AN1506 ATK-OV2640 摄像头模块使用说明

本应用文档（AN1506，对应对应战舰 V3 / 精英 STM32F103 开发板扩展实验 6）将教大家如何在战舰 V3 / 精英 STM32F103 开发板上使用 ATK-OV2640 百万高清摄像头模块。

本文档分为如下几部分：

1， OV2640 简介
2， 硬件连接
3， 软件实现
4， 验证

1、OV2640 简介

OV2640 是 OV（OmniVision）公司生产的一颗 1/4 寸的 CMOS UXGA（1632*1232）图像传感器。该传感器体积小、工作电压低，提供单片 UXGA 摄像头和影像处理器的所有功能。通过 SCCB 总线控制，可以输出整帧、子采样、缩放和取窗口等方式的各种分辨率 8/10 位影像数据。该产品 UXGA 图像最高达到 15 帧/秒（SVGA 可达 30 帧，CIF 可达 60 帧）。用户可以完全控制图像质量、数据格式和传输方式。所有图像处理功能过程包括伽玛曲线、白平衡、对比度、色度等都可以通过 SCCB 接口编程。OmmiVision 图像传感器应用独有的传感器技术，通过减少或消除光学或电子缺陷如固定图案噪声、拖尾、浮散等，提高图像质量，得到清晰的稳定的彩色图像。

OV2640 的特点有：

- 高灵敏度、低电压适合嵌入式应用
- 标准的 SCCB 接口，兼容 IIC 接口
- 支持 RawRGB、RGB(RGB565/RGB555)、GRB422、YUV(422/420)和 YCbCr（422）输出格式
- 支持 UXGA、SXGA、SVGA 以及按比例缩小到从 SXGA 到 40*30 的任何尺寸
- 支持自动曝光控制、自动增益控制、自动白平衡、自动消除灯光条纹、自动黑电平校准等自动控制功能。同时支持色饱和度、色相、伽马、锐度等设置。
- 支持闪光灯
- 支持图像缩放、平移和窗口设置
- 支持图像压缩，即可输出 JPEG 图像数据
- 自带嵌入式微处理器

OV2640 的功能框图图如图 1.1 所示：

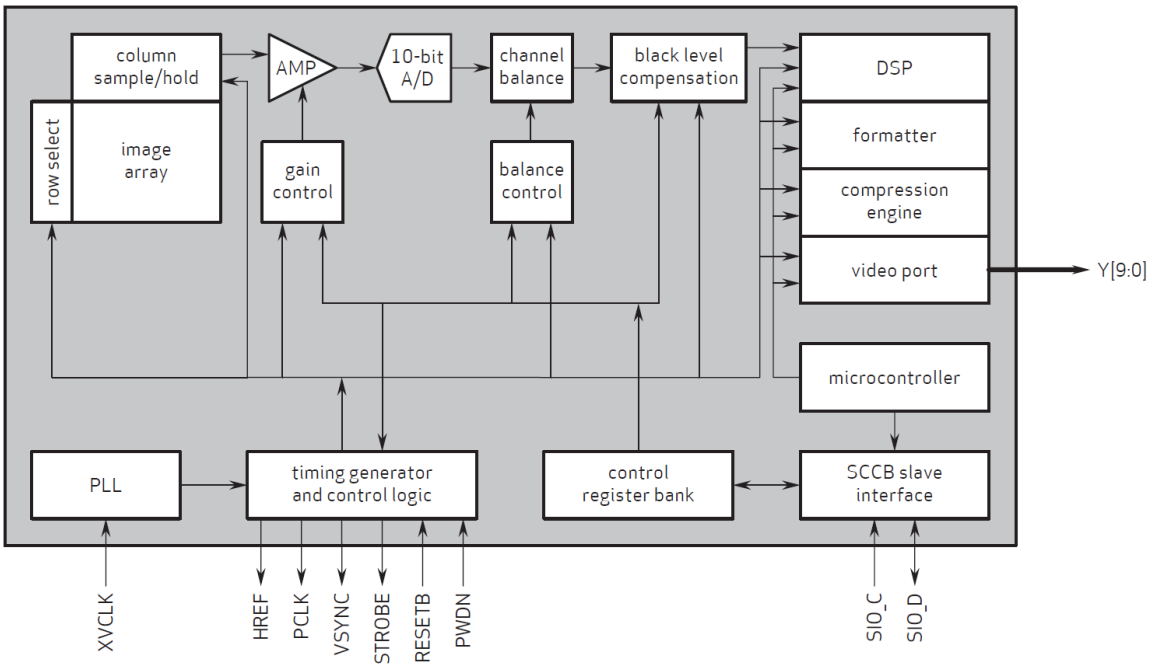

图 1.1 OV2640 功能框图

OV2640 传感器包括如下一些功能模块。

1.感光整列（Image Array）

OV2640 总共有 1632*1232 个像素，最大输出尺寸为 UXGA（1600*1200），即 200W 像素。

2.模拟信号处理（Analog Processing）

模拟信号处理所有模拟功能，并包括：模拟放大（AMP）、增益控制、通道平衡和平衡控制等。

3.10 位 A/D 转换（A/D）

原始的信号经过模拟放大后，分 G 和 BR 两路进入一个 10 位的 A/D 转换器，A/D 转换器工作频率高达 20M，与像素频率完全同步（转换的频率和帧率有关）。除 A/D 转换器外，该模块还有黑电平校正（BLC)功能。

4.数字信号处理器（DSP）

这个部分控制由原始信号插值到 RGB 信号的过程，并控制一些图像质量：

- 边缘锐化（二维高通滤波器）
- 颜色空间转换（原始信号到 RGB 或者 YUV/YCbYCr)
- RGB 色彩矩阵以消除串扰
- 色相和饱和度的控制
- 黑/白点补偿
- 降噪
- 镜头补偿
- 可编程的伽玛
- 十位到八位数据转换

5.输出格式模块（Output Formatter）

该模块按设定优先级控制图像的所有输出数据及其格式。

6.压缩引擎（Compression Engine）

压缩引擎框图如图 1.2 所示：

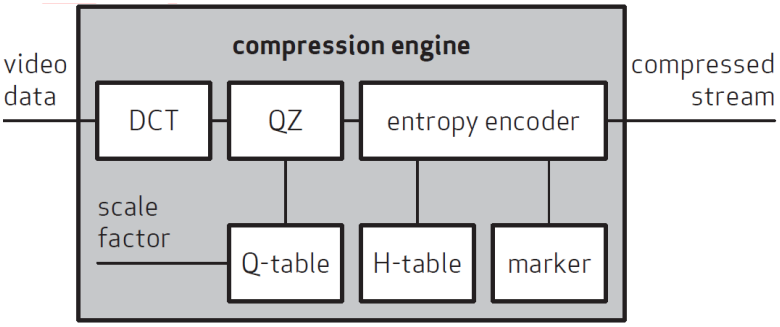

图 1.2 压缩引擎框图

从图可以看出，压缩引擎主要包括三部分：DCT、QZ 和 entropy encoder（熵编码器），将原始的数据流，压缩成 jpeg 数据输出。

7.微处理器（Microcontroller）

OV2640 自带了一个 8 位微处理器，该处理器有 512 字节 SRAM，4KB 的 ROM，它提供一个灵活的主机到控制系统的指令接口，同时也具有细调图像质量的功能。

8.SCCB 接口（SCCB Interface）

SCCB 接口控制图像传感器芯片的运行，详细使用方法参照光盘的《OmniVision Technologies Seril Camera Control Bus(SCCB) Specification》这个文档

9.数字视频接口（Digital Video Port）

OV2640 拥有一个 10 位数字视频接口(支持 8 位接法)，其 MSB 和 LSB 可以程序设置先后顺序，ALIENTEK OV2640 模块采用默认的 8 位连接方式，如图 1.3 所示：

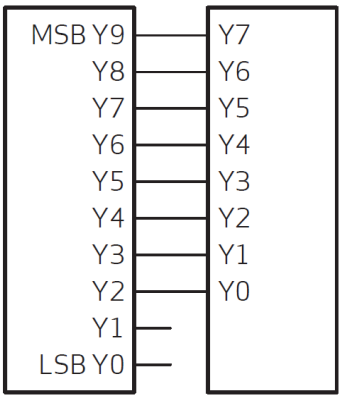

图 1.3 OV2640 默认 8 位连接方式

OV2640 的寄存器通过 SCCB 时序访问并设置，SCCB 时序和 IIC 时序十分类似，在本章我们不做介绍，请大家参考光盘《OmniVision Technologies Seril Camera Control Bus(SCCB) Specification》这个文档。

