开源共享　　携手共进

深圳普中科技有限公司

官方网站：www.prechin.cn

技术论坛：www.prechin.net

技术 QQ：2489019400

咨询电话：18926759791（公司座机）

PZ-0V7670 摄像头模块开发手册

本手册我们将向大家介绍 PZ-0V7670 摄像头模块及其使用。本手册我们将使用 STM32 驱动 PZ-0V7670 摄像头模块，实现摄像头功能。本章分为如下几部分内容：

1 0V7670 介绍

2 硬件设计

3 软件设计

4 实验现象

1 OV7670 介绍

1.1 OV7670 简介

OV7670 是 OV（ OmniVision）公司生产的一颗 1/6 寸的 CMOS VGA 图像传感器。该传感器体积小、工作电压低，提供单片 VGA 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、取窗口等方式的各种分辨率 8 位影像数据。该产品 VGA 图像最高达到 30 帧/秒。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、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、托尾、浮散等，提高图像质量，得到清晰的稳定的彩色图像。

OV7670 的特点有：

（1）高灵敏度、低电压适合嵌入式应用

（2）标准的 SCCB 接口，兼容 IIC 接口

（3）支持 RawRGB、 RGB(GBR4:2:2， RGB565/RGB555/RGB444)， YUV(4:2:2)和 YCbCr（ 4:2:2）输出格式

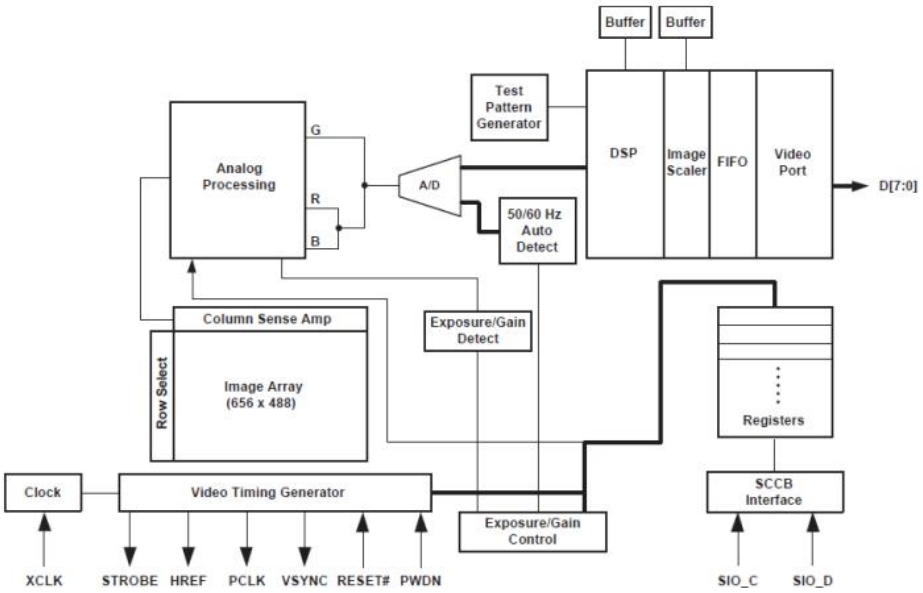
（4）支持 VGA、 CIF，和从 CIF 到 40*30 的各种尺寸输出

（5）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。

（6）支持闪光灯

（7）支持图像缩放

OV7670 的功能框图图如图所示：



OV7670 传感器包括如下一些功能模块：

1.感光整列（ Image Array）

OV7670 总共有 656*488 个像素，其中 640*480 个有效（即有效像素为 30W）。

2.时序发生器（ Video Timing Generator）

时序发生器具有的功能包括：整列控制和帧率发生（ 7 种不同格式输出）、内部信号发生器和分布、帧率时序、自动曝光控制、输出外部时序（ VSYNC、HREF/HSYNC 和 PCLK）。

3.模拟信号处理（ Analog Processing）

模拟信号处理所有模拟功能，并包括：自动增益（ AGC)和自动白平衡（ AWB)。

4.A/D 转换（ A/D）

原始的信号经过模拟处理器模块之后 ，分 G 和 BR 两路进入一个 10 位的 A/D 转换器，A/D 转换器工作在 12M 频率，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，该模块还有以下三个功能：

①黑电平校正（ BLC)

②U/V 通道延迟

③A/D 范围控制

A/D 范围乘积和 A/D 的范围控制共同设置 A/D 的范围和最大值，允许用户

根据应用调整图片的亮度。

5.测试图案发生器（ Test Pattern Generator）

测试图案发生器功能包括：八色彩色条图案、渐变至黑白彩色条图案和输出脚移位“ 1”。

6.数字处理器（ 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：

①边缘锐化（二维高通滤波器）

②颜色空间转换（原始信号到 RGB 或者 YUV/YCbYCr)

③RGB 色彩矩阵以消除串扰

④色相和饱和度的控制

⑤黑/白点补偿

⑥降噪

⑦镜头补偿

⑧可编程的伽玛

⑨十位到八位数据转换

7.缩放功能（ Image Scaler）

这个模块按照预先设置的要求输出数据格式，能将 YUV/RGB 信号从 VGA 缩小到 CIF 以下的任何尺寸。

8.数字视频接口（ Digital Video Port）

通过寄存器 COM2[1:0]，调节 IOL/IOH 的驱动电流，以适应用户的负载。

9.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《 OmniVisionTechnologies Seril Camera Control Bus(SCCB) Specification》这个文档。

10.LED 和闪光灯的输出控制（ LED and Storbe Flash Control Output）

OV7670 有闪光灯模式，可以控制外接闪光灯或闪光 LED 的工作。

OV7670 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，在这里我们不做介绍，请大家参考模块的相关文档。

