

# USB 项目开发手册

作者：向仔州

## 目录

USB 发展背景概述 .....	2
USB 编解码方式 .....	3
USB 四种传输方式 .....	6
USB 传输概念术语 .....	6
USB 主从设备通信 .....	7
控制传输解析 .....	11
USB 键盘报文基本解析 .....	12
STM32 驱动 PDIUSBD12 芯片，实现 USB 驱动 .....	15
STM32 与 PDIUSBD12 芯片硬件连接 .....	15
STM32 与 PDIUSBD12 芯片驱动代码 .....	15
USB 连接与断开 .....	18
USB 端点中断测试 .....	19
STM32F103 USB 实现鼠标功能 .....	20
USB 鼠标数据格式讲解 .....	24
STM32 配置本设备 USB 在电脑上的名字 .....	24
STM32 USB 鼠标测试 .....	25
STM32F103 USB 实现虚拟串口(CDC) .....	25
STM32F103 USB 虚拟串口接收数据 .....	34
STM32F103 USB 虚拟单字节发送数据 .....	35
STM32F103 USB 实现 UVC 摄像头(各种描述符讲解) .....	36
STM32F103 USB 实现 UVC 摄像头设备初步移植 .....	40
STM32F103 USB 使用 UVC 协议发送图片的函数实现原理 .....	42
UVC 描述符内容讲解 .....	42
UVC 传输 jpg 图片数据到 PC .....	48
STM32F103 USB 实现多个虚拟串口 .....	51
STM32F103 USB 读卡器功能(norflash 为例) .....	51

## USB 发展背景概述

- 简单易用 使用统一制式电缆和连接进行外设扩展，即插即用，支持热插拔
- 稳定性佳 使用差分信号传输，较强的纠错能力，多种差错管理和恢复机制
- 速度选择 1. 5Mbps / 12Mbps / 480Mbps / 5Gbps / 10Gbps / 20Gbps 多种等级
- 使用灵活 提供了适合各种应用的传输类型、协议
- 应用广泛 协议标准向下兼容，系统集成驱动，扩展性强，连接支持127个外部设备，拓扑结构，复合设备等。



- USB1.1: 规范了USB低全速传输
- USB2.0: 规范了USB高速传输
- USB3.0: 采用8b/10b编码，增加一对超高速差分线，供电5V/0.9A
- USB3.1: 采用128b/132b编码，速度提高一倍，供电20V/5A，同时增加了A/V影音传输标准
- USB3.2: 增加一对超高速传输通道，速度再次翻倍，只能在C型接口上运行  
也就是Type C接口上才能实现USB3. 2协议，和  
两对超高速差分线

USB2. 0 Full Speed	即 USB1. 1	USB1. 1就是USB2. 0的全速版
USB2. 0 High Speed	即 USB2. 0	
USB3. 2 gen1	即 USB3. 0	
USB3. 2 gen2	即 USB3. 1	
USB3. 2 gen2*2	即 USB3. 2	

- USB 速度等级: 1.5Mbps / 12Mbps / 480Mbps / 5Gbps / 10Gbps / 20Gbps

1. 5Mbps对应的是低速传输，一般使用的键盘，鼠标，人机接口设备都是走的USB1. 1传输，低速设备。

12Mbps传输，早期都是用在音频，声卡上面。

480Mbps高速设备，一般用在存储类设备，U盘，SD卡，摄像头，wifi等..

### 12\*2针接口

Type-C正反插，共4对超高速差分线

### 针脚 名称

1	GND	GND
2	TX1+	RX1+
3	TX1-	RX1-
4	VBUS	VBUS
5	CC1	SBU2
6	D+	D-
7	D-	D+
8	SBU1	CC2
9	VBUS	VBUS
10	RX2-	TX2-
11	RX2+	TX2+
12	GND	GND

3. TypeC接口，主要  
增加了四对高速差分  
信号线，而且可以支  
持正反插。20Gbps就  
必须用满TypeC的四  
对高速差分线。

### 9针接口

2. 九针接口增加了两对超高速差分线StaA\_SSRX-，StaA\_SSRX+，StaA\_SSTX-，StaA\_SSTX+，该接口主要用于USB3. 0/3. 1协议

### 4针接口

1. 最常规的4针接口，  
标准要求靠近VBUS引脚  
的是D-，靠近GND的就是  
D+，注意不要画反。

针脚	名称
1	VBUS
2	D-
3	D+
4	GND

## ● USB 通讯介质及信号：

带屏蔽的双绞线缆，

差分阻抗90Ω，共模阻抗30Ω

电压驱动型，电流驱动型

USB在低速通信时，电缆不需要屏蔽层，也不需要双绞线。

USB在全速和高速通信时，需要外带屏蔽层的双绞线电缆。

低速和全速通信时，内置的驱动芯片是电压型输出差分信号。

而高速通信时，驱动芯片输出信号是电流型，在线路阻抗上产生400mV通信信号。

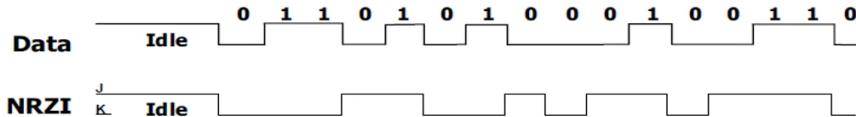
## ● USB 物理接口



## USB 编解码方式



USB编/解码方式 —— 反向不归零 (NRZI) ; 位填充



规则：数据为0，电平翻转，数据为1，电平不翻转



位填充：在数据进行NRZI编码前，每6个连续的1信号之后都会插入1个0信号，以免电平不能突变丢失同步。

## 信号传输状态

J状态 (差分)

K状态 (差分)

SE0状态 (单端)

SE1状态 (单端)

USB挂起复位这些状态都是基于J, K状态的组合和时间差异决定的

J状态

Low Speed: 差分0 Full Speed: 差分1

K状态

Low Speed: 差分1 Full Speed: 差分0

SE0状态

D+ 和 D- 都为 0V

SE1状态

D+ 和 D- 都大于 0.8V

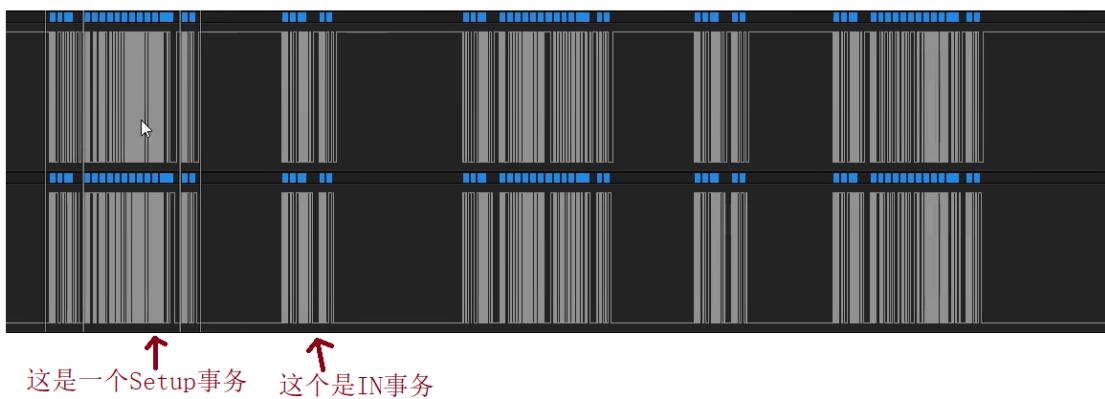
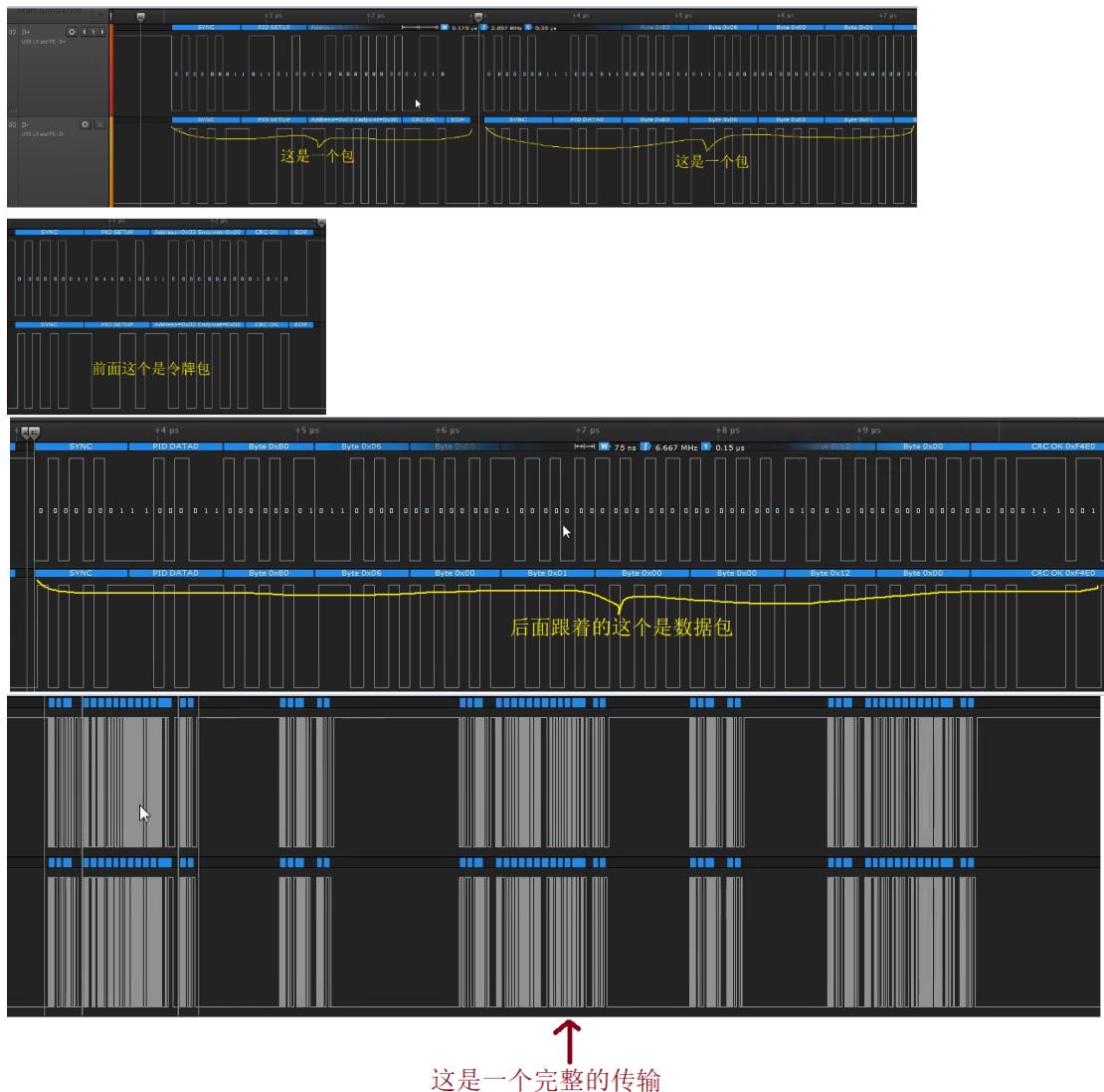
差分0: D+≈0V D-≈3V

差分1: D+≈3V D-≈0V

- 帧的概念：一个时间单位，固定为1ms（低/全速），高速-微帧 125us。  
在低速全速中1ms为1帧数据  
高速是125us一帧数据，叫做微帧。

- 通讯过程划分：多个域构成包的结构，多个包构成事务结构，多个事物构成传输。  
在USB传输协议中，会将事物作为最基本的单位。所以用户在USB传输控制上主要关注事物和传输这个层面的分配。





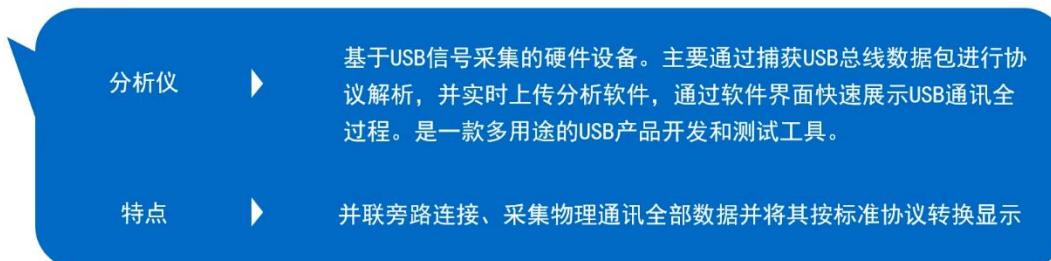
## USB 四种传输方式



## USB 传输概念术语

-  上传/下传     ● USB主机接收USB设备的数据称为上传，USB主机发送数据给USB设备称为下传
-  地址     ● 主机管理设备，而为每一个连接的设备分配，主机最多可以分配127个地址
-  端点     ● USB设备中实际的物理单元。端点和地址决定了主机和设备之间通讯的物理通道
-  USB传输特点     ● 物理传输双方角色一定是主机和设备；  
一问一答传输方式，永远主机先发起包请求；
-  常用开发工具

## USB 协议分析仪



## Bushound软件工具



逻辑分析仪及示波器

### Bushound

- 一款专用于PC端USB等总线捕捉和分析的软件
- 主要捕捉来自设备的协议包和输入输出操作  
主要是用在电脑端的USB分析软件，主要监控连接进电脑设备的输入和输出操作

### 特点

- 只采集PC驱动层数据、显示成功传输

逻辑分析仪：

以逻辑电平状态表示捕捉的全速及低速USB通讯物理信号，更清晰看到USB设备交互过程的详细内容，包括应答、时间、包结构等。对于初学者直观认识USB通讯有很大帮助。

示波器：

常见的信号捕捉仪器。对于USB应用偏向于帮助分析USB物理收发器的设计是否标准，信号质量、干扰等方面。

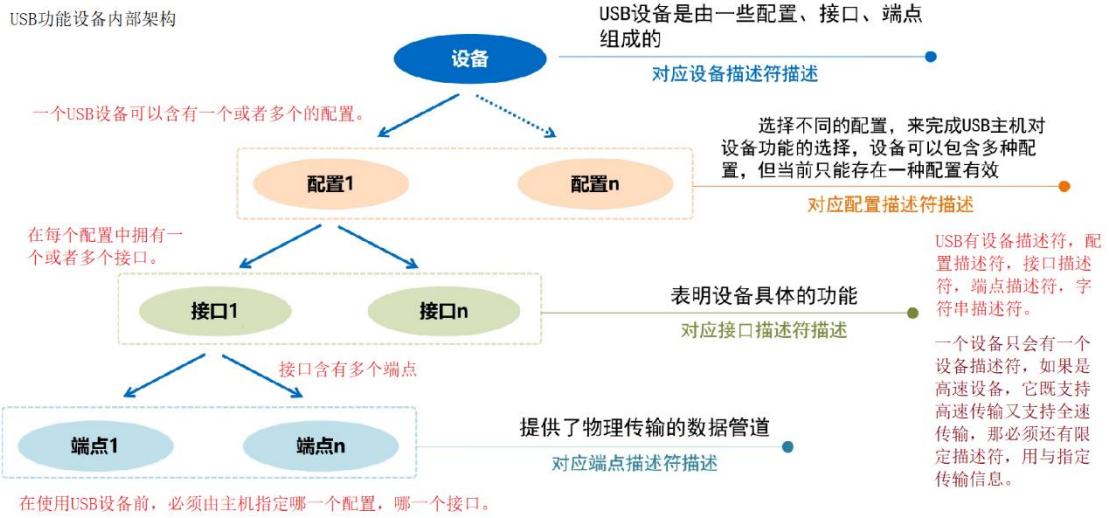
## USB 主从设备通信

集线器：

- 用于扩展USB主机的USB端口
- 结构上有一个上行端口，多个下行端口
- 支持级联，系统中最多5个集线器（不包括主机的根集线器）
- 支持速度转换

功能设备：

- 一个独立的外围设备，可以是单一功能也可以是多功能的合成设备
- 内部包含有描述自身功能和资源需求的配置信息  
U盘，USB打印机等都是功能设备

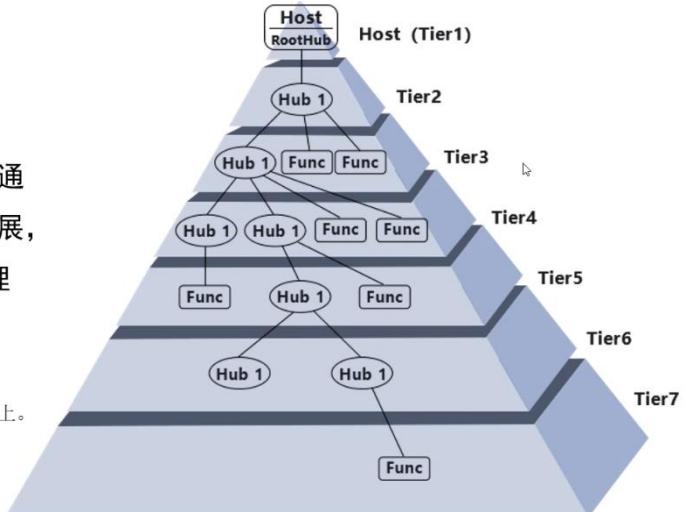


在使用USB设备前，必须由主机指定哪一个配置，哪一个接口。

**配置描述符：** 使用 USB 前必须选择一个合适的配置，在 USB 传输中只有一个配置生效。如果是高速设备就在配置中设置速率。

**接口描述符：** 是用来指定功能属性，比如一个 USB 声卡设备它可以由两个接口，一个是麦克风接口，一个为耳机输入接口。多个接口可以共存。

**端点：** 端点是实际的物理单元，USB 数据传输就是在 USB 主机和 USB 各个设备的端点中执行的。



在USB系统中，一般 USB主机通过集线器（HUB）进行 USB设备扩展，以层次性的星型拓扑结构进行物理连接。

Hub为集线器，Func为USB设备

一个USB功能设备，最多由5个集线器连接到主机上。

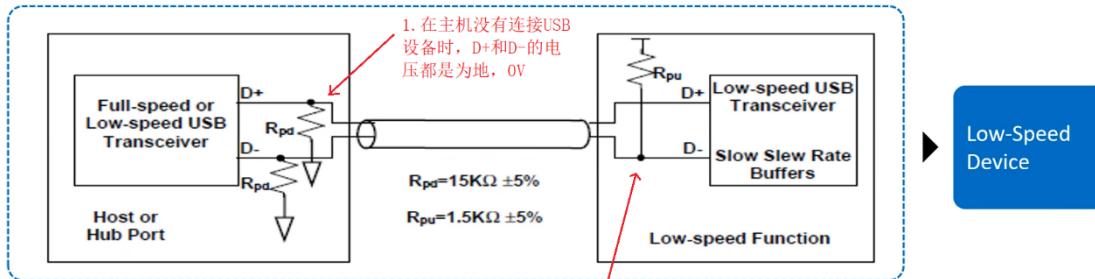
主机和USB设备之间通讯物理通道： 主设备分配的地址/默认地址0 + 从设备固有端点号。

主机和USB设备之间时间长度单位： 帧 / 微帧。

主机和USB设备之间协议处理基本单位： 事务处理。

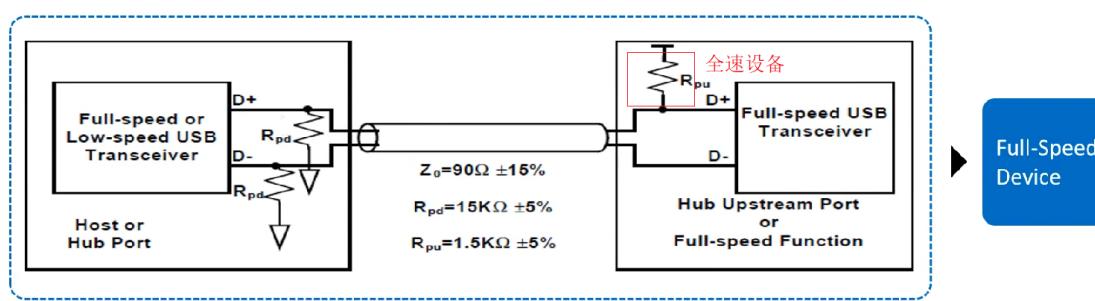
主机和USB设备之间通讯： 在基本单元“事务”中，主机总是发起者，并且和设备交互应答方式进行通讯。

## 连接与检测

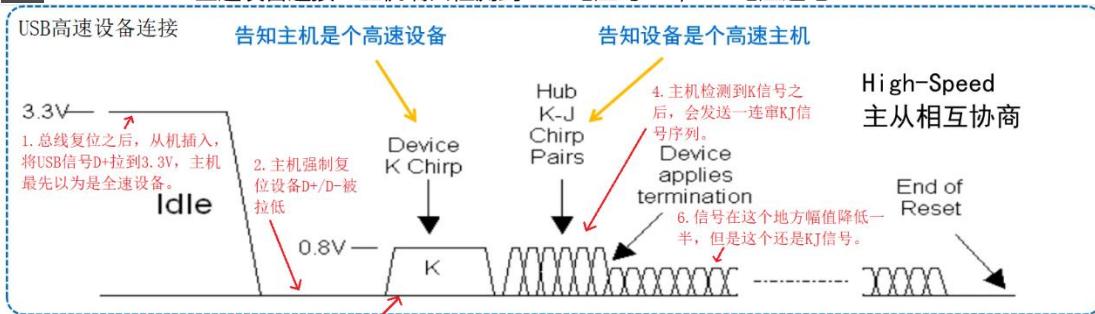


2. 当低速设备连接上来以后, D<sub>-</sub> 会出现一个上拉电阻, 此时设备端和主机的D<sub>-</sub>构成分压, 出现3V电压。主机检测这个设备为低速设备。

- 1 ● USB主机端口在 D<sub>+</sub> 和 D<sub>-</sub> 上都有15kΩ电阻接地
- 2 ● 低速USB从设备在 D<sub>-</sub> 上连有1.5kΩ的电阻到 3.0V-3.6V电压
- 3 ● 没有连接: 主机端口检测到 D<sub>+</sub> 和 D<sub>-</sub> 电压都近地 0V  
低速设备连接: 主机端口检测到 D<sub>-</sub> 电压约 3V, D<sub>+</sub> 电压近地 0V



- 1 ● USB主机端口在 D<sub>+</sub> 和 D<sub>-</sub> 上都有15kΩ电阻接地
- 2 ● 全速USB从设备在 D<sub>+</sub> 上连有1.5kΩ的电阻到 3.0V-3.6V电压
- 3 ● 没有连接: 主机端口检测到 D<sub>+</sub> 和 D<sub>-</sub> 电压都近地 0V  
全速设备连接: 主机端口检测到 D<sub>+</sub> 电压约 3V, D<sub>-</sub> 电压近地 0V



- 高速设备先以全速设备结构和主机连接, 它们之间做双向检测
- 主机输出总线复位信号期间, USB设备以是否可以产生 Chip K信号来表明高速或全速身份
- 在Chip K信号后, 主机是否发生KJ序列来表明高速主机身份或者是全速主机身份
- 匹配到高速主机和高速设备后, USB设备断开D<sub>+</sub>上的1.5kΩ的上拉电阻, 连接D<sub>+</sub>/D<sub>-</sub>上的高速终端电阻, 进入默认的高速状态, 否则以全速状态通讯

● 设备断开	主机检测到 D+ 和 D- 上近地状态 (0V) , 并持续2.5us以上
● 设备连接	主机检测到 D+ 或 D- 上有高电平 (3V) , 并持续2ms以上
● 低速设备	主机检测到 D- 上高电平
● 全速设备	主机检测到 D+ 上高电平 (有可能是高速设备)
● 高速设备	主机检测到 D+ 上高电平, 然后通过一系列协商握手信号确认双方身份 (双向检查)

### ● 总线几种状态

常见的几种状态	描述
正常工作	总线上存在周期性SOF包
总线复位	总线维持SE0状态 > 10ms
总线挂起	总线无活动 > 3ms

常见几种变化	触发点
无连接 -> 连接	D+/D- 上出现高电平>2ms
正常 -> 挂起	J状态保持 > 3ms
挂起 -> 正常 (唤醒)	出现 K状态信号并持续一段时间



**枚举：** USB主设备向USB从设备通过获取各种描述符，从而了解设备属性，知道是什么样的设备，并加载对应的USB类、功能驱动程序，然后进行后续一系列的数据通信。

- 特点：**
- 主设备连接识别从设备必须的过程
  - 由多个控制传输构成
  - 经过地址0（缺省地址）到其他地址（主设备分配地址）的通讯
  - 对于挂载多个USB从设备的系统，主设备是逐一进行枚举操作

## 枚举过程

- USB设备上电（一般从USB口取电）并连接到USB总线
- 主机检测到总线上有设备连接
- 主机会等待至少100ms用于连接的机械、电气特性稳定
- 主机执行总线复位至少10ms，并得到USB设备通讯速度
- 主机驱动总线空闲至少10ms用于做恢复时间
- 主机发出获取设备描述符请求（缺省地址）
- 主机为从设备分配唯一设备地址，后续通讯用此地址
- 主机以新地址发出获取设备描述符请求
- 主机以新地址发出获取配置描述符请求，获取设备全部配置
- 主机分析获取的描述符信息，并做相应记录和处理
- 主机发送设置配置请求，为从设备选择一个合适的配置

## 枚举过程

- 设备描述符：第一个需要获取的描述符，长度固定18字节。

```
ConfigDescr [ ] =
```

```
{
```

```
    配置描述符
```

```
    +
```

```
    接口描述符
```

```
    +
```

```
    类描述符
```

```
    +
```

```
    端点描述符
```

```
    . . .
```

```
}
```

- 配置描述符：描述了设备特定的配置，提供了当前配置下设备的功能接口，供电方式，耗电等信息。是一个配置的集合，集合长度不固定，包含了配置描述符、接口描述符、类定义描述符、端点描述。

多个接口时，继续添加  
接口+类+端点描述

## 控制传输解析

### 控制传输

1. 控制传输可以分为三种结构，每种结构都是由事务构成。

#### 2. 先是建立阶段

由SETUP事务构成

Setup Stage

SETUP (0)

DATA0

Control Write

SETUP (0)

DATA0

Control Read

SETUP (0)

DATA0

No-data Control

SETUP (0)

DATA0

#### 3. 数据阶段

由多个OUT事务构成

Data Stage

OUT (1)

DATA1

OUT (0)

DATA0

...