接下来，我们介绍一下 OV2640 的传感器窗口设置、图像尺寸设置、图像窗口设置和图像输出大小设置，这几个设置与我们的正常使用密切相关，有必要了解一下。其中，除了传感器窗口设置是直接针对传感器阵列的设置，其他都是 DSP 部分的设置了，接下来我们一个个介绍。

传感器窗口设置，该功能允许用户设置整个传感器区域（1632*1220）的感兴趣部分，也就是在传感器里面开窗，开窗范围从 2*2~1632*1220 都可以设置，不过要求这个窗口必须大于等于随后设置的图像尺寸。传感器窗口设置，通过：0X03/0X19/0X1A/0X07/0X17/0X18 等寄存器设置，寄存器定义请看 OV2640_DS(1.6).pdf 这个文档（下同）。

图像尺寸设置，也就是 DSP 输出（最终输出到 LCD 的）图像的最大尺寸，该尺寸要小于等于前面我们传感器窗口设置所设定的窗口尺寸。图像尺寸通过：0XC0/0XC1/0X8C 等寄

存器设置。

图像窗口设置，这里起始和前面的传感器窗口设置类似，只是这个窗口是在我们前面设置的图像尺寸里面，再一次设置窗口大小，该窗口必须小于等于前面设置的图像尺寸。该窗口设置后的图像范围，将用于输出到外部。图像窗口设置通过：0X51/0X52/0X53/0X54/0X55/0X57 等寄存器设置。

图像输出大小设置，这是最终输出到外部的图像尺寸。该设置将图像窗口设置所决定的窗口大小，通过内部 DSP 处理，缩放成我们输出到外部的图像大小。该设置将会对图像进行缩放处理，如果设置的图像输出大小不等于图像窗口设置图像大小，那么图像就会被缩放处理，只有这两者设置一样大的时候，输出比例才是 1：1 的。

因为 OmniVision 公司公开的文档，对这些设置实在是没有详细介绍。只能从他们提供的初始化代码（还得去 linux 源码里面移植过来）里面去分析规律，所以，这几个设置，都是作者根据 OV2640 的调试经验，以及相关文档总结出来的，不保证百分比正确，如有错误，还请大家指正。

以上几个设置，光看文字可能不太清楚，这里我们画一个简图有助于大家理解，如图 1.4 所示：

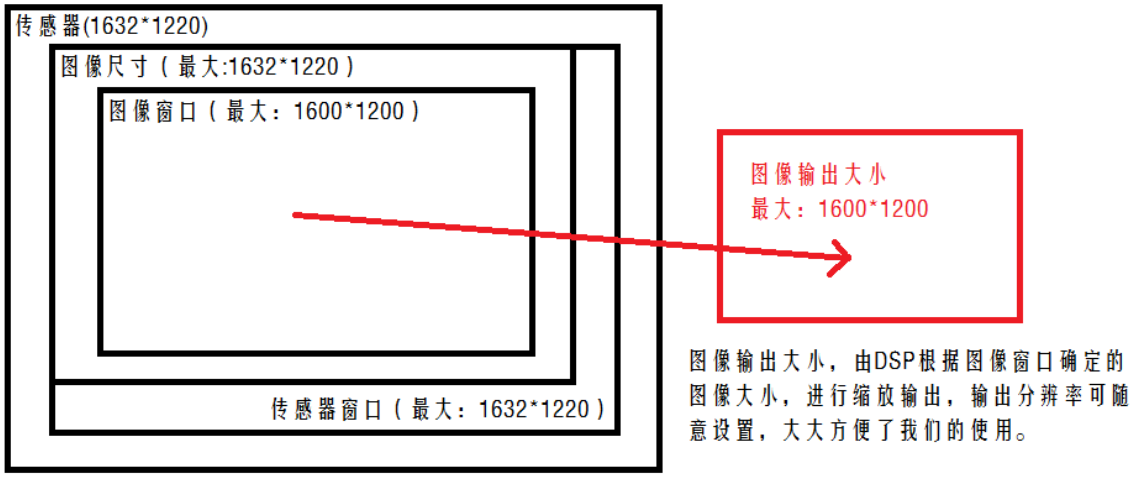

图 1.4 OV2640 图像窗口设置简图

上图，最终红色框所示的图像输出大小，才是 OV2640 输出给外部的图像尺寸，也就是显示在 LCD 上面的图像大小。当图像输出大小与图像窗口不等时，会进行缩放处理，在 LCD 上面看到的图像将会变形。

最后，我们介绍一下 OV2640 的图像数据输出格式。首先我们简单介绍一些定义：

UXGA，即分辨率位 1600*1200 的输出格式，类似的还有：SXGA(1280*1024)、WXGA+(1440*900)、XVGA(1280*960)、WXGA(1280*800)、XGA(1024*768)、SVGA(800*600)、VGA(640*480)、CIF(352*288)、WQVGA(400*240)、QCIF(176*144)和 QQVGA(160*120)等。

PCLK，即像素时钟，一个 PCLK 时钟，输出一个像素(或半个像素)。

VSYNC，即帧同步信号。

HREF，即行参考信号。

HSYNC，即行同步信号。

OV2640 的图像数据输出（通过 Y[9:0]）就是在 PCLK，VSYNC 和 HREF/ HSYNC 的控制下进行的。首先看看行输出时序，如图 1.5 所示：

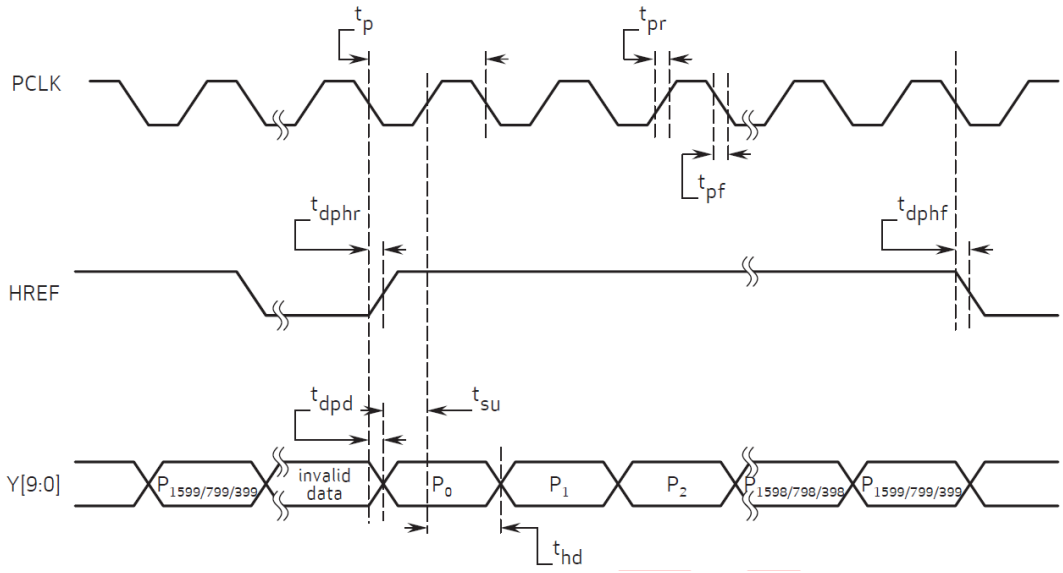

图 1.5 OV2640 行输出时序

从上图可以看出，图像数据在 HREF 为高的时候输出，当 HREF 变高后，每一个 PCLK 时钟，输出一个 8 位/10 位数据。我们采用 8 位接口，所以每个 PCLK 输出 1 个字节，且在 RGB/YUV 输出格式下，每个 tp=2 个 Tpclk，如果是 Raw 格式，则一个 tp=1 个 Tpclk。比如我们采用 UXGA 时序，RGB565 格式输出，每 2 个字节组成一个像素的颜色（高低字节顺序可通过 0XDA 寄存器设置），这样每行输出总共有 1600*2 个 PCLK 周期，输出 1600*2 个字节。

