STM32 OV7670 开发套件

Zidong404

一、OV7670 模块（V2）

1.1 概要

OV7670 模块板载了有源晶振和视频 FIFO，FIFO 最大容量为 384KB (3M 位)，可以存放一帧 640*480(30 万像素)RGB RAW 格式的图片，演示代码将摄像头配置为了 320*240 RGB565 格式输出，这样像素大小和数据格式刚好对应 2.8 寸 TFT 彩屏，数据不用经过转换便能直接使用。另外 OV7670 还支持 YVU422 和 RGB RAW 的输出格式，详细寄存器配置请参考相关的数据手册。V2 版本有 P1 和 P2 两个接口，两个接口引脚功能是相同的，P1 是 2.54mm 间距，方便在没有 STM32 主板的情况下通过跳线与其他单片机系统连接调试，P2 是 2.0mm 间距的，可以直接插在 STM32 主板上的 P2 插槽(注意摄像头应该面向屏的一方)。其中有两个引脚 XCLK 和 PWDN 是扩展功能，由于模块上已经有有源晶振给摄像头提供时钟，所以 XCLK 不用再提供时钟；PWDN 用来配置摄像头进入低功耗模式，默认情况下不进入该模式。通常情况下，只有这两个引脚悬空，其他的引脚都是需要使用的。

1.2 LDO电路

V2 版本增加了一个 LDO（U4）电路提供摄像头内核电压，由于 OV7670 内部具有 LDO 功能，所以默认情况下该 LDO 没有焊接。提供一个 LDO 电路是为了能兼容其他内部不带 LDO 电路的摄像头。

1.3 模块工作原理

OV7670 与 FIFO 的组合要需要注意一定的时序。一个简单而有效的组合是这样的：OV 的场同步接入 MCU 的外部中断(上升沿触发)，这样当一个场同步的到来时(在较窄的高电平到来后，接着就是有效的场同步了【低电平】)开启 FIFO 的写时使能，这样数据就在行同步与 MCU 的控制下按照像素时钟依次写入 FIFO。当一场图像数据已经进入了 FIFO 以后，也标志着下一场数据将要来到，也即下一个场同步的上升沿即将到来，当下一个上升沿触发并进入中断后，MCU 应该关掉 FIFO 的写使能，然后开始读 FIFO 的数据，知道数据读完以后再开始下一次的数据采集。所以调试的第一步是一个正常工作的外中断。主要注意刚进外中断时应该先清中断标志再进行中断操作，否则系统可能一直处于中断嵌套中。

再一点是 FIFO 的读写指针复位。FIFO 的这两个指针复位对于数据同步很重要，如果不能正常复位，得到图像会出现移位或畸变。按照 FIFO 的 datasheet，指针的复位分别有两种：Write Reset ，Read Reset；RE RRST，WE WRST。在读复位时采用的是 Read Reste，要求 RE 和 OE 都为低，然后在读复位引脚为低时重复两个以上的读时钟。写复位使用的是

