第1章 OV7725 摄像头驱动

本章参考资料：《STM32F10x 参考手册》、《STM32F10x 数据手册》。
关于开发板配套的 OV7725 摄像头参数可查阅《ov7725datasheet》配套资料获知。

STM32 的处理速度比传统的 8、16 位机快得多，所以使用它驱动摄像头采集图像信息并进行基本的加工处理非常适合，本章讲解使用 STM32 驱动 OV7725 型号的摄像头。

1.1 摄像头简介

在各类信息中，图像含有最丰富的信息，作为机器视觉领域的核心部件，摄像头被广泛地应用在安防、探险以及车牌检测等场合。摄像头按输出信号的类型来看可以分为数字摄像头和模拟摄像头，按照摄像头图像传感器材料构成来看可以分为 CCD 和 CMOS。现在智能手机的摄像头绝大部分都是 CMOS 类型的数字摄像头。

1.1.1 数字摄像头跟模拟摄像头区别

- 输出信号类型：数字摄像头输出信号为数字信号，模拟摄像头输出信号为标准的模拟信号。
- 接口类型：数字摄像头有 usb 接口(比如常见的 pc 端免驱摄像头)、IEE1394 火线接口(由苹果公司领导的开发联盟开发的一种高速度传送接口，数据传输率高达 800Mbps)、千兆网接口（网络摄像头）。模拟摄像头多采用 AV 视频端子（信号线+地线）或 S-VIDEO（即莲花头--SUPER VIDEO，是一种五芯的接口，由两路视频亮度信号、两路视频色度信号和一路公共屏蔽地线共五条芯线组成）。
- 分辨率：模拟摄像头的感光器件，其像素指标一般维持在 752(H)*582(V)左右的水平，像素数一般情况下维持在 41W 左右。数字摄像头分辨率一般从数十万到数百万甚至数千万。但这并不能说明数字摄像头的成像分辨率就比模拟摄像头的高，原因在于模拟摄像头输出的是模拟视频信号，一般直接输入至电视或监视器，其感光器件的分辨率与电视信号的扫描数呈一定的换算关系，图像的显示介质已经确定，因此模拟摄像头的感光器件分辨率不是不能做高，而是依据于实际情况没必要做这么高。

1.1.2 CCD 与 CMOS 的区别

摄像头的图像传感器 CCD 与 CMOS 传感器主要区别如下：

- 成像材料

  CCD 与 CMOS 的名称跟它们成像使用的材料有关，CCD 是“电荷耦合器件”(Charge Coupled Device)的简称，而 CMOS 是“互补金属氧化物半导体”(Complementary Metal Oxide Semiconductor)的简称。

- 功耗

  由于 CCD 的像素由 MOS 电容构成，读取电荷信号时需使用电压相当大(至少 12V)的二相或三相或四相时序脉冲信号，才能有效地传输电荷。因此 CCD 的取像系统除了要有多个电源外，其外设电路也会消耗相当大的功率。有的 CCD 取像系统需消耗 2~5W 的功率。而 CMOS 光电传感器件只需使用一个单电源 5V 或 3V，耗电量非常小，仅为 CCD 的 1/8~1/10，有的 CMOS 取像系统只消耗 20~50mW 的功率。

- 成像质量

  CCD 传感器件制作技术起步早，技术成熟，采用 PN 结或二氧化硅(sio2)隔离层隔离噪声，所以噪声低，成像质量好。与 CCD 相比，CMOS 的主要缺点是噪声高及灵敏度低，不过现在随着 CMOS 电路消噪技术的不断发展，为生产高密度优质的 CMOS 传感器件提供了良好的条件，现在的 CMOS 传感器已经占领了大部分的市场，主流的单反相机、智能手机都已普遍采用 CMOS 传感器。

1.2 OV7725 摄像头

本章主要讲解实验板配套的摄像头，它的实物见图 1-1，该摄像头主要由镜头、图像传感器、板载电路、FIFO 缓存及下方的信号引脚组成。

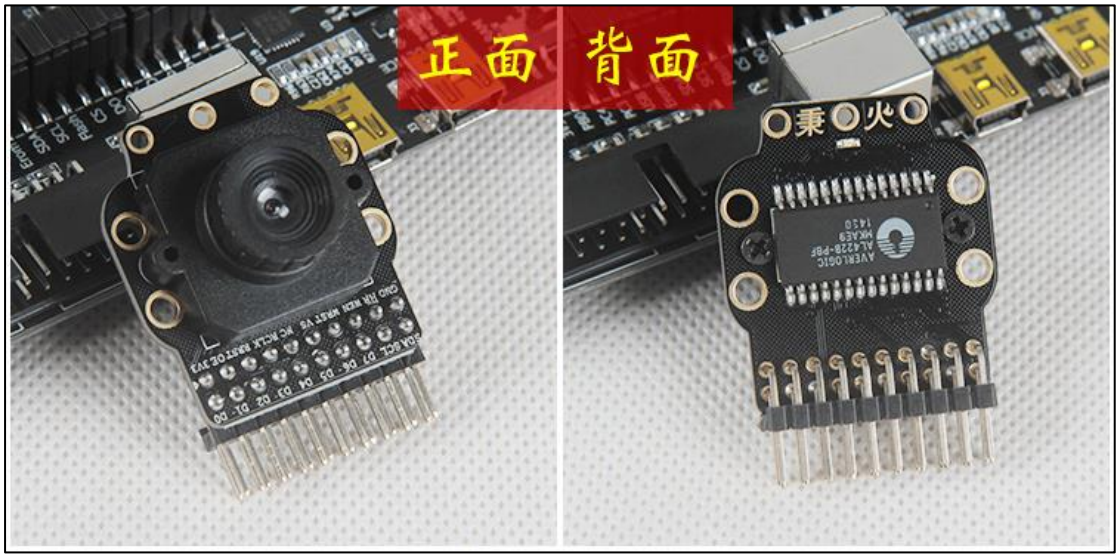

图 1-1 实验板配套的 OV7725 摄像头

镜头部件包含一个镜头座和一个可旋转调节距离的凸透镜，通过旋转可以调节焦距，正常使用时，镜头座覆盖在电路板上遮光，光线只能经过镜头传输到正中央的图像传感器，它采集光线信号，采集得的数据被缓存到摄像头背面的 FIFO 缓存中，然后外部器件通过下方的信号引脚获取拍摄得到的图像数据。

1.2.1 OV7725 传感器简介

若拆开摄像头座，在摄像头的正下方可看到 PCB 板上的一个方形器件，它是摄像头的核心部件，型号为 OV7725 的 CMOS 类型数字图像传感器。该传感器支持输出最大为 30 万像素的图像 (640x480 分辨率)，它的体积小，工作电压低，支持使用 VGA 时序输出图像数据，输出图像的数据格式支持 YUV(422/420)、YCbCr422 以及 RGB565 格式。它还可以对采集得的图像进行补偿，支持伽玛曲线、白平衡、饱和度、色度等基础处理。

1.2.2 OV7725 引脚及功能框架图

OV7725 传感器采用 BGA 封装，它的前端是采光窗口，引脚都在背面引出，引脚的分布见图 1-2。

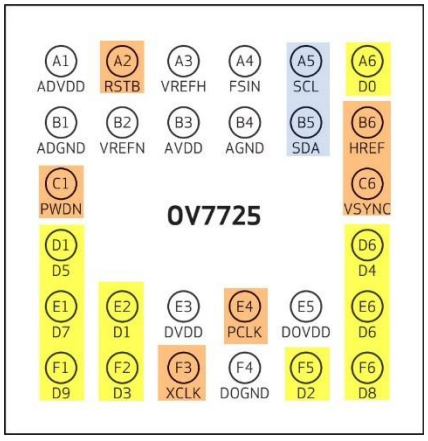

图 1-2 OV7725 管脚图

图中的非彩色部分是电源相关的引脚，彩色部分是主要的信号引脚，其介绍如表 1-1。

表 1-1 OV7725 管脚

管脚名称	管脚类型	管脚描述
RSTB	输入	系统复位管脚，低电平有效
PWDN	输入	掉电/省电模式(高电平有效)
HREF	输出	行同步信号
VSYNC	输出	场同步信号
PCLK	输出	像素时钟
XCLK	输入	系统时钟输入端口
SCL	输入	SCCB 总线的时钟线
SDA	I/O	SCCB 总线的数据线
D0…D9	输出	像素数据端口

下面我们配合图 1-3 中的 OV7725 功能框图讲解这些信号引脚。

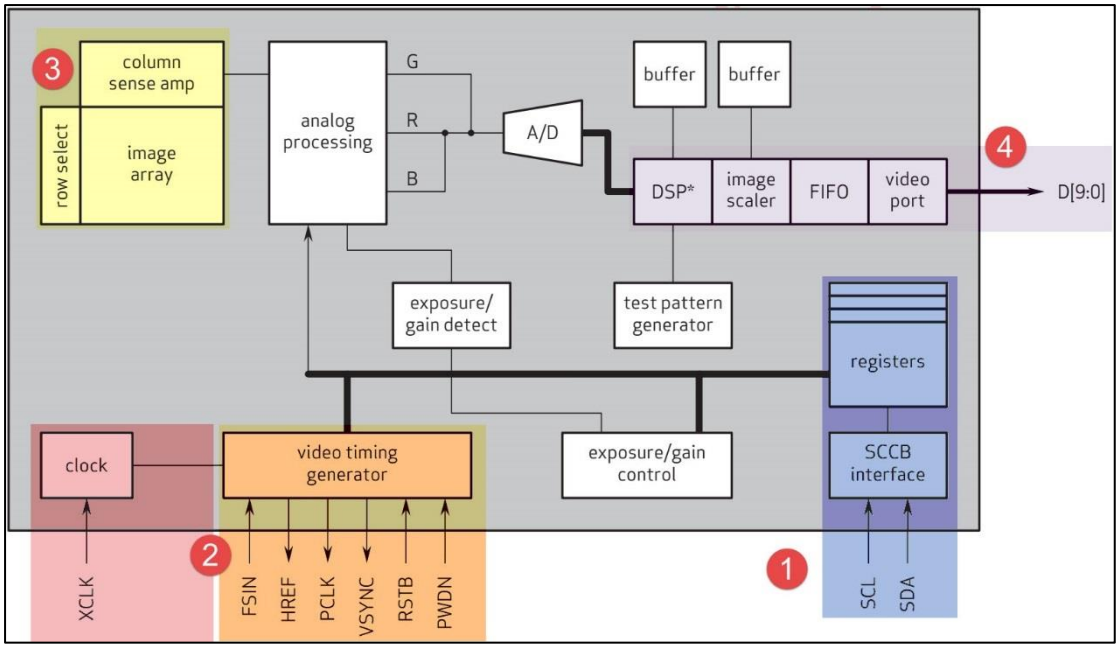

图 1-3 OV7725 功能框图

(1) 控制寄存器

标号①处的是 OV7725 的控制寄存器，它根据这些寄存器配置的参数来运行，而这些参数是由外部控制器通过 SCL 和 SDA 引脚写入的，SCL 与 SDA 使用的通讯协议 SCCB 跟 I2C 十分类似，在 STM32 中我们完全可以直接用 I2C 硬件外设来控制。

(2) 通信、控制信号及时钟

标号②处包含了 OV7725 的通信、控制信号及外部时钟，其中 PCLK、HREF 及 VSYNC 分别是像素同步时钟、行同步信号以及帧同步信号，这与液晶屏控制中的 VGA 信号是很类似的。RSTB 引脚为低电平时，用于复位整个传感器芯片，PWDN 用于控制芯片进入低功耗模式。注意最后的一个 XCLK 引脚，它跟 PCLK 是完全不同的，XCLK 是用于驱动整个传感器芯片的时钟信号，是外部输入到 OV7725 的信号；而 PCLK 是 OV7725 输出数据时的同步信号，它是由 OV7725 输出的信号。XCLK 可以外接晶振或由外部控制器提供，若要类比 XCLK 之于 OV7725 就相当于 HSE 时钟输入引脚与 STM32 芯片的关系，PCLK 引脚可类比 STM32 的 I2C 外设的 SCL 引脚。

(3) 感光矩阵

标号③处的是感光矩阵，光信号在这里转化成电信号，经过各种处理，这些信号存储成由一个个像素点表示的数字图像。

(4) 数据输出信号

标号④处包含了 DSP 处理单元，它会根据控制寄存器的配置做一些基本的图像处理运算。这部分还包含了图像格式转换单元及压缩单元，转换出的数据最终通过 D0-D9 引脚输出，一般来说我们使用 8 根据数据线来传输，这时仅使用 D2-D9 引脚。

1.2.3 SCCB 时序

外部控制器对 OV7725 寄存器的配置参数是通过 SCCB 总线传输过去的，而 SCCB 总线跟 I2C 十分类似，所以在 STM32 驱动中可以直接使用片上 I2C 外设与它通讯。关于 SCCB 协议的完整内容可查看配套资料里的《SCCB 协议》文档，下面进行简单介绍。

1. SCCB 的起始、停止信号及数据有效性

SCCB 的起始信号、停止信号及数据有效性与 I2C 完全一样，见图 1-4 及图 1-5。

- 起始信号：在 SCL（图中为 SIO_C）为高电平时，SDA(图中为 SIO_D)出现一个下降沿，则 SCCB 开始传输。
- 停止信号：在 SCL 为高电平时，SDA 出现一个上升沿，则 SCCB 停止传输。
- 数据有效性：除了开始和停止状态，在数据传输过程中，当 SCL 为高电平时，必须保证 SDA 上的数据稳定，也就是说，SDA 上的电平变换只能发生在 SCL 为低电平的时候，SDA 的信号在 SCL 为高电平时被采集。