OUT (0/1)

DATA0/1

IN (1)

DATA1

#### 状态阶段

由IN事务构成

Status Stage

IN (1)

DATA1

1

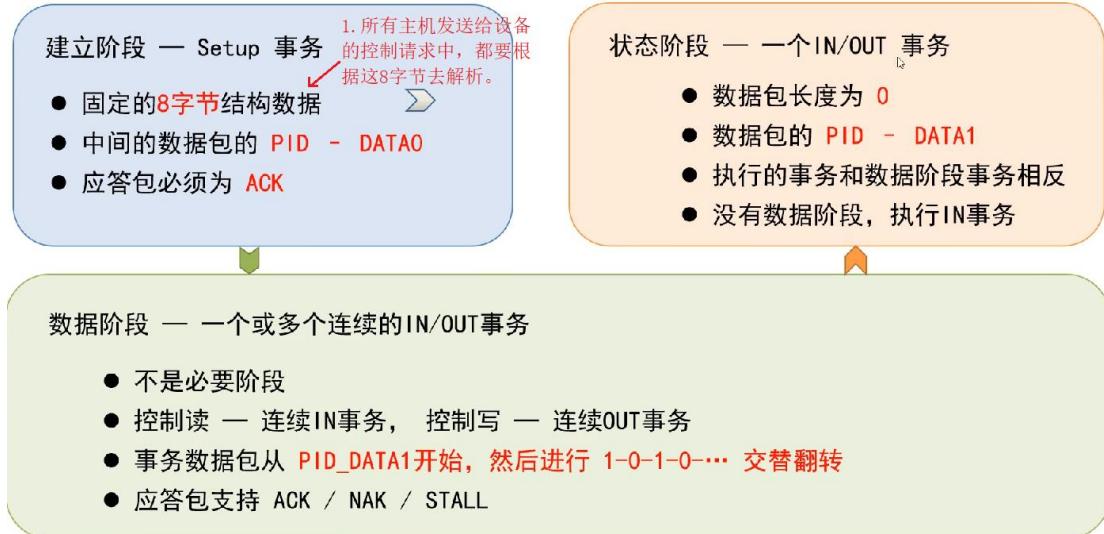
- 是所有USB从设备必须支持的传输方式，固定使用端点0通讯

2

- 控制传输的方向是双向的，既可以主机下传数据给设备，又可以从设备上传数据

3

- 多用于主设备和从设备进行信息、功能、状态等方面的获取和修改



控制传输中的 Setup Stage 部分（Setup事务），主机发出8字节命令请求，格式如下：



## USB 键盘报文基本解析

### PC电脑连接USB键盘数据解析

数据方向	令牌	地址	端点	数据	描述	事务包之间间隔
主机->从机	RESET				总线复位	主机发送复位总线令牌包
主机->从机	EOP (34)				Low Eop	34ms
主机->从机	SETUP	00	00			控制传输
主机->从机	DATA0			80 06 00 01 00 00 40 00	获取设备描述符	起始令牌包
主机<-从机	ACK					从机应答
主机->从机	IN (3)	00	00			主机发送IN3端点
主机<-从机	NAK					从机未应答
主机->从机	EOP				Low Eop	1ms
主机->从机	IN	00	00			主机再次发送EOP
主机<-从机	NAK					
主机->从机	IN	00	00			
主机<-从机	DATA1			12 01 10 01 00 00 00 08	从机返回设备描述符数据，固定8字节一个事务	
主机->从机	ACK					
主机->从机	IN (3)	00	00			
主机<-从机	NAK					
主机->从机	IN	00	00			

数据方向	令牌	地址	端点	数据	描述	事务包之间间隔
主机←从机	DATA0			3C 41 07 21 15 01 01 02	这也是设备描述符后半部分	
主机→从机	ACK				然后进入状态阶段	
主机→从机	IN(2)	00	00			
主机←从机	NAK					
主机→从机	IN	00	00			
主机←从机	DATA1			00 01		
主机→从机	ACK					
主机→从机	OUT	00	00			
主机→从机	DATA1					
主机←从机	ACK				结束	
设备描述符获取完成						
主机→从机	RESET				主机再次总线复位	
主机→从机	EOP					
主机→从机	SETUP	00	00		再次令牌包	
主机→从机	DATA0			00 05 0B 00 00 00 00 00	设置从机设备地址	
+ 16.0	SETUP	00	00	(6 byte) 00 05 0B 00 00 00 00	实际设置地址的报文格式	
→ 16.1	DATA0					
→ 16.2	ACK					
→ 16.0	IN(2)	00	00			
◆ 16.2	NAK					
→ 17.0	IN	00	00			
◆ 17.1	DATA1			(0 byte)		
→ 17.2	ACK					
+ 18.0	EOP(10)					
→ 19.0	SETUP	0B	00		Low_Exp 10 ns	
→ 19.1	DATA0			(6 byte) 80 06 00 01 00 00 12 00	Get_DevDesc:00	
→ 19.2	ACK					
→ 20.0	IN(2)	0B	00			
→ 20.2	NAK					
+ 21.0	EOP(10)					
→ 22.0	IN	0B	00			
→ 22.2	NAK					
→ 23.0	IN	0B	00			
◆ 23.1	DATA1			(8 byte) 12 01 10 01 00 00 00 08		
→ 23.2	ACK					
→ 24.0	IN(3)	0B	00			
→ 24.2	NAK					
→ 25.0	IN	0B	00			
◆ 25.1	DATA0			(6 byte) 3C 41 07 21 15 01 01 02		
→ 25.2	ACK					
→ 26.0	IN(2)	0B	00			
→ 26.2	NAK					
→ 27.0	IN	0B	00			
◆ 27.1	DATA1			(2 byte) 00 01		
→ 27.2	ACK					
→ 28.0	OUT	0B	00			
+ 29.0	EOP(2)					
→ 30.0	SETUP	0B	00		Low_Exp 12 ns	
→ 30.1	DATA0			(6 byte) 80 06 00 02 00 00 FF 00	Get_CfgDesc:00	
→ 30.2	ACK					
+ 31.0	EOP					
→ 32.0	IN(2)	0B	00			
→ 32.2	NAK					
→ 33.0	IN	0B	00			
◆ 33.1	DATA1			(8 byte) 09 02 22 00 01 01 00 A0		
→ 33.2	ACK					
→ 34.0	IN(2)	0B	00			
→ 34.2	NAK					
→ 35.0	IN	0B	00			
◆ 35.1	DATA0			(8 byte) 32 09 04 00 00 01 03 01		
→ 35.2	ACK					
→ 36.0	IN(2)	0B	00			
→ 36.2	NAK					
+ 37.0	EOP					
→ 37.1	DATA1			(8 byte) 01 00 09 21 10 01 00 01		
→ 37.2	ACK					
→ 38.0	IN(3)	0B	00			
→ 38.2	NAK					
+ 39.0	EOP					
→ 40.0	IN	0B	00			
→ 40.1	DATA0			(8 byte) 22 41 00 07 05 81 03 08		
→ 40.2	ACK					
→ 41.0	IN	0B	00			
→ 41.2	NAK					
→ 42.0	IN	0B	00			
◆ 42.1	DATA1			(2 byte) 00 0A		
→ 42.2	ACK					
→ 43.0	OUT	0B	00			
→ 43.1	DATA1			(0 byte)		
→ 43.2	ACK					
+ 44.0	SETUP	0B	00			
→ 44.1	DATA0			(8 byte) 80 06 00 03 00 00 FF 00	Get_StrDesc:00	
→ 44.2	ACK					
→ 45.0	IN(3)	0B	00			
→ 45.2	NAK					
→ 46.0	IN	0B	00			
→ 46.1	DATA1			(4 byte) 04 03 09 04		
→ 46.2	ACK					
→ 47.0	OUT	0B	00			
→ 47.1	DATA1			(0 byte)		
→ 47.2	ACK					
+ 48.0	EOP					
→ 49.0	SETUP	0B	00			
→ 49.1	DATA0			(8 byte) 80 06 02 03 09 04 FF 00	Get_StrDesc:02	
→ 49.2	ACK					
→ 50.0	IN(5)	0B	00			
→ 50.2	NAK					
→ 51.0	IN	0B	00			
◆ 51.1	DATA1			(8 byte) 30 03 44 00 65 00 6C 00		
→ 51.2	ACK					
→ 52.0	IN(2)	0B	00			
→ 52.2	NAK					
→ 53.0	IN	0B	00			
◆ 53.1	DATA0			(8 byte) 6C 00 20 00 55 00 53 00		
→ 53.2	ACK					
→ 54.0	IN(2)	0B	00			
→ 54.2	NAK					
+ 55.0	EOP					
→ 56.0	IN	0B	00	(8 byte) 42 00 20 00 45 00 6E 00		
→ 56.1	DATA1					
→ 56.2	ACK					
→ 57.0	IN(2)	0B	00			
→ 57.2	NAK					
→ 58.0	IN	0B	00			
→ 58.1	DATA0			(8 byte) 74 00 72 00 79 00 20 00		

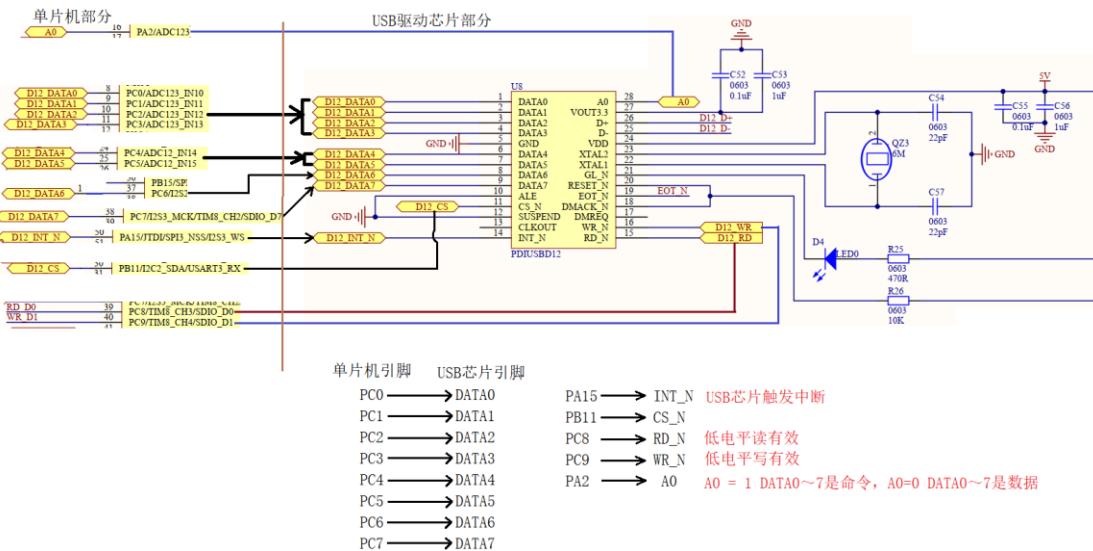
我们发现字符串描述符StrDesc发送了很多次，其实USB的设备描述符是可以反复重复的获取

## 标准USB键盘传输分析

80 06 00 01 00 00 12 00	获取设备描述符 80: 主机向设备请求数据 06: 表示获取描述符 00 01: 表示设备描述符 00 12: 长度 (表面上看是12 00 因为是小端)
00 05 03 00 00 00 00 00	设置设备地址 05: 表示设置地址 03: 表示设置的地址值是03 00 00 : 长度00
80 06 00 02 00 00 FF 00	获取配置描述符 02: 表示配置描述符 FF: 获取字符串长度 FF 00是大端(00FF)
80 06 00 03 00 00 FF 00	获取字符串描述符——语言描述
80 06 01 03 09 04 FF 00	获取字符串描述符——厂商描述
80 06 02 03 09 04 FF 00	获取字符串描述符——产品描述
80 06 03 03 09 04 FF 00	获取字符串描述符——产品序列号描述
00 09 01 00 00 00 00 00	设置设备配置
21 0A 00 00 00 00 00 00	有按键状态改变，就会返回这三行数据 <b>SET_IDLE</b> 21: 表示对接口这一类去操作
81 06 00 22 00 00 81 00	获取报表描述符
21 09 00 02 00 00 01 00	<b>SET_REPORT</b>

# STM32 驱动 PDIUSBD12 芯片，实现 USB 驱动

## STM32 与 PDIUSBD12 芯片硬件连接



## STM32 与 PDIUSBD12 芯片驱动代码

先定义芯片控制引脚的宏定义

```
#define CMD_READ_ID 0xFD //读取芯片ID命令

#define A0_PIN_H GPIO_SetBits(GPIOA,GPIO_Pin_2) //A0 = 1, PA2 = 1
#define A0_PIN_L GPIO_ResetBits(GPIOA,GPIO_Pin_2) //A0 = 0, PA2 = 0

#define WR_PIN_H GPIO_SetBits(GPIOC,GPIO_Pin_9) //WR = 1, PC9 = 1
#define WR_PIN_L GPIO_ResetBits(GPIOC,GPIO_Pin_9) //WR = 0, PC9 = 0

#define RD_PIN_H GPIO_SetBits(GPIOC,GPIO_Pin_8) //RD = 1, PC8 = 1
#define RD_PIN_L GPIO_ResetBits(GPIOC,GPIO_Pin_8) //RD = 0, PC8 = 0

#define CS_PIN_H GPIO_SetBits(GPIOB,GPIO_Pin_11) //CS = 1, PB11 = 1
#define CS_PIN_L GPIO_ResetBits(GPIOB,GPIO_Pin_11) //CS = 0, PB11 = 0

//因为GPIO组管脚是16位，为了不影响高8位引脚的数据，我只设置低8位引脚，故用下面宏实现
#define GPIO_WriteLow(GPIOx,a) GPIOx->BSRR=((uint32_t)(uint8_t)~(a))<<16)|((uint32_t)

未写完，GPIO 组管脚 16 位控制低 8 位下面实现
//因为 GPIO 组管脚是 16 位，为了不影响高 8 位引脚的数据，我只设置低 8 位引脚，故用下面宏实现
#define GPIO_WriteLow(GPIOx,a) GPIOx->BSRR=((uint32_t)(uint8_t)~(a))<<16)|((uint32_t)(uint8_t)(a))
//***************************************************************************** */
* PDIUSBD12 A0, RD, WR, CS, INT引脚初始化
//***************************************************************************** /
void config_PDIUSBD12(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //打开GPIOA时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //打开GPIOB时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //打开GPIOC时钟

    //A0-->PA2 输出模式
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //RD-->PC8,WR-->PC9 输出模式
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
```

```

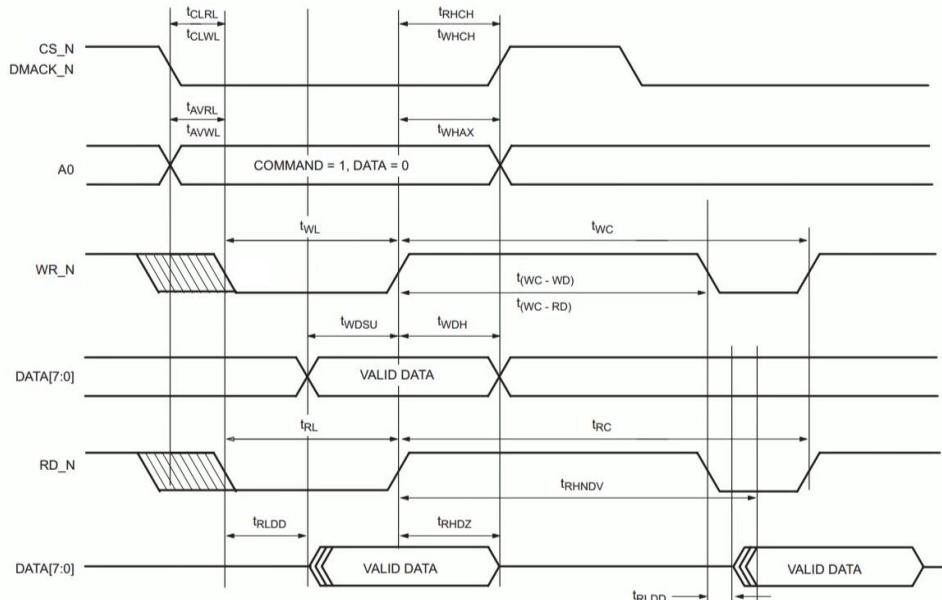
//CS-->PB11 输出模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);

//INT ->PA15
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

CS_PIN_L; //必须拉低电平选中芯片
}

```

写命令操作



```

/*****************
* 写命令
*****************/
void write_cmd(uint8_t cmd)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //打开GPIOC时钟

    //PC0~PC16 输出模式 我只用PC0~PC7做数据端口
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    A0_PIN_H; //A0=1
    WR_PIN_L; //WR=0
    // GPIO_Write(GPIOC, cmd); //单片机官方的GPIO组管脚写函数，会将16位管脚全部改变
    GPIO_WriteLow(GPIOC, cmd); //现在用自己修改的只写低8位管脚的函数
    WR_PIN_H; //WR=1
}

```

```

|*****  

* 写数据  

*****/  

void write_data(uint8_t data)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE); //打开GPIOC时钟

    //PC0~PC16 输出模式 我只用PC0~PC7做数据端口
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC,&GPIO_InitStructure);

    A0_PIN_L; //A0=0
    WR_PIN_L; //WR=0
//    GPIO_Write(GPIOC,data);
    GPIO_WriteLow(GPIOC,data);
    WR_PIN_H; //WR=1
}

|*****  

*读取1字节数据  

*****/  

uint8_t read_data(void)
{
    uint8_t tmp = 0;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE); //打开GPIOC时钟

    //PC0~PC16 输入模式 我只用PC0~PC7做数据端口
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC,&GPIO_InitStructure);

    A0_PIN_L; //A0=0
    RD_PIN_L; //RD=0
    tmp = GPIO_ReadInputData(GPIOC); //读出数据
    RD_PIN_H; //RD=1
    return (uint8_t)tmp;
}

|*****  

*读取芯片ID  

*****/  

uint16_t read_id(void)
{
    uint16_t tmp = 0;
    write_cmd(CMD_READ_ID);
    tmp = read_data(); //读低字节
    tmp |= read_data() << 8; //读高字节
    return tmp;
}

```

```

int main(void)
{
    uint16_t ChipID = 0;

    delay_init();           //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组2

    USART1_Config(115200); //printf串口初始化

    config_PDIUSBD12(); //USB芯片驱动管脚配置
    ChipID = read_id(); //读取ID
    printf("xxxxxxxx\r\n");

    while(1)
    {
        printf("Chip ID = 0x%x\r\n", ChipID);
        delay_ms(1000);

    }
    return 0;
}

```



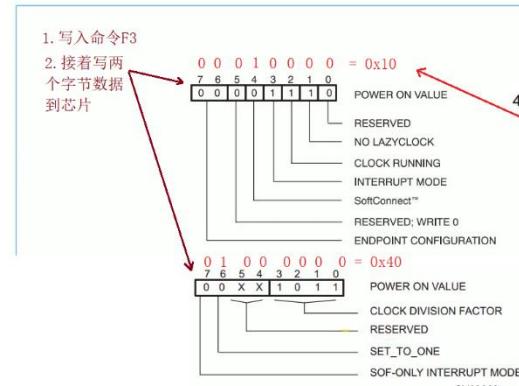
## USB 连接与断开

### 11.2.3 Set mode

**Code (Hex) — F3**

**Transaction — write 2 bytes**

The Set mode command is followed by two data writes. The first byte contains configuration bits. The second byte is the clock division factor byte.



