零死角玩转STM32

与野火同行 乐意惬无边

1 原创教程，完全开源。

2 通俗易懂，详尽解读。

3 由浅入深，结合实操。

4 配套板子，全面玩转。

5 强强联合，不断更新。

野火团队 Wild Fire Team

0、 友情提示

《零死角玩转 STM32》系列教程由初级篇、中级篇、高级篇、系统篇、四个部分组成，根据野火 STM32 开发板旧版教程升级而来，且经过重新深入编写，重新排版，更适合初学者，步步为营，从入门到精通，从裸奔到系统，让您零死角玩转 STM32。M3 的世界，与野火同行，乐意惬无边。

另外，野火团队历时一年精心打造的《STM32 库开发实战指南》将于今年 10 月份由机械工业出版社出版，该书的排版更适于纸质书本阅读以及更有利于查阅资料。内容上会给你带来更多的惊喜。是一本学习 STM32 必备的工具书。敬请期待！

6、UsbDevice（模拟 U 盘）

6.1 实验描述及工程文件清单

实验描述	这是一个 USB Mass Storage 实验，用 USB 线连接 PC 机与开发板，在电脑上就可以像操作普通 U 盘那样来操作开发板中的 MicroSD 卡，并在超级终端中打印出相应的调试信息。在这里野火用的 MicroSD 卡的容量是 1G。
硬件连接	PE3-USB-MODE (PE3 为普通 I/O) PA11-USBDM(D-) PA12-USBDP(D+)
用到的库文件	startup/start_stm32f10x_hd.c CMSIS/core_cm3.c CMSIS/system_stm32f10x.c FWlib/stm32f10x_gpio.c FWlib/stm32f10x_rcc.c FWlib/stm32f10x_usart.c FWlib/misc.c FWlib/stm32f10x_dma.c FWlib/stm32f10x_sdio.c FWlib/stm32f10x_flash.c
用户编写的文件	USER/main.c USER/stm32f10x_it.c USER/usart1.c USER/sdcard.c USER/usb_istr.c USER/usb_prop.c

	USER/usb_pwr.c USER/hw_config.c USER/memory.c
USB 文件	usb_core.c usb_init.c usb_mem.c usb_regs.c usb_bot.c usb_scsi.c usb_data.c usb_desc.c usb_endp.c

其中 USER 文件夹下红色标注的 5 个 c 文件也是 USB 库文件，只因为我们需要修改它们才没有把它们放在 USBLIB 目录下。

stm32f10x_it.c：该文件中包含 USB 中断服务程序，由于 USB 中断有很多情况，这里的中断服务程序只是调用 usb_Istr.c 文件中的 USB_Istr 函数，由 USB_Istr 函数再做轮询处理。

usb_istr.c：该文件中只有一个函数，即 USB 中断的 USB_Istr 函数，该函数对各类引起 USB 中断的事件作轮询处理。

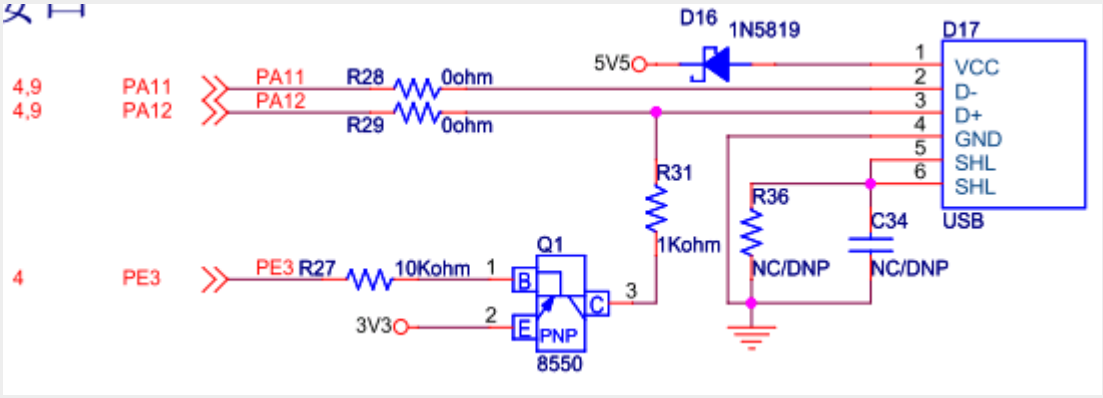
usb_prop.c：该文件用于实现相关设备的 USB 协议，例如初始化、SETUP 包、IN 包、OUT 包等等。