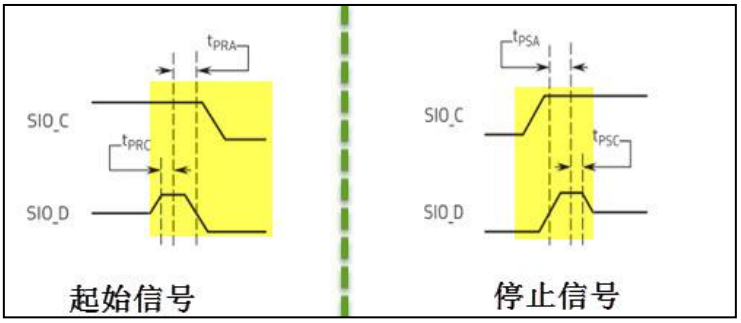

图 1-4 SCCB 停止信号

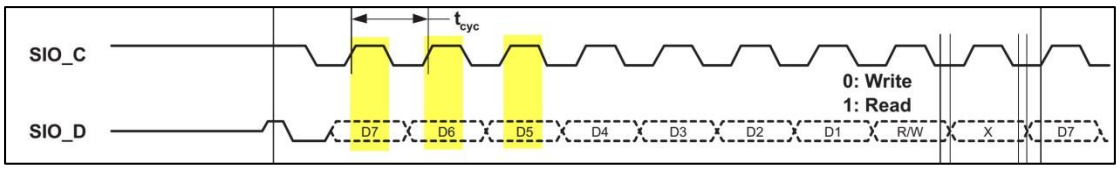

图 1-5 SCCB 的数据有效性

2. SCCB 数据读写过程

在 SCCB 协议中定义的读写操作与 I2C 也是一样的，只是换了一种说法。它定义了两种写操作，即三步写操作和两步写操作。三步写操作可向从设备的一个目的寄存器中写入数据，见图 1-6。在三步写操作中，第一阶段发送从设备的 ID 地址+W 标志(等于 I2C 的设备地址：7 位设备地址+读写方向标志)，第二阶段发送从设备目标寄存器的 8 位地址，第三阶段发送要写入寄存器的 8 位数据。图中的“X”数据位可写入 1 或 0，对通讯无影响。

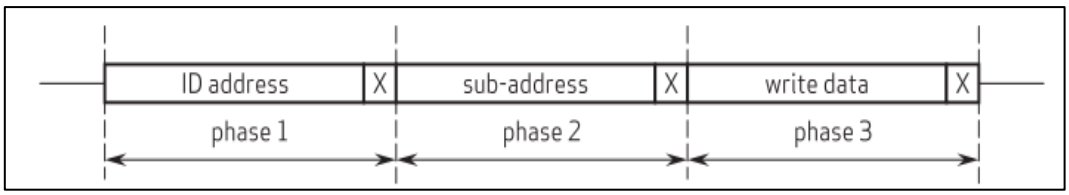

图 1-6 SCCB 的三步写操作

而两步写操作没有第三阶段，即只向从器件传输了设备 ID+W 标志和目的寄存器的地址，见图 1-7。两步写操作是用来配合后面的读寄存器数据操作的，它与读操作一起使用，实现 I2C 的复合过程。

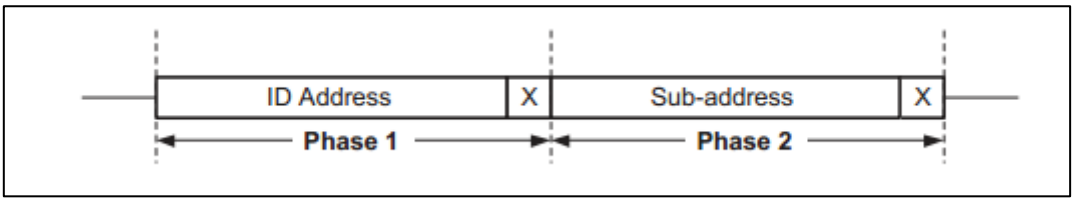

图 1-7 SCCB 的两步写操作

两步读操作，它用于读取从设备目的寄存器中的数据，见图 1-8。在第一阶段中发送从设备的设备 ID+R 标志(设备地址+读方向标志)和自由位，在第二阶段中读取寄存器中的 8 位数据和写 NA 位(非应答信号)。由于两步读操作没有确定目的寄存器的地址，所以在读操作前，必需有一个两步写操作，以提供读操作中的寄存器地址。

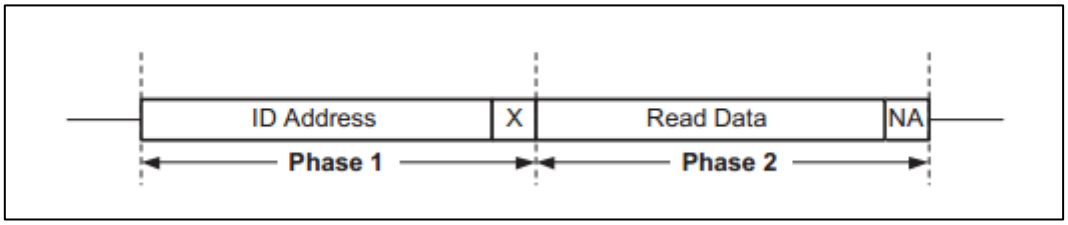

图 1-8 SCCB 的两步读操作

可以看到，以上介绍的 SCCB 特性都与 I2C 无区别，完全可以使用 STM32 的 I2C 外设来与 OV7725 进行 SCCB 通讯。

1.2.4 OV7725 的寄存器

控制 OV7725 涉及到它很多的寄存器，可直接查询《OV7725datasheet》了解，通过这些寄存器的配置，可以控制它输出图像的分辨率大小、图像格式、图像处理及图像方向等。见图 1-9。

Address (Hex)	Register Name	Default (Hex)	R/W	Description
10	AEC	40	RW	Exposure Value Bit[7:0]: AEC[7:0] (see register AECH for AEC[15:8])
11	CLKRC	80	RW	Internal Clock Bit[7]: Reserved Bit[6]: Use external clock directly (no clock pre-scale available) Bit[5:0]: Internal clock pre-scalar F(internal clock) = F(input clock)/(Bit[5:0]+1)/2 • Range: [0 0000] to [1 1111]
12	COM7	00	RW	Common Control 7 Bit[7]: SCCB Register Reset 0: No change 1: Resets all registers to default values Bit[6]: Resolution selection 0: VGA 1: QVGA Bit[5]: BT.656 protocol ON/OFF selection Bit[4]: Sensor RAW Bit[3:2]: RGB output format control 00: GBR4:2:2 01: RGB565 10: RGB555 11: RGB444 Bit[1:0]: Output format control 00: YUV 01: Processed Bayer RAW 10: RGB 11: Bayer RAW

图 1-9 0xFF=0 时的 DSP 相关寄存器说明(部分)

官方还提供了一个《OV7725 Software Application Note》的文档，它针对不同的配置需求，提供了配置范例，见图 1-10。其中 write_SCCB 是一个利用 SCCB 向寄存器写入数据的函数，第一个参数为要写入的寄存器的地址，第二个参数为要写入的内容。

```
15 fps, PCLK = 13Mhz
SCCB_salve_Address = 0x42;
write_SCCB(0x11, 0x03);
write_SCCB(0x0d, 0x41);
write_SCCB(0x2a, 0x00);
write_SCCB(0x2b, 0x00);
write_SCCB(0x33, 0x2b);
write_SCCB(0x34, 0x00);
write_SCCB(0x2d, 0x00);
write_SCCB(0x2e, 0x00);
write_SCCB(0x0e, 0x65);
```

图 1-10 调节帧率的寄存器配置范例

1.2.5 像素数据输出时序

主控器控制 OV7725 时采用 SCCB 协议读写其寄存器，而它输出图像时则使用 VGA 或 QVGA 时序，其中 VGA 在输出图像分辨率为 480*640 时采用，QVGA 是 Quarter VGA，其输出分辨率为 240*320，这些时序跟控制液晶屏输出图像数据时十分类似。

OV7725 传感器输出图像时，一帧帧地输出，在帧内的数据一般从左到右，从上到下，一个像素一个像素地输出(也可通过寄存器修改方向)，见图 1-11。

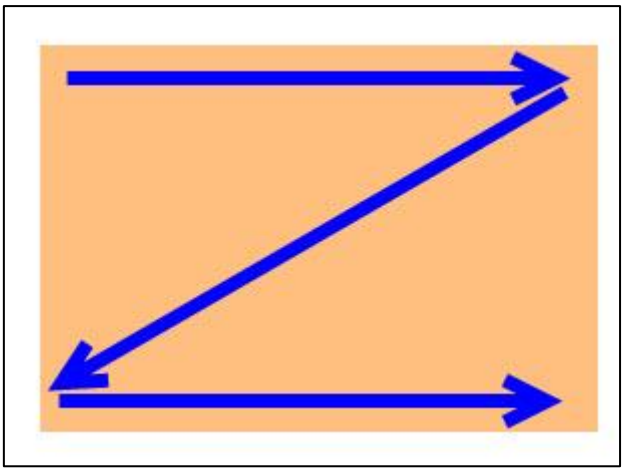

图 1-11 摄像头数据输出

例如，见图 1-12 和图 1-13，若我们使用 D2-D9 数据线，图像格式设置为 RGB565，进行数据输出时，D2-D9 数据线在 PCLK 在上升沿阶段维持稳定，并且会在 1 个像素同步时钟 PCLK 的驱动下发送 1 字节的数据信号，所以 2 个 PCLK 时钟可发送 1 个 RGB565 格式的像素数据。当 HREF 为高电平时，像素数据依次传输，每传输完一行数据时，行同步信号 HREF 会输出一个电平跳变信号间隔开当前行和下一行的数据；一帧的图像由 N 行数据组成，当 VSYNC 为低电平时，各行的像素数据依次传输，每传输完一帧图像时，VSYNC 会输出一个电平跳变信号。

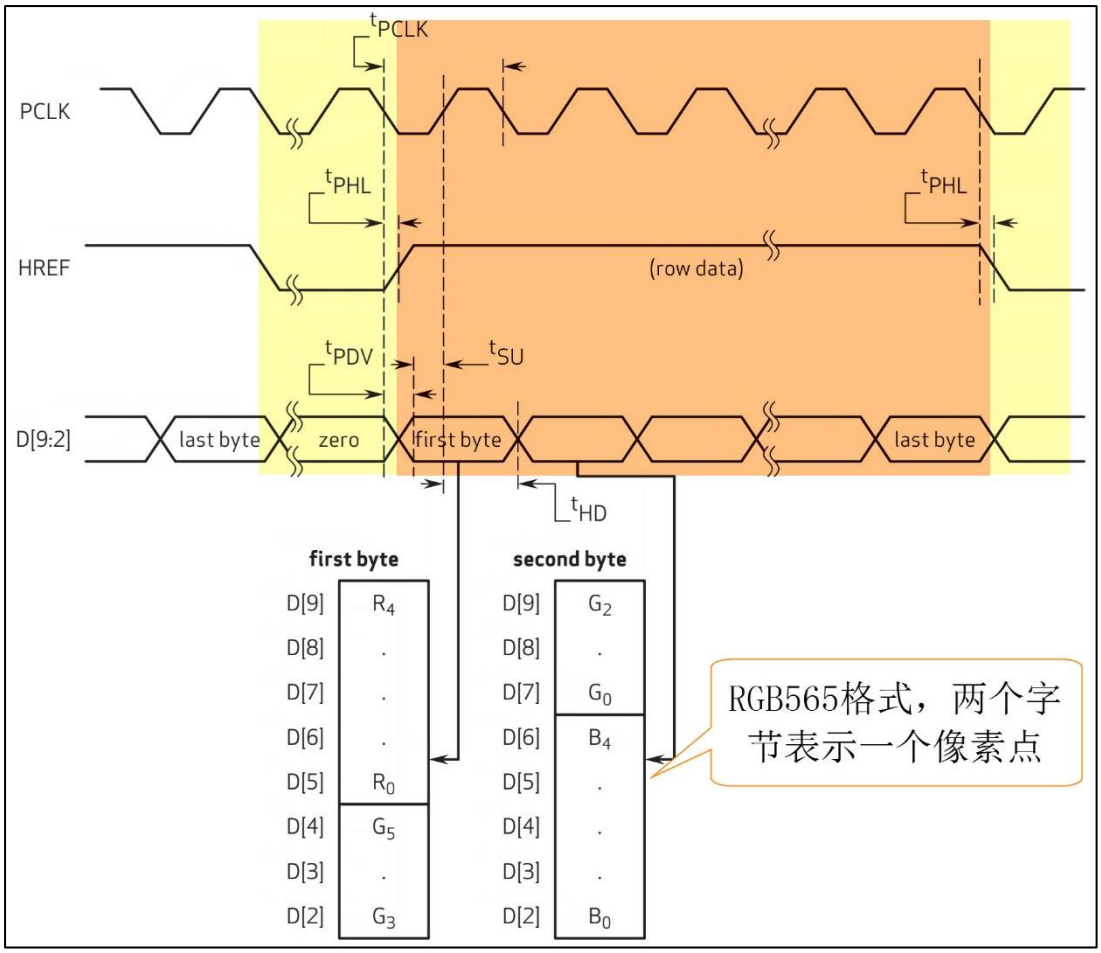

图 1-12 像素同步时序

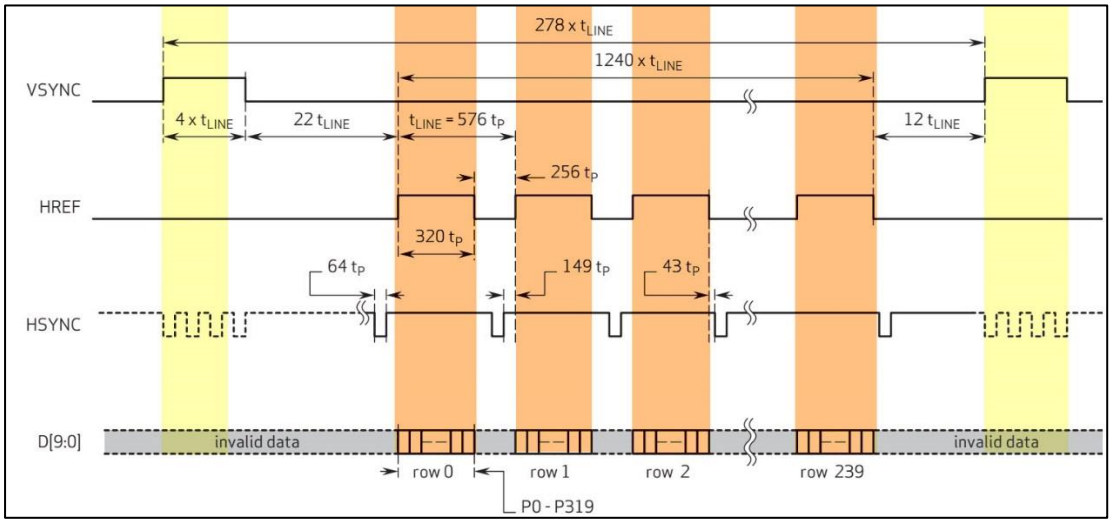

图 1-13 QVGA 帧图像同步时序

1.2.6 FIFO 读写时序

STM32F4 系列的控制器主频高、一般会扩展外部 SRAM、SDRAM 等存储器，且具有 DCMI 外设，可以直接根据 VGA 时序接收并存储摄像头输出的图像数据；而 STM32F1 系