接下来我们介绍一下 OV7670 的图像数据输出格式。首先我们简单介绍几个

定义：

VGA，即分辨率为 640*480 的输出模式；

QVGA，即分辨率为 320*240 的输出格式，也就是本手册我们需要用到的格式；

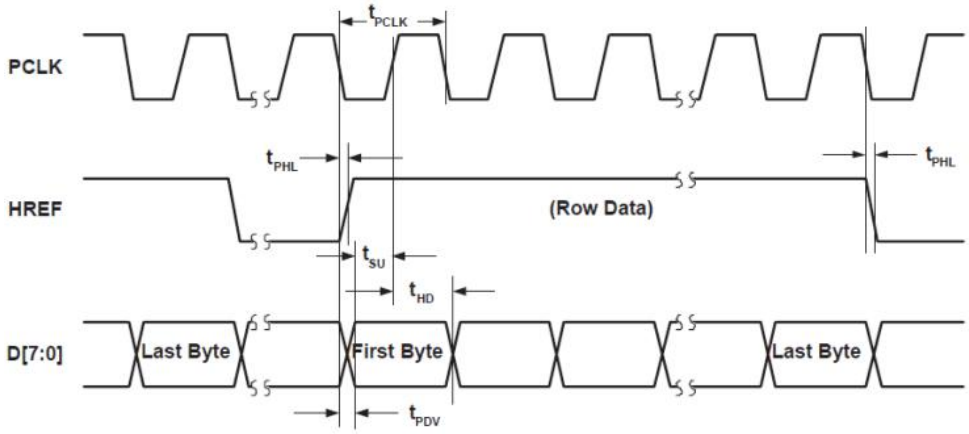
QQVGA，即分辨率为 160*120 的输出格式；

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。

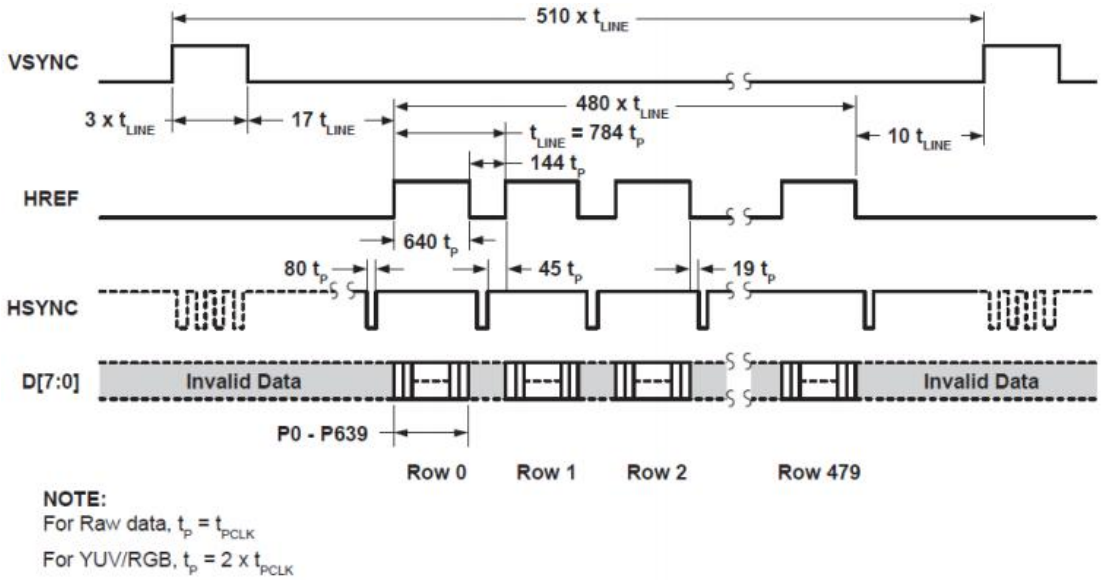
HREF /HSYNC，即行同步信号。

OV7670 的图像数据输出（通过 D[7:0]）就是在 PCLK， VSYNC 和 HREF/HSYNC 的控制下进行的。首先看看行输出时序，如图所示：



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，输出一个字节数据。比如我们采用 VGA 时序， RGB565 格式输出，每 2 个字节组成一个像素的颜色（高字节在前，低字节在后），这样每行输出总共有 640*2 个 PCLK 周期，输出 640*2 个字节。

