 **USB概述及协议(simon.yuan)**

**USB 概述及协议**

# 目录

本文章主要是基于学习《圈圈教你玩USB》书籍的学习笔记,从根本上理解USB的通信过程及底层驱动的编写，为后续做USB相关的开发打下基础.

[1 PDIUSBD12 ChipID error..........................................](#_bookmark0)2

[2 USB鼠标的实现.................................................. ......4](#_bookmark1)

[2.1 模拟USB插拔......................................................5](#_bookmark2)

[2.2 Soft connect........................................................6](#_bookmark5)

[2.3 中断类型分类散转...............................................7](#_bookmark9)

[2.4 Endpointer0 OUT缓冲区数据读取.......................8](#_bookmark21)

[2.5 Ep0 OUT缓冲区数据内容…….............................8](#_bookmark23)

[2.6 Ep0 OUT数据对比USB协议解析.........................8](#_bookmark25)

[2.7 USB PHY芯片接收数据工作原理........................9](#_bookmark26)

[2.8 清除中断标志位及缓冲区...................................10](#_bookmark26)

[2.9 主机请求获取设备描述符...................................10](#_bookmark34)

[2.10 设备描述符包结构............................................11](#_bookmark34)

[2.11 返回USB设备描述符..........................................13](#_bookmark34)

[2.12 设置USB地址.....................................................15](#_bookmark34)

[2.13 获取配置描述符................................................16](#_bookmark34)

[2.14 获取字符串描述符............................................18](#_bookmark34)

[2.15 设置配置..........................................................18](#_bookmark34)

[2.16 类输出请求......................................................18](#_bookmark34)

[2.17 HID鼠标报告上传.............................................19](#_bookmark34)

[2.18 HID报告描述符.................................................19](#_bookmark34)

[2.19 USB通信层次实质.............................................20](#_bookmark34)

[3 USB CDC....................................................................21](#_bookmark43)

[3.1 CDC定义 .............................................................22](#_bookmark44)

[3.2 CDC定义 .............................................................22](#_bookmark44)

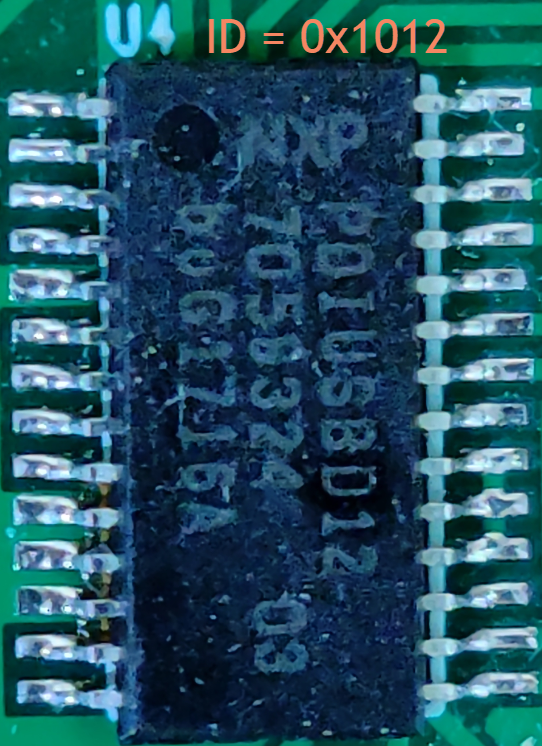
[3.3 类请求实现 ........................................................22](#_bookmark44)

# PDIUSBD12 Chip ID error

1. **USB PHY芯片PDIUSBD12的芯片ID错误**

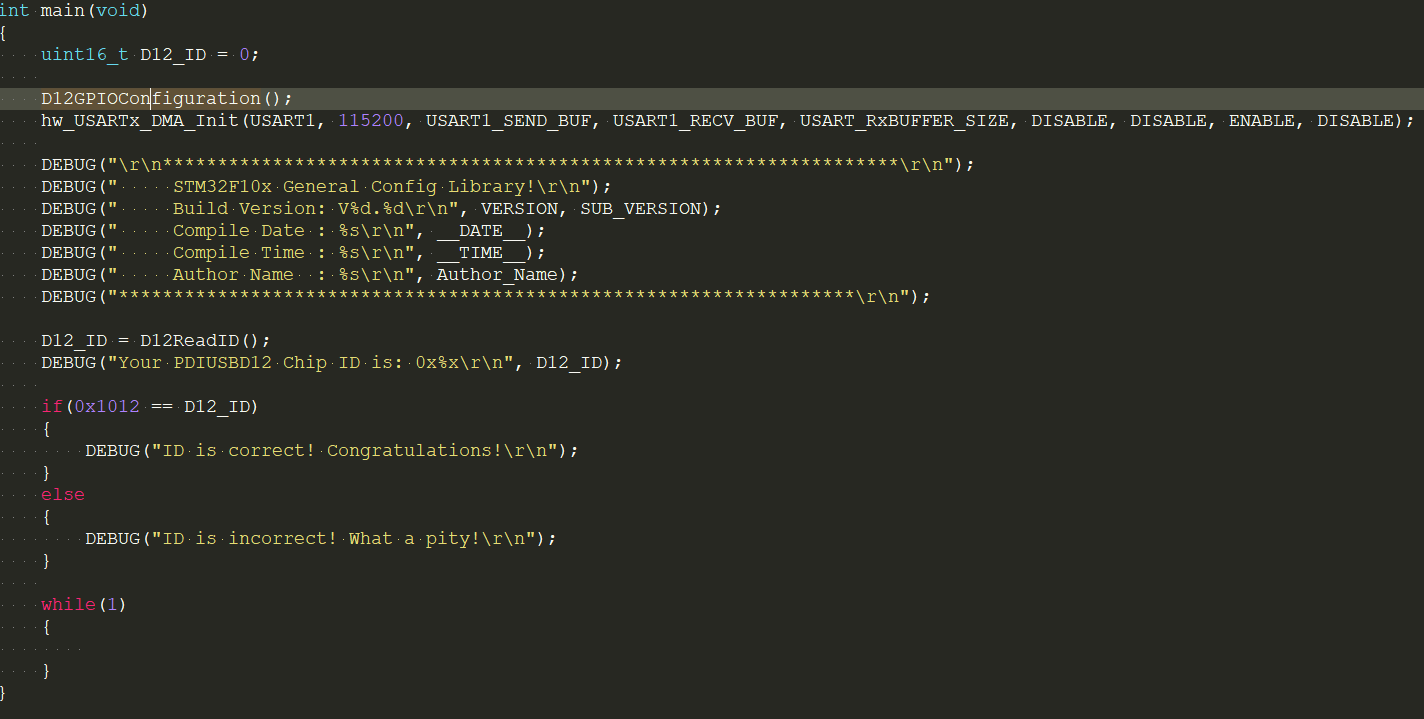
在淘宝上购买的USB PHY芯片PDUSBD12, 编写固件读取芯片ID时获取到的ID号为0x1010, NXP官方PDIUSBD12芯片的正确ID为0x1012, 验证后发现购买到了假货.

以下为真假芯片对比图片



1. **固件验证**

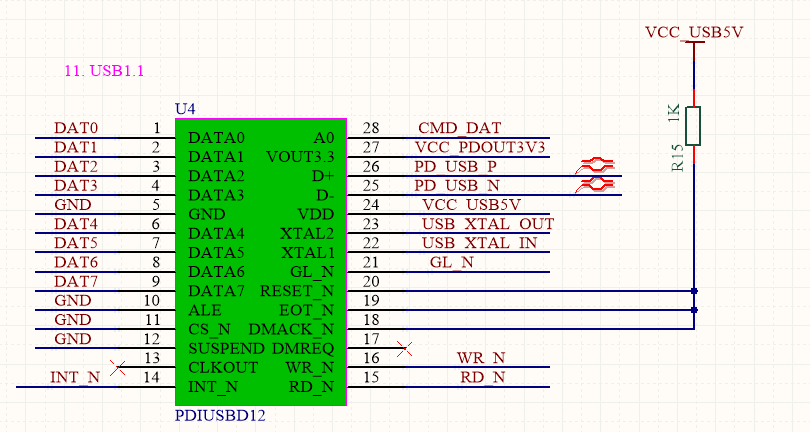
运行固件并单步调试，示波器在单步调试时测量时序控制的信号，发现信号电平及时序没有问题，但是电平信号有些许纹波，查找发现电源部分也有纹波.



1. **硬件验证**

主控MCU的IO口与USB PHY芯片之间的连接





1. 验证
2. 已知OK的板子，焊接上假芯片，用已知OK的固件测试，读取芯片ID为0x1010；已知OK的板子配已知真芯片读取ID为0x1012
3. 已知OK的板子，焊接上假芯片，用已知OK的USB模拟鼠标的固件测试，在板子上电固件初始化完成后，在没有任何外部按键触发的情况下，串口打印USB处于一直触发的情况，这在OK的芯片上是没有出现的
4. 已知FAIL的板子，焊接上OK的芯片，用已知OK的固件测试，读取芯片ID为0x0000；且已知FAIL的板子配已知假芯片读取ID为0x0000
5. 总结
6. 假芯片

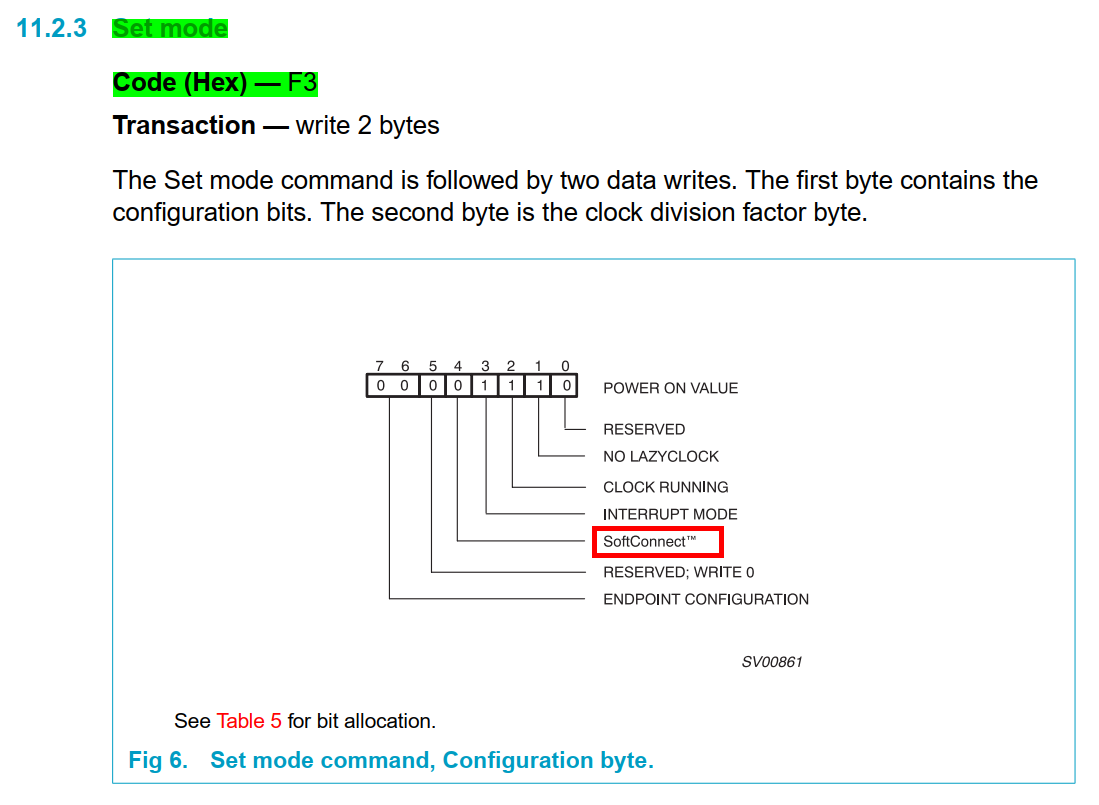
一些重要的芯片，还是在大网站(Digi- Key), 官网或官方代理采购，对于小批量试试能否索样.