列的控制器一般主频较低、为节省成本可能不扩展 SRAM 存储器，而且不具 DCMI 外设，难以直接接收和存储 OV7725 图像传感器输出的数据。

为了解决上述问题，针对类似 STM32F1 或更低级的控制器，秉火的 OV7725 摄像头在图像传感器之外还添加了一个型号为 AL422B 的 FIFO，用于缓冲数据。AL422B 的本质是一种 RAM 存储器，见图 1-14，它的容量大小为 393216 字节，支持同时写入和读出数据，这正是专门用于 FIFO 缓冲功能而设计的，关于它的详细说明可查阅《AL422_datasheet》文档。

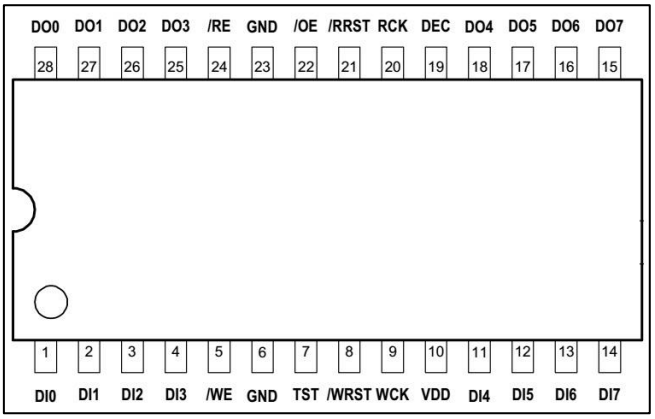

图 1-14　AL422B FIFO 引脚图

AL422B 的各引脚功能介绍见表 1-2。

表 1-2 AL422B 引脚功能说明

管脚名称	管脚类型	管脚描述
DI[0:7]	输入	数据输入引脚
WCK	输入	数据输入同步时钟
/WE	输入	写使能信号，低电平有效
/WRST	输入	写指针复位信号，低电平有效
DO[0:7]	输出	数据输出引脚
RCK	输入	数据输出同步时钟
/RE	输入	读使能信号，低电平有效
/RRST	输入	读指针复位信号，低电平有效
/OE	输入	数据输出使能，低电平有效
TST	输入	测试引脚，实际使用时设置为低电平

由于 AL422B 支持同时写入和读出数据，所以它的输入和输出的控制信号线都是互相独立的。写入和读出数据的时序类似，跟 VGA 的像素输出时序一致，读写时序介绍如下：

- 写时序

  写 FIFO（AL422B，下面统称为 FIFO）时序见图 1-15。

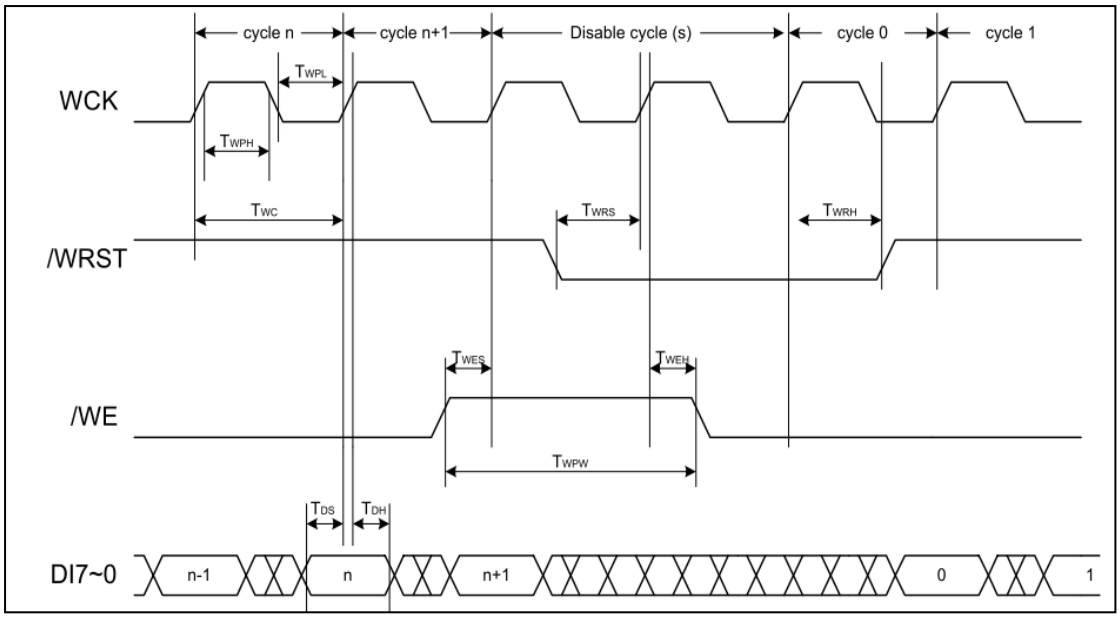

图 1-15 写 FIFO 时序

在写时序中，当 WE 管脚为低电平时，FIFO 写入处于使能状态，随着读时钟 WCK 的运转， DI[0:7]表示的数据将会就会按地址递增的方式存入 FIFO；当 WE 管脚为高电平时，关闭输入，DI[0:7]的数据不会被写入 FIFO。

在控制写入数据时，一般会先控制写指针作一个复位操作：把 WRST 设置为低电平，写指针会复位到 FIFO 的 0 地址，然后 FIFO 接收到的数据会从该地址开始按自增的方式写入。

- 读时序

  读 FIFO 时序见图 1-16。

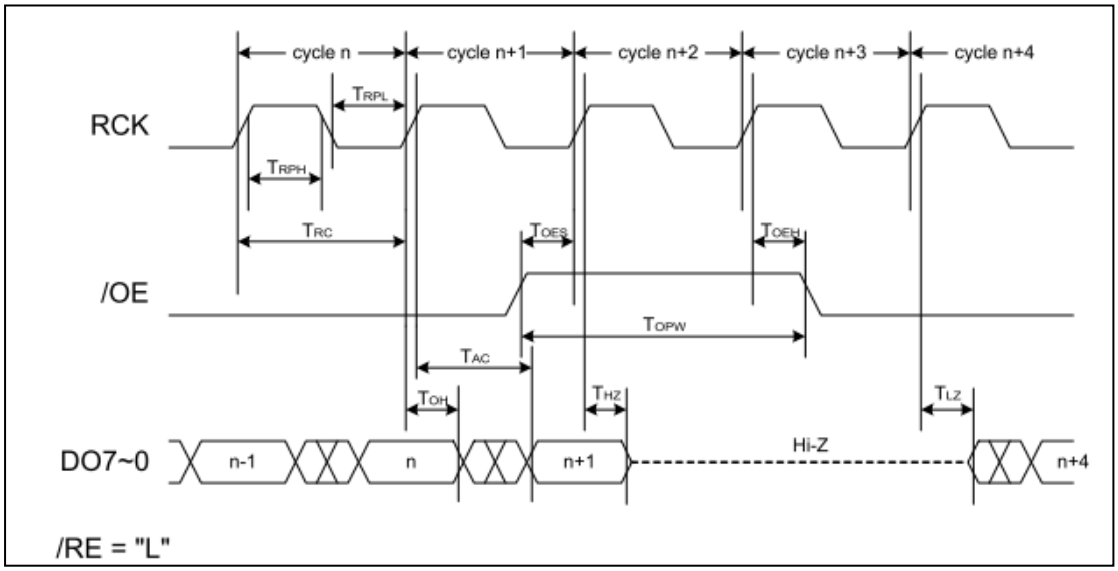

图 1-16 读 FIFO 时序

FIFO 的读时序类似，不过读使能由两个引脚共同控制，即 OE 和 RE 引脚均为低电平时，输出处于使能状态，随着读时钟 RCK 的运转，在数据输出管脚 DO[0:7]就会按地址递增的方式输出数据。

类似地，在控制读出数据时，一般会先控制读指针作一个复位操作：把 RRST 设置为低电平，读指针会复位到 FIFO 的 0 地址，然后 FIFO 数据从该地址开始按自增的方式输出。

1.2.7 摄像头的驱动原理

秉火 OV7725 摄像头中包含有 FIFO，所以外部控制器驱动摄像头时，需要协调好 FIFO 与 OV7725 传感器的关系，下面配合摄像头的原理图讲解其驱动原理。

原理图主要分为外部引出接口、OV7725 及 FIFO 部分，见图 1-17。

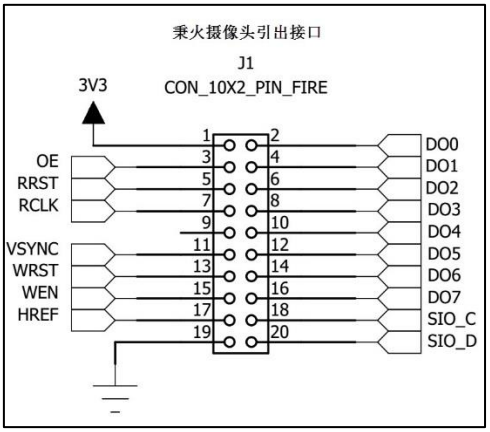

图 1-17 秉火摄像头引出的排母接口

摄像头引出的接口包含了 OV7725 传感器及 FIFO 的混合引脚，外部的控制器使用这些引脚即可驱动摄像头，其说明见表 1-3。

表 1-3 摄像头引脚列表

管脚名称	管脚关系	管脚描述
OE	FIFO 的 OE 引脚	数据输出使能，低电平有效
RRST	FIFO 的 RRST 引脚	读指针复位信号，低电平有效
RCLK	FIFO 的 RCK 引脚	数据输出同步时钟
VSYNC	OV7725 的 VSYNC 引脚	场同步信号
WRST	FIFO 的 WRST 引脚	写指针复位信号，低电平有效
WEN	与下面的 HREF 共同组成与非门的输入	与 HREF 共同控制 FIFO 的 WE 引脚，WEN 与 HREF 同时为高电平时，WE 为低电平，OV7725 可以向 FIFO 写入数据
HREF	OV7725 的 HREF 引脚	行同步信号
DO[0:7]	FIFO 的 DO[0:7]引脚	数据输出引脚
SIO_C	OV7725 的 SCL 引脚	SCCB 总线的时钟线
SIO_D	OV7725 的 SDA 引脚	SCCB 总线的数据线

从上述列表与下面的图 1-18 和图 1-19，可了解到，与 OV7725 传感器像素输出相关的 PCLK 和 D[0:7]并没有引出，因为这些引脚被连接到了 FIFO 的输入部分，OV7725 的像素输出时序与 FIFO 的写入数据时序是一致的，所以在 OV7725 时钟 PCLK 的驱动下，它输出的数据会一个字节一个字节地被 FIFO 接收并存储起来。

其中最为特殊的是 WEN 引脚，它与 OV7725 的 HREF 连接到一个与非门的输入，与非门的输出连接到 FIFO 的 WE 引脚，因此，当 WEN 与 HREF 均为高电平时，FIFO 的 WE 为低电平，此时允许 OV7725 向 FIFO 写入数据。

外部控制器通过控制 WEN 引脚，可防止 OV7725 覆盖了还未被控制器读出的旧 FIFO 数据。另外，在 OV7725 输出时序中，只有当 HREF 为高电平时，PCLK 驱动下 D[0:7]线表示的才是有效像素数据，因此，利用 HREF 控制 FIFO 的 WE 可以确保只有有效数据才被写入到 FIFO 中。

