

# AN1615A ATK-OV2640 摄像头模块使用说明

本应用文档（AN1615A，对应**阿波罗 STM32F767 开发板扩展实验 10**）将教大家如何在**阿波罗 STM32F767 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&DCMI 简介
- 2, 硬件连接
- 3, 软件实现
- 4, 验证

## 1、OV2640&DCMI 简介

本节将分为两个部分，分别介绍 OV2640 和 STM32F767 的 DCMI 接口。

### 1.1 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。OmniVision 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 的功能框图如图 1.1.1 所示：

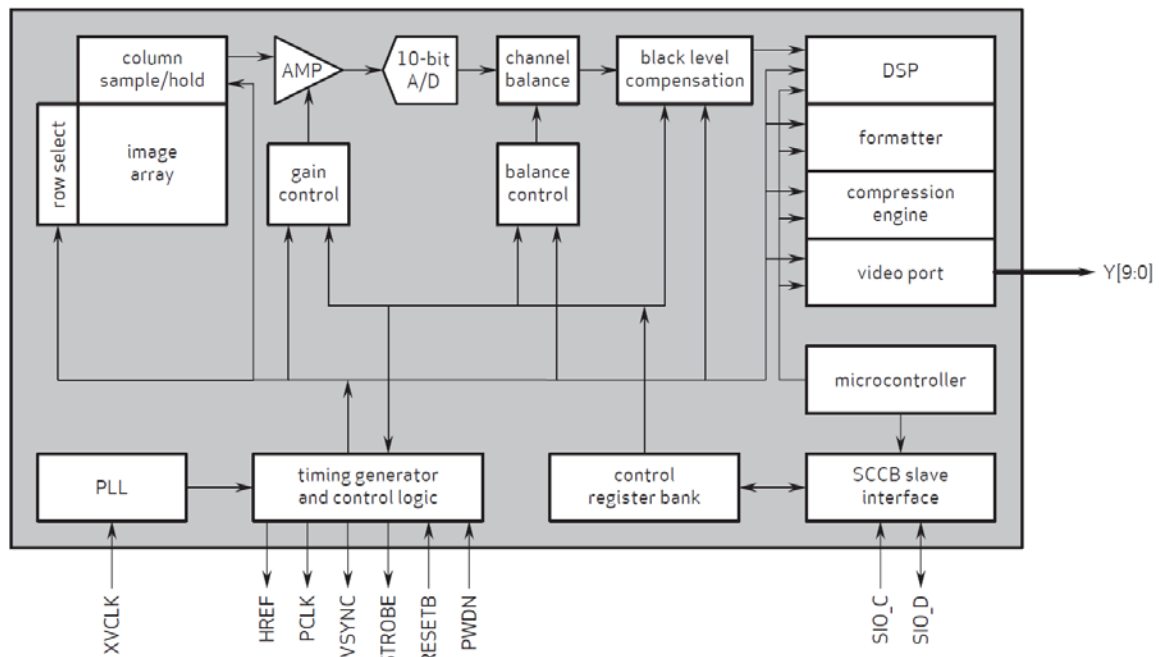


图 1.1.1 OV2640 功能框图

OV2640 传感器包括如下一些功能模块。

### 1.感光阵列 (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 1.1.2 所示：

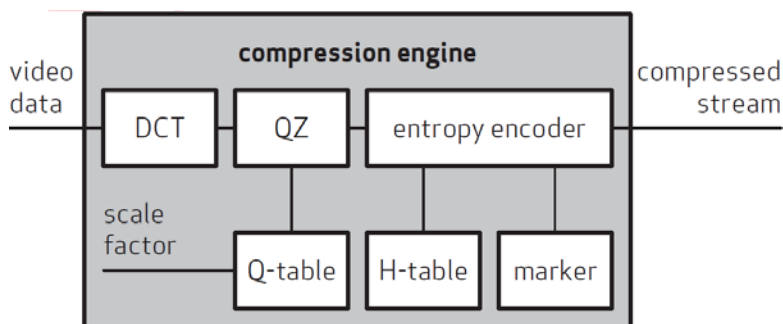


图 1.1.2 压缩引擎框图

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 Serial 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 1.1.3 所示：

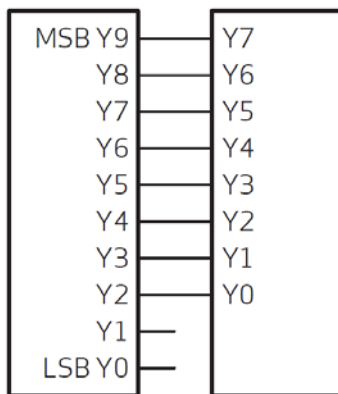


图 1.1.3 OV2640 默认 8 位连接方式

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 Serial 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

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 1.1.4 所示：

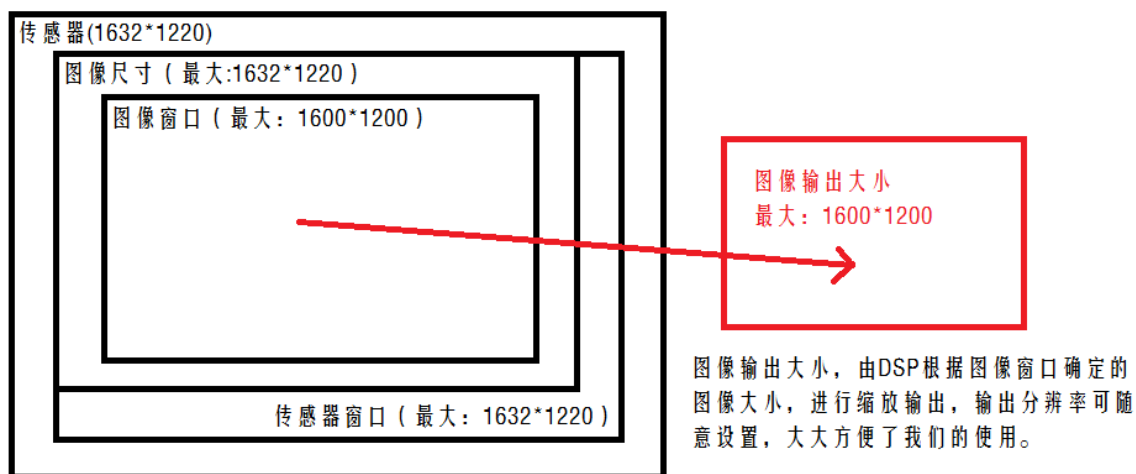


图 1.1.4 OV2640 图像窗口设置简图

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/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 1.1.5 所示：