再来看看帧时序（UXGA 模式），如图 1.6 所示：

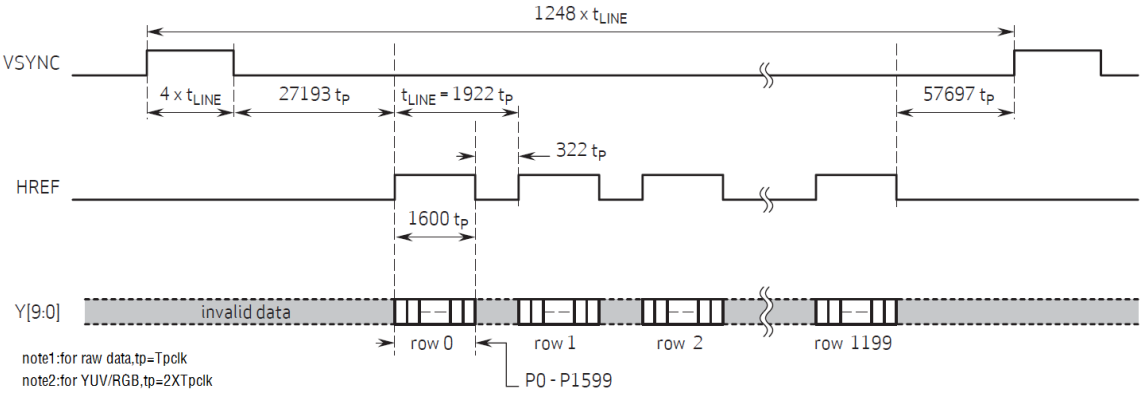

图 1.6 OV2640 帧时序

上图清楚的表示了 OV2640 在 UXGA 模式下的数据输出。我们按照这个时序去读取 OV2640 的数据，就可以得到图像数据。

最后说一下 OV2640 的图像数据格式，我们一般用 2 种输出方式：RGB565 和 JPEG。当输出 RGB565 格式数据的时候，时序完全就是上面两幅图介绍的关系。以满足不同需要。而当输出数据是 JPEG 数据的时候，同样也是这种方式输出（所以数据读取方法一模一样），不过 PCLK 数目大大减少了，且不连续，输出的数据是压缩后的 JPEG 数据，输出的 JPEG 数据以：0XFF,0XD8 开头，以 0XFF,0XD9 结尾，且在 0XFF,0XD8 之前，或者 0XFF,0XD9 之后，会有不定数量的其他数据存在（一般是 0），这些数据我们直接忽略即可，将得到的 0XFF,0XD8~0XFF,0XD9 之间的数据，保存为.jpg/.jpeg 文件，就可以直接在电脑上打开看到图像了。

OV2640 自带的 JPEG 输出功能，大大减少了图像的数据量，使得其在网络摄像头、无

线视频传输等方面具有很大的优势。

接下来我们介绍一下 ALIENTEK ATK-OV2640 摄像头模块。该模块的外观如图 1.7：

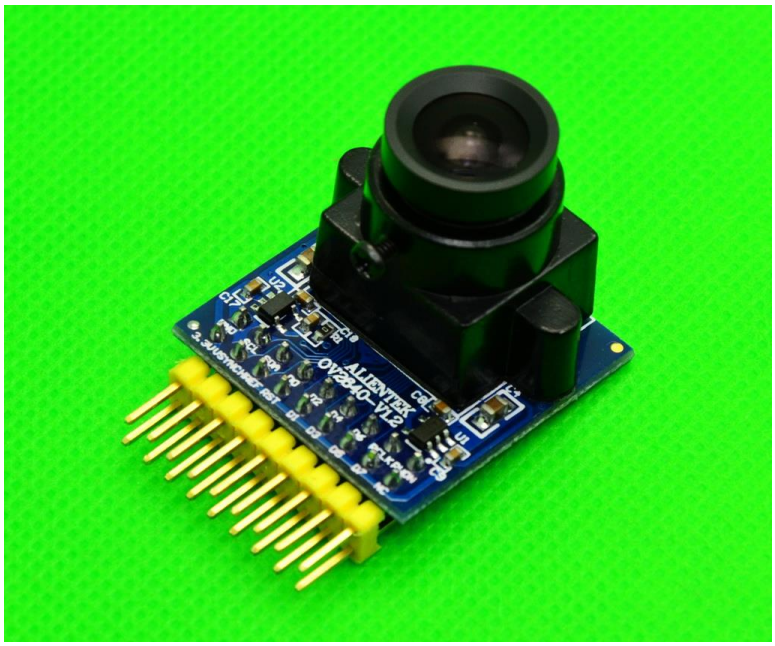

图 1.7 ATK-OV2640 摄像头模块外观图

ATK-OV2640 摄像头模块原理图如图 1.8 所示：

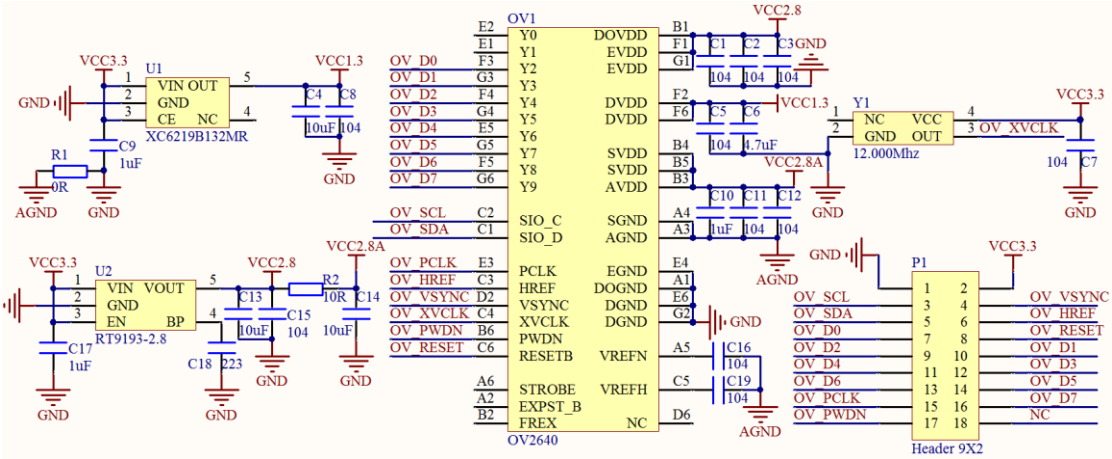

图 1.8 ATK-OV2640 摄像头模块原理图

从上图可以看出，ATK-OV2640 摄像头模块自带了有源晶振，用于产生 12M 时钟作为 OV2640 的 XVCLK 输入。同时自带了稳压芯片，用于提供 OV2640 稳定的 2.8V 和 1.3V 两个电压。模块通过一个 2*9 的双排排针（P1）与外部通信，与外部的通信信号如表 1.1 所示：

信号	作用描述	信号	作用描述
VCC3.3	模块供电脚，接 3.3V 电源	OV_HREF	行参考信号
GND	模块地线	OV_RESET	复位信号
OV_SCL	SCCB 通信时钟信号	OV_PCLK	像素时钟
OV_SDA	SCCB 通信数据信号	OV_PWDN	掉电模式控制
OV_D[7:0]	OV2640 输出数据口（8 位）	OV_VSYNC	帧同步信号

表 1.1 OV2640 模块信号及其作用描述

下面我们来看看如何使用 ALIENTEK ATK-OV2640 摄像头模块。对于该模块，我们只

关心两点：1，如何读取 RGB565 图像数据；2，如何读取 JPEG 压缩图像数据。

先了解 3 个前提：

1，本例程，我们将设置 VSYNC 为高电平有效，即 VSYNC 低电平的时候，输出图像数据，高电平的时候，做帧同步信号。

2，本例程，我们将设置 HREF 为高电平有效，即 HREF 为高的时候，数据有效，HREF 为低的时候，数据无效。

3，本例程，我们将配置 PCLK 的下降沿更新数据，即 PCLK 下降沿的时候，OV2640 更新数据到 OV_D0~D7，所以，MCU 在 PCLK 的上升沿读取数据。

首先，我们来看如何读取 RGB565 图像数据。

根据前面的 3 个前提，我们可以很容易想到读取数据的方法，因为 OV2640 的数据输出，是很有规律的，我们以行为单位，每输出一行数据，即一个 HREF 周期。OV2640 输出一行 RGB565 数据的读取过程如下：等待 VSYNC 为低电平→等待 HREF 为高电平→等待第 1 个 PCLK 的上升沿→读取第 1 个像素的低字节→等待第 2 个 PCLK 的上升沿→读取第 1 个像素的高字节→等待第 3 个 PCLK 的上升沿→读取低 2 个像素的低字节→等待第 4 个 PCLK 的上升沿……→读取 1 行数据最后 1 个像素的高字节→完成 1 行数据读取→等待 HREF 为高电平……

