USB 项目技术报告

2002 年 12 月 北航 frank

导读

本文介绍了在基于 ARM7 的嵌入式系统上开发 USB 接口的详细内容。项目使用 ARM7（MCU 处理器）和 PDIUSBD12（USB 芯片）完成了 U 盘的开发。主要内容涉及 USB 接口的 U 盘开发过程中所涉及的技术内容；重点介绍项目调试方法与步骤；并附上了自己在开发此项目过程中的一些心得与体会。

文中所涉及内容有的来自互联网上的资料、所用芯片技术资料、有关协议的文档以及他人总结的一些东西，在此不一一列举出处，希望本文能给后续做此类开发的技术人员提供一个捷径和指引，起到抛砖引玉的作用。不足与欠妥之处，肯请指正 wxjbuaa@sohu.com。

一、项目背景

1.1 PC 接口简介

PC 中的接口有两类：串行接口和并行接口。计算机内部总线，如 CPU 与存储器之间匀采用并行接口，这样速度快；但外设却以串行接口比较占优势。

传统的打印机接口为并行接口，它实际上叫 Cetronix 标准，这种接口现在已经没有发展了。SCSI 标准的全名是小型设备通用接口标准，其传输速率为 10M，早期的扫描仪一般使用此接口，硬盘与主机的联接也使用这种接口。

串行接口出现最早，使用最广的 RS232 接口，但其速度太慢，现在已经逐渐淘汰。USB 接口和 IEEE1394 接口是两种速度比较高的串行接口，还有局域网中的以太网接口，它们具有较广阔的发展前景和应用潜力。

USB 适用于低档外设与主机之间的高速数据传输，USB1.1 可以达到 1.5Mbps 或 12Mbps 的传输率，而 1394 更是可达 100／200／400Mbps。USB2.0 将速度定位在 480Mbps，而 IEEE 1394 也推出了 1394b 1.3.1 版草案，速度从 800Mbps 起步，最高可达 3.2Gbps。但这两种版本目前都还没有成熟。

局局域中用得最多的是以太网接口，速度可达 100Mbps，当使用光纤传输时，速度可达 1000Mbps。

1.2 USB 接口分析

通用串行总线(Universal Serial Bus USB)，是一种快速、灵活的总线接口。与其它通信接口比较，USB 接口的最大特点是易于使用，这也是 USB 的主要设计目标。作为一种高速总线接口，USB 适用于多种设备，比如数码相机、MP3 播放机、高速数据采集设备等。易于使用还表现在 USB 接口支持热插拔，并且所有的配置过程都由系统自动完成，无需用户

干预。

USB 接口支持 1.5Mb/s(低速)、12Mb/s(全速)和高达 480Mb/s(USB 2.0 规范)的数据传输速率，扣除用于总线状态、控制和错误监测等的数据传输，USB 的最大理论传输速率仍达 1.2Mb/s 或 9.6Mb/s，远高于一般的串行总线接口。

USB 接口芯片价格低廉，一个支持 USB 1.1 规范的 USB 接口芯片价格大多在人民币（2002 年）20～40 元之间，这也大大促进 USB 设备的开发与应用。

1.3 USB 器件的选择

在进行一个 USB 设备开发之前，首先要根据具体使用要求选择合适的 USB 控制器。目前，市场上供应的 USB 控制器主要有两种：带 USB 接口的单片机（MCU）或纯粹的 USB 接口芯片。

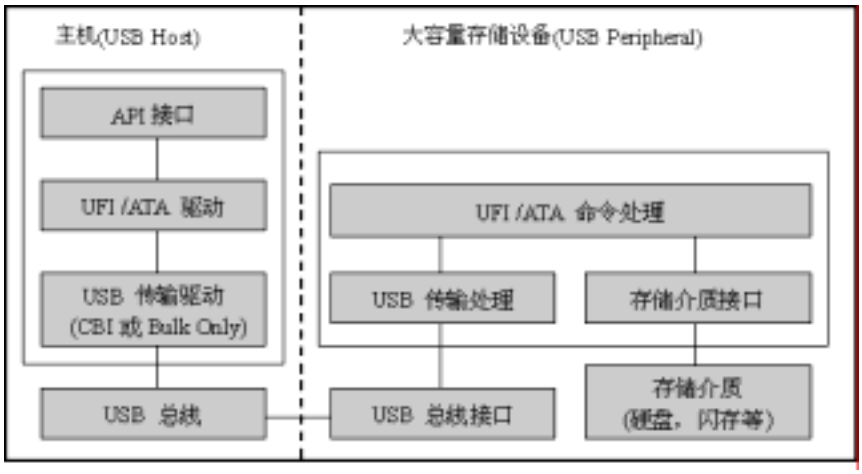
带 USB 接口的单片机从应用上又可以分成两类，一类是从底层设计专用于 USB 控制的单片机；另一类是增加了 USB 接口的普通单片机，如 Cypress 公司的 EZ－USB(基于 8051)，选择这类 USB 控制器的最大好处在于开发者对系统结构和指令集非常熟悉，开发工具简单，但对于简单或低成本系统。但价格因素也是在实际选择过程中需要考虑的因素。

纯粹的 USB 接口芯片仅处理 USB 通信，必须有一个外部微处理器来进行协议处理和数据交换。典型产品有 Philips 公司的 PDIUSBD11（I2C 接口）、PDIUSBD12(并行接口)，NS 公司的 USBN9603/9604(并行接口)，NetChip 公司的 NET2888 等。USB 接口芯片的主要特点是价格便宜、接口方便、可靠性高，尤其适合于产品的改型设计（硬件上仅需对并行总线和中断进行改动，软件则需要增加微处理器的 USB 中断处理和数据交换程序、PC 机的 USB 接口通信程序，无需对原有产品系统结构作很大的改动）。

1.4 Mass Storage 协议与 FAT16

USB 组织定义了海量存储设备类（Mass Storage Class）的规范，这个类规范包括四个独立的子类规范，即：1. USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport 2. USB Mass Storage Class Bulk-Only Transport 3. USB Mass Storage Class ATA Command Block 4. USB Mass Storage Class UFI Command Specification。前两个子规范定义了数据/命令/状态在 USB 上的传输方法。Bulk- Only 传输规范仅仅使用 Bulk 端点传送数据/命令/状态，CBI 传输规范则使用 Control/Bulk/Interrupt 三种类型的端点进行数据/命令/状态传送。后两个子规范则定义了存储介质的操作命令。ATA 命令规范用于硬盘，UFI 命令规范是针对 USB 移动存储。

Microsoft Windows 中提供对 Mass Storage 协议的支持，因此 USB 移动设备只需要遵循 Mass Storage 协议来组织数据和处理命令，即可实现与 PC 机交换数据。而 Flash 的存储单元组织形式采用 FAT16 文件系统，这样，就可以直接在 Windows 的浏览器中通过可移动磁盘来交换数据了，Windows 负责对 FAT16 文件系统的管理，USB 设备不需要干预 FAT16 文件系统操作的具体细节。



USB 移动存储结构

二、项目主要组成部分

2.1 USB 的端点

端点是 USB 中一个独特的概念，它是一个可以与 USB Host 交换数据的硬件单元。USB Host 与 USB 设备之间都是通过端点来传输数据的，端点是桥梁和纽带，不同的端点其传输数据的能力不同，适于不同的应用场合。

PDIUSBD12 的端点适用于不同类型的设备，例如图像打印机、海量存储器和通信设备。端点可通过 Set Mode 命令配置为 4 种不同的模式，分别为：

模式 0 Non-ISO 模式：非同步传输

模式 1 ISO-OUT 模式：同步输出传输

模式 2 ISO-IN 模式：同步输入传输

模式 3 ISO-IO 模式：同步输入输出传输

这几种模式下可得到的端点情况如下表：

模式 0(非同步模式)

端点	端点索引	传输类型	端点类型	方向	最大信息包(字节)
0	0	控制输出	默认	输出	16
	1	控制输入		输入	16
1	2	普通输出	普通	输出	16
	3	普通输入	普通	输入	16
2	4	普通输出	普通	输出	64[4]
	5	普通输入	普通	输入	64[4]

模式 1(同步输出模式)

端点	端点索引	传输类型	端点	类型方向	最大信息包（字节）
0	0	控制输出	默认	输出	16
	1	控制输入		输入	16
1	2	普通输出	普通	输出	16
	3	普通输入	普通	输入	16

2	4	同步输出	同步	输出	128[4]

模式 2(同步输入模式)

端点	端点索引	传输类型	端点	类型方向	最大信息包(字节)
0	0	控制输出	默认	输出	16
	1	控制输入		输入	16
1	2	普通输出	普通	输出	16
	3	普通输入	普通	输入	16
2	4	同步输入	同步	输入	128[4]

模式 3(同步输入/输出模式)

端点	端点索引	传输类型	端点	类型方向	最大信息包(字节)
0	0	控制输出	默认	输出	16
	1	控制输入		输入	16
1	2	普通输出	普通	输出	16
	3	普通输入	普通	输入	16
2	4	同步输出	同步	输出	64[4]
	5	同步输入	同步	输入	64[4]

