USB基础知识概论

版本：v0.9.2

Crifan Li

摘要

本文主要介绍了USB协议的基础知识，包括USB协议的来龙去脉，OHCI/UHCI/EHCI/xHCI之间的区别和联系，USB的firmware，以及USB 2.0协议的概览，USB的枚举的详细过程，OHCI的一些特点

本文提供多种格式供：

在线阅读	HTML [1]	HTMLs [2]	PDF [3]	CHM [4]	TXT [5]	RTF [6]	WEBHELP [7]
下载（7zip压缩包）	HTML [8]	HTMLs [9]	PDF [10]	CHM [11]	TXT [12]	RTF [13]	WEBHELP [14]

HTML版本的在线地址为：

http://www.crifan.com/files/doc/docbook/usb_basic/release/html/usb_basic.html

有任何意见，建议，提交bug等，都欢迎去讨论组发帖讨论：

http://www.crifan.com/bbs/categories/usb_basic/

修订历史		
修订 0.5	2011-10-06	crl
1. USB基础知识概论		
修订 0.9.2	2015-05-26	crl
1. 通过Docbook发布 2. 添加了USB协议细节部分的内容 3. 添加了USB OHCI学习笔记 4. 更新了所有的xml:id		

[1] http://www.crifan.com/files/doc/docbook/usb_basic/release/html/usb_basic.html
[2] http://www.crifan.com/files/doc/docbook/usb_basic/release/htmls/index.html
[3] http://www.crifan.com/files/doc/docbook/usb_basic/release/pdf/usb_basic.pdf
[4] http://www.crifan.com/files/doc/docbook/usb_basic/release/chm/usb_basic.chm
[5] http://www.crifan.com/files/doc/docbook/usb_basic/release/txt/usb_basic.txt
[6] http://www.crifan.com/files/doc/docbook/usb_basic/release/rtf/usb_basic.rtf
[7] http://www.crifan.com/files/doc/docbook/usb_basic/release/webhelp/index.html
[8] http://www.crifan.com/files/doc/docbook/usb_basic/release/html/usb_basic.html.7z
[9] http://www.crifan.com/files/doc/docbook/usb_basic/release/htmls/index.html.7z
[10] http://www.crifan.com/files/doc/docbook/usb_basic/release/pdf/usb_basic.pdf.7z
[11] http://www.crifan.com/files/doc/docbook/usb_basic/release/chm/usb_basic.chm.7z
[12] http://www.crifan.com/files/doc/docbook/usb_basic/release/txt/usb_basic.txt.7z
[13] http://www.crifan.com/files/doc/docbook/usb_basic/release/rtf/usb_basic.rtf.7z
[14] http://www.crifan.com/files/doc/docbook/usb_basic/release/webhelp/usb_basic.webhelp.7z

USB基础知识概论:

Crifan Li

版本：v0.9.2

出版日期 2015-05-26

[15] http://www.crifan.com/files/doc/docbook/soft_dev_basic/release/html/soft_dev_basic.html#cc_by_nc

目录

插图清单

表格清单

缩略词

EHCI (EHCI)	Enhanced Host Controller Interface
NRZ (NRZ)	Non-Return-to-Zero
NRZI (NRZI)	Non-Return-to-Zero Inverted
OHCI (OHCI)	Open Host Controller Interface
RZ (RZ)	Return-to-Zero
SYNC (SYNC)	Synchronize
UHCI (UHCI)	Universal Host Controller Interface
USB (USB)	Universal Serial Bus 通用串行总线
xHCI (xHCI)	eXtensible Host Controller Interface

正文之前

1. 此文目的

由于USB所涉及的知识太多，如果想要在一篇文章里，把USB的方方面面的内容，都解释的很清楚，那几乎是不可能的。

因此，此文目的，不是为了把USB的所有的事情都写出来，而是让对USB不懂的人，通过此文档，能对USB有个基本的认识，并且搞懂USB世界中的基本的术语的含义。

即，此文目的，是为了给不熟悉USB的人，一个总体的概述，以及解释一些必要的USB方面的基本知识。

这样，如果想要更细节的去了解USB的知识，也知道从哪里入手，以及如何去找相关资料区学习了。

2. 关于一些USB方面的文档

USB很复杂，所以，如果能把复杂的东西解释的清楚的，不是很容易。

而且由于USB涉及知识面也很广，所以也很难简短地描述清楚。

2.1. 大而全的USB英文资料

对于众多的现存的USB的文章或书籍，我所见过的，能把USB讲的透彻的，算是

英文资料：《USB Complete》，中文翻译为《USB大全》，目前最新版本是第四版。

其主页是：

http://www.lvr.com/usbc.htm

网上也可以找到盗版的电子版的，第三版的有中文翻译，第四版的只有英文原版。

2.2. 简明扼要的USB英文资料

另外，简明扼要地，把USB讲解的很清楚的，我觉得算是《USB in a Nutshell》了，网上随便都可以搜到此文的pdf版本，比如：

USB in a Nutshell - Making sense of the USB standard [1]

而此文，也是主要根据此贴而来，基本可以算是《USB in a Nutshell》的中文版吧，然后另外又添加了一些必要的知识，以求把USB讲解的更加清楚。

2.3. 全系列的介绍Linux下的USB中文资料

另外，关于中文方面的资料，觉得写的比较全，解释的比较清楚的，算是fudan_abc写的《Linux那些事儿系列》，是一个系列的，好多个文档的。详细资料，已整理放到这里了：

【很好的学习Linux驱动的教材】Linux那些事儿系列[全][pdf] [2]

3. 声明

由于本人知识水平有限，错误在所难免，欢迎指正。

[1] http://www.beyondlogic.org/usbnutshell/usb-in-a-nutshell.pdf
[2] http://bbs.chinaunix.net/thread-1977195-1-1.html

更多建议，意见，吐槽，都可以联系偶：admin (at) crifan.com

第 1 章 USB的来龙去脉

1.1. USB是什么

USB是Universal Serial Bus的缩写，中文译为通用串行总线。

所以，从字面意思上，善于思考的人，就问有疑问：

那么与此USB相比，其他还有哪些非串行的总线，以及和此通用的串行总线来说，其他还有哪些相对“不通用”的串行总线呢？

对此，借用《USB Complete》里面所总结的，关于USB和其他接口的区别，来解释一下：

图 1.1. USB与其他总线的异同

Table 1-1: USB is more flexible than other interfaces, which often target a specific use.

Interface	Type	Number of Devices (including PC) (max.)	Distance (max. ft)	Speed (max. bps)	Typical Use
USB 3.0	dual simplex serial	127 (per bus)	9 (typical) (up to 49 with 5 hubs)	5 G	Mass storage, video
USB 2.0	half duplex serial	127 (per bus)	16 (98 ft. with 5 hubs)	1.5M, 12M, 480M	Keyboard, mouse, drive, speakers, printer, camera
eSATA	serial	2 (port multiplier supports 16)	6	3G	Drives
Ethernet	serial	1024	1600	10G	General network communications
IEEE-1394b (FireWire 800)	serial	64	300	3.2G	Video, mass storage
IEEE-488 (GPIB)	parallel	15	60	8M	Instrumentation
I²C	synchronous serial	40	18	3.4M	Microcontroller communications
Microwire	synchronous serial	8	10	2M	Microcontroller communications
MIDI	serial current loop	2 (more with flow-through mode)	50	31.5k	Music, show control
Parallel Printer Port	parallel	2 (8 with daisy-chain support)	10–30	8M	Printers, scanners, disk drives
RS-232 (EIA/TIA-232)	asynchronous serial	2	50–100	20k (115k with some hardware)	Modem, mouse, instrumentation
RS-485 (TIA/EIA-485)	asynchronous serial	32 unit loads (some chips allow up to 256 devices)	4000	10M	Data acquisition and control systems
SPI	synchronous serial	8	10	2.1M	Microcontroller communications

从上述表格中，表面上看，好像也没看出USB相对其他接口，有多么特别明显的优点，而只是看到在某些参数上，比其他某些接口参数更高，而在别的某些参数上，比其他接口低。

关于细节的区别，不是此讨论的重点，此处，我们至少可以看出，除了USB接口外，目前已存在的接口，还是很多的，而且各种接口实际上从硬件上也是形状各异，互相也都有自己的应用领域，而且无法兼容。基于此背景，才有下面的解释，以说明，为何会出现这么个USB接口。