Bit	Symbol	Description
7 to 6	ENDPOINT CONFIGURATION	These two bits set the endpoint configurations as mode 0 (Non-ISO mode) 模式选择同步传输 mode 1 (ISO-OUT mode) mode 2 (ISO-IN mode) mode 3 (ISO-I/O mode) See Section 8 "Endpoint description" for more details

Bit	Symbol	Description
7	SOF-ONLY INTERRUPT MODE	Setting this bit to 1 will cause the interrupt line to be due to the Start Of Frame clock (SOF) only, regardless of Pin-Interrupt mode, bit 5 of set DMA.
6	SET_TO_ONE	This bit needs to be set to 1 prior to any DMA read operation. This bit should always be set to 1 after power zero after Power-on reset.
3 to 0	CLOCK DIVISION FACTOR	The value indicates the clock division factor for CLK output frequency is $48 \text{ MHz} / (N+1)$ where N is the Clock Factor. The reset value is 11. This will produce the

```

#define CMD_READ_ID 0xFD //读取芯片ID命令
#define CMD_SET_MODE 0xF3
/*****************
* USB连接函数
********************/
void connect(void)
{
    write_cmd(CMD_SET_MODE); //进入写命令模式
    write_data(0x10); //连接设备
    write_data(0x40); //设置时钟和中断
    delay_ms(1000);
}

```

```

USART1_Config(115200); //printf串口初始化
config_PDIUSBD12(); //USB芯片驱动管脚配置
ChipID = read_id(); //读取ID
printf("xxxxxxxx\r\n");
disconnect(); //先断开USB
connect(); //再连接USB
while(1)

```

```

/*****************
* 断开连接
********************/
void disconnect(void)
{
    write_cmd(CMD_SET_MODE); //进入写命令模式
    write_data(0x00); //断开连接
    write_data(0x40); //设置时钟和中断
    delay_ms(1000);
}

```

通用串行总线控制器
AMD USB 3.10 可扩展主机控制器 - 1.10 (Microsoft)
AMD USB 3.10 可扩展主机控制器 - 1.10 (Microsoft)
J-Link driver
USB Composite Device
USB 捷集线器(USB 3.0)
USB 根集线器(USB 3.0)
USB 捷集线器(USB 3.0)
VIA USB 3.0 可扩展主机控制器 - 1.0 (Microsoft)
通用 USB 集线器
未知 USB 设备(设备描述符请求失败)

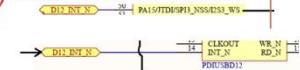
USB连接之后PC 出现未知描述符

## USB 端点中断测试

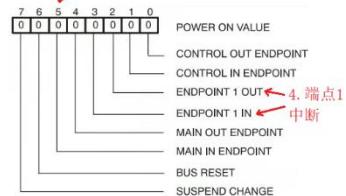
Table 4: Command summary

Name	Destination	Code (Hex)	Transaction
<b>Initialization commands</b>			
Set Address/Enable	Device	D0	Write 1 byte
Set Endpoint Enable	Device	D8	Write 1 byte
Set mode	Device	F3	Write 2 bytes
Set DMA	Device	FB	Write/Read 1 byte
<b>Data flow commands</b>			
Read Interrupt Register	Device	<b>F4</b>	Read 2 bytes <span style="float: right;">1. 该芯片有一个 读中断寄存器</span>
Select Endpoint	Control OUT	00	Read 1 byte (optional)
	Control IN	01	Read 1 byte (optional)
	Endpoint 1 OUT	02	Read 1 byte (optional)
	Endpoint 1 IN	03	Read 1 byte (optional)
	Endpoint 2 OUT	04	Read 1 byte (optional)
	Endpoint 2 IN	05	Read 1 byte (optional)
Read Last Transaction Status	Control OUT	40	Read 1 byte
	Control IN	41	Read 1 byte
	Endpoint 1 OUT	42	Read 1 byte
	Endpoint 1 IN	43	Read 1 byte
	Endpoint 2 OUT	44	Read 1 byte

2. 当单片机引脚收到中断信号的时候，  
就去读去中断寄存器里面的数据



3. 如果主输入中断发生，  
该位置1



# STM32F103 USB 实现鼠标功能

1. stm32f103的USB功能是移植官方的库例程

2. USB鼠标主要使用的  
是这个例程

3. 我们主要移植官方  
libraries目录下的USB  
驱动代码

4. 拷贝USB官方  
的驱动库到自己  
工程中

5. 将官方joy案  
例src目录下指  
定的c文件拷贝  
进自己工程新  
建的config目  
录中

6. c文件已经拷  
贝到自己工程  
目录下

7. 将joy工程inc中指  
定的h文件拷贝到自己  
工程的config目录下

8. 自己工程的config目录多了h文件

9. 将官方的ST USB库  
件和config目录下的文  
件加载进来

10. 将USB库头文件也进  
行指定

移植USB库完成之后，第一次编译报了12个错误。

下面修改platform\_config.h文件

```

30 #ifndef __PLATFORM_CONFIG_H
31 #define __PLATFORM_CONFIG_H
32
33 /* Includes */
34 #if defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(STM32L1XX_MD_PLUS)
35 #include "stm32l1xx.h"
36 #if defined(USE_STM32L152_EVAL)
37 #include "stm32l152.h"
38 #elif defined(USE_STM32L152D_EVAL)
39 #include "stm32l152d_eval.h"
40 #else
41 #error "Missing defines: USE_STM32L152_EVAL or USE_STM32L152D_EVAL"
42 #endif /* USE_STM32L152 EVAL */
43 #elif defined(STM32F10X_MD) || defined(STM32F10X_HD) || defined(STM32F10X_XL)
44 #include "stm32f10x.h"
45 #if defined(USE_STM32F105_EVAL)
46 #include "stm32f105.h"
47 #elif defined(USE_STM32F108_EVAL)
48 #include "stm32f108.h"
49 #elif defined(USE_STM32F107_EVAL)
50 #include "stm32f107.h"
51 #endif /* Missing define: USE_STM32F108_EVAL, USE_STM32F107_EVAL */
52 #endif /* USE_STM32F10X EVAL */
53 #elif defined(USE_STM32S151C_EVAL)
54 #include "stm32s151c.h"
55 #elif defined(USE_STM32S105C_EVAL)
56 #include "stm32s105c.h"
57 #elif defined(USE_STM32S107C_EVAL)
58 #include "stm32s107c.h"
59 #endif

```

直接屏蔽掉

```
/* Define to prevent recursive inclusion -----*/  
#ifndef __PLATFORM_CONFIG_H  
#define __PLATFORM_CONFIG_H  
#include "stm32f10x.h"  
#include "sys.h"          加入自定义的头文件和sys文件  
  
    #if !defined (USE_STM3210B_EVAL) &&  
        // #define USE_STM3210B_EVAL  
        // #define USE_STM3210E_EVAL  
        // #define USE_STM32L152_EVAL  
        // #define USE_STM32L152D_EVAL  
        // #define USE_STM32373C_EVAL  
        #define USE_STM32303C_EVAL  
    #endif #endif
```

## 修改hw\_config.c

```
28 L
29 /* Includes -----
30 #include "hw_config.h"
31 #include "usb_lib.h"
32 #include "usb_desc.h"
33 #include "usb_pwr.h"
34
35
36 /* Private typedef -----*/
37 /* Private define -----*/
38 /* Private macro -----*/
39 /* Private variables -----*/
40 ErrorStatus HSEStartUpStatus;
41 EXTI_InitTypeDef EXTI_InitStructure;
42
43 /* Extern variables -----*/
44 extern __IO uint8_t PrevXferComplete;
```

取消掉这段头文件和关键字

```
29 /* Includes -----*/  
30 #include "hw_config.h"  
31 #include "usb_lib.h"  
32 #include "usb_desc.h"  
33 #include "usb_pwr.h"  
34 #include "usb_istr.h"  
35  
36 //USB传输是否正在进行的标志（本例程没用到）  
37 //1,空闲; 0, 正在传输。  
38 volatile u8 PrevXferComplete = 1;//ST官方例程是在main.c定义的，我们改到这里定义  
39  
40 /* Private function prototypes -----*/  
41 static void IntToUnicode (uint32_t value , uint8_t *pbuff , uint8_t len);  
42 /* Private functions -----*/
```

hw\_config.c 包含的头文件变成这样子

删除Set System整个函数

```
删除Set_System函数
void Set_System(void)
{
#if !defined(STM32L1XX_MD) && !defined(STM32L1XX_HD)&& !defined(STM32L1XX_MD_PLUS)
    GPIO_InitTypeDef GPIO_InitStructure;
#endif /*STM32L1XX_XD */
#if defined(USB_USE_EXTERNAL_PULLUP)
    GPIO_InitTypeDef GPIO_InitStructure;
#endif /* USB_USE_EXTERNAL_PULLUP */

/*!< At this stage the microcontroller clock setting is already configured,
```

在Set System位置替换上自己定义的USB中断服务函数

```
/* Private functions */

//USB唤醒中断服务函数
void USBWakeUp_IRQHandler(void)
{
    EXTI_ClearITPendingBit(EXTI_Line18); //清除USB唤醒中断挂起位
}

//USB中断处理函数
void USB_LP_CAN1_RX0_IRQHandler(void)
{
    USB_Istr();
}
```

```

void Set_USBClock(void)
{
#if defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(STM32L1XX_MD_PLUS)
    /* Enable USB clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE);

```

取消掉Set\_USBClock函数原来的内容。

```

#else
    /* Select USBCLK source */
    RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5);

```

更换成USB时钟

```

/* Enable the USB clock */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE);
#endif /* STM32L1XX_XD */

}

//USB时钟配置函数,USBclk=48Mhz@HCLK=72Mhz
void Set_USBClock(void)
{
    RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5); //USBclk=PLLclk/1.5=48Mhz
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE); //USB时钟使能
}

删除整个GPIO_AINConfig函数
#ifndef STM32F37X
void GPIO_AINConfig(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

#if defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(
    /* Enable all GPIOs Clock*/
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ALLGPIO, ENABLE);
#else
    /* Enable all GPIOs Clock*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ALLGPIO, ENABLE);
    #endif // + comment above +
}

void Enter_LowPowerMode(void)
{
    /* Set the device state to suspend */
    bDeviceState = SUSPENDED;
    /* Clear EXTI Line18 pending bit */ 修改Enter_low...函数
    EXTI_ClearITPendingBit(KEY_BUTTON_EXTI_LINE);

    /* Request to enter STOP mode with regulator in low power mode */
    PWR_EnterSTOPMode(PWR_Regulator_LowPower, PWR_STOPEntry_WFI);
}

void Leave_LowPowerMode(void)
{
    DEVICE_INFO *pInfo = &Device_Info;
    /* Set the device state to the correct state */
    if (pInfo->Current_Configuration != 0) 修改Leave_lowPower...函数
    {
        /* Device configured */
        bDeviceState = CONFIGURED;
    }
    else
    {
        bDeviceState = ATTACHED;
    }
}

void Enter_LowPowerMode(void)
{
    printf("usb enter low power mode\r\n");
    bDeviceState=SUSPENDED;//bDeviceState记录USB连接状态，在usb_pwr.c里面定义
}

void Leave_LowPowerMode(void)
{
    DEVICE_INFO *pInfo=&Device_Info;
    printf("leave low power mode\r\n");
    if (pInfo->Current_Configuration!=0)bDeviceState=CONFIGURED;
    else bDeviceState = ATTACHED;
}

```

```

void USB_Interrupts_Config(void) 修改USB_Interrupt...函数内容，全部替换
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* 2 bit for pre-emption priority, 2 bits for subpriority */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

#if defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(STM32L1XX_MD_PLUS)
    /* Enable the USB interrupt */

```

void USB\_Interrupts\_Config(void) 替换成USB中断配置

```

    {
        NVIC_InitTypeDef NVIC_InitStructure;
        EXTI_InitTypeDef EXTI_InitStructure;
        /* Configure the EXTI line 18 connected internally to the USB IP */
        EXTI_ClearITPendingBit(EXTI_Line18); // 开启线18上的中断
        EXTI_InitStructure.EXTI_Line = EXTI_Line18; // USB resume from suspend mode
        EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //line 18上事件上升沿触发
        EXTI_InitStructure.EXTI_LineCmd = ENABLE;
        EXTI_Init(&EXTI_InitStructure);

        /* Enable the USB interrupt */
        NVIC_InitStructure.NVIC IRQChannel = USB_LP_CAN1_RX0_IRQn; //组2，优先级次之
        NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 1;
        NVIC_InitStructure.NVIC IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);

        /* Enable the USB Wake-up interrupt */
        NVIC_InitStructure.NVIC IRQChannel = USBWakeUp_IRQn; //组2，优先级最高
        NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0;
        NVIC_Init(&NVIC_InitStructure);
    }
}

void USB_Cable_Config (FunctionalState NewState) USB_Cable... 内容全部替换
{
#if defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(STM32L1XX_MD_PLUS)
    if (NewState != DISABLE)
    {
        STM32L15_USB_CONNECT;
    }
    else
    {
        STM32L15_USB_DISCONNECT;
    }

    #else
    if (NewState != DISABLE)
    {

```

void USB\_Cable\_Config (FunctionalState NewState)

```

        if (NewState!=DISABLE)printf("usb pull up enable\r\n");
        else printf("usb pull up disable\r\n");
    }
    USB_Cable替换后内容

```

```

        uint8_t JoyState(void) 删掉JoyState函数
    {
        /* "right" key is pressed */
        #if !defined(USE_STM32373C_EVAL) && !defined(USE_STM32303C_EVAL)
            if (!STM_EVAL_PBGetState(Button_RIGHT))
        #else
            if (STM_EVAL_PBGetState(Button_RIGHT))
        #endif
        }

        void Joystick_Send(uint8_t Keys)
    {
        uint8_t Mouse_Buffer[4] = { 0, 0, 0, 0 };
        int8_t X = 0, Y = 0;

        switch (Keys)
        {
            case JOY_LEFT:
                X -= CURSOR_STEP; ← Joystick_Send函数内
                break;           容需要修改
            case JOY_UP:
                Y -= CURSOR_STEP;
                break;
            case JOY_DOWN:
                Y += CURSOR_STEP;
                break;
            default:
                return;
        }

        void Set_System(void);
        void Set_USBClock(void);
        void GPIO_AINConfig(void);
        void Enter_LowPowerMode(void);
        void Leave_LowPowerMode(void);
        void USB_Interrupts_Config(void);
        void USB_Cable_Config (FunctionalState NewState);
        void Joystick_Send(uint8_t Keys);
        uint8_t JoyState(void);
        void Get_SerialNum(void);
        修改hw_config.h
    }

    void Set_System(void);
    void Set_USBClock(void);
    void GPIO_AINConfig(void);
    void Enter_LowPowerMode(void);
    void Leave_LowPowerMode(void);
    void USB_Interrupts_Config(void);
    void USB_Cable_Config (FunctionalState NewState);
    void Joystick_Send(uint8_t Keys);
    uint8_t JoyState(void);
    void Get_SerialNum(void);
    改成以下这样
    void Joystick_Send(u8 buf0,u8 buf1,u8 buf2,u8 buf3);
    //USB使能连接/断线 enable:0,断开;1,允许连接
    void USB_Port_Set(u8 enable)

```

注意分号忘记写

```

        }
    }
}
新增USB连线，断线函数
//USB使能连接/断线 enable:0,断开;1,允许连接
void USB_Port_Set(u8 enable)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE); //使能PORTA时钟
    if(enable)_SetCNTR(_GetCNTR()&(~(1<<1))); //退出断电模式
    else
    {
        _SetCNTR(_GetCNTR()|(1<<1)); // 断电模式
        GPIOA->CRH&=0XFFF00FFF;
        GPIOA->CRH|=0X00033000;
        PAout(12)=0;
    }
}

```

还要修改 `usb_pwr.c` 里面的 Suspend 函数

```
void Suspend(void)
{
    uint32_t i=0;
    uint16_t wCNTR;
    uint32_t tmpreg = 0;
    _IO uint32_t savePWR_CR=0;
/* suspend preparation */ 取消掉官方定义的内容
/* ... */

/* Store CNTR value */
wCNTR = _GetCNTR();

/* This a sequence to apply a force RESET to handle a robustness case */

/* Store endpoints registers status */
for (i=0;i<8;i++) EP[i] = _GetENDPOINT(i);

/* unmask RESET flag */
wCNTR |= CNTN_RESETM;
_SetCNTR(wCNTR);

用以下这段代码替代
void Suspend(void)
{
    uint32_t i=0;uint16_t wCNTR;_IO uint32_t savePWR_CR=0;
    wCNTR = _GetCNTR(); //Store CNTR value
    for (i=0;i<8;i++) EP[i] = _GetENDPOINT(i);
    wCNTR |= CNTN_RESETM; //unmask RESET flag
    _SetCNTR(wCNTR);
    wCNTR |= CNTN_FRES; //apply FRES
    _SetCNTR(wCNTR);
    wCNTR &= ~CNTN_FRES; //clear FRES
    _SetCNTR(wCNTR);
    while(!_GetISTR()&&_ISTR_RESET == 0); //poll for RESET flag in ISTR
    _SetISTR((uint16_t)CLR_RESET); //clear RESET flag in ISTR
    for (i=0;i<8;i++) _SetENDPOINT(i, EP[i]); //restore Enpoints
    wCNTR |= CNTN_FSUSP;
    _SetCNTR(wCNTR);
    wCNTR |= _GetCNTR(); //force low-power mode in the macrocell
    wCNTR |= CNTN_LPMODE;
    _SetCNTR(wCNTR);
    Enter_LowPowerMode();
}
```

## USB 鼠标数据格式讲解

### 数据格式

鼠标发送给PC的数据每次4个字节  
BYTE1 BYTE2 BYTE3 BYTE4  
定义分别是：  
BYTE1 –  
|~bit7: 表示 Y 坐标的增量超出 - 256 ~ 255 的范围, 0表示没有溢出  
|~bit6: 表示 X 坐标的增量超出 - 256 ~ 255 的范围, 0表示没有溢出  
|~bit5: Y 坐标变化的符号位, 1表示负数, 即鼠标向下移动  
|~bit4: X 坐标变化的符号位, 1表示负数, 即鼠标向左移动  
|~bit3: 恒为1  
|~bit2: 表示中键按下  
|~bit1: 表示右键按下  
|~bit0: 表示左键按下  
BYTE2 – Y坐标变化量, 与byte1的bit4组成9位符号数, 负数表示向左移, 正数表示右移, 用补码表示变化量  
BYTE3 – Y坐标变化量, 与byte1的bit5组成9位符号数, 负数表示向下移, 正数表示上移, 用补码表示变化量  
BYTE4 – 滚轮变化。  
BYTE1高位是可以不用关注的, 一般这5bit 在 HID 描述符中都是作为填充位使用, 置0即可。

USB鼠标HID描述符  
//每行开始的第一字节为该条目的前缀, 前缀的格式为:  
//07-04: bTag, 03-02: bType, 01-00: bSize. 以下分别对每个条目注释。  
0x05,0x01, // 是一个全局条目, 标示用途为通用桌面设备  
0x09,0x02, // 是一个局部条目, 标示用途为鼠标  
0xa1,0x01, // 表示包含集合, 必须要以END\_COLLECTION来结束它, 见最后的END\_COLLECTION  
0x09,0x01, // 是一个局部条目, 说明用途为指针集合  
0xa1,0x00, // 这是一个主要项, 并集合, 后面的数据0x00表示该集合是一个物理集合, 用由前面的局部条目定义为指针集合。  
0x09,0x01, // 这是一个全局条目, 说明用途为指针集合  
0x75,0x01, // 这是一个全局条目, 说明单个数据域的长度为1个bit。  
0x05,0x00, // 这是一个全局条目, 说明用途为报告(Button Page(0x09))  
0x19,0x01, // 这是一个全局条目, 说明用途的最小值为1, 实际上是鼠标左键。  
0x29,0x00, // 这是一个全局条目, 说明用途的最大值为3, 实际上是鼠标中键。  
0x15,0x00, // 这是一个全局条目, 说明用途回传数据的逻辑值(就是我们返回的数据域的值)最小为0。因为我们这里用bit来表示一个数据  
0x25,0x00, // 最大为1 MAXIMUM  
0x81,0x02, // 这是一个主要项, 标识上面的3个bits是独立的。  
0x95,0x01, // 这是一个全局条目, 说明用途的数据量为1个  
0x75,0x05, // 这是一个全局条目, 说明单个数据域的长度为5bit。  
0x81,0x00, // 这是一个主要项, 选择用途为普通桌面Generic Desktop Page(0x01)  
0x09,0x03, // 这是一个全局条目, 说明用途为轴  
0x09,0x03, // 这是一个全局条目, 说明用途为轴  
0x75,0x08, // 这是一个全局条目, 说明用途的数据量为8bit。  
0x05,0x01, // 这是一个全局条目, 选择用途为普通桌面Generic Desktop Page(0x01)  
0x09,0x00, // 这是一个全局条目, 说明用途为轴  
0x09,0x01, // 这是一个全局条目, 说明用途为轴  
0x15,0x01, // 这是一个全局条目, 说明用途回传数据量为 -128。  
0x25,0x7f, // 这是一个全局条目, 说明用途回传数据量为127。  
0x81,0x06, // 这是一个主要项, 标识上面的3个bits是绝对值。  
0xc0, // 我们讲了两个集合, 所以要关两次。bSize为0, 所以后面没数据。  
0xc0 // END\_COLLECTION

## STM32 配置本设备 USB 在电脑上的名字

在 `usb_desc.c` 文件中

```
const uint8_t Joystick_StringProduct[JOYSTICK_SIZ_STRING_PRODUCT] =
{
    JOYSTICK_SIZ_STRING_PRODUCT, /* bLength */
    USB_STRING_DESCRIPTOR_TYPE, /* bDescriptorType */
    'S', 0, 'T', 0, 'M', 0, '3', 0, '2', 0, ' ', 0, 'J', 0,
    'O', 0, 'Y', 0, 's', 0, 't', 0, 'i', 0, 'c', 0, 'k', 0
};

将官方的USB鼠标名字修改

const uint8_t Joystick_StringProduct[JOYSTICK_SIZ_STRING_PRODUCT] =
{
    JOYSTICK_SIZ_STRING_PRODUCT, /* bLength */
    USB_STRING_DESCRIPTOR_TYPE, /* bDescriptorType */
    'X', 0, 'X', 0, 'Z', 0, '3', 0, '2', 0, ' ', 0, 'J', 0, ←
    'O', 0, 'Y', 0, 's', 0, 't', 0, 'i', 0, 'c', 0, 'k', 0
};

当插上板子之后, USB鼠标会有一个我自定义的 XX... 名字, 但是在设备管理器里面看得到的哦, 是在其它地方看。
```

## STM32 USB 鼠标测试

```
#include "delay.h"
#include "sys.h"
#include "uart.h"
#include <stdio.h>

#include "usb_lib.h"
#include "hw_config.h"
#include "usb_pwr.h"

int main(void)
{
    delay_init(); // 延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组2

    USART1_Config(115200);

    USB_Port_Set(0); // USB先断开
    delay_ms(300);
    USB_Port_Set(1); // USB再次连接
    // USB配置
    USB_Interrupts_Config();
    Set_USBClock();
    USB_Init();
    printf("xxxxzzzz\r\n");
    while(1)
    {
        Joystick_Send(0,-100,-50,0); // 发送数据到电脑
        delay_ms(1000);
    }
    return 0;
}
```

没插入USB板子之前 →

插入USB板子之后 →

所以鼠标也是模拟的HID设备

HID设备就是人机交互设备，比如USB鼠标，USB键盘，USB游戏操作杆

板子USB插入主机之后，会发现鼠标，每一次都向左上角移动。所以-100，-50就是要求鼠标每次向x左边移动100个像素，向y方向上方移动50个像素。

## STM32F103 USB 实现虚拟串口(CDC)

USB数据传输通过一个专用的数据缓冲区来完成，它能被USB外设直接访问，其大小由所使用的端点数目和每个端点最大数据分组大小所决定，每个端点最大可使用512字节缓冲区（**专用的512字节，和CAN共用**），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，所以USB和CAN在F103上无法共用，只能用其中一个。



名称	修改日期	类型	大小
inc	2013/1/29 21:02	文件夹	
src	2013/1/29 21:02	文件夹	
Release_Notes.html	2013/1/22 2:27	Chrome HTML D...	39 KB

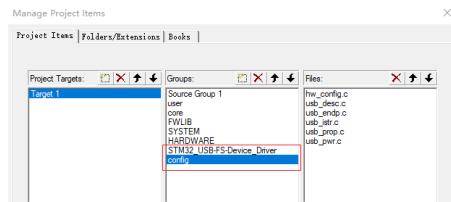
1. 在ST官方库中找到F103的USB从机驱动代码，一般在 Libraries目录下。这里面的代码直接复制到自己工程，一般不需要修改。