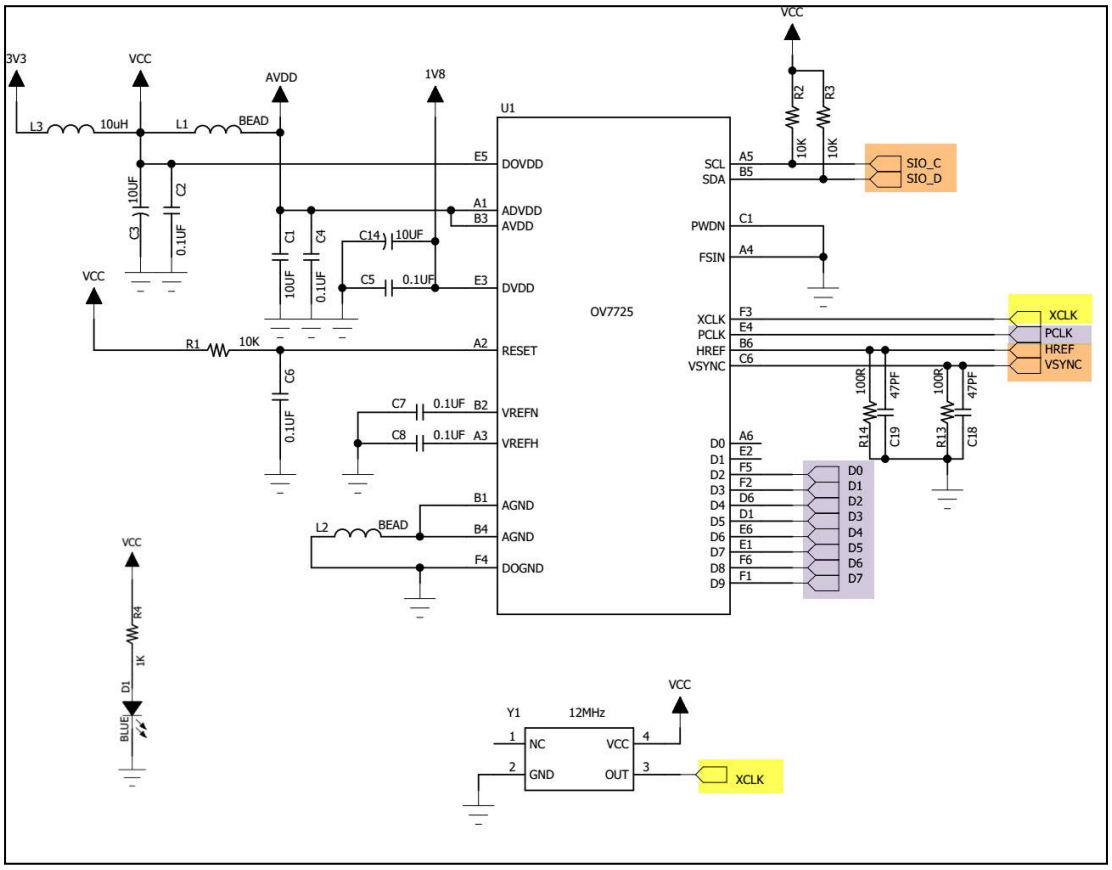

图 1-18 摄像头的 0V7725 部分硬件连接图

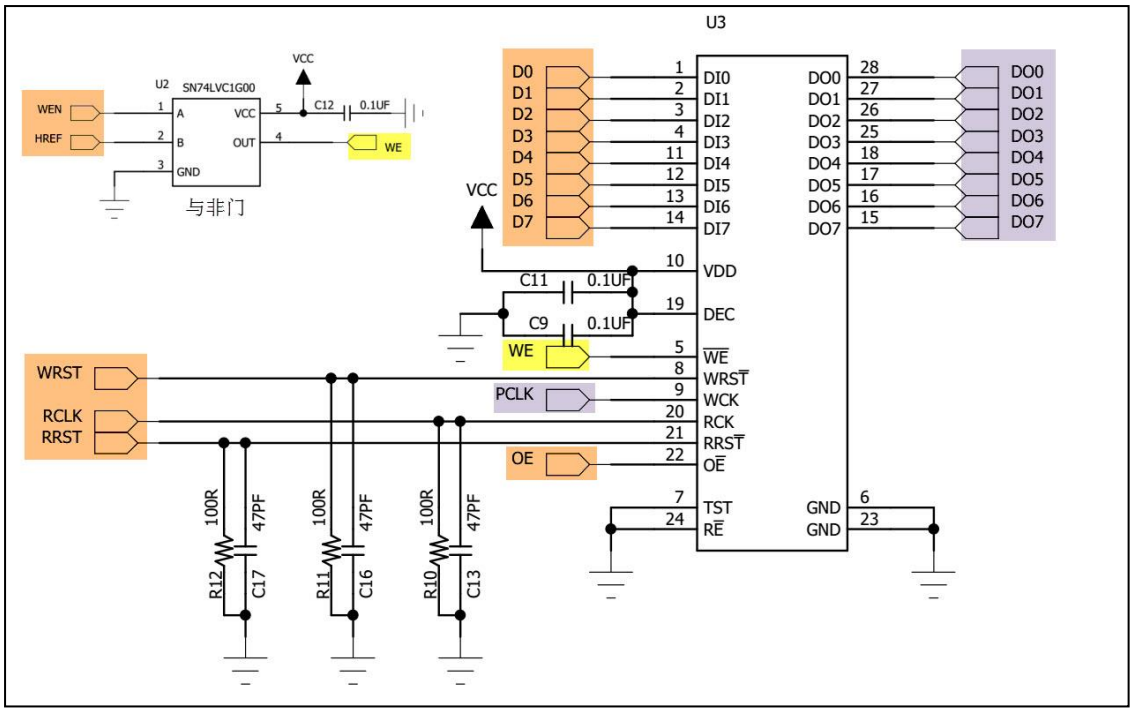

图 1-19 摄像头的 FIFO 部分硬件连接图

配合摄像头的原理图、OV7725、FIFO 的时序图，可总结出摄像头采集数据的过程如下：

(1) 利用 SIO_C、SIO_D 引脚通过 SCCB 协议向 OV7725 的寄存器写入初始化配置；

(2) 初始化完成后，OV7725 传感器会使用 VGA 时序输出图像数据，它的 VSYNC 会首先输出帧有效信号（低电平跳变），当外部的控制器（如 STM32）检测到该信号时，把 WEN 引脚设置为高电平，并且使用 WRST 引脚复位 FIFO 的写指针到 0 地址；

(3) 随着 OV7725 继续按 VGA 时序输出图像数据，它在传输每行有效数据时， HREF 引脚都会持续输出高电平，由于 WEN 和 HREF 同时为高电平输入至与非门，使得其连接到 FIFO WE 引脚的输出为低电平，允许向 FIFO 写入数据，所以在这期间，OV7725 通过它的 PCLK 和 D[0:7]信号线把图像数据存储到 FIFO 中，由于前面复位了写指针，所以图像数据是从 FIFO 的 0 地址开始记录的；

(4) 各行图像数据持续传输至 FIFO，受 HREF 控制的 WE 引脚确保了写入到 FIFO 中的都是有效的图像数据，OV7725 输出完一帧数据时，VSYNC 会再次输出帧有效信号，表示一帧图像已输出完成；

(5) 控制器检测到上述 VSYNC 信号后，可知 FIFO 中已存储好一帧图像数据，这时控制 WEN 引脚为低电平，使得 FIFO 禁止写入，防止 OV7725 持续输出的下一帧数据覆盖当前 FIFO 数据；

(6) 控制器使用 RRST 复位读指针到 FIFO 的 0 地址，然后通过 FIFO 的 RCLK 和 DO[0:7]引脚，从 0 地址开始把 FIFO 缓存的整帧图像数据读取出来。在这期间，OV7725 是持续输出它采集到的图像数据的，但由于禁止写入 FIFO，这些数据被丢弃了；

(7) 控制器使用 WRST 复位写指针到 FIFO 的 0 地址，然后等待新的 VSYNC 有效信号到来，检测到后把 WEN 引脚设置为高电平，恢复 OV7725 向 FIFO 的写入权限，OV7725 输出的新一帧图像数据会被写入到 FIFO 的 0 地址中，重复上述过程。

摄像头的整个控制过程见图 1-20。

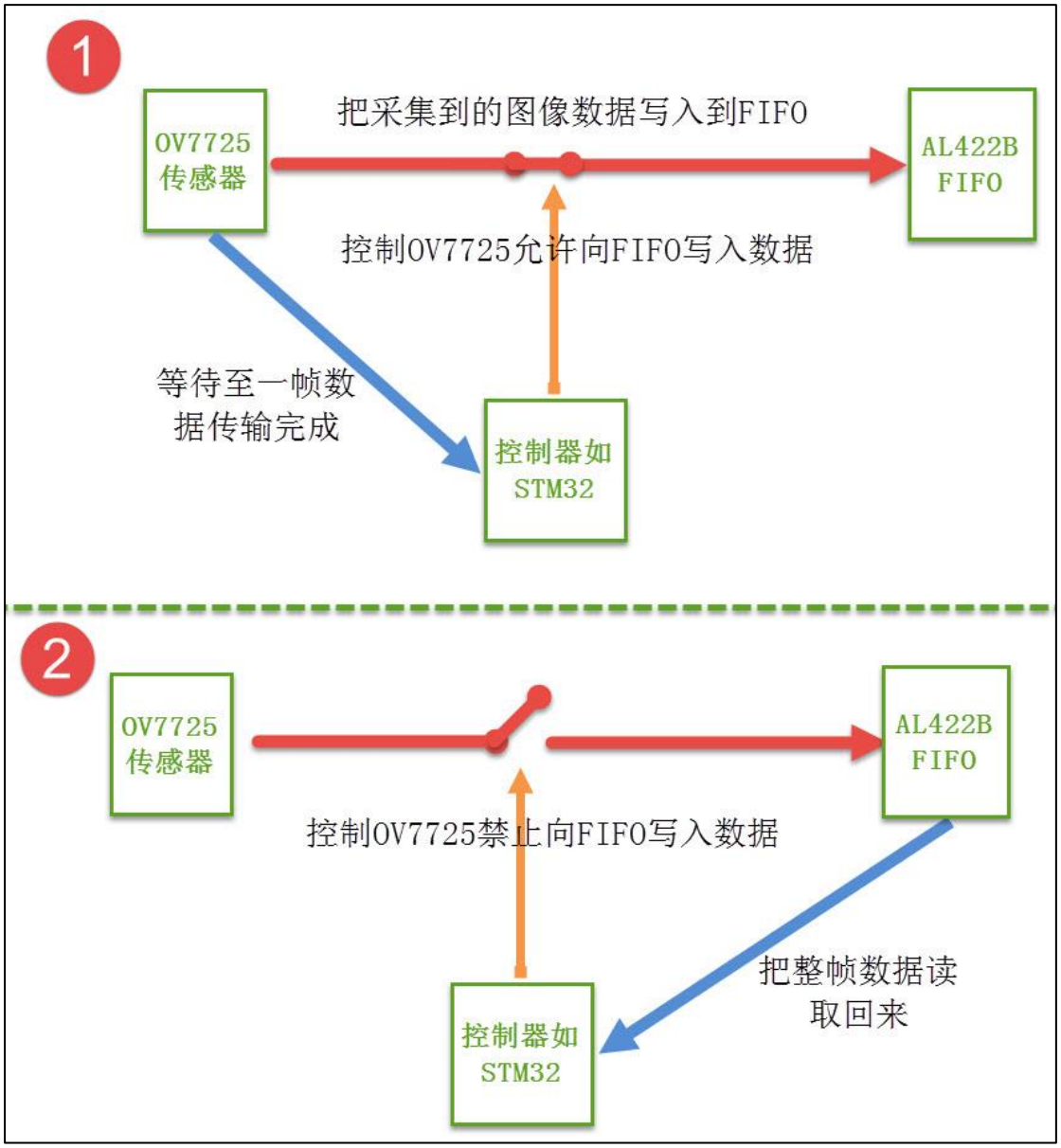

图 1-20 摄像头控制过程

在使用本摄像头时，我们一般配套开发板的液晶屏，把 OV7725 配置为 240*320 分辨率（QVGA），RGB565 格式，那么 OV7725 输出一帧的图像大小为 240*320*2=153600 字节，而本摄像头采用的 FIFO 型号 AL422B 容量为 393216 字节，最多可以缓存 2 帧这样的图像，通过这样的方式，STM32 无需直接处理 OV7725 高速输出的数据。但是，如果配置 OV7725 为 480*640 分辨率时，其一帧图像大小为 480*640*2=614400 字节，FIFO 的容量不足以直接存储一帧这样的图像，因此，当 OV7725 往 FIFO 写数据的时候，STM32 端要同时读取数据，确保在 OV7725 覆盖旧数据的之前，STM32 端已经把这部分数据读取出来了。

1.3 摄像头驱动实验

本小节讲解如何使用如何利用 OV7725 摄像头采集 RGB565 格式的图像数据，并把这些数据实时显示到液晶屏上。

学习本小节内容时，请打开配套的“摄像头-OV7725-液晶实时显示”工程配合阅读。

1.3.1 硬件设计

关于摄像头的原理图此处不再分析。在我们的实验板上有引出一个摄像头专用的排母，可直接与摄像头引出的引脚连，接入后它与 STM32 引脚的连接关系见图 1-21。

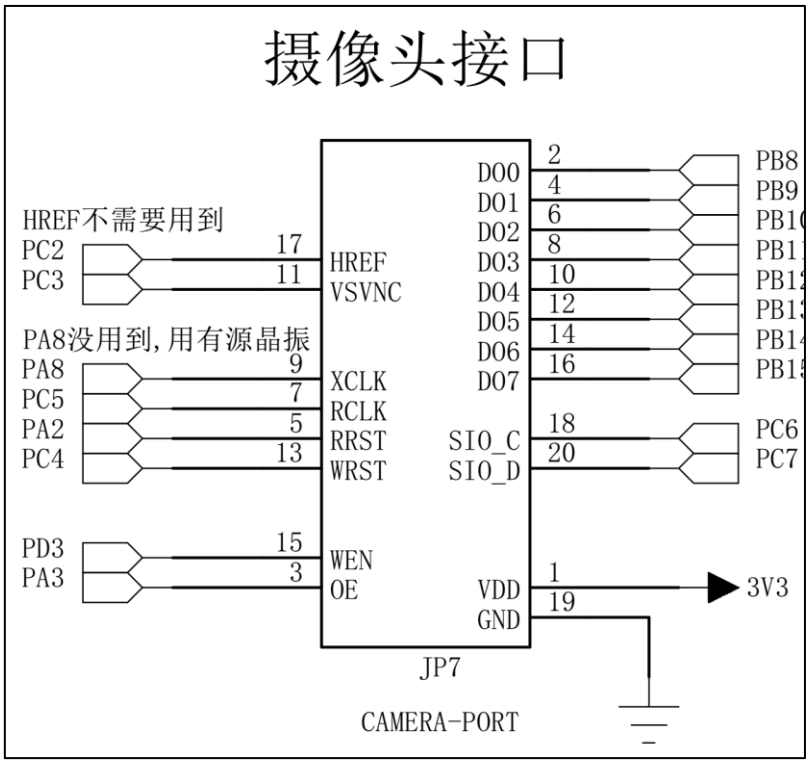

图 1-21 STM32 实验板引出的摄像头接口

摄像头与 STM32 连接关系中主要分为 SCCB 控制、VGA 时序控制、FIFO 数据读取部分，介绍如下：

(1) SCCB 控制相关

摄像头中的 SIO_C 和 SIO_D 引脚直接连接到 STM32 普通的 GPIO，它们不具有硬件 I2C 的功能，所以在后面的代码中采用模拟 I2C 时序，实际上直接使用硬件 I2C 是完全可以实现 SCCB 协议的，本设计采用模拟 I2C 是芯片资源分配妥协的结果。

(2) VGA 时序相关

检测 VGA 时序的 HREF、VSYNC 引脚，它们与 STM32 连接的 GPIO 均设置为输入模式，其中 HREF 在本实验中并没有使用，它已经通过摄像头内部的与非门控制了 FIFO 的写

使能；VSYNC 与 STM32 连接的 GPIO 引脚会在程序中配置成中断模式，STM32 利用该中断信号获知新的图像是否采集完成，从而控制 FIFO 是否写使能。

(3) FIFO 相关

与 FIFO 控制相关的 RCLK、RRST、WRST、WEN 及 OE 与 STM32 连接的引脚均直接配置成推挽输出，STM32 根据图像的采集情况利用这些引脚控制 FIFO；读取 FIFO 数据内容使用的数据引脚 DO[0:7]均连接到 STM32 同一个 GPIO 端口连续的高 8 位引脚 PB[8:15]，这些引脚使用时均配置成输入，程序设计中直接读取 GPIO 端口的高 8 位状态直接获取一个字节的 FIFO 内容，建议在连接这部分数据信号时，参考本设计采用同一个 GPIO 端口连续的 8 位（高 8 位或低 8 位均可），否则会导致读取数据的程序非常复杂。

(4) XCLK 信号

本设计中 STM32 的摄像头接口还预留了 PA8 引脚用于与摄像头的 XCLK 连接，STM32 的 PA8 可以对外输出时钟信号，所以在使用不带晶振的摄像头时，可以通过该引脚给摄像头提供时钟，秉火摄像头内部已自带晶振，在程序中没有使用 PA8 引脚。