此处，简单的说，USB就是一种接口，一种总线。

1.2. 为何要有USB

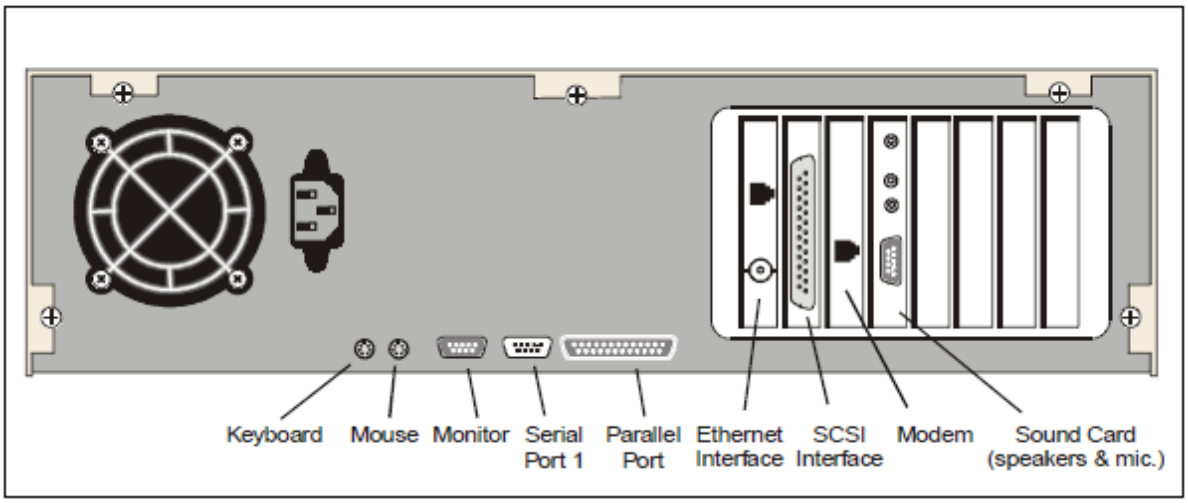
上面已经提到了，在USB出现之前，其实计算机领域中，已经存在众多的接口，而且不同的应用领域，已有一些相对来说是广泛使用的各种接口了。

但是，对于计算机等使用的普通用户来说，由于接口太多，而容易被搞得晕头转向。再加上各个接口从硬件形状和软件配置也都不一样，导致不兼容，为了不同的应用，而要配置多种不同的硬件接口，设置对于有些硬件接口来说，还需要手动去配置一些更细节的参数。

关于USB出现之前，计算机领域中的接口太多太繁杂，可以用下面这张，关于PC机箱背后的接口的图片来说明：

图 1.2. PC机箱后面的众多接口

Figure 1-2: Connectors at Backplane

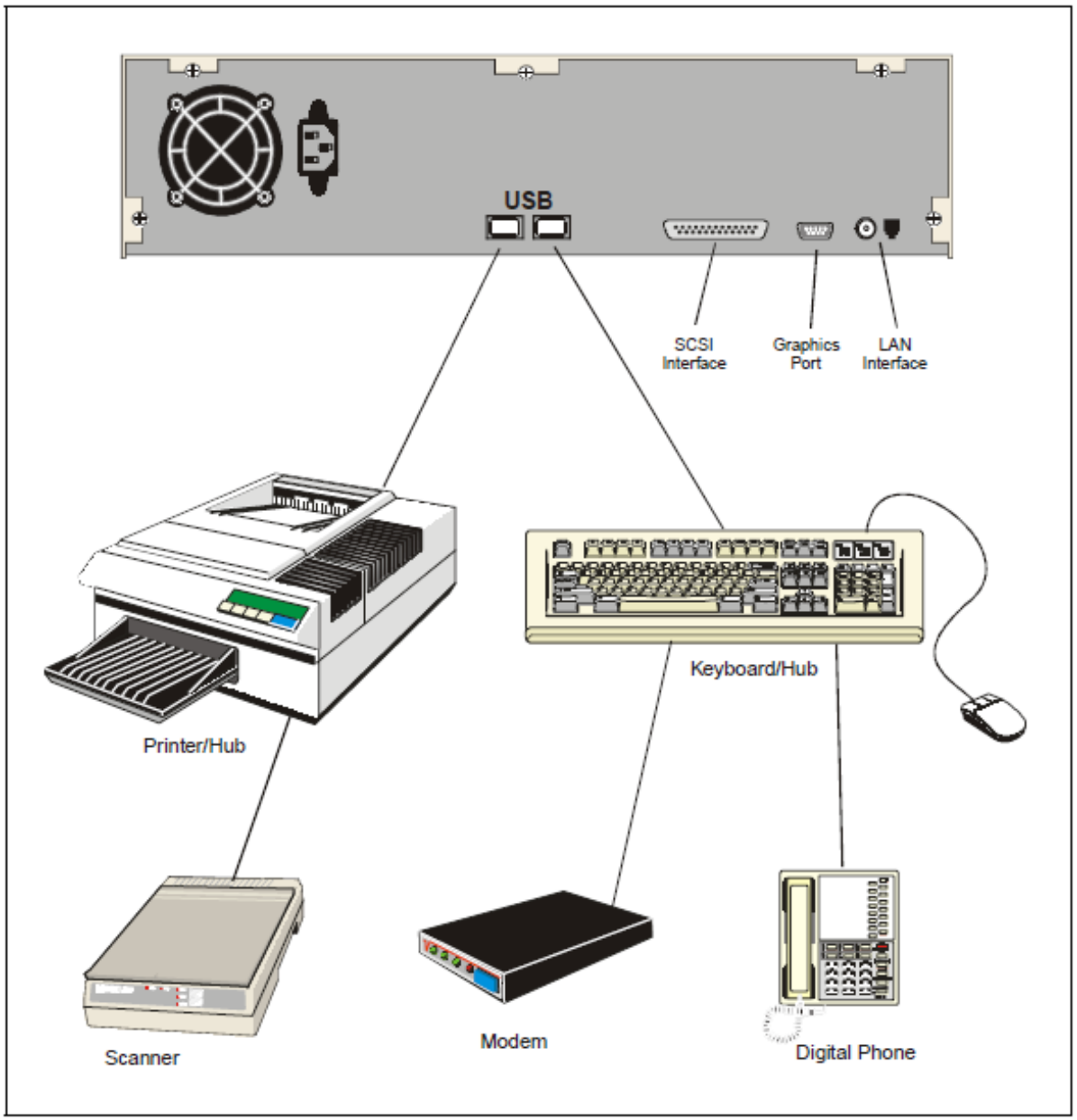


所以，总的来说，在USB出现之前，各种接口太多，而且都不太容易使用，互相之间的兼容性也较差，因此，才出现了USB。

而万能的USB接口出现的话，整个PC机箱背后的接口，就不那么繁杂，显得清静多了：

图 1.3. 有了USB接口之后的PC机箱背后的接口

Figure 1-3: USB Device Connections



USB出现的最初的目的，根据USB规范中的解释，是为了：

1. 将PC和电话能连起来
   由于大家都认识到，下一代的应用，肯定是实现计算机设备和通讯设备的完美融合。而且，为了实现移动领域内的人机数据的交互，也需要方便且不贵的连接方案。

