AN1602A ATK-OV5640 模块使用说明

（摄像头实验）

本应用文档（AN1602A，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， ATK-OV5640 摄像头模块简介
2， STM32F407 DCMI 接口简介
3， 硬件连接
4， 软件实现
5， 验证

1、ATK-OV5640 摄像头模块简介

OV5640 是 OV（OmniVision）公司生产的一颗 1/4 寸的 CMOS QSXGA（2592*1944）图像传感器，提供了一个完整的 500W 像素摄像头解决方案，并且集成了自动对焦（AF）功能，具有非常高的性价比。

该传感器体积小、工作电压低，提供单片 QSXGA 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 QSXGA 图像最高达到 15 帧/秒（1080P 图像可达 30 帧，720P 图像可达 60 帧，QVGA 分辨率时可达 120 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，得到清晰稳定的彩色图像。

OV5640 的特点有：

- 采用 1.4μm*1.4μm 像素大小，并且使用 OmniBSI 技术以达到更高性能（高灵敏度、低串扰和低噪声）
- 自动图像控制功能：自动曝光（AEC）、自动白平衡（AWB）、自动消除灯光条纹、自动黑电平校准（ABLC）和自动带通滤波器（ABF）等。
- 支持图像质量控制：色饱和度调节、色调调节、gamma 校准、锐度和镜头校准等
- 标准的 SCCB 接口，兼容 IIC 接口
- 支持 RawRGB、RGB(RGB565/RGB555/RGB444)、CCIR656、YUV(422/420)、YCbCr（422）和压缩图像（JPEG）输出格式
- 支持 QSXGA（500W）图像尺寸输出，以及按比例缩小到其他任何尺寸
- 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
- 支持数字视频接口（DVP）和 MIPI 接口
- 支持自动对焦
- 自带嵌入式微处理器

OV5640 的功能框图图如图 1.1 所示：

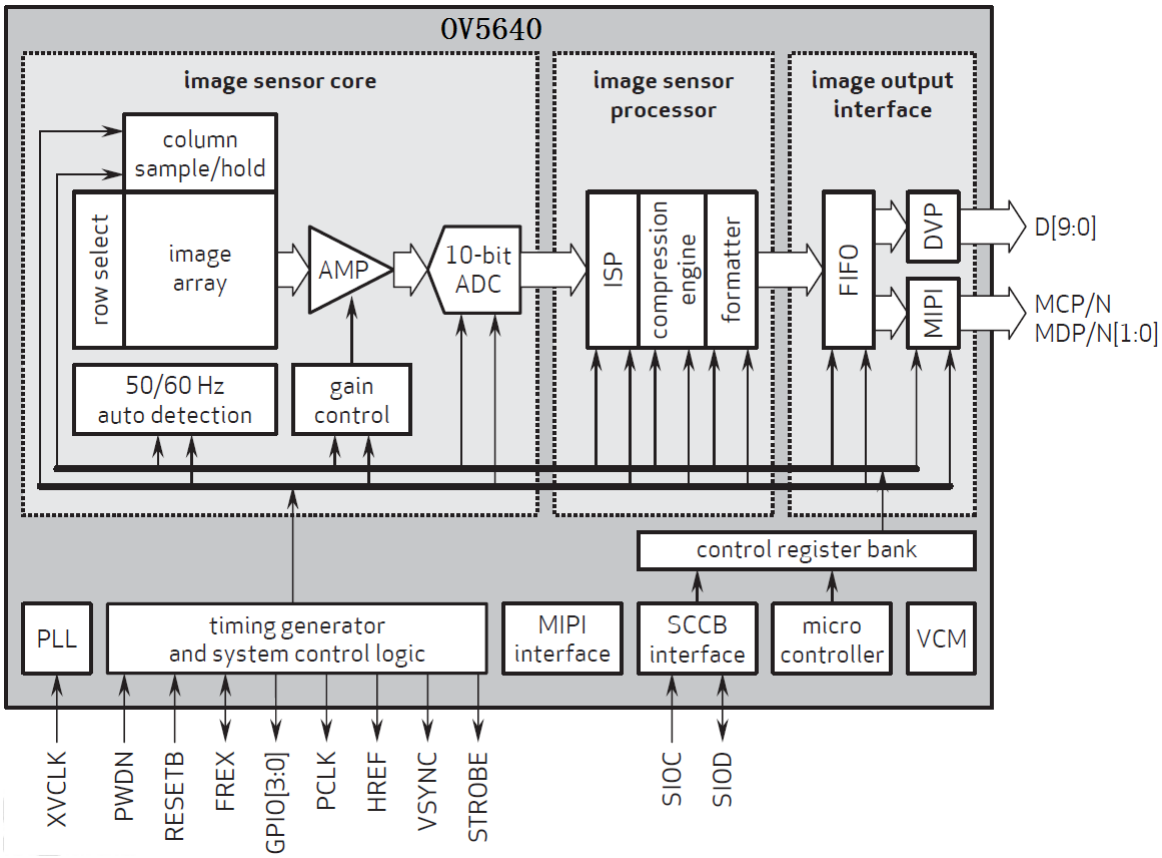

图 1.1 OV5640 功能框图

其中 image array 部分的尺寸，OV5640 的官方数据并没有给出具体的数字，其最大的有效输出尺寸为：2592*1944，即 500W 像素，我们根据官方提供的一些应用文档，发现其设置的 image array 最大为：2632*1951，所以，在接下来的介绍，我们设定其 image array 最大为 2632*1951。

1.1 DVP 接口说明

OV5640 支持数字视频接口（DVP）和 MIPI 接口，因为我们的 STM32F429 使用的 DCMI 接口，仅支持 DVP 接口，所以，OV5640 必须使用 DVP 输出接口，才可以连接我们的阿波罗 STM32 开发板。

OV5640 提供一个 10 位 DVP 接口(支持 8 位接法)，其 MSB 和 LSB 可以程序设置先后顺序，ALIENTEK OV5640 模块采用默认的 8 位连接方式，如图 1.2 所示：

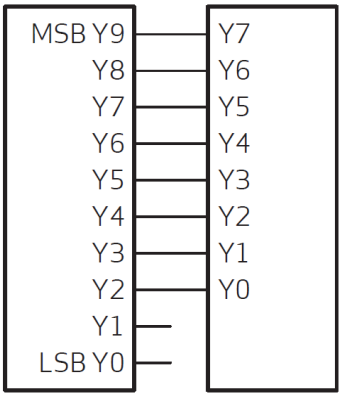

图 1.2 OV5640 默认 8 位连接方式

OV5640 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。

1.2 窗口设置说明

接下来，我们介绍一下 OV5640 的：ISP（Image Signal Processor）输入窗口设置、预缩放窗口设置和输出大小窗口设置，这几个设置与我们的正常使用密切相关，有必要了解一下。他们的设置关系，如图 1.3 所示：