usb_pwr.c：该文件中包含处理上电、调电、挂起和恢复事件的函数。

memory.c：该文件中包含 USB 读写 SD 卡的函数。

hw_config.c：该文件中包含系统配置的函数。

野火 STM32 开发板中 USB 硬件原理图



6.2 USB 简介

USB 模块为 PC 主机和微控制器所实现的功能之间提供了符合 USB 规范的通信连接。PC 主机和微控制器之间的数据传输是通过共享一专用的数据缓冲区来完成的，该数据缓冲区能被 USB 外设直接访问。这块专用数据缓冲区的大小由所使用的端点数目和每个端点最大的数据分组大小所决定，每个端点最大可使用 512 字节缓冲区，最多可用于 16 个单向或 8 个双向端点。

USB 模块同 PC 主机通信，根据 USB 规范实现令牌分组的检测，数据发送/接收的处理，和握手分组的处理。整个传输的格式由硬件完成，其中包括 CRC 的生成和校验。每个端点都有一个缓冲区描述块，描述该端点使用的缓冲区地址、大小和需要传输的字节数。当 USB 模块识别出一个有效的功能/端点的令牌分组时，(如果需要传输数据并且端点已配置)随之发生相关的数据传输。

USB 模块通过一个内部的 16 位寄存器实现端口与专用缓冲区的数据交换。在所有的数据传输完成后，如果需要，则根据传输的方向，发送或接收适当的握手分组。在数据传输结束时，USB 模块将触发与端点相关的中断，通过读状态寄存器和/或者利用不同的中断处理程序，微控制器可以确定：

- 哪个端点需要得到服务
- 产生如位填充、格式、CRC、协议、缺失 ACK、缓冲区溢出/缓冲区未满等错误时，正在进行的是哪种类型的传输。

有关更多 USB 的介绍请参考《STM32 参考手册中文》。USB 是一个很复杂的设备，要想全面的学习 USB，光靠做完这个实验和看《STM32 参考手册中文》是远远不够的，还要大量阅读 ST 的官方 USB 文档。还有网上有本关于 USB 方面的书《圈圈教你学 USB》很不错，大家可以去买来研究研究。总之，一句话，USB 耐学，^_^。

6.3 友情提示

USB 是一个比较复杂的知识点，单单 USB 就可以出一本书，在这里野火不可能为大家讲解地非常详细，但这个文档可以为大家扫清代码阅读的障碍。要想更深入的学习 USB，仍需大家的努力，网上有本叫《圈圈教你学 USB》的书就将得非常好，有兴趣的朋友可以买来看看，虽然有电子版，但大伙还是买本书支持下圈圈，毕竟圈圈写书也不容易呀^_^……