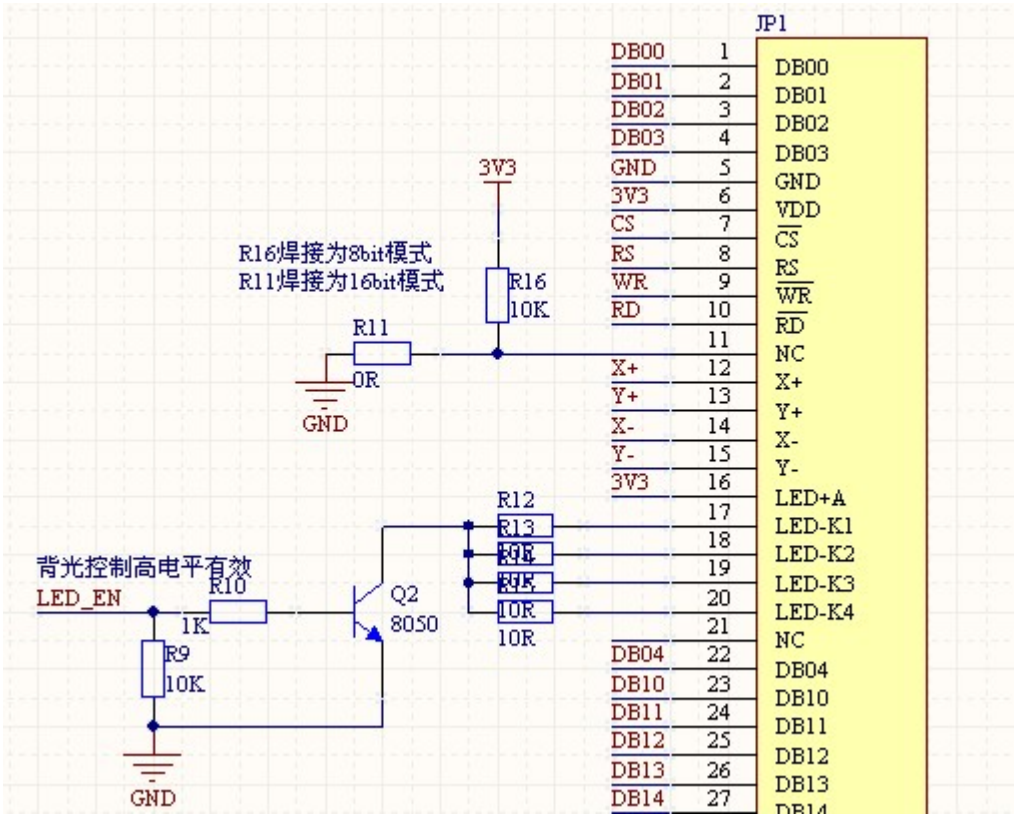
WE WRST，当 WE 为高时，拉低写复位引脚，稍微演示后拉高写复位引脚，WE 引脚状态不用再管，交由中断程序正常控制，只要记住写复位的时候 WE 要先为高即可。

当一帧数据缓存进入 FIFO 以后，单片机关掉 FIFO 的写使能，转而从 FIFO 中读取数据进行处理，这时的处理可以是将数据写入 LCD 进行显示或者存入 SD 卡完成拍照功能。数据存储较简单的一种格式是 BMP，它只用给每一帧数据加上合适的 BMP 文件头，然后依次填入图像数据即可。演示工程实现了图像显示和 BMP 存储的功能，类似于一个简易的数码相机。

二、LCD模块

LCD 模块使用的是 2.8 寸 TFT 模组，支持 8/16 位数据长度，RGB565 格式。TFT 模组有 ILI9325，ILI9320 等几种控制器型号，凡是该套件提供的 LCD 屏都有相应兼容的驱动程序。演示工程使用的是 8 位数据长度，LCD 模块在出厂时默认为 8 位数据长度，用户如果要使用 16 位数据长度，请将模块背面的 R11 短接或焊接一个 0 欧姆电阻。相应的 LCD_Driver 驱动程序只需改动一个宏定义即可兼容 16 位数据长度。注意在使用简易数码相机的演示工程时，屏只能工作在 8 位模式，因为屏的低 8 位数据接口和摄像头模块的 D0-D7 数据口是复用的。从另一点说，8 位模式省去了 IO 模拟驱动 LCD 时的数据移位、相或等操作，较 16 位模式速度要快。

