCMI DMA 接收回调函数
    //DMA2_Stream1 中断服务函数(仅双缓冲模式会用到)
    void DMA2_Stream1_IRQHandler(void)
    {
        if(DMA2->LISR&(1<<11))                          //DMA2_Steam1,传输完成标志
        {
            DMA2->LIFCR|=1<<11;                         //清除传输完成中断
            dcmi_rx_callback();
            //执行摄像头接收回调函数,读取数据等操作在这里面处理
        }
    }
```

这里，DCMI_DMA_Init 函数，和第四十章相比增加了一个参数：mem1addr，用于设置 DMA 的第二个内存地址，同时根据该值是否为非 0，来判断是否需要使用双缓冲机制，只有在双缓冲机制下，才开启 DMA 传输完成中断。DMA2_Stream1_IRQHandler 函数，是 DMA 传输完成中断，里面通过 dcmi_rx_callback 回调函数（函数指针，指向 jpeg_dcmi_rx_callback 函数），及时将满了的内存（M0AR 或 M1AR）数据读取到外部 SRAM。

剩下的，我们就只需要修改主函数即可了，打开 test.c，修改该文件代码如下：

```
u8 bmp_request=0;
//bmp 拍照请求:0,无 bmp 拍照请求;1,有 bmp 拍照请求,需要在帧中断里面,关闭 DCMI 接口.
u8 ovx_mode=0;                          //bit0:0,RGB565 模式;1,JPEG 模式

#define jpeg_dma_bufsize    5*1024
//定义 JPEG DMA 接收时数据缓存 jpeg_buf0/1 的大小(*4 字节)
volatile u32 jpeg_data_len=0;           //buf 中的 JPEG 有效数据长度(*4 字节)
volatile u8 jpeg_data_ok=0;             //JPEG 数据采集完成标志
                                        //0,数据没有采集完;
                                        //1,数据采集完了,但是还没处理;
                                        //2,数据已经处理完成了,可以开始下一帧接收

u32 *jpeg_buf0;                         //JPEG 数据缓存 buf,通过 malloc 申请内存
u32 *jpeg_buf1;                         //JPEG 数据缓存 buf,通过 malloc 申请内存
u32 *jpeg_data_buf;                     //JPEG 数据缓存 buf,通过 malloc 申请内存

//处理 JPEG 数据
//当采集完一帧 JPEG 数据后,调用此函数,切换 JPEG BUF.开始下一帧采集.
void jpeg_data_process(void)
{
    u16 i;
    u16 rlen;                   //剩余数据长度
    u32 *pbuf;
    if(ovx_mode&0X01)           //只有在 JPEG 格式下,才需要做处理.
    {
        if(jpeg_data_ok==0)     //jpeg 数据还未采集完?
        {
            DMA2_Stream1->CR&=~(1<<0);      //停止当前传输
            while(DMA2_Stream1->CR&0X01);   //等待 DMA2_Stream1 可配置
            rlen=jpeg_dma_bufsize-DMA2_Stream1->NDTR;//得到剩余数据长度
            pbuf=jpeg_data_buf+jpeg_data_len;    //偏移到有效数据末尾,继续添加
            if(DMA2_Stream1->CR&(1<<19))for(i=0;i<rlen;i++)pbuf[i]=jpeg_buf1[i];
            //读取 buf1 里面的剩余数据
            else for(i=0;i<rlen;i++)pbuf[i]=jpeg_buf0[i];     //读取 buf0 里面的剩余数据
            jpeg_data_len+=rlen;                              //加上剩余长度
            jpeg_data_ok=1;
            //标记 JPEG 数据采集完按成,等待其他函数处理
        }
        if(jpeg_data_ok==2)                         //上一次的 jpeg 数据已经被处理了
        {
            DMA2_Stream1->NDTR=jpeg_dma_bufsize;
            //传输长度为 jpeg_buf_size*4 字节
            DMA2_Stream1->CR|=1<<0;                 //重新传输
```