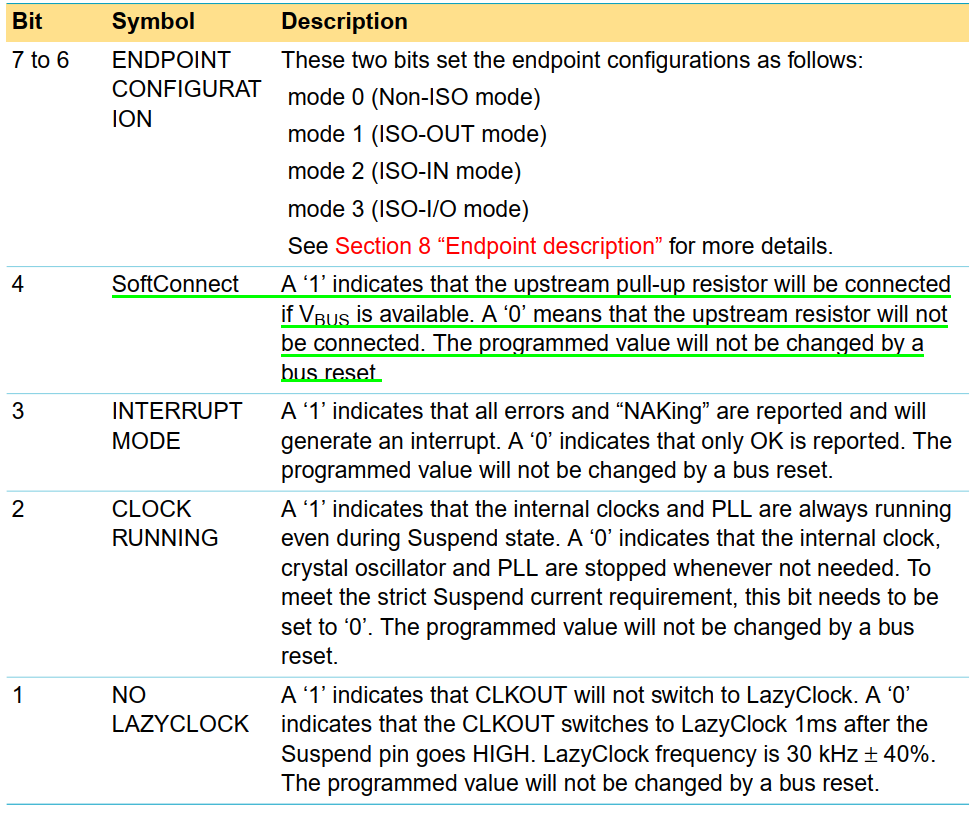
1. 硬件设计问题

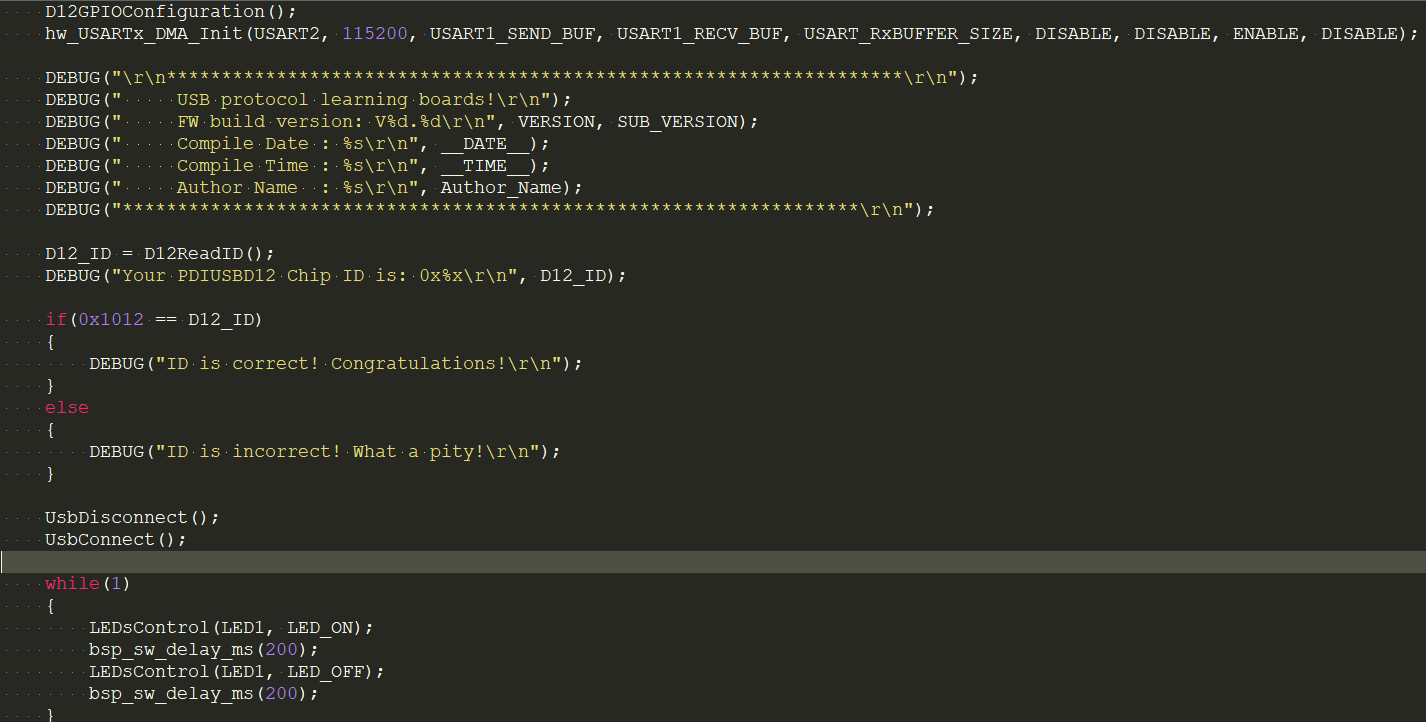
* 电源的纹波问题较严重
* JTAG的IO口在设计时用做了USB芯片数据/命令复用选择，读写选通等功能
* 信号线通信时需要交叉的情况下，先经过斟酌后再进行图纸设计
* 数据通信的是8bit并口，但是官方库函数在进行端口读写时是16bit的，所以剩余的8bit如果同时在使用的话，硬件设计上需要避开(例如：PA0-PA7作为并口数据通信，PA8-PA15中的PA9, PA10同时复用为USART时，此时如果调用官方的库函数对端口进行读写时，会影响USART的打印输出)

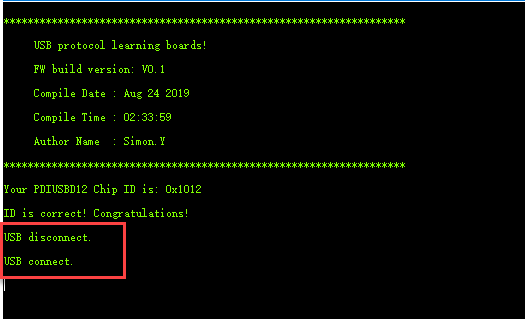
# USB鼠标的实现

## 2.1) 首先需要模拟一个USB在复位的时候USB拔出然后再插入的过程(涉及到USB PHY芯片的软连接的连接及断开)通过PDUSBD12的设置模式命令(set mode)来设置芯片内部的USBDP的上拉软连接实现USB的插入检测.需要发送命令码F3之后，再发送2个字节的数据来进行设置，具体的2字节的设置数据见以下详细注释.

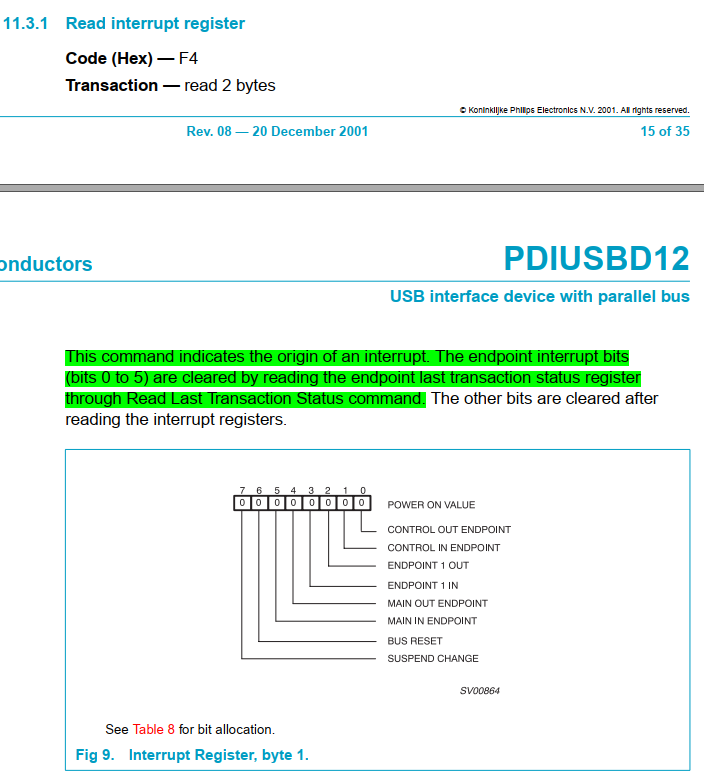
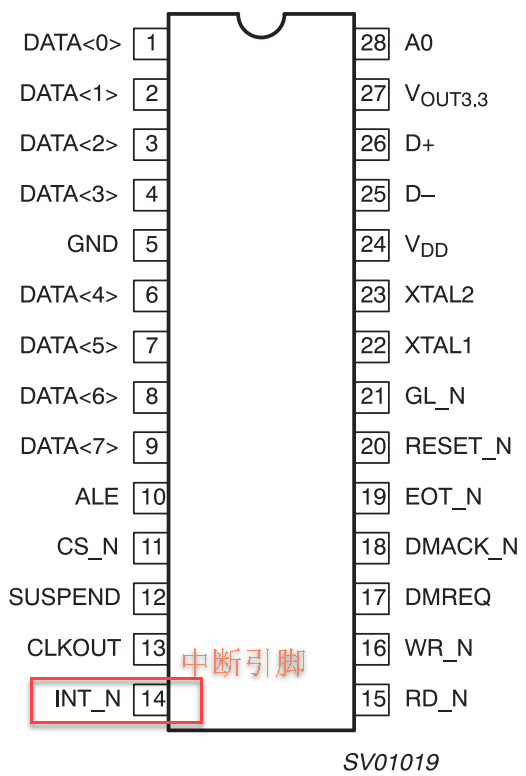