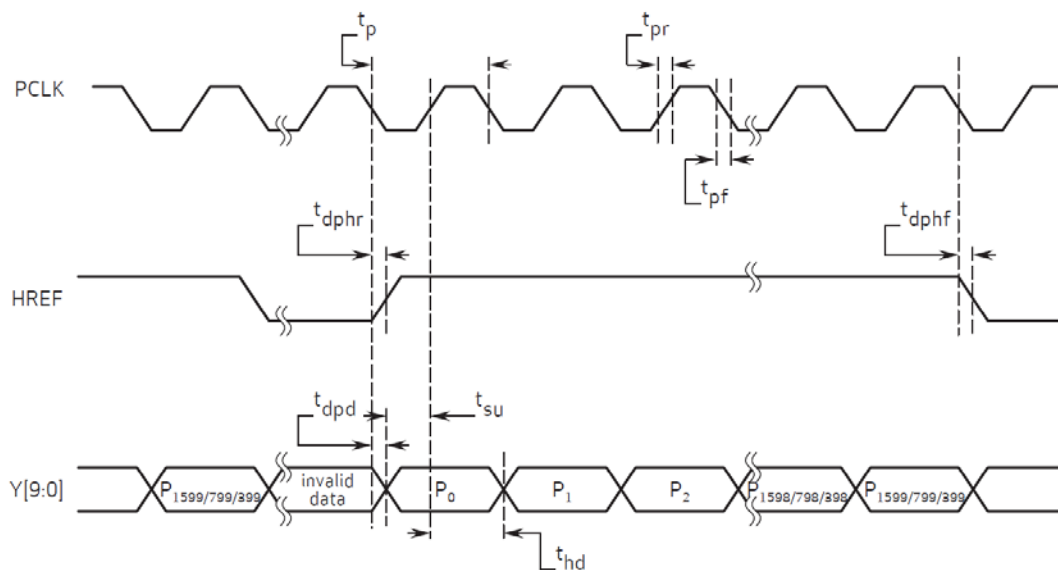


图 1.1.5 OV2640 行输出时序

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：

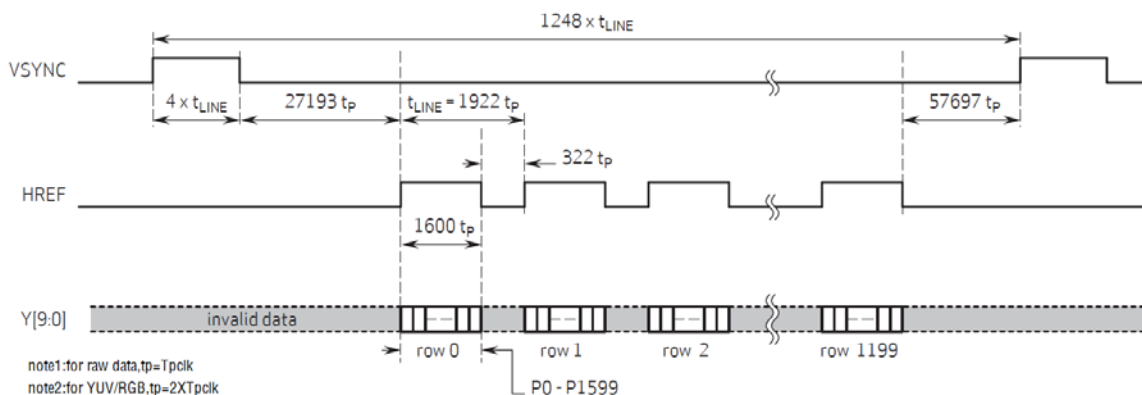


图 1.1.6 OV2640 帧时序

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。OV2640 我们就介绍到这。

## 1.2 STM32F767 DCMI 接口简介

STM32F767 自带了一个数字摄像头 (DCMI) 接口, 该接口是一个同步并行接口, 能够接收外部 8 位、10 位、12 位或 14 位 CMOS 摄像头模块发出的高速数据流。可支持不同的数据格式: YCbCr4:2:2/RGB565 逐行视频和压缩数据 (JPEG)。

STM32F4 DCM 接口特点:

- 8 位、10 位、12 位或 14 位并行接口
- 内嵌码/外部行同步和帧同步
- 连续模式或快照模式
- 裁剪功能
- 支持以下数据格式:
  - 1, 8/10/12/14 位逐行视频: 单色或原始拜尔 (Bayer) 格式
  - 2, YCbCr 4:2:2 逐行视频
  - 3, RGB 565 逐行视频
  - 4, 压缩数据: JPEG

DCMI 接口包括如下一些信号:

- 1, 数据输入 (D[0:13]), 用于接摄像头的的数据输出, 接 OV2640 我们只用了 8 位数据。
- 2, 水平同步 (行同步) 输入 (HSYNC), 用于接摄像头的 HSYNC/HREF 信号。
- 3, 垂直同步 (场同步) 输入 (VSYNC), 用于接摄像头的 VSYNC 信号。
- 4, 像素时钟输入 (PIXCLK), 用于接摄像头的 PCLK 信号。

DCMI 接口是一个同步并行接口, 可接收高速 (可达 54 MB/s) 数据流。该接口包含多达 14 条数据线(D13-D0)和一条像素时钟线(PIXCLK)。像素时钟的极性可以编程, 因此可以在像素时钟的上升沿或下降沿捕获数据。

DCMI 接收到的摄像头数据被放到一个 32 位数据寄存器(DCMI\_DR)中, 然后通过通用 DMA 进行传输。图像缓冲区由 DMA 管理, 而不是由摄像头接口管理。

从摄像头接收的数据可以按行/帧来组织 (原始 YUV/RGB/拜尔模式), 也可以是一系列 JPEG 图像。要使能 JPEG 图像接收, 必须将 JPEG 位 (DCMI\_CR 寄存器的位 3) 置 1。

数据流可由可选的 HSYNC (水平同步) 信号和 VSYNC (垂直同步) 信号硬件同步, 或者通过数据流中嵌入的同步码同步。

STM32F767 DCMI 接口的框图如图 1.2.1 所示:

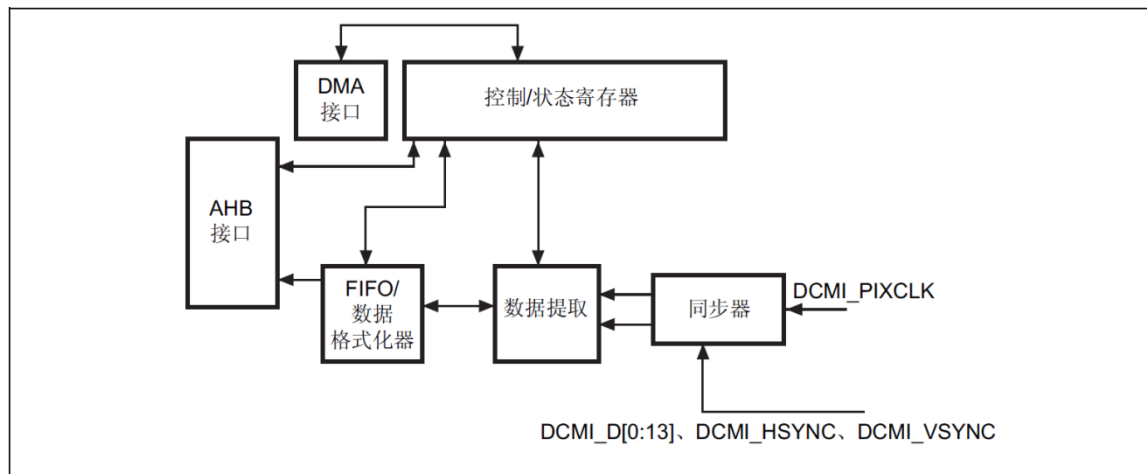


图 1.2.1 DCMI 接口框图



DCMI 接口的数据与 PIXCLK（即 PCLK）保持同步，并根据像素时钟的极性在像素时钟上升沿/下降沿发生变化。HSYNC（HREF）信号指示行的开始/结束，VSYNC 信号指示帧的开始/结束。DCMI 信号波形如图 1.2.2 所示：

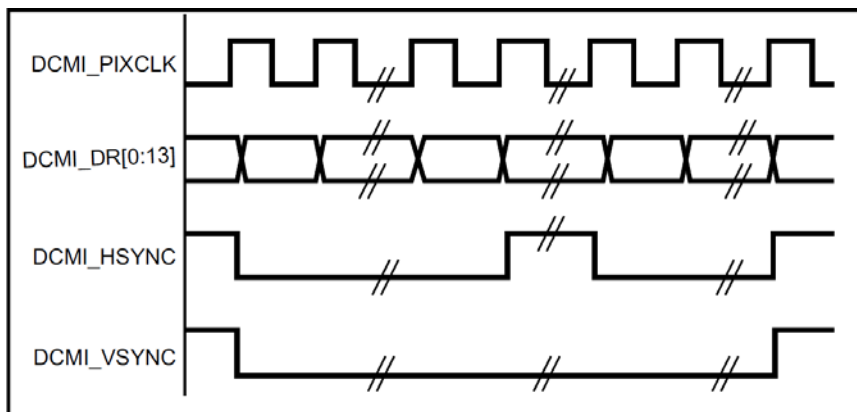


图 1.2.2 DCMI 信号波形

上图中，对应设置为：DCMI\_PIXCLK 的捕获沿为下降沿，DCMI\_HSYNC 和 DCMI\_VSYNC 的有效状态为 1，注意，这里的有效状态实际上对应的是指示数据在并行接口上无效时，HSYNC/VSYNC 引脚上面的引脚电平。

本章我们用到 DCMI 的 8 位数据宽度，通过设置 DCMI\_CR 中的 EDM[1:0]=00 设置。此时 DCMI\_D0~D7 有效，DCMI\_D8~D13 上的数据则忽略，这个时候，每次需要 4 个像素时钟来捕获一个 32 位数据。捕获的第一个数据存放在 32 位字的 LSB 位置，第四个数据存放在 32 位字的 MSB 位置，捕获数据字节在 32 位字中的排布如表 1.2.1 所示：

字节地址	31:24	23:16	15:8	7:0
0	$D_{n+3}[7:0]$	$D_{n+2}[7:0]$	$D_{n+1}[7:0]$	$D_n[7:0]$
4	$D_{n+7}[7:0]$	$D_{n+6}[7:0]$	$D_{n+5}[7:0]$	$D_{n+4}[7:0]$