名称	修改日期	类型	大小
hw_config.h	2013/1/22 2:27	C Header 源文件	3 KB
platform_config.h	2013/1/22 2:27	C Header 源文件	7 KB
stm32_it.h	2013/1/22 2:27	C Header 源文件	3 KB
stm32f10x_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
stm32f30x_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
stm32f37x_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
stm32l1xx_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_desc.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_istr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_prop.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_pwr.h	2013/1/22 2:27	C Header 源文件	3 KB

3. 在自己工程新建config文件夹，拷贝虚拟串口相关的代码，这点就是和之前USB鼠标案例不一样的地方。

名称	修改日期	类型	大小
hw_config.c	2013/1/22 2:27	C 源文件	19 KB
main.c	2013/1/22 2:27	C 源文件	4 KB
stm32_it.c	2013/1/22 2:27	C 源文件	9 KB
system_stm32f10x.c	2013/1/22 2:27	C 源文件	29 KB
system_stm32f10x.h	2013/1/22 2:27	C 头文件	14 KB
system_stm32f37x.c	2013/1/22 2:27	C 源文件	14 KB
system_stm32l1xx.c	2013/1/22 2:27	C 源文件	20 KB
usb_desc.c	2013/1/22 2:27	C 源文件	8 KB
usb_endp.c	2013/1/22 2:27	C 源文件	5 KB
usb_istr.c	2013/1/22 2:27	C 源文件	7 KB
usb_prop.c	2013/1/22 2:27	C 源文件	14 KB
usb_pwr.c	2013/1/22 2:27	C 源文件	10 KB

4. 将官方与虚拟串口相关的c代码拷贝进自己工程。



6. 在MDK工程中按照USB驱动目录一一对应创建目录，加载对应的驱动c文件。

```
.\config\platform_config.h(50): error: #35: #error directive: "Missing define: USE_STM3210E"
#error "Missing define: USE_STM3210B_EVAL or USE_STM3210E_EVAL"
config\usb_pwr.c: 0 warnings, 1 error
".\OUT\STM32PDUUSB.axf" = 12 Error(s), 2 Warning(s).
```

8. 跟移植USB鼠标一样编译有12个错误

```
#ifndef __PLATFORM_CONFIG_H
#define __PLATFORM_CONFIG_H

/* Includes */
#ifndef defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(STM32L1XX_MD_PLUS)
#include "stm32l1xx.h"
#endif
#ifndef defined(USE_STM32L152_EVAL)
#include "stm32l152_eval.h"
#endif
#ifndef defined(USE_STM32L152D_EVAL)
#include "stm32l152d_eval.h"
#endif
#ifndef defined(USE_STM32L152_EVAL)
#error "Missing define: USE_STM32L152_EVAL or USE_STM32L152D_EVAL"
#endif
#ifndef defined(STM32F10X_MD) || defined(STM32F10X_HD) || defined(STM32F10X_XL)
#include "stm32f10x.h"
#endif
#ifndef defined(USE_STM3210B_EVAL)
#include "stm3210b_eval.h"
#endif
#ifndef defined(USE_STM3210E_EVAL)
#include "stm3210e_eval.h"
#endif
#ifndef defined(USE_STM3210B_EVAL)
#error "Missing define: USE_STM3210B_EVAL or USE_STM3210E_EVAL"
#endif
#ifndef defined(USE_STM32373C_EVAL)
#include "stm32f37x.h"
#endif
#ifndef defined(USE_STM32373C_EVAL)
#include "stm32373c_eval.h"
#endif
#ifndef defined(USE_STM323203C_EVAL)
#include "stm32f30x.h"
#endif
#ifndef defined(USE_STM323203C_EVAL)
#include "stm323203c_eval.h"
#endif
```

9. 和USB鼠标案例一样，取消掉platform里面的头文件部分内容

名称	修改日期	类型	大小
CMIS	2013/1/29 21:02	文件夹	
STM32_USB-FS-Device_Driver	2013/1/29 21:02	文件夹	
STM32F10x_StdPeriph_Driver	2013/1/29 21:02	文件夹	
STM32F30x_StdPeriph_Driver	2013/1/29 21:02	文件夹	
STM32F37x_StdPeriph_Driver	2013/1/29 21:02	文件夹	
STM32L1xx_StdPeriph_Driver	2013/1/29 21:02	文件夹	

2. 将整个从机驱动代码拷贝进我的工程，和USB鼠标移植方式一样。

名称	修改日期	类型	大小
hw_config.h	2013/1/22 2:27	C Header 源文件	3 KB
platform_config.h	2013/1/22 2:27	C Header 源文件	7 KB
usb_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_desc.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_istr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_prop.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_pwr.h	2013/1/22 2:27	C Header 源文件	3 KB

自己的工程目录

名称	修改日期	类型	大小
hw_config.h	2013/1/22 2:27	C Header 源文件	19 KB
hw_config.c	2013/1/22 2:27	C 源文件	3 KB
platform_config.h	2013/1/22 2:27	C Header 源文件	7 KB
usb_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_desc.c	2013/1/22 2:27	C 源文件	8 KB
usb_desc.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_endp.c	2013/1/22 2:27	C 源文件	5 KB
usb_endp.h	2013/1/22 2:27	C Header 源文件	7 KB
usb_istr.c	2013/1/22 2:27	C 源文件	3 KB
usb_istr.h	2013/1/22 2:27	C Header 源文件	14 KB
usb_prop.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_pwr.c	2013/1/22 2:27	C 源文件	10 KB
usb_pwr.h	2013/1/22 2:27	C Header 源文件	3 KB

5. 拷贝进自己工程



7. 加载USB驱动库头文件

```

#ifndef !defined (USE_STM3210B_EVAL) && !defined (USE_STM3210E_EVAL)
#define USE_STM3210B_EVAL
#define USE_STM3210E_EVAL
#define USE_STM32L152_EVAL
#define USE_STM32L152D_EVAL
#define (USE_STM32373C_EVAL)
#define (USE_STM32303C_EVAL) 10. 屏蔽掉
#endif

hw_config.c
1 /* Includes -----*/
2 /* Private define -----*/
3 #include "stm32_it.h"
4 #include "usb_lib.h"
5 #include "usb_desc.h"
6 #include "hw_config.h"
7 #include "usb_pwr.h"
8
9 /* Private typedef -----*/
10 /* Private macro -----*/
11 /* Private variables -----*/
12. 其实只需要取消掉stm32_it.h
13. 加入新头文件
14. 取消掉hw_config.c的这部分代码
15. 在hw_config.c中加入这段代码
16. 在hw_config.h中定义fifo结构体
17. 在platform.h中加入自己系统工程的头文件。
18. 定义错误是因为没有在platform.h中加入自己工程的头文件
19. 在hw_config.c中加入这段代码
20. 在hw_config.h中加入这段代码
21. 在hw_config.c中加入这段代码
22. 在hw_config.h中加入这段代码
23. 在hw_config.c中加入这段代码
24. 在hw_config.h中加入这段代码
25. 在hw_config.c中加入这段代码
26. 在hw_config.h中加入这段代码
27. 在hw_config.c中加入这段代码
28. 在hw_config.h中加入这段代码
29. 在hw_config.c中加入这段代码
30. 在hw_config.h中加入这段代码
31. 在hw_config.c中加入这段代码
32. #include "usb_lib.h"
33. #include "usb_prop.h"
34. #include "usb_desc.h"
35. #include "hw_config.h"
36. #include "usb_pwr.h"
37. #include "string.h"
38. #include "stdarg.h" ←13. 加入新头文件
39. #include "stdio.h"

/* Private typedef -----*/
/* Private define -----*/
/* Private macro -----*/
/* Private variables -----*/
ErrorStatus HSEStartUpStatus;
USART_InitTypeDef USART_ItfStructure;
EXTI_InitTypeDef EXTI_ItfStructure;
uint8_t USART_Rx_Buffer [USART_RX_DATA_SIZE];
uint32_t USART_Rx_ptr_in = 0;
uint32_t USART_Rx_ptr_out = 0;
uint32_t USART_Rx_length = 0;
uint32_t USART_Tx_State = 0;
static void IntToUnicode (uint32_t value , uint8_t *pbu
/* Extern variables -----*/

```

11. 取消掉以前包含的内容

12. 其实只需要取消掉stm32\_it.h

13. 加入新头文件

14. 取消掉hw\_config.c的这部分代码

15. 在hw\_config.c中加入这段代码

16. 在hw\_config.h中定义fifo结构体

17. 在platform.h中加入自己系统工程的头文件。

18. 定义错误是因为没有在platform.h中加入自己工程的头文件

19. 在hw\_config.c中加入这段代码

20. 在hw\_config.h中加入这段代码

21. 在hw\_config.c中加入这段代码

22. 在hw\_config.h中加入这段代码

23. 在hw\_config.c中加入这段代码

24. 在hw\_config.h中加入这段代码

25. 在hw\_config.c中加入这段代码

26. 在hw\_config.h中加入这段代码

27. 在hw\_config.c中加入这段代码

28. 在hw\_config.h中加入这段代码

29. 在hw\_config.c中加入这段代码

30. 在hw\_config.h中加入这段代码

31. 在hw\_config.c中加入这段代码

32. #include "usb\_lib.h"

33. #include "usb\_prop.h"

34. #include "usb\_desc.h"

35. #include "hw\_config.h"

36. #include "usb\_pwr.h"

37. #include "string.h"

38. #include "stdarg.h" ←13. 加入新头文件

39. #include "stdio.h"

```

#include "usb_lib.h"
#include "usb_prop.h"
#include "usb_desc.h"
#include "usb_istr.h" 加入istr.h
#include "hw_config.h"
#include "usb_pwr.h"
#include "string.h"
#include "stdarg.h"
#include "stdio.h"

//USB唤醒中断服务函数
void USBWakeUp_IRQHandler(void) ← 在hw_config.c中加入
{
    EXTI_ClearITPendingBit(EXTI_Line18); //清除USB唤醒中断挂起位
}

//USB中断处理函数
void USB_LP_CAN1_RX0_IRQHandler(void)
{
    USB_Istr();
}

//*****************************************************************************
/* Function Name : Set_System
 * Description  : Configures Main system clocks & power
 * Input        : None.
 * Return       : None.
*****
void Set_System(void) 19. 删除整个Set_System函数
{
#if !defined(STM32L1XX_MD) && !defined(STM32L1XX_HD) && !defined(STM32L1XX_MD_PLUS)
    GPIO_InitTypeDef GPIO_InitStructure;
#endif /* STM32L1XX MD & STM32L1XX HD */
void Set_USBClock(void) 20. 修改Set_USBClock里面的内容
{
#if defined(STM32L1XX_MD) || defined(STM32L1XX_HD) || defined(STM32L1XX_MD_PLUS)
    /* Enable USB clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE);
#else
    /* Select USBCLK source */
    RCC_USARTCLKConfig(RCC_USARTSource_PLLCLK_Div5);
#endif

void Set_USBClock(void) 21. 改成配置USB时钟，和USB鼠标例程修改内容一样
{
    RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5); //USBclk=PLLclk/1.5=48MHz
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USB, ENABLE); //USB时钟使能
}

void Enter_LowPowerMode(void) 22. 改成这样
{
    /* Set the device state to suspend */
    bDeviceState = SUSPENDED;
}

void Leave_LowPowerMode(void) 23. 修改以下内容
{
    DEVICE_INFO *pInfo = &Device_Info;

    /* Set the device state to the correct state */
    if (pInfo->Current_Configuration != 0)
    {
        /* Device configured */
        bDeviceState = CONFIGURED;
    }
    else
    {
        bDeviceState = ATTACHED.

    }

void Leave_LowPowerMode(void) 改成这样
{
    DEVICE_INFO *pInfo=&Device_Info;
    printf("leave low power mode\r\n");
    if (pInfo->Current_Configuration!=0)bDeviceState=CONFIGURED;
    else bDeviceState = ATTACHED;
}

```

```

void USB_Interrupts_Config(void) 24.USB中断配置函数, 改成以下这样
{
    NVIC_InitTypeDef NVIC_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    /* Configure the EXTI line 18 connected internally to the USB IP */
    EXTI_ClearITPendingBit(EXTI_Line18);      // 开启线18上的中断
    EXTI_InitStructure.EXTI_Line = EXTI_Line18; // USB resume from suspend mode
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //line 18上事件上升沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Enable the USB interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USB_LP_CAN1_RX0_IRQn;//组2, 优先级次之
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    /* Enable the USB Wake-up interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USBWakeUp_IRQn; //组2, 优先级最高
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_Init(&NVIC_InitStructure);
}

void USB_Interrupts_Config(void)//中断配置函数
{
    NVIC_InitTypeDef NVIC_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    /* Configure the EXTI line 18 connected internally to the USB IP */
    EXTI_ClearITPendingBit(EXTI_Line18);      // 开启线 18 上的中断
    EXTI_InitStructure.EXTI_Line = EXTI_Line18; // USB resume from suspend mode
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //line 18 上事件上升沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Enable the USB interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USB_LP_CAN1_RX0_IRQn;//组 2, 优先级次之
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    /* Enable the USB Wake-up interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USBWakeUp_IRQn; //组 2, 优先级最高
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_Init(&NVIC_InitStructure);
}

void USB_Cable_Config (FunctionalState NewState) //配置USB连接与断开 25.修改如下
{
    if (NewState!=DISABLE)printf("usb pull up enable\r\n");
    else printf("usb pull up disable\r\n");
}

void USART_Config_Default(void) 26.我们用不到这个函数, 整个删除
{
    /* EVAL_COM1 default configuration */
    /* EVAL_COM1 configured as follow:
       - BaudRate = 9600 bps */

//USB使能连接/断线   enable:0,断开;1,允许连接 27.新增一个USB连接断开函数
void USB_Port_Set(u8 enable)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE); //使能PORTA时钟
    if(enable)_SetCNTR(_GetCNTR() & (~ (1<<1))); //退出断电模式
    else
    {
        _SetCNTR(_GetCNTR() | (1<<1)); // 断电模式
        GPIOA->CRH |= 0xFFFF00FF;
        GPIOA->CRH |= 0x00033000;
        PAout(12)=0;
    }
}

```

```

bool USART_Config(void) 28. 修改该函数，用来打印USB虚拟串口配置信息，和清空缓冲区
{
    bool USART_Config(void)
    {
        uu_txfifo.readptr = 0; //清空读指针
        uu_txfifo.writeptr = 0; //清空写指针
        USB_USART_RX_STA = 0; //USB_USART接收状态清0
        printf("USB format:%d\r\n",linecoding.format);
        printf("USB paritytype:%d\r\n",linecoding.paritytype);
        printf("USB datatype:%d\r\n",linecoding.datatype);
        printf("USB bitrate:%d\r\n",linecoding.bitrate);
        return (TRUE);
    }

void Handle_USBAsynchXfer (void) 29. 删除整个函数
{
    uint16_t USB_Tx_ptr;
    uint16_t USB_Tx_length;
    if(USB_Tx_State != 1)
    {
        if (USART_Rx_ptr_out == USART_RX_DATA_SIZE)
        {
            USART_Rx_ptr_out = 0;
        }
    }
}

void USART_To_USB_Send_Data(void) 30. 删除整个函数
{

if (linecoding.datatype == 7)
{
    USART_Rx_Buffer[USART_Rx_ptr_in] = USART
    //发送一个字节数据到USB虚拟串口
}

void USB_USART_SendData(uint8_t data) 31. 新增发送数据函数
{
    uu_txfifo.buffer[uu_txfifo.writeptr] = data;
    uu_txfifo.writeptr++;
    if(uu_txfifo.writeptr == USB_USART_TXFIFO_SIZE) //超出buf大小了，清0
    {
        uu_txfifo.writeptr = 0;
    }
}
void IntToUnicode (uint32_t value , uint8_t *pbuff , uint8_t len)删除前面的static关键字
                                         然后前面声明一下
//确保一次发送的数据量不超过USB_USART_REC_LEN
void usb_printf(char *fmt,...) 实现USB虚拟串口打印
{
    uint16_t i,j;
    va_list ap;
    va_start(ap,fmt);
    vsprintf((char *)USART_PRINTF_Buffer,fmt,ap);
    va_end(ap);
    i = strlen((const char *)USART_PRINTF_Buffer);
    for(j = 0; j < i; j++)
    {
        USART_USART_SendData(USART_PRINTF_Buffer[j]);
    }
}

void USB_To_USART_Send_Data(uint8_t* data_buffer, uint8_t Nb_bytes)
    在USB接收串口数据的函数中加入自己的内容
{
    uint8_t i , res;
    for(i = 0; i < Nb_bytes; i++)
    {
        res = data_buffer[i];
        if((USB_USART_RX_STA & 0x8000) == 0) //接收未完成
        {
            if(USB_USART_RX_STA & 0x4000) //接收到 0x0d

```

```

void USB_To_USART_Send_Data(uint8_t* data_buffer, uint8_t Nb_bytes) //接收函数实现
{
    uint8_t i, res;
    for(i = 0; i < Nb_bytes; i++)
    {
        res = data_buffer[i];
        if((USB_USART_RX_STA & 0x8000) == 0) //接收未完成
        {
            if(USB_USART_RX_STA & 0x4000) //接收到 0x0D
            {
                if(res != 0x0A) USB_USART_RX_STA = 0; //错误，重新开始
                else USB_USART_RX_STA |= 0x8000; //接收完成了
            }
            else //还没收到 0x0D
            {
                if(res == 0x0D) USB_USART_RX_STA |= 0x4000;
                else
                {
                    USB_USART_RX_Buffer[USB_USART_RX_STA & 0x3FFF] = res; //USB串口接收数据存入
                    USB_USART_RX_STA++;
                    if(USB_USART_RX_STA > (USB_USART_REC_LEN - 1))
                        USB_USART_RX_STA = 0; //错误，重收
                }
            }
        }
    }
}

void IntToUnicode (uint32_t value , uint8_t *pbuff , uint8_t len);
//发送一个字节数据到USB虚拟串口
void USB_USART_SendData(uint8_t data);
//USB使能连接/断线 enable:0,断开;1,允许连接
void USB_Port_Set(u8 enable);
//确保一次发送的数据量不超过
void usb_printf(char *fmt,...);

在hw_config.h中加入这几个新增函数声明

```

```

.\OUT\STM32PDIUSB.axf: Error: L6210E: Undefined symbol Handle_USBAsynchXfer (referred from usb_endp.o).
.\OUT\STM32PDIUSB.axf: Error: L6210E: Undefined symbol USART_Rx_Buffer (referred from usb_endp.o).
.\OUT\STM32PDIUSB.axf: Error: L6210E: Undefined symbol USART_Rx_length (referred from usb_endp.o).
.\OUT\STM32PDIUSB.axf: Error: L6210E: Undefined symbol USART_Rx_ptr_out (referred from usb_endp.o).
.\OUT\STM32PDIUSB.axf: Error: L6210E: Undefined symbol USB_Tx_State (referred from usb_endp.o).
.\OUT\STM32PDIUSB.axf: Error: L6210E: Undefined symbol USART_Config_Default (referred from usb_prop.o).

```

编译过程中出现6个错误，这时候要修改usb\_endp.c里面的内容

```

void EP1_IN_Callback (void) usb_endp.c里面EP1_IN_Callback函数需要修改
{
    uint16_t USB_Tx_ptr;          这是端点1
    uint16_t USB_Tx_length;

    if (USB_Tx_State == 1)
    {

void EP1_IN_Callback (void)
{
    uint16_t USB_Tx_ptr;
    uint16_t USB_Tx_length;          这是修改后，端点1的内容
    if(uu_txfifo.readptr == uu_txfifo.writeptr)
        return;
    if(uu_txfifo.readptr < uu_txfifo.writeptr)
    {
        USB_Tx_length = uu_txfifo.writeptr - uu_txfifo.readptr;
    }
    else
        USB_Tx_length = USB_USART_TXFIFO_SIZE - uu_txfifo.readptr;
    if(USB_Tx_length > VIRTUAL_COM_PORT_DATA_SIZE) //超过64字节
    {
        USB_Tx_length = VIRTUAL_COM_PORT_DATA_SIZE;
    }
    USB_Tx_ptr = uu_txfifo.readptr;
    uu_txfifo.readptr += USB_USART_TXFIFO_SIZE;
    if(uu_txfifo.readptr >= USB_USART_TXFIFO_SIZE)
    {
        uu_txfifo.readptr = 0;
    }

UserToPMABufferCopy(&uu_txfifo.buffer[USB_Tx_ptr],ENDP1_TXADDR,USB_Tx_length); //拷贝数据
SetEPTxCount(ENDP1,USB_Tx_length); //设置端点1发送数据长度
SetEPTxValid(ENDP1); //设置端点1发送有效
}

```

```

void EP1_IN_Callback(void) //端点1 源码
{
    uint16_t USB_Tx_ptr;
    uint16_t USB_Tx_length;

    if(uu_txfifo.readptr == uu_txfifo.writeptr)
        return;
    if(uu_txfifo.readptr < uu_txfifo.writeptr)
    {
        USB_Tx_length = uu_txfifo.writeptr - uu_txfifo.readptr;
    }
    else
        USB_Tx_length = USB_USART_TXFIFO_SIZE - uu_txfifo.readptr;

    if(USB_Tx_length > VIRTUAL_COM_PORT_DATA_SIZE) //超过 64 字节
    {
        USB_Tx_length = VIRTUAL_COM_PORT_DATA_SIZE;
    }

    USB_Tx_ptr = uu_txfifo.readptr;
    uu_txfifo.readptr += USB_USART_TXFIFO_SIZE;
    if(uu_txfifo.readptr >= USB_USART_TXFIFO_SIZE)
    {
        uu_txfifo.readptr = 0;
    }

    UserToPMABufferCopy(&uu_txfifo.buffer[USB_Tx_ptr],ENDP1_TXADDR,USB_Tx_length); //拷贝数据
    SetEPTxCount(ENDP1,USB_Tx_length); //设置端点1发送数据长度
    SetPTxValid(ENDP1); //设置端点1发送有效
}

void SOF_Callback(void)
{
    static uint32_t FrameCount = 0;

    if(bDeviceState == CONFIGURED)
    {
        if (FrameCount++ == VCOMPORT_IN_FRAME_INTERVAL)
        {
            /* Reset the frame counter */
            FrameCount = 0;

            /* Check the data to be sent through IN pipe */
            Handle_USBAsynchXfer();
        }
    }
}

void Virtual_Com_Port_init(void)
{
    /* Update the serial number string descriptor with the data from the unique
    ID*/
    Get_SerialNum();

    pInformation->Current_Configuration = 0;

    /* Connect the device */
    PowerOn();

    /* Perform basic device initialization operations */
    USB_SIL_Init();

    /* configure the USART to the default settings */
    // USART_Config_Default(); // usb_prop.c里面去注释掉USART_Config...函数,
    bDeviceState = UNCONNECTED; // 因为我们没有用, 编译报错
}

```

修改usb\_pwr.c的Suspend电源管理函数

```

void Suspend(void) // USB鼠标案例里面有, 直接拷贝过来就算
{
    uint32_t i = 0; uint16_t wCNTR; __IO uint32_t savePWR_CR=0;
    wCNTR = _GetCNTR(); //Store CNTR value
    for (i=0;i<8;i++) EP[i] = _GetENDPOINT(i);
    wCNTR |= CNTR_RESETM; //unmask RESET flag
    _SetCNTR(wCNTR);
    wCNTR |= CNTR_FRES; //apply FRES
    _SetCNTR(wCNTR);
    wCNTR &= ~CNTR_FRES; //clear FRES
    _SetCNTR(wCNTR);
    while(!_GetISTR()&&_ISTR_RESET) == 0; //poll for RESET flag
    _SetISTR((uint16_t)CLR_RESET); //clear RESET flag in ISTR
    for (i=0;i<8;i++) _SetENDPOINT(i, EP[i]); //restore Epoin
    wCNTR |= CNTR_FSUSE;
    _SetCNTR(wCNTR);
    wCNTR = _GetCNTR(); //force low-power mode in the macrocell
    wCNTR |= CNTR_LPMODE;
    _SetCNTR(wCNTR);
    Enter_LowPowerMode();
}