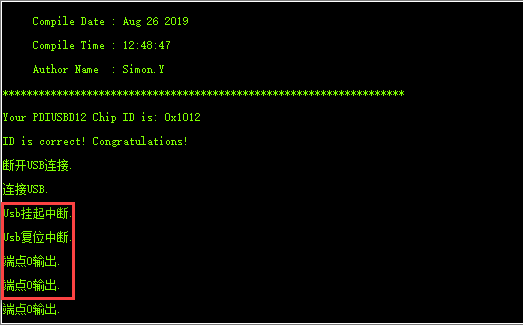
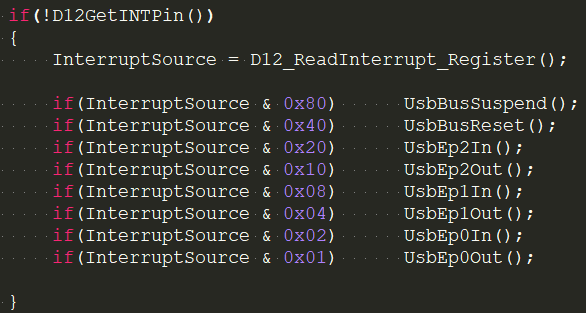




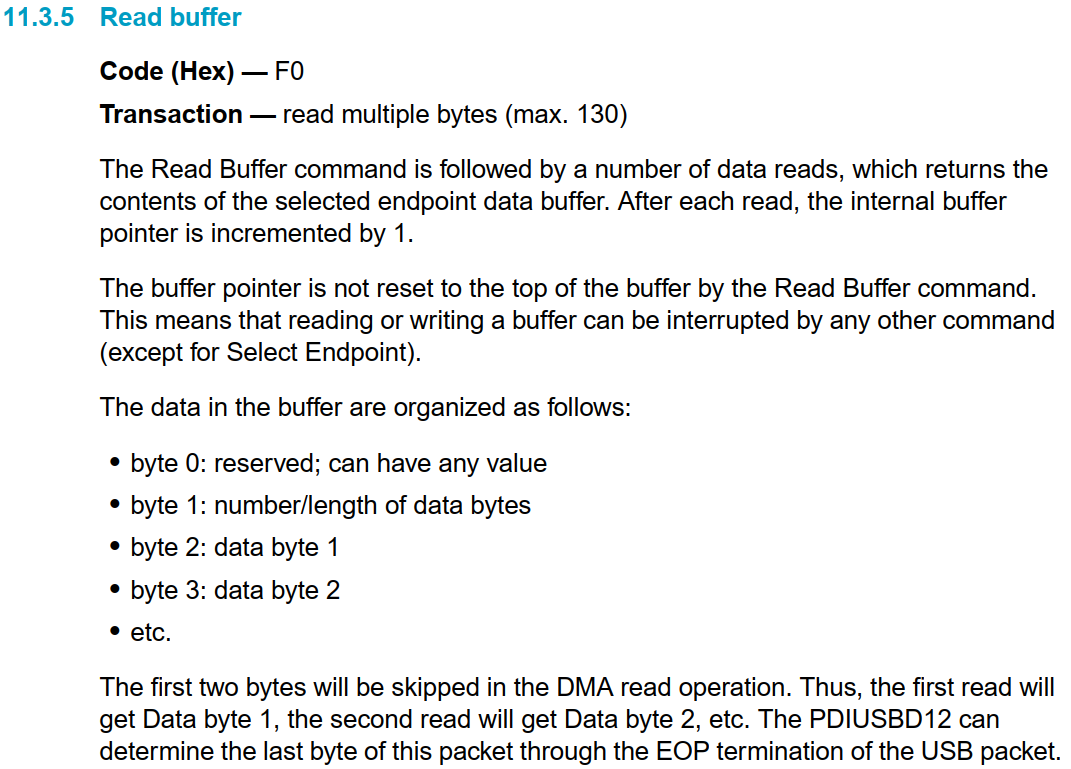
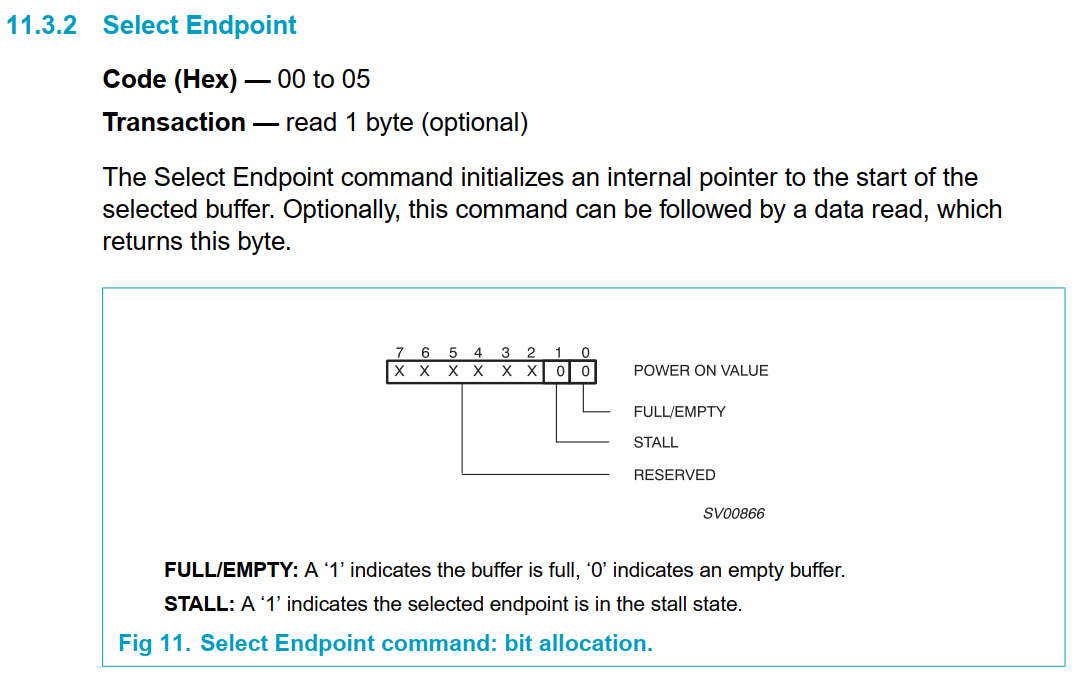
## 2.2) USB的DP端的上拉软连接连接上后，相当于USB进行了插入主机，USB主机此时会发送一些获取描述符的请求，而对于USB PHY芯片而言，由于在set mode寄存器中设置了所有的中断都会有中断响应产生，这里将通过D12的中断产生后INT引脚置高对其中断寄存器中的终端标志位进行查询来查看是产生了哪种中断.

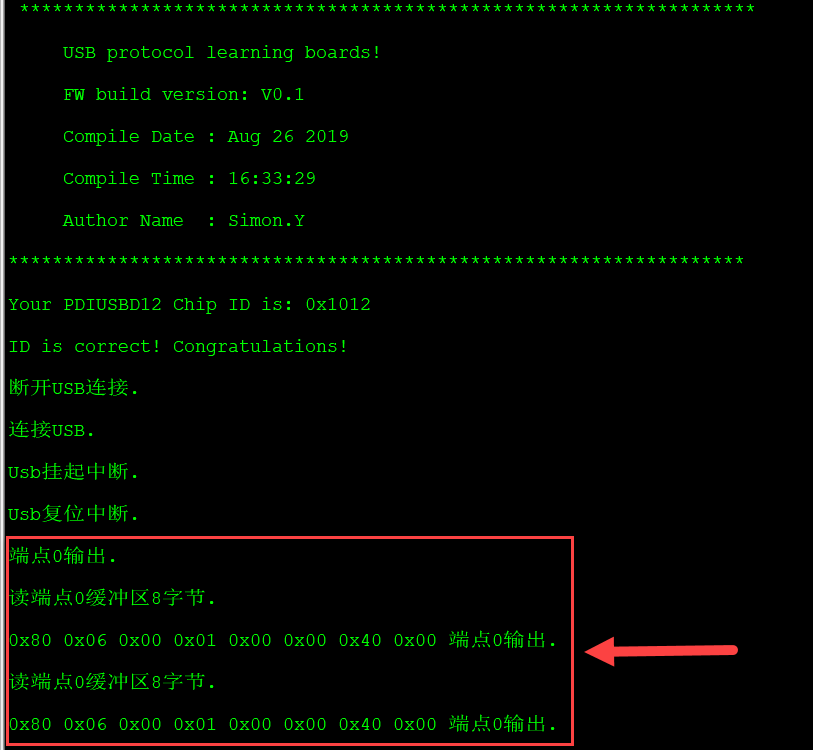


## 2.3) 对中断标志位进行判断，发现USB主机与USB设备在没有发生通信时(3ms没有数据通信)，USB的总线被挂起，挂起后USB主机对USB设备进行了复位，复位完成后USB主机与USB设备的端点0进行通信，可以看出USB主机向USB设备的端点0输出了数据.



## 2.4) USB主机向USB设备的端点0究竟输出了什么数据呢? 将数据读出来进行分析.对于USB PHY芯片D12的的数据读取方式为：(1)首先需要选取待读取数据的端点. (2)读取指定端点的数据并存放到相应的缓冲区.





## 2.5)可以看出USB总线复位之后，USB主机向USB设备的端点0发送了8个字节，具体的数据为上面的截图所示。0x80, 0x06, 0x00, 0x01, 0x00, 0x00, 0x40, 0x00

## 2.6)对这些数据进行分析，由于是USB主机发送到USB设备的，这就涉及到USB协议的标准请求. 首先了解下USB标准设备请求的结构，摘录USB2.0协议的第9章节，可以查看USB设备的通用请求结构，可以将之前的数据与这个结构进行一一对应，来理解具体含义.