   但是，计算机领域和通讯领域却是各自为政的发展，没有考虑互联性。由此，USB的出现，就是为了解决这一类互联问题的

2. 方便用户使用
   以前的一些设备，多数不支持即插即用，而且很多设备还需要懂行的用户去手动配置，然后才可以正常工作

   而USB的出现，使得用户不用关心设备的细节，不需要去另外再配置什么参数，直接插上就可以用了，而且还支持即插即用，很是方便

3. 接口扩展性要好
   之前的众多接口，导致不同的应用，需要使用不同的接口，很是繁琐。

   USB的出现，支持众多的应用，都使用统一的USB的接口，方便了用户，不需要再搞懂各种接口的用途和差异。

总的来说，USB的出现，是希望通过此单个的USB接口，同时支持多种不同的应用，而且用户用起来也很方便，直接插上就能用了，也方便不同的设备的之间的互联。

说白了，就相当于在之前众多的接口之上，设计出一个USB这么个万能的接口，以后各种外设，都可以用这一种接口即可。

这估计也是USB的名称中的Universal通用的这一个词的来历吧。

第 2 章 USB相关的基础知识

在介绍USB协议的细节知识之前，有很多相关的软硬件的基础知识，需要了解一下。

TODO：

http://www.silabs.com/Support%20Documents/Software/USB_Overview.pdf去学习USB的基础知识后整理过来。

2.1. USB相关的硬件

USB设备，从物理上的逻辑结构来说，包含了主机Host端和设备Device端。

其中，主机Host端，有对应的硬件的USB的主机控制器Host Controller，而设备端，连接的是对应的USB设备。

2.1.1. USB控制器类型：OHCI，UHCI，EHCI，xHCI

由于历史原因，导致USB的主机控制器，出现了多种不同的类型，即OHCI和UHCI，EHCI，和xHCI。

不论是那种USB主机控制器，简称主控，都是符合对应的USB的规范的，都是实现了对应的USB规范中所规定的USB主控所要的那些事情的。只不过是不同的USB主控的类型，有着不同的特点。

下面对这些不同类型的USB主控制器，进行简要的解释。

2.1.1.1. OHCI和UHCI

OHCI，Open Host Controller Interface，创立者是Compaq，Microsoft和National Semiconductor。

UHCI，Universal Host Controller Interface，创立者是Intel。

两者之间的相同点是：

不论是OHCI还是UHCI都是对应于USB 1.1的标准的，都是完全符合USB协议标准的。

区别在于：

只是各自的实现方式有些略微不同而已。当然对应的具体的性能，也略有差别，具体的差异，，和实际的应用有关系。

但是本身OHCI和UHCI的区别在于：

虽然都是实现了USB1.1协议规范，但是在功能划分上，OHCI更多地把要做的事情，用硬件来实现，因此，实现OHCI的USB控制器的软件驱动的开发工作，相对要容易些，软件要做的事情，相对较少。

对应地，OHCI更多地应用在扩展卡，尤其是嵌入式领域中，常见的很多开发板中的USB的控制器，很多都是OHCI的。

而UHCI把更多的功能，留给了软件，相对来说，软件做的事情，即负担要重些。但是实现对应的UHCI的硬件的USB控制器，价格上，就相对便宜些。

对应地，UHCI更多地应用在PC机中的主板上的USB控制器。

2.1.1.1.1. 为何Intel设计的UHCI把更多的任务都留给软件实现？

对于两者的区别和实际的应用，自己分析，不难发现，其是有着内在的逻辑关系的。

因此，作为UHCI的创立者Intel，创立了UHCI，把更多的USB需要做的事情，留给了软件，这样就可以实现出相对教便宜的USB的主控制器了，可以用于PC端的CPU所对应的主板上，便宜的USB主控，当然相对市场来说，更容易多卖出去一点，有利于市场推广。

PC主板卖出的多了， 自然对应的Intel的CPU，也会多卖点，Intel自然可以赚更多的钱了。

2.1.1.1.2. 为何嵌入式系统中的USB主控多用OHCI，而非UHCI？

而对应的Compaq，Microsoft和National Semiconductor所创立的OHCI，由于把更多的USB要做的事情，都用硬件实现了，这样对应的软件驱动所要做的事情，就少了，这样就有利于实现对应的OHCI的USB主控的驱动了，这点对于嵌入式系统来说，尤其重要，因为本身嵌入式系统就是资源有限，所以要尽量少的利用其他资源，比如CPU资源，去实现特点的功能，所以，倾向于采用对软件资源要求少的OHCI，而不是UHCI，否则用了UHCI的USB主控的话，需要实现对应的驱动，软件要做的事情太多，不利于在嵌入式系统这有限的资源环境下实现。

2.1.1.1.3. OHCI和UHCI技术细节上的区别

关于OHCI和UCHI在技术细节方面，更加详细的区别主要有这些：

1. 单帧内的stage的个数
   对于控制传输来说：
   - OHCI：在单个帧内，可以调度多个stage；
   - UHCI：在单个帧内，只调度一个stage。
2. 单帧内的transaction的个数
   对于最大数据包大小小于64字节的Bulk端点来说：
   - OHCI：单个帧内，可能会有多个transaction。
   - UHCI：单个帧内，不超过一个transaction；
3. 轮询的频率
   - OHCI：，即使端点描述符中，已经指定了最大延迟是255ms，但是OHCI主控还是会，至少每32ms就去轮询一次中断端点
   - UHCI：UHCI主控可以支持，但是不是必须要支持，更低频率地轮询

2.1.1.2. EHCI

EHCI，Enhanced Host Controller Interface。

简单说就是，EHCI定义了USB 2.0的主机控制器的规范，定义了USB 2.0的主控，需要包括哪些硬件实现，需要实现哪些功能，其也对应着对应的系统软件，所面对的是哪些接口。

EHCI对USB主控的定义，详细到了寄存器的级别了，即定义了你USB主控，都要实现哪些对应的功能和对应的寄存器有哪些，分别是何种功能等。然后对应的软件驱动人员，去写USB主控的驱动的时候，也就清楚有哪些可以利用的系统资源，如何去使用这些资源，读取，设置对应的寄存器，实现对应的功能了。

对应的EHCI规范，可以去Intel的官网找到：

EHCI Specification [1]

2.1.1.3. xHCI

xHCI，Extensible Host Controller Interface

[1] http://www.intel.com/technology/usb/ehcispec.htm

同EHCI是针对USB 2.0类似，xHCI是针对的USB 3.0规范。也是定义了USB 3.0主控需要如何实现，需要包含哪些功能，也是提供了寄存器级别的定义。

对应的xHCI规范，可以去Intel的官网找到：

Extensible Host Controller Interface (xHCI) Specification for USB 3.0 [2]

2.1.1.4. OHCI，UHCI，EHCI，xHCI的区别和联系

针对上述的解释，对USB的不同类型的主机控制器，简要概括如下：

表 2.1. 不同USB控制器类型OHCI，UHCI，EHCI，xHCI的区别和联系

USB主机控制器类型	共同点	区别			
		对应的USB的协议和支持的速率	创立者	功能划分	常用于
OHCI	都实现了对应的USB的规范中所要求的功能	USB 1.1=Low Speed和Full Speed	Compaq，Microsoft和National Semiconductor	硬件功能 > 软件功能⇒硬件做的事情更多，所以实现对应的软件驱动的任务，就相对较简单	扩展卡，嵌入式开发板的USB主控
UHCI			Intel	软件功能 > 硬件功能⇒软件的任务重，可以使用较便宜的硬件的USB控制器	PC端的主板上的USB主控
EHCI		USB 2.0=High Speed	Intel	定义了USB 2.0主控中所要实现何种功能，以及如何实现	各种USB 2.0主控
xHCI		USB 3.0=Super Speed	Intel	定义了USB 3.0主控中所要实现何种功能，以及如何实现	各种USB 3.0主控

2.1.2. USB接口的引脚定义

USB接口的物理上的对应的引脚和对应含义等，可用下表概括：

表 2.2. USB 1.x/2.0的引脚定义

引脚	名称	电缆颜色	描述
1	VBUS	Red	+5 V，电源
2	D−	White	Data −，数据线
3	D+	Green	Data +，数据线
4	GND	Black	Ground，接地

表 2.3. USB 3.0的引脚定义

Pin	Color	Signal name('A' connector)	Signal name('B' connector)
1	Red	VBUS	
2	White	D−	
3	Green	D+	
4	Black	GND	

[2] http://www.intel.com/technology/usb/xhcispec.htm

Pin	Color	Signal name('A' connector)	Signal name('B' connector)
5	Blue	StdA_SSRX−	StdA_SSTX−
6	Yellow	StdA_SSRX+	StdA_SSTX+
7	Shield	GND_DRAIN	
8	Purple	StdA_SSTX−	StdA_SSRX−
9	Orange	StdA_SSTX+	StdA_SSRX+
Shell	Shell	Shield	