端点 2 叫做主端点，它在有些方面是比较特别的，它是进行吞吐大数据的主要端点，同样地它执行主机的特性以减轻传输大数据的任务：

1 双缓冲，允许 USB 与本地 CPU 之间的并行读写操作，这样就增加了数据的吞吐量。缓冲区切换是自动处理的。这导致了透明的缓冲区操作。

2 支持 DMA 直接存储器访问操作，可以和对其它端点的正常 I/O 操作交叉进行。

3 缓冲区的最大信息包长度较其它端点大。

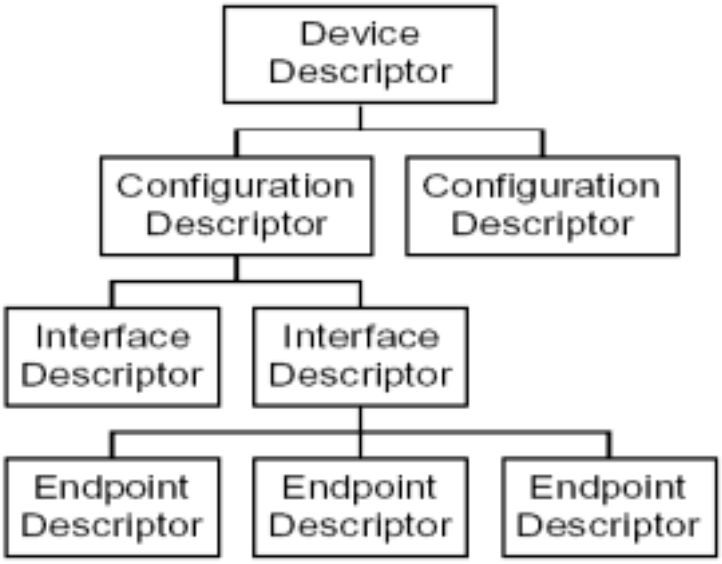
4 可配置为同步传输或非同步批量和中断传输

2.2 Mass Storage 协议

USB 协议能够在启动或是当设备插入系统时对设备进行备置，这就是 USB 设备为什么可以执插拨的原因。USB 设备被分成以下几类：显示器（Monitors）、通讯设备（Communication devices）、音频设备（Audio）、人机输入（Human input）、海量存储（Mass storage）。

特定类（class）的设备又可划分成子类（subclass），划分子类的后软件就可以搜索总线并选择所有它可以支持的设备。每个设备可以有一个或多个配置（Configuration），配置用于定义设备的功能。如果某个设备有几种不同的功能，则每个功能都需要一个配置。配置（*configuration*）是接口（*interface*）的集合。接口指定设备中的哪些硬件与 USB 交换数据。每一个与 USB 交换数据的硬件就叫做一个端点（*endpoint*）。因此，接口是端点的集合。USB 的设备类别定义（USB Device Class Definitions）定义特定类或子类中的设备需要提供的缺省配置、接口和端点。

描述符（*descriptor*）描述设备、配置、接口或端点的一般信息，下图为 USB 描述符的层次结构。



USB 描述符的层次结构

USB（Host）唯一通过描述符了解设备的有关信息，根据这些信息，建立起通信，在这些描述符中，规定了设备所使用的协议、端点情况等。因此，正确地提供描述符，是 USB 设备正常工作的先决条件。

USB 海量存储设备（USB Mass Storage Class）包括 General Mass Storage Subclass、CD－ROM、Tape、Solid State。Mass Storage Class 只需要支持一个接口，即数据（Data）接口，选择缺省配置时此接口即被激活。数据接口允许与设备之间进行数据传输，它提供三个端点：Bulk Input 端点、Bulk Output 端点和中断端点。

通用海量存储设备（General Mass Storage Device）是随机存取、基于块／扇区存储的设备。它只能存储和取回来自 CPU 的数据。这种设备的接口遵循 SCSI－2 标准的直接存取存储设备（Direct Access Storage Device）协议。USB 设置上的介质使用与 SCSI－2 设备相同的逻辑块（logical blocks）方式寻址。

下面介绍基于 Bulk Only（批量传输）模式的 Mass Storage 设备的描述符：

每个 USB 设备都必须有一个设备描述符。Mass Stroage 设备的设备类型和子类代码均在接口描述符中设置，这里置 0。其中 *iSerialNumber* 可为零，即不指定 Serial Number。

Offset	Field	Size	Value	Description
0	*bLength*	Byte	12h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	01h	DEVICE descriptor type.
2	*bcdUSB*	Word	????h	*USB Specification* Release Number in Binary-Coded Decimal (i.e. 2.10 = 210h). This field identifies the release of the *USB Specification* with which the device and is descriptors are compliant.
4	*bDeviceClass*	Byte	00h	**Class is specified in the interface descriptor.**
5	*bDeviceSubClass*	Byte	00h	**Subclass is specified in the interface descriptor.**
6	*bDeviceProtocol*	Byte	00h	**Protocol is specified in the interface descriptor.**
7	*bMaxPacketSize0*	Byte	??h	Maximum packet size for endpoint zero. (only 8, 16, 32, or 64 are valid **(08h, 10h, 20h, 40h)**).
8	*idVendor*	Word	????h	Vendor ID (assigned by the USB-**IF**).
10	*idProduct*	Word	????h	Product ID (assigned by the manufacturer).
12	*bcdDevice*	Word	????h	Device release number in binary-coded decimal.
14	*iManufacturer*	Byte	??h	Index of string descriptor describing the manufacturer.
15	*iProduct*	Byte	??h	Index of string descriptor describing this product.
16	*iSerialNumber*	Byte	??h	Index of string descriptor describing the device's serial number. **(Details in 4.1.1 below)**
17	*bNumConfigurations*	Byte	??h	Number of possible configurations.

设备（Device）描述符

配置描述符如下图，第 4 字节处的接口数应为 1。

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	02h	CONFIGURATION Descriptor Type.
2	*wTotalLength*	Word	????h	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	*bNumInterfaces*	Byte	??h	Number of interfaces supported by this configuration. **The device shall support at least the Bulk-Only Data Interface.**
5	*bConfigurationValue*	Byte	??h	Value to use as an argument to the *SetConfiguration()* request to select this configuration.
6	*iConfiguration*	Byte	??h	Index of string descriptor describing this configuration.
7	*bmAttributes*	Byte	?0h	Configuration characteristics: **Bit** **Description** 7 Reserved (set to one) 6 Self-powered 5 Remote Wakeup 4..0 Reserved (reset to zero) **Bit** 7 is reserved and must be set to one for historical reasons. **For a full description of this *bmAttributes* bitmap, see the *USB 1.1 Specification*.**
8	*MaxPower*	Byte	??h	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2mA units (i.e. 50 = 100mA)

配置（Configuration）描述符

设置应至少支持一个接口，这里为Bulk－Only Data接口，此接口使用三个端点：控制端点（默认）、Bulk－In和Bulk－Out。其中*bInterfaceSubClass*指定所使用的工业标准命令块，*bInterfaceProtocol*为所使用的传输协议，其定义见后。

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	04h	INTERFACE Descriptor Type.
2	*bInterfaceNumber*	Byte	0?h	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	Byte	??h	Value used to select alternate setting for the interface identified in the prior field.
4	*bNumEndpoints*	Byte	??h	Number of endpoints used by this interface (excluding endpoint zero). **This value shall be at least 2.**
5	*bInterfaceClass*	Byte	08h	**MASS STORAGE Class.**
6	*bInterfaceSubClass*	Byte	0?h	Subclass code (assigned by the USB-**IF**). **Indicates which industry standard command block definition to use. Does not specify a type of storage device such as a floppy disk or CD-ROM drive.** **(See *USB Mass Storage Overview Specification*)**
7	*bInterfaceProtocol*	Byte	50h	**BULK-ONLY TRANSPORT.** **(See *USB Mass Storage Overview Specification*)**
8	*iInterface*	Byte	??h	Index to string descriptor describing this interface.