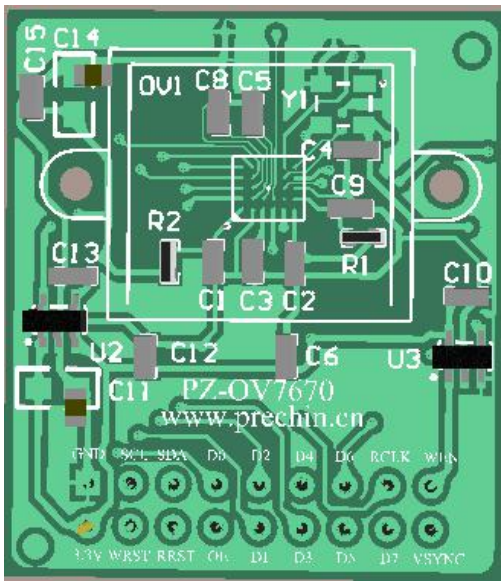
再来看看帧时序（ VGA 模式），如图所示：



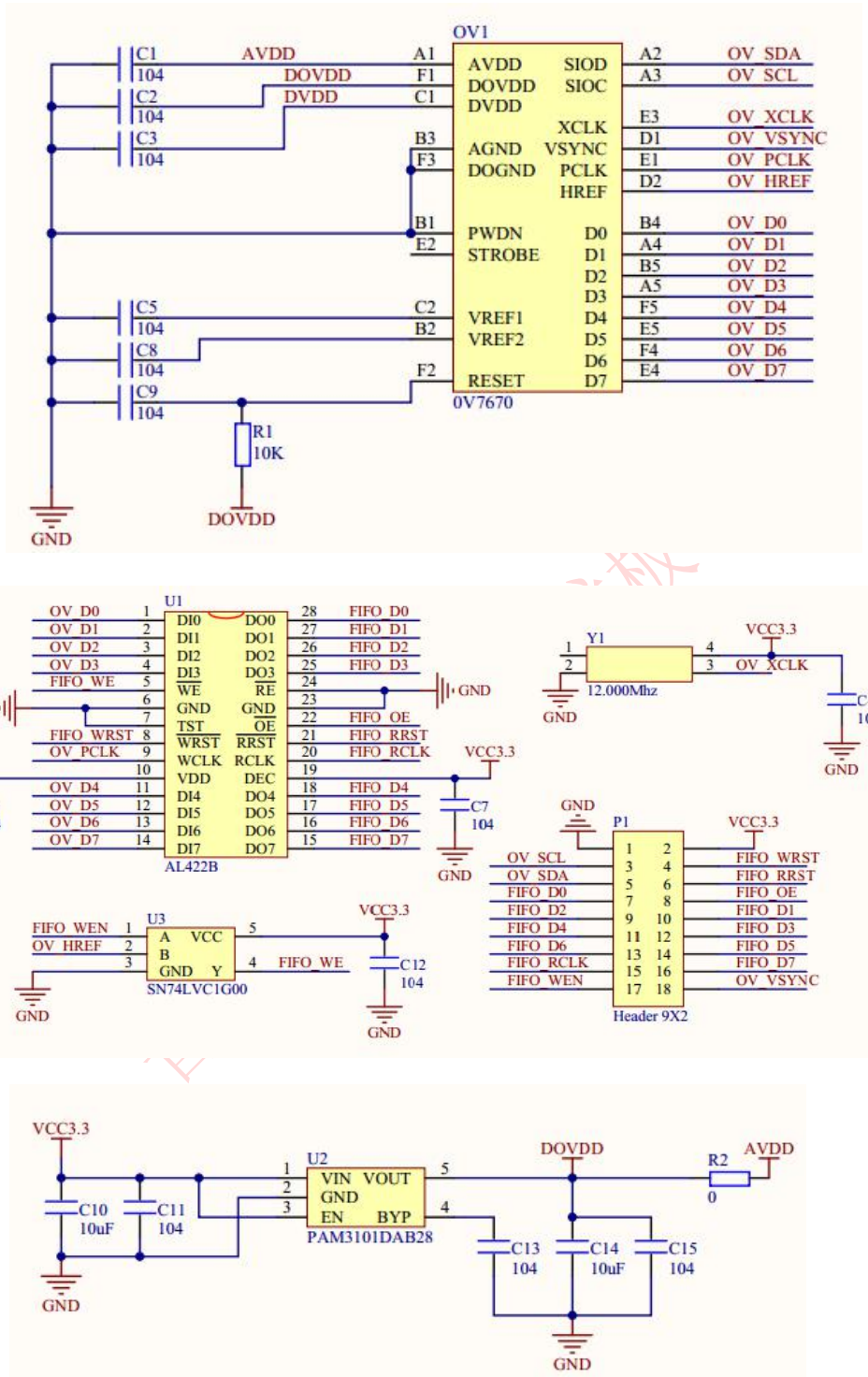
上图清楚的表示了 OV7670 在 VGA 模式下的数据输出，注意，图中的 HSYNC 和 HREF 其实是同一个引脚产生的信号，只是在不同场合下面，使用不同的信号方式，我们用到的是 HREF。

因为 OV7670 的像素时钟（ PCLK）最高可达 24Mhz，我们用 STM32F103ZET6 的 IO 口直接抓取，是非常困难的，也十分占耗 CPU（可以通过降低 PCLK 输出频率，来实现 IO 口抓取，但是不推荐这样操作）。所以，我们并不是采取直接抓取来自 OV7670 的数据，而是通过 FIFO 读取，PZ-OV7670 摄像头模块自带了一个 FIFO 芯片，用于暂存图像数据，有了这个芯片，我们就可以很方便的获取图像数据了，而不再需要单片机具有高速 IO，也不会耗费多少 CPU，可以说，只要是个单片机，都可以通过 PZ-OV7670 摄像头模块实现拍照的功能。

接下来我们介绍一下 PZ-OV7670 摄像头模块。该模块的外观如图所示：



模块原理图如图所示：



从上图可以看出， PZ-OV7670 摄像头模块自带了有源晶振 Y1，用于产生 12M 时钟作为 OV7670 的 XCLK 输入。同时自带了稳压芯片，用于提供 OV7670 稳定的 2.8V 工作电压，并带有一个 FIFO 芯片（AL422B），该 FIFO 芯片的容量是 384K 字节，足够存储 2 帧 QVGA 的图像数据。模块通过一个 2*9 的双排

排针（P1）与外部通信，该管脚功能如图所示：

信号	作用描述	信号	作用描述
VCC3.3	模块供电脚，接 3.3V 电源	FIFO_WEN	FIFO 写使能
GND	模块地线	FIFO_WRST	FIFO 写指针复位
OV_SCL	SCCB 通信时钟信号	FIFO_RRST	FIFO 读指针复位
OV_SDA	SCCB 通信数据信号	FIFO_OE	FIFO 输出使能（片选）
FIFO_D[7:0]	FIFO 输出数据（8 位）	OV_VSYNC	OV7670 帧同步信号
FIFO_RCLK	读 FIFO 时钟		