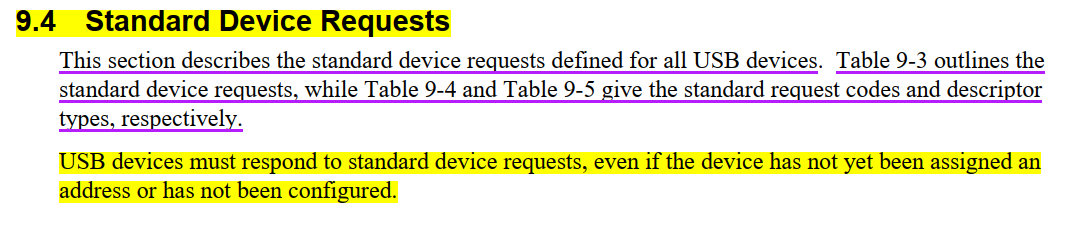
### 0x80: bmRequestType – 主机需要获取数据，数据需要从设备传向主机，请求的数据类型为标准类型，指定接收这为设备

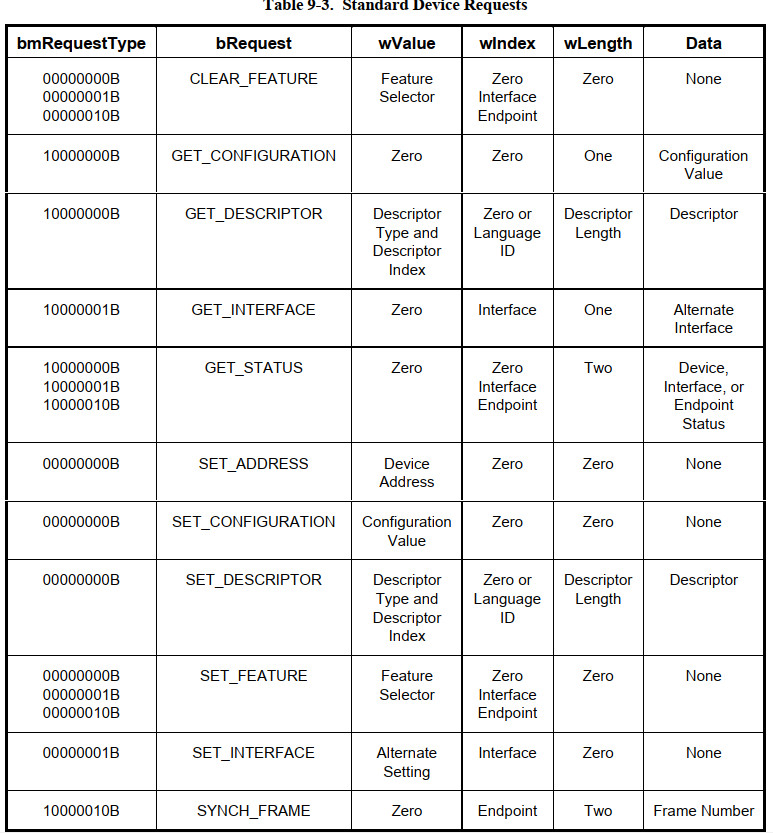
### 0x06: bRequest – 此项由 bmrRequestType中断额D6:5的Type来修饰的特殊类型(非标准类型)，由标准需求代码可知，USB主机需要获取USB设备的描述符GET\_DESCRIPTOR

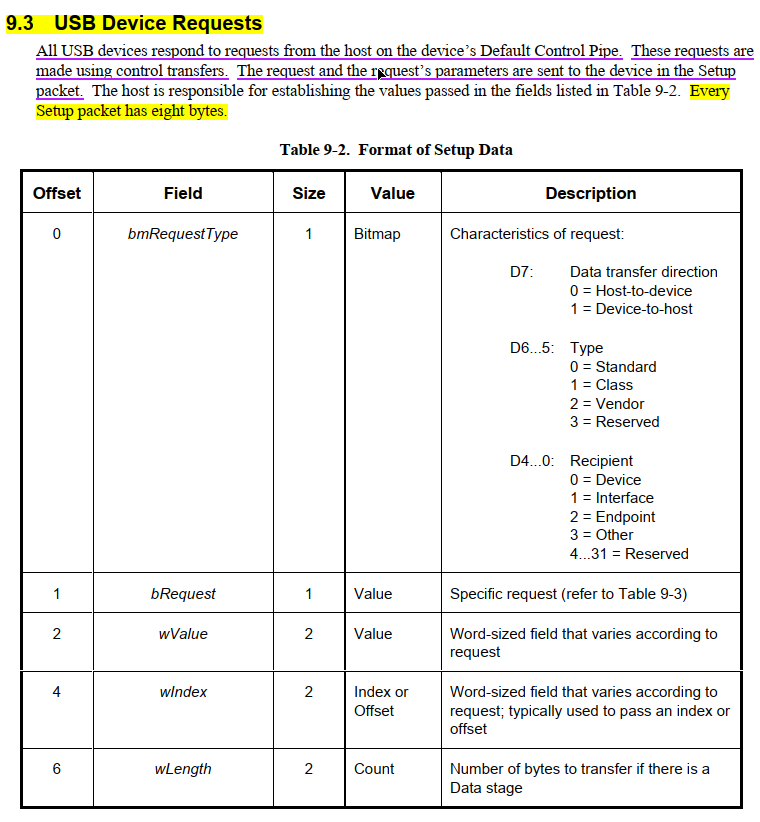
### 0x00, 0x01: wValue – 此值为变量用于传输一个参数到设备，具体化需求. 两个字节，低字节表示索引号(用来指明同一种描述符中的具体某个具体描述符，如配置描述符集合中某个描述符)，高字节表示描述符的类型编号(即表9-5中的描述符类型)，这里的0x01表示主机需要获取设备描述符

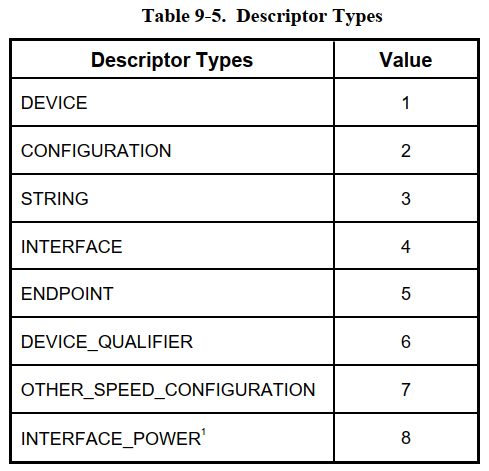
### 0x00, 0x00: wIndex – 只在获取字符描述符时有用，表示字符串的语言ID，获取除字符串描述符的其它描述符时，wIndex值为0

### 0x40,0x00: wLength – 请求设备返回数据的字节数

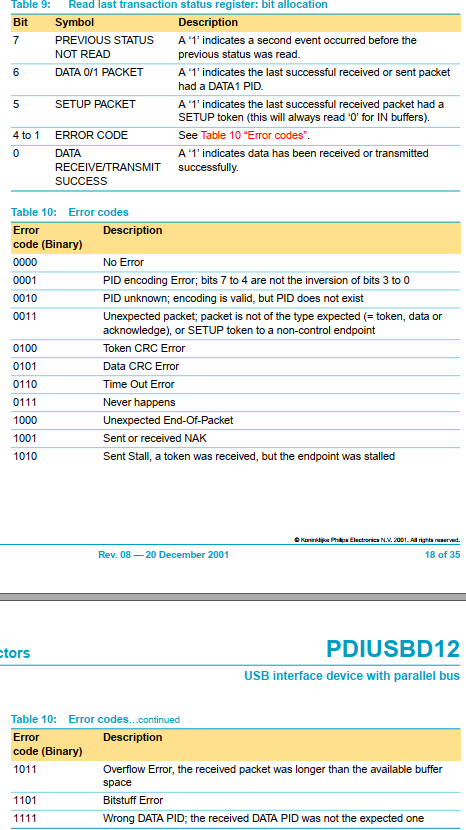
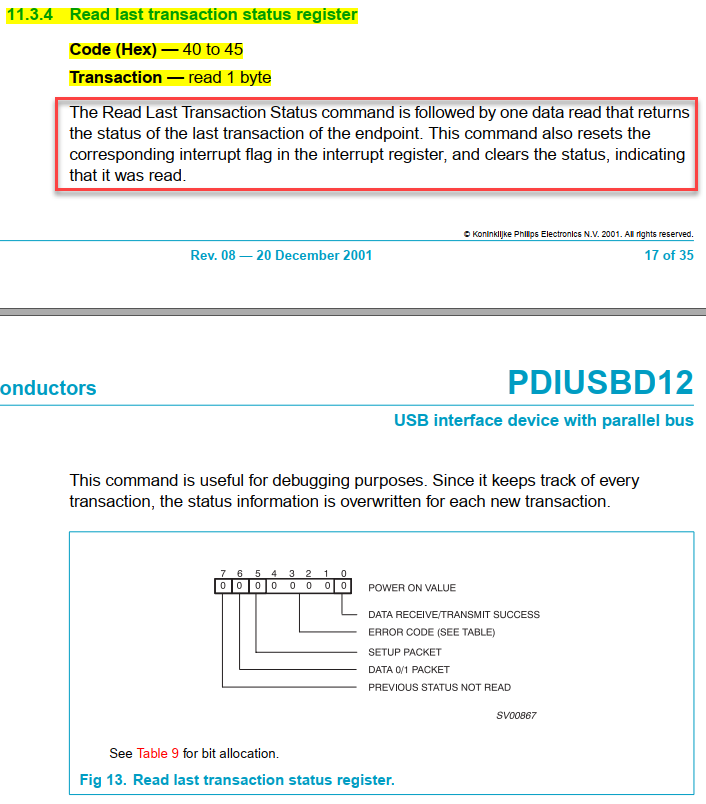