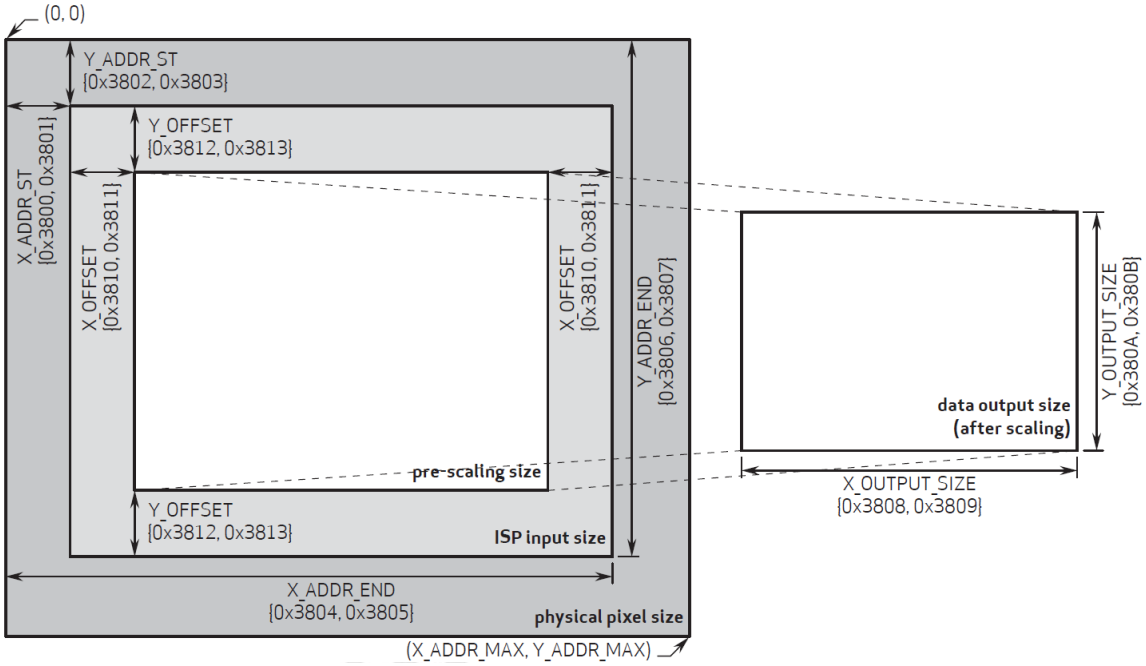

图 1.3 OV5640 各窗口设置关系

ISP 输入窗口设置（ISP input size）

该设置允许用户设置整个传感器区域（physical pixel size ，2632*1951）的感兴趣部分，也就是在传感器里面开窗（X_ADDR_ST、Y_ADDR_ST、X_ADDR_END 和 Y_ADDR_END），开窗范围从 0*0~2632*1951 都可以设置，该窗口所设置的范围，将输入 ISP 进行处理。

ISP 输入窗口，通过：0X3800~0X3807 等 8 个寄存器进行设置，这些寄存器的定义请看：OV5640_CSP3_DS_2.01_Ruisipusheng.pdf 这个文档（下同）。

预缩放窗口设置（pre-scaling size）

该设置允许用户在 ISP 输入窗口的基础上，再次设置将要用于缩放的窗口大小。该设置仅在 ISP 输入窗口内进行 x/y 方向的偏移（X_OFFSET/Y_OFFSET）。通过：0X3810~0X3813 等 4 个寄存器进行设置。

输出大小窗口设置（data output size）

该窗口是以预缩放窗口为原始大小，经过内部 DSP 进行缩放处理后，输出给外部的图像窗口大小。它控制最终的图像输出尺寸（X_OUTPUT_SIZE/Y_OUTPUT_SIZE）。通过：0X3808~0X380B 等 4 个寄存器进行设置。注意：当输出大小窗口与预缩放窗口比例不一致时，图像将进行缩放处理（会变形），仅当两者比例一致时，输出比例才是 1:1（正常）。

图 42.1.4 中，右侧 data output size 区域，才是 OV5640 输出给外部的图像尺寸，也就是显示在 LCD 上面的图像大小。输出大小窗口与预缩放窗口比例不一致时，会进行缩放处理，在 LCD 上面看到的图像将会变形。

1.3 输出时序说明

接下来，我们介绍一下 OV5640 的图像数据输出时序。首先我们简单介绍一些定义：

QSXGA，这里指：分辨率为 2592*1944 的输出格式，类似的还有：QXGA(2048*1536)、UXGA(1600*1200) 、 SXGA(1280*1024) 、 WXGA+(1440*900) 、 WXGA(1280*800) 、XGA(1024*768)、SVGA(800*600)、VGA(640*480)、QVGA(320*240)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。

OV5640 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 1.4 所示：

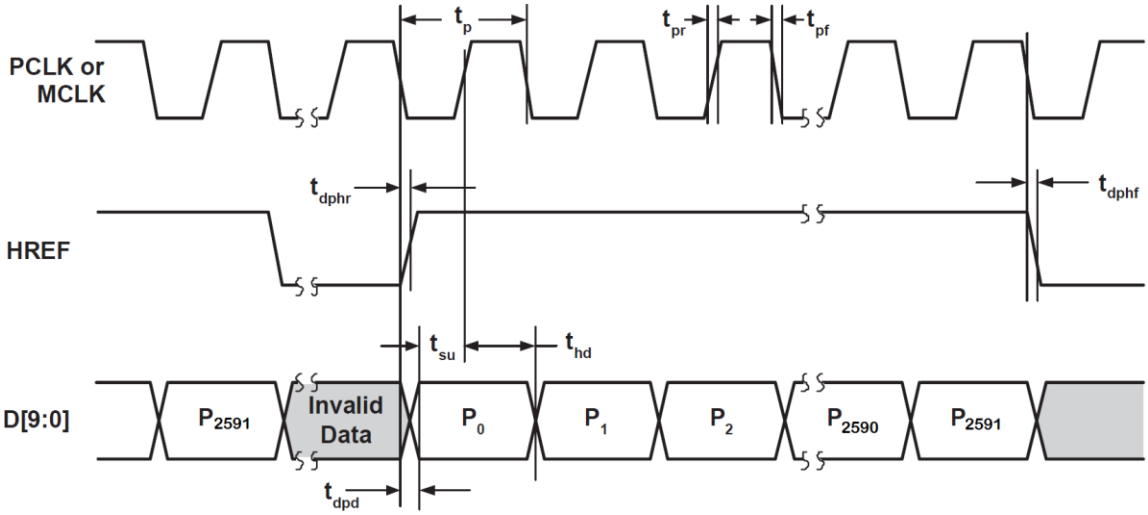

图 1.4 OV5640 行输出时序

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 QSXGA 时序，RGB565 格式输出，每 2 个字节组成一个像素的颜色（低字节在前，高字节在后），这样每行输出总共有 2592*2 个 PCLK 周期，输出 2592*2 个字节。