接口（Interface）描述符

由于控制端点为每个设备都使用的缺省端点，因此不需要定义，只需定义 Bulk－In 和 Bulk－Out 两个端点，其端点描述符如下：

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT Descriptor Type.
2	*bEndpointAddress*	Byte	8?h	The address of this endpoint on the USB device. The address is encoded as follows. **Bit** **Description** 3..0 The endpoint number 6..4 Reserved, set to 0 7 1 = In
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

Bulk－In 端点描述符

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	Byte	0?h	The address of this endpoint on the USB device. This address is encoded as follows: **Bit** **Description** 3..0 Endpoint number 6..4 Reserved, set to 0 7 0 = Out
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, or 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

Bulk－Out 端点描述符

SubClass Code	Command Block Specification	Comment
01h	Reduced Block Commands (RBC) T10 Project 1240-D	Typically, a Flash device uses RBC command blocks. However, any Mass Storage device can use RBC command blocks.
02h	SFF-8020i, MMC-2 (ATAPI)	Typically, a C/DVD device uses SFF-8020i or MMC-2 command blocks for its Mass Storage interface.
03h	QIC-157	Typically, a tape device uses QIC-157 command blocks.
04h	UFI	Typically a floppy disk drive (FDD) device
05h	SFF-8070i	Typically, a floppy disk drive (FDD) device uses SFF-8070i command blocks. However, an FDD device can be in another subclass (for example, RBC) and other types of storage devices can belong to the SFF-8070i subclass.
06h	SCSI transparent command set	
07h – FFh	Reserved for future use.	

bInterfaceSubClass 处的工业标准命令块代码

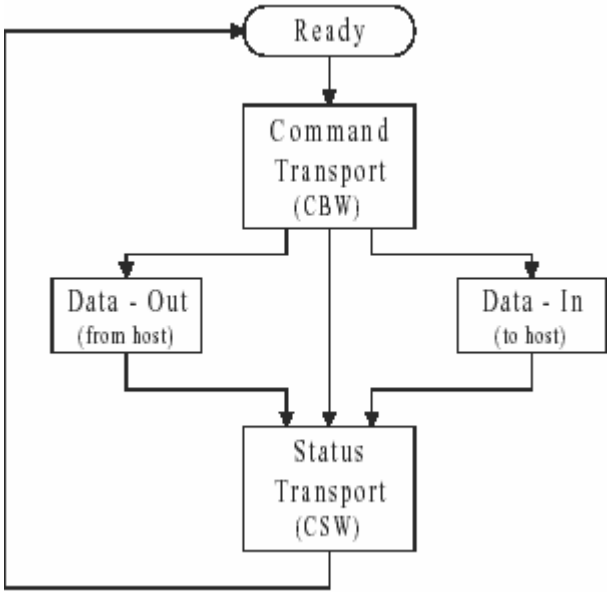
bInterfaceProtocol	Protocol Implementation	Comment
00h	Control/Bulk/Interrupt protocol (with command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
01h	Control/Bulk/Interrupt protocol (with no command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
50h	Bulk-Only Transport	USB Mass Storage Class Bulk-Only Transport
02h – 4Fh	Reserved	
51h – FFh	Reserved	

Mass Storage 传输协议

2.3 Bulk－Only 传输协议

设备插入到 USB 后，USB 即对设备进行搜索，并要求设备提供相应的描述符。在 USB Host 得到上述描述符后，即完成了设备的配置，识别出为 Bulk－Only 的 Mass Storage 设备，然后即进入 Bulk－Only 传输方式。在此方式下，USB 与设备间的所有数据均通过 Bulk－In 和 Bulk－Out 来进行传输，不再通过控制端点传输任何数据。

在这种传输方式下，有三种类型的数据在 USB 和设备之间传送，CBW、CSW 和普通数据。CBW（Command Block Wrapper，即命令块包）是从 USB Host 发送到设备的命令，命令格式遵从接口中的 bInterfaceSubClass 所指定的命令块，这里为 SCSI 传输命令集。USB 设备需要将 SCSI 命令从 CBW 中提取出来，执行相应的命令，完成以后，向 Host 发出反映当前命令执行状态的 CSW（Command Status Wrapper），Host 根据 CSW 来决定是否继续发送下一个 CBW 或是数据。Host 要求 USB 设备执行的命令可能为发送数据，则此时需要将特定数据传送出去，完毕后发出 CSW，以使 Host 进行下一步的操作。USB 设备所执行的操作可用下图描述：



USB Host 按照下面的格式向设备端发送 CBW，

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

CBW

其中 dCBWSignature 的值为 43425355h（LSB），表示当前发送的是一个 CBW；dCBWTag 的内容需要原样作为 dCSWTag 再发送给 Host；dCBWDataTransferLength 为本次 CBW 需要传输的数据，bmCBWFlags 反映数据传输的方向，0 表示来自 Host，1 表示发至 Host；bCBWLUN 一般为零，但当设备有多个逻辑单元时，用此位指定本次命令是发给谁的；bCBWCBLength 为本次命令字的长度；CBWCB 即为真正的传输命令集的命令。

得到一个 CBW 后，解析出 CBWCD 中所代表的命令，然后按照 SCSI 命令集中的定义来执行相应的操作，或是需要接收下一个 Bulk－Out 发来的数据，或是需要向 Host 传送数据，完成以后需要向 USB Host 发送 CSW，反映命令执行的状态。USB 也是通过此来了解设备的工作情况的。下面是 CSW 的格式和定义：

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

CSW

dCSWSignature 的内容为 53425355h，dCSWTag 即为 dCBWTag 的内容，dCSWDataResidue 还需要传送的数据，此数据根据 dCBWDataTransferLength－本次已经传送的数据得到。Host 端根据此值决定下一次 CBW 的内容，如果没有完成则继续；如果命令正确执行，bCSWStatus 返回 0 即可。按这个规则组装好 CSW 后，通过 Bulk－In 端点将其发出即可。

2.4 SCSI 指令集

Bulk-Only 的 CBW 中的 CBWCB 中的内容即为如下格式的命令块描述符（Command Block Descriptor）。SCSI-2 有三种字长的命令，6 位、10 位和 12 位，Microsoft Windows 环境下支持 12 位字长的命令。

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code							
1	Logical unit number			Reserved				
2	(MSB)							
3	Logical block address (if required)							
4								
5								(LSB)
6	(MSB)							
7	Transfer length (if required) Parameter list length (if required) Allocation length (if required)							
8								
9								(LSB)
10	Reserved							
11	Control							

12 位字长的 SCSI 命令

Operation Code 是操作代码，表示特定的命令。高 3 位为 Group Code，共有 8 种组合，即 8 个组，低 5 五位为 Command Code，可以有 32 种命令。

Logicol unit Number 是为了兼容 SCSI－1 而设的，此处可以不必关心。

Logical block address 为高位在前，低位在后的逻辑块地址，即扇区地址。第 2 位为高位，第 3、4、5 依次为低位。

Transfer length 为需要从逻辑块地址处开始传输的扇区数（比如在 Write 命令中）；Parameter list length 为需要传输的数据长度（比如在 Mode Sense 命令中）；Allocation length 为初始程序为返回数据所分配的最大字节数，此值可以为零，表示不需要传送数据。

SCSI 指令集的 Direct Accesss 类型存储介质的传输命令有许多，所幸运的是 Mass Storage 协议只用到了其中的一些。下面黑体部分即为需要 USB 设备作出响应的请求，一般是要求向 Host 发送一些有关设备的数据：

SCSI – Direct Access	Opcode
INQUIRY	12h
TEST UNIT READY	00h
FORMAT UNIT	04h
LOCK-UNLOCK CACHE	36h
MODE SELECT(6)	15h
MODE SENSE(6)	1Ah
PRE-FETCH	34h
PREVENT-ALLOW MEDIUM REMOVAL	1Eh
READ(6)	08h
READ(10)	28h
READ CAPACITY	25h
READ DEFECT DATA	37h
READ LONG	3Eh
WRITE LONG	3Fh
REASSIGN BLOCKS	07h
RECEIVE DIAGNOSTIC RESULTS	1Ch
SEND DIAGNOSTIC	1Dh
RELEASE	17h
RESERVE	16h
REZERO UNIT	01h
SEEK(10)	2Bh
SET LIMITS	33h
START STOP UNIT	1Bh
SYNCHRONIZE CACHE	35h
VERIFY	2Fh
WRITE(6)	0Ah
WRITE(10)	2Ah
WRITE AND VERIFY	2Eh
WRITE SAME	41h

对于不同的命令，其命令块描述符略有不同，其要求的返回内容也有所不同，根据相

应的文档，可以对每种请求作出适当的回应。比如，下面是 INQUIRY 请求的命令块描述符和其返回内容的数据格式：

1）INQUIRY

命令块描述符：

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (12h)							
1	Logical Unit Number			Reserved				EVPD (0)
2	Page Code							
3	Reserved							
4	Allocation Length							
5	Reserved							
6	Reserved							
7	Reserved							
8	Reserved							
9	Reserved							
10	Reserved							
11	Reserved							

返回数据格式：查看数据

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved			Peripheral Device Type				
1	RMB	Reserved						
2	ISO Version		ECMA Version			ANSI Version (00h)		
3	Reserved				Response Data Format			
4	Additional Length (31)							
5 – 7	Reserved							
8 – 15	Vendor Information							
16 – 31	Product Identification							
32 – 35	Product Revision Level n.nn							

Host 会依次发出 INQUIRY、Read Capacity、UFI Mode Sense 请求，如果上述请求的返回结果都正确，则 Host 会发出 READ 命令，读取文件系统 0 簇 0 扇区的 MBR 数据，进入文件系统识别阶段。

对于 **PREVENT-ALLOW MEDIUM REMOVAL** 和 **TEST UNIT READY** 命令，只需直接返回 CSW 即可，对于后者，由于 Flash 盘总是处于 READY 状态，故可直接返回 CSW。