表 1.2.1 8 位捕获数据在 32 位字中的排布

从表 1.2.1 可以看出，STM32F767 的 DCMI 接口，接收的数据是低字节在前，高字节在后的，所以，要求摄像头输出数据也是低字节在前，高字节在后才可以，否则就还得程序上处理字节顺序，会比较麻烦。

DCMI 接口支持 DMA 传输，当 DCMI\_CR 寄存器中的 CAPTURE 位置 1 时，激活 DMA 接口。摄像头接口每次在其寄存器中收到一个完整的 32 位数据块时，都将触发一个 DMA 请求。

DCMI 接口支持两种同步方式：内嵌码同步和硬件（HSYNC 和 VSYNC）同步。我们简单介绍下硬件同步，详细介绍请参考《STM32F7xx 中文数据手册》第 13.5.3 节。

硬件同步模式下将使用两个同步信号（HSYNC/VSYNC）。根据摄像头模块/模式的不同，可能在水平/垂直同步期间内发送数据。由于系统会忽略 HSYNC/VSYNC 信号有效电平期间内接收的所有数据，HSYNC/VSYNC 信号相当于消隐信号。

为了正确地将图像传输到 DMA/RAM 缓冲区，数据传输将与 VSYNC 信号同步。选择硬件同步模式并启用捕获（DCMI\_CR 中的 CAPTURE 位置 1）时，数据传输将与 VSYNC 信号的无效电平同步（开始下一帧时）。之后传输便可以连续执行，由 DMA 将连续帧传输到多个连续的缓冲区或一个具有循环特性的缓冲区。为了允许 DMA 管理连续帧，每一帧结束时都将激活 VSIF（垂直同步中断标志，即帧中断），我们可以利用这个帧中断来判断是否有一帧数据采集完成，方便处理数据。

DCMI 接口的捕获模式支持：快照模式和连续采集模式。一般我们使用连续采集模式，

通过 DCMI\_CR 中的 CM 位设置。另外，DCMI 接口还支持实现了 4 个字深度的 FIFO，配有一个简单的 FIFO 控制器，每次摄像头接口从 AHB 读取数据时读指针递增，每次摄像头接口向 FIFO 写入数据时写指针递增。因为没有溢出保护，如果数据传输率超过 AHB 接口能够承受的速率，FIFO 中的数据就会被覆盖。如果同步信号出错，或者 FIFO 发生溢出，FIFO 将复位，DCMI 接口将等待新的数据帧开始。

关于 DCMI 接口的其他特性，我们这里就不再介绍了，请大家参考《STM32F7xx 中文参考手册》第 13 章相关内容。

本章，我们将使用 STM32F767IGT6 的 DCMI 接口连接 ALIENTEK OV2640 摄像头模块，该模块采用 8 位数据输出接口，自带 24M 有源晶振，无需外部提供时钟，采用百万高清镜头，单独 3.3V 供电即可正常使用。

ALIENTEK OV2640 摄像头模块外观如图 1.2.3 所示：

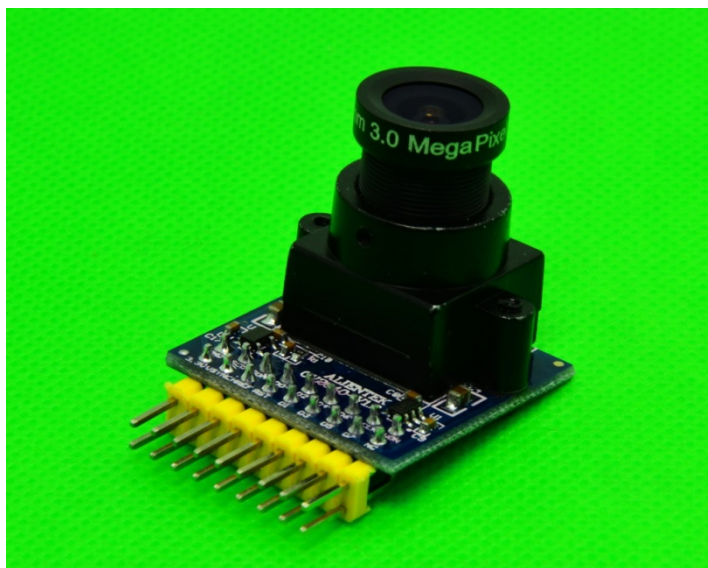


图 1.2.3 ALIENTEK OV2640 摄像头模块外观图

模块原理图如图 1.2.4 所示：

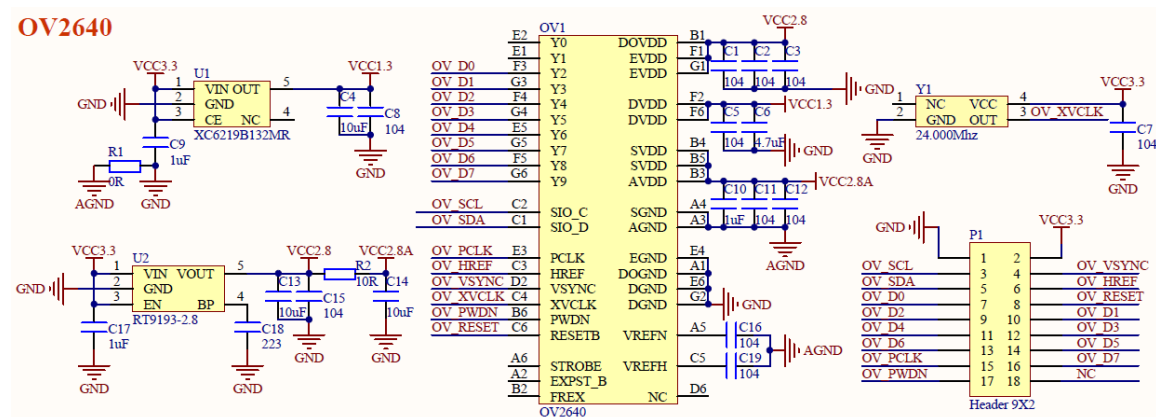


图 1.2.4 ALIENTEK OV2640 摄像头模块原理图

从上图可以看出，ALIENTEK OV2640 摄像头模块自带了有源晶振，用于产生 24M 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 工作电压，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 1.2.2 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_PCLK	像素时钟输出



GND	模块地线	OV_PWDN	掉电使能(高有效)
OV_SCL	SCCB 通信时钟信号	OV_VSYNC	帧同步信号输出
OV_SDA	SCCB 通信数据信号	OV_HREF	行同步信号输出
OV_D[7:0]	8 位数据输出	OV_RESET	复位信号(低有效)

表 1.2.2 OV2640 模块信号及其作用描述

本章，我们将 OV2640 默认配置为 UXGA 输出，也就是 1600\*1200 的分辨率，输出信号设置为：VSYNC 高电平有效，HREF 高电平有效，输出数据在 PCLK 的下降沿输出（即上升沿的时候，MCU 才可以采集）。这样，STM32F4 的 DCMI 接口就必须设置为：VSYNC 低电平有效、HSYNC 低电平有效和 PIXCLK 上升沿有效，这些设置都是通过 DCMI\_CR 寄存器控制的，该寄存器描述如图 1.2.5 所示：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																	ENABLE	Reserved		EDM		FCRC		VSPOL	HSPOL	PCKPOL	ESS	JPEG	CROP	CM	CAPTURE
																	r/w			r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

图 1.2.5 DCMI\_CR 寄存器各位描述

ENABLE，该位用于设置是否使能 DCMI，不过，在使能之前，必须将其他配置设置好。

FCRC[1:0]，这两个位用于帧率控制，我们捕获所有帧，所以设置为 00 即可。

VSPOL，该位用于设置垂直同步极性，也就是 VSYNC 引脚上面，数据无效时的电平状态，根据前面说所，我们应该设置为 0。

HSPOL，该位用于设置水平同步极性，也就是 HSYNC 引脚上面，数据无效时的电平状态，同样应该设置为 0。

PCKPOL，该位用于设置像素时钟极性，我们用上升沿捕获，所以设置为 1。

CM，该位用于设置捕获模式，我们用连续采集模式，所以设置为 0 即可。

CAPTURE，该位用于使能捕获，我们设置为 1。该位使能后，将激活 DMA，DCMI 等待第一帧开始，然后生成 DMA 请求将收到的数据传输到目标存储器中。注意：该位必须在 DCMI 的其他配置（包括 DMA）都设置好了之后，才设置！！