```
                jpeg_data_ok=0;                         //标记数据未采集
                jpeg_data_len=0;                        //数据重新开始
            }
        }else
        {
            if(bmp_request==1)              //有 bmp 拍照请求,关闭 DCMI
            {
                DCMI_Stop();                //停止 DCMI
                bmp_request=0;              //标记请求处理完成.
            }
            LCD_SetCursor(0,0);
            LCD_WriteRAM_Prepare();         //开始写入 GRAM
        }
    }
    //jpeg 数据接收回调函数
    void jpeg_dcmi_rx_callback(void)
    {
        u16 i;
        u32 *pbuf;
        pbuf=jpeg_data_buf+jpeg_data_len;                   //偏移到有效数据末尾
        if(DMA2_Stream1->CR&(1<<19))                        //buf0 已满,正常处理 buf1
        {
            for(i=0;i<jpeg_dma_bufsize;i++)pbuf[i]=jpeg_buf0[i];//读取 buf0 里面的数据
            jpeg_data_len+=jpeg_dma_bufsize;                //偏移
        }else                                               //buf1 已满,正常处理 buf0
        {
            for(i=0;i<jpeg_dma_bufsize;i++)pbuf[i]=jpeg_buf1[i];//读取 buf1 里面的数据
            jpeg_data_len+=jpeg_dma_bufsize;                //偏移
        }
    }
    //切换为 OV5640 模式
    void sw_ov5640_mode(void)
    {
        OV5640_WR_Reg(0X3017,0XFF);             //开启 OV5650 输出(可以正常显示)
        OV5640_WR_Reg(0X3018,0XFF);             //GPIOC8/9/11 切换为 DCMI 接口
        GPIO_AF_Set(GPIOC,8,13);                //PC8,AF13  DCMI_D2
        GPIO_AF_Set(GPIOC,9,13);                //PC9,AF13  DCMI_D3
        GPIO_AF_Set(GPIOC,11,13);               //PC11,AF13 DCMI_D4
    }
    //切换为 SD 卡模式
    void sw_sdcard_mode(void)
    {
        OV5640_WR_Reg(0X3017,0X00);     //关闭 OV5640 全部输出(不影响 SD 卡通信)
        OV5640_WR_Reg(0X3018,0X00);     //GPIOC8/9/11 切换为 SDIO 接口
```

```
    GPIO_AF_Set(GPIOC,8,12);            //PC8,AF12
    GPIO_AF_Set(GPIOC,9,12);            //PC9,AF12
    GPIO_AF_Set(GPIOC,11,12);           //PC11,AF12
}
//文件名自增（避免覆盖）
//mode:0,创建.bmp 文件;1,创建.jpg 文件.
//bmp 组合成:形如"0:PHOTO/PIC13141.bmp"的文件名
//jpg 组合成:形如"0:PHOTO/PIC13141.jpg"的文件名
void camera_new_pathname(u8 *pname,u8 mode)
{
    u8 res;
    u16 index=0;
    while(index<0XFFFF)
    {
        if(mode==0)sprintf((char*)pname,"0:PHOTO/PIC%05d.bmp",index);
        else sprintf((char*)pname,"0:PHOTO/PIC%05d.jpg",index);
        res=f_open(ftemp,(const TCHAR*)pname,FA_READ);      //尝试打开这个文件
        if(res==FR_NO_FILE)break;                   //该文件名不存在=正是我们需要的.
        index++;
    }
}
//OV5640 拍照 jpg 图片
//返回值:0,成功
//其他,错误代码
u8 ov5640_jpg_photo(u8 *pname)
{
    FIL* f_jpg;
    u8 res=0;
    u32 bwr;
    u32 i;
    u8* pbuf;
    f_jpg=(FIL *)mymalloc(SRAMIN,sizeof(FIL));  //开辟 FIL 字节的内存区域
    if(f_jpg==NULL)return 0XFF;                 //内存申请失败.
    ovx_mode=1;
    jpeg_data_ok=0;
    sw_ov5640_mode();                           //切换为 OV5640 模式
    OV5640_JPEG_Mode();                         //JPEG 模式
    OV5640_OutSize_Set(16,4,1600,1200);         //设置输出尺寸(UXGA)
    dcmi_rx_callback=jpeg_dcmi_rx_callback;     //JPEG 接收数据回调函数
    DCMI_DMA_Init((u32)jpeg_buf0,(u32)jpeg_buf1,jpeg_dma_bufsize,2,1);
                                                //DCMI DMA 配置(双缓冲模式)
    DCMI_Start();                               //启动传输
    while(jpeg_data_ok!=1);                     //等待第一帧图片采集完
    jpeg_data_ok=2;             //忽略本帧图片,启动下一帧采集
```