上面，红色部分，循环 N 次，就可以读取 N 行的数据，直到读取完整个一帧图像的输出，最后 VSYNC 变成高电平，完成一帧图像数据的读取。当 VSYNC 再次变低的时候，开始下一帧数据输出循环。

一次数据输出多少行，每一行多少个像素，则是根据我们设定的图像输出大小（分辨率）来确定的，假定我们设定图像输出大小为：320*480，那么每一行就有 640 个 PCLK 上升沿，每一帧图像就有 480 行图像数据。

然后，我们来看看如何读取 JPEG 压缩后的图像数据。

同样根据前面的 3 个前提，不过**在 JPEG 输出的时候，VSYNC 变成了：高电平时输出图像数据，低电平时数据无效**，这个要特别注意下。

OV2640 输出 JPEG 数据的读取过程如下：等待 VSYNC 为高电平→等待 HREF 为高电平→等待 PCLK 的上升沿→读取 JPEG 数据→等待 HREF 为高电平→等待 PCLK 的上升沿→读取 JPEG 数据→等待 HREF 为高电平……→VSYNC 为低电平→完成一帧 JPEG 数据读取。

从上面的流程可以看出，读取 JPEG 数据，相对读取 RGB565 数据来说，简单不少，只需要在 HREF 为高时，在 PCLK 的上升沿读取数据即可，直到 VSYNC 为低电平，完成一帧 JPEG 数据采集。

不过，需要注意的是：HREF 在一帧 JPEG 数据输出的时候，并不是持续的高电平，所以任何时候，都要先判断 HREF 为高电平，再去读取数据。

以上，就是 RGB565 输出以及 JPEG 输出时，OV2640 的图像数据读取方法。不过特别注意：OV2640 的 PCLK 速度可以很快（最高达 36Mhz），所以，代码一定要尽可能优化，而且对 PCLK 也要进行分频，才可能用在 STM32F103 这类不带 DCMI 接口的 MCU 上面。

2、硬件连接

本实验将实现如下功能：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 OV2640，在初始化成功之后，就一直在屏幕显示 OV2640 拍到的内容。当按下 KEY0 按键的时候，即进行 bmp 拍照（分辨率为：LCD 辨率）。按下 KEY1 可以拍 JPEG 图片照片（分辨率为 VGA，即 1024*768）。拍照时，DS1 会亮，在拍照保存成功之后，蜂鸣器会发出“滴”的一声，提示拍照成功，同时 DS1

灭。另外，可以借助 USMART 设置 OV2640 的寄存器，方便大家调试。DS0 还是用于指示程序运行状态。

本实验用到的硬件资源有：

1） 指示灯 DS0

2） KEY0 和 KEY1 两个按键

3） 串口 1

4） TFTLCD 模块

5） ATK-OV2640 摄像头模块

ALIENTEK ATK-OV2640 摄像头模块在上一节已经有详细介绍过，这里我们主要介绍该模块与 ALIETEK 战舰 V3 / 精英 STM32F103 开发板的连接。

在开发板的左下角的 2*9 的排座（战舰 V3 是：P6 / 精英是：P4），是摄像头模块/OLED 模块共用接口。我们只需要将 ALIENTEK ATK-OV2640 摄像头模块插入这个接口（P6/P4）即可，该接口与 STM32 的连接关系（以战舰板 V3 为例，精英板的连接关系也是完全一样的）如图 2.1 所示：

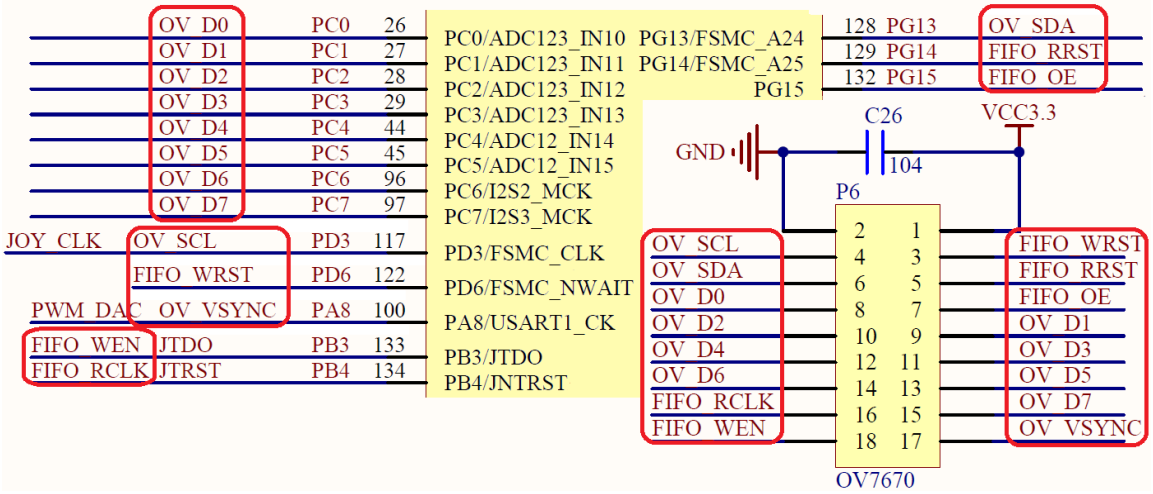

图 2.1 摄像头模块接口与 STM32 连接图

从上图可以看出，摄像头模块接口的各信号脚与 STM32 的连接关系为：

OV_SDA 接 PG13；

OV_SCL 接 PD3；

FIFO_RCLK 接 PB4；

FIFO_WEN 接 PB3；

FIFO_WRST 接 PD6；

FIFO_RRST 接 PG14；

FIFO_OE 接 PG15；

OV_VSYNC 接 PA8；

OV_D[7:0]接 PC[7:0]；

这些线的连接，在开发板上直接就已经连接好了，我们只需要将 OV2640 摄像头模块插上去就好了。注意，这些信号线里面，有几个信号线和其他外设共用了，OV_SCL 与 JOY_CLK 共用 PD3，所以摄像头和手柄不可以同时使用；FIFO_WEN 和 FIFO_RCLK 则和 JTAG 的信号线 JTDO 和 JTRST 共用了，所以使用摄像头的时候，不能使用 JTAG 模式调试，而应该选择 SW 模式（SW 模式不需要用到 JTDO 和 JTRST）；OV_VSYNC 和 PWM_DAC 共用了 PA8，所以他们也不可以同时使用。