再来看看帧时序（QSXGA 模式），如图 1.5 所示：

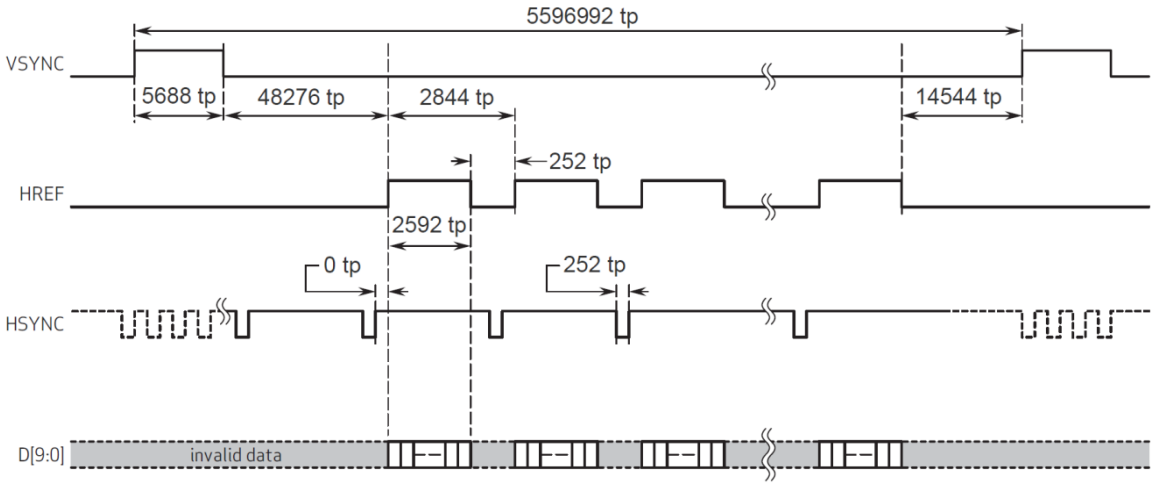

图 1.5 OV5640 帧时序

上图清楚的表示了 OV5640 在 QSXGA 模式下的数据输出。我们按照这个时序去读取 OV5640 的数据，就可以得到图像数据。

1.4 自动对焦（Auto Focus）说明

OV5640 由内置微型控制器完成自动对焦，并且 VCM（Voice Coil Motor，即音圈马达）驱动器也已集成在传感器内部。微型控制器的控制固件（firmware）从主机下载。当固件运行后，内置微型控制器从 OV5640 传感器读得自动对焦所需的信息，计算并驱动 VCM 马达带动镜头到达正确的对焦位置。主机可以通过 IIC 命令控制微型控制器的各种功能。

OV5640 的自动对焦命令（通过 SCCB 总线发送），如表 1.6 所示：

地址	寄存器名	描述	值
0X3022	CMD_MAIN	AF 主命令寄存器	0X03：触发单次自动对焦过程 0X04：启动持续自动对焦过程 0X06：暂停自动对焦过程 0X08：释放马达回到初始状态 0X12：设置对焦区域 0X00：命令完成
0X3023	CMD_ACK	命令确认	0X00：命令完成 0X01：命令执行中
0X3029	FW_STATUS	对焦状态	0X7F：固件下载完成，但未执行，可能是：固件有问题/微控制器关闭 0X7E：固件初始化中 0X70：释放马达，回到初始状态 0X00：正在自动对焦 0X10：自动对焦完成

表 1.6 OV5640 自动对焦命令

OV5640 内部的微控制器收到自动对焦命令后会自动将 CMD_MAIN（0X3022）寄存器数据清零，当命令完成后会将 CMD_ACK（0X3023）寄存器数据清零。

自动对焦（AF）过程

① 在第一次进入图像预览的时候（图像可以正常输出时），下载固件（firmware）

② 拍照前，自动对焦，对焦完成后，拍照

③ 拍照完毕，释放马达到初始状态

接下来，我们分别说明。

① 下载固件

OV5640 初始化完成后，就可以下载 AF 自动对焦固件了，其操作和下载初始化参数类似，AF 固件下载地址为：0X8000，初始化数组由厂家提供（本例程该数组保存在 ov5640af.h 里面），下载固件完成后，通过检查 0X3029 寄存器的值，来判断固件状态（等于 0X70，说明正常）。

② 自动对焦

OV5640 支持单次自动对焦和持续自动对焦，通过 0X3022 寄存器控制。单次自动对焦过程如下：

1，将 0X3022 寄存器写为 0X03，开始单点对焦过程。

2，读取寄存器 0X3029，如果返回值为 0X10，代表对焦已完成。

3，写寄存器 0X3022 为 0X06，暂停对焦过程，使镜头将保持在此对焦位置。

其中，前两步是必须的，第三步，可以不要，因为单次自动对焦完成以后，就不会继续自动对焦了，镜头也就不会动了。

持续自动对焦过程如下：

1， 将 0X22 寄存器写为 0X08，释放马达到初始位置（对焦无穷远）。

2， 将 0X3022 寄存器写为 0X04，启动持续自动对焦过程。

3， 读取寄存器 0X3023，等待命令完成。

4， 当 OV5640 每次检测到失焦时，就会自动进行对焦（一直检测）。

③ 释放马达，结束自动对焦

最后，在拍照完成，或者需要结束自动对焦的时候，我们对在寄存器 0X3022 写入 0X08，即可释放马达，结束自动对焦。

最后说一下 OV5640 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。

OV5640 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。OV5640 我们就介绍到这，关于 OV5640 更详细的介绍，请大家参考：**A 盘→7，硬件资料→OV5640 资料→ OV5640_CSP3_DS_2.01_Ruisipusheng.pdf**。

2、STM32F407 DCMI 接口简介

STM32F407 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)。

STM32F429 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]），用于接摄像头的数据输出，接 OV5640 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。