2.1.3. USB的接口（connector）类型

由于USB的产生就是为了支持众多种应用的，而由于各种应用中，对于硬件接口的大小也有一些限制，比如有些小型设备或者移动式设备中，接口不能太大等，所以而设计出多种类型的接口，用于不同的应用。

在介绍插头和插座之前，先多解释一下，基本的叫法。

插头，plug，对应的也叫公口，即插别人的；

插座，receptacle，对应也叫做母口，即被插的；

对上述解释，想多了的，面壁去；没想多的，继续看技术介绍。

下面就来简单的介绍一下不同的USB接口类型，即各种不同的插头插座：

USB的接口类型，根据接口形状不同，主要可以分为三大类：

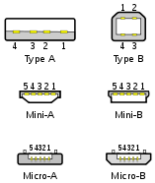
1. 普通的硬件直接叫做Type
2. 然后有小型版本的叫Mini迷你的
3. 和更加小的，叫做Micro微小的

其中每一种大类中，又都可以分为两类

1. A类（Type A）
2. B类（Type B）

下面就用表格的形式，详细对比USB的各种接口，包括对应的插头和插座：

表 2.4. USB接口分类

USB接口（插头）概览	大的分类	细分	特点	插头图片示例	对应的插座	常见用途
	Type	Type A	长方形			普通PC端
		Type B	内部正方近乎形，外部是梯形			USB设备的接口

USB接口（插头）概览	大的分类	细分	特点	插头图片示例	对应的插座	常见用途
	Mini	Mini A	小型版的梯形			数码相机，移动硬盘等移动设备
		Mini B	小型版的长方形			
	Micro	Micro A	比Mini更扁			手机等移动设备
		Micro B				

提示

目前多数手机厂商已宣布统一使用Micro USB接口作为手机充电器标准接口。

2.2. USB相关的软件

如果某个USB设备正常工作，除了对应的硬件之外，还需要对应的软件支持。

2.2.1. USB设备端的固件（Firmware）

而对于USB设备端来说，内部是需要有对应的设备端的驱动，常常称其为固件Firmware，其实现了对应的设备端的USB所要做的事情，主要是相应一些标准的请求，完成对应的数据读取和写入等。

2.2.2. USB主机（Host）端的USB驱动和软件

对应的，主机Host端，也需要对应的驱动，此部分驱动，不论是Linux下，还是Windows下，都已经实现了常见的驱动了，所以一般来说，很少需要驱动开发者再去写相关的驱动。

2.2.3. 其他一些USB测试和协议分析等软件

在设备驱动开发阶段或者USB出现问题，需要调试的时候，往往就需要一些调试工具了。

一般来说，都包含了对应的USB硬件测试工具，加上对应的软件工具，去捕获对应USB总线上的数据，即所谓的USB抓包，然后再去分析抓取出来的数据，是否是期望的，是否符合USB协议的规范定义。

我所见过的一些USB抓包工具有：

1. USB Explorer 260 + Ellisys USB Analysis Software
   Ellisys的USB Explorer 260硬件，加上对应的USB软件Ellisys USB Analysis Software，实现USB数据抓包和分析

   Ellisys USB Explorer 260的外观是这样的：

图 2.1. USB协议分析工具：Ellisys的USB Explorer 260

   TODO：

   Ellisys USB Explorer 260的对应功能特点，可以参考：

   http://www.ellisys.com/products/usbex260/features.php#usbex260g

   从：

   http://www.ellisys.com/support/download.php

   可以看到截图：

   ->

   http://www.ellisys.com/products/usbex260/download.php

   可以看到各种下载。

2. Catalyst Enterprises公司的硬件，加上对应的软件SBAE USB，实现USB数据捕获和分析

另外，当然也有一些USB开发相关的工具，比如：

1. usbview
   用于查看USB设备的详细信息
2. USB20CV
   用来做USB兼容性测试的工具，具体工具下载和详细解释可以去USB官网[3]找到。
3. bus hound
   好像是个纯软件的USB抓包工具，据说不支持USB枚举过程的抓包。具体没用过，只是听说过

[3] http://www.usb.org/developers/tools/

第 3 章 USB协议概览

3.1. USB 2.0协议内容概览

USB协议，由于涉及内容太多，所以在此一个文档中解释清楚，是不现实的。

此处能做的和要做的，就是对于USB协议简明地介绍一下关于USB本身协议部分的内容。

当前最新的USB协议，已经发展到USB 3.0了。但是主流的USB设备和技术，还是以USB 2.0居多。所以此文，主要是以USB 2.0为基础来解释USB协议的基础知识，当然，会在相关内容涉及到USB 3.0的时候，也把USB 3.0的相关内容添加进来。

关于USB 2.0和USB 3.0等USB的协议规范，可以去官网下载：

http://www.usb.org/developers/docs/

其实，说实话，不论是谁，如果开始看USB协议的时候，发现单独对于USB 2.0规范本身这一个文档来说，竟然都有650页，而如果再加上，新的USB 3.0规范的482页，和其他一些辅助的USB相关的规范定义等文档，即使你是英语为母语的人，如果要看完这么多页的协议规范，估计也会吐的，更别说我们中国人了。

所以，此处，就来简单以USB 2.0规范为例，分析一下，具体其都主要包含哪些内容，然后你会发现，其实和USB协议本身相关的内容，相对则不会那么多，大概只有97页左右的内容，是我们所要关心的。

下面就来分析看看，USB 2.0的规范中，具体都包含了哪些内容：

表 3.1. USB 2.0协议的内容组成

章节	名称	内容描述	页数
1	介绍	介绍了为何要有USB以及USB协议内容的涵盖范围。此章节最重要的信息就是，引用了USB Device Class规范。不需要看。	2
2	术语和缩写	名词解释，一般的协议都会有这一章节的。无需看。	8
3	背景	说明了USB的来由，以及目的是为了是USB的用户，注意不是为了是USB的开发者，更加容易使用。介绍了Low，Full，High Speed三种不同的速度以及对应的应用领域。所以也不需要看。	4
4	系统架构综述	可以从这章开始看。此章介绍了USB系统的基本架构，包括拓扑关系，数据速度，数据流类型，基本的电气规范。	10
5	USB数据流模型	此章开始介绍USB中数据是如何流向的。其先介绍了端点和管道，然后对控制，中断，等时，批量四种传输类型进行了详细阐述。其中，重要的一点是，要搞懂每种传输类型，当然，这对于初学者来说可能会有那么一点难。	60
6	机械的	此章详细介绍了USB的两种标准的连接头，即接口的类型，其中需要了解的一点是，A类接口旨在用于数据向下流的（downstream），而B类接口旨在用于数据向上流的（upstream）。因此，你应该知道，不应该也不可能去将一个USB线，连到两个都是upstream的端口上。而所有的full或high speed的USB线，都是可拔插的，而低速的USB线，应该是焊死的。如果你不是USB接口的制造商，那么就没必要细看这章，而只需要大概浏览一下其中关于USB的接口类型的相关内容即可。	33
7	电子的	此章详解了USB总线上的电子信号，包括线阻，上下沿的时间，驱动者和接受者的规范定义，以及比特位编码，比特位填充等。此章中需要知道的，更重要的一点是，关于使用电阻在数据线上的偏压，去实现USB设备的速度类型检测，以及设备是总线供电还是自供电。除非你是在晶元级别上设计USB数据收发模块的相关人士，	75