```

```

void SOF_Callback(void) //该函数是帧首信号回调
{
    static uint32_t FrameCount = 0; //函数, 对应全速设备, 每1ms调用一次。
    if(bDeviceState == CONFIGURED)
    {
        if (FrameCount++ == VCOMPORT_IN_FRAME_INTERVAL)
        {
            /* Reset the frame counter */
            FrameCount = 0;
            EPI_IN_Callback(); // endp.c里面的SOF函数改成我的EP1包发送
        }
    }
}

```



## USB虚拟串口发送案例实验

```
#include "usb_lib.h"
#include "hw_config.h"
#include "usb_pwr.h"
int main(void)
{
    delay_init();           //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组2
    USART1_Config(115200);

    delay_ms(1800);
    USB_Port_Set(0); //USB先断开
    delay_ms(700);
    USB_Port_Set(1); //USB再次连接
    Set_USBClock();
    USB Interrupts_Config();
    USB_Init();

    while(1)
    {
        printf("xxxxxxxx\r\n"); //普通串口打印
        usb_printf("VirPort Test\r\n"); //USB打印
        delay_ms(1000);
    }
    return 0;
}
```

## USB虚拟串口发送案例实验

```
#include "usb_lib.h"
#include "hw_config.h"
#include "usb_pwr.h"
int main(void)
{
    delay_init();           //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组2
    USART1_Config(115200);

    delay_ms(1800);
    USB_Port_Set(0); //USB先断开
    delay_ms(700);
    USB_Port_Set(1); //USB再次连接
    Set_USBClock();
    USB Interrupts_Config();
    USB_Init();

    while(1)
    {
        printf("xxxxxxxx\r\n"); //普通串口打印
        usb_printf("VirPort Test\r\n"); //USB打印
        delay_ms(1000);
    }
    return 0;
}
```

虚拟串口无限制发送死机的问题如下：

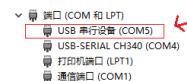
```
void EP1_IN_Callback (void)
{
    uint16_t USB_Tx_ptr;
    uint16_t USB_Tx_length;

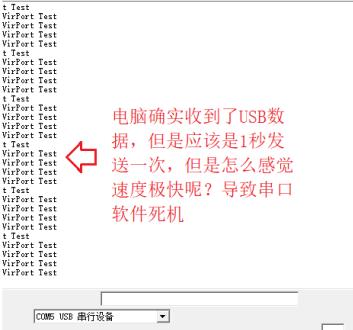
    if(uu_txfifo.readptr == uu_txfifo.writeptr)
        return;
    if(uu_txfifo.readptr < uu_txfifo.writeptr)
    {
        USB_Tx_length = uu_txfifo.writeptr - uu_txfifo.readptr;
    }
    else
        USB_Tx_length = USB_USART_TXFIFO_SIZE - uu_txfifo.readptr;

    if(USB_Tx_length > VIRTUAL_COM_PORT_DATA_SIZE) //超过64字节
    {
        USB_Tx_length = VIRTUAL_COM_PORT_DATA_SIZE;
    }

    USB_Tx_ptr = uu_txfifo.readptr;
    uu_txfifo.readptr += USB_USART_TXFIFO_SIZE; 这就解决问题的根源
    if(uu_txfifo.readptr >= USB_USART_TXFIFO_SIZE)
    {
        uu_txfifo.readptr = 0;
    }

    UserToPMABufferCopy(&uu_txfifo.buffer[USB_Tx_ptr],ENDP1_TXADDR,USB_Tx_length);//拷贝数据
    SetEPTxCount(ENDP1,USB_Tx_length); //设置端点1发送数据长度
    SetEPTxValid(ENDP1); //设置端点1发送有效
}
```

 开发板插上电脑，电脑收到了USB设备接入，只不过名字统一为USB串行设备。

 电脑确实收到了USB数据，但是应该是1秒发送一次，但是怎么感觉速度极快呢？导致串口软件死机。

STM32F103 USB 虚拟串口接收数据

stm32f103虚拟串口接收数据使用的就算usb库里面的USB\_SIL\_Read函数来接收

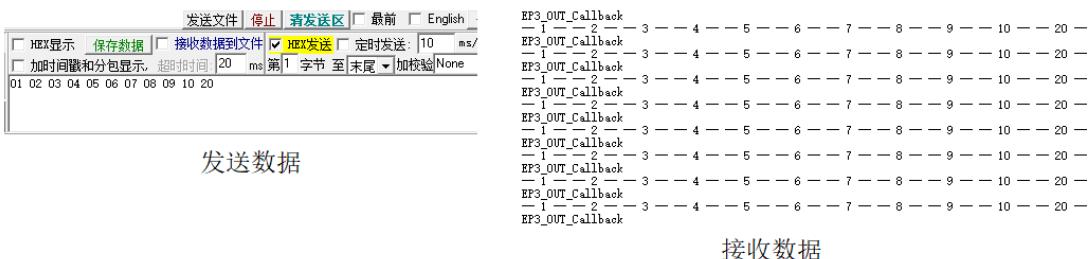
```
void EP3_OUT_Callback(void)
{
    u16 USB_Rx_Cnt;
    USB_Rx_Cnt = USB_SIL_Read(EP3_OUT, USB_Rx_Buffer); //得到USB接收到的数据及其长度
    USB_To_USART_Send_Data(USB_Rx_Buffer, USB_Rx_Cnt); //处理数据（其实就是保存数据）
    SetEP3RXValid(ENDP3); //时能端点3的数据接收
}

我使用自定义的函数来显示接收的数据

void USB_To_USART_Send_Data(u8* data_buffer, u8 Nb_bytes) 这个接收函数实现过于复杂和冗余了，其实只需要获取接收的数据即可
{
    u8 i;
    u8 res;
    for(i=0;i<Nb_bytes;i++)
    {
        res=data_buffer[i];
        if((USB_USART_RX_STA&0x8000)==0) //接收未完成
        {
            if(USB_USART_RX_STA&0x4000) //接收到了0x0d
            {
                if(res!=0xa)USB_USART_RX_STA|=0x8000; //接收错误，重新开始
                else USB_USART_RX_STA|=0x8000; //接收完成了
            }else //还没收到0x0d
            {
                if(res==0xd)USB_USART_RX_STA|=0x4000;
                else
                {
                    USB_USART_RX_BUF[USB_USART_RX_STA&0X3FFF]=res;
                    USB_USART_RX_STA++;
                    if(USB_USART_RX_STA>(USB_USART_REC_LEN-1))USB_USART_RX_STA=0; //接收数据错误，重新开始接收
                }
            }
        }
    }
}
```

```
void USB_To_USART_Send_Data(uint8_t* data_buffer, uint8_t Nb_bytes)
{
    u8 i;

    for(i=0;i<Nb_bytes;i++)
    {
        printf("-- %x -- ",data_buffer[i]); // 修改成这样接收数据
    }
    printf("\r\n");
}
```



从接收数据实验可以看出，USB虚拟串口接收数据是一帧一帧接收，而不是接收一个字节就执行EP3 OUT Callback回调函数。类似串口的DMA空闲中断。

STM32F103 USB 虚拟单字节发送数据

```
void EP1_IN_Callback (void) 对于PC端IN是输入，也就是端点1发数据到PC
{
    uint8_t XtxBuffer[64] = {0}; //这样发送数据会导致串口无限接收数据导致串口卡死，因为EP1_IN...与
    XtxBuffer[1] = 0x01;           SOF_Callback函数有关
    UserToPMABufferCopy(XtxBuffer,ENDP1_TXADDR,sizeof(XtxBuffer)); //拷贝数据
    SetEPFTxCount(ENDP1,sizeof(XtxBuffer)); //设置端点1发送数据长度
    SetEPFTxValid(ENDP1); //设置端点1发送有效
}

void SOF_Callback(void)
{
    static uint32_t FrameCount = 0;

    if(bDeviceState == CONFIGURED)
    {
        if (FrameCount++ == VCOMMPT_IN_FRAME_INTERVAL)
        {
            /* Reset the frame counter */
            FrameCount = 0;

            /* Check the data to be sent through IN pipe */
            EP1_IN_Callback(); //通过EP1_IN_Callback函数实现TX数据发送给USB
            //Handle_USBAsynchXfer();
        }
    }
}
```

1. 我们SOF函数在不停的调用EP1\_IN... 函数，为什么呢？

2. 因为USB\_Istr中断函数调用的内部的SOF\_Cal被不停的调用

### 3. 我们知道，USB都是主机寻址，然后从机

这就是设备USB无限发送，数据，串口助手收到数据的现象，太多了跟本收不完。卡死串口助手。

```

typedef struct
{
    uint8_t buffer[USB_USART_TXFIFO_SIZE]; // 1. 解决办法就是自己建立个缓冲区
    uint16_t writeptr; // 写指针
    uint16_t readptr; // 读指针
} _usb_usart_fifo;

void EP1_IN_Callback (void) extern _usb_usart_fifo uu_txfifo; //usb串口发送fifo

{
    u16 USB_Tx_ptr;
    u16 USB_Tx_length;
    if(uu_txfifo.readptr==uu_txfifo.writeptr) //无任何数据要发送,直接退出
    {
        return;
    }
    if(uu_txfifo.readptr<uu_txfifo.writeptr) //没有超过数组,读指针<写指针
    {
        USB_Tx_length=uu_txfifo.writeptr-uu_txfifo.readptr; //得到要发送的数据长度
    }else //超过数组了 读指针>写指针
    {
        USB_Tx_length=USB_USART_TXFIFO_SIZE-uu_txfifo.readptr; //得到要发送的数据长度
    }
    if(USB_Tx_length>VIRTUAL_COM_PORT_DATA_SIZE) //超过64字节?
    {
        USB_Tx_length=VIRTUAL_COM_PORT_DATA_SIZE; //此次发送数据量
    }
    USB_Tx_ptr=uu_txfifo.readptr; //发送起始地址
    uu_txfifo.readptr+=USB_Tx_length; //读指针偏移
    if(uu_txfifo.readptr>=USB_USART_TXFIFO_SIZE) //读指针归零
    {
        uu_txfifo.readptr=0;
    }
    UserToPMABufferCopy(&uu_txfifo.buffer[USB_Tx_ptr], ENDPI_TXADDR, USB_Tx_length); // 5. 发送数据
    SetEPTxCOUNT(ENDPI, USB_Tx_length);
    SetEPTXValid(ENDPI);
}

delay_ms (1800);
USB_Port_Set(0); //USB先断开
delay_ms (700);
USB_Port_Set(1); //USB再次连接
Set_USBClock();
USB_Interrupts_Config();
USB_Init();

printf("xxxxxxxxxx\r\n"); //普通串口打印
delay_ms (5000);

// usb_printf("VirPort Test\r\n"); //USB打印

while(1)
{
    printf("111111\r\n"); //普通串口打印
    // usb_printf("VirPort Test\r\n"); //USB打印

    USB_USART_SendData(0xf1); //发送单字节数据
    delay_ms (1000);
}
return 0;
}

```

## STM32F103 USB 实现 UVC 摄像头(各种描述符讲解)

当一个USB设备插入主机后，会有以下活动：

供电	USB上电
复位	USB设备复位
获取Device Descriptor	PC主机向地址00发送命令，要求USB设备提供设备描述符。
复位(可选)	主机再次复位USB设备/也可以不再次复位
分配地址	主机分配一个新的地址给当前插入得USB设备。
获取Device Descriptor	主机向新的地址发送获取USB设备的设备描述符。
获取Configuration Descriptor	主机获取USB设备配置描述符
获取String Descriptor(可选)	主机获取字符串描述符
配置	

### 1 、USB描述符

通过一套描述符，USB设备向USB主机描述自己的功能、属性、配置等信息

标准描述符：

- 设备描述符（Device Descriptor）
- 配置描述符（Configuration Descriptor）
- 接口描述符（Interface Descriptor）
- 端点描述符（Endpoint Descriptor） 用于数据传输的端点
- 字符串描述符（String Descriptor） 用于向PC提示该USB设备是什么厂家的

## (1)、设备描述符 (Device Descriptor)

描述设备的类型、厂商的信息、USB的协议类型、端点的包数据的最大长度等  
每个USB设备只有一个Device Descriptor

Offset	Field	Size 字节	Value	FUSB 110 code	Description
0	bLength	1	Number	12	Total length is 18 bytes
1	bDescriptor Type	1	Constant	01	Device descriptor
2	bcdUSB	2	BCD	10,01	USB ver. : 1.1
4	bDeviceClass	1	class	D_Class	Class code
5	bDeviceSubClass	1	Subclass	D_SubClass	Sub Class code
6	bDeviceProtocol	1	Protocol	D_Protocol	Protocol Code
7	bMaxPacketSize	1	Number	08	MaxPacketSize is 8 bytes (ENDPO)
8	idVendor	2	ID	V_IDL V_IDH	Vendor ID
10	idProduct	2	ID	P_IDL P_IDH	Product ID
12	bcdDevice	2	BCD	01,00	Device release number
14	iManufacturer	1	Index	10	IA 03 “FARADAY INC.”
15	iProduct	1	Index	20	10 03 “FUSB110”
16	iSerialNumber	1	Index	00	04 03 09 04
17	bNumConfiguration	1	Number	01	Number of possible configuration

设备描述符信息

在C语言中，USB描述符都是用数组表示

### Joystick\_DeviceDescriptor[ ] =

Offset	Field	Size 字节	Value	
0	bLength	1	Number	0x12, /* 整个Descriptor的长度: 18字节 */
1	bDescriptor Type	1	Constant	0x01, /* Descriptor的类别: Device Descriptor(0x01) */
2	bcdUSB	2	BCD	0x00, 0x02, /* 设备所遵循的USB协议的版本号: 2.00 */
4	bDeviceClass	1	class	0x00, /* 设备所实现的类: 由每个接口描述符描述所实现的类 */
5	bDeviceSubClass	1	Subclass	0x00, /* 设备所实现的子类: 由每个接口描述符描述 */
6	bDeviceProtocol	1	Protocol	0x00, /* 设备所遵循的协议类别: 由每个接口描述符描述 */
7	bMaxPacketSize	1	Number	0x40, /* 端点0的最大数据包长度: 64字节 */
8	idVendor	2	ID	0x83, 0x04, /* IDVendor: 0x0483 (for ST) */
10	idProduct	2	ID	0x10, 0x57, /* IDProduct: 0x5710 */
12	bcdDevice	2	BCD	0x00, 0x02, /* bcdDevice: 2.00 */
14	iManufacturer	1	Index	1, /* 用于描述生产厂商的字符描述符的索引号 */
15	iProduct	1	Index	2, /* 用于描述产品的字符描述符的索引号 */
16	iSerialNumber	1	Index	3, /* 用于描述产品系列号的字符描述符的索引号 */
17	bNumConfiguration	1	Number	0x01 /* 设备所支持的配置数目: 1 */

设备配置描述符，接口描述符，类描述符，端点描述符都可以写在一个数组里面

### Joystick\_ConfigDescriptor[ ] =

```
{
    配置描述符: Configuration Descriptor
    +
    接口描述符: Interface Descriptor
    +
    类描述符: Class Descriptor
    +
    端点描述符: Endpoint Descriptor
}
```

配置描述符

Offset	Field	Size	Value	FUSB 110 code	Description
0	bLength	1	Number	09	Total length is 9 bytes
1	bDescriptor Type	1	Constant	02	Configuration descriptor
2	wTotalLength	2	Number	27,00	27H=39D, 9(Cong.)+9(Inter.)+7(End.)*3=39
4	bNumInterfaces	1	Number	01	Number of Interface
5	bConfigurationValue	1	Number	01	Configuration Value
6	iConfiguration	1	Index	30	1C 03 “CONFIG STRING”
7	bmAttributes	1	Bitmap	[bmAttribute]	D7 : bus-powered (set to one) D6 : self-powered D5 : Remote Wakeup
8	Maxpower	2	mA	[Maxpower]	Max. Power consumption of the USB device from the bus. Expressed in 2mA units (i.e. 50=100mA)

配置描述符数组字节长度

描述符类型，一般是02，

表示配置描述符

后面要写的接口描述符，端点描述符和类描述符一共的长度。

支持的接口数目

配置号

索引号

供电配置，是支持自己供电，还是远程唤醒。

设备的最大功耗，也就是USB设备需要的最大电流。这样PC才好分配输出电流。

配置描述符

```
0x09,  
/* 描述符的长度: 9字节 */  
USB_CONFIGURATION_DESCRIPTOR_TYPE,  
/* 描述符的类型: 0x02 配置描述符(Configuration) */  
JOYSTICK_SIZ_CONFIG_DESC, 0x00,  
/* 完整的描述符包括接口描述符、端点描述符和类描述符的长度 */  
0x01,  
/* 配置所支持的接口数目: 1 */  
0x01,  
/* 用SetConofiguration()选择此配置, 所指定的配置号: 1 */  
0x00,  
/* 用于描述此配置的字符描述符的索引号: 0 */  
0xE0,  
/* 供电配置: B7(1 保留), B6(自供电), B5(远程唤醒),  
B4-B0(0 保留) */  
0x32,  
/* 最大功耗, 以2mA为单位计算: 0x32表示  $50 \times 2 = 100\text{mA}$  */
```

接口描述符

Offset	Field	Size	Value	FUSB 110 code	Description
0	bLength	1	Number	09	Total length is 9 bytes
1	bDescriptor Type	1	Constant	04	Interface descriptor
2	wInferfaceNumber	1	Number	00	Concurrent interface supported by this configuration
3	bAlternateSetting	1	Number	00	The value used to select alternate setting for the interface identified in the prior field
4	bNumEndpoints	1	Number	03	Number of endpoints
5	bInterfaceClass	1	Class	[I_Class]	Class code
6	bInterfaceSubClass	1	SubClass	[I_subClass]	Subclass code
7	bInterfaceProtocol	1	Protocol	[I_Protocol]	Protocol code
8	Interface	1	Index	40	22 03 "INTERFACE STRING"

接口描述符

我现在有一个设备，既有U盘功能，又有键盘功能。这时候就可以用两个接口描述符分别描述两个功能。

```
0x09,  
/* 描述符的长度: 9字节 */  
USB_INTERFACE_DESCRIPTOR_TYPE,  
/* 描述符的类型: 0x04接口描述符(Interface) */  
0x00,  
/* 选择此接口的索引号, 从0开始计算: 0 */  
0x00,  
/* 用于选择此设置的索引号: 0 */  
0x01,  
/* 实现此接口需要使用的端点数目: 1 */  
0x03,  
/* 此接口所遵循的类: HID Class */ 人机接口类  
0x01,  
/* 此接口所遵循的子类: 1=BOOT, 0=no boot: requiring BIOS support */  
0x02, /* 此接口所支持的协议: 0: 自定义、1: 键盘、2: 鼠标 */  
0, /* 用于描述此接口的字符描述符的索引号 */
```

接口描述符长度

描述符类型

此接口索引号

设置的索引号

端点数目

接口类型

接口子类型

接口协议

此接口描述符的字符  
描述符索引号

接口里面可以配置很多个端点，所以有下面的端点描述符

Offset	Field	Size	Value	End. 1	End. 2	Description	
0	bLength	1	Number	07	07	Total length is 7 bytes	长度
1	bDescriptor Type	1	Constant	05	05	Endpoint descriptor	类型
2	wEndpointAddress	1	Endpoint	81	02	81:input ,Endpoint1 02:output ,Endpoint2	端点地址
3	bmAttributes	1	Bitmap	02	02	00: control Transfer 01: Isochronous Transfer 02: Bulk Transfer 03: Interrupt Transfer	
4	wMaxPacketSize	2	Number	40,00	40,00	40H=64 bytes	
6	bInterval	1	Number	00	00	Interval for polling endpoint for data transfer . Bulk & Control: Ignored Isochronous : Set to 1 Interrupt : 1m~255ms	

### 端点描述符(Endpoint Descriptor):

```

0x07,
/* 描述符长度: 7字节 */
USB_ENDPOINT_DESCRIPTOR_TYPE,
/* 描述符类型: 端点描述符 */
0x81,
/* 端点的特性:
   B3-B0(端点号), B6-B4(0), B7(1=IN, 0=OUT): 0x81: Endpoint1/ IN */ 端点1是输入属性
0x03,
/* 端点的类型:
   B1-B0(00=控制 01=同步 10=大容量 11=中断): 0x03: 中断端点 */ 端点类型作为中断端点
0x04, 0x00
/* 此端点的最大有效数据长度: 4 字节 */
0x20,
/* 主机查询此端点数据的间隔时间: (1ms或125us单位): 0x20: 32 ms */

```

字符串描述符

```

_String_Descriptor:
.dw 0x12          //bLength
.dw 0x03          //bDescriptorType
.dw 'G', 0x00     //bString
.dw 'E', 0x00
.dw 'N', 0x00
.dw 'E', 0x00
.dw 'R', 0x00
.dw 'A', 0x00
.dw 'L', 0x00
.dw 'P', 0x00
.dw 'L', 0x00
.dw 'U', 0x00
.dw 'S', 0x00
.dw ' ', 0x00
.dw 'M', 0x00
.dw 'S', 0x00
.dw 'D', 0x00
.dw 'C', 0x00

```

字符串描述符就是USB设备插入PC之后，显示的名称。



# STM32F103 USB 实现 UVC 摄像头设备初步移植

我们以之前的虚拟串口工程为例，实现UVC摄像头的移植  
我这里只是将图片写入的数组里面，静态连续传输数组的图像数据到PC，用这种方式来模拟摄像头图片视频。



usb_conf.h	2015/4/29 9:29	C Header 源文件	4 KB	usb_desc.c	2024/10/31 17:33	C 源文件	11 KB
usb_desc.c	2024/10/31 17:33	C 源文件	11 KB	usb_desc.h	2015/3/25 14:10	C Header 源文件	4 KB
usb_desc.h	2015/3/25 14:10	C Header 源文件	4 KB				
usb_prop.c	2015/3/25 14:27	C 源文件	15 KB	usb_prop.c	2024/10/31 18:12	C 源文件	15 KB
usb_prop.h	2015/3/21 16:36	C Header 源文件	4 KB	usb_prop.h	2015/3/21 16:36	C Header 源文件	4 KB