```
        while(jpeg_data_ok!=1);     //等待第二帧图片采集完,第二帧,才保存到 SD 卡去.
        DCMI_Stop();                //停止 DMA 搬运
        ovx_mode=0;
        sw_sdcard_mode();           //切换为 SD 卡模式
        res=f_open(f_jpg,(const TCHAR*)pname,FA_WRITE|FA_CREATE_NEW);
                                    //模式 0,或者尝试打开失败,则创建新文件
        if(res==0)
        {
            printf("jpeg data size:%d\r\n",jpeg_data_len*4);        //串口打印 JPEG 文件大小
            pbuf=(u8*)jpeg_data_buf;
            for(i=0;i<jpeg_data_len*4;i++)
            //查找 0XFF,0XD8 和 0XFF,0XD9,获取 jpg 文件大小
            {
                if((pbuf[i]==0XFF)&&(pbuf[i+1]==0XD8))break;//找到 FF D8
            }
            if(i==jpeg_data_len*4)res=0XFD;     //没找到 0XFF,0XD8
            else                                //找到了
            {
                pbuf+=i;                        //偏移到 0XFF,0XD8 处
                res=f_write(f_jpg,pbuf,jpeg_data_len*4-i,&bwr);
                if(bwr!=(jpeg_data_len*4-i))res=0XFE;
            }
        }
        jpeg_data_len=0;
        f_close(f_jpg);
        sw_ov5640_mode();                   //切换为 OV5640 模式
        OV5640_RGB565_Mode();               //RGB565 模式
        DCMI_DMA_Init((u32)&LCD->LCD_RAM,0,1,1,0);
                                            //DCMI DMA 配置,MCU 屏,竖屏
        myfree(SRAMIN,f_jpg);
        return res;
    }

    int main(void)
    {
        u8 res,fac;
        u8 *pname;                  //带路径的文件名
        u8 key;                     //键值
        u8 i;
        u8 sd_ok=1;                 //0,sd 卡不正常;1,SD 卡正常.
        u8 scale=1;                 //默认是全尺寸缩放
        u8 msgbuf[15];              //消息缓存区

        Stm32_Clock_Init(336,8,2,7);//设置时钟,168Mhz
```

```
    delay_init(168);                    //延时初始化
    uart_init(84,115200);               //初始化串口波特率为 115200
    LED_Init();                         //初始化 LED
    usmart_dev.init(84);                //初始化 USMART
    TIM3_Int_Init(10000-1,8400-1);//10Khz 计数,1 秒钟中断一次
    LCD_Init();                         //LCD 初始化
    FSMC_SRAM_Init();                   //初始化外部 SRAM.
    BEEP_Init();                        //蜂鸣器初始化
    KEY_Init();                         //按键初始化
    W25QXX_Init();                      //初始化 W25Q128
    my_mem_init(SRAMIN);                //初始化内部内存池
    my_mem_init(SRAMEX);                //初始化内部内存池
    my_mem_init(SRAMCCM);               //初始化 CCM 内存池
    exfuns_init();                      //为 fatfs 相关变量申请内存
    POINT_COLOR=RED;
    while(font_init())                  //检查字库
    {
        LCD_ShowString(30,50,200,16,16,"Font Error!");
        delay_ms(200);
        LCD_Fill(30,50,240,66,WHITE);       //清除显示
        delay_ms(200);
    }
    while(OV5640_Init())
    {
        LCD_ShowString(30,50,200,16,16,"OV5640 Init error");
        delay_ms(200);
        LCD_Fill(30,50,240,66,WHITE);       //清除显示
        delay_ms(200);
    }
    sw_sdcard_mode();                       //切换为 SD 卡模式
    if(SD_Init())
    {
        LCD_ShowString(30,50,240,16,16,"SD Init error！ ");
        delay_ms(200);
        LCD_Fill(30,50,240,66,WHITE);       //清除显示
        delay_ms(200);
    }
    f_mount(fs[0],"0:",1);                  //挂载 SD 卡
    Show_Str(30,50,210,16,"Explorer STM32F4 开发板",16,0);
    Show_Str(30,70,200,16,"基于 OV5640 的照相机实验",16,0);
    Show_Str(30,90,200,16,"KEY0:拍照(bmp 格式)",16,0);
    Show_Str(30,110,200,16,"KEY1:拍照(jpg 格式)",16,0);
    Show_Str(30,130,200,16,"KEY2:自动对焦",16,0);
    Show_Str(30,150,200,16,"WK_UP:FullSize/Scale",16,0);
```