2.4 FAT16 文件系统

2.4.1 FAT 文件系统结构

一个 FAT（FAT12／FAT16／FAT32）文件系统卷（卷可以理解为是一张软盘、一个硬盘或是一个 Flash 电子盘）由四个部分组成：

1）保留区（Reserved Region）

分区的保留区（Reserved Region）中的第一个扇区必须是 BPB(BIOS Parameter Block)，

此扇区有时也称作“引导扇区”、“保留扇区”或是“零扇区”，因为它含有对文件系统进行识别的关键信息，因此十分重要。下表是此扇区的结构：

名称	偏	长	典型值	说明
BS_jmpBoot	0	3	EB 03 90	jmpBoot[0] = 0xEB, jmpBoot[1] = 0x??, jmpBoot[2] = 0x90 和 jmpBoot[0] = 0xE9, jmpBoot[1] = 0x??, jmpBoot[2] = 0x?? 0x??表示此处可以为任意字节，任一种选择都可以。
BS_OEMName	3	8		Microsoft 的操作系统并不关心此域，但一些 FAT 驱动比较在乎，推荐使用“MSWIN4.1”字符串，你完全可以用你自己的名字，无所谓。
BPB_BytsPerSec	11	2		每扇区字节数。只能是：512，1024，2048 或 4096。对于一些老的系统，只能使用 512，Dos 下均为 512，建议采用 512。三星的 Flash 的每个 Page 的大小为 512，这在做电子盘的时候也比较方便。
BPB_SecPerClus	13	1		每簇扇区数。此值不能为零，且必须是 2 的整数次方。如 1，2，4，8，16，32，64 及 128。但是此值不要使 BPB_BytsPerSec*BPB_SecPerClus>32K (32 * 1024)。即每簇不要超过 32K 字节。
BPB_RsvdSecCnt	14	2		保留区域中的保留扇区数。保留扇区从第 1 个扇区开始，对于 FAT12 和 FAT16，这时必须填 1。对于 FAT32，此处为 32。
BPB_NumFATs	16	1		此卷中 FAT 结构的份数。必须为 2
BPB_RootEntCnt	17	2		对于 FAT12 和 FAT16 卷，此域中为根目录项数（每个项长度为 32 字节），对于 FAT32，此域为零。此值乘上 32 后必须为 BPB_BytsPerSec 的整数倍。为了达到最好的兼容性，FAT16 卷应使用 512。
BPB_TotSec16	19	2		此域为存储卷上的扇区总数。包括 FAT 表的四个区域的所有扇区数。此域可以为零，当为零时，BPB_TotSec32 必须非零。对于 FAT32，此域必须为零。对于 FAT12 或 FAT16，此域为扇区总数，如果总扇区数小于 0x10000，则 BPB_TotSec32 为零。
BPB_Media	21	1		对于固定存储介质，使用 0xF8，对于可移动存储介质，使用 F0。0xF0，0xF8，0xF9，0xFA，0xFB，0xFC，0xFD，0xFE，和 0xFF 都是合法的值，但是在 FAT 表中的 FAT[0]的低位必须与之一致。
BPB_FATSz16	22	2		FAT12/FAT16 每个分区表所占的扇区数。对于 FAT32，此域为 0， 在 BPB_FATSz32 中包含 FAT32 的分区数。
BPB_SecPerTrk	24	2		每道扇区数。对于非磁头、柱面、扇区结构的介质，此域可为零
BPB_NumHeads	26	2		磁头数。对于非磁头、柱面、扇区结构的介质，此域可为零
BPB_HiddSec	28	4		此 FAT 表所在的分区前面的隐藏扇区数。对于非分区的介质，此值可为零，与操作系统有关。
BPB_TotSec32	32	4		对于 FAT32，此域非零；对于 FAT12/FAT16，如果扇区总数超过 0x10000，则此域为扇区总数。

名称	偏	长	典型值	说明
BS_DrvNum	36	1		操作系统有关参数，软盘使用 0x00，硬盘使用 0x80。
BS_Reserved1	37	1		保留（供 NT 使用）。必须为 0。
BS_BootSig	38	1		扩展引导标记(0x29)。此标记用来指示其后的三个域可用。
BS_VolID	39	4		卷的序列号。此域与 BS_VolLab 一起可支持对可移动磁盘的跟踪，用来判断是否是正确的磁盘。FAT 文件系统可据此判断是否将错误的软盘插到了软驱中。此域常用当前日期和时间来组成。
BS_VolLab	43	11		11 个字节长的卷标，此域需与要目录中的卷标一致
BS_FilSysType	54	8		“ FAT12 ”，“ FAT16 ”或“ FAT ”之一。此域仅仅是一个标志，操作系统并不关心它，也不用它来确实文件系统的类型。
Executable Code	62			如果是引导分区，它接过系统控制权后，可以执行一列引导操作
Executable Marker	510			55 AA

2）FAT 区

FAT 即 File Allocation Table，文件分配表。

操作系统分配磁盘空间按簇来分配的。因此，文件占用磁盘空间时，基本单位不是字节而是簇，即使某个文件只有一个字节，操作系统也会给他分配一个最小单元——即一个簇。为了可以将磁盘空间有序地分配给相应的文件，而读取文件的时候又可以从相应的地址读出文件，我们把整个磁盘空间分成 32K 字节长的簇来管理，每个簇在 FAT 表中占据着一个 16 位的位置，称为一个表项。

对于大文件，需要分配多个簇。同一个文件的数据并不一定完整地存放在磁盘的一个连续的区域内，而往往会分成若干段，像一条链子一样存放。这种存储方式称为文件的链式存储。为了实现文件的链式存储，硬盘上必须准确地记录哪些簇已经被文件占用，还必须为每个已经占用的簇指明存储后继内容的下一个簇的簇号，对一个文件的最后一簇，则要指明本簇无后继簇。这些都是由 FAT 表来保存的，FAT 表的对应表项中记录着它所代表的簇的有关信息：诸如是否空，是否是坏簇，是否已经是某个文件的尾簇等。FAT 区的结构如下：

表项	代码示例	含义
0	FFF8	磁盘标识字，必须为 FFF8，注意高位在后
1	FFFF	第一簇已经被占用
2	0003	0000h 可用簇 0002h-FFEFh 已占用，代码代表存放文件的簇链中的下一个簇的簇号 FFF0h-FFF6h 保留簇 FFF7h 坏簇 FFF8h-FFFF 文件的最后一簇
3	0004	
4	0005	
5	FFFF	
6	0000	

FAT 区结构

FAT 的项数与硬盘上的总簇数相关（因为每一个项要代表一个簇，簇越多当然需要的 FAT 表项越多），每一项占用的字节数也与总簇数有关（因为其中需要存放簇号，簇号越大当然每项占用的字节数就大）。FAT 的格式有多种，最为常见是 FAT16 和 FAT32，其中 FAT16

是指文件分配表使用 16 位，由于 16 位分配表最多能管理 65536（即 2 的 16 次方）个簇，又由于每个簇的存储空间最大只有 32KB，所以在使用 FAT16 管理硬盘时，每个分区的最大存储容量只有（65536×32 KB）即 2048MB，也就是我们常说的 2G。现在的硬盘容量是越来越大，由于 FAT16 对硬盘分区的容量限制，所以当硬盘容量超过 2G 之后，用户只能将硬盘划分成多个 2G 的分区后才能正常使用。

由于 FAT 对于文件管理的重要性，所以 FAT 有一个备份，即在原 FAT 的后面再建一个同样的 FAT。

3）根目录区（Root Directory Region）

紧接着第二个 FAT 表的后面一个扇区，就是根目录区了。根目录区中存放目录项，每个目录项为 32 个字节，记录一个文件或目录的信息 (长文件名例外) 。以下是目录项的结构：

偏移	长度／字节	说明	格式								备注
00H	8	文件名	ASCII 字符，当首字母如下时为特殊代码： 00H=未用名称 05H=当文件的第一个字符为 E5H 时，必须换成 05H,因为 E5H 在首字母时另有含义。 E5H=文件已使用，但已经删除 2EH=本项为目录								不足八个字节时，必须以空格填满
08H	3	文件类型（扩展名）	ASCII 字符								不足三个字节时必须填满
0BH	1	文件属性	7	6	5	4	3	2	1	0	
			未定义	未定义	存档	目录	卷标	系统	隐藏	只读	
0CH	10	保留									
16H	2	上次更新时间	须经编码 :(unsigned 16 bit-bit integer) time=Hr*2048+Min*32+Sec+2								*:高位在后，低位在前 LSB
18H	2	上次更新日期	须经编码 :(unsigned 16 bit-bit integer) time=(Yr-1980)*512+Mon*32+Day								*:高位在后，低位在前 LSB
1AH	2	起始簇号	此文件开始的簇号，如果文件有多簇，根据 FAT 中与此对应项的信息可得下一簇簇号								
1CH	4	文件大小	文件长度								

目录项结构

目录项所占的扇区数与有多少个目录项有关，它将占用（目录项＊32／512）个扇区。

4）文件和目录数据区

目录项的所占的最后一个扇区之后，便是真正存放文件数据或是目录的位置了。

2.4.2 硬盘结构

FAT 结构是所有按照 FAT 文件系统来组织存储单元的介质都必须遵守的一种文件系统格式，而对于不同的介质，其结构又有些差异，下面介绍一下文件系统格式为 FAT 的硬盘的结构。