当 ATK-OV2640 摄像头模块插入开发板的摄像头模块接口后，模块的信号线同 STM32

的 IO 连接关系为：

OV_D0~D7 接 PC0~7
OV_SCL 接 PD3
OV_SDA 接 PG13
OV_VSYNC 接 PD6
OV_PWDN 接 PB3
OV_RST 接 PG15
OV_HREF 接 PG14
OV_PCLK 接 PB4

ATK-OV2640 摄像头模块与开发板接好后，实物连接图如图 2.2 和图 2.3 所示：

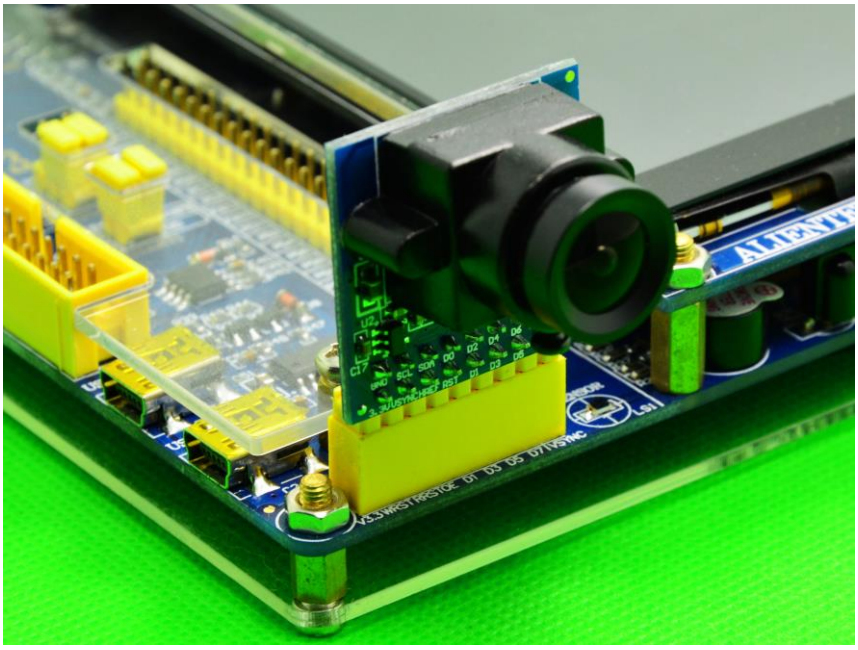

图 2.2 ATK-OV2640 摄像头模块连接战舰 V3 开发板

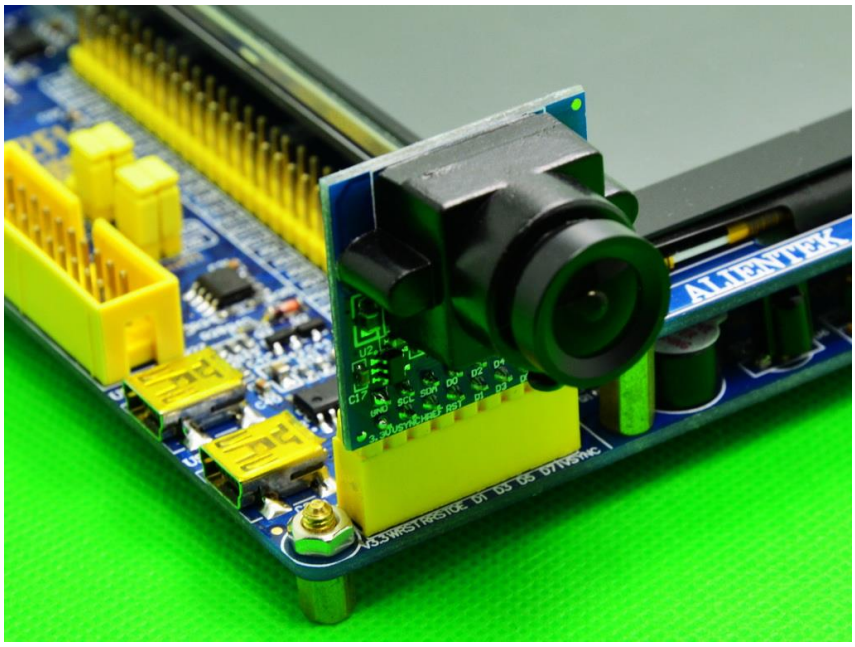

图 2.3 ATK-OV2640 摄像头模块连接精英开发板

3、软件实现

本扩展例程（扩展实验 6 ATK-OV2640 摄像头模块测试实验），我们在战舰 V3 的照相机实验基础上进行修改。具体修改细节，我们这里就不详细介绍了，请大家参考本例程源码即可。这里，我们直接给出本例程的工程结构，如图 3.1 所示：

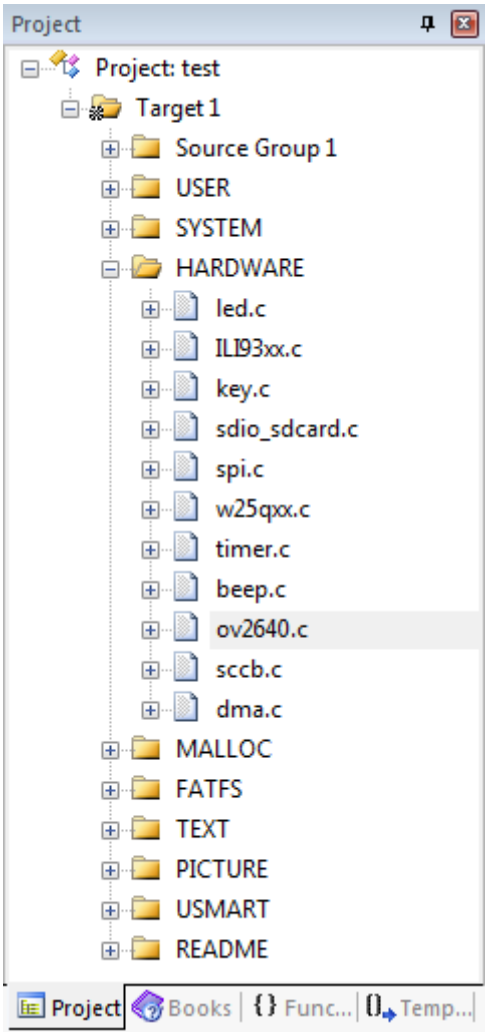

图 3.1 ATK-OV2640 摄像头模块测试实验工程结构

由于代码比较多，这里我们就不给大家详细介绍所有代码了，仅对一些重要函数进行介绍，其他的请大家参考本例程源码进行理解。

首先，我们来看 ov2640.c 里面的 OV2640_Init 函数，该函数代码如下：

```
//初始化 OV2640
//配置完以后,默认输出是 1600*1200 尺寸的图片!!
//返回值:0,成功
//      其他,错误代码
u8 OV2640_Init(void)
{
    u16 i=0;
    u16 reg;
    //设置 IO
    RCC->APB2ENR|=1<<3;          //先使能外设 PORTB 时钟
    RCC->APB2ENR|=1<<4;          //先使能外设 PORTC 时钟
```

```
RCC->APB2ENR|=1<<5;             //先使能外设 PORTD 时钟
RCC->APB2ENR|=1<<8;             //先使能外设 PORTG 时钟
GPIOB->CRL&=0XFFF00FFF;
GPIOB->CRL|=0X00083000;         //PB3 输出,PB4 输入
GPIOB->ODR|=3<<3;
GPIOC->CRL=0X88888888;          //PC0~7  输入
GPIOC->ODR|=0x00ff;
GPIOD->CRL&=0XF0FFFFFF;         //PD6 输入
GPIOD->CRL|=0X08000000;
GPIOD->ODR|=1<<6;
GPIOG->CRH&=0X00FFFFFF;
GPIOG->CRH|=0X38000000;
GPIOG->ODR=3<<14;               //PG14 输入，PG15 输出
JTAG_Set(SWD_ENABLE);
OV2640_PWDN=0;                  //POWER ON
delay_ms(10);
OV2640_RST=0;                   //复位 OV2640
delay_ms(10);
OV2640_RST=1;                   //结束复位
SCCB_Init();                    //初始化 SCCB  的 IO 口
SCCB_WR_Reg(OV2640_DSP_RA_DLMT, 0x01);   //操作 sensor 寄存器
SCCB_WR_Reg(OV2640_SENSOR_COM7, 0x80);   //软复位 OV2640
delay_ms(50);
reg=SCCB_RD_Reg(OV2640_SENSOR_MIDH);     //读取厂家 ID  高八位
reg<<=8;
reg|=SCCB_RD_Reg(OV2640_SENSOR_MIDL);    //读取厂家 ID  低八位
if(reg!=OV2640_MID)
{
    printf("MID:%d\r\n",reg);
    return 1;
}
reg=SCCB_RD_Reg(OV2640_SENSOR_PIDH);     //读取厂家 ID  高八位
reg<<=8;
reg|=SCCB_RD_Reg(OV2640_SENSOR_PIDL);    //读取厂家 ID  低八位
if(reg!=OV2640_PID)
{
    printf("HID:%d\r\n",reg);
    //return 2;
}
//初始化  OV2640,采用 SXGA 分辨率(1600*1200)
for(i=0;i<sizeof(ov2640_uxga_init_reg_tbl)/2;i++)
{
    SCCB_WR_Reg(ov2640_uxga_init_reg_tbl[i][0],ov2640_uxga_init_reg_tbl[i][1]);
}
```

```
        return 0x00;    //ok
    }
```

此部分代码先初始化 OV2640 相关的 IO 口（包括 SCCB_Init），然后最主要的是完成 OV2640 的寄存器序列初始化。OV2640 的寄存器特多（百几十个），配置特麻烦，幸好厂家有提供参考配置序列（详见《OV2640 Software Application Notes 1.03.pdf》），本章我们用到的配置序列，存放在 ov2640_uxga_init_reg_tbl 这个数组里面，该数组是一个 2 维数组，存储初始化序列寄存器及其对应的值，该数组存放在 ov2640cfg.h 里面。