DCMI\_CR 寄存器的其他位，我们就不介绍了，另外 DCMI 的其他寄存器这里也不再介绍，请大家参考《STM32F4xx 中文参考手册》第 13.8 节。

最后，我们来看下用 DCMI 驱动 OV2640 的步骤：

### 1) 配置 OV2640 控制引脚，并配置 OV2640 工作模式。

在启动 DCMI 之前，我们先设置好 OV2640。OV2640 通过 OV\_SCL 和 OV\_SDA 进行寄存器配置，同时还有 OV\_PWDN/OV\_RESET 等信号，我们也需要配置对应 IO 状态，先设置 OV\_PWDN=0，退出掉电模式，然后拉低 OV\_RESET 复位 OV2640，之后再设置 OV\_RESET 为 1，结束复位，然后就是对 OV2640 的大把寄存器进行配置了。然后，可以根据我们的需要，设置成 RGB565 输出模式，还是 JPEG 输出模式。

### 2) 配置相关引脚的模式和复用功能（AF13），使能时钟。

OV2640 配置好之后，再设置 DCMI 接口与摄像头模块连接的 IO 口，使能 IO 和 DCMI 时钟，然后设置相关 IO 口为复用功能模式，复用功能选择 AF13(DCMI 复用)。

### 3) 配置 DCMI 相关设置。

这一步，主要通过 DCMI\_CR 寄存器设置，包括 VSPOL/HSPOL/PCKPOL/数据宽度等重要参数，都在这一步设置，同时我们也开启帧中断，编写 DCMI 中断服务函数，方便进行数据处理（尤其是 JPEG 模式的时候）。不过对于 CAPTURE 位，我们等待 DMA 配置好之后再设置，另外对于 OV2640 输出的 JPEG 数据，我们也不使用 DCMI 的 JPEG 数据模式（实测设置不设置都一样），而是采用正常模式，直接采集。

#### 4) 配置 DMA。

本章采用连续模式采集，并将采集到的数据输出到 LCD (RGB565 模式) 或内存 (JPEG 模式)，所以源地址都是 DCMI\_DR，而目的地址可能是 LCD->RAM 或者 SRAM 的地址。DCMI 的 DMA 传输采用的是 DMA2 数据流 1 的通道 1 来实现的，关于 DMA 的介绍，请大家参考前面的 DMA 实验章节。

#### 5) 设置 OV2640 的图像输出大小，使能 DCMI 捕获。

图像输出大小设置，分两种情况：在 RGB565 模式下，我们根据 LCD 的尺寸，设置输出图像大小，以实现全屏显示（图像可能因缩放而变形）；在 JPEG 模式下，我们可以自由设置输出图像大小（可不缩放）；最后，开启 DCMI 捕获，即可正常工作了。

## 2、硬件设计

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 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 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 UXGA，即 1600\*1200）。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 和 KEY\_UP 按键
- 3) PCF8574（驱动蜂鸣器）
- 4) 串口
- 5) LCD 模块（MCU 屏/RGB 屏）
- 6) SD 卡
- 7) SPI FLASH
- 8) ATK-OV2640 摄像头模块

这些资源，除了最后一个，其他的，我们在标准例程都介绍过。这里我们重点介绍下阿波罗开发板的摄像头接口与 ALIENTEK OV2640 摄像头模块的连接。在开发板的左下角的 2\*9 的 P7 排座，是摄像头模块/OLED 模块共用接口，在标准例程的 OLED 实验，我们曾简单介绍过这个接口。本章，我们只需要将 ALIENTEK OV2640 摄像头模块插入这个接口即可，该接口与 STM32 的连接关系如图 2.1 所示：

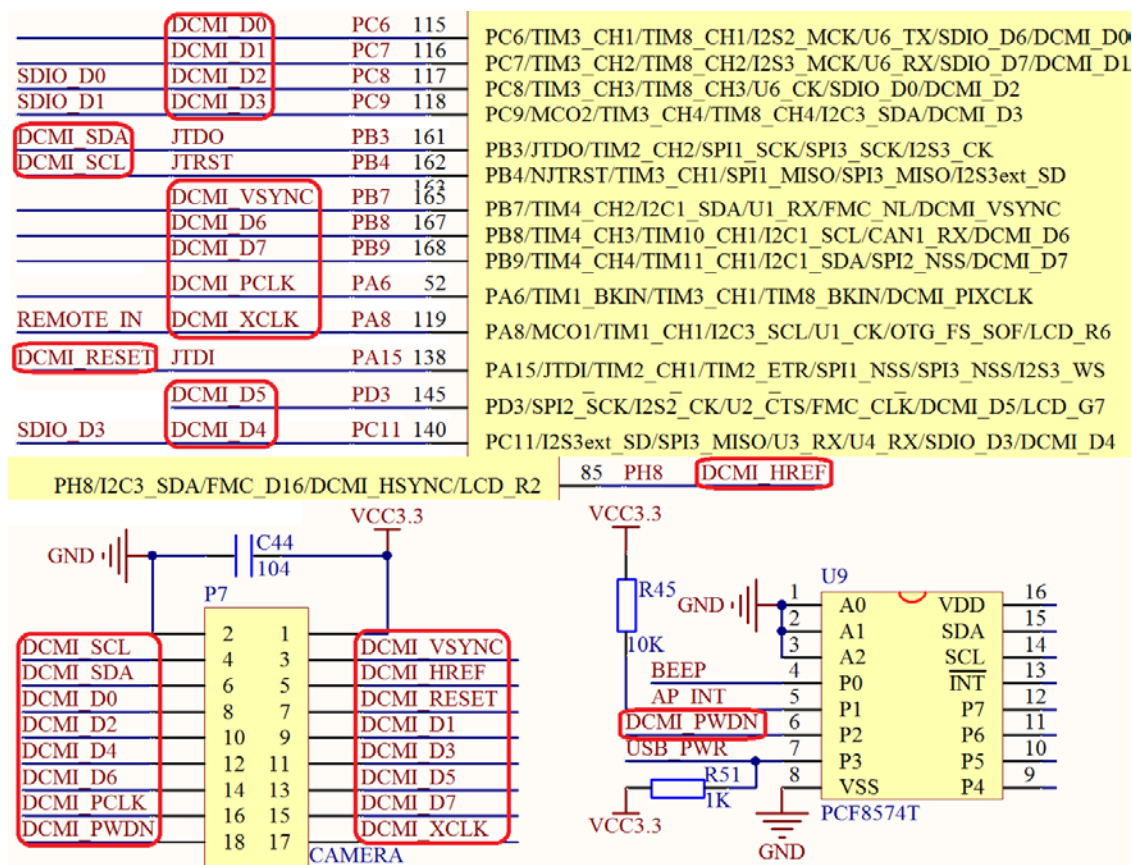


图 2.1 摄像头模块接口与 STM32 连接图

从图 2.1 可以看出，OV2640 摄像头模块的各信号脚与 STM32 的连接关系为：

- DCMI\_VSYNC 接 PB7;
- DCMI\_HREF 接 PH8;
- DCMI\_PCLK 接 PA6;
- DCMI\_SCL 接 PB4;
- DCMI\_SDA 接 PB3;
- DCMI\_RESET 接 PA15;
- DCMI\_PWDN 接 PCF8574T 的 P2 脚;
- DCMI\_XCLK 接 PA8（本章未用到）;
- DCMI\_D[7:0]接 PB9/PB8/PD3/PC11/PC9/PC8/PC7/PC6;

这些线的连接，阿波罗 STM32F767 开发板的内部已经连接好了，我们只需要将 OV2640 摄像头模块插上去就好了。**特别注意：**DCMI 摄像头接口和 SDIO 以及红外接收头有冲突，使用的时候，必须分时复用才可以，不可同时使用。另外，DCMI\_PWDN 连接在 PCF8574T 的 P2 脚上，所以本章必须使用 PCF8574T，来间接控制 DCMI\_PWDN。