STM32F407 DCMI 接口的框图如图 2.1 所示：

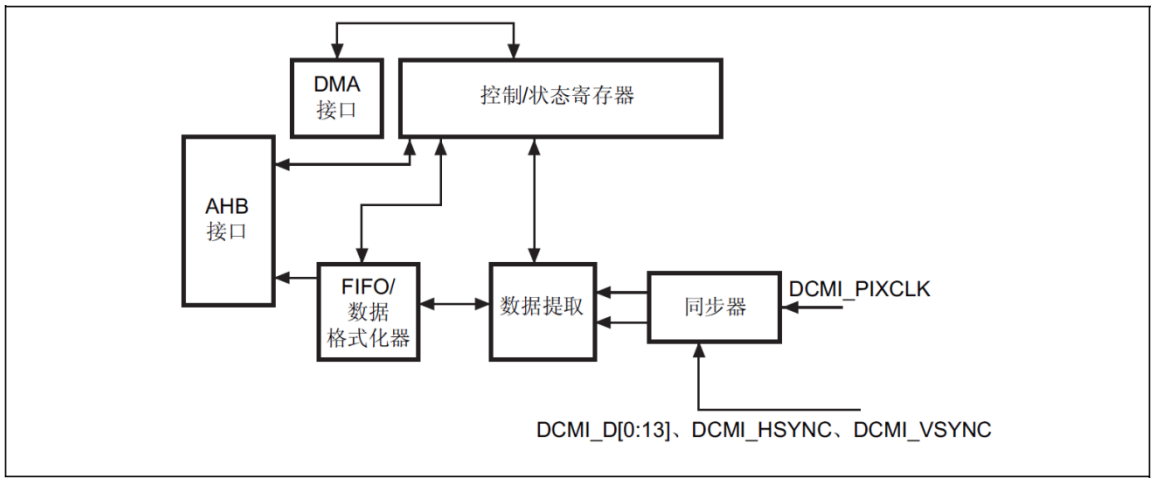

图 2.1 DCMI 接口框图

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 2.2 所示：

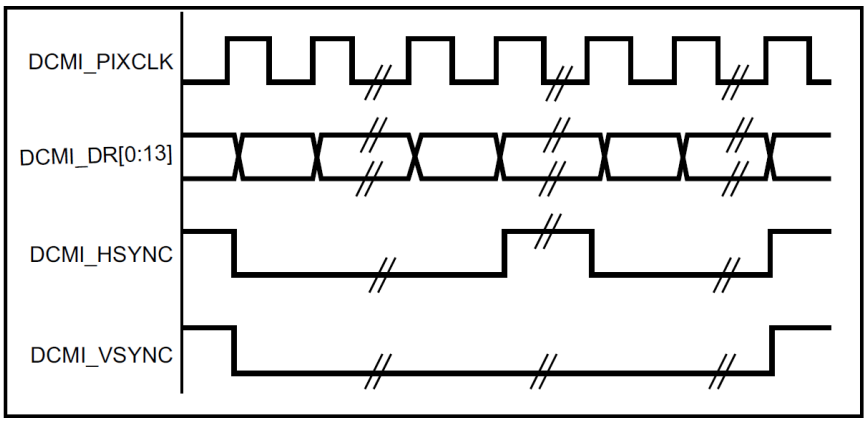

图 2.2 DCMI 信号波形

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 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 2.1 8 位捕获数据在 32 位字中的排布

从表 2.1 可以看出，STM32F407 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《STM32F4xx 中文参考手册》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《STM32F4xx 中文参考手册》第 13 章相关内容。

本章，我们将使用 STM32F407ZGT6 的 DCMI 接口连接 ALIENTEK OV5640 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，模组支持自动对焦功能，且支持闪光灯，整个模块只需提供 3.3V 供电即可正常使用。

ATK-OV5640 摄像头模块外观如图 2.3 所示：

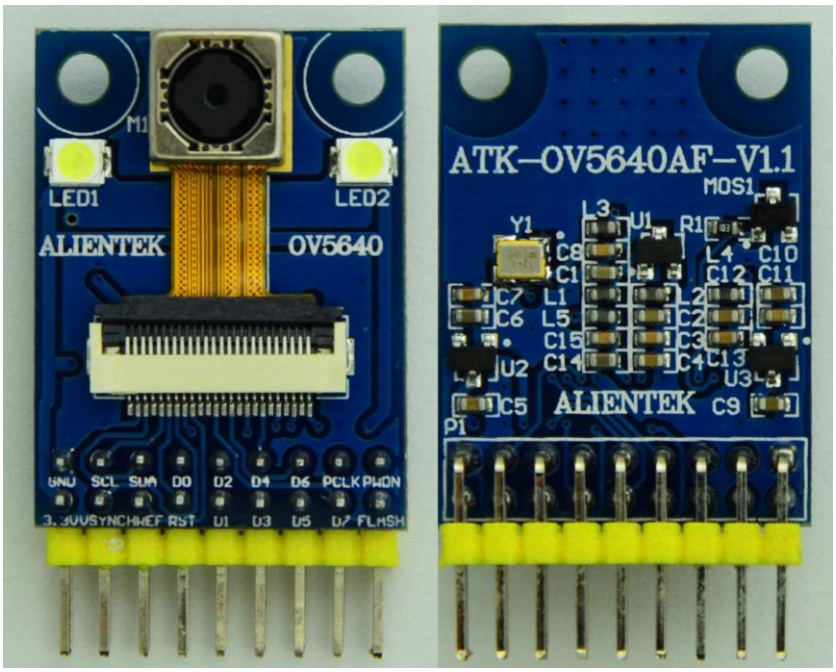

图 2.3 ATK-OV5640 摄像头模块外观图

模块原理图如图 2.4 所示：

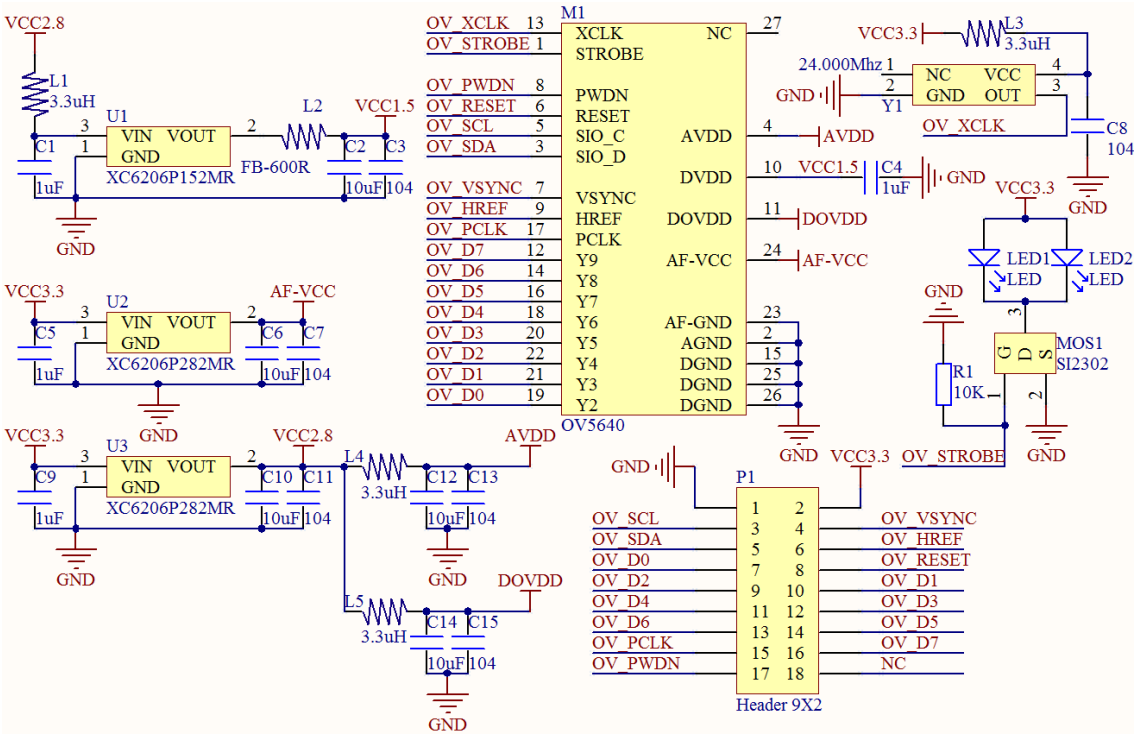

图 2.4 ALIENTEK OV5640 摄像头模块原理图

从上图可以看出，ALIENTEK OV5640 摄像头模块自带了有源晶振，用于产生 24M 时钟作为 OV5640 的 XCLK 输入，模块的闪光灯（LED1&LED2）由 OV5640 的 STROBE 脚控制（可编程控制）。同时自带了稳压芯片，用于提供 OV5640 稳定的 2.8V 和 1.5V 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 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 2.2 OV5640 模块信号及其作用描述

我们将 OV5640 默认配置为 WXGA 输出，也就是 1280*800 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，STM32F429 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 2.5 所示：

31–15	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
	rw		rw rw	rw rw	rw	rw	rw	rw	rw	rw	rw	rw

图 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 OV5640 的步骤：

1）配置 OV5640 控制引脚，并配置 OV5640 工作模式。

在启动 DCMI 之前，我们先设置好 OV5640。OV5640 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 OV5640，之后再设置 OV_RESET 为 1，结束复位，然后就是对 OV5640 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。

OV5640 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 OV5640 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 OV5640 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。