硬盘上的数据按照其不同的特点和作用大致可分为 5 部分：MBR 区、DBR 区、FAT 区、DIR 区和 DATA 区。下列分别介绍：

0 磁道 0 柱面 1 扇区	MBR 区（主引导记录区）
0 磁道 1 柱面 1 扇区	DBR 区（操作系统引导记录区）
0 磁道 1 柱面 2 扇区～ 0 磁道 1 柱面 2+i-1 扇区	FAT 区（文件分配表区） 视磁盘容量而定，其占用的扇区数为 i， 磁盘总空间/32K＝总簇数，对于 FAT16，则所占扇区数 i=（总簇数＊2／512），每扇区字节数为 512 字节
0 磁道 1 柱面 2+i 扇区～ 0 磁道 1 柱面 2+2i-1 扇区	第二个 FAT 区，内容与第一个 FAT 区一样
0 磁道 1 柱面 2+2i 扇区～ 0 磁道 1 柱面 2+2i+j-1 扇区	DIR 区（根目录区） 视磁盘根目录项而定，其占用扇区数为 j
0 磁道 1 柱面 2+2i+j 扇区	DATA 区（数据区） 文件数据真正开始存放的地方

磁盘上的数据结构

（1）MBR（Main Boot Record）区

按其字面上的理解即为主引导记录区，位于整个硬盘的 0 磁道 0 柱面 1 扇区。不过，在总共 512 字节的主引导扇区中，MBR 只占用了其中的 446 个字节（偏移 0－偏移 1BDH），另外的 64 个字节（偏移 1BEH－偏移 1FDH）交给了 DPT(Disk Partition Table 硬盘分区表)（见下表）,最后两个字节"55，AA"（偏移 1FEH－偏移 1FFH）是分区的结束标志。这个整体构成了硬盘的主引导扇区。大致的结构如下：

0000 ～ 01BD	MBR 主引导记录（446 字节）
01BE ～ 01FD	四个分区信息表， 每个分区信息表占 16 字节， 总共 64 字节
01FE	55
01FF	AA

MBR 区的结构

主引导记录中包含了硬盘的一系列参数和一段引导程序。其中的硬盘引导程序的主要作用是检查分区表是否正确并且在系统硬件完成自检以后引导具有激活标志的分区上的操

作系统，并将控制权交给启动程序。MBR 是由分区程序（如 Fdisk.com）所产生的，它不依赖任何操作系统，而且硬盘引导程序也是可以改变的，从而实现多系统共存。

MBR 中可以定义四个分区信息表，每个分区信息表的 16 个字节定义如下：

偏移	长度	所表达的意义
0	字节	分区状态：如0-->非活动分区 80--> 活动分区
1	字节	该分区起始头（HEAD）
2	字	该分区起始扇区和起始柱面
4	字节	该分区类型：如82--> Linux Native分区 83--> Linux Swap 分区
5	字节	该分区终止头（HEAD）
6	字	该分区终止扇区和终止柱面
8	双字	该分区起始绝对分区
C	双字	该分区扇区数

图 分区信息表结构

比如，以一个实例来直观地介绍分区信息表中的内容：

例：80 01 01 00 0B FE BF FC 3F 00 00 00 7E 86 BB 00

最前面的"80"是一个分区的激活标志，表示系统可引导；"01 01 00"表示分区开始的磁头号为 01，开始的扇区号为 01，开始的柱面号为 00；"0B"表示分区的系统类型是 FAT32，其他比较常用的有 04（FAT16）、07（NTFS）；"FE BF FC"表示分区结束的磁头号为 254，分区结束的扇区号为 63、分区结束的柱面号为 764；"3F 00 00 00"表示首扇区的相对扇区号为 63；"7E 86 BB 00"表示总扇区数为 12289622。

（2）DBR 区

DBR（Dos Boot Record）是操作系统引导记录区的意思。它通常位于硬盘的 0 磁道 1 柱面 1 扇区，是操作系统可以直接访问的第一个扇区，它包括一个引导程序和一个被称为 BPB（Bios Parameter Block）的本分区的参数记录表。引导程序的主要任务是当 MBR 将系统控制权交给它时，判断本分区根目录前两个文件是不是操作系统的引导文件（以 DOS 为例，即是 Io.sys 和 Msdos.sys）。如果确定存在，就把其读入内存，并把控制权交给该文件。BPB 参数块记录着本分区的起始扇区、结束扇区、文件存储格式、硬盘介质描述符、根目录大小、FAT 个数，分配单元的大小等重要参数，如在 FAT 结构中所介绍的那样。

（3）FAT 区

在 DBR 之后的即是 FAT（File Allocation Table 文件分配表）区。文件分配表负责给文件分配空间，故称之为文件分配表。

以簇为单位的存储方法存在着必然的缺陷，即总是无法占满整簇的空间，存在着空闲的空间。簇的大小与磁盘的规格有关，一般情况下，软盘每簇是 1 个扇区，硬盘每簇的扇区数与硬盘的总容量大小有关，可能是 4、8、16、32、64……

（4）DIR 区

DIR（Directory）是根目录区，紧接着第二 FAT 表（即备份的 FAT 表）之后，记录着根目录下每个文件（目录）的起始单元，文件的属性等。定位文件位置时，操作系统根据 DIR 中的起始单元，结合 FAT 表就可以知道文件在硬盘中的具体位置和大小了。

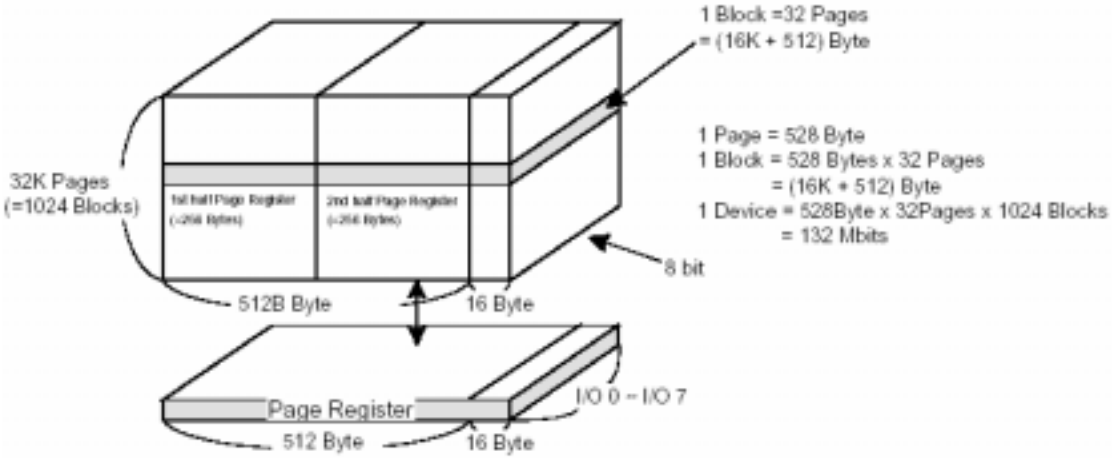
（5）数据（DATA）区

数据区是真正意义上的数据存储的地方，位于 DIR 区之后，占据硬盘上的大部分数据空间。

2.4.3 Flash 盘的 FAT 结构

Flash 硬盘与普通的磁头、柱面式介质不一样。在开发 U 盘的过程中，使用 Flash 作为存储介质。它有其特定的结构特点。

以 16M 的三星 K9F2808U0A-YCB0 Flash 为例，它有 1024 个 Block，每个 Block 有 32 个 Page，每个 Page 有 512+16＝528 个字节。



Flash 的结构

Flash 的读写有其自身特点：1）必须以 Page 为单位进行读写；2）写之前必须先擦除原有内容；3）擦除操作必须对 Block 进行，即一次至少擦除一个 Block 的内容。

针对这种情况，将 Flash 的一个 Page 定为 1 个扇区，将其 2 个 Block，64 个扇区定为一个簇，这样，簇的容量刚好为 512＊64＝32K，满足 FAT16 对簇大小的要求。

FAT 分配空间的时候，是按簇来分配的，但是其给出的地址却是 LBA（Logical Block Address），即它只给出一个扇区号，比如对此 Flash 而言，若给出 LBA 为 0x40，实代表簇 1 的扇区 1。因此需要将 Logical Block Address 转换为物理地址，这样，才可以对数据进行存取操作。根据我们定义的结构，转换公式为：

Flash 的 Block ＝ Logical Block Address／0x20

Flash 的 Page ＝ Logical Block Address %0x20

实际上，如果定义每个簇为 32 个扇区是最好的，因为这样物理结构和逻辑结构刚好一致。但是这也无防，因为不管 Logical Block Address 给出什么值，只要按上述公式，总可以得到物理上正确的 Block 和 Page，再使用 Flash 的读写命令读取对应的 Block 和 Page 就可以了，读的问题复杂一些，在后面介绍。

因此簇和扇区的概念只是在 BPB 中给出存储介质信息的时候告之系统就可以了，我们只要做好 LBA 与物理地址间的转换就可以了。

由于做为 U 盘的 Flash 不要求启动，因此可以没有 MBR 区，只包含 DBR、FAT、DIR 和 DATA 四个区。

因此，Flash 的前两个 Block 的内容如下：

LBA	Block／Page	长度	内容说明
0	0,0	512 字节	MBR＝BPB＋Excutable Code+55AA（查看内容）
1~2	0,1~0,2	1024 字	FAT 区（第一份 FAT）

		节	
3~4	0,3~0,4	1024 字节	FAT 区备份（第二份 FAT）
5~39H			目录区(在 BPB 中调整目录项数，使其刚好占尽本簇)
40H~			数据区（因目录区占尽一个簇，故数据区始于新簇首扇）