主要就是移植官方做好的UVC驱动文件

官方做好的UVC文件有：

usb\_conf.h 该文件不一定要移植，看自己是否需要

usb\_desc.c/h 该文件必须移植，里面做好了官方的UVC描述符

usb\_prop.c/h

该文件必须移植，因为UVC系统初始化内容和虚拟串口有些不一样的地方。

这是之前虚拟串口的驱动文件，直接覆盖掉。

JPEGEncoder

ejpeg.c  
ejpeg.h  
srcjpg.c  
srcjpg.h

加入自己实现的图片文件，用于测试。

在之前虚拟串口hw\_config中加入如下代码：用于UVC数据传输

```
#include "srcjpg.h"

#define CAMERA_SIZ_STREAMHD 2
u8 sendbuf[PACKET_SIZE] = { 0x02, 0x01 }; // 发送数据缓冲区
u32 sendsize = 0; // 已发送字节数

void myMemcpy(const u8* src, u8* dst, u32 len)
{
    u32 i = 0;
    for (i = 0; i < len; ++i)
    {
        dst[i] = src[i];
    }
}

/* EP1 */
/* tx buffer base address */
#define ENDP1_BUFOAddr (0x90) // 使用端点1缓存地址发送图片数据
#define ENDP1_BUFIAddr (0x90+PACKET_SIZE)

void Get_SerialNum(void)
{
    uint32_t Device_Serial0, Device_Serial1, Device_Serial2;

    Device_Serial0 = *(uint32_t*)ID1;
    Device_Serial1 = *(uint32_t*)ID2;
    Device_Serial2 = *(uint32_t*)ID3;

    Device_Serial0 += Device_Serial2;

    if (Device_Serial0 != 0)
    {
        // IntToUnicode (Device_Serial0, &Virtual_Com_Port_StringSerial[2], 8); 屏蔽掉虚拟串口的时候
        // IntToUnicode (Device_Serial1, &Virtual_Com_Port_StringSerial[18], 4); 实现的内容
    }
}

#include "srcjpg.h"

#define CAMERA_SIZ_STREAMHD 2
u8 sendbuf[PACKET_SIZE] = { 0x02, 0x01 }; // 发送数据缓冲区
u32 sendsize = 0; // 已发送字节数

void myMemcpy(const u8* src, u8* dst, u32 len)
{
    u32 i = 0;
    for (i = 0; i < len; ++i)
    {
        dst[i] = src[i];
    }
}

/* EP1 */
/* tx buffer base address */
#define ENDP1_BUFOAddr (0x90)
#define ENDP1_BUFIAddr (0x90+PACKET_SIZE)

/*****************
* Function Name :UsbCamera_Fillbuf
* Description   :准备待发送的视频流缓冲区(该函数必须实现)在 hw_config.c 最后行实现
* Input         :
* Output        :
* Return        :
*****************/
void UsbCamera_Fillbuf(void)
{
    s32 datalen = 0; // 本次发送的字节数
    u8 *payload = 0; // 发送数据指针

    // 发送缓冲区有效数据地址
    payload = sendbuf + CAMERA_SIZ_STREAMHD;
    // 读数据到发送缓冲区
    if (0 == sendsize)
```

```

{
    sendbuf[1] &= 0x01;           // 清除 BFH
    sendbuf[1] ^= 0x01;          // 切换 FID

    // 计算本次发送数据长度
    datalen = PACKET_SIZE - CAMERA_SIZ_STREAMHD;
    // 读出发送数据
    myMemcpy(sbuf + sendsize, payload, datalen);

    sendsize += datalen;
    datalen += CAMERA_SIZ_STREAMHD;
} else{
    // 图像的后续包
    datalen = PACKET_SIZE - CAMERA_SIZ_STREAMHD;
    // 判断是否为最后一个数据包
    if (sendsize + datalen >= SBUF_SIZE)
    {
        datalen = SBUF_SIZE - sendsize;
        // 结束包标记(EOF 置位,帧结束位指示视频结束,仅在属于图像帧的最后一个 USB 传输操作中设置该位)
        sendbuf[1] |= 0x02;
    }
    // 读出发送数据
    myMemcpy(sbuf + sendsize, payload, datalen);

    sendsize += datalen;
    datalen += CAMERA_SIZ_STREAMHD;
}

// 复制数据到 PMA
if (_GetENDPOINTINT(ENDP1) & EP_DTOG_TX)
{
    // User use buffer0
    UserToPMABufferCopy(sendbuf, ENDP1_BUFOAddr, datalen);
    SetPPBbIBuf0Count(ENDP1, EP_DBUF_IN, datalen);
}
else{
    // User use buffer1
    UserToPMABufferCopy(sendbuf, ENDP1_BUFIAddr, datalen);
    SetPPBbIBuf1Count(ENDP1, EP_DBUF_IN, datalen);
}
_ToggleDTOG_TX(ENDP1);           // 反转 DTOG_TX
_SetEPTxStatus(ENDP1, EP_TX_VALID); // 允许数据发送

// 判断本帧图像是否发送完成
if (sendsize >= SBUF_SIZE)
{
    sendsize = 0;
}

return;
}

```

在usb\_endp.c中填充EP1\_IN\_Callback函数

```

void EP1_IN_Callback (void) 该函数会被PC不停的发送数据，造成单片机中断回调执行
{
    UsbCamera_Fillbuf(); 发送图片数据到PC
}

```

修改usb\_endp.c里面的SOF函数内容

```

void SOF_Callback(void)
{
    static uint32_t FrameCount = 0;

    if(bDeviceState == CONFIGURED)
    {
        if (FrameCount++ == VCOMPORT_IN_FRAME_INTERVAL)
        {
            /* Reset the frame counter */
            FrameCount = 0;

            /* Check the data to be sent through IN pipe */
//            EP1_IN_Callback(); //通过EP1_IN_Callback函数实现TX数据发送给USB
            //Handle_USBAsynchXfer(); 注意，既然是UVC自动调用了EP1_IN_.. 回调，那么SOF里面的
//            EP1_IN_... 回调必须取消掉，不然PC相机会出现卡顿，显示不
//            完全的现象。
        }
    }
}

```

移植完成，设备管理器  
会出现相机节点



得到我发送的画面

STM32F103 USB 使用 UVC 协议发送图片的函数实现原理

## UVC 描述符内容讲解

STM32USB 的usb\_desc.c文件下的设备描述符

```

/* USB Standard Device Descriptor */
const u8 Camera_DeviceDescriptor[CAMERA_SIZ_DEVICE_DESC] =
{
    . . .
    /* bLength */18字节(设备描述符字节数)
    USB_DEVICE_DESCRIPTOR_TYPE,           /* bDescriptorType */01类型
    0x00, 0x02,                         /* bcdUSB 2.10 */
    0xEF,                                /* bDeviceClass */
    0x02,                                /* bDeviceSubClass */
    0x01,                                /* bDeviceProtocol */
    0x40,                                /* bmMaxPacketSize 40 */
    0x92, 0x19,                          /* idVendor = 0x1985*/
    0x01, 0x01,                          /* idProduct = 0x1017 */
    0x00, 0x01,                          /* bcdDevice */
    1,                                    /* iManufacturer */
    2,                                    /* iProduct */
    3,                                    /* iSerialNumber */
    0x01,                                /* bNumConfigurations */
};



| Offset | Value | Device Phas |
|--------|-------|-------------|
| 19.0   | CTL   | -----       |
| 19.0   | IN    | -----       |
| 19.0   | CTL   | -----       |
| 19.0   | IN    | -----       |
| 19.0   | CTL   | -----       |



UVG摄像头插入P



再看第二行, "IN"


```

其中PC就是通过画红圈的那三个字节数据知道该设备为UVC设备的。→

## 协议解析

UVC摄像头插入PC后，PC向其端点0发送了8个Byte的数据：80 06 00 01 00 00 12 00

再看第二行，“IN”表示数据传输方向为输入，即PC接收UVC摄像头返回的数据。

Table 2-1 Device Descriptor				
Offset	Field	Size	Type	Description
0	bLength	1	0x12	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x01	DEVICE descriptor
2	bcdUSB	2	0x0200	2.00 – current revision of the USB specification
4	bDeviceClass	1	0xEF	Miscellaneous Device Class
5	bDeviceSubClass	1	0x02	Common Class
6	bDeviceProtocol	1	0x01	Interface Association Descriptor
7	bMaxPacketSize0	1	0x40	Control endpoint packet size is 64 bytes
8	idVendor	2	0xFFFF	Vendor ID
10	idProduct	2	0xFFFF	Product ID
12	bcdDevice	2	0xFFFF	Device release code
14	bManufacturer	1	0x01	Indirect string descriptor that contains the string <Your Name> in ASCII
15	bProduct	1	0x02	Indirect string descriptor that contains the string <Your Product Name> in ASCII

```

Device Phase Data
-----
19.0 CTL 80 06 00 01 00 00 12 00
19.0 IN 12 01 00 02 ef 02 01 40 93 69 14 b0 01 00 01 02 00 01
19.0 CTL 80 06 00 02 00 00 09 00
19.0 IN 09 02 8c 01 02 01 00 80 fa
19.0 CTL 80 06 00 02 00 00 8c 01 @51CTO博客

```

PC收到UVC摄像头返回的18Byte配置描述符后，紧接着又发出了第二条控制命令

80 06 00 02 00 00 09 00,

“请求UVC发送自己的配置描述符至PC，数据长度9Byte”。

随后UVC摄像头返回了9Byte数据至PC: 09 02 c0 00 02 01 00 80 fa  
我这里是c0，因为后面还有100多个字节

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x02	CONFIGURATION descriptor
2	wTotalLength	2	0x00C0	Length of the total configuration block, including this descriptor, in bytes
4	bNumInterfaces	1	0x02	This device has two interfaces
5	bConfigurationValue	1	0x01	ID of this configuration
6	iConfiguration	1	0x00	Unused
7	bm.Attributes	1	0x80	Bs-powered device, no remote wakeup capability
8	bMaxPower	1	0xF0	500 mA maximum power consumption

其中，我们暂时先关注下第3、4字节，即“wTotalLength”参数，由于USB传输是低字节先传输，故我们知道，该配置描述符以及其下所有的接口描述符和端点描述符的

紧接着，PC开始向UVC摄像头端点0发送第三次控制命令啦：80 06 00 02 00 00 8c 01，第7、8字节是不是很眼熟啊？没错，第4字节还是“02”，PC这次还是发送的读取JPG没有配置文件命令。但这次读取的字符串是“0x8c”字符，那这次就将“读JPG文件”和“读取控制文件”合二为一啦！数据量比较大，如下图所示。

取UVC设备配置抽还行 吱々，但是这个读取的字节长度是 0x10C 字节，那么就是得接 口抽还行 和 端点抽还行 全都读出来啦：数据量比较大，如下图所示。

```

/* 1. Standard Video Interface Collection IAD */
0x08,          /* bLength */
USB_ASSOCIATION_DESCRIPTOR_TYPE, /* bDescriptorType */
0x00,          /* bFirstInterface: Interface number of the VideoControl interface */
0x02,          /* Number of contiguous Video interfaces that are associated with */
0x0E,          /* bFunction Class: CC_VIDEO*/
0x03,          /* bFunction sub Class: SC_VIDEO_INTERFACE_COLLECTION */
0x00,          /* bFunction protocol : PC_PROTOCOL_UNDEFINED*/
0x02,          /* iFunction */

```

首先我们要先看IAD描述符IAD是“Interface Association Descriptor”的简写，译为“接口联合描述符”，一个配置描述符是接口描述符的集合，在UVC中，IAD描述符就是描述一个视频接口集合的，对于每一个设备功能需要一个视频控制接口(VideoControl Interface)和一个或者多个视频流接口(VideoStreaming Interface)。如下图所示。

Table 3-3 Standard Video Interface Collection IAD

Offset	Field	Size	Value	Description
0	bLength	1	0x08	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x0B	INTERFACE ASSOCIATION Descriptor.
2	bFirstInterface	1	0x00	Interface number of the VideoControl interface that is associated with this function.
3	bInterfaceCount	1	0x02	Number of contiguous Video interfaces that are associated with this function. ← 对比文档“bInterfaceCount”参数为0x02，可知该配置下有两个视频接口—“VC和VS”。
4	bFunctionClass	1	0x0E	CC_VIDEO
5	bFunctionSubClass	1	0x03	SC_VIDEO_INTERFACE_COLLECTION
6	bFunctionProtocol	1	0x00	Not used. Must be set to PC_PROTOCOL_UNDEFINED.
7	iFunction	1	0x04	Index of string descriptor. Must match the iInterface field of the Standard VC Interface Descriptor.

#### 接口描述符

```

/* 1.1 Video control Interface Descriptor */ 此描述符为VC端口描述符，可知VC的接口为0x0E, VS的接口为0x01
/* 1.1.1 Standard VideoControl Interface(VC) Descriptor */
0x09,          /* bLength */
0x04,          /* bDescriptorType */
0x00,          /* bInterfaceNumber */
0x00,          /* bAlternateSetting */
0x00,          /* bNumEndpoints:1 endpoint (interrupt endpoint) */
0x0e,          /* bInterfaceClass : CC_VIDEO */
0x01,          /* bInterfaceSubClass : SC_VIDEOCONTROL */
0x00,          /* bInterfaceProtocol : PC_PROTOCOL_UNDEFINED */
0x02,          /* iInterface:Index to string descriptor that contains

```

Table 2-4 Standard VC Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x04	INTERFACE descriptor type
2	bInterfaceNumber	1	0x00	Index of this interface
3	bAlternateSetting	1	0x00	Index of this setting
4	bNumEndpoints	1	0x01	1 endpoint (interrupt endpoint)
5	bInterfaceClass	1	0x0E	CC_VIDEO VC接口
6	bInterfaceSubClass	1	0x01	SC_VIDEOCONTROL VS接口
7	bInterfaceProtocol	1	0x01	PC_PROTOCOL_15
8	iInterface	1	0x02	Index to string descriptor that contains the string <Your Product Name> in Unicode. Have to match iFunction field of the Standard Video Interface Collection IAD

bLength : 描述符大小. 固定为0x09.

bDescriptorType : 接口描述符类型. 固定为0x04.

bInterfaceNumber: 该接口的编号, 接口编号从0开始.

bAlternateSetting : 用于为上一个字段选择可供替换的位置. 即备用的接口描述符编号.

bNumEndpoint : 使用的端点数目. 端点0除外.

bInterfaceClass : 类型代码, 固定为CC\_VIDEO, 值为0x0E.

bInterfaceSubClass : 子类型代码, 固定为SC\_VIDEOCONTROL, 值为0x01.

bInterfaceProtocol : 协议代码, PC\_PROTOCOL\_15为0x01, PC\_PROTOCOL\_UNDEFINED未定义为0x00.

iInterface : 字符串描述符的索引

#### 类特定视频控制接口描述符

```
/* 1.1.2 Class-specific VideoControl Interface Descriptor */
0x0d,          /* bLength */
0x24,          /* bDescriptorType : CS_INTERFACE */
0x01,          /* bDescriptorSubType : VC_HEADER subtype */
0x10, 0x01,    /* bcdUVC : Revision of class specification that this device is */
0x1e, 0x00,    /* wTotalLength : 30 */
0x80, 0x8d, 0x5b, 0x00, /* dwClockFrequency : 0x005b8d80 -> 6,000,000 == 6MHz */
0x01,          /* bInCollection : Number of streaming interfaces. */
0x01,          /* baInterfaceNr(1) : VideoStreaming interface 1 belongs to this Video */

```

Table 2-5 Class-specific VC Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x0D	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE
2	bDescriptorSubType	1	0x01	VC_HEADER subtype
3	bcdUVC	2	0x0150	Revision of class specification that this device is based upon. For this example, the device complies with Video Class specification version 1.5.
5	wTotalLength	2	0x0042	Total size of class-specific descriptors
7	dwClockFrequency	4	0x005B8D80	Use of this field has been deprecated. This device will provide timestamps and a device clock reference based on a 6MHz clock.
11	bInCollection	1	0x01	Number of streaming interfaces.
12	baInterfaceNr(1)	1	0x01	VideoStreaming interface 1 belongs to this VideoControl interface.

特定于类的VC接口描述符是用于完全描述视频功能的所有描述符的连接，即所有单元描述符（UDs）和终端描述符（TDs）。

特定于类的VC接口描述符的总长度取决于视频功能中的单元和终端的数量。因此，描述符以一个头（VC\_HEADER）开头，该头中wTotalLength字段反映了整个特定于类的VC接口描述符的总字节长度。bcdUVC字段标识了与此视频功能及其描述符兼容的视频设备类规范的版本。bInCollection字段指示此视频控制接口所属的视频接口集合（Video Interface Collection）中有多少个视频流接口（Video Stream Interface）。baInterfaceNr（）数组包含Collection中所有VideoStreaming接口的口号。bInCollection和baInterfaceNr（）字段共同提供了所有必要的信息，以确定哪些接口共同构成了视频功能的整个USB接口，即描述视频接口集合（Video Interface Collection）。

对单元（Unit）描述符和终端（Terminal）描述符的报告顺序并不重要，因为每个描述符都可以通过其描述符类型

（bDescriptorType）和描述符子类型（bDescriptorSubType）字段进行标识。“描述符类型”字段将描述符标识为特定于类的接口描述符。描述符子类型字段进一步限定了描述符的确切性质。

类特定视频控制接口头描述符位于UVC 标准视频控制接口描述符（Standard VC Interface Descriptor）之后，是控制单元和端点的描述符头。

The following table defines the class-specific VC interface header descriptor.

Offset	Field	Size	Value	Description
0	bLength	1	Number	结构体自身长度12+bInCollection(即12+n)
1	bDescriptorType	1	Constant	描述符类型：CS_INTERFACE，值为0x24
2	bDescriptorSubType	1	Constant	描述符子类型VC_HEADER，值为0x01，用于进一步确定描述符的性质
3	bcdUVC	2	BCD	UVC规范协议版本号：UVC15: 0x0150; UVC10: 0x0100
5	wTotalLength	2	Number	类特定VC接口描述符总长度，包含这个描述符的header长度和Until描述符及terminal描述符的长度。即表示整个特定类描述符的总长度。
7	dwClockFrequency	4	Number	时钟频率，以HZ为单位。新规范已经丢弃了
11	bInCollection	1	Number	此Video Control接口所属的Video Interface Collection中有多少个Video Streaming接口数量：n
12	baInterfaceNr(1)	1	Number	在这个VIC集合中第一个VideoStreaming的口号
...	...	...	...	...
12+(n-1)	baInterfaceNr(n)	1	Number	在这个VIC集合中最后一个VideoStreaming的口号

#### 输入端子描述符 Input Terminal Descriptor

```
/* 1.1.3 Video Input Terminal Descriptor (Composite) */
0x08,          /* bLength */
0x24,          /* bDescriptorType : CS_INTERFACE */
0x02,          /* bDescriptorSubType : VC_INPUT_TERMINAL subtype */
0x02,          /* bTerminalID: ID of this input terminal */
0x01, 0x04,    /* wTerminalType: 0x0401 COMPOSITE_CONNECTOR type. This terminal */
0x00,          /* bAssocTerminal: No association */
0x00,          /* iTerminal: Unused*/

```

#### Input Terminal Descriptor in section 3.7.2.1

输入终端描述符 (ITD) 向主机提供与输入终端的功能方面相关的信息。

输入终端由**bTerminalID**字段中的值唯一标识。同一视频功能内的其他单元或终端不得具有相同的ID。该值必须在每个定向到终端的请求的**bTerminalID**字段中传递。

**wTerminalType**类型字段提供了有关输入终端所代表的物理实体的相关信息。这可以是一个USB输出端点，一个外部复合视频连接，一个摄像头传感器，等等。第B.2节“输入终端类型”提供了终端类型代码的完整列表。

**bAssocTerminal**字段用于将输出终端与该输入终端关联，有效地实现了双向终端的配对。这方面的一个例子是摄像机上的一个胶带(胶卷)单元，它将分别有输入和输出终端来接收和源视频。如果使用了这个**bAssocTerminal**字段，则两个关联的终端都必须属于双向终端类型组。如果不存在关联，则必须将**bAssocTerminal**字段设置为零。

主机软件可以将相关的终端视为物理上相关的终端。在许多情况下，一个终端不能因为没有另一个终端而存在。提供了一个到字符串描述符的索引，以进一步描述输入终端。

The following table presents an outline of the Input Terminal descriptor.

Table 3-4 Input Terminal Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	结构体自身长度。即8(+x)
1	bDescriptorType	1	Constant	描述符类型：CS_INTERFACE，值为0x24
2	bDescriptorSubType	1	Constant	描述符子类型VC_INPUT_TERMINAL，值为0x02，用于进一步确定描述符的性质
3	bTerminalID	1	Constant	唯一标识终端的非零常数。此值用于处理此终端的所有请求。
4	wTerminalType	2	Constant	描述终端类型的常数。见附录B，“终端类型”。即：Input Terminal Types
6	bAssocTerminal	1	Constant	与此输入终端关联的输出终端的ID，如果不存在此类关联，则为零(0)。
7	iTerminal	1	Index	描述输入终端的字符串描述符的一个的索引。
...	...	...	...	根据终端类型，某些输入终端描述符具有附加字段。这些特殊终端类型的描述符将在特定于这些终端的单独部分和随附的文档中进行描述。

#### 输出端子描述符 Output Terminal Descriptor

```
/* 1.1.4 Video Output Terminal Descriptor */
0x09,           /* bLength */
0x24,           /* bDescriptorType : CS_INTERFACE */
0x03,           /* bDescriptorSubType : VC_OUTPUT_TERMINAL subtype */
0x03,           /* bTerminalID: ID of this output terminal */
0x01, 0x01,      /* wTerminalType: 0x0101 TT_STREAMING type. This terminal is a */
0x00,           /* bAssocTerminal: No association */
0x02,           /* bSourceID: The input pin of this unit is connected to the output */
0x00,           /* iTerminal: Unused*/
```

#### Output Terminal Descriptor in section 3.7.2.2

输出终端描述符 (OTD) 向主机提供与输出终端的功能方面相关的信息。

输出终端由**bTerminalID**字段中的值作为唯一标识。同一视频功能内的其他单元或终端不得具有相同的ID。该值必须在每个定向到终端的请求的**bTerminalID**字段中传递。

**wTerminalType**字段提供了有关输出终端所代表的物理实体的相关信息。这可以是一个USB IN Endpoint，一个外部复合视频输出连接(external Composite Video Out Connection)，一个液晶显示器(LCD Display)，等等。第B.3节“输出终端类型”(Output Terminal Types)提供了终端类型代码的完整列表。

**bAssocTerminal**字段用于将输出终端与该输入终端关联，有效地实现了双向终端的配对。这方面的一个例子是摄像机上的一个胶带(胶卷)单元，它将分别有输入和输出终端来接收和源视频。如果使用了这个**bAssocTerminal**字段，则两个关联的终端都必须属于双向终端类型组。如果不存在关联，则必须将**bAssocTerminal**字段设置为零。

**bSourceID**字段用于描述此终端的连接性。它包含此输出终端通过其输入引脚连接到的单元或终端的ID。提供了一个到字符串描述符的索引，以进一步描述输出终端。

The following table presents an outline of the Output Terminal descriptor.