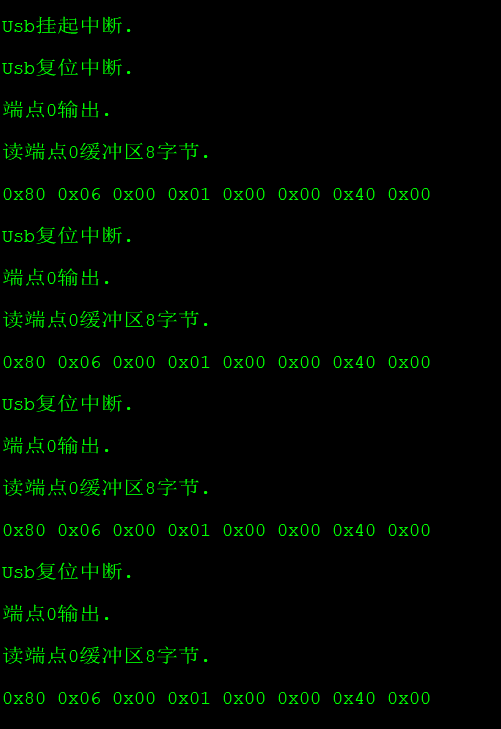




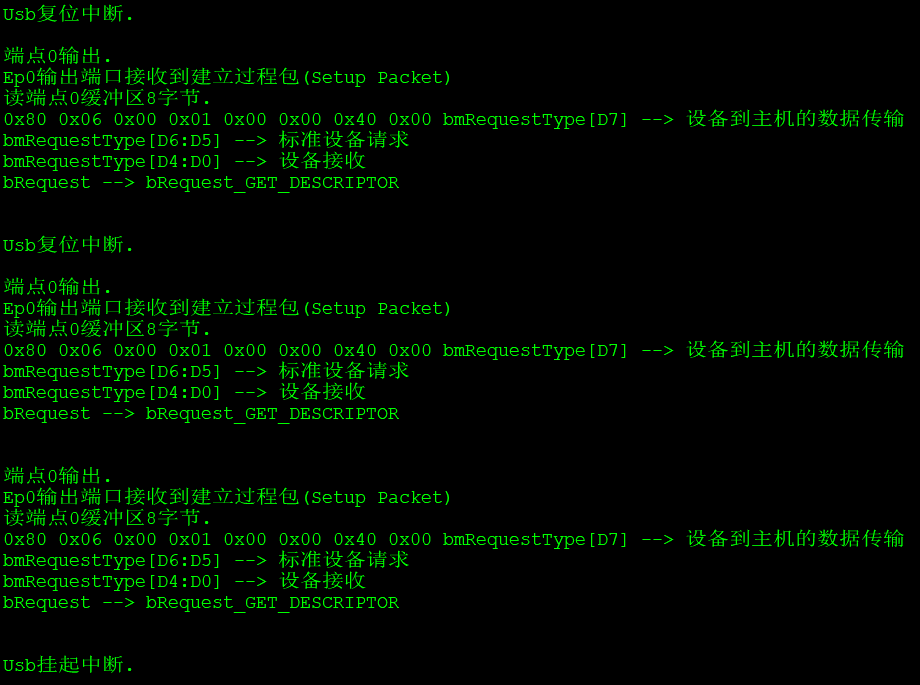
## 2.7)由于USB PHY接收到数据的查看方式是通过在接收到数据后，将其INT引脚拉低并对相应的中断标识位置位一系列动作实现的，那么需要将中断标识位清零且对USB PHY中已经读取了缓存内容的缓存区进行清空，如果中断标志位没有进行清除，那么就会一直提示有中断发生.



## 2.8)在使用清除缓冲区命令对缓冲区进行清除之前需要确认当前Endpoint0 out收到的数据包是否为建立过程包，如果是的话，需要先发送建立包ACK(发送后认为建立包已经被正确接收，做为确保通信过程数据的正确性及完整性)，然后才能使用ClearBuffer命令清除OUT缓冲区及ValidateBuffer命令IN缓冲区.清除中断标志位及缓冲区后，由于USB主机无法收到USB设备的返回数据包，会对USB总线进行再次复位并再次发送获取设备描述符的请求，在发送大约4次之后，USB主机会因此无法加载驱动而无法识别USB设备.



## 2.9)USB主机需要USB设备返回设备描述符，整个过程总结如下图，这个完整的过程的解析，USB主机在多次复位设备之后，无法获取到相应回包后将设备挂起。

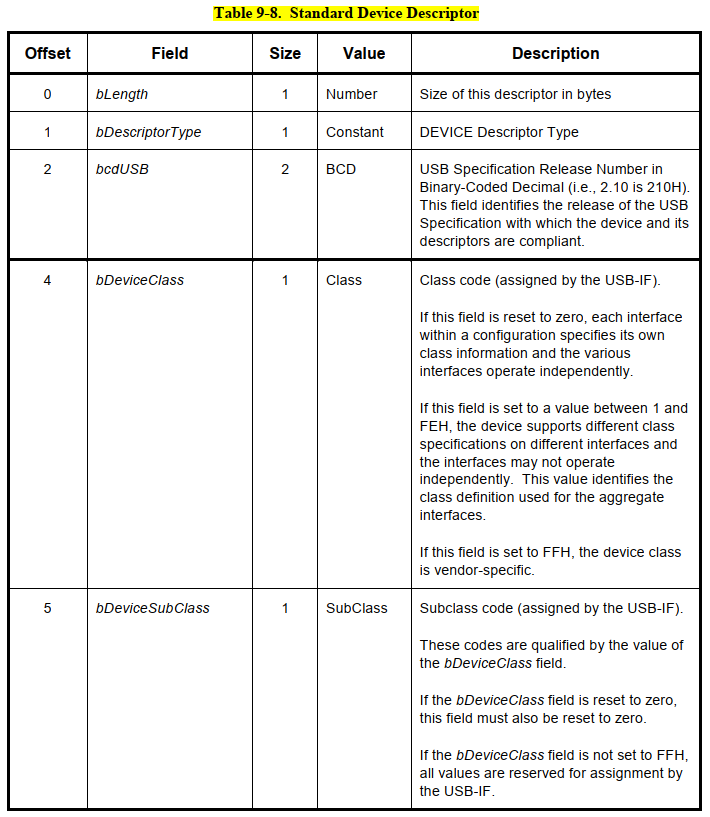


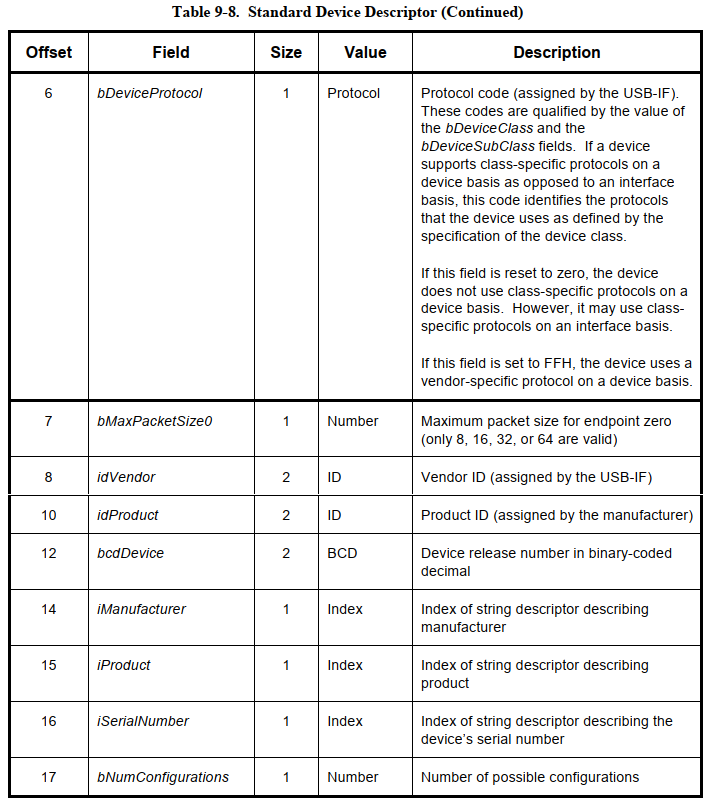
## 2.10)USB主机需要USB设备返回设备描述符，那么需要构建一个USB设备描述符通过端点0发送给USB主机。

### 首先需要明白设备描述的数据结构及相应的数据含义，标准设备描述符包含18个字节，具体含义如下图的Table9-8(摘自USB2.0官方协议).

### 有了设备描述符的包结构后，代码中需要构建一个数据结构作为数据包回包的结构.以下为数据结构相对应的意义.







## 2.11)USB设备返回设备描述符.

### 对于D12，USB主机发送了获取设备描述符的请求，USB设备做为被动执行，其Ep0Out(端点0输出口会有数据接收)产生接收中断从而标志位置位. 通过Ep0Out端点的读缓冲区操作获取接收到的主机发送过来的数据，如下图中断额Read buffer的详细描述进行操作.

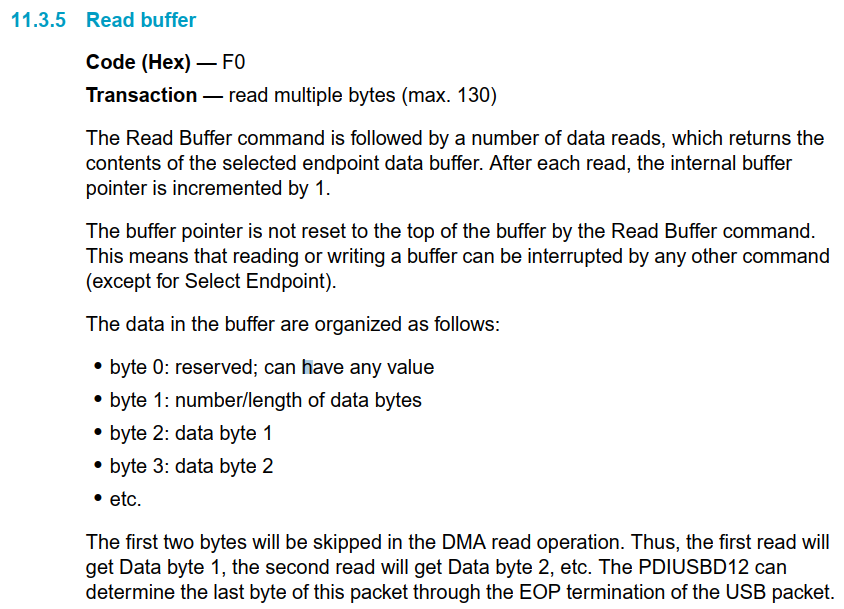
### 然后对读取到的数据包的字段进行解析，解析出来为设备描述符的获取请求

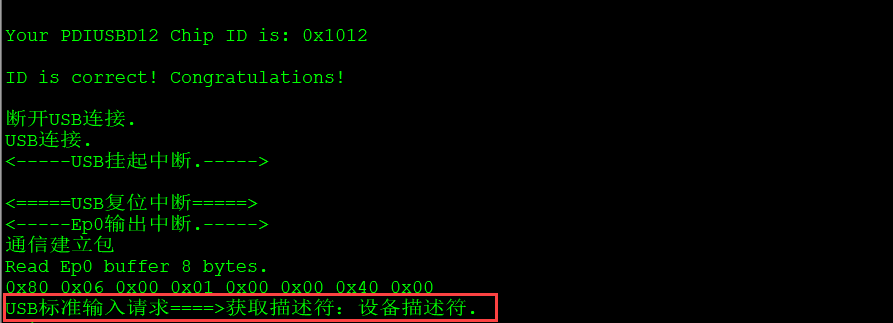
### 构建设备描述符的数据包(共18个字节)并进行发送，由于设备描述符的bMaxPacketSize0我们只能设置最大为16个字节(D12芯片的端点0最大包长)，所以数据需要作多次发送(一次最多只能发送16个字节)

### 注意：当 USB设备发送数据包的长度为端点最大包长的整数倍时，需要发送一个0长度包作为数据结束包给USB主机.(由于USB设备发送设备描述符给USB主机后，USB主机可以了解到USB设备的端点的最大包长为16个字节,但是USB主机不通过解析数据包中的每个包长字节来做判断数据是否结束，毕竟有时会存在丢帧或者丢数据的可能性，而是当数据包为发送端点最大包长的整数倍时，在没有多余的数据发送时需要发送一个0长度的数据包做为结束包.打个比方，USB设备需要发送48个字节数据包，但是每次只能发送16个字节长度，需要连续发送3次才能发送完成，虽然USB主机在接收到第一个包的时候知道需要接收48个字节的数据包，但是USB主机不会去统计总长度，因为可能有丢数据的可能性，比如了丢了2个字节，但是USB设备在发送完48个字节后，再发送一个0长度数据包，在USB主机接收到这个包后，可以判断丢包率).