LCD 模块的背光默认是关闭的，要使用屏时需要先打开背光，背光控制高电平有效。见下图：

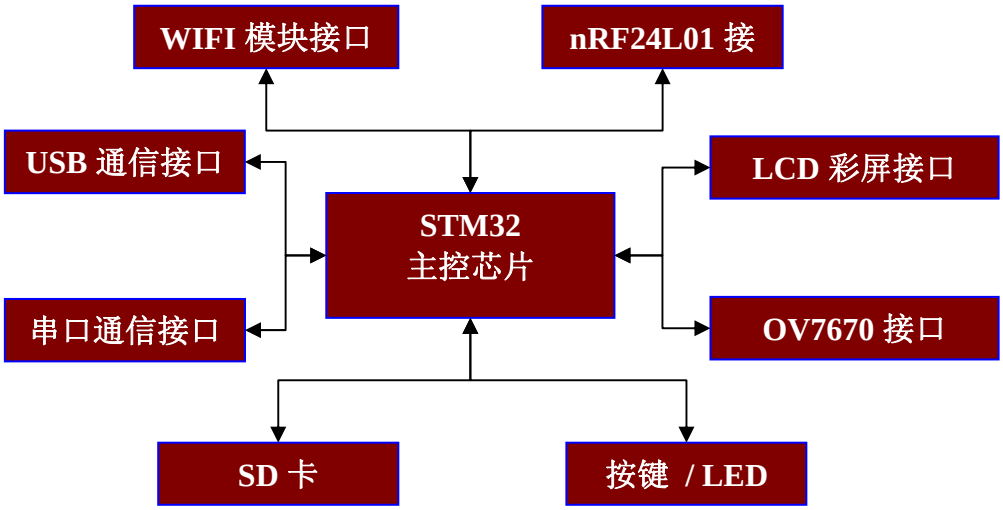


LCD 模块背板有触摸屏驱动控制器电路，作为简易简易数码相机的演示，默认情况下该控制器没有焊接，另外 SD 卡接口电路已经在 STM32 主板上，所

以该卡槽也未焊接。用户可根据扩展需要要求焊接相应的电路。

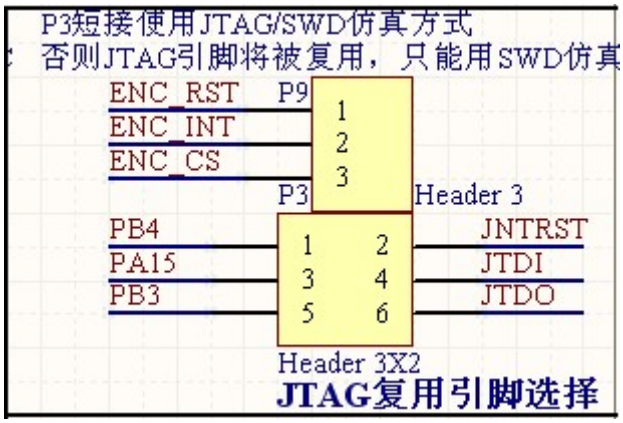
三、STM32 主板

下图是 STM32 主板的功能框图：



STM32 主板板载串口电路，RTC 供电，USB，SD 卡，LCD 接口，摄像头接口，nRF24L01 无线模块接口，WIFI 接口(88W8686)。我们提供相应的接口模块，“即插即用”。

主板上有跳针选择 JTAG 和 SWD 下载方式，JTAG 的引脚是与 WIFI 模块复用的，所以要使用 WIFI 接口时应选择 SWD 仿真方式，具体看下图：



主板上提供了 BOOT0 和 BOOT1 的跳针选择，不仅可以使用仿真器进行下载仿真，还能通过串口和 USB 下载程序，具体使用方法请参见 ST 官方的文档和相应软件。

四、演示工程代码

演示工程分为两部分，分别是简易数码相机的拍照功能演示和 WIFI 模块的 SPI 固件下载演示。第一部分的演示涵盖了 SD 卡，LCD，OV 摄像头，FatFs，BMP，相关的软件都是模块化封装，便于用户快速移植使用。工程使用了 3.3 的固件库，使用 MDK4.12 编译，MDK4.0 以上的编译器都能打开。该工程还包括了 USB 固件库，并对固件库进行了注释和串口打印设备枚举信息(具体演示见下文)。