3、硬件连接

本章实验功能简介：开机后，初始化摄像头模块（OV5640），如果初始化成功，则提示选择模式：RGB565 模式，或者 JPEG 模式。KEY0 用于选择 RGB565 模式，KEY1 用于选择 JPEG 模式。

当使用 RGB565 时，输出图像（固定为：WXGA）将经过缩放处理（完全由 OV5640 的 DSP 控制），显示在 LCD 上面（默认开启连续自动对焦）。我们可以通过 KEY_UP 按键选择：1:1 显示，即不缩放，图片不变形，但是显示区域小（液晶分辨率大小），或者缩放显示，即

将 1280*800 的图像压缩到液晶分辨率尺寸显示，图片变形，但是显示了整个图片内容。通过 KEY0 按键，可以设置对比度；KEY1 按键，可以启动单次自动对焦；KEY2 按键，可以设置特效。

当使用 JPEG 模式时，图像可以设置尺寸（QQVGA~VGA），采集到的 JPEG 数据将先存放到 STM32F407 的 RAM 内存里面，每当采集到一帧数据，就会关闭 DMA 传输，然后将采集到的数据发送到串口 2（此时可以通过上位机软件（ATK-CAM.exe）接收，并显示图片），之后再重新启动 DMA 传输。我们可以通过 KEY_UP 设置输出图片的尺寸（QQVGA~VGA）。通过 KEY0 按键，可以设置对比度；KEY1 按键，可以启动单次自动对焦；KEY2 按键，可以设置特效。

同时时可以通过串口 1，借助 USMART 设置/读取 OV5640 的寄存器，方便大家调试。DS0 指示程序运行状态，DS1 用于指示帧中断。

所要用到的硬件资源如下：

1） 指示灯 DS0 和 DS1

2） 4 个按键

3） 串口 1 和串口 2

4） LCD 模块

5） OV5640 摄像头模块

这些资源，基本上都介绍过了，这里我们用到串口 2 来传输 JPEG 数据给上位机，其配置同串口 1 几乎一模一样，只是串口 2 的时钟来自 APB1，频率为 42Mhz。发板板载的摄像头模块接口与 MCU 的连接如图 3.1 所示：

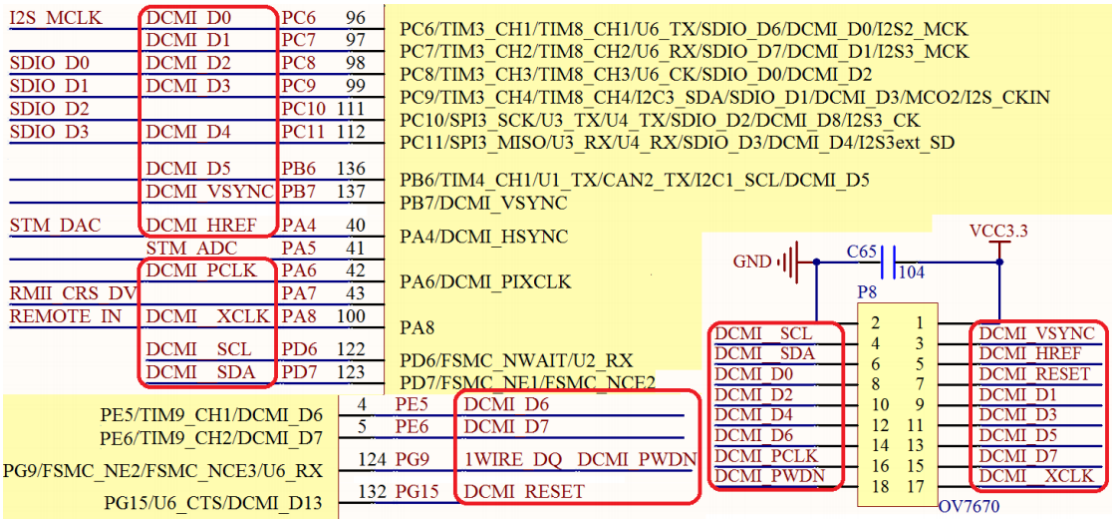

图 3.1 摄像头模块接口与 STM32 连接图

从上图可以看出， OV5640 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 PA4;

DCMI_PCLK 接 PA6;

DCMI_SCL 接 PD6;

DCMI_SDA 接 PD7;

DCMI_RESET 接 PG15;

DCMI_PWDN 接 PG9;

DCMI_XCLK 接 PA8（本章未用到） ;

DCMI_D[7:0]接 PE6/PE5/PB6/PC11/PC9/PC8/PC7/PC6;

这些线的连接， 探索者 STM32F4 开发板的内部已经连接好了，我们只需要将 OV5640 摄像头模块插上去就好了。特别注意： DCMI 摄像头接口和 I2S 接口、DAC、SDIO 以及 1WIRE_DQ 等有冲突，使用的时候，必须分时复用才可以，不可同时使用。 实物连接如图 40.2.2 所示：

图 3.2 OV5640 摄像头模块与开发板连接实物图

3、软件实现

本实验，我们在探索者标准例程 35：**摄像头实验**的基础上修改，本例程我们用的是 OV5640 所以我们删除工程的 ov2640.c。

然后，在 HARDWARE 文件夹里面新建一个 OV5640 文件夹，并新建 ov5640.c，ov5640.h，ov5640af.h，ov5640cfg.h 四个文件。然后在工程 HARDWARE 组里面添加 ov5640.c，并在工程添加 ov5640.h，ov5640af.h，ov5640cfg.h 的头文件包含路径。

由于代码比较多，我们这里就不给大家介绍所有代码了，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 ov5640.c 里面的 OV5640_Init 函数，该函数代码如下：

```
//初始化 OV5640
//返回值:0,成功
//       其他,错误代码
u8 OV5640_Init(void)
{
    u16 i=0;
    u16 reg;
    //设置 IO
    RCC->AHB1ENR|=1<<6;          //使能外设 PORTG 时钟

GPIO_Set(GPIOG,PIN9|PIN15,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_50M
,GPIO_PUPD_PU);                  //PG9,15 推挽输出
    OV5640_RST=0;                //必须先拉低 OV5640 的 RST 脚,再上电
```

```
        delay_ms(20);
        OV5640_PWDN=0;          //POWER ON
        delay_ms(5);
        OV5640_RST=1;           //结束复位
        delay_ms(20);
        SCCB_Init();            //初始化 SCCB 的 IO 口
        delay_ms(5);
        reg=OV5640_RD_Reg(OV5640_CHIPIDH);      //读取 ID 高八位
        reg<<=8;
        reg|=OV5640_RD_Reg(OV5640_CHIPIDL);     //读取 ID 低八位
        if(reg!=OV5640_ID)
        {
            printf("ID:%d\r\n",reg);
            return 1;
        }
        OV5640_WR_Reg(0x3103,0X11);     //system clock from pad, bit[1]
        OV5640_WR_Reg(0X3008,0X82);     //软复位
        delay_ms(10);
        for(i=0;i<sizeof(ov5640_init_reg_tbl)/4;i++)      //初始化 OV5640
        {
            OV5640_WR_Reg(ov5640_init_reg_tbl[i][0],ov5640_init_reg_tbl[i][1]);
        }
        //检查闪光灯是否正常
        OV5640_Flash_Ctrl(1);               //打开闪光灯
        delay_ms(50);
        OV5640_Flash_Ctrl(0);               //关闭闪光灯
        return 0x00;    //ok
    }
```