另外，在 ov2640.c 里面，还有几个函数比较重要，这里贴代码了，只介绍功能：

OV2640_Window_Set 函数，该函数用于设置传感器输出窗口；

OV2640_ImageSize_Set 函数，用于设置图像尺寸；

OV2640_ImageWin_Set 函数，用于设置图像窗口大小；

OV2640_OutSize_Set 函数，用于设置图像输出大小；

这就是我们在第 1 节所介绍的 4 个设置，他们共同决定了图像的输出。接下来，我们看看 ov2640cfg.h 里面 ov2640_uxga_init_reg_tbl 的内容，ov2640cfg.h 文件的代码如下：

```
//OV2640 UXGA 初始化寄存器序列表
//此模式下帧率为 15 帧
//UXGA(1600*1200)
const u8 ov2640_uxga_init_reg_tbl[][2]=
{
    0xff, 0x00,
    ……//省略部分代码
    0x05, 0x00,
};
//OV2640 SVGA 初始化寄存器序列表
//此模式下,帧率可以达到 30 帧
//SVGA 800*600
const u8 ov2640_svga_init_reg_tbl[][2]=
{
    0xff, 0x00,
    ……//省略部分代码
    0x05, 0x00,
};
const u8 ov2640_yuv422_reg_tbl[][2]=
{
    0xFF, 0x00,
    ……//省略部分代码
    0x00, 0x00,
};
const u8 ov2640_jpeg_reg_tbl[][2]=
{
    0xff, 0x01,
    ……//省略部分代码
    0xe0, 0x00,
};
```

```
const u8 ov2640_rgb565_reg_tbl[][2]=
{
    0xFF, 0x00,
    ……//省略部分代码
    0xe0, 0x00,
};
```

以上代码，我们省略了很多（全部贴出来太长了），里面总共有 5 个数组。我们大概了解下数组结构，每个数组条目的第一个字节为寄存器号（也就是寄存器地址），第二个字节为要设置的值，比如{0xff, 0x01}，就表示在 0Xff 地址，写入 0X01 这个值。

五个数组里面 ov2640_uxga_init_reg_tbl 和 ov2640_svga_init_reg_tbl，分别用于配置 OV2640 输出 UXGA 和 SVGA 分辨率的图像，我们只用了 ov2640_uxga_init_reg_tbl 这个数组，完成对 OV2640 的初始化(设置为 UXGA)。最后 OV2640 要输出数据是 RGB565 还是 JPEG，就得通过其他数组设置，输出 RGB565 时，通过一个数组：ov2640_rgb565_reg_tbl 设置即可；输出 JPEG 时，则要通过 ov2640_yuv422_reg_tbl 和 ov2640_jpeg_reg_tbl 两个数组设置。

接下来，我们看看 dma.c 里面的两个函数：MYDMA_SRAMLCD_Init 和 MYDMA_SRAMLCD_Enable。为了提高 RGB565 图像数据的效率，本例程采用 DMA 的方式，将图像数据传输给 LCD，以尽量提高显示帧率。通过前面的学习，我们知道，要让 LCD 直接显示 OV2640 的图像，必须用 RGB565 格式输出，同时，我们又知道，OV2640 的输出，在每一行图像输出完毕后，有一个空挡（HREF 为低电平），这样，我们便可以采集完一行图像数据后，在 HREF 为低电平的这段时间内，设置 DMA，通过 DMA 将数据发送给 LCD，而不需要每采集一个点，就写一次 LCD，这样大大提高了采集效率。

这两个函数的代码如下：

```
//SRAM --> LCD_RAM dma 配置
//caddr：数据源地址
//16 位,从 SRAM 传输到 LCD_RAM.
void MYDMA_SRAMLCD_Init(u32 caddr)
{
    RCC->AHBENR|=1<<1;              //开启 DMA2 时钟
    DMA2_Channel5->CPAR=caddr;      //DMA2，源存储器地址
    DMA2_Channel5->CMAR=(u32)&LCD->LCD_RAM;//目标地址为 LCD_RAM
    DMA2_Channel5->CNDTR=0;         //DMA2,传输数据量，暂定为 0
    DMA2_Channel5->CCR=0X00000000;//复位
    DMA2_Channel5->CCR|=0<<4;       //从外设读
    DMA2_Channel5->CCR|=0<<5;       //普通模式
    DMA2_Channel5->CCR|=1<<6;       //外设地址增量模式
    DMA2_Channel5->CCR|=0<<7;       //存储器非增量模式
    DMA2_Channel5->CCR|=1<<8;       //外设数据宽度为 16 位
    DMA2_Channel5->CCR|=1<<10;      //存储器数据宽度 16 位
    DMA2_Channel5->CCR|=1<<12;      //中等优先级
    DMA2_Channel5->CCR|=1<<14;      //存储器到存储器模式(不需要外部请求)
}
//开启一次 SRAM 到 LCD 的 DMA 的传输
void MYDMA_SRAMLCD_Enable(void)
```

```
{
    DMA2_Channel5->CCR&=~(1<<0);          //关闭 DMA 传输
    DMA2_Channel5->CNDTR=lcddev.width;  //设置传输长度
    DMA2_Channel5->CCR|=1<<0;             //开启 DMA 传输
}
```

其中，MYDMA_SRAMLCD_Init 函数，用于设置 DMA，这里采用 DMA2 通道 5，实现存储器到存储器的数据传输，源地址为：caddr，通过该函数的参数传递进来，而目的地址为：LCD->LCD_RAM 的地址，也就是 LCD 的 GRAM。

而 MYDMA_SRAMLCD_Enable 函数，则用于启动一次数据传输，传输数量等于 LCD 的宽度，因为我们每次从 OV2640 采集的数据，以行为单位，然后我们设置 OV2640 的图像输出尺寸等于 LCD 的分辨率，这样输出的行像素就等于 LCD 的宽度，所以 DMA 每次传输数据长度就是 LCD 的宽度（lcddev.width）。

我们只需要在每一行图像数据采集完成后，调用 MYDMA_SRAMLCD_Enable 函数，即可启动一次 DMA 传输，然后就可以开始采集下一行数据了，从而提高图像采集效率。