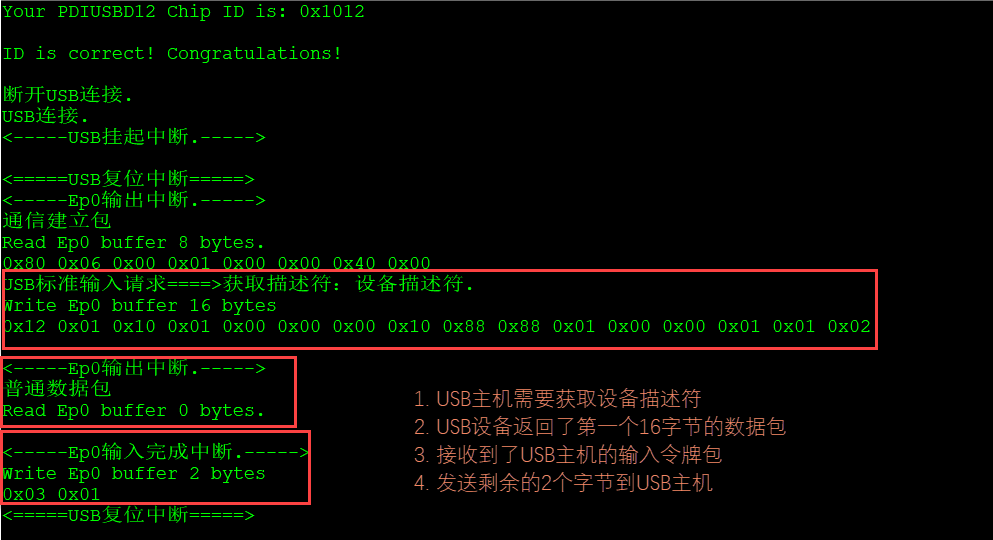
### 在设备描述符的数据包构建好了之后，通过端点0将数据包发送出去UsbEp0SendData函数发送数据出去，这里注意，USB端点的发送和接收缓冲区其实是共用的，可以看Datasheet中的命令代码是共用一个.

### 当发送数据包长度大于端点最大包长时，第一次在Ep0Out的ISR中断中解析处主机获取设备描述符之后，先发送最大包长16个字节后，缓冲区数据发送完成后，后面的数据或者数据包在主机发送了输入令牌包之后产生UsbEp0In中断，然后再UsbEp0InISR中发送剩余的数据包或者数据即可.