```
Show_Str(30,170,200,16,"2016 年 6 月 1 日",16,0);
res=f_mkdir("0:/PHOTO");                //创建 PHOTO 文件夹
if(res!=FR_EXIST&&res!=FR_OK)    //发生了错误
{
    Show_Str(30,190,240,16,"PHOTO 文件夹错误!!!!!",16,0);
    delay_ms(200);
    Show_Str(30,210,240,16,"拍照功能将不可用!",16,0);
    sd_ok=0;
}
jpeg_buf0=mymalloc(SRAMIN,jpeg_dma_bufsize*4);      //为 jpeg dma 接收申请内存
jpeg_buf1=mymalloc(SRAMIN,jpeg_dma_bufsize*4);      //为 jpeg dma 接收申请内存
jpeg_data_buf=mymalloc(SRAMEX,960*1024); //为 jpeg 文件申请内存(最大 960KB)
pname=mymalloc(SRAMIN,30);  //为带路径的文件名分配 30 个字节的内存
while(pname==NULL||!jpeg_buf0||!jpeg_buf1||!jpeg_data_buf)        //内存分配出错
{
    Show_Str(30,230,240,16,"内存分配失败!",16,0);
    delay_ms(200);
    LCD_Fill(30,230,240,146,WHITE);//清除显示
    delay_ms(200);
}
sw_ov5640_mode();                   //切换为 OV5640 模式
//自动对焦初始化
OV5640_RGB565_Mode();       //RGB565 模式
OV5640_Focus_Init();                //初始化自动对焦
OV5640_Light_Mode(0);               //自动模式
OV5640_Color_Saturation(3);     //色彩饱和度 0
OV5640_Brightness(4);               //亮度 0
OV5640_Contrast(3);                 //对比度 0
OV5640_Sharpness(33);               //自动锐度
OV5640_Focus_Constant();        //启动持续对焦
DCMI_Init();                        //DCMI 配置

DCMI_DMA_Init((u32)&LCD->LCD_RAM,0,1,1,0);
//DCMI DMA 配置,MCU 屏,竖屏
OV5640_OutSize_Set(16,4,lcddev.width,lcddev.height);    //满屏缩放显示
DCMI_Start();                                           //启动传输
while(1)
{
    key=KEY_Scan(0);                                    //不支持连按
    if(key)
    {
        if(key!=KEY2_PRES)
        {
            if(key==KEY0_PRES)
```

```
                    //如果是 BMP 拍照,则等待 1 秒钟,去抖动,以获得稳定的 bmp 照片
                    {
                        delay_ms(300);
                        bmp_request=1;          //BMP 拍照请求
                        while(bmp_request);     //等待请求处理完成
                    }else DCMI_Stop();
                }
                if(key==WKUP_PRES)              //缩放处理
                {
                    scale=!scale;
                    if(scale==0)
                    {
                        fac=800/lcddev.height;//得到比例因子

    OV5640_OutSize_Set((1280-fac*lcddev.width)/2,(800-fac*lcddev.height)/2,lcddev.width,lcd
dev.height);
                        sprintf((char*)msgbuf,"Full Size 1:1");
                    }else
                    {
                        OV5640_OutSize_Set(16,4,lcddev.width,lcddev.height);
                        sprintf((char*)msgbuf,"Scale");
                    }
                    delay_ms(800);
                }else if(key==KEY2_PRES)OV5640_Focus_Single(); //手动单次自动对焦
                else if(sd_ok)                                  //SD 卡正常才可以拍照
                {
                    sw_sdcard_mode();               //切换为 SD 卡模式
                    if(key==KEY0_PRES)              //BMP 拍照
                    {
                        camera_new_pathname(pname,0);       //得到文件名
                        res=bmp_encode(pname,0,0,lcddev.width,lcddev.height,0);
                        sw_ov5640_mode();                   //切换为 OV5640 模式
                    }else if(key==KEY1_PRES)                //JPG 拍照
                    {
                        camera_new_pathname(pname,1);       //得到文件名
                        res=ov5640_jpg_photo(pname);
                        if(scale==0)
                        {
                            fac=800/lcddev.height;          //得到比例因子