实物连接如图 2.2 所示：

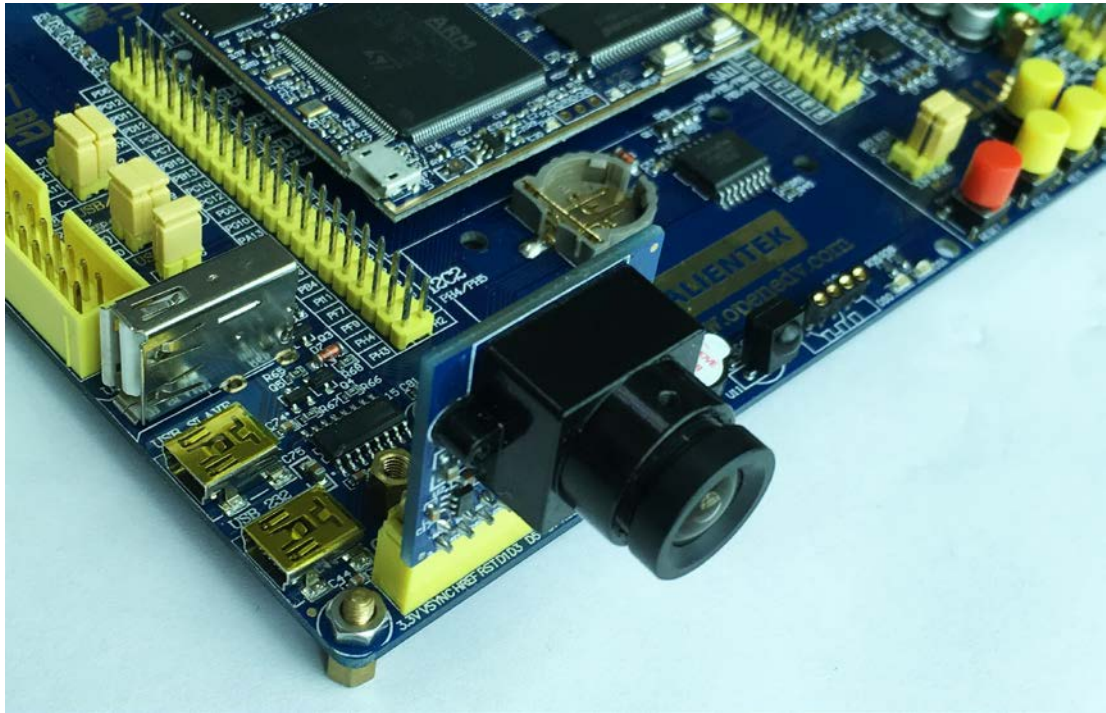


图 2.2 OV2640 摄像头模块与开发板连接实物图

### 3、软件实现

本扩展例程（扩展实验 10 ATK-OV2640 摄像头模块测试实验），我们在阿波罗 STM32 开发板标准例程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：

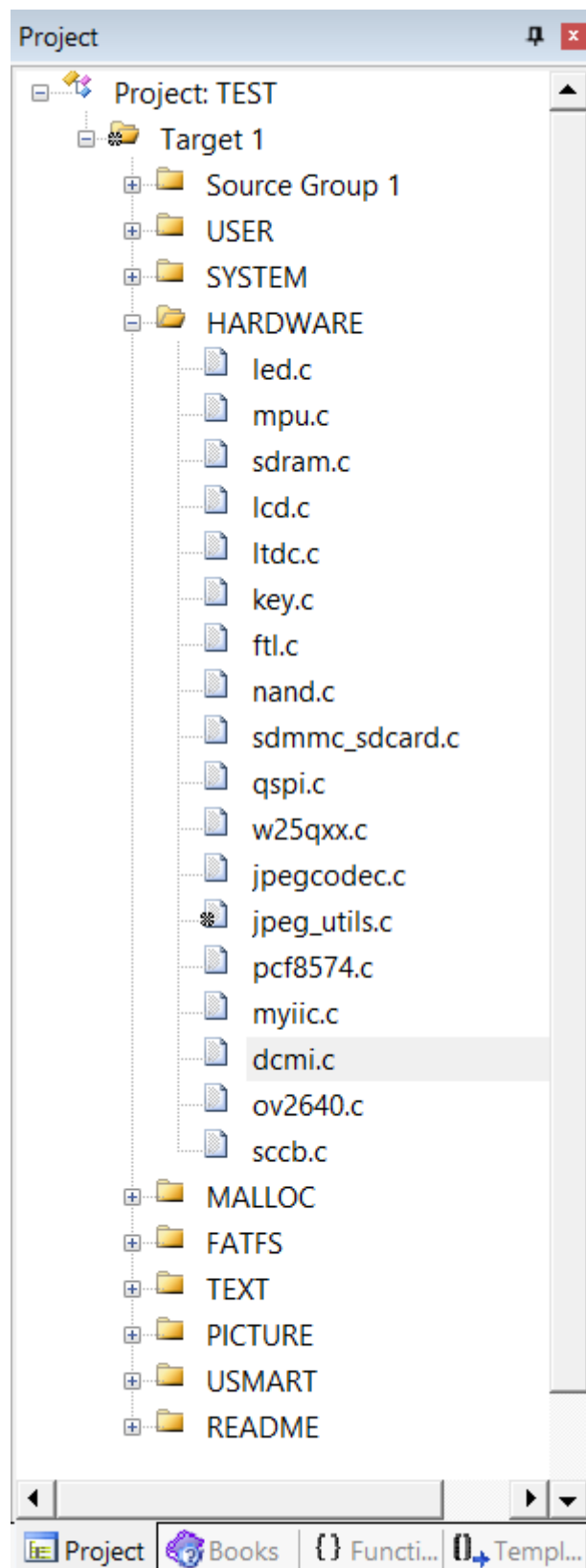


图 3.1 ATK-OV2640 摄像头模块测试实验工程结构

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->AHB1ENR|=1<<0;      //使能外设 PORTA 时钟
    GPIO_Set(GPIOA,PIN15,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_50M,
    GPIO_PUPD_PU);            //PA15 推挽输出
    PCF8574_Init();           //初始化 PCF8574
    OV2640_PWDN_Set(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 dcmi.c 里面的代码，如下：

```
u8 ov_frame=0; //帧率
extern void jpeg_data_process(void); //JPEG 数据处理函数
u8 ov_frame=0; //帧率
extern void jpeg_data_process(void); //JPEG 数据处理函数
//DCMI 中断服务函数
void DCMI_IRQHandler(void)
{
    static u8 led1sta=1;
    if(DCMI->MISR&0X01) //捕获到一帧图像
    {
        jpeg_data_process();//jpeg 数据处理
        DCMI->ICR|=1<<0; //清除帧中断
        LED1(led1sta^=1);
        ov_frame++;
    }
}
//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)
{
    u32 tempreg=0;
    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
tempreg|=0<<6;                          //外设到存储器模式
tempreg|=1<<8;                          //循环模式
tempreg|=0<<9;                          //外设非增量模式
tempreg|=meminc<<10;                    //存储器增量模式
tempreg|=2<<11;                         //外设数据长度:32 位
tempreg|=memblen<<13;                   //存储器位宽,8/16/32bit
tempreg|=2<<16;                         //高优先级
tempreg|=0<<21;                         //外设突发单次传输
tempreg|=0<<23;                         //存储器突发单次传输
tempreg|=1<<25;                         //通道 1 DCMI 通道
if(mem1addr)                            //双缓冲的时候,才需要开启
{
    tempreg|=1<<18;                      //双缓冲模式
    tempreg|=1<<4;                      //开启传输完成中断
    MY_NVIC_Init(2,3,DMA2_Stream1_IRQn,2); //抢占 1, 子优先级 3, 组 2
}
DMA2_Stream1->CR=tempreg;                //设置 CR 寄存器
}
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_Stream1,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
        SCB_CleanInvalidateDCache();      //清除无效的 D-Cache
    }
}
//DCMI 初始化
//摄像头模块 ----- STM32 开发板
// OV_D0~D7 ----- PB8/PB9/PD3/PC11/PC9/PC8/PC7/PC6
// OV_SCL ----- PB4
// OV_SDA ----- PB3
// OV_VSYNC ----- PB7
// OV_HREF ----- PH8
// OV_RESET ----- PA15