1.2 PZ-OV7670 模块使用方法

下面我们来看看如何使用 PZ-OV7670 摄像头模块（以 QVGA 模式，RGB565 格式为例）。对于该模块，我们只关心两点： 1，如何存储图像数据； 2，如何读取图像数据。

（1）如何存储图像数据。

PZ-OV7670 摄像头模块存储图像数据的过程为：等待 OV7670 同步信号→FIFO 写指针复位→FIFO 写使能→等待第二个 OV7670 同步信号→FIFO 写禁止。通过以上 5 个步骤，我们就完成了 1 帧图像数据的存储。

（2）如何读取图像数据。

在存储完一帧图像以后，我们就可以开始读取图像数据了。读取过程为：FIFO 读指针复位→给 FIFO 读时钟（ FIFO_RCLK）→读取第一个像素高字节→给 FIFO 读时钟→读取第一个像素低字节→给 FIFO 读时钟→读取第二个像素高字节→循环读取剩余像素→结束。

可以看出，PZ-OV7670 摄像头模块数据的读取也是十分简单，比如 QVGA 模式，RGB565 格式，我们总共循环读取 320*240*2 次，就可以读取 1 帧图像数据，把这些数据写入 LCD 模块，我们就可以看到摄像头捕捉到的画面了。

OV7670 还可以对输出图像进行各种设置，详见模块资料《 OV7670 software application note》文档，对 AL422B 的操作时序，请大家参考 AL422B 的数据手册。

了解了 OV7670 模块的数据存储和读取，我们就可以开始设计程序了，本实验我们用一个外部中断来捕捉帧同步信号（ VSYNC），然后在中断里面启动

OV7670 模块的图像数据存储，等待下一次 VSHNC 信号到来，我们就关闭数据存储，然后一帧数据就存储完成了，在主函数里面就可以慢慢的将这一帧数据读出来，放到 LCD 即可显示了，同时开始第二帧数据的存储，如此循环，实现摄像头功能。

本实验我们将使用摄像头模块的 QVGA 输出（ 320*240），这个在我们使用的 TFTLCD 模块分辨率范围内。注意：PZ-OV7670 摄像头模块自带的 FIFO 是没办法缓存一帧的 VGA 图像的，如果使用 VGA 输出，那么你必须在 FIFO 写满之前开始读 FIFO 数据，保证数据不被覆盖。

2 硬件设计

本实验使用到硬件资源如下：

（1）D1 指示灯

（2）K_UP、K_DOWN、K_LEFT、K_RIGHT 按键

（3）串口 1

（4）TFTLCD 模块

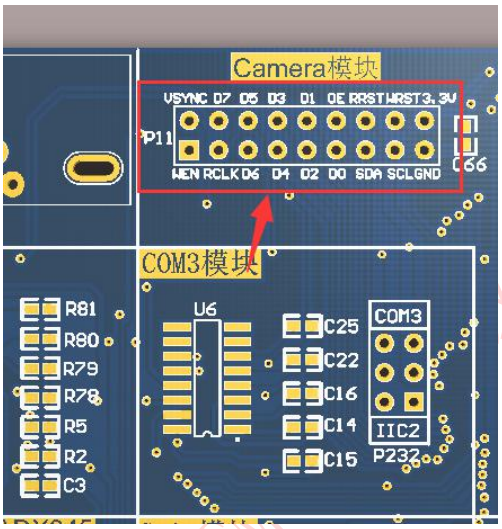
（5）PZ-OV7670 摄像头模块

前四部分电路在前面章节都介绍过，这里就不多说，下面我们来看下 PZ-OV7670 摄像头模块与开发板如何连接的。前面我们介绍了该模块的接口管脚功能，如果您的开发板未含摄像头接口，我们可以通过杜邦线将 PZ-OV7670 摄像头模块与 STM32 开发板的管脚连接，连接关系如图所示：

PZ-OV7670摄像头模块	普中STM32F1开发板（IO）
D0-D7	PF0-PF7
WEN	PB6
RCLK	PB7
VSYNC	PA7
SDA	PC6
SCL	PC4
RRST	PC2
OE	PC3
WRST	PE6
GND	GND
3.3V	3.3V

注意：由于之前 STM32F1 开发板版本上没有摄像头接口，所以连接的导线较多，特别注意，使用杜邦线连接摄像头模块时，注意将数据线单独绑起来，控制信号线单独绑起来，其他的 SDA 和 SCL 及 GND、3.3V 绑在一起，如果没有绑在一起，可能会出现干扰等原因导致画面显示不清楚。

考虑到客户的接线问题，在后面的 STM32F1 开发板中，我们已集成了摄像头接口，如下图所示：



我们只需将 PZ-OV7670 摄像头模块的管脚与开发板上接口对应插入即可，非常方便，而且显示画面也比较清晰。

3 软件设计

本实验所实现的功能为：开机后，初始化摄像头模块（ OV7670），如果初始化成功，则在 LCD 模块上面显示摄像头模块所拍摄到的内容。我们可以通过 K_UP 键设置光照模式（ 5 种模式）、通过 K_DOWN 键设置色饱和度，通过 K_LEFT 键设置亮度，通过 K_RIGHT 键设置对比度。通过串口我们可以查看当前的帧率（这里是指 LCD 显示的帧率，而不是指 OV7670 的输出帧率）。 D1 指示灯提示系统运行状态。

我们打开本实验工程，可以看到我们的工程 APP 列表中多了 ov7670.c 和 sccb.c 源文件，以及头文件 ov7670.h、sccb.h 和 ov7670cfg.h 等 5 个文件。本实验工程代码比较多，我们就不一一列出了，仅挑两个重要的地方进行讲解。首先，我们来看 ov7670.c 里面的 OV7670_Init 函数，该函数代码如下：