当 Host 发出 READ 命令后，Flash 读写操作即告开始，Host 首先读取 MBR，得到有关存储介质的有关信息，诸如扇区长度、每簇扇区数以及总扇区数等内容，以便知道此盘有多大。如果读取正确，会接着读取文件分配表，借以在 PC 机上的可移动盘符中显示文件目录，并可以复制、删除或是创建文件。系统自动将这些命令都转换成 Read 或 Write 两种命令，通过 USB 的 READ 或 WRITE 命令块描述符来从 Flash 中相应扇区读取数据，或是将特定长度的数据写入 Flash 相应簇中。

2.5 Flash 的读写

针对 Flash 读写的特点，特别是其可随机读，但无法随机写的问题，需要通过设置缓冲区来解决。在与 USB Host 进行数据交换的过程中，最小的单位是扇区：512 字节。由于 Flash 在写之前必须先擦除，而一擦又必须擦一个 Block，因此在擦除某 Block 之前必须保存同一个 Block 中有关扇区的数据。因此，如果每收到一个扇区的内容就进行一次擦、保存、写的操作，系统任务将十分繁重，无法及时响应 USB Host 端的请求。

因此，在系统中设置 32K 的缓冲区（ARM7 系统具有 2M SDRAM，因此内存足够，如果在 8051 平台上，则需要另外想办法），每完成一次数据传输后，记下本次要写的开始扇区和总扇区数，将本次要写的数据所涉及的扇区以外的数据从 Flash 中读出来，存放在缓冲区中对应位置，然后擦除一个 Block，再将缓冲区中内容一次全部重新写入 Flash。

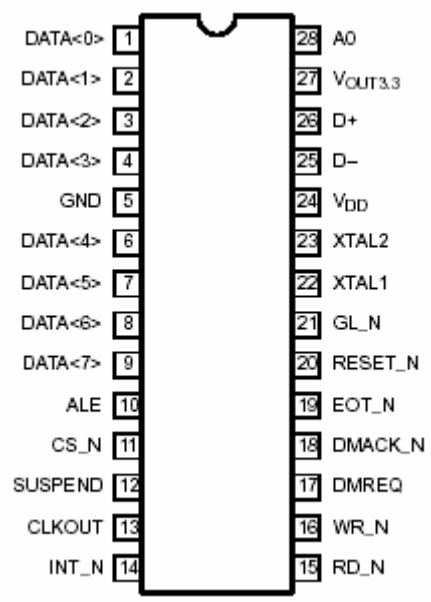
三、项目开发调试过程

在进行 USB 开发过程中，比较难的是对设备端的程序（称为固件，Firmware）进行调试。由于 USB 协议有严格的时间要求，这就使得程序必须在有效时间内对某些请求或状态进行处理，否则，USB 将无法正常工作。因此，在调试过程中，较多借助串口输出程序输出的一些信息来辅助调试，定位问题所在。比如，在某个函数中加入输出语句，程序运行时看有否特定的输出内容，借此来判断此函数是否得到了执行，并通过输出一些量来查看状态。

调试工作基本分三步进行：首先对外部设备（单片机部分）借助 PC 调试软件(芯片生产商提供或从网上下载 Bus Hound，WINRT－USB 等调试软件）将设备端的 USB 协议(主要有描述符请求、端口配置、地址设置以及基本数据交换）调通。然后，用调试好的 USB 设备接口来开发、调试 PC 软件，这一步相对比较容易。最后，加上 USB 设备端的其它用户程序，对整个完整的系统进行系统调试。

3.1 硬件电路和基本程序结构

下图为 PDIUSBD12 的引脚定义，其中，下列几个方面在制作电路板时应该注意。



PDIUSBD12 的引脚定义

1)GND 接地，VDD 接正（3.3V 或 5V），如果芯片工作在 3.3V，则 Vout3.3 与 VDD 都接 3.3V；如果芯片工作在 5V，则这时 Vout3.3 会输出 3.3V 的电压，用于提供给 D＋作参考电压，因为此参考电压必须为 3.3V。对于 U 盘来说，由于 USB 接供的是 5V 电压，因此应该按后一种接法接。即 GND 接 USB 接口中的 GND，VDD 接 USB 接口中的 VDD，D＋D－分别接 USB 接口中的对应位。

2)DATA<0>~ DATA<7>、WR_N、RD_N、INT_N、XTAL1 和 XTAL2 按传统接法接。

3)由 GND 接一个 1M 电阻，再从 USB 电源 VCC（USB 四根线，分别为 VCC，D＋、D＋和 GND）接一个 10K 的电阻，然后将 1M 和 10K 接在一起，引至 EOT_N 引脚，借此检测 USB 设备是否已经连接到 USB 口，SoftConnect 功能只有在检测到此信号时才会进行连接，换言之，如果不接此信号，则 D12 总认为还没有插入到 USB 接口中，SoftConnect 永远不会连接。

4)CS_N 和 DMACK_N 信号任一为低时，均可选中此芯片，因此，如果不使用 DMA 方式，则应将 DMACK_N 接高电平，使用 CS_N 作为片选。

5）ALE 和 A0 的接法必须组合在一起，根据 USB 芯片与 MCU 之间数据地址总线情况的不同，有两种接法：a)如果总线和地址复用，则可以将 ALE 接至 MCU 的 ALE，A0 接高电平，这种情况下 D12 会在 ALE 的下降沿锁存地址信号，直接将数据写入对应的 USB 地址码中。比如 D0 是 D12 的地址使能命令字，则直接将要使能的 USB 地址写入 D0 中，在 MCU 的 ALE 下降沿，D12 先将 D0 保存下来，然后再将端口地址作为数据送至 D12；b)如果 MCU 总线和地址是分开的，则 ALE 总接低，A0 为高时表示 DATA<0>~DTAT<7>上收到的是命令字，A0 为低时表明收到的是数据，通过将地址线的高位接至 A0，D12 就可以有独立的命令和数据端口，同样，端口使用时，先将 D0 写入命令端口，再将端口号写入数据端口，不同的端口其实只有 A0 的变化，从而告之 D12 当前发送的是什么内容。

6）CLOCKOUT 可以作为 MCU 的时钟源；SUSPEND 接至一个双向 IO 口；GL_N 接发光二极管后加电阻接高电平，它的亮暗将反映 USB 芯片的工作状态。

完成上述硬件电路的连接后，即可开始准备固件程序，主要包括三部分：a)初始化单片机和所有的外围电路 （包括 PDIUSBD12）；b)主循环部分，主要完成对来自 USB Host 端的 Setup 包，其任务是可以中断的；c)中断服务程序，其任务是对时间敏感的，必须马上执行。

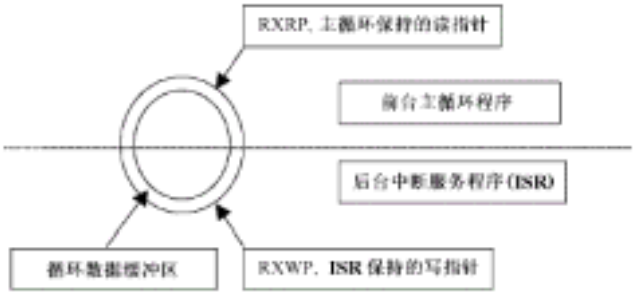
根据 USB 协议，任何传输都是由主机（host）开始的，这样，单片机作它的前台工作

（可能就是空循环，什么也不做），等待中断。当 USB 设备插入主机时，主机将使用缺省的端口 0 向 USB 发送信息，会发送一系列令牌包给 USB 设备（这里是 PDIUSBD12），PDIUSBD12 接收到令牌包后就给单片机发中断，单片机进入中断服务程序，首先读 PDIUSBD12 的中断寄存器，判断 USB 令牌包的类型，然后执行相应的操作。因此，USB 单片机程序主要就是中断服务程序的编写。

下面用形象通俗的语言描述一下 USB 设备端是如何与主机建立起通信的。当一个 USB 设备在插入主机之前，主机对这个 USB 设备的情况一无所知，这时当然无法建立起通信。但 USB 协议规定了一些最基本的准则，比如说每个设备的端点 0 都是可用的，属于控制端点。有了这个基本的沟通途径，主机就开始通过端点 0 向设备提出一些问题，这些问题是有关设备的基本情况的。这些基本情况可以反映 USB 设备所属的类别及子类，反映配置情况、接口情况和端点情况，一旦得知了这些信息，那么主机就大体了解这个设备是个什么样的设备了，按照 USB 协议中的相应规定，就逐步建立起了一条介于设备之间的高速数据通道，用于数据的传输。主机向设备提出的这些问题实际上就是 USB 协议中规定的各种标准请求，设备必须对这些问题进行回答，而回答的方式就是向主机传送相应的描述符：设备描述符、配置描述符、接口描述符、端点描述符。