```

```
// OV_PCLK ----- PA6
// OV_PWDN ----- PCF8574_P2
void DCMI_Init(void)
{
    u32 tempreg=0;
    //设置 IO
    RCC->AHB1ENR|=1<<0;      //使能外设 PORTA 时钟
    RCC->AHB1ENR|=1<<1;      //使能外设 PORTB 时钟
    RCC->AHB1ENR|=1<<2;      //使能外设 PORTC 时钟
    RCC->AHB1ENR|=1<<3;      //使能外设 PORTD 时钟
    RCC->AHB1ENR|=1<<7;      //使能外设 PORTH 时钟
    RCC->AHB2ENR|=1<<0;      //能 DCMI 时钟

    GPIO_Set(GPIOA,PIN6,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_100M,
    GPIO_PUPD_PU);           //PA6 复用功能输出
    GPIO_Set(GPIOB,PIN7|PIN8|PIN9,GPIO_MODE_AF,GPIO_OTYPE_PP,
    GPIO_SPEED_100M,GPIO_PUPD_PU); //PB7/8/9 复用功能输出
    GPIO_Set(GPIOC,PIN6|PIN7|PIN8|PIN9|PIN11,GPIO_MODE_AF,GPIO_OTYPE_PP,
    GPIO_SPEED_100M,GPIO_PUPD_PU); //PC6/7/8/9/11 复用功能输出
    GPIO_Set(GPIOD,PIN3,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_100M,
    GPIO_PUPD_PU);           //PD3 复用功能输出
    GPIO_Set(GPIOH,PIN8,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_100M,
    GPIO_PUPD_PU);           //PH8 复用功能输出
    GPIO_AF_Set(GPIOH,8,13); //PH8,AF13 DCMI_HSYNC
    GPIO_AF_Set(GPIOA,6,13); //PA6,AF13 DCMI_PCLK
    GPIO_AF_Set(GPIOB,7,13); //PB7,AF13 DCMI_VSYNC
    GPIO_AF_Set(GPIOC,6,13); //PC6,AF13 DCMI_D0
    GPIO_AF_Set(GPIOC,7,13); //PC7,AF13 DCMI_D1
    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
    GPIO_AF_Set(GPIOD,3,13); //PD3,AF13 DCMI_D5
    GPIO_AF_Set(GPIOB,8,13); //PB8,AF13 DCMI_D6
    GPIO_AF_Set(GPIOB,9,13); //PB9,AF13 DCMI_D7
    //清除原来的设置
    DCMI->IER=0x0;
    DCMI->ICR=0x1F;
    DCMI->ESCR=0x0;
    DCMI->ESUR=0x0;
    DCMI->CWSTRTR=0x0;
    DCMI->CWSIZER=0x0;
    tempreg|=0<<1;      //连续模式
    tempreg|=0<<2;      //全帧捕获
    tempreg|=0<<4;      //硬件同步 HSYNC,VSYNC
```



```
    tempreg|=1<<5;        //PCLK 上升沿有效
    tempreg|=0<<6;        //HSYNC 低电平有效
    tempreg|=0<<7;        //VSYNC 低电平有效
    tempreg|=0<<8;        //捕获所有的帧
    tempreg|=0<<10;       //8 位数据格式
    DCMI->IER|=1<<0;      //开启帧中断
    tempreg|=1<<14;       //DCMI 使能
    DCMI->CR=tempreg;     //设置 CR 寄存器
    MY_NVIC_Init(2,2,DCMI_IRQn,2);    //抢占 1，子优先级 2，组 2
}
//DCMI,启动传输
void DCMI_Start(void)
{
    LCD_SetCursor(0,0);
    LCD_WriteRAM_Prepare();           //开始写入 GRAM
    DMA2_Stream1->CR|=1<<0;           //开启 DMA2,Stream1
    DCMI->CR|=1<<0;                   //DCMI 捕获使能
}
//DCMI,关闭传输
void DCMI_Stop(void)
{
    DCMI->CR&=~(1<<0);               //DCMI 捕获关闭
    while(DCMI->CR&0X01);             //等待传输结束
    DMA2_Stream1->CR&=~(1<<0);        //关闭 DMA2,Stream1
}
/////////////////////////////////////////////////////////////////
//以下两个函数,供 usmart 调用,用于调试代码
//DCMI 设置显示窗口
//sx,sy;LCD 的起始坐标
//width,height;LCD 显示范围.
void DCMI_Set_Window(u16 sx,u16 sy,u16 width,u16 height)
{
    DCMI_Stop();
    LCD_Clear(WHITE);
    LCD_Set_Window(sx,sy,width,height);
    OV2640_OutSize_Set(width,height);
    LCD_SetCursor(0,0);
    LCD_WriteRAM_Prepare();           //开始写入 GRAM
    DMA2_Stream1->CR|=1<<0;           //开启 DMA2,Stream1
    DCMI->CR|=1<<0;                   //DCMI 捕获使能
}

//通过 usmart 调试,辅助测试用.
//pclk/hsync/vsync:三个信号的有限电平设置
```

```

void DCMI_CR_Set(u8 pclk,u8 hsync,u8 vsync)
{
    DCMI->CR=0;
    DCMI->CR|=pclk<<5;      //PCLK 有效边沿设置
    DCMI->CR|=hsync<<6;     //HSYNC 有效电平设置
    DCMI->CR|=vsync<<7;     //VSYNC 有效电平设置
    DCMI->CR|=1<<14;        //DCMI 使能
    DCMI->CR|=1<<0;         //DCMI 捕获使能
}

```

其中：DCMI\_IRQHandler 函数，用于处理帧中断，可以实现帧率统计（需要定时器支持）和 JPEG 数据处理等。DCMI\_DMA\_Init 函数，则用于配置 DCMI 的 DMA 传输，其外设地址固定为：DCMI->DR，而存储器地址可变（LCD 或者 SRAM）。DMA 被配置为循环模式，一旦开启，DMA 将不停的循环传输数据。DMA2\_Stream1\_IRQHandler 函数，用于在使用 RGB 屏的时候，双缓冲存储时，数据的搬运处理（通过 dcmi\_rx\_callback 函数实现）。DCMI\_Init 函数用于初始化 STM32F767 的 DCMI 接口，这是根据 1.2 节提到的配置步骤进行配置的。最后，DCMI\_Start 和 DCMI\_Stop 两个函数，用于开启或停止 DCMI 接口。

其他部分代码我们就不再细说了，请参考光盘本例程源码(扩展实验 10 ATK-OV2640 摄像头模块测试实验)。

最后，我们看 test.c 里面的代码，如下：

```

//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.
vu8 bmp_request=0;
u8 ovx_mode=0;          //bit0:0,RGB565 模式;1,JPEG 模式
u16 curline=0;          //摄像头输出数据,当前行编号
u16 yoffset=0;          //y 方向的偏移量

#define jpeg_buf_size    4*1024*1024 //定义 JPEG 数据缓存 jpeg_buf 的大小(4M 字节)
#define jpeg_line_size   2*1024     //定义 DMA 接收数据时,一行数据的最大值
u32 *dcmi_line_buf[2];           //RGB 屏时,摄像头采用一行一行读取,定义行缓存
u32 *jpeg_data_buf;              //JPEG 数据缓存 buf

volatile u32 jpeg_data_len=0;     //buf 中的 JPEG 有效数据长度
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

//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;
    curline=yoffset;       //行数复位
}