最好，我们看 test.c 里面的代码，如下：

```
#define OV2640_JPEG_WIDTH  1024      //JPEG 拍照的宽度
#define OV2640_JPEG_HEIGHT 768       //JPEG 拍照的高度
u8* ov2640_framebuf;                 //帧缓存
extern u8 ov_frame;                  //在 timer.c 里面定义
//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++;
    }
}
//OV2640 速度控制
//根据 LCD 分辨率的不同，设置不同的参数
void ov2640_speed_ctrl(void)
{
    u8 clkdiv,pclkdiv;              //时钟分频系数和 PCLK 分频系数
    if(lcddev.width==240){ clkdiv=1; pclkdiv=28;}       //2.8 寸 LCD
    else if(lcddev.width==320){ clkdiv=3; pclkdiv=15;}  //3.5 寸 LCD
    else{ clkdiv=15; pclkdiv=4;}                        //4.3/7 寸 LCD
```

```
        SCCB_WR_Reg(0XFF,0X00);
        SCCB_WR_Reg(0XD3,pclkdiv);  //设置 PCLK 分频
        SCCB_WR_Reg(0XFF,0X01);
        SCCB_WR_Reg(0X11,clkdiv);    //设置 CLK 分频
}
//OV2640 拍照 jpg 图片
//pname:要保存的 jpg 照片路径+名字
//返回值:0,成功
//      其他,错误代码
u8 ov2640_jpg_photo(u8 *pname)
{
        FIL* f_jpg;
        u8 res=0;
        u32 bwr;
        u32 i=0;
        u32 jpeglen=0;
        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.
        OV2640_JPEG_Mode();                                 //切换为 JPEG 模式
        OV2640_OutSize_Set(OV2640_JPEG_WIDTH,OV2640_JPEG_HEIGHT);
        SCCB_WR_Reg(0XFF,0X00);
        SCCB_WR_Reg(0XD3,30);
        SCCB_WR_Reg(0XFF,0X01);
        SCCB_WR_Reg(0X11,0X1);
        for(i=0;i<10;i++)           //丢弃 10 帧,等待 OV2640 自动调节好(曝光白平衡之类的)
        {
                while(OV2640_VSYNC==1);
                while(OV2640_VSYNC==0);
        }
        while(OV2640_VSYNC==1)//开始采集 jpeg 数据
        {
                while(OV2640_HREF)
                {
                        while(OV2640_PCLK==0);
                        ov2640_framebuf[jpeglen]=OV2640_DATA;
                        while(OV2640_PCLK==1);
                        jpeglen++;
                }
        }
        res=f_open(f_jpg,(const TCHAR*)pname,FA_WRITE|FA_CREATE_NEW);
        //尝试打开失败,则创建新文件
        if(res==0)
        {
```

```
            printf("jpeg data size:%d\r\n",jpeglen); //串口打印 JPEG 文件大小
            pbuf=(u8*)ov2640_framebuf;
            for(i=0;i<jpeglen;i++)//查找 0XFF,0XD8
            {
                if((pbuf[i]==0XFF)&&(pbuf[i+1]==0XD8))break;
            }
            if(i==jpeglen)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,jpeglen-i,&bwr);
                if(bwr!=(jpeglen-i))res=0XFE;
            }
        }
        f_close(f_jpg);
        OV2640_RGB565_Mode();  //RGB565 模式
        myfree(SRAMIN,f_jpg);
        return res;
}
int main(void)
{
        u8 res;
        u8 *pname;                  //带路径的文件名
        u8 key;                     //键值
        u8 sd_ok=1;                 //0,sd 卡不正常;1,SD 卡正常.
        u16 pixcnt=0;               //像素统计
        u16 linecnt=0;              //行数统计
        Stm32_Clock_Init(9);        //系统时钟设置
        uart_init(72,115200);       //串口初始化为 115200
        delay_init(72);             //延时初始化
        usmart_dev.init(72);        //初始化 USMART
        LED_Init();                 //初始化与 LED 连接的硬件接口
        KEY_Init();                 //初始化按键
        LCD_Init();                 //初始化 LCD
        BEEP_Init();                //蜂鸣器初始化
        W25QXX_Init();              //初始化 W25Q128
        my_mem_init(SRAMIN);        //初始化内部内存池
        exfuns_init();              //为 fatfs 相关变量申请内存
        f_mount(fs[0],"0:",1);      //挂载 SD 卡
        f_mount(fs[1],"1:",1);      //挂载 FLASH.
        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
    }
    Show_Str(30,50,200,16,"STM32F103 开发板",16,0);
    Show_Str(30,70,200,16,"OV2640 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,"2015 年 4 月 16 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(30,150,240,16,"SD 卡错误,无法拍照!",16,0);
        sd_ok=0;
    }
    ov2640_framebuf=mymalloc(SRAMIN,52*1024);//申请帧缓存
    pname=mymalloc(SRAMIN,30);  //为带路径的文件名分配 30 个字节的内存
    while(!pname||!ov2640_framebuf) //内存分配出错
    {
        Show_Str(30,150,240,16,"内存分配失败!",16,0); delay_ms(200);
        LCD_Fill(30,150,240,146,WHITE); delay_ms(200);   //清除显示
    }
    while(OV2640_Init())                //初始化 OV2640
    {
        Show_Str(30,150,240,16,"OV2640 错误!",16,0); delay_ms(200);
        LCD_Fill(30,150,239,206,WHITE); delay_ms(200);
    }
    Show_Str(30,170,200,16,"OV2640 正常",16,0);
    delay_ms(1500);
    //TIM6_Int_Init(10000,7199);      //10Khz 计数频率,1 秒钟中断,屏蔽则不打印帧率
    OV2640_RGB565_Mode();         //RGB565 模式
    OV2640_OutSize_Set(lcddev.width,lcddev.height);
    ov2640_speed_ctrl();
    MYDMA_SRAMLCD_Init((u32)ov2640_framebuf);
    while(1)
    {
        while(OV2640_VSYNC)      //等待帧信号
        {
            key=KEY_Scan(0);      //不支持连按
            if(key==KEY0_PRES||key==KEY1_PRES)
            {
                LED1=0;             //DS1 亮
                if(sd_ok)           //SD 卡正常才可以拍照
                {
                    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);
            }else if(key==KEY1_PRES)                   //JPG 拍照
            {
                camera_new_pathname(pname,1);          //得到文件名
                res=ov2640_jpg_photo(pname);
                OV2640_OutSize_Set(lcddev.width,lcddev.height);
                ov2640_speed_ctrl();
            }
            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);
            }
        }else                                          //提示 SD 卡错误
        {
            Show_Str(40,130,240,12,"SD 卡错误!",12,0);
            Show_Str(40,150,240,12,"拍照功能不可用!",12,0);
        }
        LED1=1;                                        //DS1 灭
        BEEP=0;                                        //关闭蜂鸣器
        delay_ms(1800);
    }
}
LCD_SetCursor(0,0);          //设置坐标
LCD_WriteRAM_Prepare();      //开始写入 GRAM
linecnt=0;                   //行统计清零
pixcnt=0;                    //像素计数器清零
while(linecnt<lcddev.height)
{
    while(OV2640_HREF)
    {
        while(OV2640_PCLK==0);
        ov2640_framebuf[pixcnt++]=OV2640_DATA;
        while(OV2640_PCLK==1);
        while(OV2640_PCLK==0);
        ov2640_framebuf[pixcnt++]=OV2640_DATA;
        while(OV2640_PCLK==1);
```

```
                }
                if(pixcnt)
                {
                    MYDMA_SRAMLCD_Enable(); //启动 DMA 数据传输
                    pixcnt=0;
                    linecnt++;
                }
            }
            ov_frame++;
            LED0=!LED0;
    }
```

这里总共有 4 个函数。首先是：camera_new_pathname 函数，用于创建带路径的文件名。该函数可以确保所创建的文件名都是新的（不会覆盖旧的文件），在 bmp/jpg 拍照的时候需要先调用该函数，获取一个新的文件名。

然后，ov2640_speed_ctrl 函数，用于设置 OV2640 的 PCLK 的频率。因为我们用的是 STM32F103 直接驱动 OV2640，而 STM32 的 IO 速度又比较慢，当 PCLK 过快的时候，会采集不过来，导致数据丢失，所以，这里我们通过该函数，针对不同尺寸的 LCD，对 OV2640 进行不同的配置，确保可以正常采集 OV2640 输出的数据。

接着，ov2640_jpg_photo 函数，用于 JPG 拍照。进入该函数，我们先设置 OV2640 为 jpeg 模式，然后根据 OV2640_JPEG_WIDTH 和 OV2640_JPEG_HEIGHT 这两个宏，确定输出图像的尺寸。因为 STM32F103ZET6 内部内存最大也就 64KB，我们申请了 52K 用于存放 JPEG 数据，最大也就可以存放 1024*768 左右的 jpg 图片。所以，OV2640_JPEG_WIDTH 的值为 1024，OV2640_JPEG_HEIGHT 的值为 768。如果你的内存足够（得 150KB 以上），则可以定义这两个的值，最大是 1600*1200。

在设置好输出尺寸后，再设置 PCLK 频率，以便完整采集 JPEG 图像。因为在设置 JPEG 图像输出后，OV2640 需要一段时间来调整参数（自动调整曝光，白平衡什么的），所以，我们丢弃前面 10 帧的数据（否则采集的图像可能很黑），然后才开始采集一帧 JPEG 数据，这里的采集方式在第一节末尾部分，已经详细介绍了，完全就是按照第一节介绍的方式来采集的，在完成 JPEG 图像采集后，进行简单处理（丢弃一些无用数据，JPEG 有效数据流是以 0XFF，0XD8 开头，以 0XFF，0XD9 结尾的），然后将整个采集到的 jpeg 数据写入 SD 卡，完成一次 JPG 拍照。拍好完成后，设置 OV2640 恢复 RGB565 模式，以便将图像显示到 LCD。

最后，main 函数，则先初始化所有要用到的外设，最后初始化 ATK-OV2640 摄像头模块，在初始化成功之后，设置 OV2640 为 RGB565 输出模式，并设置图像输出尺寸为 LCD 分辨率，以便在 LCD 上面显示采集到的图像数据。

当OV2640_VSYNC信号为高电平的时候，不采集数据，执行按键扫描，以便进行bmp/jpg拍照。

当 OV2640_VSYNC 信号为低电平的时候，进行 OV2640 的图像数据采集，以行为单位，结合 DMA 传输，完成 OV2640 的图像采集并通过 DMA 传输给 LCD 显示，采集方法在第一节末尾已经介绍过了，这里就不再多说了。