此部分代码先初始化 OV5640 相关的 IO 口（包括 SCCB_Init），然后最主要的是完成 OV5640 的寄存器序列初始化。OV5640 的寄存器很多（一百几十个），配置比较麻烦，幸好厂家有提供参考配置序列《OV5640_camera_module_software_application_notes_1.3_Sonix.pdf,》），本章我们用到的配置序列，存放在 ov5640_ 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 ov5640cfg.h 里面。

另外，在 ov5640.c 里面，还有几个函数比较重要，这里不贴代码了，只介绍功能：

OV5640_ImageWin_Set 函数，该函数用于设置 ISP 输入窗口；

OV5640_OutSize_Set 函数，用于设置预缩放窗口和输出大小窗口；

OV5640_Focus_Init 函数，用于初始化自动对焦功能；

OV5640_Focus_Single 函数，用于实现一次自动对焦；

OV5640_Focus_Constant 函数，用于开启持续自动对焦功能；

OV5640_ImageWin_Set 和 OV5640_OutSize_Set 这就是我们在 1.2 节所介绍的 3 个窗口的设置，他们共同决定了图像的输出。

接下来，我们看看 ov5640cfg.h 里面 ov5640_init_reg_tbl 的内容，ov2640cfg.h 文件的代码如下：

```
//JPEG 配置.7.5 帧
```

```
//最大支持 2592*1944 的 JPEG 图像输出
const u16 OV5640_jpeg_reg_tbl[][2]=
{
    0x4300, 0x30, // YUV 422, YUYV
    ……//省略部分代码
    0x3503, 0x00, // AEC/AGC on
};
//RGB565 配置.15 帧
//最大支持 1280*800 的 RGB565 图像输出
const u16 ov5640_rgb565_reg_tbl[][2]=
{
    0x4300, 0X6F,
    ……//省略部分代码
    0x3503, 0x00, // AEC/AGC on
};
//OV5640 初始化寄存器序列表
const u16 ov5640_init_reg_tbl[][2]=
{
    // 24MHz input clock, 24MHz PCLK
    0x3008, 0x42, // software power down, bit[6]
    ……//省略部分代码
    0x4740, 0X21, //VSYNC  高有效
};
```

以上代码，我们省略了很多（全部贴出来太长了），里面总共有 3 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{0x4300, 0x30}，就表示在 0x4300 地址，写入 0X30 这个值。

这里面：ov5640_init_reg_tbl 数组，用于初始化 OV5640，该数组必须最先进行配置；ov5640_rgb565_reg_tbl 数组，用于设置 OV5640 的输出格式为 RGB565，分辨率为 1280*800，帧率为 15 帧，在 RGB 模式下使用；OV5640_jpeg_reg_tbl 用于设置 OV5640 的输出格式为 JPEG，**分辨率为 2592*1944**，帧率为 7.5 帧，在 JPEG 模式下使用。

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
//DCMI 中断服务函数
void DCMI_IRQHandler(void)
{
    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=!LED1;
        ov_frame++;
    }
}
```

```
//DCMI DMA 配置
//memaddr:存储器地址       将要存储摄像头数据的内存地址(也可以是外设地址)
//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 memaddr,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=memaddr;         //memaddr 作为目标地址
    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
}
//DCMI 初始化
void DCMI_Init(void)
{
    //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<<4;             //使能外设 PORTE 时钟
    RCC->AHB2ENR|=1<<0;             //能 DCMI 时钟

    GPIO_Set(GPIOA,PIN4|PIN6,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_10
0M,GPIO_PUPD_PU);  //PA4/6     复用功能输出
    GPIO_Set(GPIOB,PIN6|PIN7,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_10
0M,GPIO_PUPD_PU);  //PB6/7     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(GPIOE,PIN5|PIN6,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_10
```

```
0M,GPIO_PUPD_PU);  //PE5/6   复用功能输出

        GPIO_AF_Set(GPIOA,4,13);      //PA4,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(GPIOB,6,13);      //PB6,AF13   DCMI_D5
        GPIO_AF_Set(GPIOE,5,13);      //PE5,AF13   DCMI_D6
        GPIO_AF_Set(GPIOE,6,13);      //PE6,AF13   DCMI_D7
        //清除原来的设置
        DCMI->CR=0x0;
        DCMI->IER=0x0;
        DCMI->ICR=0x1F;
        DCMI->ESCR=0x0;
        DCMI->ESUR=0x0;
        DCMI->CWSTRTR=0x0;
        DCMI->CWSIZER=0x0;

        DCMI->CR|=0<<1;           //连续模式
        DCMI->CR|=0<<2;           //全帧捕获
        DCMI->CR|=0<<4;           //硬件同步 HSYNC,VSYNC
        DCMI->CR|=1<<5;           //PCLK 上升沿有效
        DCMI->CR|=0<<6;           //HSYNC 低电平有效
        DCMI->CR|=1<<7;           //VSYNC 高电平有效
        DCMI->CR|=0<<8;           //捕获所有的帧
        DCMI->CR|=0<<10;          //8 位数据格式
        DCMI->IER|=1<<0;          //开启帧中断
        DCMI->CR|=1<<14;          //DCMI 使能
        MY_NVIC_Init(2,3,DCMI_IRQn,2);        //抢占 2，子优先级 3，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);
    OV5640_OutSize_Set(4,0,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。DCMI_Init 函数用于初始化 STM32F407 的 DCMI 接口。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细说了，请大家参考本例程源码。