以上原理图可查阅《指南者开发板—原理图》文档获知，若您使用的摄像头或实验板不一样，请根据实际连接的引脚修改程序。

1.3.2 软件设计

本实验的工程名称为“液晶实时显示”，学习时请打开该工程配合阅读。为了方便展示及移植，我们把模拟 SCCB 时序相关的代码写到 bsp_sccb.c 及 bsp_sccb.h 文件中，而摄像头模式控制相关的代码都编写到“bsp_ov7725.c”、“bsp_ov7725.h”文件中，这些文件是我们自己编写的，不属于标准库的内容，可根据您的喜好命名文件。

1. 编程要点

(1) 初始化 SCCB 通讯使用的目标引脚及端口时钟；
(2) 初始化 OV7725 的 VGA 和 FIFO 控制相关的引脚和时钟；
(3) 使用 SCCB 协议向 OV7725 写入初始化配置；
(4) 配置筛选器的工作方式；
(5) 编写测试程序，收发报文并校验。

2. 代码分析

摄像头硬件相关宏定义

我们把摄像头控制硬件相关的配置以宏的形式定义到 “bsp_ov7725.h”及“bsp_sccb.h”文件中，其中包括 VGA 部分接口、FIFO 控制及 SCCB(即模拟 I2C)相关的引脚，见代码清单 1-1。

代码清单 1-1 摄像头硬件配置相关的宏(bsp_ov7725.h 文件)

```
1 /***************** OV7725 连接引脚定义*********************************/
 2 // FIFO 输出使能，即模块中的 OE
 3 #define      OV7725_OE_GPIO_CLK                RCC_APB2Periph_GPIOA
 4 #define      OV7725_OE_GPIO_PORT               GPIOA
```

```
 5 #define      OV7725_OE_GPIO_PIN                    GPIO_Pin_3
 6
 7 // FIFO 写复位
 8 #define      OV7725_WRST_GPIO_CLK                  RCC_APB2Periph_GPIOC
 9 #define      OV7725_WRST_GPIO_PORT                 GPIOC
10 #define      OV7725_WRST_GPIO_PIN                  GPIO_Pin_4
11
12 // FIFO 读复位
13 #define      OV7725_RRST_GPIO_CLK                  RCC_APB2Periph_GPIOA
14 #define      OV7725_RRST_GPIO_PORT                 GPIOA
15 #define      OV7725_RRST_GPIO_PIN                  GPIO_Pin_2
16
17 // FIFO 读时钟
18 #define      OV7725_RCLK_GPIO_CLK                  RCC_APB2Periph_GPIOC
19 #define      OV7725_RCLK_GPIO_PORT                 GPIOC
20 #define      OV7725_RCLK_GPIO_PIN                  GPIO_Pin_5
21
22 // FIFO 写使能
23 #define      OV7725_WE_GPIO_CLK                   RCC_APB2Periph_GPIOD
24 #define      OV7725_WE_GPIO_PORT                  GPIOD
25 #define      OV7725_WE_GPIO_PIN                   GPIO_Pin_3
26
27 // 8 位数据口
28 #define      OV7725_DATA_GPIO_CLK                  RCC_APB2Periph_GPIOB
29 #define      OV7725_DATA_GPIO_PORT                 GPIOB
30 #define      OV7725_DATA_0_GPIO_PIN                GPIO_Pin_8
31 #define      OV7725_DATA_1_GPIO_PIN                GPIO_Pin_9
32 #define      OV7725_DATA_2_GPIO_PIN                GPIO_Pin_10
33 #define      OV7725_DATA_3_GPIO_PIN                GPIO_Pin_11
34 #define      OV7725_DATA_4_GPIO_PIN                GPIO_Pin_12
35 #define      OV7725_DATA_5_GPIO_PIN                GPIO_Pin_13
36 #define      OV7725_DATA_6_GPIO_PIN                GPIO_Pin_14
37 #define      OV7725_DATA_7_GPIO_PIN                GPIO_Pin_15
38
39 // OV7725 场中断
40 #define      OV7725_VSYNC_GPIO_CLK                 RCC_APB2Periph_GPIOC
41 #define      OV7725_VSYNC_GPIO_PORT                GPIOC
42 #define      OV7725_VSYNC_GPIO_PIN                 GPIO_Pin_3
43
44 #define      OV7725_VSYNC_EXTI_SOURCE_PORT          GPIO_PortSourceGPIOE
45 #define      OV7725_VSYNC_EXTI_SOURCE_PIN           GPIO_PinSource3
46 #define      OV7725_VSYNC_EXTI_LINE                EXTI_Line3
47 #define      OV7725_VSYNC_EXTI_IRQ                 EXTI3_IRQn
48 #define      OV7725_VSYNC_EXTI_INT_FUNCTION         EXTI3_IRQHandler
49
50 /*FIFO 输出使能，低电平使能*/
51 #define FIFO_OE_H()     OV7725_OE_GPIO_PORT->BSRR =OV7725_OE_GPIO_PIN
52 #define FIFO_OE_L()     OV7725_OE_GPIO_PORT->BRR  =OV7725_OE_GPIO_PIN
53
54 /*FIFO 写复位，低电平复位 */
55 #define FIFO_WRST_H()   OV7725_WRST_GPIO_PORT->BSRR =OV7725_WRST_GPIO_PIN
56 #define FIFO_WRST_L()   OV7725_WRST_GPIO_PORT->BRR  =OV7725_WRST_GPIO_PIN
57
58 /*FIFO 读复位，低电平复位 */
59 #define FIFO_RRST_H()   OV7725_RRST_GPIO_PORT->BSRR =OV7725_RRST_GPIO_PIN
60 #define FIFO_RRST_L()   OV7725_RRST_GPIO_PORT->BRR  =OV7725_RRST_GPIO_PIN
61
62 /*FIFO 输出数据时钟，上升沿有效*/
63 #define FIFO_RCLK_H()   OV7725_RCLK_GPIO_PORT->BSRR =OV7725_RCLK_GPIO_PIN
64 #define FIFO_RCLK_L()   OV7725_RCLK_GPIO_PORT->BRR  =OV7725_RCLK_GPIO_PIN
65
66 /*WE 引脚，高电平使能，与 HREF 通过与非门控制 FIFO 的 WEN*/
67 #define FIFO_WE_H()     OV7725_WE_GPIO_PORT->BSRR =OV7725_WE_GPIO_PIN
68 #define FIFO_WE_L()     OV7725_WE_GPIO_PORT->BRR  =OV7725_WE_GPIO_PIN
```

代码清单 1-2 SCCB 接口相关的宏定义

```
/************************ SCCB 引脚定义*******************************/
#define      OV7725_SIO_C_SCK_APBxClock_FUN           RCC_APB2PeriphClockCmd
#define      OV7725_SIO_C_GPIO_CLK                    RCC_APB2Periph_GPIOC
#define      OV7725_SIO_C_GPIO_PORT                   GPIOC
#define      OV7725_SIO_C_GPIO_PIN                    GPIO_Pin_6

#define      OV7725_SIO_D_SCK_APBxClock_FUN           RCC_APB2PeriphClockCmd
#define      OV7725_SIO_D_GPIO_CLK                    RCC_APB2Periph_GPIOC
#define      OV7725_SIO_D_GPIO_PORT                   GPIOC
#define      OV7725_SIO_D_GPIO_PIN                    GPIO_Pin_7

/*模拟时序的引脚控制*/
#define SCL_H   GPIO_SetBits(OV7725_SIO_C_GPIO_PORT , OV7725_SIO_C_GPIO_PIN)
#define SCL_L  GPIO_ResetBits(OV7725_SIO_C_GPIO_PORT , OV7725_SIO_C_GPIO_PIN)

#define SDA_H   GPIO_SetBits(OV7725_SIO_D_GPIO_PORT , OV7725_SIO_D_GPIO_PIN)
#define SDA_L   GPIO_ResetBits(OV7725_SIO_D_GPIO_PORT , OV7725_SIO_D_GPIO_PIN)

#define SCL_read GPIO_ReadInputDataBit(OV7725_SIO_C_GPIO_PORT,OV7725_SIO_C_GPIO_PIN)
#define SDA_read GPIO_ReadInputDataBit(OV7725_SIO_D_GPIO_PORT, OV7725_SIO_D_GPIO_PIN)

#define ADDR_OV7725   0x42

```

以上代码根据硬件的连接，使用宏封装了各种控制信号，包括控制输出电平或读取输入时使用的库函数操作。若使用 STM32 与摄像头的引脚连接与我们的设计的不同，修改这两个文件的引脚连接关系即可。

SCCB 总线的软件实现

本设计中使用普通 GPIO 来模拟 SCCB 时序，需要根据 SCCB 时序，编写读、写字节的模拟函数，在后面的 OV7725_Init 会利用这些函数向 OV7725 相应的寄存器写入配置参数，初始化摄像头。本小节介绍的与 SCCB 时序相关函数都位于 bsp_sccb.c 文件中，这些函数跟模拟 I2C 的基本一致。

(1) 初始化 SCCB 用到的 GPIO

在本实验中，使用 SCCB_GPIO_Config 函数初始化 SCCB 使用的两个通讯引脚，把它们初始化成普通开漏输出模式，其代码见代码清单 1-3。

代码清单 1-3：SCCB_GPIO_Config 函数

```
/*****************************************************************
 * 函数名：SCCB_Configuration
 * 描述  ：SCCB 管脚配置
 * 输入  ：无
 * 输出  ：无
 * 注意  ：无
 *****************************************************************/
void SCCB_GPIO_Config(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    /* SCL、SDA 管脚配置 */
    OV7725_SIO_C_SCK_APBxClock_FUN ( OV7725_SIO_C_GPIO_CLK, ENABLE );
    GPIO_InitStructure.GPIO_Pin =  OV7725_SIO_C_GPIO_PIN ;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
    GPIO_Init(OV7725_SIO_C_GPIO_PORT, &GPIO_InitStructure);

    OV7725_SIO_D_SCK_APBxClock_FUN ( OV7725_SIO_D_GPIO_CLK, ENABLE );
    GPIO_InitStructure.GPIO_Pin =  OV7725_SIO_D_GPIO_PIN ;
    GPIO_Init(OV7725_SIO_D_GPIO_PORT, &GPIO_InitStructure);
}
```

(2) SCCB 起始与结束时序

SCCB 通讯需要有起始与结束信号，这分别由 SCCB_Start 和 SCCB_Stop 函数实现。

SCCB_Start 代码见代码清单 1-4。

代码清单 1-4：SCCB_Start 函数

```
/*****************************************************************
 * 函数名：SCCB_Start
 * 描述  ：SCCB 起始信号
 * 输入  ：无
 * 输出  ：无
 * 注意  ：内部调用
 *****************************************************************/
static int SCCB_Start(void)
{
    SDA_H;
    SCL_H;
    SCCB_delay();
    if (!SDA_read)
        return DISABLE; /* SDA 线为低电平则总线忙,退出 */
    SDA_L;
    SCCB_delay();
    if (SDA_read)
        return DISABLE; /* SDA 线为高电平则总线出错,退出 */
    SDA_L;
    SCCB_delay();
    return ENABLE;
}
```

参照前面介绍的 SCCB 时序，当 SCL 线为高电平时，SDA 线出现下降沿，表示 SCCB 时序的起始信号，SCCB_Start 函数就是实现了这样功能，其中的 SDA_H 和 SCL_H 是用于控制 SDA 和 SCL 引脚电平的宏。为了提高程序的健壮性，还使用 SDA_read 宏检测 SDA 线是否忙碌或是否正常。

SCCB 结束信号的函数实现类似，其 SCCB_Stop 代码见代码清单 1-5。

代码清单 1-5：SCCB_Stop 函数

```
/*****************************************************************
 * 函数名：SCCB_Stop
 * 描述  ：SCCB 停止信号
 * 输入  ：无
 * 输出  ：无
 * 注意  ：内部调用
 *****************************************************************/
static void SCCB_Stop(void)
{
    SCL_L;
    SCCB_delay();
    SDA_L;
    SCCB_delay();
    SCL_H;
    SCCB_delay();
    SDA_H;
    SCCB_delay();
}
```

参照前面 SCCB 时序的介绍，当 SCL 线为高电平的时候，使 SDA 线出现一个上升沿，表示 SCCB 时序的结束信号。

(3) 写寄器与读寄存器

与 I2C 时序类似，在 SCCB 时序也使用自由位(Don’t care bit)和非应答(NA)信号来保证正常通讯。自由位和非应答信号位于 SCCB 每个传输阶段中的第九位。

在写数据的第一个传输阶段中，第 9 位为自由位，在一般的正常通讯中，第 9 位时，主机的 SDA 线输出高电平，而从机把 SDA 线拉低作为响应，第二、三阶段类似，只是传输的内容分别为目的寄存器地址和要写入的数据。见图 1-22。

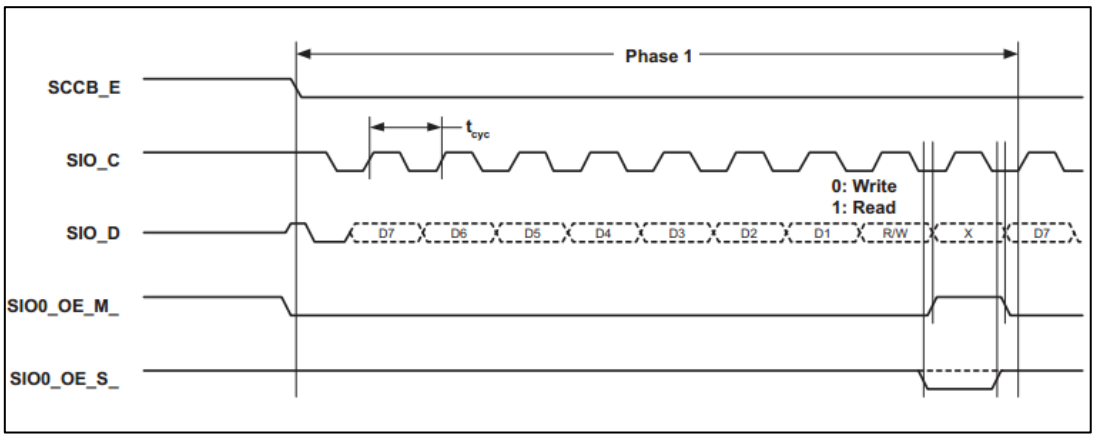

图 1-22 写操作第一阶段(传输器件地址)

因此，在数据传输的第 9 位，主机使用 SCCB_WaitAck 函数来等待从机的应答，代码见代码清单 1-6。

代码清单 1-6：SCCB_WaitAck 函数

```
/*****************************************************************
 * 函数名：SCCB_WaitAck
 * 描述  ：SCCB 等待应答
 * 输入  ：无
 * 输出  ：返回为:=1 有 ACK,=0 无 ACK
 * 注意  ：内部调用
 *****************************************************************/
static int SCCB_WaitAck(void)
{
    SCL_L;
    SCCB_delay();
    SDA_H;
    SCCB_delay();
    SCL_H;
    SCCB_delay();
    if (SDA_read) {
        SCL_L;
        return DISABLE;
    }
    SCL_L;
    return ENABLE;
}
```