```

```
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_line_size-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]=dcmi_line_buf[1][i];
        //读取 buf1 里面的剩余数据
        else for(i=0;i<rlen;i++)pbuf[i]=dcmi_line_buf[0][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_line_size;
        //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_line_size;i++)pbuf[i]=dcmi_line_buf[0][i];//读取 buf0 里面的数据
```

```
        jpeg_data_len+=jpeg_line_size;        //偏移
    }else                                     //buf1 已满,正常处理 buf0
    {
        for(i=0;i<jpeg_line_size;i++)pbuf[i]=dcmi_line_buf[1][i];//读取 buf1 里面的数据
        jpeg_data_len+=jpeg_line_size;        //偏移
    }
}
//RGB 屏数据接收回调函数
void rgblcd_dcmi_rx_callback(void)
{
    u16 *pbuf;
    if(DMA2_Stream1->CR&(1<<19))            //DMA 使用 buf1,读取 buf0
    {
        pbuf=(u16*)dcmi_line_buf[0];
    }else                                     //DMA 使用 buf0,读取 buf1
    {
        pbuf=(u16*)dcmi_line_buf[1];
    }
    LTDC_Color_Fill(0,curline,lcddev.width-1,curline,pbuf); //DM2D 填充
    if(curline<lcddev.height)curline++;
    if bmp_request==1&&curline==(lcddev.height-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.
    }
}

//切换为 OV2640 模式
void sw_ov2640_mode(void)
{
    OV2640_PWDN_Set(0); //OV2640 Power Up
    //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)
{
    OV2640_PWDN_Set(1);        //OV2640 Power Down
    //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++;
    }
}
//OV2640 拍照 jpg 图片
//返回值:0,成功
//其他,错误代码
u8 ov2640_jpg_photo(u8 *pname)
{
    FIL* f_jpg;
    u8 res=0,headok=0;
    u32 bwr;
    u32 i,jpgstart,jpglen;
    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_ov2640_mode();                          //切换为 OV2640 模式
    OV2640_JPEG_Mode();                        //JPEG 模式
    OV2640_ImageWin_Set(0,0,1600,1200);
    OV2640_OutSize_Set(1600,1200);            //拍照尺寸为 1600*120
    dcmi_rx_callback=jpeg_dcmi_rx_callback;    //JPEG 接收数据回调函数
    DCMI_DMA_Init((u32)dcmi_line_buf[0],(u32)dcmi_line_buf[1],jpeg_line_size,2,1);
    //DCMI DMA 配置
    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;
    jpglen=0;//设置 jpg 文件大小为 0
    headok=0;    //清除 jpg 头标记
    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))//找到 FF D8
        {
            jpgstart=i;
            headok=1;                //标记找到 jpg 头(FF D8)
        }
        if((pbuf[i]==0XFF)&&(pbuf[i+1]==0XD9)&&headok)
        //找到头以后,再找 FF D9
        {
            jpglen=i-jpgstart+2;
            break;
        }
    }
    if(jpglen)                //正常的 jpeg 数据
    {
        pbuf+=jpgstart;        //偏移到 0XFF,0XD8 处
        res=f_write(f_jpg,pbuf,jpglen,&bwr);
        if(bwr!=jpglen)res=0XFE;

    }else res=0XFD;
}
jpeg_data_len=0;
f_close(f_jpg);
sw_ov2640_mode();          //切换为 OV2640 模式
OV2640_RGB565_Mode();      //RGB565 模式
if(lcdltdc.pwidth!=0)      //RGB 屏
{
    dcmi_rx_callback=rgblcd_dcmi_rx_callback;    //RGB 屏接收数据回调函数
    DCMI_DMA_Init((u32)dcmi_line_buf[0],(u32)dcmi_line_buf[1],lcddev.width/2,1,1);
    //DCMI DMA 配置
}else                //MCU 屏
{
```

```
        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 led0sta=1;
    u8 res;
    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
    u16 outputheight=0;

    Stm32_Clock_Init(432,25,2,9); //设置时钟,216Mhz
    delay_init(216);             //延时初始化
    uart_init(108,115200);        //初始化串口波特率为 115200
    usmart_dev.init(108);         //初始化 USMART
    LED_Init();                   //初始化与 LED 连接的硬件接口
    MPU_Memory_Protection();      //保护相关存储区域
    SDRAM_Init();                 //初始化 SDRAM
    LCD_Init();                   //初始化 LCD
    KEY_Init();                   //初始化按键
    PCF8574_Init();               //初始化 PCF8574
    OV2640_Init();                //初始化 OV2640
    sw_sdcard_mode();             //首先切换为 OV5640 模式
    W25QXX_Init();                //初始化 W25Q256
    my_mem_init(SRAMIN);           //初始化内部内存池
    my_mem_init(SRAMEX);           //初始化外部内存池
    my_mem_init(SRAMTCM);          //初始化 TCM 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
    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);
    }
}
```

```
}
Show_Str(30,50,200,16,"阿波罗 STM32F4/F7 开发板",16,0);
Show_Str(30,70,200,16,"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,"WK_UP:FullSize/Scale",16,0);
Show_Str(30,150,200,16,"2016 年 9 月 31 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
{
    res=f_mkdir("0:/PHOTO");       //创建 PHOTO 文件夹
    Show_Str(30,190,240,16,"SD 卡错误!",16,0);
    delay_ms(200);
    Show_Str(30,190,240,16,"拍照功能将不可用!",16,0);
    delay_ms(200);
    sd_ok=0;
}
dcmi_line_buf[0]=mymalloc(SRAMIN,jpeg_line_size*4);
//为 jpeg dma 接收申请内存
dcmi_line_buf[1]=mymalloc(SRAMIN,jpeg_line_size*4);
//为 jpeg dma 接收申请内存
jpeg_data_buf=mymalloc(SRAMEX,jpeg_buf_size);
//为 jpeg 文件申请内存(最大 4MB)
pname=mymalloc(SRAMIN,30);        //为带路径的文件名分配 30 个字节的内存
while(pname==NULL||!dcmi_line_buf[0]||!dcmi_line_buf[1]||!jpeg_data_buf)
//内存分配出错
{
    Show_Str(30,190,240,16,"内存分配失败!",16,0);
    delay_ms(200);
    LCD_Fill(30,190,240,146,WHITE); //清除显示
    delay_ms(200);
}
while(OV2640_Init())              //初始化 OV2640
{
    Show_Str(30,190,240,16,"OV2640 错误!",16,0);
    delay_ms(200);
    LCD_Fill(30,190,239,206,WHITE);
    delay_ms(200);
}
Show_Str(30,210,230,16,"OV2640 正常",16,0);
OV2640_RGB565_Mode();             //RGB565 模式
OV2640_Light_Mode(0);             //自动模式
OV2640_Color_Saturation(3);       //色彩饱和度 0
OV2640_Brightness(4);             //亮度 0
```

```
OV2640_Contrast(3);           //对比度 0
DCMI_Init();                  //DCMI 配置
if(lcdltdc.pwidth!=0)          //RGB 屏
{
    dcmi_rx_callback=rgblcd_dcmi_rx_callback;    //RGB 屏接收数据回调函数
    DCMI_DMA_Init((u32)dcmi_line_buf[0],(u32)dcmi_line_buf[1],lcddev.width/2,1,1);
    //DCMI DMA 配置
}else                            //MCU 屏
{
    DCMI_DMA_Init((u32)&LCD->LCD_RAM,0,1,1,0);
    //DCMI DMA 配置,MCU 屏,竖屏
}
if(lcddev.height>800)
{
    yoffset=(lcddev.height-800)/2;
    outputheight=800;
    SCCB_WR_Reg(0xff,0x01);
    SCCB_WR_Reg(0x11,0x01);    //需要降低帧率
}else
{
    if(lcddev.id==0X7084)
    {
        SCCB_WR_Reg(0xff,0x01);
        SCCB_WR_Reg(0x11,0x01);    //800*480 的 RGB 屏需要降低帧率
    }
    yoffset=0;
    outputheight=lcddev.height;
}
curline=yoffset;              //行数复位
OV2640_OutSize_Set(lcddev.width,outputheight);    //全屏缩放显示
LCD_Clear(BLACK);
DCMI_Start();                 //启动传输
while(1)
{
    key=KEY_Scan(0);           //不支持连按
    if(key)
    {
        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;        //请求关闭 DCMI
            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)
    {
        OV2640_ImageWin_Set((1600-lcddev.width)/2,(1200-outputheight)/2,
        lcddev.width,outputheight);    //1:1 真实尺寸
        sprintf((char*)msgbuf,"Full Size 1:1");
    }else
    {
        OV2640_ImageWin_Set(0,0,1600,1200);
        //全尺寸缩放
        sprintf((char*)msgbuf,"Scale");
    }
    OV2640_OutSize_Set(lcddev.width,outputheight);
    delay_ms(800);
}
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,yoffset,lcddev.width,outputheight,0);
        sw_ov2640_mode();    //切换为 OV2640 模式
    }else if(key==KEY1_PRES)    //JPG 拍照
    {
        camera_new_pathname(pname,1);    //得到文件名
        res=ov2640_jpg_photo(pname);
        if(scale==0)
        {
            OV2640_ImageWin_Set((1600-lcddev.width)/2,(1200-outputheight)/2,
            lcddev.width,outputheight);    //1:1 真实尺寸
        }else
        {
            OV2640_ImageWin_Set(0,0,1600,1200);    //全尺寸缩放
        }
        OV2640_OutSize_Set(lcddev.width,outputheight);
    }
    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);
        PCF8574_WriteBit(BEEP_IO,0); //蜂鸣器短叫，提示拍照完成
        delay_ms(100);
        PCF8574_WriteBit(BEEP_IO,1); //关闭蜂鸣器
    }
    delay_ms(1000); //等待 1 秒钟
    DCMI_Start();
    //这里先使能 dcmi,然后立即关闭 DCMI,后面再开启 DCMI,可以防止 RGB 屏的侧移问题.
    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);
}
DCMI_Start(); //开始显示
}
delay_ms(10);
i++;
if(i==20) //DS0 闪烁.
{
    i=0;
    LED0(led0sta^=1);
}
}
}