//初始化 OV7670

```
//返回 0:成功
//返回其他值:错误代码
u8 OV7670_Init(void)
{
    u8 temp;
    u16 i=0;
    //设置 IO
    GPIO_InitTypeDef  GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOC,ENABLE);

    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;//速度为 50M
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_6|GPIO_Pin_7;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_SetBits(GPIOB,GPIO_Pin_6|GPIO_Pin_7);


    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Pin=0xff;
    GPIO_Init(GPIOF, &GPIO_InitStructure);
```

```
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_6;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    GPIO_SetBits(GPIOE,GPIO_Pin_6);

    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_2|GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    GPIO_SetBits(GPIOC,GPIO_Pin_2|GPIO_Pin_3);

    SCCB_Init();              //初始化 SCCB 的 IO 口
    if(SCCB_WR_Reg(0x12,0x80))return 1;    //复位 SCCB
    delay_ms(50);
    //读取产品型号
    temp=SCCB_RD_Reg(0x0b);
    if(temp!=0x73)return 2;
    temp=SCCB_RD_Reg(0x0a);
    if(temp!=0x76)return 2;
    //初始化序列
    for(i=0;i<sizeof(ov7670_init_reg_tbl)/sizeof(ov7670_init_reg_t
bl[0]);i++)
    {

  SCCB_WR_Reg(ov7670_init_reg_tbl[i][0],ov7670_init_reg_tbl[i][1]);
    }
        return 0x00;  //ok
  }
```

此部分代码先初始化 OV7670 相关的 IO 口（包括 SCCB_Init），然后最主要的是完成 OV7670 的寄存器序列初始化。 OV7670 的寄存器特别多（上百来

个），配置非常麻烦，幸好厂家有提供参考配置序列（详见《 OV7670 software application note》),本实验我们用到的配置序列,存放在 ov7670_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 ov7670cfg.h 里面。

接下来，我们看看 ov7670cfg.h 里面 ov7670_init_reg_tbl 的内容，ov7670cfg.h 文件的代码如下：

```
//初始化寄存器序列及其对应的值
const u8 ov7670_init_reg_tbl[][2]=
{
    /*以下为 OV7670 QVGA RGB565 参数  */
    {0x3a, 0x04},//dummy
    {0x40, 0xd0},//565
    {0x12, 0x14},//QVGA,RGB 输出

    //输出窗口设置
    {0x32, 0x80},//HREF control bit[2:0] HREF start 3 LSB bit[5:3] HSTOP HREF end 3LSB
    {0x17, 0x16},//HSTART start high 8-bit MSB
    {0x18, 0x04},//5 HSTOP end high 8-bit
    {0x19, 0x02},
    {0x1a, 0x7b},//0x7a,
    {0x03, 0x06},//0x0a,帧竖直方向控制

    {0x0c, 0x00},
    {0x15, 0x00},//0x00
    {0x3e, 0x00},//10
    {0x70, 0x3a},
    {0x71, 0x35},
    {0x72, 0x11},
```

```
{0x73, 0x00},//

{0xa2, 0x02},//15
{0x11, 0x81},//时钟分频设置,0,不分频.
{0x7a, 0x20},
{0x7b, 0x1c},
{0x7c, 0x28},

{0x7d, 0x3c},//20
{0x7e, 0x55},
{0x7f, 0x68},
{0x80, 0x76},
{0x81, 0x80},

{0x82, 0x88},
{0x83, 0x8f},
{0x84, 0x96},
{0x85, 0xa3},
{0x86, 0xaf},

{0x87, 0xc4},//30
{0x88, 0xd7},
{0x89, 0xe8},
{0x13, 0xe0},
{0x00, 0x00},//AGC

{0x10, 0x00},
{0x0d, 0x00},//全窗口， 位[5:4]: 01 半窗口，10 1/4 窗口，11 1/4
```
窗口

```
{0x14, 0x28},//0x38, limit the max gain
{0xa5, 0x05},
{0xab, 0x07},

{0x24, 0x75},//40
{0x25, 0x63},
{0x26, 0xA5},
{0x9f, 0x78},
{0xa0, 0x68},

{0xa1, 0x03},//0x0b,
{0xa6, 0xdf},//0xd8,
{0xa7, 0xdf},//0xd8,
{0xa8, 0xf0},
{0xa9, 0x90},

{0xaa, 0x94},//50
{0x13, 0xe5},
{0x0e, 0x61},
{0x0f, 0x4b},
{0x16, 0x02},

{0x1e, 0x27},//图像输出镜像控制.0x07
{0x21, 0x02},
{0x22, 0x91},
{0x29, 0x07},
{0x33, 0x0b},

{0x35, 0x0b},//60
```

```
{0x37, 0x1d},
{0x38, 0x71},
{0x39, 0x2a},
{0x3c, 0x78},

{0x4d, 0x40},
{0x4e, 0x20},
{0x69, 0x00},
{0x6b, 0x40},//PLL*4=48Mhz
{0x74, 0x19},
{0x8d, 0x4f},

{0x8e, 0x00},//70
{0x8f, 0x00},
{0x90, 0x00},
{0x91, 0x00},
{0x92, 0x00},//0x19,//0x66

{0x96, 0x00},
{0x9a, 0x80},
{0xb0, 0x84},
{0xb1, 0x0c},
{0xb2, 0x0e},

{0xb3, 0x82},//80
{0xb8, 0x0a},
{0x43, 0x14},
{0x44, 0xf0},
{0x45, 0x34},
```

```
{0x46, 0x58},
{0x47, 0x28},
{0x48, 0x3a},
{0x59, 0x88},
{0x5a, 0x88},

{0x5b, 0x44},//90
{0x5c, 0x67},
{0x5d, 0x49},
{0x5e, 0x0e},
{0x64, 0x04},
{0x65, 0x20},

{0x66, 0x05},
{0x94, 0x04},
{0x95, 0x08},
{0x6c, 0x0a},
{0x6d, 0x55},


{0x4f, 0x80},
{0x50, 0x80},
{0x51, 0x00},
{0x52, 0x22},
{0x53, 0x5e},
{0x54, 0x80},

//{0x54, 0x40},//110
```

```
    {0x09, 0x03},//驱动能力最大

    {0x6e, 0x11},//100
    {0x6f, 0x9f},//0x9e for advance AWB
    {0x55, 0x00},//亮度
    {0x56, 0x40},//对比度 0x40
    {0x57, 0x40},//0x40,  change according to Jim's request
    ////////////////////////////////////////////////////////////////////
//////
    //以下部分代码由开源电子网网友:duanzhang512 提出
    //添加此部分代码将可以获得更好的成像效果,但是最下面一行会有蓝色的抖动.
    //如不想要,可以屏蔽此部分代码.然后将:OV7670_Window_Set(12,176,240,320);
    //改为:OV7670_Window_Set(12,174,240,320);,即可去掉最下一行的蓝色抖动
    {0x6a, 0x40},
    {0x01, 0x40},
    {0x02, 0x40},
    {0x13, 0xe7},
    {0x15, 0x00},


    {0x58, 0x9e},

    {0x41, 0x08},
    {0x3f, 0x00},
```

```
{0x75, 0x05},
{0x76, 0xe1},
{0x4c, 0x00},
{0x77, 0x01},
{0x3d, 0xc2},
{0x4b, 0x09},
{0xc9, 0x60},
{0x41, 0x38},

{0x34, 0x11},
{0x3b, 0x02},

{0xa4, 0x89},
{0x96, 0x00},
{0x97, 0x30},
{0x98, 0x20},
{0x99, 0x30},
{0x9a, 0x84},
{0x9b, 0x29},
{0x9c, 0x03},
{0x9d, 0x4c},
{0x9e, 0x3f},
{0x78, 0x04},

{0x79, 0x01},
{0xc8, 0xf0},
{0x79, 0x0f},
{0xc8, 0x00},
{0x79, 0x10},
```

```
    {0xc8, 0x7e},
    {0x79, 0x0a},
    {0xc8, 0x80},
    {0x79, 0x0b},
    {0xc8, 0x01},
    {0x79, 0x0c},
    {0xc8, 0x0f},
    {0x79, 0x0d},
    {0xc8, 0x20},
    {0x79, 0x09},
    {0xc8, 0x80},
    {0x79, 0x02},
    {0xc8, 0xc0},
    {0x79, 0x03},
    {0xc8, 0x40},
    {0x79, 0x05},
    {0xc8, 0x30},
    {0x79, 0x26},
    {0x09, 0x00},
    ////////////////////////////////////////////////////////////////
//////

};
```