该函数让主机把 SDA 线设为高电平，延时一段时间后再检测 SDA 线的电平，若为低则返回 ENABLE 表示接收到从机的应答，反之返回 DISABLE。

最后，整个三相写过程由函数 SCCB_WriteByte 实现，它的具体代码见代码清单 1-7。

代码清单 1-7：SCCB_WriteByte 函数

```
/*以下宏位于 bsp_ov7725.h 文件*/
#define ADDR_OV7725   0x42
/*以下宏位于 bsp_sccb.h 文件*/
#define DEV_ADR  ADDR_OV7725
```

```
 5
 6
/*******************************************************************************
 7 * 函数名: SCCB_WriteByte
 8 * 描述   : 写一字节数据
 9 * 输入: -WriteAddress:待写入地址- SendByte:待写入数据 - DeviceAddress: 器件类型
10 * 输出   : 返回为:=1 成功写入,=0 失败
11 * 注意   : 无
12 *******************************************************************************/
13 int SCCB_WriteByte( uint16_t WriteAddress , uint8_t SendByte )
14 {
15     if (!SCCB_Start()) {
16         return DISABLE;
17     }
18     SCCB_SendByte( DEV_ADR );                    /* 器件地址 */
19     if ( !SCCB_WaitAck() ) {
20         SCCB_Stop();
21         return DISABLE;
22     }
23     SCCB_SendByte((uint8_t)(WriteAddress & 0x00FF));   /* 设置低起始地址 */
24     SCCB_WaitAck();
25     SCCB_SendByte(SendByte);
26     SCCB_WaitAck();
27     SCCB_Stop();
28     return ENABLE;
29 }
```

SCCB_WriteByte 函数调用 SCCB_Start 产生起始信号，接着调用 SCCB_SendByte 把器件地址 DEV_ADR(这是一个宏，数值是 0x42)一位一位地发送出去，在第 9 位时，调用 SCCB_WaitAck 函数检测从机的应答，若接收到应答则进入第二阶段——发送目的寄存器地址，再进入第三阶段——发送要写入的内容。在第二、三阶段没有加条件判断语句判断是否接收到从机的应答，这是因为 SCCB 规定在数据传输阶段允许从机不应答(实际上，OV7725 芯片在这两个阶段都会有应答讯号 。在最后，三阶段都传输结束时，要调用 SCCB_Stop 函数结束本次 SCCB 传输。

与自由位相对应的非应答信号用在两相读操作的第二阶段的第 9 位，见图 1-23。在这第 9 位中，从机把 SDA 线置为高电平，而主机把 SDA 线拉低表示非应答，接着本次读数据的操作就结束了。

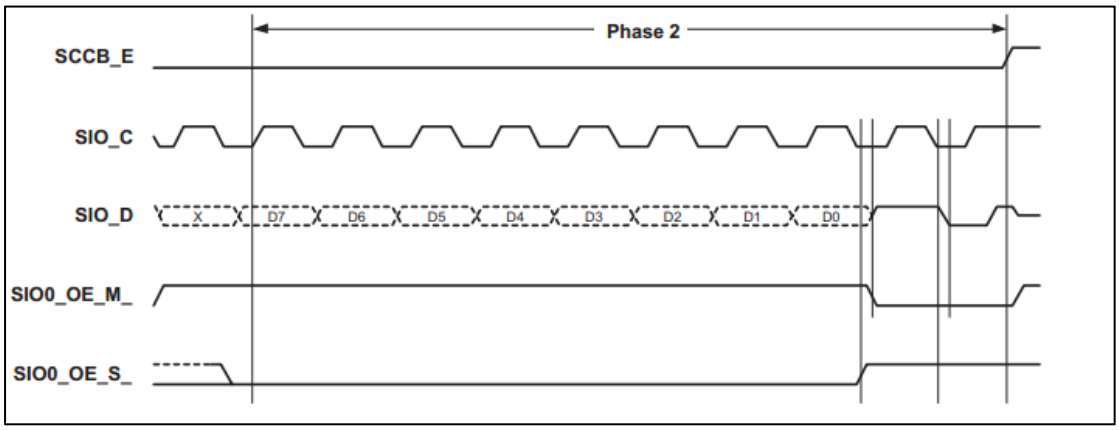

图 1-23 两相读操作第二阶段(读寄存器内容)

主机的非应答信号，由 SCCB_NoAck 函数实现，其代码见代码清单 1-8。

代码清单 1-8：SCCB_NoAck 函数

```
1 /*******************************************************************
```

```
 * 函数名：SCCB_NoAck
 * 描述  ：SCCB 无应答方式
 * 输入  ：无
 * 输出  ：无
 * 注意  ：内部调用
 ***************************************************************************/
static void SCCB_NoAck(void)
{
    SCL_L;
    SCCB_delay();
    SDA_H;
    SCCB_delay();
    SCL_H;
    SCCB_delay();
    SCL_L;
    SCCB_delay();
}
```

最后，整个读寄存器的过程由 SCCB_ReadByte 函数完成，其代码见代码清单 1-9。

代码清单 1-9：SCCB_ReadByte 函数

```
/***************************************************************************
 * 函数名：SCCB_ReadByte
 * 描述  ：读取一串数据
 * 输入：- pBuffer:存放读出数据- length:待读出长度- ReadAddress:待读出地址     -
 DeviceAddress: 器件类型
 * 输出  ：返回为:=1 成功读入,=0 失败
 * 注意  ：无
 ***************************************************************************/
int SCCB_ReadByte(uint8_t* pBuffer, uint16_t length, uint8_t ReadAddress)
{
    if (!SCCB_Start()) {
        return DISABLE;
    }
    SCCB_SendByte( DEV_ADR );         /* 器件地址 */
    if ( !SCCB_WaitAck() ) {
        SCCB_Stop();
        return DISABLE;
    }
    SCCB_SendByte( ReadAddress );     /* 设置低起始地址 */
    SCCB_WaitAck();
    SCCB_Stop();

    if (!SCCB_Start()) {
        return DISABLE;
    }
    SCCB_SendByte( DEV_ADR + 1 );     /* 器件地址 */
    if (!SCCB_WaitAck()) {
        SCCB_Stop();
        return DISABLE;
    }
    while (length) {
        *pBuffer = SCCB_ReceiveByte();
        if (length == 1) {
            SCCB_NoAck();
        } else {
            SCCB_Ack();
        }
        pBuffer++;
        length--;
    }
    SCCB_Stop();
    return ENABLE;
}
```

本函数在两相读操作前，加入了一个两相写操作，用于向从机发送要读取的寄存器地址。在两相写操作的第一个阶段(第 26 行)，使用 SCCB_SendByte 函数发送的数据是 DEV_ADR+1 (即 0x43)，这与写操作中发送的 DEV_ADR 有区别，这是因为在第一阶段发送的这个器件地址的最低位是用于表示数据传送方向的，最低位为 0 时表示主机写数据，最低位为 1 时表示主机读数据，所以 DEV_ADR+1 就表示读数据了。读操作的第二阶段使用 SCCB_ReceiveByte 函数，一位一位地接收数据，然后存放到 PBuffer 指向的单元中的。接收完 8 位数据后，主机调用 SCCB_NoAck 发送非应答位，最后调用 SCCB_Stop 结束本次读操作。

初始化 OV7725

在上一小节编写了模拟 SCCB 的读写寄存器的时序后，就可以向 OV7725 的寄存器发送配置参数，对 OV7725 进行初始化了。该过程由 bsp_ov7725.c 文件中的 OV7725_Init 函数完成，代码见代码清单 1-10。

代码清单 1-10：OV7725_Init 函数（bsp_ov7725.c 文件）

```
1  /*************************************************
2   * 函数名: Sensor_Init
3   * 描述   : Sensor 初始化
4   * 输入   : 无
5   * 输出   : 返回 1 成功，返回 0 失败
6   * 注意   : 无
7   *************************************************/
8  ErrorStatus OV7725_Init(void)
9  {
10     uint16_t i = 0;
11     uint8_t Sensor_IDCode = 0;
12
13     //开始配置 ov7725
14     if ( 0 == SCCB_WriteByte ( 0x12, 0x80 ) ) { /*复位 ov7725  */
15         //sccb 写数据错误
16         return ERROR ;
17     }
18         /* 读取 ov7725  ID 号*/
19     if ( 0 == SCCB_ReadByte( &Sensor_IDCode, 1, 0x0b ) ) {
20         //读取 ov7725  ID 失败
21         return ERROR;
22     }
23
24     if (Sensor_IDCode == OV7725_ID) {
25         for ( i = 0 ; i < OV7725_REG_NUM ; i++ ) {
26             if ( 0 == SCCB_WriteByte(Sensor_Config[i].Address,
                                        Sensor_Config[i].Value) )
27              {   //DEBUG("write reg faild", Sensor_Config[i].Address);
28                 return ERROR;
29             }
30         }
31     } else {
32         return ERROR;
33     }
34     //ov7725 寄存器配置成功
35     return SUCCESS;
36 }
```

这个函数的执行流程如下：

(1) 调用 SCCB_WriteByte 向地址为 0x12 的寄存器写入数据 0x80，进行复位操作。根据 OV7725 的数据手册，把该寄存器的位 7 置 1，可控制它对寄存器进行复位。

(2) 调用 SCCB_ReadByte 函数从地址为 0x0b 的寄存器读取出 OV7725 芯片的 ID 号，并与默认值进行对比，这个操作可以用来确保 OV7725 是否正常工作。

(3) 利用 for 语句循环调用 SCCB_WriteByte 函数，向各个寄存器写入配置参数，其中 SCCB_WriteByte 的输入参数为 Sensor_Config[i].Address 和 Sensor_Config[i].Value，这是自定义的结构体数组，分别对应于要配置的寄存器地址和寄存器配置参数。Sensor_Config 数组是在 bsp_ov7725.c 文件定义的，文件中首先定义了 Reg_Info 结构体类型，它包含地址和寄存器两个结构体成员，见代码清单 1-11。

代码清单 1-11：Register_Info 结构体

```
typedef struct Reg {
   uint8_t Address;  /*寄存器地址*/
   uint8_t Value;    /*寄存器值*/
} Reg_Info;
```

再利用这个自定义的结构体，定义结构体数组，每组的内容就表示了寄存器地址及相应的配置参数。若要修改对 OV7725 的配置，可参考 OV7725 数据手册的说明，修改相应地址的内容即可，SCCB_WriteByte 函数会在 for 循环中把这些参数写入 OV7725 芯片，下面是结构体数组的部分代码，它省略了部分寄存器，完整部分请参考源代码，见代码清单 1-12。

代码清单 1-12：部分寄存器控制参数

```

/* （bsp_ov7725.h 文件）寄存器地址宏定义 */
#define REG_GAIN      0x00
#define REG_BLUE      0x01
#define REG_RED       0x02
#define REG_GREEN     0x03
#define REG_BAVG      0x05
#define REG_GAVG      0x06
#define REG_RAVG      0x07
#define REG_AECH      0x08
#define REG_COM2      0x09
/*...以下省略大部分寄存器地址*/


/* （bsp_ov7725.c 文件）寄存器参数配置，左侧为地址，右侧为要写入的值 */
Reg_Info Sensor_Config[] = {
   {REG_CLKRC,     0x00}, /*时钟配置*/
   {REG_COM7,      0x46}, /*QVGA RGB565 */
   {REG_HSTART,    0x3f}, /*水平图像开始*/
   {REG_HSIZE,     0x50}, /*水平图像宽度*/
   {REG_VSTRT,     0x03}, /*垂直开始*/
   {REG_VSIZE,     0x78}, /*垂直高度*/
   {REG_HREF,      0x00}, /*杂项*/
   {REG_HOutSize,  0x50}, /*水平输出宽度*/
   {REG_VOutSize,  0x78}, /*垂直输出高度*/
   {REG_EXHCH,     0x00}, /*杂项*/

   /*...以下省略大部分内容.*/
};
```

采集并显示图像

OV7725 初始始化完成后，该芯片开始正常工作，由于 OV7725 采集得的图像保存到 FIFO，我们使用 STM32 只需要检测摄像头模块的 VSYNC 输出的帧结束信号，然后从 FIFO 中读取图像数据即可。

(1) 初始化 VSYNC 引脚

由于使用中断的方式来检测 VSYNC 的信号，所以要把相应的引脚初始化并为它配置 EXTI 中断，本实验使用 VSYNC_GPIO_Config 函数完成该工作，见代码清单 1-13。。

代码清单 1-13： VSYNC_GPIO_Config 函数（bsp_ov7725.c 文件）

```
/*************************************************
 * 函数名：VSYNC_GPIO_Config
 * 描述  ：OV7725 VSYNC 中断相关配置
 * 输入  ：无
 * 输出  ：无
 * 注意  ：无
 *************************************************/
static void VSYNC_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    /*初始化时钟，注意中断要开 AFIO*/
    RCC_APB2PeriphClockCmd ( RCC_APB2Periph_AFIO|
                             OV7725_VSYNC_GPIO_CLK, ENABLE );

    /*初始化引脚*/
    GPIO_InitStructure.GPIO_Pin =  OV7725_VSYNC_GPIO_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(OV7725_VSYNC_GPIO_PORT, &GPIO_InitStructure);

    /*配置中断*/
    GPIO_EXTILineConfig(OV7725_VSYNC_EXTI_SOURCE_PORT,
                        OV7725_VSYNC_EXTI_SOURCE_PIN);
    EXTI_InitStructure.EXTI_Line = OV7725_VSYNC_EXTI_LINE;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling ;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    EXTI_GenerateSWInterrupt(OV7725_VSYNC_EXTI_LINE);

    /*配置优先级*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    NVIC_InitStructure.NVIC_IRQChannel = OV7725_VSYNC_EXTI_IRQ;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

代码中把 VSYNC 引脚配置为浮空模式，并使用下降沿中断（配置成上升沿中断也是可以的），正好对应 VGA 时序中 VSYNC 输出信号时电平跳变产生的下降沿。

(2) 编写检测 VSYNC 的中断服务函数

由于 VSYNC 出现两次下降沿，才表示 FIFO 保存了一幅图像，所以在检测 VSYNC 下降沿的中断服务函数中，使用一个变量 Ov7725_vsync 作为标志。Ov7725_vsync 标志的初始值为 0，当检测到第一次上升沿时，控制 FIFO 的相应 GPIO 引脚，允许 OV7725 向 FIFO 写入图像数据，并把标志值设置为 1；检测到第二次上升沿时，禁止 OV7725 写 FIFO，把标志设置为 2，而我们将会在 main 函数的循环中对该标志进行判断，当 Ov7725_vsync=2 时，STM32 开始从 FIFO 读取数据并显示，读取完毕后把 Ov7725_vsync 标置复位为 0，重新开始下一幅图像的采集。

中断服务函数位于 stm32f10x_it.c 文件，见代码清单 1-14。

代码清单 1-14：VSYNC 的中断服务函数