```

这部分代码比较长，总共有 8 个函数，我们接下来分别介绍。

### 1, jpeg\_data\_process 函数

该函数用于处理 JPEG 数据的接收，在 DCMI\_IRQHandler 函数（在 dcmi.c 里面）里面被调用，它与 jpeg\_dcmi\_rx\_callback 函数和 ov2640\_jpg\_photo 函数共同控制 JPEG 的数据的采集。JPEG 数据的接收，采用 DMA 双缓冲机制，缓冲数组为：dcmi\_line\_buf（u32 类型，RGB 屏接收 RGB565 数据时，也是用这个数组）；数组大小为：jpeg\_line\_size，我们定义的是 2\*1024，即数组大小为 8K 字节（数组大小不能小于存储摄像头一行输出数据的大小）；JPEG 数据接收处理流程就是按图 1.2.1 所示流程来实现的。由 DMA 传输完成中断和 DCMI 帧中断，两个中断服务函数共同完成 jpeg 数据的采集。采集到的 JPEG 数据，全部存储在 jpeg\_data\_buf 数组里面，jpeg\_data\_buf 数组采用内存管理，从外部 SDRAM 申请 4MB 内存作为 JPEG 数据的缓存。

### 2, jpeg\_dcmi\_rx\_callback 函数

这是 jpeg 数据接收的主要函数，通过判断 DMA2\_Stream1->CR 寄存器，读取不同 dcmi\_line\_buf 里面的数据，存储到 SDRAM 里面（jpeg\_data\_buf）。该函数由 DMA 的传输完成中断服务函数：DMA2\_Stream1\_IRQHandler 调用。

### 3, rgblcd\_dcmi\_rx\_callback 函数

该函数仅在使用 RGB 屏的时候用到。当使用 RGB 屏的时候，我们每接收一行数据，

就使用 DMA2D 填充到 RGB 屏的 GRAM，这里同样也是使用 DMA 的双缓冲机制来接收 RGB565 数据，原理参照图 1.2.1。该函数由 DMA 传输完成中断服务函数调用。

#### 4, sw\_ov2640\_mode

因为 SD 卡和 OV2640 有几个 IO 共用，所以这几个 IO 需要分时复用。该函数用于切换 GPIO8/9/11 的复用功能为 DCMI 接口，并开启 OV2640，这样摄像头模块，可以开始正常工作。

#### 5, sw\_sdcard\_mode

该函数用于切换 GPIO8/9/11 的复用功能为 SDIO 接口，并关闭 OV2640，这样，SD 卡可以开始正常工作。

#### 6, camera\_new\_pathname 函数

该函数用于生成新的带路径的文件名，且不会重复，防止文件互相覆盖。该函数可以生成.bmp/.jpg 的文件名，方便拍照的时候，保存到 SD 卡里面。

#### 7, ov2640\_jpg\_photo 函数

该函数实现 OV2640 的 JPEG 图像采集，并保存图像到 SD 卡，完成 JPEG 拍照。该函数首先设置 OV2640 工作在 JPEG 模式，然后，设置输出分辨率为最高的 UXGA(1600\*1200)。然后，开始采集 JPEG 数据，将第二帧 JPEG 数据，保留下来，并写入 SD 卡里面，完成一次 JPEG 拍照。这里，我们丢弃第一帧 JPEG 数据，是防止采集到的图像数据不完整，导致图片错误。

另外，在保存 jpeg 图片的时候，我们将 0xFF,0xD8 和 0xFF,0xD9 之外的数据，进行了剔除，只留下 0xFF,0xD8~0xFF,0xD9 之间的数据，保证图片文件最小，且无其他乱的数据。

注意，在保存图片的时候，必须将 PC8/9/11 切换为 SD 卡模式，并关闭 OV2640 的输出。在图片保存完成以后，切换回 OV2640 模式，并重新使能 OV2640 的输出。

#### 8, main 函数

该函数完成对各相关硬件的初始化，然后检测 OV2640，初始化 OV2640 为 RGB565 模式，显示采集到的图像到 LCD 上面，实现对图像进行预览。进入主循环以后，按 KEY0 按键，可以实现 BMP 拍照（实际上就是截屏，通过 bmp\_encode 函数实现）；按 KEY1 按键，可实现 JPEG 拍照（1600\*1200 分辨率，通过 ov2640\_jpg\_photo 函数实现）；按 KEY2 按键，可以实现自动对焦（单次）；按 KEY\_UP 按键，可以实现图像缩放/不缩放预览。main 函数实现了我们在 49.2 节所提到的功能。

至此照相机实验代码编写完成。最后，本实验可以通过 USMART 来设置 OV2640 的相关参数，我们将 OV2640\_Light\_Mode、OV2640\_Color\_Saturation、OV2640\_Brightness、SCCB\_WR\_Reg 和 SCCB\_RD\_Reg 等函数添加到 USMART 管理，即可通过串口设置 OV2640 的参数，方便调试。

## 4、验证

在代码编译成功之后，我们通过下载代码到 ALIENTEK 阿波罗 STM32F767 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 UXGA(1600\*1200)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



图 4.3 jpg 拍照样张 2

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 UXGA（1600\*1200）分辨率，这样图像就是真实的尺寸，没有任何变形。

同时，你还可以在串口，通过 USART 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：

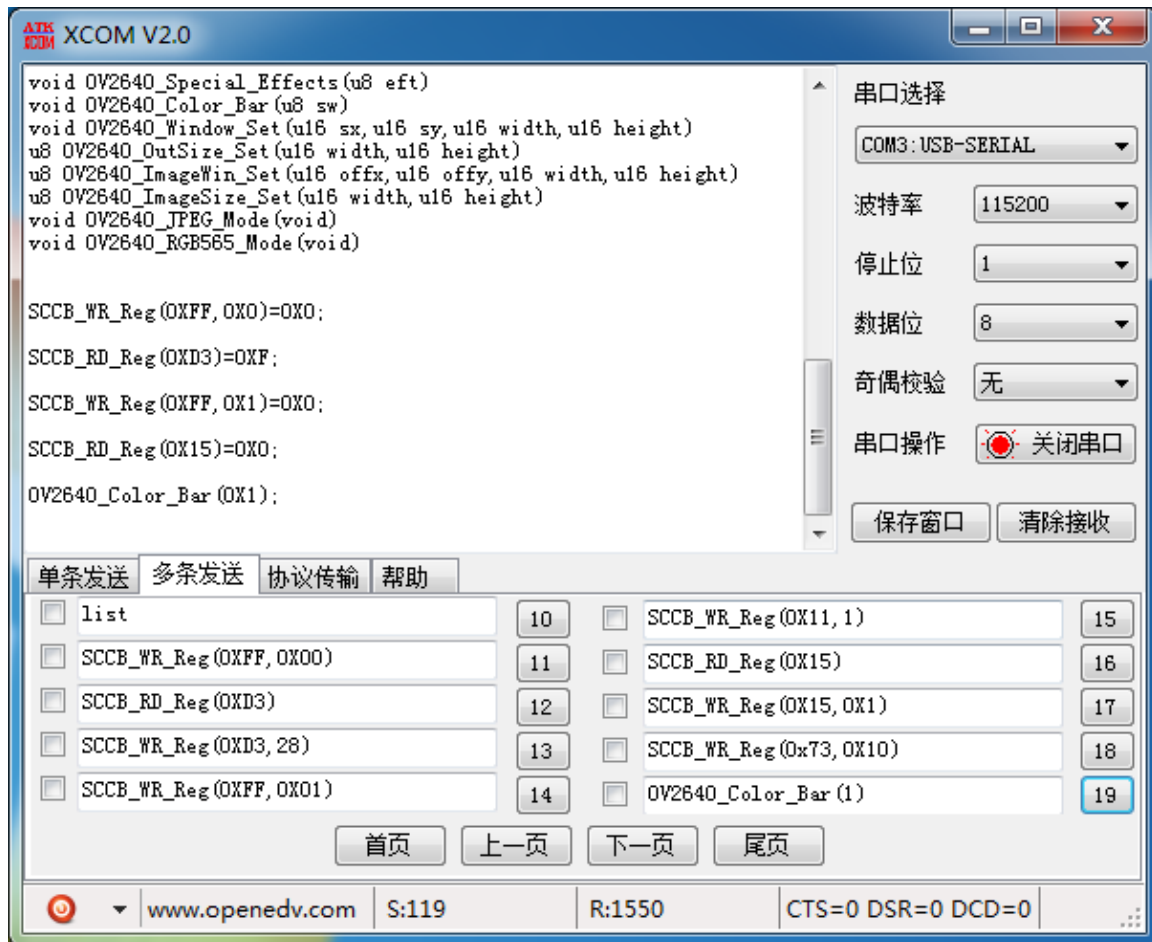


图 4.4 USART 调试 OV2640

至此，ATK-OV2640 在阿波罗 STM32 开发板上的使用就介绍完了。

正点原子@ALIENTEK

公司网址：[www.alientek.com](http://www.alientek.com)

技术论坛：[www.openedv.com](http://www.openedv.com)

电话：020-38271790

传真：020-36773971