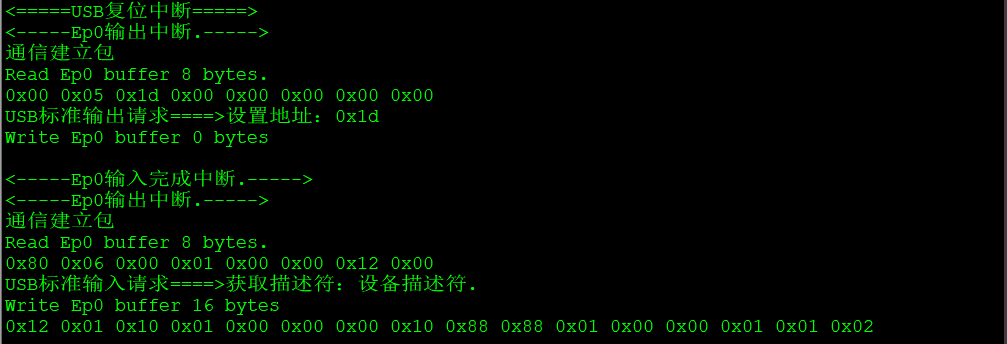


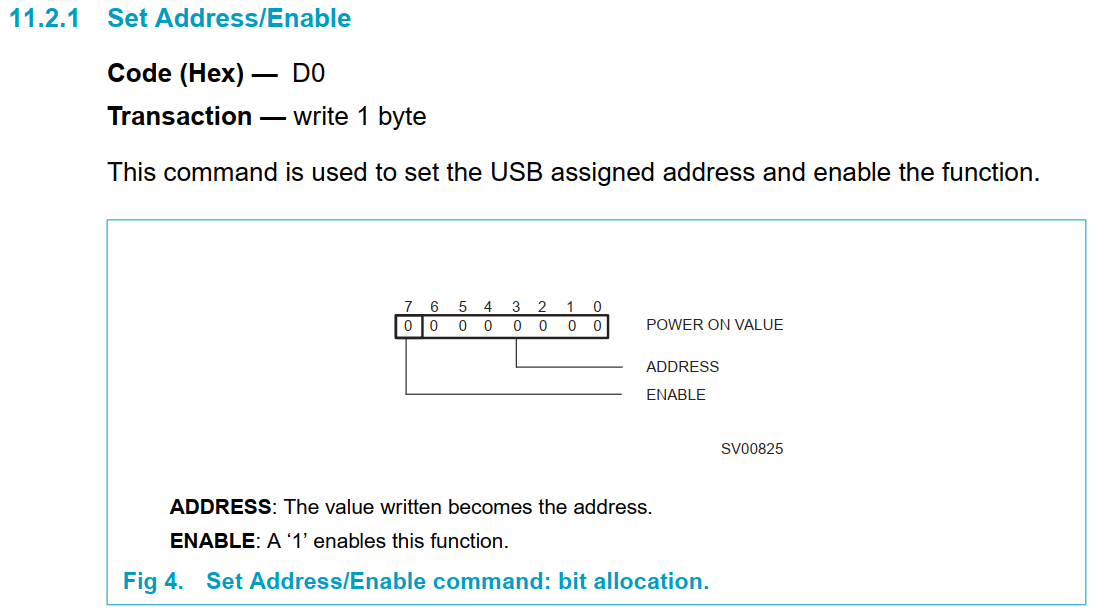




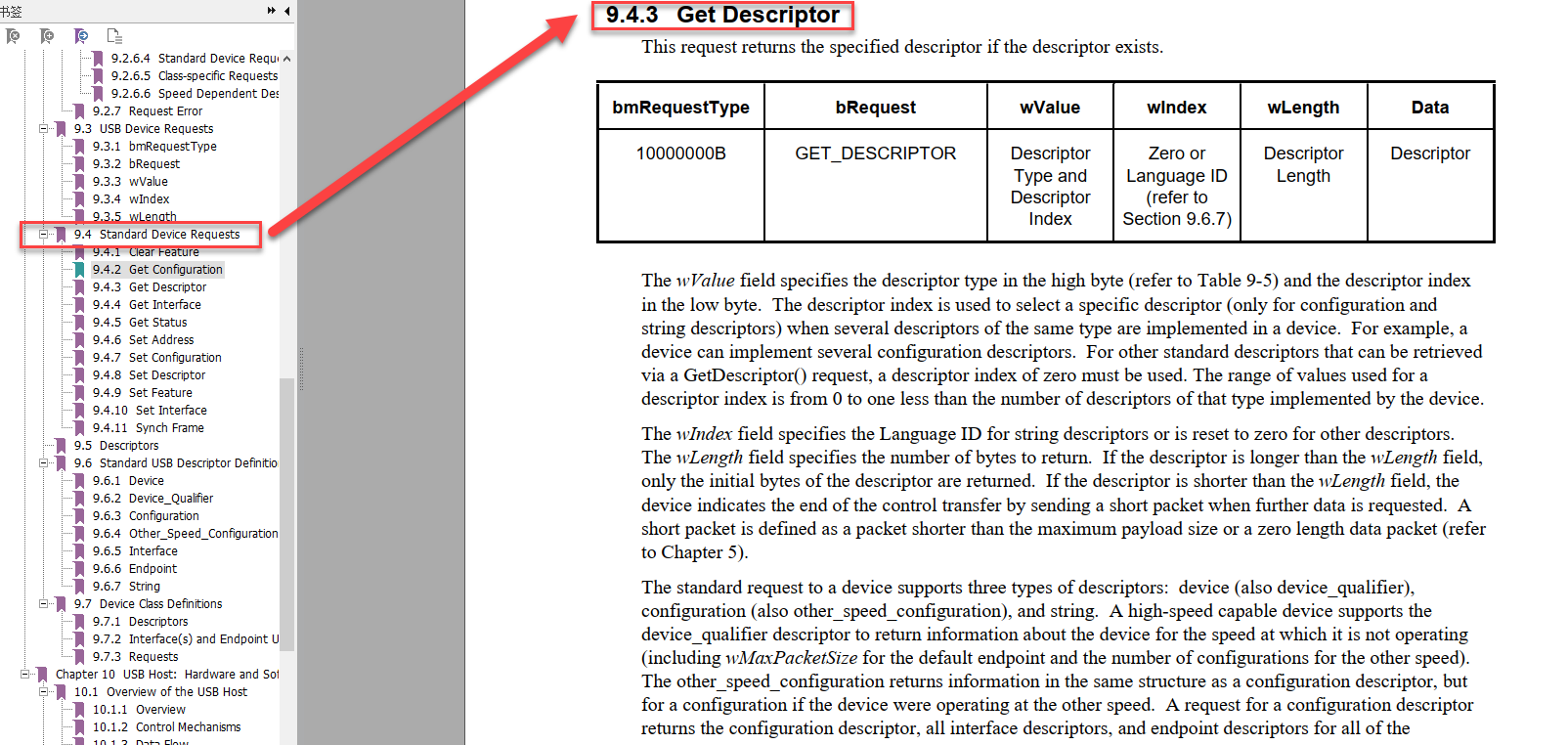


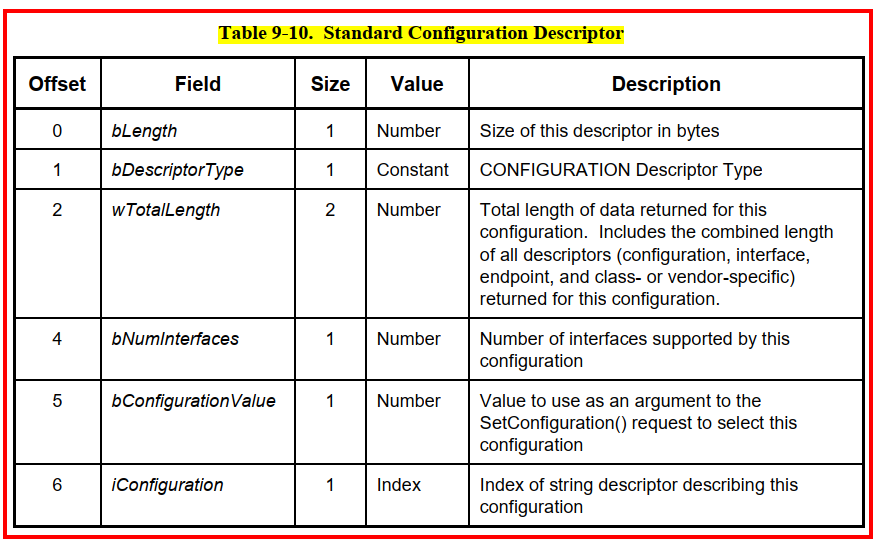
## 2.12)USB主机设置USB设备的地址，因为USB的地址为7位的，所以最多能同时挂128个USB设备，但是由于USB设备刚接入在没有分配USB地址时，使用的是USB地址0的端点0进行通信的，所以最大USB地址只能是127个，地址0需要预留给新接入的USB设备.USB主机设置USB设备地址的数据包中会带有USB主机分配给USB设备的USB地址. 在接收到设置地址包之后，固件操作D12芯片将获取到的USB地址写入到USB设备中，以便下次使用这个地址通信，设置完地址后，USB主机会使用新设置的地址与USB设备进行通信，重新获取设备描述符.

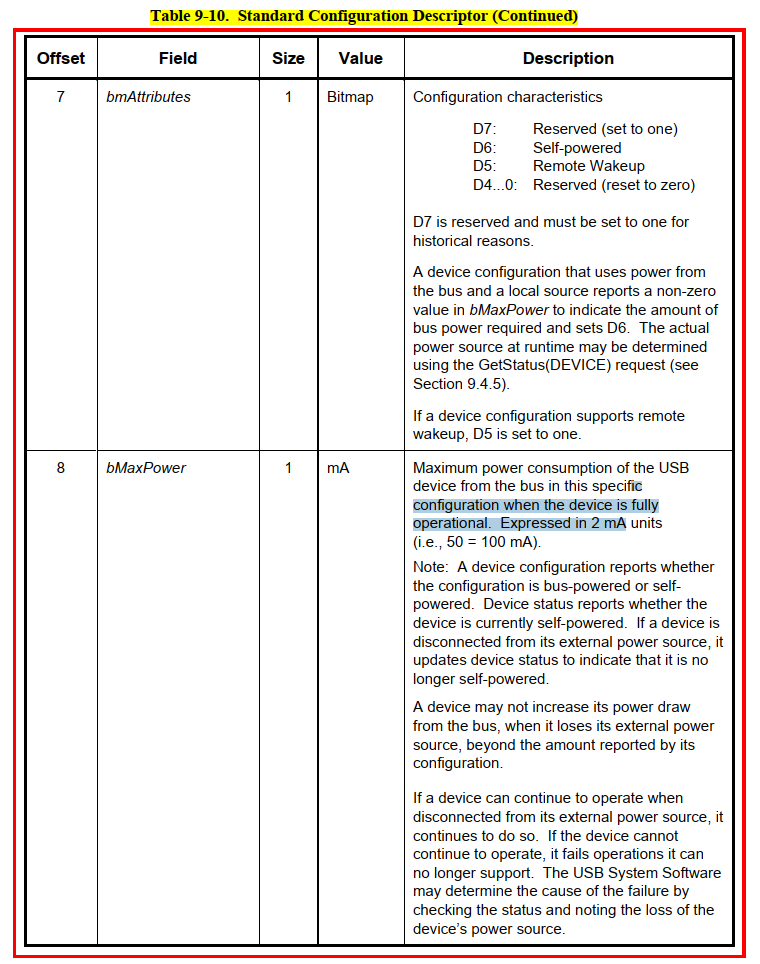




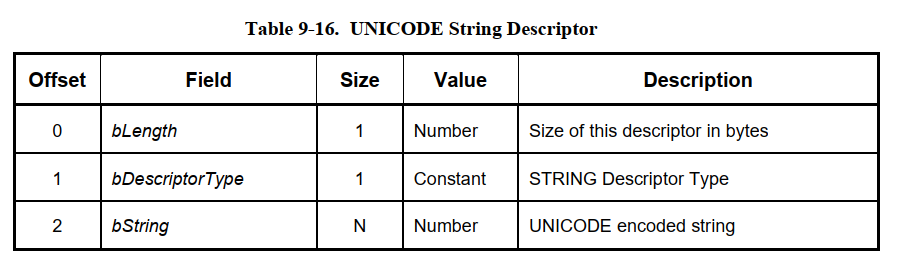
## 2.13)USB主机需要获取配置描述符.以下为USB的标准设备请求的格式及USB配置描述符的数据格式

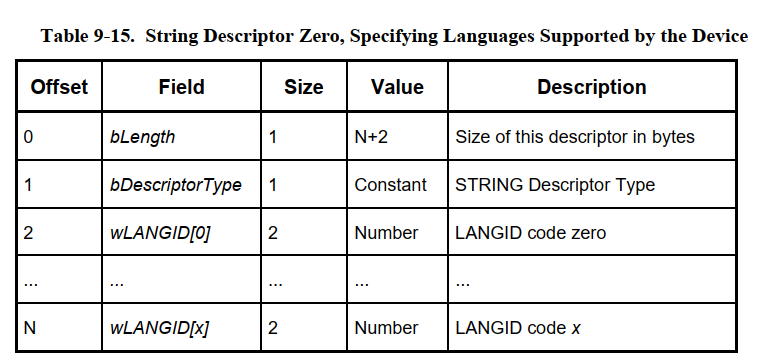






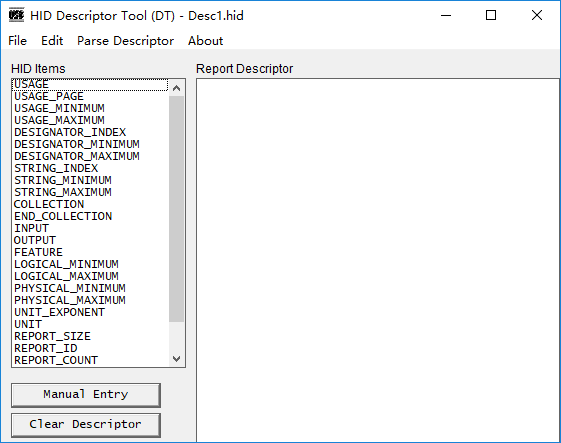
## 2.14)USB主机需要获取字符串描述符.字符串描述符包含厂商字符串描述符，产品字符串描述符及序列号字符串描述符，字符串描述符的index对应了设备描述符中的iManufacture, iProduct, iSerialNumber. 字符串描述符的字符内容编码都是unicode编码的，还有一种特殊的零字符描述符，指定设备支持的语言.



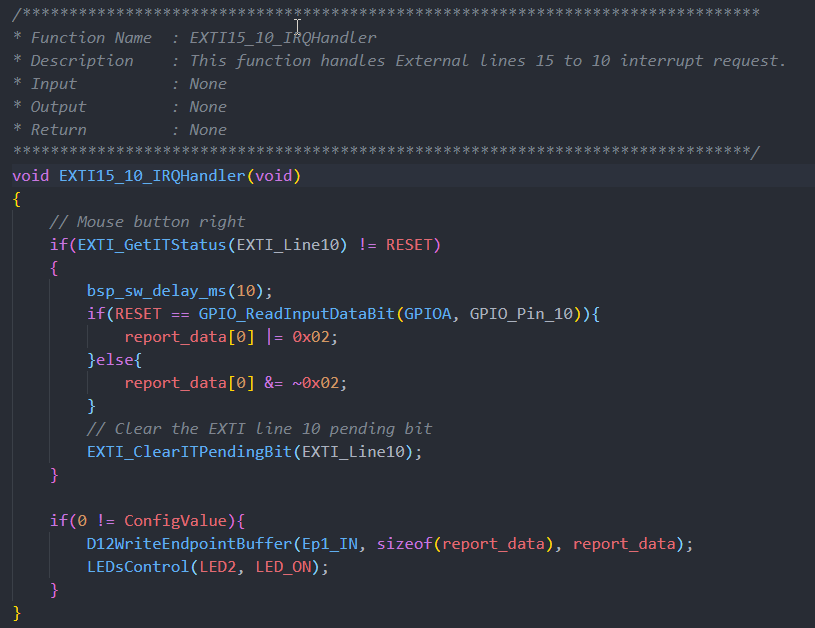


## 2.15)USB主机重新获取设备描述符，分两次获取配置描述符，完成后USB主机会对USB设备进行设置配置，设置配置的目的是通过非零的配置值来使能USB设备的非零端点，使能非零端点后就可以通过非零端点进行数据通信了.设置配置确认后USB设备需要给USB主机发送ACK来进行确认.

## 2.16)USB主机对USB设备进行USB类输出请求(因为USB设备需要作为USB HID设备使用)，输出请求USB设备设置空闲；然后USB主机获取报告描述符的请求，可以通过USB-IF的HID描述符工具来生成报告描述符. USB主机再次获取配置描述符.



## 2.17)USB设备的按键中断的方式来模拟鼠标的各种功能，例如：光标左右，上下移动及鼠标的按键等等.这里有一点需要注意，报告描述符中的鼠标光标及按键定义的顺序与后面USB通过端点返回的中断传输方式的报告的数据的顺序是一一对应的，且还有一个知识点是USB主机输出时，可以通过控制端点(Endpoint0)，但是HID鼠标设备回报数据时，必须通过之前进行设置配置的非零端点号进行数据回包(即USB输入).还有一个问题是鼠标按键按下回报数据时，按键按下和按键释放是两个不同的动作，会进行两次数据上报.