6.4 实验讲解

在配置好我们需要用到的库文件之后，我们就从 main 函数开始分析，关于如何添加库文件请参考前面的教程，这里不再详述。

```
/*
 * 函数名：main
 * 描述  ：无
 * 输入  ：无
 * 输出  ：无
 */
int main(void)
{
    /* 配置系统时钟为 72M */
    SystemInit();

    /* 串口 1 配置 15200-8-N-1 */
    USART1_Config();

    /* SD 卡中断配置，优先级最高 */
    NVIC Configuration();

    /* 等待 SD 卡底层初始化成功 */
    while ( SD_USER_Init() != SD_OK );


    /* 获取 SD 卡容量信息，并在串口打印出来 */
    Get_Medium_Characteristics();

    /* 配置 USB 时钟为 48M */
    Set USBClock();

    /* 配置 USB 中断 */
    USB_Interrupts_Config();
```

```
30.
31.     /* 配置 USB D+ 线为全速模式 */
32.     USB Cable Config (ENABLE);
33.
34.     /* USB 系统初始化 */
35.     USB Init();
36.
37.    /* 等待 USB 中断到来，在中断函数中进行数据的传输 */
38.     while (1)
39.     {
40.     }
41. }
```

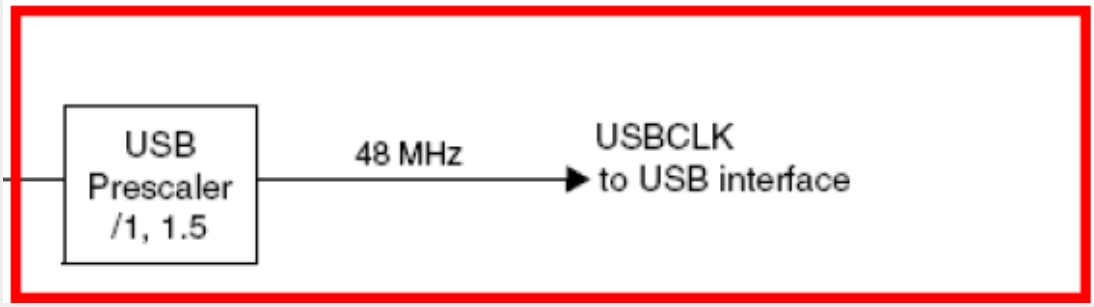
系统库函数 `SystemInit();`将系统时钟配置为 72MHZ， `USART1_Config();`初始化等下要用到的串口 1，用于打印 MicroSD 卡的容量信息。有关这两个函数的详细介绍请参考前面的教程，这里不再详述。

`NVIC_Configuration();`用于配置 MicroSD 卡的中断优先级，本实验中配置为最高优先。

`while ( SD_USER_Init() != SD_OK );`用于等待 MicroSD 卡底层硬件初始化成功，`SD_USER_Init()`在 sdcard.c 中实现。只有这个初始化成功了，接下来才能通过 USB 的方式来访问，所以才采用 `while ( );`的写法，如果初始化不成功的话则一直等待，直到初始化成功为止。`SD_USER_Init()`在 main.c 中实现。

`Get_Medium_Characteristics();`用来获取 MicroSD 卡的容量信息，并通过串口 1 在超级终端中打印出来。`Get_Medium_Characteristics();`也在 main.c 中实现。

`Set_USBClock();`将 USB 的时钟设置为 48MHZ，其实 USB 的时钟必须设置为 48MHZ，如下图所示，截图来自《STM32 参考手册中文》第 47 页。`Set_USBClock();`在 hw_config.c 中实现。




```
1.  /*
2.   * 函数名：Set_USBClock
3.   * 描述  ：配置 USB 时钟(48M)
4.   * 输入  ：无
5.   * 输出  ：无
```