章节	名称	内容描述	页数
		否则都可以直接跳过此章节。而正常的USB设备的数据手册中，都会有相关的解释，说明关于USB总线阻抗需要匹配电阻的阻值是多少。	
8	协议层	此章，从字节的级别，解释了USB数据包的细节，包括了同步，PID，地址，端点，CRC域。多数的开发人员都还没注意到这部分的底层的协议层，因为USB的设备中的硬件IC，会帮你做这些事情的。然而，多学习和了解一些关于报告状态和握手协议方面的知识，还是有必要的。。	45
9	USB设备框架工作[1]	此章，是整个USB协议中，用到的最多的一章。此章详细阐述了USB总线枚举的过程，以及一些USB Request的详细语法和含义，比如set address，get descriptor等，这些相关内容在一起，就构成了最常用的USB的协议层，也是通常USB编程人员和开发者所看到的这一层。此章节，必须详细阅读和学习。	36
10	USB主机的硬件和软件	此章介绍了和USB Host相关的知识。包括了数据帧frame和微帧microframe的产生，主机控制器的需求，软件机制和USB的驱动模型等。如果你不是去设计USB Host的话，那么就直接跳过此章即可。	23
11	Hub规范	此章定义了USB Hub相关的规范，包括了Hub的配置，分离传输，Hub类的标准描述符等。同理，如果你不是去设计USB Hub，那么也可以忽略此章。	143

提示

1. 关于第九章=chapter 9=ch9，多说明一下，由于其特殊性，特殊在于大部分和USB协议相关的内容，都在此章节内

   所以，你会在其他地方看到有关此ch9的说法。比如Linux源码中关于USB协议实现的部分的代码，会看到有对应的头文件是
   include\linux\usb\ch9.h

   此文件，就是指的是USB规范中的chapter 9，第九章。

   这也意味着，以后其他人如果谈及USB的话，说到第九章，指的就是此USB规范中的chapter 9，因为其包含了USB协议的软件实现所有关的多数的内容。

所以，由上述总结，我们可以看出：

对于只是为USB外设开发驱动的开发者的话，那么有关的章节只有：

- 4 系统架构综述
- 5 USB数据流模型
- 9 USB设备框架工作
- 10 USB主机的硬件和软件

如果是对于USB外设的电子设计研发人员，有关系的章节有：

- 4 系统架构综述
- 5 USB数据流模型
- 6 机械的
- 7 电子的

这么一来，如果是打算做USB设备驱动开发的话，其实我们要看的，只是USB 2.0规范中的一部分，大概有10+60+36+23=129页，所以，相对来说，还算能够接受，至少比要看那600页，要少了很多。

当然，这其中的内容，也还是不少。

而下面这些内容，就是将其中最基本的内容，精简出来，以方便想要快速了解USB基础知识的人，尽快地，更加清晰地，了解到USB的基础知识。

3.2. USB协议的版本和支持的速度

USB协议，也像其他协议一样，经历过很多个版本，但是正式发布出来的，主要有4个。

其中，从开始的USB 1.1，发展到后来的USB 2.0，以及最新的协议版本是USB 3.0.

不过这三个版本都是针对的是有线的（corded）设备来说的，在USB 2.0和USB 3.0之间，发布过一个是针对无线设备的USB协议，叫做USB Wireless，也被称为USB 2.5。

其中，USB 1.1中所支持的速度是低速（Low Speed）的1.5Mbits/s，全速（Full Speed）的12Mbits/s，而USB 2.0提高了速度至高速（High Speed）的480Mbits/s，而最新的USB 3.0，支持超高速（Super Speed）的5Gbits/s。

下面简要总结一下，各个USB协议版本的演化历史：

表 3.2. USB协议的版本的演化

USB协议		针对的设备	对应的速度		备注
版本号	发布日期		名称	速率	
1.1	1998年8月	有线的	Low Speed Full Speed	1.5Mbits/s=192KB/s 12Mbits/s=1.5MB/s	
2.0	2000年4月	有线的	High Speed	480Mbits/s=60MB/s	
2.5	2010年9月	无线的			Wireless USB 1.1
3.0	2008年11月	有线的	Super Speed	5.0Gbits/s=640MB/s	

3.2.1. 为何USB的速度，最开始没有设计的更快些？

有人会问，既然USB技术本身可以设计成速度更快的，为何最开始不把USB的设计成速度更快的呢？比如，最开始为啥没有把USB设计成2.0的那样的速度呢？

那是因为，任何规范和协议，都离不开当时的背景。关于USB的速度发展，其有其自身的考虑。

比如最开始的USB 1.1，对于低速的1.5Mbits/s的速度，虽然速度很低，但是由于此速度，主要用于USB鼠标，键盘等低速设备，所以本身就够用了，而且速度低还有个好处，那就是对于电磁辐射EMI的抗干扰能力较强些，而使得设计和制造对应的硬件设备的成本要降低些，比如可以使用相对便宜的陶瓷振荡器（resonator）做晶振（crystal）。

而后来的USB 2.0,的出现，则是为了满足人民群众日益增长的对于高速速度传输方面的需求，比如你从MP3里面拷贝歌曲出来，如果是USB 1.1，那么实际效果最快也就1MB左右，而如果是USB 2.0，平均效果大概有3MB/s,5MB/s，性能好的可达10MB/s，20MB/s，所以，如果拷贝个1G的东西，相当于USB 1.1要1小时左右，而USB 2.0只要1分钟左右。因为如果没有USB 2.0的出现的话，那么现在的人们，早就放弃了USB了，因为谁也忍受不了这个太慢的速度。所以为了满足大家的需求，才有了USB 2.0的出现。

而对于最新的USB 3.0，同理，也是为了满足现在的一些及以后的可能的需求，即希望拷贝蓝光光碟的内容到硬盘上，动辄都是几个G的内容，以USB 2.0的速度，那怎么说也得个几分钟，而有了USB 3.0后，就有望实现，几秒或者几十秒，哗的一下，就把多少个G的东西，拷贝传输到别的介质上了。当然，这只是理论上的，实际的USB 3.0的速度，受到USB设备的硬件本身能力，和对应的软件驱动，以及所设计的介质不同，而会有不同的速度。

3.3. USB系统的核心是Host

USB是Host端控制整个的总线数据传输的。单个USB总线上，只能有一个Host。USB协议本身，是不支持多个Host端的。

不过，相对有点特殊的是，USB为了支持多个设备互相，而不需要另外接Host，比如一个数码相机和一个打印机，希望把打印机和数码相机直接相连接，然后就可以实现通过USB，把数据从数码相机传送到打印机中，打印机就可以打印了。

此处，OTG引入了一个新的概念，HNP（Host Negotiation Protocol），主机协商协议，允许两个设备之间互相协商谁去当Host。不过，即使在OTG中，也只是同一时刻，只存在单个的Host，而不允许存在多个Host的。

USB中的Host端，负责所有底层的数据传输的控制以及数据带宽的安排调度。

底层数据的发送，是使用一种基于令牌环的协议，通过不同类型的传输方法而实现的。

USB的设备连接方式，即拓扑方式，是星形的，所以，在需要连接多个设备的时候，常需要用到对应的Hub。这种星形拓扑的好处是，对于每一个设备的供电情况，都可以控制，也可以同时支持多个不同类型的设备。而万一某设备电流过高了，也不会影响到其他设备。

同一USB总线中，最多可连接127个设备。当然，实际中你所看到的，往往都比较少，至少普通用户用到的，也就是几个的数量级。常见的是一个USB Host，提供，2,3个或者4,5个接口。而USB Host的内部实现，也有最开始的只有1个USB主控Host Controller，变得有些内部包含2个甚至更多个USB主控，以提高所接的USB设备的带宽，即速率，因为单个USB主控所接多个USB设备的话，那么多个设备是共享此USB主控的带宽的。

3.4. USB中用NRZI来编码数据

之前已经介绍过了USB的引脚定义了，但是对于其中的USB 2.0的两根数据线D+和D-所对应的数据传输，却没有详细介绍。此处就是介绍，在此串行数据线中，数据是如何被编码和传送的。

USB所传输的数据，用的数据编码方式是NRZI（Non-Return-to-Zero Inverted），其具体的含义解释，个人觉得某人解释的很清楚，所以在此转载其对应的这篇文章：

USB 的 NRZI 编码 [1]