最后，打开 test.c 文件，修改代码如下：

```
u8 ovx_mode=0;                                  //bit0:0,RGB565 模式;1,JPEG 模式

#define jpeg_buf_size     31*1024       //定义 JPEG 数据缓存 jpeg_buf 的大小(*4 字节)
__align(4) u32 jpeg_buf[jpeg_buf_size];     //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

    //JPEG 尺寸支持列表
    const u16 jpeg_img_size_tbl[][2]=
    {
        160,120,  //QQVGA
        320,240,  //QVGA
        640,480,  //VGA
        800,600,  //SVGA
        1024,768,     //XGA
        1280,800,     //WXGA
        1440,900,     //WXGA+
        1280,1024,    //SXGA
        1600,1200,    //UXGA
        1920,1080,    //1080P
        2048,1536,    //QXGA
        2592,1944,    //500W
    };

    const u8*EFFECTS_TBL[7]={"Normal","Cool","Warm","B&W","Yellowish
","Inverse","Greenish"};//7 种特效
    const u8*JPEG_SIZE_TBL[12]={“QQVGA”,”QVGA”,”VGA”,”SVGA”,
    ”XGA”,”WXGA”,”WXGA+”,”SXGA”,”UXGA”,”1080P”,”QXGA”,
    “500W”};//JPEG 图片 12 种尺寸
    //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)
    {
        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
                jpeg_data_len=jpeg_buf_size-DMA2_Stream1->NDTR;
                //得到此次数据传输的长度
                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_buf_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
            }
        }else
        {
            LCD_SetCursor(0,0);
            LCD_WriteRAM_Prepare();     //开始写入 GRAM
        }
    }
    //JPEG 测试
    //JPEG 数据,通过串口 2 发送给电脑.
    void jpeg_test(void)
    {
        u32 i,jpgstart,jpglen;
        u8 *p;
        u8 key,headok=0;
        u8 effect=0,contrast=2;
        u8 size=2;                  //默认是 QVGA 640*480 尺寸
        u8 msgbuf[15];              //消息缓存区
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
        LCD_ShowString(30,50,200,16,16,"ALIENTEK STM32F4");
        LCD_ShowString(30,70,200,16,16,"OV5640 JPEG Mode");
        LCD_ShowString(30,100,200,16,16,"KEY0:Contrast");          //对比度
        LCD_ShowString(30,120,200,16,16,"KEY1:Auto Focus");        //执行自动对焦
        LCD_ShowString(30,140,200,16,16,"KEY2:Effects");           //特效
        LCD_ShowString(30,160,200,16,16,"KEY_UP:Size");            //分辨率设置
        sprintf((char*)msgbuf,"JPEG Size:%s",JPEG_SIZE_TBL[size]);
        LCD_ShowString(30,180,200,16,16,msgbuf);                   //显示当前 JPEG 分辨率
        //自动对焦初始化
        OV5640_RGB565_Mode();       //RGB565 模式
        OV5640_Focus_Init();        //初始化自动对焦
        OV5640_JPEG_Mode();         //JPEG 模式
        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)&jpeg_buf,jpeg_buf_size,2,1);    //DCMI DMA 配置
```

```
OV5640_OutSize_Set(4,0,jpeg_img_size_tbl[size][0],jpeg_img_size_tbl[size][1]);
//设置输出尺寸
DCMI_Start();                    //启动传输
while(1)
{
    if(jpeg_data_ok==1)     //已经采集完一帧图像了
    {
        p=(u8*)jpeg_buf;
        printf("jpeg_data_len:%d\r\n",jpeg_data_len*4);        //打印帧率
        LCD_ShowString(30,210,210,16,16,"Sending JPEG data...");
        //提示正在传输数据
        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((p[i]==0XFF)&&(p[i+1]==0XD8))//找到 FF D8
            {
                jpgstart=i;
                headok=1;      //标记找到 jpg 头(FF D8)
            }
            if((p[i]==0XFF)&&(p[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
        {
            p+=jpgstart;                //偏移到 0XFF,0XD8 处
            for(i=0;i<jpglen;i++)       //发送整个 jpg 文件
            {
                while((USART2->SR&0X40)==0);      //循环发送,直到发送完毕
                USART2->DR=p[i];
                key=KEY_Scan(0);
                if(key)break;
            }
        }
        if(key)    //有按键按下,需要处理
        {
            LCD_ShowString(30,210,210,16,16,"Quit Sending data      ");
            //提示退出数据传输
            switch(key)
```

```
                    {
                        case KEY0_PRES:       //对比度设置
                            contrast++;
                            if(contrast>6)contrast=0;
                            OV5640_Contrast(contrast);
                            sprintf((char*)msgbuf,"Contrast:%d",(signed char)contrast-3);
                            break;
                        case KEY1_PRES:             //执行一次自动对焦
                            OV5640_Focus_Single();
                            break;
                        case KEY2_PRES:             //特效设置
                            effect++;
                            if(effect>6)effect=0;
                            OV5640_Special_Effects(effect);     //设置特效
                            sprintf((char*)msgbuf,"%s",EFFECTS_TBL[effect]);
                            break;
                        case WKUP_PRES:                 //JPEG 输出尺寸设置
                            size++;
                            if(size>2)size=0;               //QQVGA~VGA
                            //设置输出尺寸
        OV5640_OutSize_Set(16,4,jpeg_img_size_tbl[size][0],jpeg_img_size_tbl[size][1]);
        sprintf((char*)msgbuf,"JPEG Size:%s",JPEG_SIZE_TBL[size]);
                            break;
                    }
                    LCD_Fill(30,180,239,190+16,WHITE);
                    LCD_ShowString(30,180,210,16,16,msgbuf);//显示提示内容
                    delay_ms(800);
                }else LCD_ShowString(30,210,210,16,16,"Send data complete!!");
                //提示传输结束设置
                jpeg_data_ok=2;
                //标记 jpeg 数据处理完了,可以让 DMA 去采集下一帧了.
            }
        }
    }
    //RGB565 测试
    //RGB 数据直接显示在 LCD 上面
    void rgb565_test(void)
    {
        u8 key;
        u8 effect=0,contrast=2,fac;
        u8 scale=1;         //默认是全尺寸缩放
        u8 msgbuf[15];      //消息缓存区
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
```

```
LCD_ShowString(30,50,200,16,16,"ALIENTEK STM32F4");
LCD_ShowString(30,70,200,16,16,"OV5640 RGB565 Mode");

LCD_ShowString(30,100,200,16,16,"KEY0:Contrast");          //对比度
LCD_ShowString(30,130,200,16,16,"KEY1:Saturation");        //色彩饱和度
LCD_ShowString(30,150,200,16,16,"KEY2:Effects");           //特效
LCD_ShowString(30,170,200,16,16,"KEY_UP:FullSize/Scale");
//1:1 尺寸(显示真实尺寸)/全尺寸缩放

//自动对焦初始化
OV5640_RGB565_Mode();  //RGB565 模式
OV5640_Focus_Init();
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,1,1,0);//DCMI DMA 配置
OV5640_OutSize_Set(4,0,lcddev.width,lcddev.height);
DCMI_Start();           //启动传输
while(1)
{
    key=KEY_Scan(0);
    if(key)
    {
        if(key!=KEY1_PRES)DCMI_Stop(); //非 KEY1 按下,停止显示
        switch(key)
        {
            case KEY0_PRES: //对比度设置
                contrast++;
                if(contrast>6)contrast=0;
                OV5640_Contrast(contrast);
                sprintf((char*)msgbuf,"Contrast:%d",(signed char)contrast-3);
                break;
            case KEY1_PRES: //执行一次自动对焦
                OV5640_Focus_Single();
                break;
            case KEY2_PRES: //特效设置
                effect++;
                if(effect>6)effect=0;
                OV5640_Special_Effects(effect);//设置特效
```

```
                    sprintf((char*)msgbuf,"%s",EFFECTS_TBL[effect]);
                    break;
                case WKUP_PRES:     //1:1 尺寸(显示真实尺寸)/缩放
                    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*lcd
dev.height)/2,lcddev.width,lcddev.height);
                        sprintf((char*)msgbuf,"Full Size 1:1");
                    }else
                    {
                        OV5640_OutSize_Set(4,0,lcddev.width,lcddev.height);
                        sprintf((char*)msgbuf,"Scale");
                    }
                    break;
            }
            if(key!=KEY1_PRES)   //非 KEY1 按下
            {
                LCD_ShowString(30,50,210,16,16,msgbuf);//显示提示内容
                delay_ms(800);
                DCMI_Start();//重新开始传输
            }
        }
        delay_ms(10);
    }
}
int main(void)
{
    u8 key;
    u8 t;
    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
    usart2_init(42,921600);         //初始化串口 2 波特率为 921600
    LED_Init();                     //初始化 LED
    LCD_Init();                     //LCD 初始化
    KEY_Init();                     //按键初始化
    TIM3_Int_Init(10000-1,8400-1);//10Khz 计数,1 秒钟中断一次
    usmart_dev.init(84);            //初始化 USMART
    POINT_COLOR=RED;//设置字体为红色
    LCD_ShowString(30,50,200,16,16,"Explorer STM32F4");
    LCD_ShowString(30,70,200,16,16,"OV5640 TEST");
    LCD_ShowString(30,90,200,16,16,"ATOM@ALIENTEK");
```

```
        LCD_ShowString(30,110,200,16,16,"2016/5/30");
        while(OV5640_Init())//初始化 OV5640
        {
            LCD_ShowString(30,130,240,16,16,"OV5640 ERROR");
            delay_ms(200);
            LCD_Fill(30,130,239,170,WHITE);
            delay_ms(200);
            LED0=!LED0;
        }
        LCD_ShowString(30,130,200,16,16,"OV5640 OK");
        while(1)
        {
            key=KEY_Scan(0);
            if(key==KEY0_PRES)              //RGB565 模式
            {
                ovx_mode=0;
                break;
            }else if(key==KEY1_PRES)  //JPEG 模式
            {
                ovx_mode=1;
                break;
            }
            t++;
            if(t==100)LCD_ShowString(30,150,230,16,16,"KEY0:RGB565   KEY1:JPEG");
            //闪烁显示提示信息
            if(t==200)
            {
                LCD_Fill(30,150,210,150+16,WHITE);
                t=0;
                LED0=!LED0;
            }
            delay_ms(5);
        }
        if(ovx_mode==1)jpeg_test();
        else rgb565_test();
    }
```