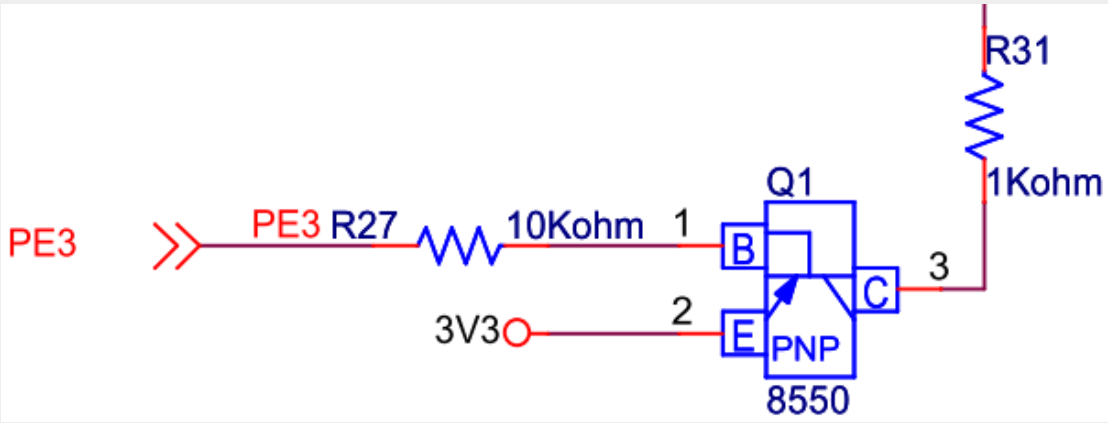
```
6.   */
7.  void Set_USBClock(void)
8.  {
9.     /* USBCLK = PLLCLK */
10.   RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5);
11.
12.   /* Enable USB clock */
13.   RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE);
14. }
```

USB_Interrupts_Config();用于配置 USB 的中断优先级，本实验中 USB 的中断优先级次于 MicroSD 卡的中断优先级。USB_Interrupts_Config();在 hw_config.c 中实现。

USB_Cable_Config (ENABLE); 用于配置 USB D+ 这跟数据线为全速模式，即 D+这根数据线接一个上拉电阻。当 D-数据线接上拉电阻时则工作于低速模式。USB_Cable_Config (); 在 hw_config.c 中实现：

```
1.  /*
2.   * 函数名: USB_Cable_Config
3.   * 描述  : Software Connection/Disconnection of USB Cable
4.   * 输入  : -NewState: new state
5.   * 输出  : 无
6.   */
7.  void USB_Cable_Config (FunctionalState NewState)
8.  {
9.      GPIO_InitTypeDef GPIO_InitStructure;
10.     RCC APB2PeriphClockCmd(RCC APB2Periph GPIOE, ENABLE);
11.
12.   /* PE3 输出 0 时 D+ 接上拉电阻工作于全速模式 */
13.   GPIO InitStructure.GPIO Pin = GPIO Pin 3;
14.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
15.   GPIO InitStructure.GPIO Mode = GPIO Mode Out OD;      /* 开漏输出 */
16.   GPIO_Init(GPIOE, &GPIO_InitStructure);
17.
18.   if (NewState != DISABLE)
19.   {
20.     GPIO_ResetBits(GPIOE, GPIO_Pin_3);                    /* USB 全速模式 */
21.   }
22.   else
23.   {
24.     GPIO_SetBits(GPIOE, GPIO_Pin_3);                          /* 普通模式 */
25.   }
26. }
```

在硬件设计上我们通过 I/O 控制一个三极管的通断来决定 D+这跟数据线是否上拉：



`USB_Init();`用于系统初始化，`USB_Init();`在 usb_init.c 中实现：

```
1.  /*******************************************************
2.  * Function Name  : USB_Init
3.  * Description    : USB system initialization
4.  * Input          : None.
5.  * Output         : None.
6.  * Return         : None.
7.  *******************************************************/
8.  void USB_Init(void)
9.  {
10.   pInformation = &Device_Info;
11.   pInformation->ControlState = 2;
12.   pProperty = &Device_Property;
13.   pUser_Standard_Requests = &User_Standard_Requests;
14.   /* Initialize devices one by one */
15.   pProperty->Init();
16. }
```