这里还要明白一个概念，USB 设备中与主机直接打交道的是 USB 芯片，这里为 D12，因此，D12 负责数据的接收和发送，并且在接收数据或是完成数据发送后，会产生相应的中断，不同的中断以中断寄存器中不同的状态来表示。当主机首先通过端点 0 向 D12 发出最初的配置包的时候，D12 会产生端点 0 接收中断，单片机的中断服务程序要做两件事：a)读取中断及相应的数据，处理寄存器，为下一次中断创建条件；b)对中断进行处理，根据接收到的请求内容，或向主机发送相应数据，或在接收到数据后将数据从缓冲区中读走并进行分析处理，或是在上一帧数据发送成功后继续后续发送……

Philips 提供的 D12 手册中，介绍了下图所示的一种程序结构，即使用相同的数据缓冲区，中断服务程序专注了读取中断寄存中的数据并对寄存器相应的设置和处理，主程序专注于对数据的处理。这种思想，其实只是结构上的一种表现形式，即形式上主程序专门处理一些事情，中断服务程序专门处理一些事情。但实质上，还是以中断服务程序为中心的，因此采用这种结构，只是将一部分程序挪到了主程序，但其仍是由中断服务程序中的相应内容来驱动的。而且，一般来说，当 USB 主机发送来一个请求，D12 产生一次中断后，就需要设备对此中断进行响应，比如回传一些数据，而这些数据在回传之前，一般是不会再来中断的，因此，没有必要一定要将数据放到某个缓冲区里面，然后退出中断服务程序，再跑到主程序中去处理。完全可以直接在中断服务程序中处理完毕后，再退出中断，因为主机在退出中断之间一般不会再来中断。这样使程序结构上思路更加清晰、以请求类型为驱动的中断服务程序也较好构建，这时其实可以认为主程序其实是在轮询，什么也不干，就是等着中断，来什么中断就进相应的中断服务程序内容进行处理，执行完毕后再到主程序里去等着新中断的到来。



Philips 手册中提供的程序结构

3.2 U 盘调试的主要步骤和内容

USB 设备端的固件分以下几个层次：

文件模块名称	主要功能
Main.c	进行各种初始化操作、寄存器设置、中断设置
Fat16.c flash.c	负责按照 Fat16 文件系统的组织向 Flash 中写入数据或是从 Flash 中读出数据
Chap9.c bulk-only.c	完成不同的中断请求，Chap9 完成来自端点 0 的 USB 标准设备请求，Bulk-Only 完成来自批量模式端点的 Mass Storage Bulk-Only 传输中断请求
Isr.c	中断服务程序，负责将不同类型的中断转向一同的地方
D12ci.c	函数化的 D12 的命令集合，可以直接调用这些函数，而不必再自己根据手册查每个命令的代码 另外，此文件中包括一些与硬盘有关的地址定义

在调试的时候，从现象上来看，分成以下几个阶段性的步骤：1、USB 芯片正常工作，可以实现软连接，此时 PC 机上会出现“未知设备类型”的 USB 设备；2、使用他人已经高度成功的 USB 通用接口，按普通 USB 设备提供描述符，提供正确的 VID 和 PID 后，PC 能够识别设备，但要求提供设备的驱动程序；3、安装驱动程序后，调试几个端点，使其均可传输数据，用 PC 端的测试程序对其进行测试，验证硬件及固件的正确性；4、按 Mass Storage Bulk－Only 模式提供描述符，PC 机上设备类型变成 Mass Storage Device；5、响应了 Bulk－Only 的 Inquiry 命令，可以出现盘符了，但尚无法访问磁盘；6、提供了其他所有的 UFI 命令（SCSI 子集），开始读取磁盘 0 扇区（BPB 区）的内容，按照 FAT16 的格式格式化 Flash，可以正确读取信息，可以访问盘符，列目录为空；7、创建文件时，向设备发出 Write 命令，调整 Flash 的读写问题，解决写某几个扇区要先保存整个簇的内容，然后擦除整簇，再回写，可以正常创建文件；8、完成最后的调试，U 盘高度完毕。

在此基础上，还需要提供支持 FAT16 的文件系统接口函数，比如，可以从 FAT16 中读取文件，可以创建文件并将其保存到 FAT16 中去。

3.3 检查 USB 器件工作与否

确定 USB 芯片是否已经正常工作，是所有调试的基础，得到电路板之后，这是一个首先要解决的问题。

判断 USB 芯片是否已经工作，可以从两个方面来判断，一是将 CLKOUT 设为 12M（缺省为 4M），然后用示波器量 CLKOUT 引脚，若确为 12M 而不是 4M，则说明 USB 的地址和数据都可以正确传送，各种信号线接法正确，USB 芯片可以正常工作，硬件基本没有问题。

另一方面，如果使用 SoftConnect，则在主程序中进行软连接后，如果 GL_N 灯闪烁几下，则 PC 机上出现未知设备，则说明 USB 进行软连接正常，也说明芯片已经工作了。

除了这两点之外，在整个调试标准设备请求阶段，都应该将中断寄存器的内容通过串口发送出来，以便进行查看。另外就是还可以使用 Bus Hound，观察 USB 总线上的数据，从而判断主机与设备之间的通信已经进行到哪一步了。

在主程序中，对 USB 初始化过程为：

① 初始化 MCU 的各种端口；

② 进行中断初始化，设置中断服务程序入口地址，将 MCU 的中断方式设置为低电平触发，因为 D12 只要进入中断后 INT_N 就一直为低。

③ 断开 SoftConnect，延时 1 秒后再次连接

④ 进入主程序循环，等待中断的到来，端点 0 的中断处理程序会将数据放至缓冲区中，主程序在前台判断是否有 Setup 包（通过一个变量，当中断服务程序检测到有 Setup 包时，设置该变量），然后执行相应的控制端点的传输。

主程序的这些工作完成以后，便可以根据前面提供的两种方法测试 D12 芯片是否已经工作了。 以下两点还应该引起注意：

PDIUSBD12 的中断输出引脚 INT_N 只要中断寄存器不为 0 就保持低电平，所以单片机的对应中断应设置成电平触发；中断处理后要用读上次传输状态寄存器清除中断寄存器中对应位(D0－D5)。

PDIUSBD12 对内部寄存器的读写没有边界限制，程序设计中一定不要读写超过端点深度的数据。特别对于描述符请求，由于其长度大于 Control IN 深度（64Bytes），要分几个数据周期传输。

3.4 提供描述符

USB 设备的调试过程其实就是根据主机的请求，不断地向主机提供各种信息的过程。因此，了解主机按照什么样的顺序向设备发出请求，即 Windows 对 USB 设备的枚举顺序是非常有必要的：

① GetDeviceDescriptor。主机主要对 Length 域感兴趣，发送内容一定要正确，特别是第 2 字节 type 一定为 0x01，即 Device；否则，主机将不响应，或 者再重复 2 次后放弃。这时由于主机对 Device 的描述符将有多长实际上都不知道，所以这个步骤只是试探性的，目的是得到真正长度，第三步中才正而巴经地读取 DeviceDescriptor。查看 DeviceDescriptor

② SetAddress。一般为 02 或 03。

③ 连续 3 次 GetDeviceDescriptor，读取全部设备描述符，一般为 18 B，分为多次传输。如果不正确，主机将不响应或重复 2 次后放弃。

④ GetConfigDescriptor。注意第 2 字节一定为 0x02，即 config 。这部分内容包括 Configuration Descriptor、Interface Descriptor 和所有要用到的端点的 EndPooint Descriptor。

⑤ GetStringDescriptor（可能没有），根据在设备描述符中是否有 String 索引而定。一般先读取 LanguageID，再读取 product string。

⑥ 读取全部 ConfigDescriptor，次数根据描述符的大小决定（端点个数不同，描述符大小不同），如 果不正确，主机将不响应或再重复 2 次后放弃。

⑦ 如果以上步骤都正确，主机将找到新设备，提示安装驱动程序；否则找到未知设备，不可用。

安装驱动程序后，以后的每次 PlugIn，枚举次序与以上步骤略有不同，之后会有 SetConfiguration、GetConfiguration 和 GetInterface 等调用。

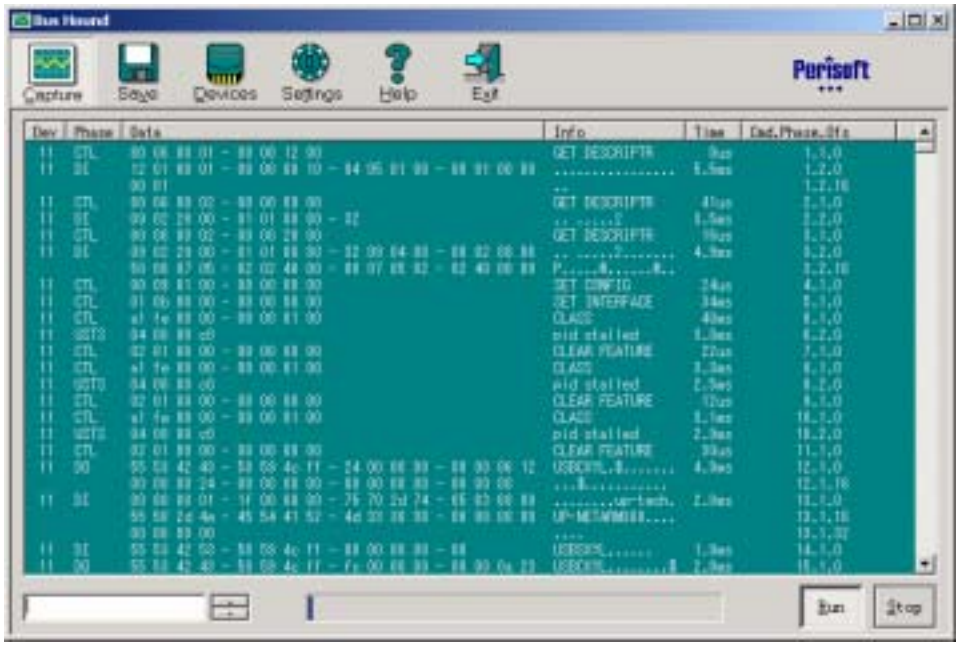
需要说明的是，USB 协议中所有字数据均定义为低字节在前传输（LSB），因此如果所使用 MCU 对字的处理方式与之不同时，就需要进行转换。比如，ARM 平台上也是 LSB，因此不必转换，而在 8051 平台上，是 MSB，因此 Phlips 的 VID 为 0471H，但在 Device Descriptor 中的 iDVendor 应定义成 71H、04H。