这两天继续看 USB 相关的内容，准备用纯软件实现一下 USB 设备传输，为将来的项目打好基础。

首先碰到的就是这个 NRZI 编码的问题了，基础太薄弱，看了一上午总算明白了大概。

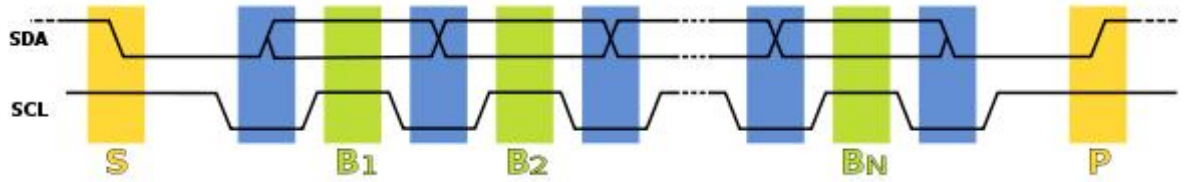
首先，USB 的数据是串行发送的，就像 UART、I2C、SPI 等等，连续的01 信号只通过一根数据线发送给接受者。

但是因为发送者和接收者运行的频率不一样，信号的同步就是个问题，比如，接受者接收到了一个持续一段时间的低电平，无法得知这究竟是代表了5个0 还是1000个0。

一个解决办法，就是在传输数据信号的同时，附加一个时钟信号，用来同步两端的传输，接受者在时钟信号的辅助下对数据信号采样，就可以正确解析出发送的数据了，比如 I2C 就是这样做的，SDA 来传输数据，SCL 来传输同步时钟：

[1] http://galeki.is-programmer.com/posts/10054.html

图 3.1. I2C数据编码格式

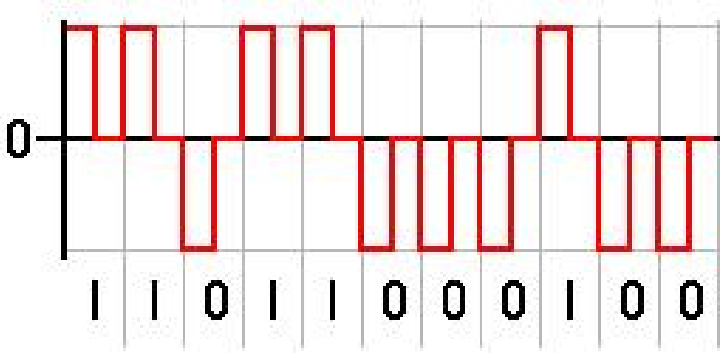


虽然这样解决了问题，但是却需要附加一根时钟信号线来传输时钟。有没有不需要附加的时钟信号，也能保持两端的同步呢？

有的，这就是 RZ 编码（Return-to-zero Code），也叫做归零编码。

在 RZ 编码中，正电平代表逻辑 1，负电平代表逻辑 0，并且，每传输完一位数据，信号返回到零电平，也就是说，信号线上会出现 3 种电平：正电平、负电平、零电平：

图 3.2. 归零编码

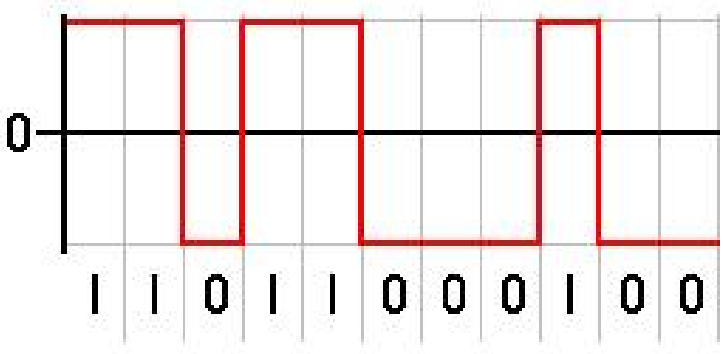


从图上就可以看出来，因为每位传输之后都要归零，所以接收者只要在信号归零后采样即可，这样就不在需要单独的时钟信号。实际上， RZ 编码就是相当于把时钟信号用归零编码在了数据之内。这样的信号也叫做自同步（self-clocking）信号。

这样虽然省了时钟数据线，但是还是有缺点的，因为在 RZ 编码中，大部分的数据带宽，都用来传输“归零”而浪费掉了。

那么，我们去掉这个归零步骤，NRZ 编码（Non-return-to-zero Code）就出现了，和 RZ 的区别就是 NRZ 是不需要归零的：

图 3.3. 非归零编码

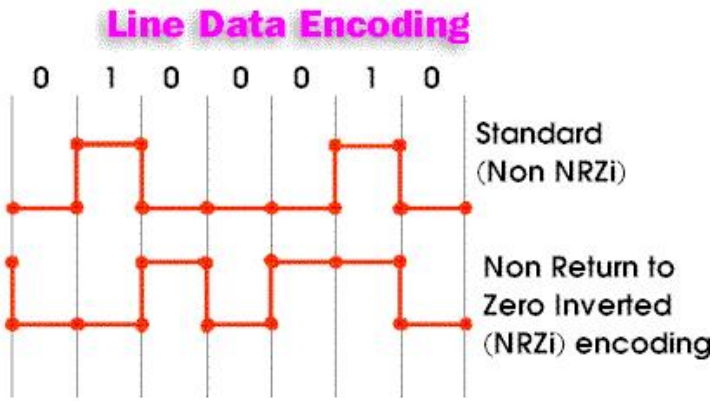


这样，浪费的带宽又回来了，不过又丧失宝贵的自同步特性了，貌似我们又回到了原点，其实这个问题也是可以解决的，不过待会儿再讲，先看看什么是 NRZI：

NRZI 编码（Non-Return-to-Zero Inverted Code）和 NRZ 的区别就是 NRZI 用信号的翻转代表一个逻辑，信号保持不变代表另外一个逻辑。

USB 传输的编码就是 NRZI 格式，在 USB 中，电平翻转代表逻辑 0，电平不变代表逻辑1：

图 3.4. NRZ和NRZI



翻转的信号本身可以作为一种通知机制，而且可以看到，即使把 NRZI 的波形完全翻转，所代表的数据序列还是一样的，对于像 USB 这种通过差分线来传输的信号尤其方便~

现在再回到那个同步问题：

的确，NRZ 和 NRZI 都没有自同步特性，但是可以用一些特殊的技巧解决。

比如，先发送一个同步头，内容是 0101010 的方波，让接受者通过这个同步头计算出发送者的频率，然后再用这个频率来采样之后的数据信号，就可以了。

在 USB 中，每个 USB 数据包，最开始都有个同步域（SYNC），这个域固定为 0000 0001，这个域通过 NRZI 编码之后，就是一串方波（复习下前面：NRZI 遇 0 翻转遇 1

此外，因为在 USB 的 NRZI 编码下，逻辑 0 会造成电平翻转，所以接收者在接收数据的同时，根据接收到的翻转信号不断调整同步频率，保证数据传输正确。

但是，这样还是会有一个问题，就是虽然接收者可以主动和发送者的频率匹配，但是两者之间总会有误差。

假如数据信号是 1000个逻辑1，经过 USB 的 NRZI 编码之后，就是很长一段没有变化的电平，在这种情况下，即使接受者的频率和发送者相差千分之一，就会造成把数据采样成 1001个或者 999个了。

3.4.1. USB中用Bit-Stuffing来同步时钟信号

USB对这个问题的解决办法，就是强制插0，也就是传说中的bit-stuffing，如果要传输的数据中有7个连续的1，发送前就会在第6个1后面强制插入一个0，让发送的信号强制出现翻转，从而强制接受者进行频率调整。接受者只要删除6个连续 1 之后的0，就可以恢复原始的数据了。

第 4 章 USB协议细节

4.1. USB Class

关于USB的Class，对于学习USB协议的人，估计早就听到过此名词了。

而对于USB的Class的分类，此处先列出那个最基本的分类表：

表 4.1. USB Class表

Base Class	Descriptor Usage	Description
00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Fh	Interface	Personal Healthcare
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

4.1.1. 为何要搞这么多USB的Class

其实，看到表 4.1 “USB Class表” 中的一堆的Class分类，一般人，都会晕的。

所以，就要简单解释一下，为何会有这么多USB的Class分类。

首先我们要知道，USB协议设计的目的，就是为了第 1.2 节 “为何要有USB” 中所提到的，用单一的USB接口，取代之前种类繁多的各种其他接口。

而为了取代其他各种接口，那意味着就要实现，或者是支持，之前别的接口，所对应的各种功能。

因此，USB协议设计的时候，就是要把鼠标，键盘，大容量存储，图像等，这些之前是通过其他接口所实现的，各种的功能，都囊括进来。并且在协议中有对应的规范定义，支持这些功能。