这里我们定义了一个数组 jpeg_buf（124KB），用来存储 JPEG 数据，因为 STM32F407ZGT 有 192KB RAM，你可以根据你的代码定义为其他尺寸的 buf，另外我们的探索者是有外扩 1MB SRAM 的你也可以利用起来。

在 test.c 里面，总共有 4 个函数：jpeg_data_process、jpeg_test、rgb565_test 和 main 函数。其中，jpeg_data_process 函数用于处理 JPEG 数据的接收，在 DCMI_IRQHandler 函数里面被调用，通过一个 jpeg_data_ok 变量与 jpeg_test 函数共同控制 JPEG 数据传送。jpeg_test 函数将 OV5640 设置为 JPEG 模式，并开启持续对焦，该函数接收 OV5640 的 JPEG 数据，并通过串口 2 发送给上位机。rgb565_test 函数将 OV5640 设置为 RGB565 模式，并将接收到的数

据，直接传送给 LCD，处理过程完全由硬件实现，CPU 完全不用理会。最后，main 函数就相对简单了，大家看代码即可。

前面提到，我们要用 USMART 来设置摄像头的参数，我们只需要在 usmart_nametab 里面添加 OV5640_WR_Reg 和 OV5640_RD_Reg 等相关函数，就可以轻松调试摄像头了。

4、下载验证

代码编译下载成功之后，我们通过下载代码到 ALIENTEK 探索者 STM32F407 开发板上，在 OV5640 初始化成功后，屏幕提示选择模式，此时我们可以按 KEY0，进入 RGB565 模式测试，也可以按 KEY1，进入 JPEG 测试。

当按下 KEY0 后，选择 RGB565 模式，LCD 满屏显示压缩后的图像（有变形），如图 4.1 所示：

图 4.1 RGB565 模式测试图片

此时，可以按下 KEY_UP 切换为 1:1 显示（不变形）。同时还可以通过 KEY0 按键，设置对比度；KEY1 按键，执行一次自动对焦；KEY2 按键，设置特效。

当按下 KEY1 后，选择 JPEG 模式，此时屏幕上显示 JPEG 数据传输进程，如图 4.2 所示：

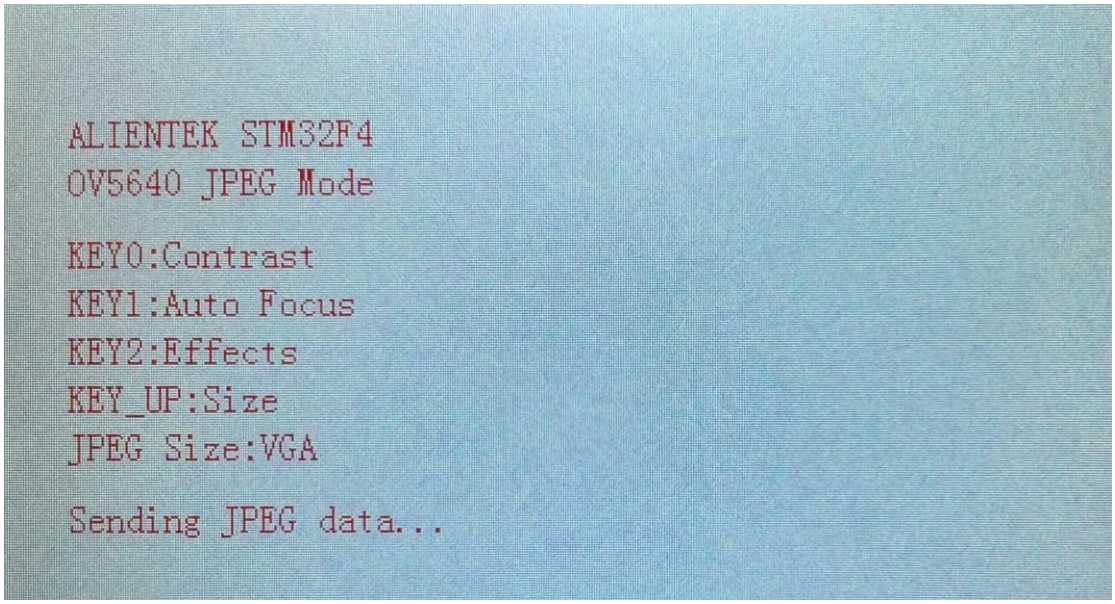

图 4.2 JPEG 模式测试图

默认条件下，图片的分辨率是 VGA（640*480）的，硬件上：我们需要一根 RS232 串口线连接开发板的 COM2(注意要用跳线帽将 P9 的:COM2_RX 连接到 PA2(TX),COM2_TX 连接到 PA3（RX））。如果没有 RS232 线，也可以借助我们开发板板载的 USB 转 TTL 实现（杜邦线连接 P9 的 PA2（TX）和 P6 的 RXD）。

我们打开上位机的软件：ATK-ACM.exe(路径： 3，配套软件→串口&网络摄像头软件→ATK-CAM.exe)，选择正确的串口，然后波特率设置为 921600，打开即可收到下位机传输过来的图片了，如图 4.3 所示：

图 4.3 ATK-CAM 软件接收并显示 JPEG 图片

我们可以通过 KEY_UP 设置输出图像的尺寸（QQVGA~VGA）。通过 KEY0 按键，设

置对比度；KEY1 按键，执行一次自动对焦；KEY2 按键，设置特效。

同时，你还可以在串口（开发板的串口 1），通过 USMART 调用 SCCB_WR_Reg 等函数，来设置 OV5640 的个寄存器，达到调试测试 OV5640 的目的，如图 4.4 所示：

图 4.4 USMART 调试 OV5640

从上图可以看出，帧率为 7/8 帧，每张 JPEG 图片的大小时 89KB 左右（分辨率为：640*480 的时候）。

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