4.1 数码相机拍照

4.1.1 工程文件组织

工程使用了 3.3 固件库，和固件库有关的代码都放在工程文件夹下的 Libraries 文件夹下；

Doc 文件夹下是相关的版本更新和新建一个基于固件库工程的步骤截图；

MDK_Project 文件夹下存放的是工程索引和相关的工程配置文件，这些配置文件是 MDK 编译器生成的；

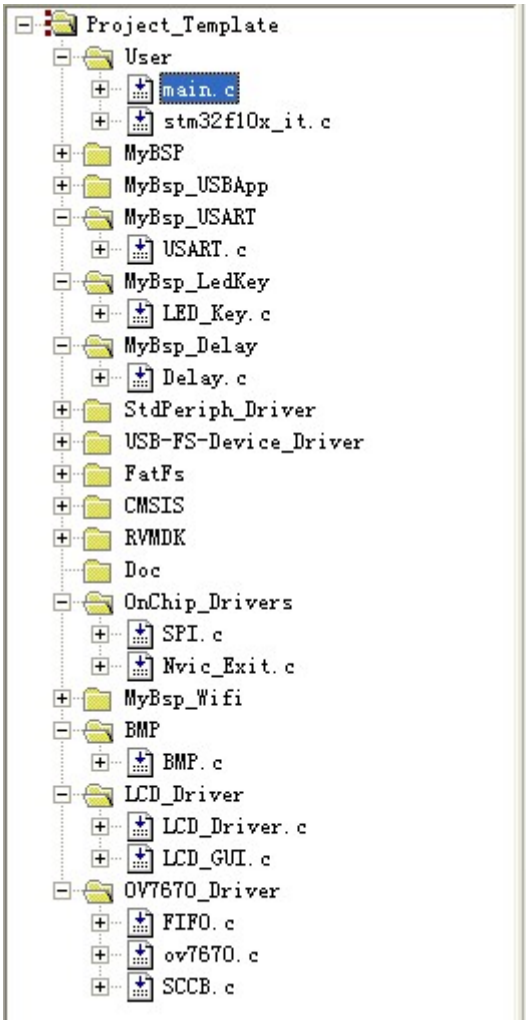
MyBSP 文件下是和 STM32 主板板级外设有关的代码，各板级外设对应一个模块文件名，里面是模块化的驱动代码，比如 SD_Driver，LCD_Driver，Wifi，USB 等，用户可以根据需要添加或移植相关的功能模块，其中 MyBspInc.h 里面包含的是使用到的外设模块(注意文件层级关系)，如此便不用再在工程配置里面添加相关的包含路径。

OnChip_Drivers 文件下对片级使用的外设如 SPI，EXIT 等进行了二次封装以方便使用，如 SPI 模块提供了 SPI 传输位数，传输模式，片选引脚等的快速配置 API 函数，用户可以不使用该部分代码而直接调用固件库函数。同样 OnChip_Driver_Inc.h 里面包含了使用到的片级外设模块的头文件。

Output 文件下是编译生成的连接文件等，*.out 文件等；

User 文件下包含了 main.c，FatFs，BMP，STM32_cfg 文件夹等，这些文件或模块是跟用户应用息息相关的，用户可以根据自己的应用进行裁剪配置。特别的，STM32_cfg 下的 stm32f10x_conf.h 文件用来配置使用到的固件库；STM32_it 下的 stm32f10x_it.c/h 是有关的中断入口函数，所有的中断处理函数，如外中断，USB 中断等执行函数的入口都在这里实现。

以下是 MDK 的工程文件关联截图：



值得注意的是，这只是我们提供的一个工程参考模板，用户可以根据需要裁剪或移植里面相关的模块。

4.1.2 BMP模块

我们封装的 BMP 模块目前可以实现 32/24/16 位 BMP 的读取显示和 16 位 BMP 的存储，BMP 分为文件头，信息头，可选的调色板和数据区几个部分，其中文件头和信息头已经封装成了结构体，由于 BMP 格式的关系，这两个结构体要按照一字节对齐，而 MDK 默认为 4 字节对齐，所以在组织这两个结构时需要加上：

```
#pragma pack (1)   //1 字节对齐
…..
…..
#pragma pack ()          //恢复编译器默认对齐
```

如此才能使用类型强制转换，实现结构体各字段的准确定位。有关 BMP 的详细介绍请参考相关的文档介绍。

4.1.3 LCD_Driver模块

该模块封装了有关 LCD 基本的初始化、画点、读点、块写等 API 函数，支持 8/16 位，支持较多常用的驱动 IC(9320,9325,1514，具体看初始化函数的 switch 分支等)。在 LCD_Driver.h 中的宏定义：