因此，才有了如此多的各种USB的Clas，即分类，根据功能而分出的各种类别。不同的Class分类，用于实现对应的功能，适用于相应的设备。

比如我们常见的鼠标和键盘，都属于Class 3的HID，U盘属于Class 8的Mass Storage等。

而关于这些分类，每种分类都对应着哪些具体的应用和功能，感兴趣的可以去参考USB classes[1]中的Overview of the various USB classes[2]，该页面，相对形象地列出了各种Class所对应的应用。

再回到上述的表 4.1 “USB Class表”，其中的“Descriptor Usage”中的Interface，Device等，对应的是来自表 4.2 “USB Descriptor Type”

表 4.2. USB Descriptor Type

Descriptor Type	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5
DEVICE_QUALIFIER	6
OTHER_SPEED_CONFIGURATION	7
INTERFACE_POWER	8

而关于什么是Device，什么是Interface等内容，暂时先简述他们之间的关系：

Device ⇒ Configuration ⇒ Interface ⇒ Endpoint

更新详细内容，待以后有空再深入解释。

4.2. USB的框架

对于USB的框架，暂时先贴出一些相关图片，细节的解释，有空再写。

[1] http://www.xat.nl/en/riscos/sw/usb/
[2] http://www.xat.nl/en/riscos/sw/usb/class.htm