以上代码，我们大概了解下结构，每个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{0x3a, 0x04}，就表示在 0X03 地址，写入 0X04 这个值。

通过这么一长串（ 110 多个）寄存器的配置，我们就完成了 OV7670 的初始化，本实验我们配置 OV7670 工作在 QVGA 模式， RGB565 格式输出。 在完成初始化之后，我们既可以开始读取 OV7670 的数据了。

OV7670 文件夹里面的其他代码我们就不逐个介绍了，请大家参考该例程源码。

因为本实验我们还用到了帧率（ LCD 显示的帧率）统计和中断处理，所以我们还需要修改 time.c、 time.h、 exti.c 及 exti.h 这几个文件。

在 time.c 里面，我们使用 TIM4 用于统计帧率，其代码如下：

```
/**************************************************************************
***************
* 函 数 名          : TIM4_Init
* 函数功能          : TIM4 初始化函数
* 输    入          : per:重装载值
                      psc:分频系数
* 输    出          : 无
**************************************************************************
**************/
void TIM4_Init(u16 per,u16 psc)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);//使能 TIM4
时钟

    TIM_TimeBaseInitStructure.TIM_Period=per;   //自动装载值
    TIM_TimeBaseInitStructure.TIM_Prescaler=psc; //分频系数
    TIM_TimeBaseInitStructure.TIM_ClockDivision=TIM_CKD_DIV1;
    TIM_TimeBaseInitStructure.TIM_CounterMode=TIM_CounterMode_Up;
//设置向上计数模式
    TIM_TimeBaseInit(TIM4,&TIM_TimeBaseInitStructure);
```

```
    TIM_ITConfig(TIM4,TIM_IT_Update,ENABLE); //开启定时器中断
    TIM_ClearITPendingBit(TIM4,TIM_IT_Update);

    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;//定时器中断通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=2;// 抢 占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority =3;        //子优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;          //IRQ通道使能
    NVIC_Init(&NVIC_InitStructure);

    TIM_Cmd(TIM4,ENABLE); //使能定时器
}

u8 ov_frame; //统计帧数

/**********************************************************************
***************
* 函 数 名          : TIM4_IRQHandler
* 函数功能          : TIM4 中断函数
* 输    入          : 无
* 输    出          : 无
**********************************************************************
**************/
void TIM4_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM4,TIM_IT_Update))
    {
```

```
            led2=!led2;
            printf("frame:%dfps\r\n",ov_frame);    //打印帧率
            ov_frame=0;
        }
        TIM_ClearITPendingBit(TIM4,TIM_IT_Update);
    }
```