Table 3-5 Output Terminal Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	结构体自身长度。即8(+x)
1	bDescriptorType	1	Constant	描述符类型：CS_INTERFACE，值为0x24
2	bDescriptorSubType	1	Constant	描述符子类型VC_OUTPUT_TERMINAL，值为0x02，用于进一步确定描述符的性质
3	bTerminalID	1	Constant	唯一标识终端的非零常数。此值用于处理此终端的所有请求。
4	wTerminalType	2	Constant	描述终端类型的常数。见附录B，“终端类型”。即：Input Terminal Types
6	bAssocTerminal	1	Constant	与此输入终端关联的输出终端的ID，如果不存在此类关联，则为零(0)。
7	bSourceID	1	Constant	此终端所连接到的单元或终端的ID。
8	iTerminal	1	Index	描述输入终端的字符串描述符的一个的索引。
...	...	...	...	根据终端类型，某些输出终端描述符具有附加字段。这些特殊终端类型的描述符将在特定于这些终端的单独部分和随附的文档中进行描述。

#### 标准视频流接口描述符 Standard VS Interface Descriptor

```
/* 1.2 Video Streaming Interface Descriptor */
/* 1.2.1 Operational Alternate Setting 0 */
/* 1.2.1.1 Standard VideoStream Interface Descriptor*/
0x09,           /* bLength */
0x04,           /* bDescriptorType : INTERFACE */
0x01,           /* bInterfaceNumber: Index of this interface */
0x00,           /* bAlternateSetting: Index of this alternate setting */
0x00,           /* bNumEndpoints : 0 endpoints - no bandwidth used*/
0x0e,           /* bInterfaceClass : CC_VIDEO */
0x02,           /* bInterfaceSubClass : SC_VIDEOSTREAMING */
0x00,           /* bInterfaceProtocol : PC_PROTOCOL_UNDEFINED */
0x00,           /* iInterface : unused */

Table 3-13 Standard VS Interface Descriptor
```

Offset	Field	Size	Value	Description
0	bLength	1	Number	描述符的全部长度, 9个字节
1	bDescriptorType	1	Constant	指描述符的类型, 接口描述符该值0x04
2	bInterfaceNumber	1	Number	接口编号。如果一个配置有多个接口的话, 那么每个接口的编号都有一个独立的编号, 编号从0开始递增。 <b>如果只有一个接口这里可以设置为0, 如果不只有一个接口, 从0开始依次递增。</b>
3	bAlternateSetting	1	Number	备用接口编号, 一般很少用, 设置为0。即, 比如一个接口1中可能有多种备选接口进行选择, 这个字段就是用来标识在接口1中我是第几个备选接口, 也是从0开始依次递增。如果没有备选接口该值为0。通常UVC设备都会有备选接口。
4	bNumEndpoints	1	Number	该接口使用的端点个数, 前面讲过一个接口就是一种功能, 每个接口需要用户为其分配端点来实现对应的功能, 注意一点, <b>这个端点个数不包括端点0</b> 。
5	bInterfaceClass	1	Class	Class code (assigned by the USB-IF). 零值保留用于将来的标准化。如果此字段设置为 FFH, 则接口类是特定于供应商的进行自定义。所有其他值保留供 USB-IF 分配。该值取自 <b>defined-class-codes</b> 约定的值。通常USB设备是作为UVC、HID等功能都是这个字段进行定义的。此处作为UVC设备, 设置为 CC_VIDEO, 值为0x0E
6	bInterfaceSubClass	1	SubClass	Subclass code (assigned by the USB-IF). 这些代码由 bInterfaceClass 字段的值限定。如果 bInterfaceClass 字段重置为零, 则该字段也应重置为零。如果 bInterfaceClass 字段未设置为 FFH, 则所有值都保留供 USB-IF 分配。作为UVC 接口类 CC_VIDEO, 此处为SC_VIDEOSTREAMING, 值为0x02
7	bInterfaceProtocol	1	Protocol	Protocol code (assigned by the USB). 这些代码由 bInterfaceClass 和 bInterfaceSubClass 字段的值限定。如果接口支持特定于类的请求, 则此代码标识设备使用的协议, 如设备类的规范所定义。如果此字段重置为零, 则设备不会在此接口上使用特定于类的协议。如果此字段设置为 FFH, 则设备为此接口使用供应商特定的协议。作为 UVC设备, 此处必须为: PC_PROTOCOL_15, 值为0x01
8	iInterface	1	index	描述此接口的描述索引值

#### 类特定视频流接口描述符 Class-Specific VS Interface Descriptors

```
/* 1.2.1.2 Class-specific VideoStream Header Descriptor (Input) */
0x0e,           /* bLength */
0x24,           /* bDescriptorType : CS_INTERFACE */
0x01,           /* bDescriptorSubType : VC_HEADER subtype */
0x01,           /* bNumFormats : One format descriptor follows. */
0x3f, 0x00,     /* wTotalLength : 63 */
0x81,           /* bEndpointAddress : 0x81 */
0x00,           /* bmInfo : No dynamic format change supported. */
0x03,           /* bTerminalLink : This VideoStreaming interface supplies ter */
0x00,           /* bStillCaptureMethod : Device supports still image capture */
0x00,           /* bTriggerSupport : Hardware trigger supported for still ima */
0x00,           /* bTriggerUsage : Hardware trigger should initiate a still i */
0x01,           /* bControlSize : Size of the bmaControls field */
0x00,           /* bmaControls : No VideoStreaming specific controls are supp
```

类特定VS接口描述符由输入头(Input Header)、输出头(Output Header)、格式(Format)和帧(Frame)描述符组成。

每个 VS 接口都有一个输入或输出头描述符, 每个支持的视频流格式都有一个单独的格式描述符, 每个格式描述符都有一个单独的帧描述符列表 (如果格式需要帧描述符)。标头、格式和帧描述符仅在相关接口的备用设置 0 中定义。它们不会在同一界面的后续备用设置中重复。

#### 8.2.1 输入头描述符

##### Input Header Descriptor 3.9.2.1

Input Header Descriptor用于包含流式视频数据的IN Endpoint的 VS Interface。它提供了关于后面的不同格式描述符数量的信息, 以及在此接口的备用设置0中所有特定于类的描述符的总大小。

Offset	Field	Size	Value	Description
0	bLength	1	Number	这个描述符的自身长度。13+(p*n)
1	bDescriptorType	1	Constant	描述符类型: CS_INTERFACE, 值为0x24
2	bDescriptorSubType	1	Constant	描述符子类型VS_INPUT_HEADER, 值为0x01, 用于进一步确定描述符的性质
3	bNumFormats	1	Number	此接口的视频有效载荷格式描述符(Format Descriptor)数量 (不包括视频帧描述符(Frame Descriptor)) : p
4	wTotalLength	2	Number	类特定VS接口描述符总长度, 包含这个描述符的header长度, 即表示整个特定类VS描述符的总长度。
6	bEndpointAddress	4	Endpoint	用于视频数据的等时(isochronous)或批量(BULK)端点的地址。地址编码如下: D7: Direction 1 = IN endpoint D6...4: Reserved, set to zero. D3...0: The endpoint number, determined by the designer.
7	bmlInfo	1	bitmap	Indicates the capabilities of this VideoStreaming interface: 指示此视频流接口的能力 D0: Dynamic Format Change supported D7...1: Reserved, set to zero.
8	bTerminalLink	1	Constant	该接口的视频端点(video endpoint)所连接到的输出终端(output terminal)的终端ID。
9	bStillCaptureMethod	1	Number	支持如第2.4.2.4节“静止图像捕获”所述的静态图像捕获方法: 0: None (Host software will not support any form of still image capture) 1: Method 1 2: Method 2 3: Method 3
10	bTriggerSupport	1	Number	指定通过此接口是否支持硬件触发: 0: Not supported 1: Supported
11	bTriggerUsage	1	Number	指定主机软件应如何响应来自此接口的硬件触发中断事件。这是忽略了如果bTriggerSupport字段为0。 0: Initiate still image capture 启动静态图像捕捉 1: General purpose button event. Host driver will notify client application of button press and button release events 通用按钮事件。主机驱动程序将通知客户端应用程序的按钮按下和按钮释放事件
12	bControlSize	1	Number	每个bmaControls(x)字段的大小, 以字节表示: n
13	bmaControls(1)	1	bitmap	对于位D3...0, 位设置为1表示当bFormatIndex为1时, 视频探测和提交控制支持命名字段: D0: wKeyFrameRate D1: wPFrameRate D2: wCompQuality D3: wCompWindowSize For bits D5...4, a bit set to 1 indicates that the named control is supported by the device when bFormatIndex is 1: D4: Generate Key Frame D5: Update Frame Segment D6...(n*8-1): Reserved, set to zero Note going forward from version 1.5 the proper means to detect whether a field is supported by Probe & Commit is to issue a VS_PROBE_CONTROL(GET_CUR).
...	...	...	...	...
13+(p*n-n)	bmaControls§	1	Bitmap	For bits D3...0, a bit set to 1 indicates that the named field is supported by the Video Probe and Commit Control when bFormatIndex is p: D0: wKeyFrameRate D1: wPFrameRate D2: wCompQuality D3: wCompWindowSize For bits D5...4, a bit set to 1 indicates that the named control is supported by the device when bFormatIndex is p: D4: Generate Key Frame D5: Update Frame Segment D6...(n*8-1): Reserved, set to zero Note D0-D3 are deprecated. Going forward from version 1.5 the proper means to detect whether a field is supported by Probe & Commit is to issue a VS_PROBE_CONTROL(GET_CUR).

```

/* 1.2.1.3 Class-specific VideoStream Format (MJPEG) Descriptor */
0x0b,          /* bLength */
0x24,          /* bDescriptorType : CS_INTERFACE */
0x06,          /* bDescriptorSubType : VS_FORMAT_MJPEG subtype */
0x01,          /* bFormatIndex : First (and only) format descriptor */
0x01,          /* bNumFrameDescriptors : One frame descriptor for this format fo.
0x01,          /* bmFlags : Uses fixed size samples.. */
0x01,          /* bDefaultFrameIndex : Default frame index is 1. */
0x00,          /* bAspectRatioX : Non-interlaced stream - not required. */
0x00,          /* bAspectRatioY : Non-interlaced stream - not required. */
0x00,          /* bmInterlaceFlags : Non-interlaced stream */
0x00,          /* bCopyProtect : No restrictions imposed on the duplication of t]
/* 1.2.1.4 Class-specific VideoStream Frame Descriptor */
0x26,          /* bLength */
0x24,          /* bDescriptorType : CS_INTERFACE */
0x07,          /* bDescriptorSubType : VS_FRAME_MJPEG */
0x01,          /* bFrameIndex : First (and only) frame descriptor */
0x02,          /* bmCapabilities : Still images using capture method 0 are supported at t
MAKE_WORD((IMG_WIDTH / IMG_VIDEO_SCALE)),      /* wWidth : Width of frame, pixels. */
MAKE_WORD((IMG_HEIGHT / IMG_VIDEO_SCALE)),      /* wHeight : Height of frame, pixels. */
MAKE_DWORD(MIN_BIT_RATE),                      /* dwMinBitRate : Min bit rate in bits/s */
MAKE_DWORD(MAX_BIT_RATE),                      /* dwMaxBitRate : Max bit rate in bits/s */
MAKE_DWORD(MAX_FRAME_SIZE),                    /* dwMaxVideoFrameBufSize : Maximum video or still frame size,
MAKE_DWORD(FRAME_INTERVAL),                   /* dwDefaultFrame Interval time, unit=100ns */
0x00,          /* bFrameIntervalType : Continuous frame interval */
MAKE_DWORD(FRAME_INTERVAL),                   /* dwDefaultFrame Interval time, unit=100ns */
MAKE_DWORD(FRAME_INTERVAL),                   /* dwDefaultFrame Interval time, unit=100ns */
0x00, 0x00, 0x00, 0x00,                      /* dwFrameIntervalStep : No frame interval step supported. */
/* 38 bytes, total size 128 */
/* 1.2.2 Operational Alternate Setting 1 */
/* 1.2.2.1 Standard VideoStream Interface Descriptor */
0x09,          /* bLength */
0x04,          /* bDescriptorType: INTERFACE descriptor type */
0x01,          /* bInterfaceNumber: Index of this interface */
0x01,          /* bAlternateSetting: Index of this alternate setting */
0x01,          /* bNumEndpoints: endpoints, 1 - data endpoint */
0x0e,          /* bInterfaceClass: CC_VIDEO */
0x02,          /* bInterfaceSubClass: SC_VIDEOSTREAMING */
0x00,          /* bInterfaceProtocol: PC_PROTOCOL_UNDEFINED */
0x00,          /* iInterface: Unused */
/* 9 bytes, total size 137 */
/* 1.2.2.2 Standard VideoStream Isochronous Video Data Endpoint Descriptor */
0x07,          /* bLength */
0x05,          /* bDescriptorType: ENDPOINT */
0x81,          /* bEndpointAddress: IN endpoint 1 */
0x05,          /* bmAttributes: Isochronous transfer type. Asynchronous */
MAKE_WORD(PACKET_SIZE),                      /* wMaxPacketSize: Max packet size, in bytes */
0x01,          /* bInterval: One frame interval */
/* 7 bytes, total size 144 */
};


```

## UVC 传输 jpg 图片数据到 PC

### UVC 负载数据头 Payload Header Information-摄像头数据包格式分析

UVC数据传输时，每次USB传输，数据包中有一个负载数据头(Payload Header Information)，数据头后为有效的数据。其数据包格式见图：



数据头      数据包

负载数据头为最大为12个字节，包括固定的前2字节的负载数据头和10个字节的扩展负载数据头

数据偏移	数据标识	长度	数据类型	描述
0	bHeaderLength	1	数字	负载数据长 (包括bHeaderLength)
1	bmHeaderInfo	1	位图	采样数据的信息
/	dwPresentationTime	4	数字	时间戳 (PTS)
/	scrSourceClock	6	数字	设备时钟源

Offset	Field	Size	Value	Description
0	bHeaderLength	1	Number	有效负载头的长度（以字节为单位），包括此字段。 bHeaderLength占一个字节，表示帧头的数据长度（包括自己）。
1	bmHeaderInfo	1	Bitmap	bmHeaderInfo 占一个字节，表示的信息比较多。 <ul style="list-style-type: none"> <li>◦ BIT0:经常在0和1之间切换。每次切换表示一个新帧数据的产生。</li> <li>◦ BIT1:帧的结束标志位。这个是可选的。</li> <li>◦ BIT2:如果标识为1，说明 dwPresentationTime 有效并被传输。</li> <li>◦ BIT3:如果标识为1，说明 dwSourceClock 有效并被传输。</li> <li>◦ BIT4:特定位，详见 individual payload specifications 的用法。</li> <li>◦ BIT5:如果为1，表示后面的为抓取的静态图片数据，仅支持2, 3方法</li> <li>◦ BIT6:如果为1，表示传输错误。如果此有效载荷的视频或静止图像传输出错，则设置这位。流错误代码控制将反映错误原因。详见 <a href="http://www.usbz.com/article/detail-9.html">http://www.usbz.com/article/detail-9.html</a></li> <li>◦ BIT7:如果为1，表示是负载数据头的最后一个字节，即 bHeaderLength=2.</li> </ul>

• dwPresentationTime 占4个字节：

演示时间戳 (PTS)。设备中的源时钟，帧捕获时的时钟。

• scrSourceClock 占6个字节,包含2部分：

- D31..D0: Source Time Clock in native device clock units
- D42..D32: 1KHz SOF token counter
- D47..D43: Reserved, set to zero

#### 负载数据头布局

以下是对应未压缩的包头格式的描述。长度48位

1. HLE	Header Length
2. -----	-----
3. BFH[0]   EOH   ERR   STI   RES   SCR   PTS   EOF   FID	-----
4. -----	-----
5. PTS	PTS [7:0]
6.	PTS [15:8]
7.	PTS [23:16]
8.	PTS [31:24]
9. -----	-----
10. SCR	SCR [7:0]
11.	SCR [15:8]
12.	SCR [23:16]
13.	SCR [31:24]
14.	SCR [39:32]
15.	SCR [47:40]

• HLE, 标头长度字段，标头长度字段指定标头的长度（以字节为单位）。

• BFH, 位字节头字段

- FID: 链路识别符，该位在每个帧开始边界处切换，并在其余帧中保持不变。
- EOF: 帧结束，该位指示视频帧的结束，并在属于帧的最后一个视频样本中设置。
- PTS: 演示时间戳。该位置1时表示存在PTS字段。
- SCR: 源时钟参考，该位置1时表示存在SCR字段。
- RES: 保留，设置为0。
- STI: 静止图像，置位时，将视频样本标识为静止图像。
- ERR: 错误位，该位置1时，表明设备帧中存在错误。
- EOH: 标头结尾，该位置1时，指示BFH字段的尾部。

#### void UsbCamera\_Fillbuf(void) 发送视频数据到PC

```

{
    s32 datalen = 0; // 本次发送的字节数
    u8 *payload = 0; // 发送数据指针

    // 发送缓冲区有效数据地址
    payload = sendbuf + CAMERA_SIZ_STREAMHD;
    // 读数据到发送缓冲区
    if (0 == sendsize)
    {
        sendbuf[1] &= 0x01; // 清除BFH
        sendbuf[1] |= 0x01; // 切换FID

        // 计算本次发送数据长度
        datalen = PACKET_SIZE - CAMERA_SIZ_STREAMHD;
        // 读出发送数据
        myMemcpy(sbuf + sendsize, payload, datalen);
        // 图片数据在sbuf数组里面
        sendsize = datalen;
        datalen += CAMERA_SIZ_STREAMHD;
    }
    else
    // 图像的后续包
    {
        datalen = PACKET_SIZE - CAMERA_SIZ_STREAMHD;
        // 判断是否为最后一个数据包
        if (sendsize + datalen >= SBUF_SIZE)
        {
            datalen = SBUF_SIZE - sendsize;
            // 结束包标记(EOF置位,帧结束位指示视频结束,仅在属于图像帧的最后一个USB传输操作中
            // 设置该位)
            sendbuf[1] |= 0x02;
        }
        // 读出发送数据
        myMemcpy(sbuf + sendsize, payload, datalen);
    }
}
// 读出发送数据
myMemcpy(sbuf + sendsize, payload, datalen);

```

• PTS, 图像时间戳记 (PTS) 字段。当BFH[0]字段中的PTS位置1时，将显示PTS字段。请参见“视频设备的USB设备类别定义”规范中的第2.4.3.3节“视频和静止图像有效载荷标题”。

• SCR, 源时钟参考 (SCR) 字段。当在BFH[0]字段中设置SCR时，将出现SCR字段。请参见“视频设备的USB设备类别定义”规范中的第2.4.3.3节“视频和静止图像有效载荷标题”。

YUV帧格式需要看懂下面的H264帧格式，然后再看这个帧YUV格式

#### YUV2视频和静态图像有效载荷帧头数据示例

每包只抓取前16个字节

1. Device Phase Data
2. -----
3. 6.1 IN 0c 0f 00 20 00 00 00 20 00 00 00 20 df 80 df 83
4. 6.1 IN 0c 0e 01 20 00 00 01 20 00 00 01 20 e0 80 e0 83
5. 6.1 IN 0c 0f 02 20 00 00 02 20 00 00 02 20 df 80 dd 83
6. 6.1 IN 0c 0e 03 20 00 00 03 20 00 00 03 20 e3 7f e4 83
7. 6.1 IN 0c 0f 04 20 00 00 04 20 00 00 04 20 df 80 e0 83
8. 6.1 IN 0c 0e 05 20 00 00 05 20 00 00 05 20 e4 7f e0 83
9. 6.1 IN 0c 0f 06 20 00 00 06 20 00 00 06 20 e2 80 de 83
10. 6.1 IN 0c 0e 07 20 00 00 07 20 00 00 07 20 e2 7f e1 83
11. 6.1 IN 0c 0f 08 20 00 00 08 20 00 00 08 20 e1 7f e1 83
12. 6.1 IN 0c 0e 09 20 00 00 09 20 00 00 09 20 e0 7f e0 82
13. 6.1 IN 0c 0f 0a 20 00 00 0a 20 00 00 0a 20 df 7f df 83

下面我们看看H264抓包格式

H264视频和静态图像有效载荷帧头数据示例

下面我们通过BUSBOD抓取一个H264摄像头的数据并进行分析举例，这里我们每包只抓取前16个字节

1.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 00 00 00 01	1包
2.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 2f fa b1 0a	2包
3.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 69 c8 92 51	3包
4.	30.3	IN	0c 0e 00 00 00 00 00 00 00 00 00 ed e7 3e 2c	4包
5.	30	IN		
6.	30.3	IN	0c 0d 00 00 00 00 00 00 00 00 00 00 00 00 01	
7.	30.3	IN	0c 0d 00 00 00 00 00 00 00 00 00 ae 0c 8e 63	
8.	30.3	IN	0c 0d 00 00 00 00 00 00 00 00 00 63 68 78 d0	
9.	30.3	IN	0c 0f 00 00 00 00 00 00 00 00 00 d0 93 b5 aa	
10.	30	IN		
11.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 00 00 00 01	
12.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 df ac 8a 22	
13.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 55 e3 23 cc	
14.	30.3	IN	0c 0e 00 00 00 00 00 00 00 00 00 3c f8 30 f4	
				1帧
				2帧
				3帧
1.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 00 00 00 01	1. 第1帧，第2个字节0c(1100)其中起始帧为0
2.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 2f fa b1 0a	2. 每包帧头都是0c 0c
3.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 69 c8 92 51	3. 每包帧头都是0c 0c
4.	30.3	IN	0c 0e 00 00 00 00 00 00 00 00 00 ed e7 3e 2c	4. 你看，第2字节变成0e(1110) EOF位置1，帧结束
5.	30	IN		
6.	30.3	IN	0c 0d 00 00 00 00 00 00 00 00 00 00 00 00 01	5. 第2帧从第2字节0d(1101)开始，可以看到起始帧从
7.	30.3	IN	0c 0d 00 00 00 00 00 00 00 00 00 ae 0c 8e 63	第1帧的0变成了1。EOF位为0，该帧未结束
8.	30.3	IN	0c 0d 00 00 00 00 00 00 00 00 00 63 68 78 d0	
9.	30.3	IN	0c 0f 00 00 00 00 00 00 00 00 00 d0 93 b5 aa	6. 第2帧第4包第2字节为0f(1111)，EOF位置1，结束帧
10.	30	IN		
11.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 00 00 00 01	7. 第3帧又是第2字节0c(1100)起始帧位从1变为0，表示新的一帧又来了。
12.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 df ac 8a 22	
13.	30.3	IN	0c 0c 00 00 00 00 00 00 00 00 00 55 e3 23 cc	
14.	30.3	IN	0c 0e 00 00 00 00 00 00 00 00 00 3c f8 30 f4	FID: 帧标识符，该位在每个帧起始边界处切换，并在其余帧中保持不变

我的 STM32F103 UVC jpg 图片发送完整代码如下：

```

void UsbCamera_Fillbuf(void) //发生图片数据到 PC
{
    s32 datalen = 0;           // 本次发送的字节数
    u8 *payload = 0;           // 发送数据指针
    // 发送缓冲区有效数据地址
    payload = sendbuf + CAMERA_SIZ_STREAMHD;
    // 读数据到发送缓冲区
    if (0 == sendsize)
    {
        sendbuf[1] &= 0x01;           // 清除 BFH
        sendbuf[1] ^= 0x01;          // 切换 FID
        // 计算本次发送数据长度
        datalen = PACKET_SIZE - CAMERA_SIZ_STREAMHD;
        // 读出发送数据
        myMemcpy(sbuf + sendsize, payload, datalen);
        sendsize += datalen;
        datalen += CAMERA_SIZ_STREAMHD;
    }
    else
    {
        // 图像的后续包
        datalen = PACKET_SIZE - CAMERA_SIZ_STREAMHD;
        // 判断是否为最后一个数据包
        if (sendsize + datalen >= SBUF_SIZE)
        {
            datalen = SBUF_SIZE - sendsize;
            // 结束包标记(EOF 置位),帧结束位指示视频结束,仅在属于图像帧的最后一个 USB 传输操作中设置该位
            sendbuf[1] |= 0x02;
        }
        // 读出发送数据
        myMemcpy(sbuf + sendsize, payload, datalen);
        sendsize += datalen;
        datalen += CAMERA_SIZ_STREAMHD;
    }
    // 复制数据到 PMA
    if (_GetENDPOINT(ENDP1) & EP_DTOG_TX)
    {
        // User use buffer0
        UserToPMABufferCopy(sendbuf, ENDP1_BUFOAddr, datalen);
        SetEPDblBuf0Count(ENDP1, EP_DBUF_IN, datalen);
    }
    else
    {
        // User use buffer1
        UserToPMABufferCopy(sendbuf, ENDP1_BUFIAddr, datalen);
        SetEPDblBuf1Count(ENDP1, EP_DBUF_IN, datalen);
    }
    _ToggleDTOG_TX(ENDP1);           // 反转 DTOG_TX
    _SetEPTxStatus(ENDP1, EP_TX_VALID); // 允许数据发送
    // 判断本帧图像是否发送完成
    if (sendsize >= SBUF_SIZE)
    {
        sendsize = 0;
    }
    return;
}

```

## STM32F103 USB 实现多个虚拟串口

未找到解决方法.....