如果我们在调试程序时，不成功的话，一般都是这个函数出了问题，而一般的问题又会是硬件的问题，我当初调试的时候就郁闷了很久。

最后让程序在一个 `while (1) { }` 无限循环总等待 USB 中断的到来，然后进行中断服务程序的处理。USB 中断函数 `void USB_Istr(void)` 在 usb_istr.c 中实现：

```
1.  /*******************************************************
2.  * Function Name  : USB_Istr
3.  * Description    : ISTR events interrupt service routine
4.  * Input          : None.
5.  * Output         : None.
6.  * Return         : None.
7.  *******************************************************/
8.  void USB_Istr(void)
```

```
{
  wIstr =  GetISTR();

#if (IMR_MSK & ISTR_RESET)
  if (wIstr & ISTR_RESET & wInterrupt_Mask)
  {
    _SetISTR((u16)CLR_RESET);
    Device Property.Reset();
#ifdef RESET CALLBACK
    RESET Callback();
#endif
  }
#endif
  /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#if (IMR_MSK & ISTR_DOVR)
  if (wIstr & ISTR_DOVR & wInterrupt_Mask)
  {
    _SetISTR((u16)CLR_DOVR);
#ifdef DOVR_CALLBACK
    DOVR Callback();
#endif
  }
#endif
  /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#if (IMR_MSK & ISTR_ERR)
  if (wIstr & ISTR_ERR & wInterrupt_Mask)
  {
    _SetISTR((u16)CLR_ERR);
#ifdef ERR CALLBACK
    ERR Callback();
#endif
  }
#endif
  /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#if (IMR_MSK & ISTR_WKUP)
  if (wIstr & ISTR_WKUP & wInterrupt_Mask)
  {
     SetISTR((u16)CLR WKUP);
    Resume(RESUME_EXTERNAL);
#ifdef WKUP_CALLBACK
    WKUP Callback();
#endif
  }
#endif
  /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#if (IMR_MSK & ISTR_SUSP)
  if (wIstr & ISTR_SUSP & wInterrupt_Mask)
  {

    /* check if SUSPEND is possible */
    if (fSuspendEnabled)
    {
      Suspend();
    }
    else
    {
      /* if not possible then resume after xx ms */
      Resume(RESUME LATER);
    }
    /* clear of the ISTR bit must be done after setting of CNTR FSUSP */
    _SetISTR((u16)CLR_SUSP);
#ifdef SUSP_CALLBACK
    SUSP_Callback();
#endif
  }
#endif
  /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#if (IMR_MSK & ISTR_SOF)
  if (wIstr & ISTR_SOF & wInterrupt_Mask)
  {
    _SetISTR((u16)CLR_SOF);
    bIntPackSOF++;

#ifdef SOF_CALLBACK
    SOF Callback();
#endif
#endif
  }
```

```
#endif
  /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/
#if (IMR_MSK & ISTR_ESOF)
  if (wIstr & ISTR_ESOF & wInterrupt_Mask)
  {
    _SetISTR((u16)CLR_ESOF);
    /* resume handling timing is made with ESOFs */
    Resume(RESUME_ESOF); /* request without change of the machine state */

#ifdef ESOF_CALLBACK
    ESOF_Callback();
#endif
  }
#endif
          /*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
    */
        #if (IMR_MSK & ISTR_CTR)
          if (wIstr & ISTR_CTR & wInterrupt_Mask)
          {
            /* servicing of the endpoint correct transfer interrupt */
            /* clear of the CTR flag into the sub */
            CTR_LP();
        #ifdef CTR_CALLBACK
            CTR_Callback();
        #endif
          }
        #endif
        } /* USB_Istr */
```

实验到了这里，我们已经可以通过 USB 来操作我们开发板上的 MicroSD 卡了。有关更多实验的细节请大家阅读源码。

6.5 实验现象

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上 MicroSD 卡，插上方口的 USB 线，将编译好的程序下载到开发板，如果程序运行成功，则可在电脑上看到开发板上的 U 盘(我用的是 1G 的卡)，如下所示：

打开 U 盘，可看到里面的文件。还可以进行新建、复制、粘贴、格式化等操作。与操作我们的普通 U 盘没什么区别。