这里我们用到通用定时器 TIM4 来统计帧率，也就是 1 秒钟中断一次，打印 ov_frame 的值， ov_frame 用于统计 LCD 帧率。

在 exti.c 里面添加 EXTI7_Init 和 EXTI9_5_IRQHandler 函数，用于 OV7670 模块的 FIFO 写控制， exti.c 文件新增部分代码如下：

```
u8 ov_sta;    //帧中断标记
 //外部中断 5~9 服务程序
void EXTI9_5_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line7)==SET) //是 8 线的中断
    {
        OV7670_WRST=0;    //复位写指针
        OV7670_WRST=1;
        OV7670_WREN=1;    //允许写入 FIFO
        ov_sta++;     //帧中断加 1
    }
    EXTI_ClearITPendingBit(EXTI_Line7);  //清除 EXTI8 线路挂起位

}
//外部中断 7 初始化
void EXTI7_Init(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
```

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);

    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource7);

    EXTI_InitStructure.EXTI_Line=EXTI_Line7;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);    //根据 EXTI_InitStruct 中指
定的参数初始化外设 EXTI 寄存器

    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;          //
使能按键所在的外部中断通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;   //
抢占优先级 0
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //子优先级 0
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        //使能外部中断通道
    NVIC_Init(&NVIC_InitStructure);      //根据 NVIC_InitStruct 中
指定的参数初始化外设 NVIC 寄存器
}
```

因为 OV7670 的帧同步信号（ OV_VSYNC）接在 PA7 上面，所以我们这里配置 PA7 作为中断输入，因为 STM32 的外部中断 5~9 共用一个中断服务函数（ EXTI9_5_IRQHandler），所以在该函数里面，我们需要先判断中断是不是来自中断线 7 的，然后再做处理。

中断处理部分很简单，通过一个 ov_sta 来控制 OV7670 模块的 FIFO 写操作。当 ov_sta=0 的时候，表示 FIFO 存储的数据已经被成功读取了（ ov_sta 在

读完 FIFO 数据的时候被清零），然后只要 OV_VSYNC 信号到来，我们就先复位一下写指针，然后 ov_sta=1，标志着写指针已经复位，目前正在往 FIFO 里面写数据。再等下一个 OV_VSYNC 到来，也就表明一帧数据已经存储完毕了，此时我们设置 OV7670_WREN 为 0，禁止再往 OV7670 写入数据，此时 ov_sta 自增为 2。其他程序，只要读到 ov_sta 为 2，就表示一帧数据已经准备好了，可以读出，在读完数据之后，程序设置 ov_sta 为 0，则开启下一轮 FIFO 数据存储。

最后我们看下 main.c 文件，代码如下：

```
extern u8 ov_sta;    //在 exit.c 里面定义
extern u8 ov_frame;  //在 time.c 里面定义



//更新 LCD 显示
void camera_refresh(void)
{
   u32 j;
   u16 i;
   u16 color;
   u16 temp;
   if(ov_sta)//有帧中断更新？
   {
```

```
//LCD_Set_Window((tftlcd_data.width-320)/2,(tftlcd_data.height-240)/2,320,240-1);//将显示区域设置到屏幕中央

LCD_Set_Window(0,(tftlcd_data.height-240)/2,320-1,240-1);//将显示区域设置到屏幕中央
```

```
OV7670_RRST=0;                    //开始复位读指针
OV7670_RCK_L;
OV7670_RCK_H;
OV7670_RCK_L;
OV7670_RRST=1;                    //复位读指针结束
OV7670_RCK_H;
/*for(i=0;i<240;i++)    //此种方式可以兼容任何彩屏,但是速度很慢
{
    for(j=0;j<320;j++)
    {
        OV7670_RCK_L;
        color=GPIOF->IDR&0XFF;   //读数据
        OV7670_RCK_H;
        color<<=8;
        OV7670_RCK_L;
        color|=GPIOF->IDR&0XFF;  //读数据
        OV7670_RCK_H;
        LCD_DrawFRONT_COLOR(j,i,color);
    }
}*/
for(j=0;j<76800;j++)    //此种方式需清楚 TFT 内部显示方向控制寄存器值 速度较快
{
    OV7670_RCK_L;
    color=GPIOF->IDR&0XFF;   //读数据
    OV7670_RCK_H;
    color<<=8;
    OV7670_RCK_L;
```

```
            color|=GPIOF->IDR&0XFF;  //读数据
            OV7670_RCK_H;
            LCD_WriteData_Color(color);
            //printf("%x  ",color);
            //if(j%20==0)printf("\r\n");
            //delay_us(50);
        }
        ov_sta=0;                   //清零帧中断标记
        ov_frame++;

    }
}

const u8*LMODE_TBL[5]={"Auto","Sunny","Cloudy","Office","Home"};
const
u8*EFFECTS_TBL[7]={"Normal","Negative","B&W","Redish","Greenish","Blu
ish","Antique"};  //7 种特效


int main()
{
    u8 i=0;
    u8 key;
    u8 lightmode=0,saturation=2,brightness=2,contrast=2;
    u8 effect=0;
    u8 sbuf[15];
    u8 count;

    SysTick_Init(72);
```

```
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  //中断优先级
分组 分 2 组
    LED_Init();
    USART1_Init(9600);
    TFTLCD_Init();           //LCD 初始化
    KEY_Init();
    EN25QXX_Init();              //初始化 EN25Q128
    my_mem_init(SRAMIN);     //初始化内部内存池

    FRONT_COLOR=RED;//设置字体为红色

  // while(SD_Init()!=0)
  // {
  //
  LCD_ShowString(10,10,tftlcd_data.width,tftlcd_data.height,16,"SD
Card Error!");
  // }
  // FATFS_Init();                    //为 fatfs 相关变量申请内存

  //     f_mount(fs[0],"0:",1);                 //挂载 SD 卡
  //     f_mount(fs[1],"1:",1);              //挂载 FLASH.