```
1
2  #define     OV7725_VSYNC_EXTI_INT_FUNCTION          EXTI3_IRQHandler
3
4  /* ov7725 场中断 服务程序 */
5  void OV7725_VSYNC_EXTI_INT_FUNCTION ( void )
6  {
7     //检查 EXTI_Line 线路上的中断请求是否发送到了 NVIC
8     if ( EXTI_GetITStatus(OV7725_VSYNC_EXTI_LINE) != RESET ) {
9        if ( Ov7725_vsync == 0 ) {
10           FIFO_WRST_L();      //拉低使 FIFO 写(数据 from 摄像头)指针复位
11           FIFO_WE_H();      //拉高使 FIFO 写允许
12
13           Ov7725_vsync = 1;
14           FIFO_WE_H();       //使 FIFO 写允许
15           FIFO_WRST_H();     //允许使 FIFO 写(数据 from 摄像头)指针运动
16       } else if ( Ov7725_vsync == 1 ) {
17           FIFO_WE_L();       //拉低使 FIFO 写暂停
18           Ov7725_vsync = 2;
19       }
20       //清除 EXTI_Line 线路挂起标志位
21       EXTI_ClearITPendingBit(OV7725_VSYNC_EXTI_LINE);
22    }
23 }
24
```

(3) 读 FIFO 并显示图像

采集得的图像数据都保存到摄像头模块的 FIFO 中，在 Ov7725_vsync 标志变为 2 的时候，STM32 即可读取它并显示到 LCD 上。与 FIFO 相关的函数有 FIFO_GPIO_Config、FIFO_PREPARE 和 ImagDisp 。

- FIFO_GPIO_Config 类似 SCCB_GPIO_Config 函数，完成了基本的 GPIO 初始化，见代码清单 1-15。

代码清单 1-15：FIFO_GPIO_Config 函数

```
1  /*************************************************
2   * 函数名: FIFO_GPIO_Config
3   * 描述  : FIFO GPIO 配置
4   * 输入  : 无
5   * 输出  : 无
6   * 注意  : 无
7   *************************************************/
8  static void FIFO_GPIO_Config(void)
9  {
10    GPIO_InitTypeDef GPIO_InitStructure;
11
12    /*开启时钟*/
13    RCC_APB2PeriphClockCmd (OV7725_OE_GPIO_CLK|OV7725_WRST_GPIO_CLK|
14                            OV7725_RRST_GPIO_CLK|OV7725_RCLK_GPIO_CLK|
15                        OV7725_WE_GPIO_CLK|OV7725_DATA_GPIO_CLK, ENABLE );
16
17    /*(FIFO_OE--FIFO 输出使能)*/
18    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
19    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
20    GPIO_InitStructure.GPIO_Pin = OV7725_OE_GPIO_PIN;
21    GPIO_Init(OV7725_OE_GPIO_PORT, &GPIO_InitStructure);
22
23    /*(FIFO_WRST--FIFO 写复位)*/
```

```
    GPIO_InitStructure.GPIO_Pin = OV7725_WRST_GPIO_PIN;
    GPIO_Init(OV7725_WRST_GPIO_PORT, &GPIO_InitStructure);

    /*(FIFO_RRST--FIFO 读复位) */
    GPIO_InitStructure.GPIO_Pin = OV7725_RRST_GPIO_PIN;
    GPIO_Init(OV7725_RRST_GPIO_PORT, &GPIO_InitStructure);

    /*(FIFO_RCLK-FIFO 读时钟)*/
    GPIO_InitStructure.GPIO_Pin = OV7725_RCLK_GPIO_PIN;
    GPIO_Init(OV7725_RCLK_GPIO_PORT, &GPIO_InitStructure);

    /*(FIFO_WE--FIFO 写使能)*/
    GPIO_InitStructure.GPIO_Pin = OV7725_WE_GPIO_PIN;
    GPIO_Init(OV7725_WE_GPIO_PORT, &GPIO_InitStructure);

    /*(FIFO_DATA--FIFO 输出数据)*/
   GPIO_InitStructure.GPIO_Pin =OV7725_DATA_0_GPIO_PIN | OV7725_DATA_1_GPIO_PIN |
                          OV7725_DATA_2_GPIO_PIN | OV7725_DATA_3_GPIO_PIN |
                          OV7725 DATA 4 GPIO PIN |  OV7725 DATA 5 GPIO PIN |
                          OV7725 DATA 6 GPIO PIN |  OV7725 DATA 7 GPIO PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(OV7725_DATA_GPIO_PORT, &GPIO_InitStructure);

    FIFO_OE_L();              /*拉低使 FIFO 输出使能*/
    FIFO_WE_H();              /*拉高使 FIFO 写允许*/
}
```

- FIFO_PREPAGE 实际是一个宏，它是在 main 函数中，判断到接收完成一幅图像后被调用的，它的作用是把 FIFO 读指针复位，使后面的数据读取从 FIFO 的 0 地址开始，其代码见**代码清单** 1-16(在工程中，要把宏定义写在同一行或用续行符)。

代码清单 1-16：FIFO_PREPAGE 宏

```
#define FIFO_PREPARE            do{\
                              FIFO_RRST_L();\
                              FIFO_RCLK_L();\
                              FIFO_RCLK_H();\
                              FIFO_RRST_H();\
                              FIFO_RCLK_L();\
                              FIFO_RCLK_H();\
                              }while(0)
```

- ImagDisp 函数完成了从 FIFO 读取图像及显示到 LCD 的工作，每当 OV7725 输出完一幅图像，它被调用一次，在调用前，要用上面的 FIFO_PREAGE 宏复位 FIFO 读指针。它的代码见**代码清单** 1-17。

代码清单 1-17：ImagDisp 函数

```
/**
  * @brief  设置显示位置
  * @param  sx:x 起始显示位置
  * @param  sy:y 起始显示位置
  * @param  width:显示窗口宽度,要求跟 OV7725_Window_Set 函数中的 width 一致
  * @param  height:显示窗口高度，要求跟 OV7725_Window_Set 函数中的 height 一致
  * @retval 无
  */
void ImagDisp(uint16_t sx,uint16_t sy,uint16_t width,uint16_t height)
{
    uint16_t i, j;
    uint16_t Camera_Data;

    ILI9341_OpenWindow(sx,sy,width,height);
    ILI9341_Write_Cmd ( CMD_SetPixel );
```

```
16
17      for (i = 0; i < width; i++) {
18          for (j = 0; j < height; j++) {
19              /* 从 FIFO 读出一个 rgb565 像素到 Camera_Data 变量 */
20              READ_FIFO_PIXEL(Camera_Data);
21              ILI9341_Write_Data(Camera_Data);
22          }
23      }
24  }
```

在代码中，先根据输入参数在液晶屏设置了显示窗口，然后循环调用宏 READ_FIFO_PIXEL 读取 FIFO 数据，循环的次数就是摄像头输出的像素个数，代码中使用 width 和 height 控制，最后使用 LCD_WR_Data 函数把该图像数据显示到 LCD 上。宏 READ_FIFO_PIXEL 代码见**代码清单 1-18**(在工程中，要把宏定义写在同一行或用续行符)。

代码清单 1-18： READ_FIFO_PIXEL 宏

```
1  #define READ_FIFO_PIXEL(RGB565)     do{\
2                                      RGB565=0;\
3                                      FIFO_RCLK_L();\
4                    RGB565 = (OV7725_DATA_GPIO_PORT->IDR) & 0xff00;\
5                                      FIFO_RCLK_H();\
6                                      FIFO_RCLK_L();\
7                   RGB565 |= (OV7725_DATA_GPIO_PORT->IDR >>8) & 0x00ff;\
8                                      FIFO_RCLK_H();\
9                                      }while(0)
```

这个宏把 FIFO 读取到的数据按 RGB565 的处理，保存到一个 16 位的变量中，LCD_WR_Data 函数可以直接利用这个数据，显示一个像素点到 LCD 上。

main 文件

利用前面介绍的函数，就可以驱动摄像头采集并显示图像了，关于摄像头模式或分辨率的配置本工程还提供了其它的函数进行修改，首先来看了解实现了采集流程的最基本的 main 函数，见**代码清单 1-19**。

代码清单 1-19：摄像头例程的 main 函数（main.c 文件）

```
1
2  /**
3    * @brief  主函数
4    * @param  无
5    * @retval 无
6    */
7  int main(void)
8  {
9     float frame_count = 0;
10    uint8_t retry = 0;
11
12    /* 液晶初始化 */
13    ILI9341_Init();
14    ILI9341_GramScan ( 3 );
15
16    LCD_SetFont(&Font8x16);
17    LCD_SetColors(RED,BLACK);
18
19    ILI9341_Clear(0,0,LCD_X_LENGTH,LCD_Y_LENGTH); /* 清屏，显示全黑 */
20
21    /********显示字符串示例*******/
22    ILI9341_DispStringLine_EN(LINE(0),"BH OV7725 Test Demo");
23
24    USART_Config();
25    LED_GPIO_Config();
```

```
26
27    SysTick_Init();
28    printf("\r\n ** OV7725 摄像头实时液晶显示例程** \r\n");
29
30    /* ov7725 gpio 初始化 */
31    OV7725_GPIO_Config();
32
33    LED_BLUE;
34    /* ov7725 寄存器默认配置初始化 */
35    while (OV7725_Init() != SUCCESS) {
36        retry++;
37        if (retry>5) {
38            printf("\r\n 没有检测到 OV7725 摄像头\r\n");
39   ILI9341_DispStringLine_EN(LINE(2),"No OV7725 module detected!");
40            while (1);
41        }
42    }
43
44    /* 设置液晶扫描模式 */
45    ILI9341_GramScan( 3 );
46
47    ILI9341_DispStringLine_EN(LINE(2),"OV7725 initialize success!");
48    printf("\r\nOV7725 摄像头初始化完成\r\n");
49
50    Ov7725_vsync = 0;
51
52    while (1) {
53        /*接收到新图像进行显示*/
54        if ( Ov7725_vsync == 2 ) {
55            frame_count++;
56            FIFO_PREPARE;       /*FIFO 准备*/
57            ImagDisp(0,0,320,240);      /*采集并显示*/
58
59            Ov7725_vsync = 0;
60            LED1_TOGGLE;
61
62        }
63
64        /*每隔一段时间计算一次帧率*/
65        if (Task_Delay[0] == 0) {
66            printf("\r\nframe_ate = %.2f fps\r\n",frame_count/10);
67            frame_count = 0;
68            Task_Delay[0] = 10000;//该变量在 systick 每 ms 减 1，
69        }
70    }
71 }
72
```

main 函数的执行流程说明如下：

(1) main 函数首先调用了 ILI9341_Init、USART_Config 、SysTick_Init 和 LED_GPIO_Config 等函数初始化液晶、串口、Systick 定时器和 LED 外设，其中 Systick 每 ms 中断一次，为下面计算帧率的代码提供时间。

(2) 接下来 OV7725_GPIO_Config 函数，该函数内部封装了前面介绍的 SCCB_GPIO_Config、FIFO_GPIO_Config 及 VSYNC_GPIO_Config 函数，对控制摄像头使用的相关引脚都进行了初始化。

(3) 初始化好 SCCB 相关的引脚，就可通过 OV7725_Init 向 OV7725 芯片写入配置参数，代码中使用 while 循环在初始化失败时进行多次尝试。

(4) 调用 ILI9341_GramScan 设置液晶屏的扫描方向，使得液晶屏与摄像头的分辨率一致，做好显示的准备。

(5) 在 while 循环中，根据 Ov7725_vsync 标志，判断 FIFO 是否接收完了一幅图像。在中断服务程序中，若检测到两次 VSYNC 的下降沿(表示接收完一幅图像)，会把 Ov7725_vsync 变量设置为 2。

(6) 判断接收完成一幅图像后，调用宏 FIFO_PREPARE 使读 FIFO 指针复位，使 ImagDisp 读取 FIFO 时，能读取得正确的数据并显示到液晶屏。

(7) 记录帧数目的变量 frame_count 加 1，这个变量用来统计帧率，每过一段时间后计算帧率通过串口输出到上位机。最后把 Ov7725_vsync 置 0，使重新开始计数。

OV7725 的其它模式配置

以上是最基本的摄像头采集过程，而提供的工程在以上基础还增加了一些摄像头的配置，包括分辨率、光照度、饱和度、对比度及特殊模式等，如 OV7725_Window_Set、OV7725_Brightness、OV7725_Color_Saturation、OV7725_Contrast 和 OV7725_Special_Effect 等函数，这些函数的本质都是根据函数的输入参数，转化成对应的配置写入到 OV7725 摄像头的寄存器中，完成相应的配置，下面仅以 OV7725_Special_Effect 函数为例进行介绍，见代码清单 1-20。

代码清单 1-20 OV7725_Special_Effect 函数

```
/**
 * @brief  设置特殊效果
 * @param  eff:特殊效果，参数范围[0~6]:
   @arg 0:正常
   @arg 1:黑白
   @arg 2:偏蓝
   @arg 3:复古
   @arg 4:偏红
   @arg 5:偏绿
   @arg 6:反相
 * @retval 无
 */
void OV7725_Special_Effect(uint8_t eff)
{
    switch (eff) {
    case 0://正常
        SCCB_WriteByte(0xa6, 0x06);
        SCCB_WriteByte(0x60, 0x80);
        SCCB_WriteByte(0x61, 0x80);
        break;

    case 1://黑白
        SCCB_WriteByte(0xa6, 0x26);
        SCCB_WriteByte(0x60, 0x80);
        SCCB_WriteByte(0x61, 0x80);
        break;

    case 2://偏蓝
        SCCB_WriteByte(0xa6, 0x1e);
        SCCB_WriteByte(0x60, 0xa0);
        SCCB_WriteByte(0x61, 0x40);
        break;

    case 3://复古
        SCCB_WriteByte(0xa6, 0x1e);
        SCCB_WriteByte(0x60, 0x40);
        SCCB_WriteByte(0x61, 0xa0);
        break;
```