#define LCD_8_BIT_EN 1

如果为 1 则为 8 位模式，如果为 0 则为 16 位模式。

同样在该头文件中有相关的硬件接口宏定义，如果需要移植代码，根据注释修改相关定义接口即可。

4.1.4 FatFs模块

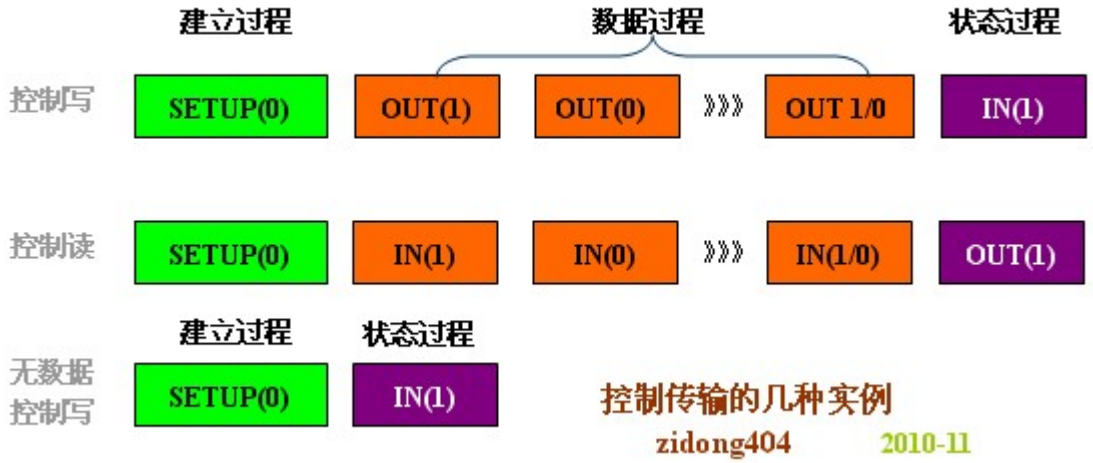
FatFs 是一个开源是文件系统，可以方便的移植到 8 位、16 位和 32 位嵌入式 MCU 上，该工程模板已经完成了 FatFs 的移植，使用的是较新的 R008a 版本，SD 卡驱动方面支持 128M-8GB 的卡(更大容量的卡没有测试过)，文件系统相关的 API 都已经测试过，包括格式化。有关 FatFs 的使用请参考具体的文档介绍。

4.1.5 USB模块

该工程添加了 USB 固件库，并对 usb_core.c 添加了注释，该文件的函数完成了 USB 枚举的过程，而 USB 设备的枚举是开发 USB 中最重要的一部分。同时枚举过程中交换的数据和步骤可以通过 USART1 输出到电脑上，用户可以通过宏定义控制要打印的信息：

```
////////////调试信息输出控制/////////////
//输出基本枚举步骤
#define USB_DEBUG0                  1
//输出详细通信数据
#define USB_DEBUG1                  0
//输出请求字符串描述的请求代码
#define USB_DEBUG_STR_DESC    0
//////////////////////////////////////////
```

工程演示代码完了一个模拟 SD 卡读卡器的设备，当在 SD 卡槽插入 SD 卡以后，可以在电脑上看到一个移动存储设备，并可以往里面考入考出数据。该 Mass Storage 是在 ST 代码的基础实现的，并对 SD 卡的操作部分进行了优化，传输出速度可以到 500KB/S 左右，这个速度还有优化的余地。每个控制传输都是由建立过程开始的，最后结束于 In 状态过程或者是 Out 状态过程。通过该输出信息，用户可以方便地追逐枚举的步骤和通信的数据，以实现自己的设备枚举。



下图是串口的输出信息(部分截图)：

```
SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www

Setup0 中断-->建立过程
设备可以接收新的数据
有数据的建立过程:
获取描述符(0x06)(1设备;2配置;3字符;4接口;5端点):
->设备->设备描述符(18字节)
设备返回: 0x0012个数据
设备正在返回数据->DataStageIn()...
本次传输返回: 0x0012个数据
IN0中断-->In数据过程
IN数据过程完成,即将进入Out状态过程
Out0中断-->Out状态过程

Setup0 中断-->建立过程
设备可以接收新的数据
无数据的建立过程
标准请求->设备请求设置地址
IN0中断-->IN状态过程
设置设备地址: 0x02

Setup0 中断-->建立过程
设备可以接收新的数据
有数据的建立过程:
获取描述符(0x06)(1设备;2配置;3字符;4接口;5端点):
->设备->设备描述符(18字节)
设备返回: 0x0012个数据
设备正在返回数据->DataStageIn()...
本次传输返回: 0x0012个数据
IN0中断-->In数据过程
IN数据过程完成,即将进入Out状态过程
Out0中断-->Out状态过程

Setup0 中断-->建立过程
设备可以接收新的数据
有数据的建立过程:
获取描述符(0x06)(1设备;2配置;3字符;4接口;5端点):
->设备->配置描述符(9字节+(接口)9*x +(端点)7*x)
设备返回: 0x0009个数据
设备正在返回数据->DataStageIn()...
本次传输返回: 0x0009个数据
IN0中断-->In数据过程
IN数据过程完成,即将进入Out状态过程
Out0中断-->Out状态过程
```