    LCD_ShowFont12Char(10, 10, "普中科技");
    LCD_ShowFont12Char(10, 30, "www.prechin.net");
    LCD_ShowFont12Char(10, 50, "摄像头应用--OV7670");
    i=OV7670_Init();
    printf("i=%d\n",i);
    while(OV7670_Init())//初始化 OV7670
    {
```

```
    LCD_ShowString(10,80,tftlcd_data.width,tftlcd_data.height,16,"OV7
670 Error!");
            delay_ms(200);
            LCD_Fill(10,80,239,206,WHITE);
            delay_ms(200);
        }

    LCD_ShowString(10,80,tftlcd_data.width,tftlcd_data.height,16,"OV7
670 OK!     ");
        delay_ms(1500);
        OV7670_Light_Mode(0);
        OV7670_Color_Saturation(2);
        OV7670_Brightness(2);
        OV7670_Contrast(2);
        OV7670_Special_Effects(0);

        TIM4_Init(10000,7199);          //10Khz 计数频率,1 秒钟中断

        EXTI7_Init();
        OV7670_Window_Set(12,176,240,320); //设置窗口
        OV7670_CS=0;
        LCD_Clear(BLACK);
        while(1)
        {
            key=KEY_Scan(0);
            if(key)count=20;
            switch(key)
            {
```

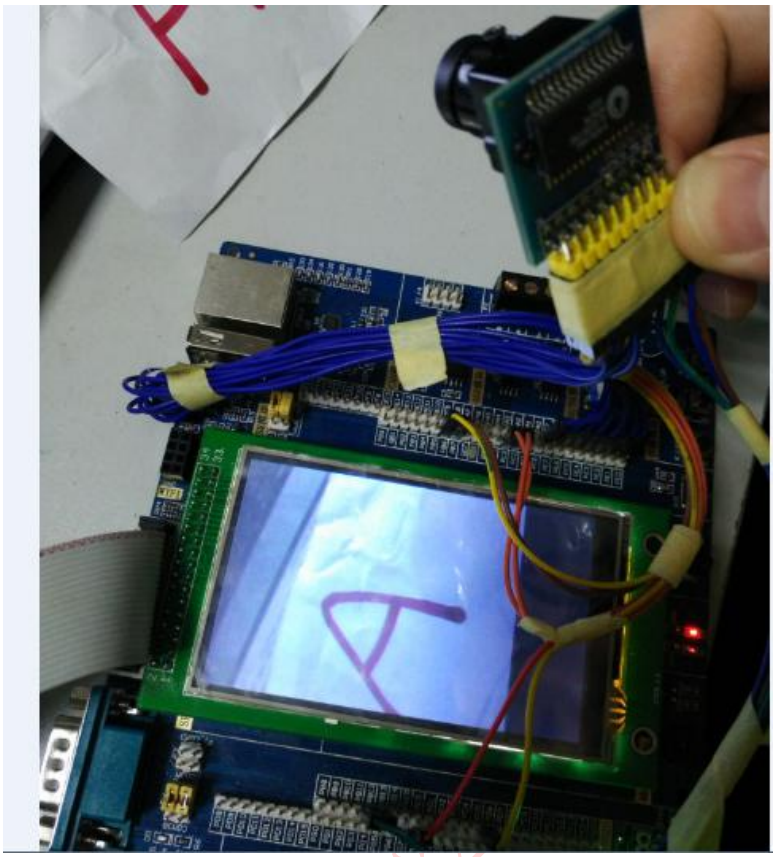
```
            case KEY_UP:            //灯光模式设置
                lightmode++;
                if(lightmode>4)lightmode=0;
                OV7670_Light_Mode(lightmode);
                sprintf((char*)sbuf,"%s",LMODE_TBL[lightmode]);
                break;
            case KEY_DOWN:          //饱和度
                saturation++;
                if(saturation>4)saturation=0;
                OV7670_Color_Saturation(saturation);

sprintf((char*)sbuf,"Saturation:%d",(char)saturation-2);
                break;
            case KEY_LEFT:          //亮度
                brightness++;
                if(brightness>4)brightness=0;
                OV7670_Brightness(brightness);

sprintf((char*)sbuf,"Brightness:%d",(char)brightness-2);
                break;
            case KEY_RIGHT:     //对比度
                contrast++;
                if(contrast>4)contrast=0;
                OV7670_Contrast(contrast);

sprintf((char*)sbuf,"Contrast:%d",(char)contrast-2);
                break;
        }
        if(count)
```

```
            {
                count--;

    LCD_ShowString((tftlcd_data.width-240)/2+30,(tftlcd_data.height-3
20)/2+60,200,16,16,sbuf);
            }
            camera_refresh();//更新显示
            i++;
            if(i%20==0)
            {
                led1=!led1;
            }
            //delay_ms(5);
        }

    }
```

此部分代码除了 mian 函数，还有一个 camera_refresh 函数，该函数用于将摄像头模块 FIFO 的数据读出，并显示在 LCD 上面。 main 函数则比较简单，我们就不细说了。

4 实验现象

将工程程序编译下载到开发板内并且将模块连接到开发板上，注意对于早期 STM32F1 开发板的要将摄像头的线扎好，以防干扰导致画面模糊，可以看到 TFTLCD 上显示如下界面：（在最新 STM32F1 开发板上我们已经集成了摄像头接口，所以只需将模块按照管脚顺序对应插入到接口内即可，而无需繁琐的接线）



此时，我们可以按不同的按键（ K_UP、K_DOWN、K_LEFT 、K_RIGHT），来设置摄像头的相关参数和模式，得到不同的成像效果。还可以通过串口打印输出 LCD 显示的帧率，如图所示：

从上图还可以看出， LCD 显示帧率为 8 帧左右，则可以推断 OV7670 的输出帧率则至少是 3*8=24 帧以上（实际是 30 帧）。