## STM32F103 USB 读卡器功能(norflash 为例)

按照 STM32F1\_Software\_Base\_opration 文件打 SPI flash 移植完成，也就是 W25Q64 norflash 移植完成。

```

spi_flash
└── norflash.c
└── spi_flash.c

delay_ms(1800);
USB_Port_Set(0); //USB先断开
delay_ms(700);
USB_Port_Set(1); //USB再次连接
Set_USBClock();
USB Interrupts_Config();
USB_Init();
spi_FUN_init(); //SPI初始化
printf("xxxxxxxx\r\n"); //普通串口打印
delay_ms(5000);

usb_printf("VirPort Test\r\n"); //USB打印

while(1)
{
    printf("1111111\r\n"); //普通串口打印
    usb_printf("VirPort Test\r\n"); //USB打印
    SPI_Flash_Read_ID();
    delay_ms(1000);
}

我本来想使用STM32F103 SPI的硬件NSS操作CS,发现还是不行。所以
CS交由软件GPIO控制。

```

SPI flash读取成功

usb pull up enable  
xxxxxxxxxxleave low power mode  
1111111  
Elash ID = 4f  
Elash ID = 4017  
1111111  
Elash ID = 4f  
Elash ID = 4017  
1111111  
Elash ID = 4f  
Elash ID = 4017

## 下面移植 USB 读卡器的配置程序

1. 主要移植官方工程的 Mass\_Storage 存储工程

2. 因为我现有的虚拟串口已经移植了USB驱动库，所以驱动不用再移植。

3. 硬件配置文件应该不变，所以也不用移植

4. 所以我们只需要拷贝官方的这几选中的c文件，进行移植修改。

5. C文件拷贝进我工程了

6. 将对应的h文件移植进自己的工程

7. 工程中加载新传入的文件和覆盖需要修改的老文件

名称	修改日期	类型	大小
Audio_Speaker	2013/1/29 21:02	文件夹	
Composite_Example	2013/1/29 21:02	文件夹	
Custom_HID	2013/1/29 21:02	文件夹	
Device_Firmware_Upgrade	2013/1/29 21:02	文件夹	
JoyStickMouse	2013/1/29 21:02	文件夹	
Mass_Storage	2013/1/29 21:02	文件夹	
Virtual_COM_Port	2013/1/29 21:02	文件夹	
VirtualComport_Loopback	2013/1/29 21:02	文件夹	

名称	修改日期	类型	大小
fsmc_nand.c	2013/1/22 2:27	C 源文件	21 KB
hw_config.c	2013/1/22 2:27	C 源文件	16 KB
main.c	2013/1/22 2:27	C 源文件	4 KB
mass_mal.c	2013/1/22 2:27	C 源文件	8 KB
memory.c	2013/1/22 2:27	C 源文件	6 KB
nand_if.c	2013/1/22 2:27	C 源文件	18 KB
scsi_data.c	2013/1/22 2:27	C 源文件	4 KB
stm32_it.c	2013/1/22 2:27	C 源文件	10 KB
system_stm32f10x.c	2013/1/22 2:27	C 源文件	29 KB
system_stm32f0x.c	2013/1/22 2:27	C 源文件	14 KB
system_stm32f37x.c	2013/1/22 2:27	C 源文件	14 KB
system_stm32l1xx.c	2013/1/22 2:27	C 源文件	20 KB
usb_both.c	2013/1/22 2:27	C 源文件	11 KB
usb_desc.c	2013/1/22 2:27	C 源文件	5 KB
usb_endp.c	2013/1/22 2:27	C 源文件	3 KB
usb_jstr.c	2013/1/22 2:27	C 源文件	6 KB
usb_prop.c	2013/1/22 2:27	C 源文件	13 KB
usb_pwr.c	2013/1/22 2:27	C 源文件	10 KB
usb_scsic.c	2013/1/22 2:27	C 源文件	15 KB

名称	修改日期	类型	大小
hw_config.c	2024/10/27 11:53	C 源文件	11 KB
hw_config.h	2024/10/27 11:58	C Header 源文件	4 KB
mass_mal.c	2013/1/22 2:27	C 源文件	8 KB
memory.c	2013/1/22 2:27	C 源文件	6 KB
platform_config.h	2024/10/20 13:47	C Header 源文件	6 KB
scsi_data.c	2013/1/22 2:27	C 源文件	4 KB
usb_both.c	2013/1/22 2:27	C 源文件	11 KB
usb_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_desc.c	2013/1/22 2:27	C 源文件	5 KB
usb_desch.c	2013/1/22 2:27	C Header 源文件	4 KB
usb_endp.c	2013/1/22 2:27	C 源文件	3 KB
usb_jstr.c	2013/1/22 2:27	C 源文件	7 KB
usb_jstr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_prop.c	2013/1/22 2:27	C 源文件	13 KB
usb_prop.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_pwr.c	2024/10/20 14:58	C 源文件	8 KB
usb_pwr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_scsic.c	2013/1/22 2:27	C 源文件	15 KB
usb_scsic.h	2013/1/22 2:27	C Header 源文件	7 KB

名称	修改日期	类型	大小
fsmc_nand.h	2013/1/22 2:27	C Header 源文件	5 KB
hw_config.h	2013/1/22 2:27	C Header 源文件	3 KB
mass_mal.h	2013/1/22 2:27	C Header 源文件	3 KB
memory.h	2013/1/22 2:27	C Header 源文件	2 KB
nand_if.h	2013/1/22 2:27	C Header 源文件	3 KB
platform_config.h	2013/1/22 2:27	C Header 源文件	8 KB
stm32_it.h	2013/1/22 2:27	C Header 源文件	3 KB
stm32f10x_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
stm32f0x_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
stm32f37x_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
stm32l1xx_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_both.c	2013/1/22 2:27	C Header 源文件	4 KB
usb_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_desc.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_jstr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_prop.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_pwr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_scsic.h	2013/1/22 2:27	C Header 源文件	7 KB

名称	修改日期	类型	大小
hw_config.c	2024/10/27 11:53	C 源文件	11 KB
hw_config.h	2024/10/27 11:58	C Header 源文件	4 KB
mass_mal.c	2013/1/22 2:27	C 源文件	8 KB
memory.c	2013/1/22 2:27	C 源文件	6 KB
platform_config.h	2024/10/20 13:47	C Header 源文件	6 KB
scsi_data.c	2013/1/22 2:27	C 源文件	4 KB
usb_both.c	2013/1/22 2:27	C 源文件	11 KB
usb_conf.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_desc.c	2013/1/22 2:27	C 源文件	5 KB
usb_desch.c	2013/1/22 2:27	C Header 源文件	3 KB
usb_endp.c	2013/1/22 2:27	C 源文件	3 KB
usb_jstr.c	2013/1/22 2:27	C 源文件	7 KB
usb_jstr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_prop.c	2013/1/22 2:27	C 源文件	13 KB
usb_prop.h	2013/1/22 2:27	C Header 源文件	4 KB
usb_pwr.c	2024/10/20 14:58	C 源文件	8 KB
usb_pwr.h	2013/1/22 2:27	C Header 源文件	3 KB
usb_scsic.c	2013/1/22 2:27	C 源文件	15 KB
usb_scsic.h	2013/1/22 2:27	C Header 源文件	7 KB

修改mass\_mal.c文件

```
#include "platform_config.h"  
#include "mass_mal.h"  
#include "norflash.h"  
#include "spi_flash.h"
```

在mass\_mal.c中加入norflash文件

```
uint32_t Mass_Memory_Size[2];  
uint32_t Mass_Block_Size[2];  
uint32_t Mass_Block_Count[2];
```

\_\_IO uint32\_t Status = 0; ← 8. 取消掉Status的定义

9. 官方默认的是32位存储，也就是最大4G的数据

10. 改成以下这样

```
long long Mass_Memory_Size[MAX_LUN+1]; //改成long long就是64位存储  
uint32_t Mass_Block_Size[MAX_LUN+1]; //MAX_LUN在h文件有定义，为1。1代表两个磁盘  
uint32_t Mass_Block_Count[MAX_LUN+1];
```

```
uint16_t MAL_Init(uint8_t lun)  
{  
    uint16_t status = MAL_OK;  
    switch (lun)  
    {  
        case 0:  
            Status = SD_Init();  
            break;  
#ifdef USE_STM3210E_EVAL ← 11. 取消掉官方原定义的内容  
        case 1:  
            NAND_Init();  
            break;  
#endif  
        default:  
            return MAL_FAIL;  
    }  
}
```

12. 修改成这样

```
uint16_t MAL_Init(uint8_t lun)  
{  
    u16 Status=MAL_OK;  
    switch (lun)  
    {  
        case 0:      //磁盘0为 SPI FLASH  
            break;  
        case 1:      //磁盘1为 SD卡  
            break;  
        default:  
            return MAL_FAIL;  
    }  
    return Status;  
}
```

uint16\_t MAL\_Write(uint8\_t lun, uint32\_t Memory\_Offset, uint32\_t \*Writebuff, uint16\_t Transfer\_Length)  
{

switch (lun) ← 13. 官方些磁盘的内容  
{  
 case 0:  
 Status = SD\_WriteMultiBlocks((uint8\_t\*)Writebuff, Memory\_Offset, Transfer\_Length, 1);  
#if defined(USE\_STM3210E\_EVAL) || defined(USE\_STM32L152D\_EVAL)  
 Status = SD\_WaitWriteOperation();  
 while(SD\_GetStatus() != SD\_TRANSFER\_OK)  
 {  
 if ( Status != SD\_OK )  
 {  
 return MAL\_FAIL;  
 }  
#endif /\* USE\_STM3210E\_EVAL || USE\_STM32L152D\_EVAL \*/  
 break;  
#ifdef USE\_STM3210E\_EVAL  
 case 1:  
 NAND\_Write(Memory\_Offset, Writebuff, Transfer\_Length);

如果要支持4G以上的卡，记得改成uint64\_t

```
uint16_t MAL_Write(uint8_t lun, uint32_t Memory_Offset, uint32_t *Writebuff, uint16_t Transfer_Length)  
{  
    u8 STA;  
    switch (lun) //这里，根据lun的值确定所要操作的磁盘  
    {  
        case 0:      //磁盘0为 SPI FLASH盘  
            STA=0;  
            WRITE_norflash_sector(Memory_Offset, (u8*)Writebuff, Transfer_Length);  
            break;  
        case 1:      //磁盘1为SD卡  
            //SD卡未加入  
            break;  
        default:  
            return MAL_FAIL;  
    }  
    if(STA!=0) return MAL_FAIL;  
    return MAL_OK;  
}
```

14. 改成SPI Flash写数据

```

15. 读数据函数需要也修改
uint16_t MAL_Read(uint8_t lun, uint32_t Memory_Offset, uint32_t *Readbuff, uint16_t Transfer_Length)
{
    switch (lun)
    {
        case 0:
            SD_ReadMultiBlocks((uint8_t*)Readbuff, Memory_Offset, Transfer_Length, 1);
#if defined(USE_STM3210E_EVAL) || defined(USE_STM32L152D_EVAL)
            Status = SD_WaitReadOperation();
            如果是大于4G的卡, 改成uint64_t
#endif
        uint16_t MAL_Read(uint8_t lun, uint32_t Memory_Offset, uint32_t *Readbuff, uint16_t Transfer_Length)
        {
            u8 STA;
            switch (lun) //这里, 根据lun的值确定所要操作的磁盘
            {
                case 0: //磁盘0为 SPI FLASH盘
                    STA=0;
                    READ_norflash_sector(Memory_Offset, (u8*)Readbuff, Transfer_Length);
                    break;
                case 1: //磁盘1为SD卡
                    break;
                default:
                    return MAL_FAIL;
            }
            if(STA!=0) return MAL_FAIL;
            return MAL_OK;
        }
    }
}

17. 修改掉官方定义的获取磁盘状态函数
uint16_t MAL_GetStatus (uint8_t lun)
{
#ifdef USE_STM3210E_EVAL
    NAND_IDTypeDef NAND_ID;
    uint32_t DeviceSizeMul = 0, NumberOfBlocks = 0;
#else
    SD_CSD SD_csd;
    uint32_t DeviceSizeMul = 0;
#endif
}

```

```

16. 修改读flash函数
uint16_t MAL_GetStatus (uint8_t lun)
{
    switch(lun)
    {
        case 0:
            return MAL_OK; 改成这样, 简单写
        case 1:
            return MAL_OK;
        default:
            return MAL_FAIL;
    }
}

```

修改mass\_mal.h文件

```

uint16_t MAL_Init (uint8_t lun);
uint16_t MAL_GetStatus (uint8_t lun); 如果是>4G的卡, 记得改成64位
uint16_t MAL_Read(uint8_t lun, uint32_t Memory_Offset, uint32_t *Readbuff, uint16_t Transfer_Length);
uint16_t MAL_Write(uint8_t lun, uint32_t Memory_Offset, uint32_t *Writebuff, uint16_t Transfer_Length);

extern long long Mass_Memory_Size[MAX_LUN+1]; //改成long long就是64位存储
extern uint32_t Mass_Block_Size[MAX_LUN+1]; //MAX_LUN在h文件有定义, 为1。1代表两个磁盘
extern uint32_t Mass_Block_Count[MAX_LUN+1]; //18. 加入外部声明

```

修改memory.c文件

```

2 extern uint8_t Bulk_Data_Buff[BULK_MAX_PACKET_SIZE]; /* data buffer*/
3 extern uint16_t Data_Len;
4 extern uint8_t Bot_State;
5 extern Bulk_Only_CBW CBW;
6 extern Bulk_Only_CSW CSW;
7 extern uint32_t Mass_Memory_Size[2]; 取消掉声明, 因为我在mass_mal中修改实现和声明了
8 extern uint32_t Mass_Block_Size[2];

```

```

/////////////////////////////自己定义的一个标记USB状态的寄存器/
//bit0:表示电脑正在向SD卡写入数据
//bit1:表示电脑正从SD卡读出数据
//bit2:SD卡写数据错误标志位
//bit3:SD卡读数据错误标志位
//bit4:1, 表示电脑有轮询操作(表明连接还保持着)
vu8 USB_STATUS_REG=0;

```

```

void Read_Memory(uint8_t lun, uint32_t Memory_Offset, uint32_t Transfer_Length)
{
    static uint32_t Offset, Length;

    if (TransferState == TXFR_IDLE )
    { 存储>4G这里要强制转换long long
        Offset = Memory_Offset * Mass_Block_Size[lun];
    }
}

```

注意, 不管是内存读写, 如果>4G的卡都要改成uint64\_t

```

void Write_Memory (uint8_t lun, uint32_t Memory_Offset, uint32_t Transfer_Length)
{
    static uint32_t W_Offset, W_Length;

    uint32_t temp = Counter + 64;
    if (TransferState == TXFR_IDLE )

```

```

void Read_Memory(uint8_t lun, uint32_t Memory_Offset, uint32_t T)
{
    static uint32_t Offset, Length;
    if (TransferState == TXFR_IDLE)
    {
        Offset = Memory_Offset * Mass_Block_Size[lun];
        Length = Transfer_Length * Mass_Block_Size[lun];
        TransferState = TXFR_ONGOING;
    }
    if (TransferState == TXFR_ONGOING)
    {
        if (!Block_Read_count)
        {
            MAL_Read(lun,
                      Offset,
                      Data_Buffer,
                      Mass_Block_Size[lun]);
        }
    }
}

void Read_Memory(uint8_t lun, uint32_t Memory_Offset, uint32_t T)
{
    static uint32_t Offset, Length;
    u8 STA; //新增一个定义
    if (TransferState == TXFR_IDLE)
    {
        Offset = Memory_Offset * Mass_Block_Size[lun];
        Length = Transfer_Length * Mass_Block_Size[lun];
        TransferState = TXFR_ONGOING;
    }
    if (TransferState == TXFR_ONGOING)
    {
        if (!Block_Read_count)
        {
            STA=MAL_Read( lun ,
                           Offset ,
                           Data_Buffer,
                           Mass_Block_Size[lun]);
            if(STA)USB_STATUS_REG|=0X08;//SD卡读错误!
        }
    }
}

void Write_Memory (uint8_t lun, uint32_t Memory_Offset, uint32_t Transfer_Length)
{
    static uint32_t W_Offset, W_Length;
    u8 STA;
    uint32_t temp = Counter + 64;
    if (!(W_Length % Mass_Block_Size[lun]))
    {
        Counter = 0;
        STA=MAL_Write( lun ,
                       W_Offset - Mass_Block_Size[lun],
                       Data_Buffer,
                       Mass_Block_Size[lun]);
        if(STA)USB_STATUS_REG|=0X04;//SD卡写错误!
    }
}

这里内存>4G也要写long long

W_Offset = Memory_Offset * Mass_Block_Size[lun];
W_Length = Transfer_Length * Mass_Block_Size[lun];

```

## // Led\_RW\_ON();

修改usb\_scsi.c

```

extern uint32_t Mass_Memory_Size[2];
extern uint32_t Mass_Block_Size[2];
extern uint32_t Mass_Block_Count[2];

```

取消掉

在usb\_conf.h文件

```

/* EP2 */
/* Rx buffer base address */
#define ENDP2_RXADDR (0xD8) 定义ENDP2

```

usb\_conf.h

```

//#define EP1_IN_Callback NOP_Process
#define EP2_IN_Callback NOP_Process
#define EP3_IN_Callback NOP_Process
#define EP4_IN_Callback NOP_Process
#define EP5_IN_Callback NOP_Process

```

这是虚拟串口

时候的配置

```

/*#define CTR_CALLBACK*/
/*#define DOVR_CALLBACK*/
/*#define ERR_CALLBACK*/
/*#define WKUP_CALLBACK*/
/*#define SUSP_CALLBACK*/
/*#define RESET_CALLBACK*/
/*#define SOF_CALLBACK*/
/*#define ESOF_CALLBACK*/
/* CTR service routines */
/* associated to defined endpoints */
//#define EP1_IN_Callback NOP_Process
#define EP2_IN_Callback NOP_Process
#define EP3_IN_Callback NOP_Process
#define EP4_IN_Callback NOP_Process
#define EP5_IN_Callback NOP_Process
#define EP6_IN_Callback NOP_Process
#define EP7_IN_Callback NOP_Process

#define EP1_OUT_Callback NOP_Process
#define EP2_OUT_Callback NOP_Process
#define EP3_OUT_Callback NOP_Process
#define EP4_OUT_Callback NOP_Process
#define EP5_OUT_Callback NOP_Process
#define EP6_OUT_Callback NOP_Process
#define EP7_OUT_Callback NOP_Process
#endif /* __USB_CONF_H */

```

屏蔽掉EP1\_IN和EP2\_OUT，不然编译的时候报两个.o文件重复性错误

类似官方 LED 这种，屏蔽掉

在hw\_config.c中屏蔽掉虚拟串口的内容

```

//extern LINE_CODING linecoding;
bool USART_Config(void)
{
    // uu_txfifo.readptr=0; //清空读指针
    // uu_txfifo.writeptr=0; //清空写指针
    // USB_USART_RX_STA=0; //USB USART接收状态清零
    // printf("linecoding.format:%d\r\n",linecoding.format);
    // printf("linecoding.paritytype:%d\r\n",linecoding.paritytype);
    // printf("linecoding.datatype:%d\r\n",linecoding.datatype);
    // printf("linecoding.bitrate:%d\r\n",linecoding.bitrate);
    return (TRUE);
}

```

```

#include "spi_flash.h"
#include "norflash.h"

```

```

int main(void)
{
    delay_init(); //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    USART1_Config(115200);
    spi_FUN_init(); //SPI初始化
    flash初始化要放在USB
    delay_ms(1800);
    USB_Port_Set(0); //USB先断开 之前
    delay_ms(700);
    USB_Port_Set(1); //USB再次连接
    Set_USARTClock();
    USB Interrupts_Config();
    USB_Init();

    printf("xxxxxxxx\r\n"); //普通串口打印
    delay_ms(5000);

    usb_printf("VirPort Test\r\n"); //USB打印

    while(1)
    {
        printf("1111111\r\n"); //普通串口打印
        usb printf("VirPort Test\r\n"); //USB打印
        SPI_Flash_Read_ID();
        delay_ms(1000);
    }
    return 0;
}

```

```

int main(void)
{
    delay_init();          //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组2
    USART1_Config(115200);

    spi_FUN_init(); //SPI初始化
    delay_ms(500);
    SPI_Flash_Read_ID(); // 调整了一下代码，让 flash初始化在USB之前

    delay_ms(1800);
    USB_Port_Set(0); //USB先断开
    delay_ms(700);
    USB_Port_Set(1); //USB再次连接
    Set_USBClock();
    USB Interrupts_Config();
    USB_Init();

    printf("xxxxxxxx\r\n"); //普通串口打印
    delay_ms(5000);

    while(1)
    {
        printf("1111111\r\n"); //普通串口打印
        delay_ms(1000);
    }
    return 0;
}

```

```

int main(void)
{
    delay_init();          //延时函数初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 设置中断优先级分组2
    USART1_Config(115200);

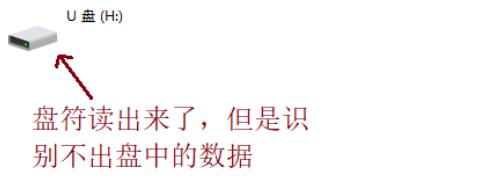
    spi_FUN_init(); //SPI初始化
    delay_ms(500);
    SPI_Flash_Read_ID(); // 因为[0]是SPI FLASH [1]是SD卡，所以先初始化flash容量信息
    Mass_Memory_Size[0]=1024*1024*12; //前12M字节
    Mass_Block_Size[0] =512;           //设置SPI FLASH的操作扇区大小为512
    Mass_Block_Count[0]=Mass_Memory_Size[0]/Mass_Block_Size[0];
    delay_ms(1800);
    USB_Port_Set(0); //USB先断开
    delay_ms(700);
    USB_Port_Set(1); //USB再次连接
    Set_USBClock();
    USB Interrupts_Config();
    USB_Init();

    printf("xxxxxxxx\r\n"); //普通串口打印
    delay_ms(5000);

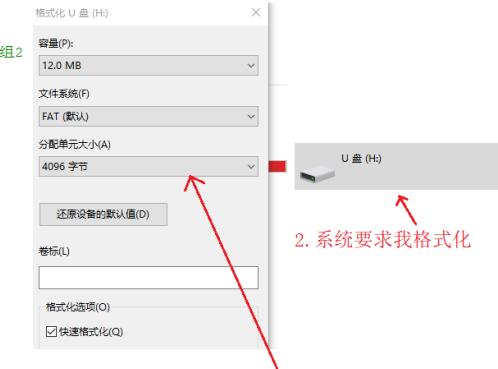
    while(1)
    {
        printf("1111111\r\n"); //普通串口打印
        delay_ms(1000);
    }
}

/*写数据进flash扇区*/
/*addr:要写入flash扇区的起始地址
 *writebuff:将256个字节写入flash扇区的缓存
 *byte_number:写多少个字节
 */
void WRITE_norflash_sector(uint32_t addr,uint8_t *writebuff,uint32_t byte_number)
{
    if(byte_number > 256) //判断如果写入大于256个字节就报错，退出程序
    {
        printf("write flash data error\r\n");
        return;
    }
    else
    {
        spi_Erase_sector(addr); //写扇区之前要擦除扇区
        wait_flash_Erase(); //等待flash内部擦除完成
        norflash Write_ENABLE(); //擦除扇区完了之后，写扇区也要使能
    }
}

```



盘符读出来了，但是识别不出盘中的数据



1. 初始化加入flash信息之后

3. 但是我flash每次写不能>256字节，而系统最小格式扇区都是4096

4. 也许W25Q64 flash 容量不适合做U盘

所以可能使用 SDIO 驱动 SD 卡可以成功实现 USB 读卡器存储设备。也可以加 FATFS 文件系统尝试下能否单片机和 PC 的 USB 端都可以分部访问 SD 卡里面得文件内容。