```
39
40      case 4://偏红
41          SCCB_WriteByte(0xa6, 0x1e);
42          SCCB_WriteByte(0x60, 0x80);
43          SCCB_WriteByte(0x61, 0xc0);
44          break;
45
46      case 5://偏绿
47          SCCB_WriteByte(0xa6, 0x1e);
48          SCCB_WriteByte(0x60, 0x60);
49          SCCB_WriteByte(0x61, 0x60);
50          break;
51
52      case 6://反相
53          SCCB_WriteByte(0xa6, 0x46);
54          break;
55
56      default:
57          OV7725_DEBUG("Special Effect error!");
58          break;
59      }
60  }
```

从代码中可了解到，函数支持 0~6 作为输入参数，分别对应不同的模式，在函数内部根据不同的输入对寄存器写入相应配置。

摄像头配置结构体

由于分辨率、光照度、饱和度、对比度及特殊模式等摄像头配置涉及众多内容，特别是关于分辨率的配置，需要与液晶扫描方向匹配，否则容易出现显示错误的现象，为了方便使用，工程中定义了一个结构体类型专门用于设置摄像头的这些配置，在初始化摄像头或想修改配置的时候，修改该变量的内容，然后把它作为参数输入到各种配置函数调用即可。该摄像头配置结构体类型见代码清单 1-21。

代码清单 1-21 摄像头配置结构体（bsp_ov7725.h 文件）

```
1  /*摄像头配置结构体*/
2  typedef struct {
3      uint8_t QVGA_VGA; //0: QVGA 模式，1: VGA 模式
4
5      /*VGA:sx + width <= 320 或 240 ,sy+height <= 320 或 240*/
6      /*QVGA:sx + width <= 320 ,sy+height <= 240*/
7      uint16_t cam_sx; //摄像头窗口 X 起始位置
8      uint16_t cam_sy; //摄像头窗口 Y 起始位置
9
10     uint16_t cam_width;//图像分辨率，宽
11     uint16_t cam_height;//图像分辨率，高
12
13     uint16_t lcd_sx;//图像显示在液晶屏的 X 起始位置
14     uint16_t lcd_sy;//图像显示在液晶屏的 Y 起始位置
15     uint8_t lcd_scan;//液晶屏的扫描模式（0-7）
16
17     uint8_t light_mode;//光照模式，参数范围[0~5]
18     int8_t saturation;//饱和度,参数范围[-4 ~ +4]
19     int8_t brightness;//光照度，参数范围[-4~+4]
20     int8_t contrast;//对比度，参数范围[-4~+4]
21     uint8_t effect; //特殊效果，参数范围[0~6]:
22
23 } OV7725_MODE_PARAM;
```

结构体类型定义中的代码注释有注明各种配置参数的范围，其中 QVGA_VGA 可以配置采样图像的模式，cam_sx/y 可以设置摄像头采样坐标的原点，cam_width/ height 可设置图像分辨率，分辨率调小时，可以提高图像的采集帧率，lcd_sx/y 可以配置图像显示在液晶屏的起始位置，而 lcd_scan 可以设置液晶屏的扫描模式，其余的配置如光照度、饱和度等可以按需求设置。

在配置分辨率时，必须注意调节范围，如 QVGA 模式下最大为 320*240（宽*高），若设置分辨率为 240*320（宽*高）时，由于设置分辨率的高度超出 QVGA 的 240 限制，会导致出错，此时把对应的模式改成 VGA 模式即可，因为 VGA 模式的最大分辨率为 640*480（宽*高），除了不超过 QVGA、VGA 模式的极限外，还要注意它们在液晶屏的扫描模式和起始位置的配置不会超出液晶显示范围。

在工程中，提供了三组摄像头及显示配置范例，实验时可以亲自尝试一下来了解，见代码清单 1-22。

代码清单 1-22 三组摄像头初始化配置（bsp_ov7725.c 文件）

```
1  //摄像头初始化配置
2  //注意：使用这种方式初始化结构体，要在 c/c++选项中选择 C99 mode
3  OV7725_MODE_PARAM cam_mode = {
4
5      /*以下包含几组摄像头配置，可自行测试，保留一组，把其余配置注释掉即可*/
6      /************配置 1*********横屏显示*****************************/
7
8      .QVGA_VGA = 0,  //QVGA 模式
9      .cam_sx = 0,
10     .cam_sy = 0,
11
12     .cam_width = 320,
13     .cam_height = 240,
14
15     .lcd_sx = 0,
16     .lcd_sy = 0,
17     .lcd_scan = 3, //LCD 扫描模式，本横屏配置可用 1、3、5、7 模式
18
19     //以下可根据自己的需要调整，参数范围见结构体类型定义
20     .light_mode = 0,//自动光照模式
21     .saturation = 0,
22     .brightness = 0,
23     .contrast = 0,
24     .effect = 0,    //正常模式
25
26     /**********配置 2*********竖屏显示*****************************/
27     /*竖屏显示需要 VGA 模式，同分辨率情况下，比 QVGA 帧率稍低*/
28     /*VGA 模式分辨率为 640*480，从中取出 240*320 的图像进行竖屏显示*/
29     /*本工程不支持超过 320*240 或 240*320 的分辨率配置*/
30
31 //  .QVGA_VGA = 1,  //VGA 模式
32 //  //取 VGA 模式居中的窗口，可根据实际需要调整
33 //  .cam_sx = (640-240)/2,
34 //  .cam_sy = (480-320)/2,
35 //
36 //  .cam_width = 240,
37 //  .cam_height = 320, //在 VGA 模式下，此值才可以大于 240
38 //
39 //  .lcd_sx = 0,
40 //  .lcd_sy = 0,
41 //  .lcd_scan = 0, //LCD 扫描模式，本竖屏配置可用 0、2、4、6 模式
```

```
//
//  //以下可根据自己的需要调整，参数范围见结构体类型定义
//  .light_mode = 0,//自动光照模式
//  .saturation = 0,
//  .brightness = 0,
//  .contrast = 0,
//  .effect = 0,    //正常模式

    /*******配置 3************小分辨率***************************/
    /*小于 320*240 分辨率的，可使用 QVGA 模式,设置的时候注意液晶屏边界*/

//  .QVGA_VGA = 0,  //QVGA 模式
//  //取 QVGA 模式居中的窗口，可根据实际需要调整
//  .cam_sx = (320-100)/2,
//  .cam_sy = (240-150)/2,
//
//  .cam_width = 100,
//  .cam_height = 150,
//
//  /*液晶屏的显示位置也可以根据需要调整，注意不要超过边界即可*/
//  .lcd_sx = 50,
//  .lcd_sy = 50,
//  .lcd_scan = 3, //LCD 扫描模式，0-7 模式都支持，注意不要超过边界即可

//  //以下可根据自己的需要调整，参数范围见结构体类型定义
//  .light_mode = 0,//自动光照模式
//  .saturation = 0,
//  .brightness = 0,
//  .contrast = 0,
//  .effect = 0,    //正常模式

};
```

代码中使用 OV7725_MODE_PARAM 类型定义了一个 cam_mode 变量并对其结构体成员赋予了初始值。本工程默认使用以上第一组配置，采集 320*240 的 QVGA 图像在液晶屏上横屏显示；而第二组配置是 240*320 的 VGA 图像在液晶屏上竖屏显示，注意两组配置中 QVGA_VGA 和 lcd_mode 变量值的区别；第三组配置是 50*50 的分辨率，由于分辨率比较小，其宽和高都没有超出 QVGA 及液晶屏显示范围，所以液晶 0-7 的扫描方式都支持。可亲自尝试以上各组配置，使用时注释掉其余两组即可，也可以在范例的基础上，自己进行修改测试。

使用摄像头配置结构体时，在初始化摄像头时要调用相应的函数对寄存器进行赋值，所以，main 函数需要作出相应的修改，见代码清单 1-23。

代码清单 1-23 根据摄像头配置结构体初始化的 main 函数

```

extern OV7725_MODE_PARAM cam_mode;
/**
  * @brief  主函数
  * @param  无
  * @retval 无
  */
int main(void)
{
    float frame_count = 0;
    uint8_t retry = 0;

    /* 液晶初始化 */
    ILI9341_Init();
    ILI9341_GramScan ( 3 );
```

```
16
17      LCD_SetFont(&Font8x16);
18      LCD_SetColors(RED,BLACK);
19
20      ILI9341_Clear(0,0,LCD_X_LENGTH,LCD_Y_LENGTH); /* 清屏，显示全黑 */
21      ILI9341_DispStringLine_EN(LINE(0),"BH OV7725 Test Demo");
22
23      USART_Config();
24      LED_GPIO_Config();
25      Key_GPIO_Config();
26      SysTick_Init();
27      printf("\r\n ** OV7725 摄像头实时液晶显示例程** \r\n");
28
29      /* ov7725 gpio 初始化 */
30      OV7725_GPIO_Config();
31
32      LED_BLUE;
33      /* ov7725 寄存器默认配置初始化 */
34      while (OV7725_Init() != SUCCESS) {
35          retry++;
36          if (retry>5) {
37              printf("\r\n 没有检测到 OV7725 摄像头\r\n");
38      ILI9341_DispStringLine_EN(LINE(2),"No OV7725 module detected!");
39              while (1);
40          }
41      }
42
43      /*根据摄像头参数组配置模式*/
44      OV7725_Special_Effect(cam_mode.effect);
45      /*光照模式*/
46      OV7725_Light_Mode(cam_mode.light_mode);
47      /*饱和度*/
48      OV7725_Color_Saturation(cam_mode.saturation);
49      /*光照度*/
50      OV7725_Brightness(cam_mode.brightness);
51      /*对比度*/
52      OV7725_Contrast(cam_mode.contrast);
53      /*特殊效果*/
54      OV7725_Special_Effect(cam_mode.effect);
55
56      /*设置图像采样及模式大小*/
57      OV7725_Window_Set(cam_mode.cam_sx,
58                        cam_mode.cam_sy,
59                        cam_mode.cam_width,
60                        cam_mode.cam_height,
61                        cam_mode.QVGA_VGA);
62
63      /* 设置液晶扫描模式 */
64      ILI9341_GramScan( cam_mode.lcd_scan );
65
66
67
68      ILI9341_DispStringLine_EN(LINE(2),"OV7725 initialize success!");
69      printf("\r\nOV7725 摄像头初始化完成\r\n");
70
71      Ov7725_vsync = 0;
72
73      while (1) {
74          /*接收到新图像进行显示*/
75          if ( Ov7725_vsync == 2 ) {
76              frame_count++;
77              FIFO_PREPARE;       /*FIFO 准备*/
78              ImagDisp(cam_mode.lcd_sx,
79                       cam_mode.lcd_sy,
80                       cam_mode.cam_width,
```

```
81                  cam_mode.cam_height);      /*采集并显示*/
82
83              Ov7725_vsync = 0;
84              LED1_TOGGLE;
85
86          }
87
88          /*检测按键*/
89          if ( Key_Scan(KEY1_GPIO_PORT,KEY1_GPIO_PIN) == KEY_ON  ) {
90              /*LED 反转*/
91              LED2_TOGGLE;
92
93          }
94          /*检测按键*/
95          if ( Key_Scan(KEY2_GPIO_PORT,KEY2_GPIO_PIN) == KEY_ON  ) {
96              /*LED 反转*/
97              LED3_TOGGLE;
98
99              /*动态配置摄像头的模式，
100              有需要可以添加使用串口、用户界面下拉选择框等方式修改这些变量，
101              达到程序运行时更改摄像头模式的目的*/
102
103             cam_mode.QVGA_VGA = 0,  //QVGA 模式
104             cam_mode.cam_sx = 0,
105             cam_mode.cam_sy = 0,
106
107             cam_mode.cam_width = 320,
108             cam_mode.cam_height = 240,
109
110             cam_mode.lcd_sx = 0,
111             cam_mode.lcd_sy = 0,
112             cam_mode.lcd_scan = 3, //LCD 扫描模式，本横屏配置可用 1、3、5、7 模式
113
114             //以下可根据自己的需要调整，参数范围见结构体类型定义
115             cam_mode.light_mode = 0,//自动光照模式
116             cam_mode.saturation = 0,
117             cam_mode.brightness = 0,
118             cam_mode.contrast = 0,
119             cam_mode.effect = 1,    //黑白模式
120
121             /*根据摄像头参数写入配置*/
122             OV7725_Special_Effect(cam_mode.effect);
123             /*光照模式*/
124             OV7725_Light_Mode(cam_mode.light_mode);
125             /*饱和度*/
126             OV7725_Color_Saturation(cam_mode.saturation);
127             /*光照度*/
128             OV7725_Brightness(cam_mode.brightness);
129             /*对比度*/
130             OV7725_Contrast(cam_mode.contrast);
131             /*特殊效果*/
132             OV7725_Special_Effect(cam_mode.effect);
133
134             /*设置图像采样及模式大小*/
135             OV7725_Window_Set(cam_mode.cam_sx,
136                               cam_mode.cam_sy,
137                               cam_mode.cam_width,
138                               cam_mode.cam_height,
139                               cam_mode.QVGA_VGA);
140
141             /* 设置液晶扫描模式 */
142             ILI9341_GramScan( cam_mode.lcd_scan );
143         }
144
145         /*每隔一段时间计算一次帧率*/
```

```
146         if (Task_Delay[0] == 0) {
147             printf("\r\nframe_ate = %.2f fps\r\n",frame_count/10);
148             frame_count = 0;
149             Task_Delay[0] = 10000;
150         }
151     }
152 }
```

相对于前面介绍的摄像头基本初始化过程，本 main 函数主要增加了对 OV7725_Window_Set、OV7725_Brightness、OV7725_Color_Saturation、OV7725_Contrast 和 OV7725_Special_Effect 等函数的调用，根据摄像头配置结构体 cam_mode 向 OV7725 写入寄存器内容，初始化好后，摄像头图像的采集和显示与普通方式无异。

代码中还增加了按键检测，按下了开发板的 KEY2 按键后，会向摄像头配置结构体 cam_mode 赋予新的配置并写入到 OV7725 寄存器中，以上代码把采集的图像设置成了黑白模式。

3. 下载验证

把摄像头模块接入到开发板的摄像头接口上，用 USB 线连接开发板，编译程序下载到实验板并上电复位。即可看到 LCD 上输出摄像头拍到的图像，若图片显示不够清晰，可调整镜头进行调焦，使得到清晰的图像。

1.4 每课一问

1. 若想使得 FIFO 能直接缓存一帧 VGA 模式的图像，FIFO 的容量至少要多大？