正确安装驱动程序以后，还可以通过一些工具来查看 USB 设备的描述符情况，以确定自己所提供的描述符信息是否都正确。

在调试过程中，指望得到一个绝对正确的提供描述符的顺序是没有什么用处的，最好的办法就是掌握调试的方法，因为一旦掌握这个方法，那么在后续的调试过程中就更加得心应手了。下面就介绍如何使用 Bus Hound 软件来辅助调试。

下图所示即为 Bus Hound 获取 USB 总线上的数据的情形，有三个部分的数据对调试来讲比较有用，Phase 可以让调试者知道当前处于什么阶段，是控制命令发出，数据发出还是接收到数据。Data 则可以让我们清楚地知道主机向 Device 发了些什么，或是 Device 向主机发了些什么，将此数据与 Device 从主机收到的数据或是发向主机的数据进行对比，可以确保数据是否传输正确。Info 阶段则可以让我们知道当前处于 USB 协议的什么阶段，它可以清晰地表示出是在 GET DESCRIPTOR 阶段还是 SET CONFIG 阶段……

在此工具的最上面一行中，还可以通过 Save 将所得到的数据作为文件保存下来再进行分析；从 Devices 中可以选择要对哪些设备的数据进行捕获；Settings 中可以设置缓冲区的总长度和每个 Phase 数据长度的限制。总之，Bus Hound 是进行 USB 开发的利器，是调试过程中的必备工具。

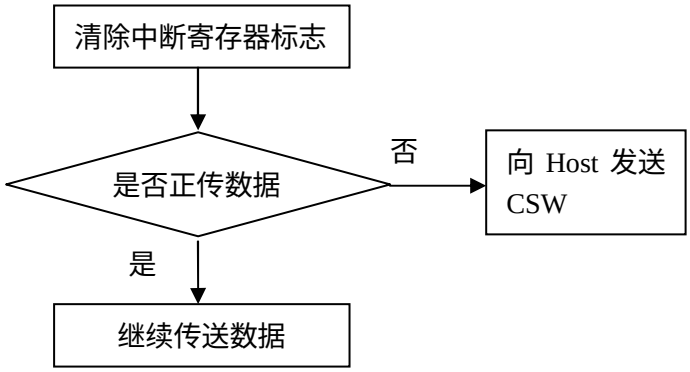


Bus Hound 调试工具

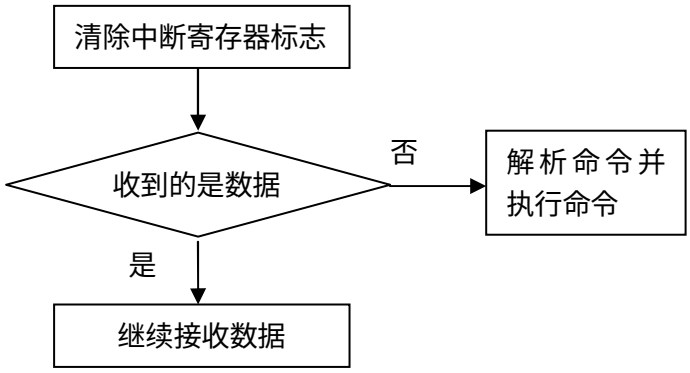
3.5 Mass Storage 协议

如果是按照 Mass Storage Bulk－Only 传输协议提供的描述符，则当 Bus Hound 中 DI 阶段中的数据出现 55 53 42 53 字样，说明已经开始开始发送 Bulk－Only 协议的 CBW 了。这时可以将前面的工作告一段落，即不用太多考虑 USB 中断啦、端点啦之类的问题，到此阶段后只有两个端点工作，即 Bulk－In 和 Bulk－Out，In 或 Out 都是从主机的角度来讲的，前者用于 Device 向 Host 发送数据，后者用于 Host 向 Device 发送数据。Bulk－In 端点的处理比较简单，在需要的时候，Device 将要发往 Host 的数据通过此端点送出即可，如果数据一次不能发完，则设置标志位，通过发送中断的产生可以实现连续发送。Bulk－Out 端点的情况比较复杂一些，要判断收到的是协议内容还是数据，如果是协议内容，要对协议包进行解析，根据协议中的内容得到 UFI（SCSI）命令，然后再根据这些命令处理处理相应的请求。

对 Bulk－In 端点的处理见下列流程图：



对 Bulk－Out 端点的处理见下列流程图：



3.6 SCSI 命令集

在 SCSI 命令集的处理中，比较难处理的是 WRITE 和 READ 比较难处理。因为 Host 传送数据至少一次为 512 个字节，不管是读还是写，都需要一次传送 512 个字节，但是批量传输端点一次只能传输 64 个字节，即要传 8 次才可以传完。因此，在向 Host 传送数据时，设置传输标志，当 Device 的发送中断出现后，判断此传输标志，如果正处于传输状态，则继续传输，直至需要的数据传输完毕，再传输 CSW。在从 Host 接收数据时，也设置接收标志，当收到 WRITE 命令后，即进入此状态，直至从 Host 端接收数据完毕，再将数据存入 Flash 中。

3.7 Flash 的读写

Flash 的读写直接调用两个函数 ReadPage（）和 WritePage 即可，他们可以一次写入 528 个字节（512＋16）的内容。但是注意必须在写数据之前首先将相应的簇擦除，擦除之前又需要先将不覆盖的数据保存。

四、总结

4.1 一点体会☺

在此项目的开发过程中，感觉 C 语言的熟练程度是项目开发的关键。许多问题其实原理上都是非常成孰的东西，从一些资料上都可以找到。在自己的项目中，要做的也就是将原理表述的东西在特定的平台上实现就可以了。因此，开发工作本身没有多少创造性，没有多少高深的知识，要具备的仅是对 C 语言的熟练，以及学会解决遇到的问题。

一个协议或是一个方法，其原理和实现的不同在于，原理比较抽象，比较粗略，不需要精确；而要真正用代码实现时，则要求具体、详细、精确，不点差错都不能有，因此，对于语言的熟练程度的确可以事半功倍，提高效率。

在众多的 C 语言的内容中间，在项目开发过程中值得引起注意的内容有：循环控制、数组的使用、指针的使用。灵活运用数组名作为地址指针，可以使用程序编制过程中许多问题变得容易解决。

另外就是动手实际开发，是学习的好机会，只有真正经历了开发过程以后，才会有更深的体会。

项目开发过程中，经验与能力的增长是相辅相成的。

有些内容是要凭经验，理论上的东西，是别人总结出来的东西，因此非常抽象，跟实践相比往往省略了许多细节，而项目开发有时候就是根据抽象的理论，来重新实现细节。如果某个问题在做第一遍不清楚，那么做第二遍时就一定知道了，就可以直接得到正确的结论，这就叫做经验。一次的经验并不能成为能力，凭一次的经验快速解决问题称不上能力的高低，但是，当进行长期的积累，对某一类问题有了丰富的经验之后，那么能力的高低就有所体现了。多次的经验积累，可以在能力上得到提高，以后遇到问题，就能够找到问题的原因，并分析问题所在，从而解决实际问题。

在此项目的开发过程中，最重要的就是掌握调试手段，不断解决调试过程中发现的问题。比如，在响应 Host 端的 WRITE 指令时，由于一次至少写 512 个字节，而 USB 的批量传输端点一次只可以发送 64 个字节，因此共需要传 8 次，传完 8 次后 Device 端需要返回 CSW，但在实际调试过程中，却发现这样的更象：传完 7 次数据后主机就不继续传了，然后报告超时，而如果在第 7 次传输后发送 CSW，Host 端又会再传过来一帧数据。这就和常理显得比较不一致。但是即使在高度过程中发现这个问题，就在所有传数过程中，传到还剩下一帧时，向 Host 发送 CSW，然后再接收一帧数据，结果发现这种处理方式可靠。这种现象在已有的资料中并没有描述，是在调试过程中发现的，因此，如果完全凭借资料，是很难解决这一问题的。

4.2 后记

本文中所涉及的许多技术资料,可以到 www.t10.org 下载,BusHound 软件,可以到 www.DriverDevelop.com 下载。