## 2.18)USB HID设备的报告描述符解析

**以下为一个鼠标的报告描述符为例来了解下描述符中的16进制数据的具体含义，一般的网文多是如上的报告描述**

**符，据说是用一个工具产生的，有一些还配上了中文的说明，但也多是描述性的，而不是拆分解释这些字符所代表的含义，比如，数组第一行的 0x05, 0x01,  // USAGE\_PAGE (Generic Desktop)**

**我们一眼就能看出双斜线后面的文字，是对前面数字的说明，即0x05,0x01所表达的是USAGE\_PAGE (Generic Desktop) 的含义但是，为何如此表达，则描述的不太清楚，对于熟悉的人而言，这自然不是问题，但对于新手，可就要费点功夫了。0x05, 0x01,代表USAGE\_PAGE(Generic Desktop)的功能，是由《device class definition for human interface device (HID)》第24页的Report Descriptors 规定的。**

**分两部分，0x05为一部分，表示前缀，0x01为数据部分。**

**0x05转换成二进制，就是0000  01001，按照HID类协议5.3 generic item format的定义，这个字节被分成3个部分，bit0~bit1代表的是这个前缀后面跟的数据长度，两位可以表示最大4字节的数据，即bsize;bit2~bit3代表的是这个前缀的类型，总共可以有三种类型：0=main，1=global，2=local，3=reserved；bit4~bit7代表前缀的tag，一般分为input(二进制的1000 00 nn,即bit4~bit7=1000,代表一个tag，bit2~bit3=00，代表main，bit0~bit1=nn，代表这个前缀后面还有nn所代表的数据)，output(二进制的 1001 00 nn)，feature(1011 00 nn)，collection(1010 00 nn)，end collection(1100 00 nn)，遵照这个原则，我们就可以解析0x05所表达的含义。**

**0x05转换为二进制就是0000 0101,其高4位全为0，表示的tag为usage page  tag(协议45页)，bit2~bit3=01，表示的是类型，从协议中可以知道，这是一个全局类型的item（协议36页），bit0~bit1=01，表示的是这个前缀后面跟着的数据长度为1字节，即0x05后面，有0x01作为这个前缀的数据部分，而0x01表示的是general desktop page(《universal serial bus HID usage table》第五页，目录)，因此，这两个数字合起来就是USAGE\_PAGE (Generic Desktop)的含义。**

**用一个比较形象的描述方式来描述：每一个报告描述符需要描述的是USB设备的结构以及实现的功能.**

1. **首先报告描述符需要一个Usage page来描述一个最外层的用途页类来表示USB设备的用途大类(Page ID)**
2. **对于上述的用途大类，需要进一步详述说明USB设备的具体用途Usage(Usage ID来申明这个用途)，如Keyboard**
3. **step1中的Usage ID是一个键盘用途，需要进一步对这个用途的类型进行说明分解以便主机找到驱动，而Usage type(用途类型)包含Collection(集合)，Control(控制)及Data(数据)这些基本的用途类型**
4. **总结：step1中在定义了Usage Page后，通过step2中的Usage ID对其进行说明(可以对应到HID协议中)，在step3在用途类型Usage type中对Usage page进行详细说明**

code char MouseReportDescriptor[63] = {      
      0x05, 0x01,                    // USAGE\_PAGE (Generic Desktop)     前缀字节(bTag, bType, bSize) 0x05, Usage page 0x01  
     0x09, 0x06,                   // USAGE (Keyboard)     前缀字节(bTag, bType, bSize) 0x09, Usage page 0x01下的Usage ID 0x06  
     0xa1, 0x01,                    // COLLECTION (Application)

    0x05, 0x07,                    //   USAGE\_PAGE (Keyboard)      
    0x19, 0xe0,                    //   USAGE\_MINIMUM (Keyboard LeftControl)      
    0x29, 0xe7,                    //   USAGE\_MAXIMUM (Keyboard Right GUI)      
    0x15, 0x00,                    //   LOGICAL\_MINIMUM (0)      
    0x25, 0x01,                    //   LOGICAL\_MAXIMUM (1)      
    0x75, 0x01,                    //   REPORT\_SIZE (1)      
    0x95, 0x08,                    //   REPORT\_COUNT (8)      
    0x81, 0x02,                    //   INPUT (Data,Var,Abs)    
    0x95, 0x01,                    //   REPORT\_COUNT (1)      
    0x75, 0x08,                    //   REPORT\_SIZE (8)      
    0x81, 0x03,                    //   INPUT (Cnst,Var,Abs)

    0x95, 0x05,                    //   REPORT\_COUNT (5)      
    0x75, 0x01,                    //   REPORT\_SIZE (1)      
    0x05, 0x08,                    //   USAGE\_PAGE (LEDs)      
    0x19, 0x01,                    //   USAGE\_MINIMUM (Num Lock)      
    0x29, 0x05,                    //   USAGE\_MAXIMUM (Kana)      
    0x91, 0x02,                    //   OUTPUT (Data,Var,Abs)

    0x95, 0x01,                    //   REPORT\_COUNT (1)      
    0x75, 0x03,                    //   REPORT\_SIZE (3)      
    0x91, 0x03,                    //   OUTPUT (Cnst,Var,Abs)

    0x95, 0x06,                    //   REPORT\_COUNT (6)      
    0x75, 0x08,                    //   REPORT\_SIZE (8)      
    0x15, 0x00,                    //   LOGICAL\_MINIMUM (0)      
    0x25, 0xFF,                    //   LOGICAL\_MAXIMUM (255)      
    0x05, 0x07,                    //   USAGE\_PAGE (Keyboard)      
    0x19, 0x00,                    //   USAGE\_MINIMUM (Reserved (no event indicated))      
    0x29, 0x65,                    //   USAGE\_MAXIMUM (Keyboard Application)      
    0x81, 0x00,                    //   INPUT (Data,Ary,Abs)

    0xc0                           // END\_COLLECTION

};

## 2.19)USB通信层次实质

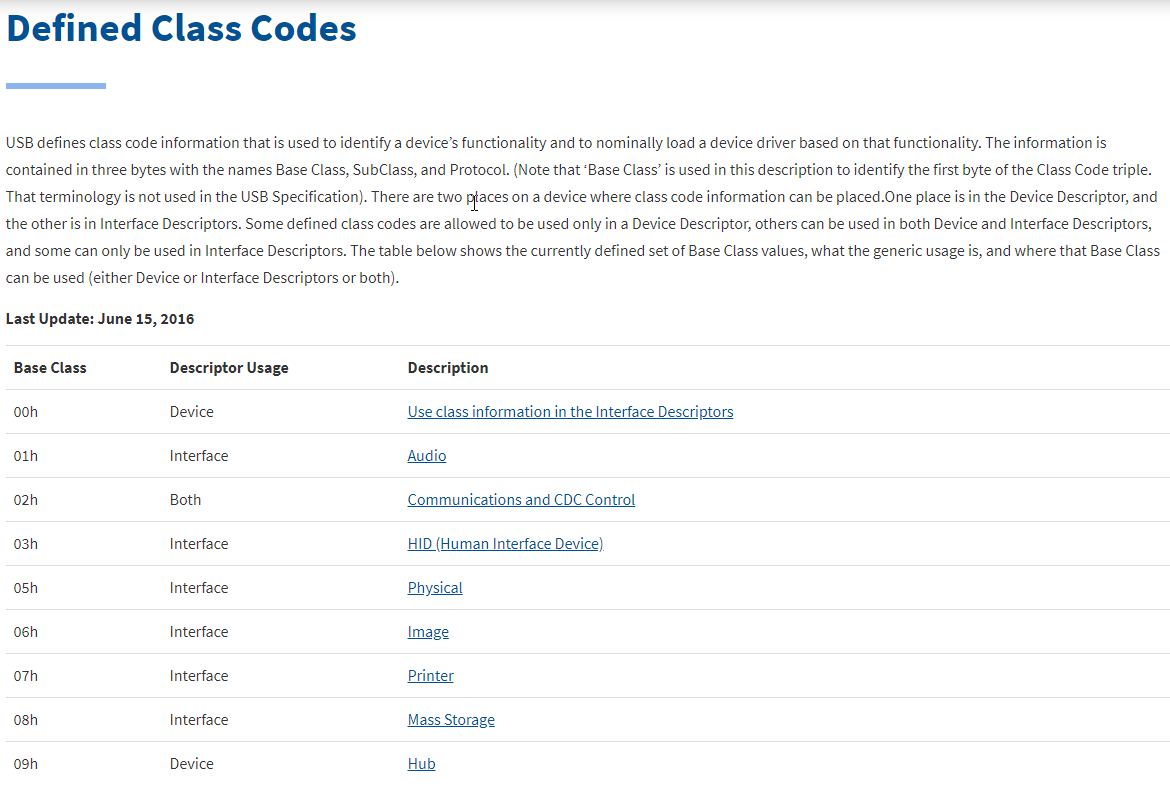
**USB的通信过程实质上是从两个层面来进行的(或者理解为实现的)，分别为USB底层驱动层以及设备与主机之间的通信数据交互层.**

1. **USB驱动层(协议层用于驱动加载)：USB主机能实现对USB设备的驱动加载完成.**
2. **USB数据通信层(基于协议基础上的数据层)：USB主机及USB设备之间进行数据的交互，已实现某些特定数据的传输，识别及显示.**

# CDC的实现

## 3.1) CDC的定义

CDC(Communication Device Class)即为通信设备类，其是通过USB协议中的一个设备类，设备类字段为0x02.设备类代码可以在USB-IF的官网上找到.具体网页地址参考：<https://www.usb.org/defined-class-codes>.



## 3.2) CDC底层驱动实现

## a) 基于USB Keyboard的固件，修改设备描述符中的设备类为0x02(参考上面的图片中CDC类)

## b) CDC类的子类及所用的协议必须指定为0x00

## c) 字符串描述符自定义

## d) 配置描述符：使用两个接口的，一个作为bulk的数据传输(端点2)，一个作为参数设置的(端点0)；使用端点2作为bulk数据传输的依据：由于USB的数据发送和接收都是以数据包的格式进行的，所以在发送和接收是就会存在封包及解包的过程，为了实现串口的全双工及避免数据丢失，需要用到USB端点的数据缓存区，而端点2的USB端点缓存区缓存空间最大，为64bytes且为双缓存.

## e) CDC接口描述符(USB的接口需要一个数据类接口依附，例如HID设备中的接口描述符需要HID描述符一样)，要实现VCP功能，需要在接口描述符中指定使用CDC类,ACM(Abstract Control Model)子类及Common AT Commands协议.

## f) 类特殊接口描述符(功能描述符)

## 3.3) 类请求实现

## a)在上面的部分实现后，USB host识别除了USB device作为CDC的虚拟串口.

## b) USB host在成功安装USB device的CDC驱动后会发出设置配置的输出请求，然后是USB类输入请求，有没当前FW没有实现所以出现timeout,然后USB host又发送了USB类输入请求，同样是由于没有实现而timeout

## c)基于USB host的这些请求实现相应的应答.了解USB类输入及输出请求中的数据包所代表的意思，参考USB-IF的USB CDC协议的page30(0x21: GET\_LINE\_CODING, 0x22:SET\_CONTROL\_LINE\_STATE),其中GET\_LINE\_CODING为设置CDC串口通信的参数(波特率，停止位，校验及数据位数)，SET\_CONTROL\_LINE\_STATE主要用于串口的流控制的，没有用到.