用户要使用 U 盘的演示，直接调用 USB_StorageTest()即可，该函数在 main.c 里面定义。

4.1.6 OV7670 和FIFO

OV7670 的寄存器较多，以下是我遇到的一些目前认为较为重要的寄存器，用户可根据 Datasheet 介绍对寄存器进行配置：

0x12：这个寄存器用来设置图像的输出格式，有 VGA(640*480),QVGA(320*240)等，另外就是配置图像数据的格式，有 YUV，RGB565，Bayer RGB RAW。最后这个寄存器的最高位是用来软件复位所有寄存器的值的。

0x71：这个寄存器对于调试的时候是很有用的，通过将其值配置为 0x80，可以让摄像头输出 8 条彩带，当采集不到数据，或采集到的图像不正确的时候不妨设置一下该寄存器的值看图像是否为彩带，或是发生了什么样的畸变。下图是采集到的彩带图形(其实彩带图形 0x70 和 0x71 配合使用的，一般设置 0x70 寄存器为 0 即可)。

0x11：这个寄存器用来配置 OV 的内部时钟相对于外部时钟的分频，即内部时钟是通过外部时钟分频得到的。资料上说 OV 的外部时钟典型值是 24M，我按要求给了，但是当设置分频 3 时按道理像素时钟应该为 8M，但得到的结果并非如此，但通过设置不同的分频系数，像素时钟确实是按照分频系数线性变化的，具体的时钟变化公式需要进一步研究。这个寄存器也是通过与 0x6b 寄存器配合使用的。接下来讲一下 0x6b。

0x6b：这个寄存器是内部时钟的倍频系数配置寄存器。我也尚未清楚为何有了分频(0x11)还要一个倍频。通过实验确实发现这两个寄存器的配置以不同的组合出现时对应的像素时钟是安线性关系改变的。但无论怎么组合，最后的内部时钟是无法超过外部时钟的。这个寄存器还可以配置内部 LDO 是否开启(默认开启)。开启了内部 LDO 功能后硬件上可以少一个 1.8V 的线性稳压器给内核供电。

FIFO 的控制主要在于读写复位指针的控制，当一帧数据缓存进入 FIFO 以后，MCU 要关掉 FIFO 的写使能，然后从 FIFO 中读取数据进行处理，在读写过程，读写指针都是自动增加的。如果 FIFO 要缓存下一帧数据，就需要将起写指针复位到 0 地址，读指针也是同样的道理。

程序上是通过捕获 VSYNC 场同步实现的前后台同步的，场同步是上升沿有效，当一个场同步触发 MCU 中断以后，在中断程序中首先开启 FIFO 的写使能，这时图像数据便在 PCLK 像素时钟的同步下依次写入 FIFO，当下一个场同步到的时候中断会再次触发，这时便要关闭写使能，因为 FIFO 中已经缓存了一帧待处理的数据。main()函数中会一直查询 FIFO 中是否有一帧数据的标志，具体是通过判断 VsyncCnt 是否等于 2 实现的。当 VsyncCnt 等于 2 时，说明 FIFO 中已经缓存了一帧图片，MCU 便从 FIFO 中取出数据进行处理(如显示到 LCD 或者进行颜色识别等)。当处理完一帧数据后，MCU 会清零 VsyncCnt，并复位读写指针，开始下一次采集处理。

4.2 WIFI固件下载

WIFI 模块的核心是目前大量使用的 88W8686 SOC，STM32 通过 SPI1 与其连接，片选为复用的 JTAG 引脚 PB3，所以要使用 WIFI 时需要关闭 JTAG 仿真功能，使用 SWD 进行仿真。WIFI 模块支持 SPI 和 SDIO 接口，详细的介绍请参考另一篇文档：STM32 下 WiFi 开发套件的使用说明.doc。

4.3 实物图片

店铺地址：

http://store.taobao.com/shop/view_shop.htm?asker=wangwang&shop_nick=zidong404