    OV5640_OutSize_Set((1280-fac*lcddev.width)/2,(800-fac*lcddev.height)/2,lcddev.width,lcd
dev.height);
                        }else
                        {
```

```
                        OV5640_OutSize_Set(16,4,lcddev.width,lcddev.height);
                    }
                }
                if(res)//拍照有误
                {
                    Show_Str(30,130,240,16,"写入文件错误!",16,0);
                }else
                {
                    Show_Str(30,130,240,16,"拍照成功!",16,0);
                    Show_Str(30,150,240,16,"保存为:",16,0);
                    Show_Str(30+42,150,240,16,pname,16,0);
                    BEEP=1;             //蜂鸣器短叫，提示拍照完成
                    delay_ms(100);
                    BEEP=0;             //关闭蜂鸣器
                }
                delay_ms(1000);         //等待 1 秒钟
                DCMI_Start();
                DCMI_Stop();
            }else //提示 SD 卡错误
            {
                Show_Str(30,130,240,16,"SD 卡错误!",16,0);
                Show_Str(30,150,240,16,"拍照功能不可用!",16,0);
            }
            if(key!=KEY2_PRES)DCMI_Start();         //开始显示
        }
        delay_ms(10);
        i++;
        if(i==20)                                   //DS0 闪烁.
        {
            i=0;
            LED0=!LED0;
        }
    }
}
```

## 4、验证

在代码编译成功之后，我们通过下载代码到 ALIENTEK 探索者 STM32F4 开发板上，得到如图 4.1 所示界面：

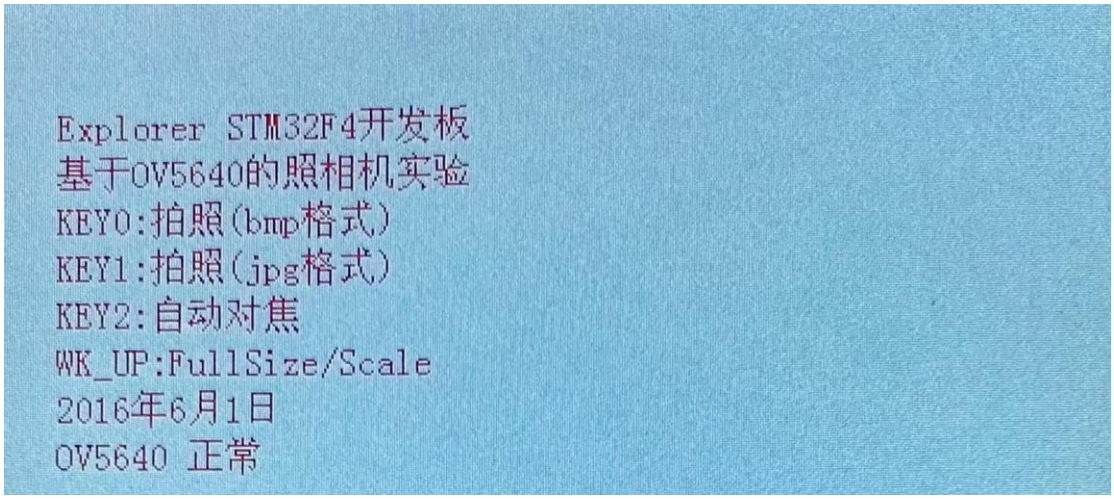


图 4.1 程序运行效果图

随后，进入监控界面。此时，我们可以按下 KEY0 和 KEY1，即可进行 bmp/jpg 拍照。拍照得到的照片效果如图 4.2 和图 4.3 所示：


图 4.2 拍照样图（bmp 拍照样图）


图 4.3 拍照样图（jpg 拍照样图）

最后，我们还可以通过 USMART 调用 bmp_encode 函数，实现串口控制 bmp 拍照，还可以拍成各种尺寸哦（不过必须≤LCD 分辨率）！

正点原子@ALIENTEK
公司网址：www.alientek.com
技术论坛：www.openedv.com
电话：020-38271790
传真：020-36773971