最好，我们要通过 USMART 来设置和调节摄像头的参数，所以，在 usmart_config.h 里面修改 usmart_nametab 的内容如下：

```
struct _m_usmart_nametab usmart_nametab[]=
{
```

```
#if USMART_USE_WRFUNS==1      //如果使能了读写操作
    (void*)read_addr,"u32 read_addr(u32 addr)",
    (void*)write_addr,"void write_addr(u32 addr,u32 val)",
#endif
    (void*)SCCB_WR_Reg,"u8 SCCB_WR_Reg(u8 reg,u8 data)",
    (void*)SCCB_RD_Reg,"u8 SCCB_RD_Reg(u8 reg)",
    (void*)OV2640_Auto_Exposure,"void OV2640_Auto_Exposure(u8 level)",
    (void*)OV2640_Light_Mode,"OV2640_Light_Mode(u8 mode)",
    (void*)OV2640_Color_Saturation,"void OV2640_Color_Saturation(u8 sat)",
    (void*)OV2640_Brightness,"void OV2640_Brightness(u8 bright)",
    (void*)OV2640_Contrast,"void OV2640_Contrast(u8 contrast)",
    (void*)OV2640_Special_Effects,"void OV2640_Special_Effects(u8 eft)",
    (void*)OV2640_Color_Bar,"void OV2640_Color_Bar(u8 sw)",
    (void*)OV2640_Window_Set,"void OV2640_Window_Set(u16 sx,u16 sy,u16 width
                                        ,u16 height)",
    (void*)OV2640_OutSize_Set,"u8 OV2640_OutSize_Set(u16 width,u16 height)",
    (void*)OV2640_ImageWin_Set,"u8 OV2640_ImageWin_Set(u16 offx,u16 offy,u16
                                        width,u16 height)",
    (void*)OV2640_ImageSize_Set,"u8 OV2640_ImageSize_Set(u16 width,u16 height)",
    (void*)OV2640_JPEG_Mode,"void OV2640_JPEG_Mode(void)",
    (void*)OV2640_RGB565_Mode,"void OV2640_RGB565_Mode(void)",
};
```

这样，我们就可以通过 USMART 设置摄像头的曝光、色饱和度、亮度、对比度和特效等。另外，我们还可以通过 SCCB_WR_Reg 和 SCCB_RD_Reg 这两个函数，来修改和读取 OV2640 的各项设置，轻松实现摄像头的调试。

4、验证

在代码编译成功之后，我们通过下载代码到 ALIENTEK 战舰 V3 或者精英 STM32F103 开发板上，假定 SD 卡和 ATK-OV2640 都已经连接在开发板上了。注意：如果没有 SD 卡，则无法进行拍照!!

程序在 OV2640 初始化成功后，显示提示信息，然后在开发板的 LCD 上面，便开始显示 OV2640 摄像头模块所拍摄到的图像了。DS0 开始不停的闪烁，提示程序正在运行。

此时，按 KEY0 可以进行 BMP 拍照，拍照尺寸为屏幕的分辨率。按 KEY1 可以进行 JPG 拍照，拍照尺寸固定为 VGA(1024*768)分辨率。

拍照样张，如图 4.1~4.3 所示：

图 4.1 bmp 拍照样张（320*480 分辨率）

图 4.2 jpg 拍照样张 1（1024*768 分辨率）

图 4.3 jpg 拍照样张 2（1024*768 分辨率）

图 4.1 我们采用的是 ALIENTEK 3.5 寸的 LCD 模块，分辨率为 320*480 所以，拍出的图片也是 320*480 分辨率，因为我们设置的图像尺寸是 UXGA(1600*1200)分辨率，而图像

输出大小设置为 320*480，所以图像被压缩了，导致变形严重，看上去压扁了。

图 4.2 和图 4.3，则是采用 JPG 拍照，所得到的 jpg 图片，采用的图像输出大小为 VGA（1024*768）分辨率，这样图像就是真实的尺寸，没有任何变形（1024/768=1600/1200）。

同时，你还可以在串口，通过 USMART 调用 SCCB_WR_Reg 等函数，来设置 OV2640 的各寄存器，达到调试测试 OV2640 的目的，如图 4.4 所示：

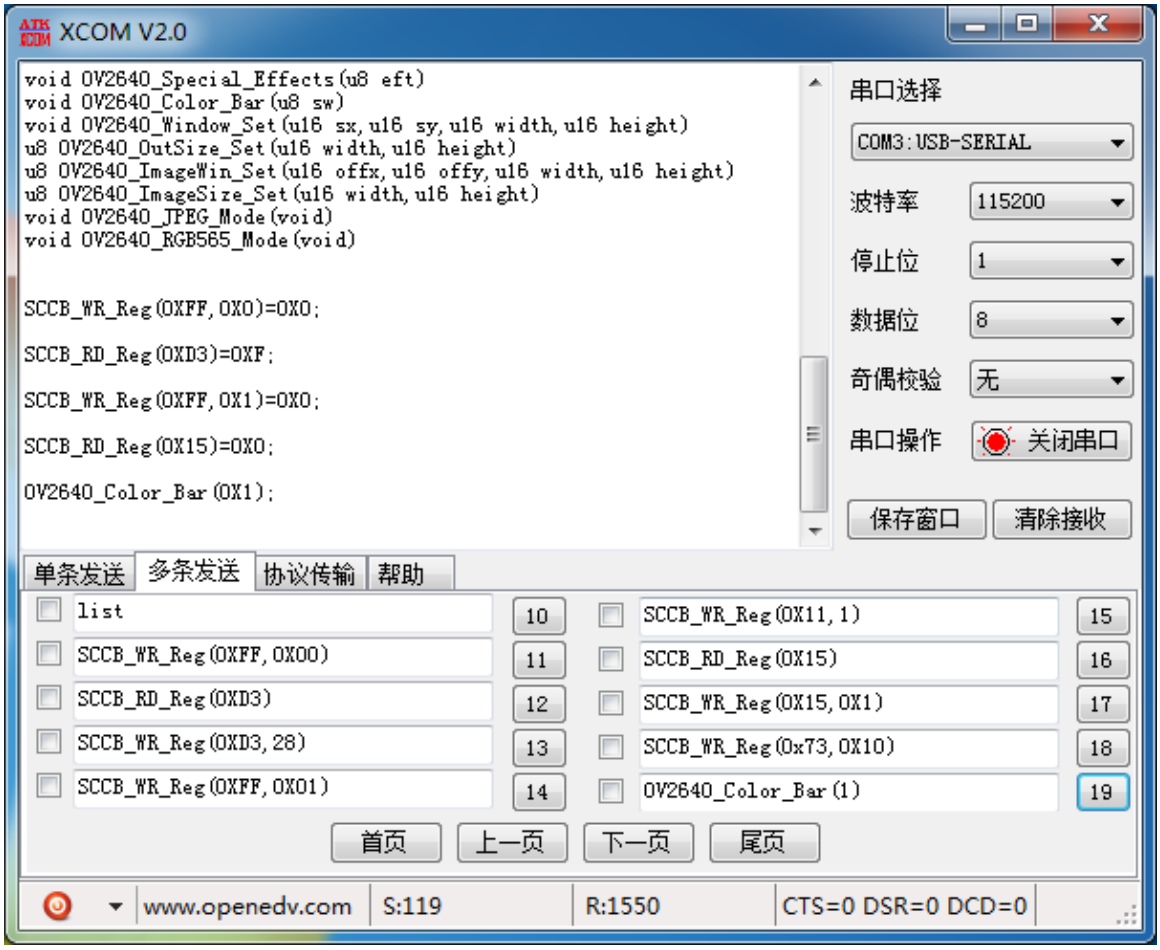

图 4.4 USMART 调试 OV2640

最后，在 main 函数里面，我们取消 TIM6_Int_Init 函数的屏蔽，就可以将 LCD 的帧率给打印出来，因为采用中断的方式打印，会导致读 OV2640 数据被打断，所以开启帧打印后，屏幕会有抖动。

经测试，在战舰 V3 / 精英 STM32F103 开发板上，对于 2.8 寸 TFTLCD（240*320），本例程显示帧率可达 7 帧，对于 3.5 寸 TFTLCD（320*480），本例程显示帧率可达 3 帧，对于 4.3 寸/7 寸的 TFTLCD（480*800），则不到 1 帧。

虽说在 STM32F103 开发板上，OV2640 帧率偏低，但是，在探索者 STM32F407 开发板上，ATK-OV2640 摄像头模块，则可以全速运行（30 帧无压力）。如果要求显示流畅的，可以考虑使用探索者 STM32F407 开发板。

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