图 4.1. USB Implementation Areas

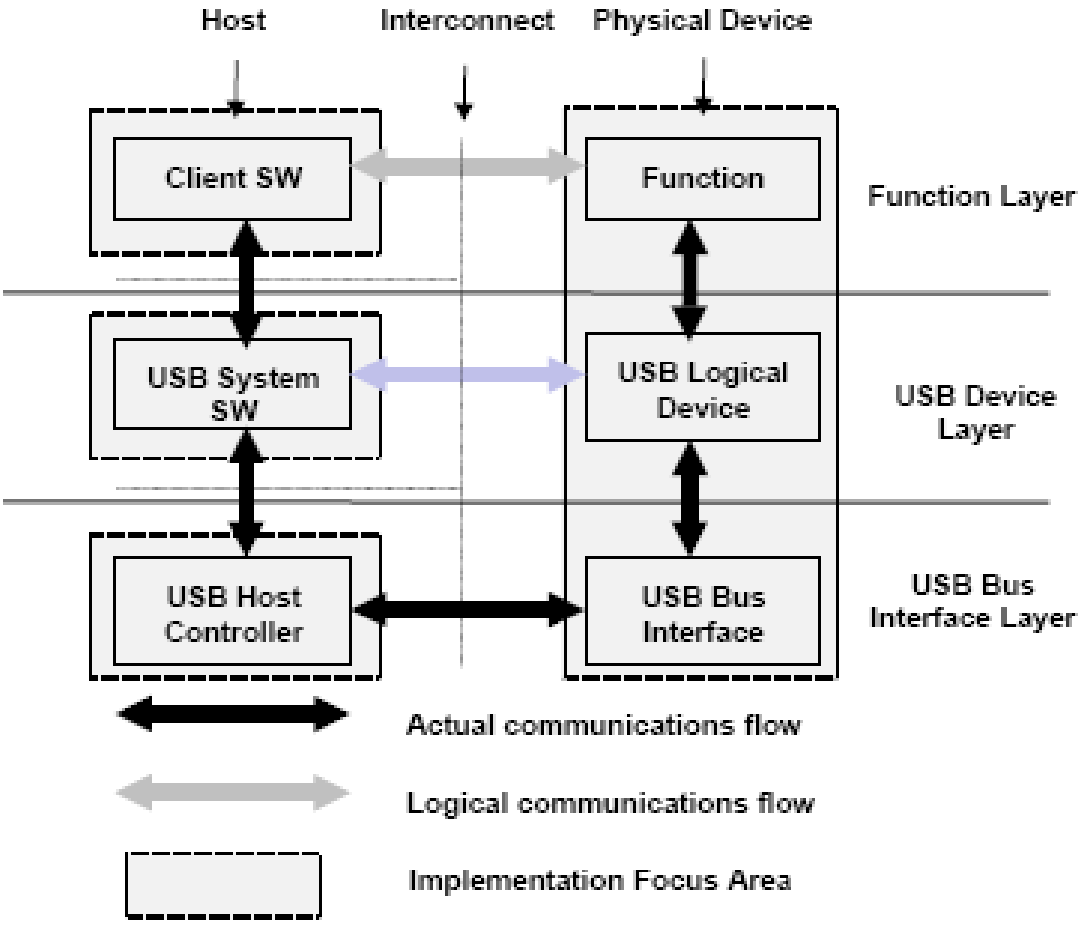


Figure 5-2. USB Implementation Areas

图 4.2. USB Physical Bus Topology

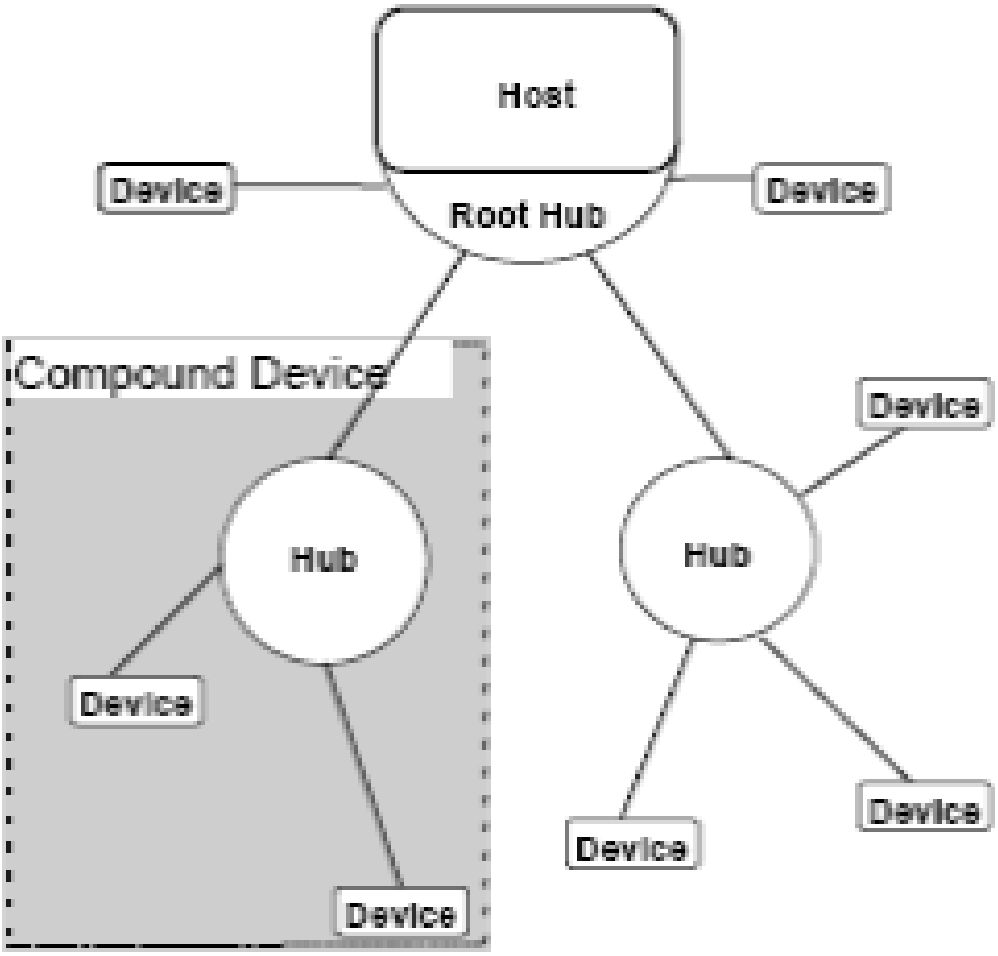


Figure 5-5. USB Physical Bus Topology

图 4.3. USB Logical Bus Topology

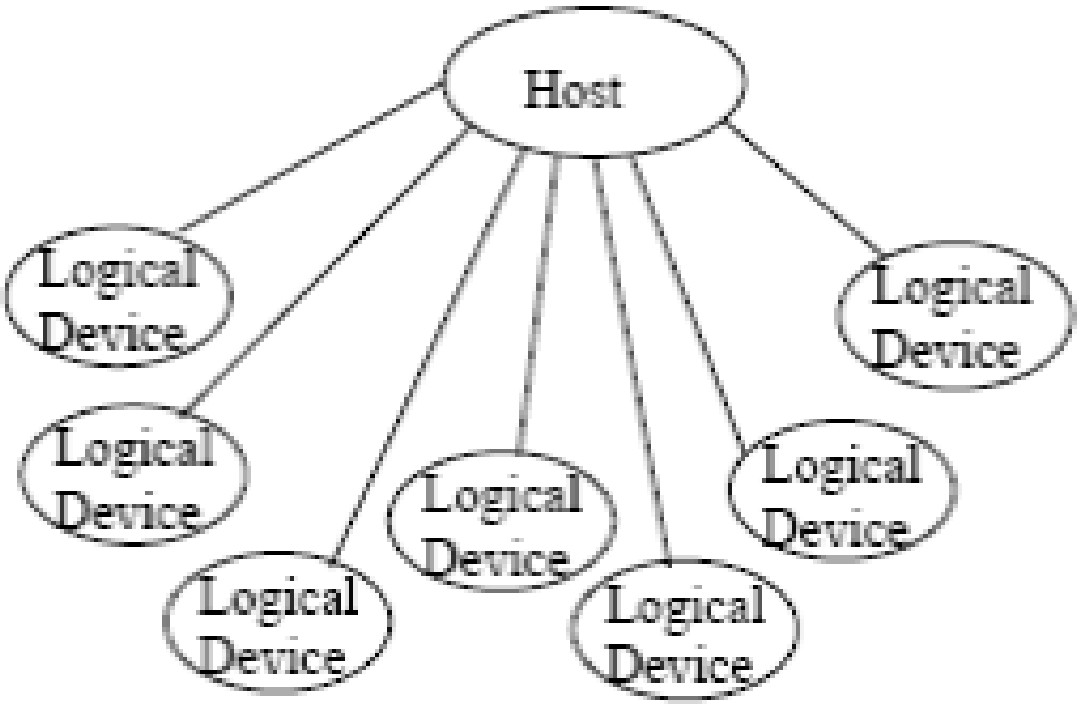


Figure 5-7. USB Logical Bus Topology

图 4.4. USB Communication Flow

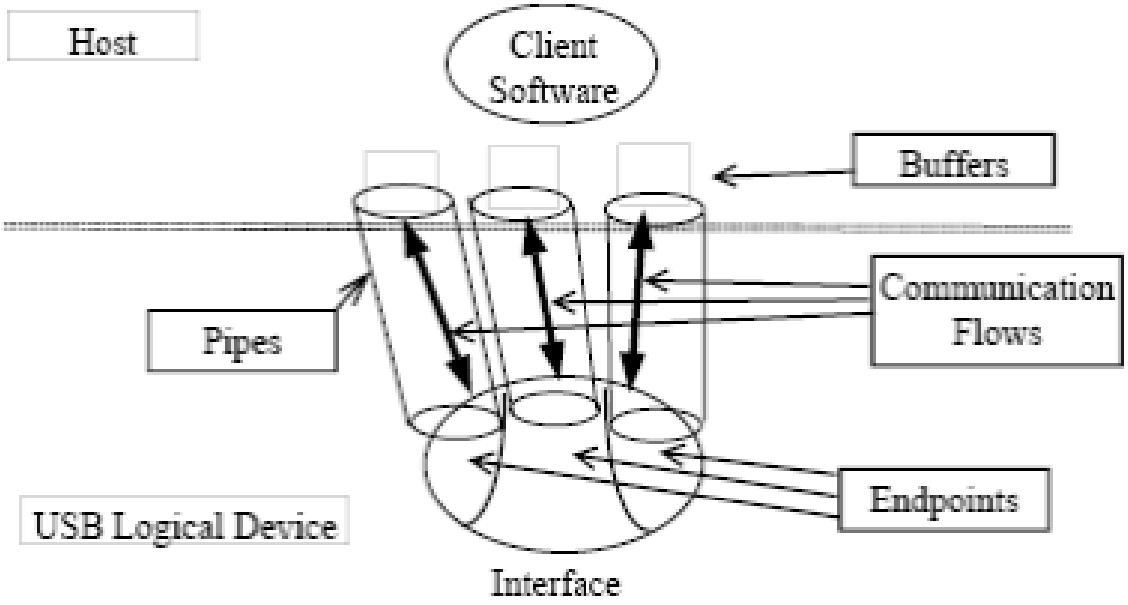
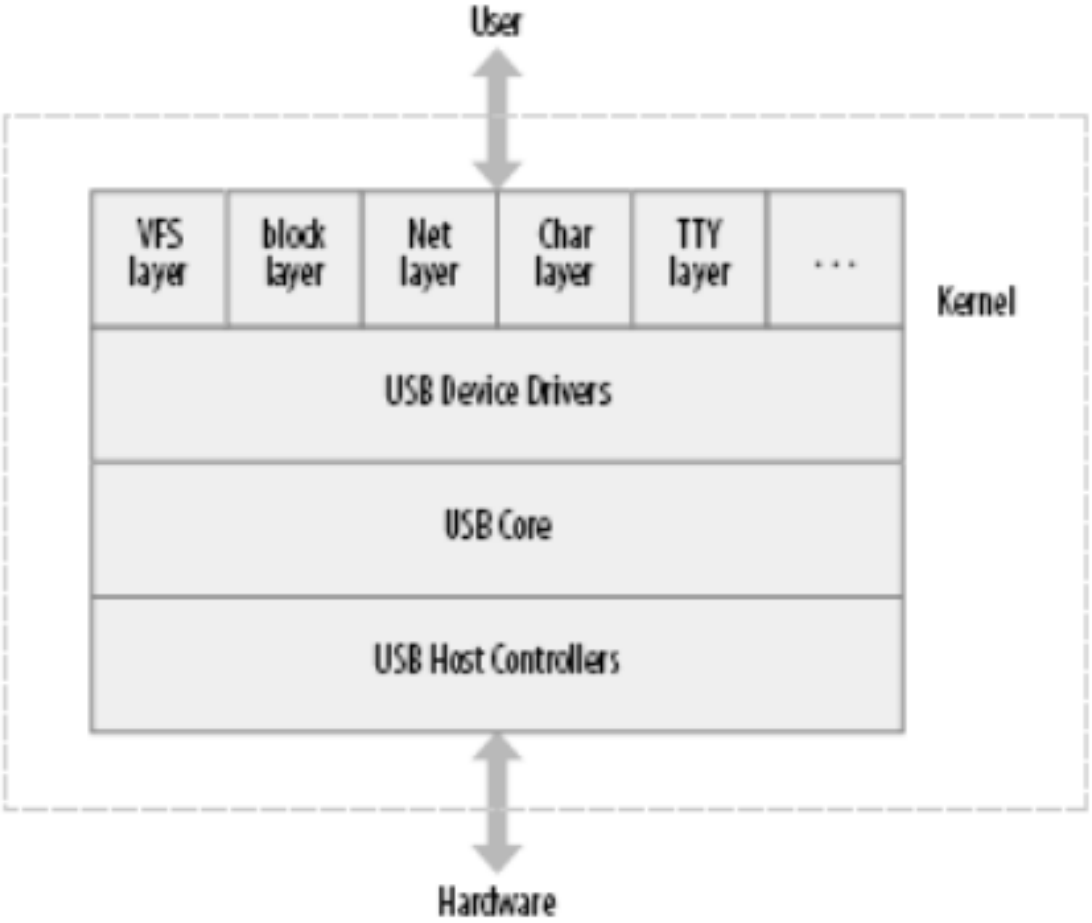


Figure 5-10. USB Communication Flow

图 4.5. USB Layers in Linux



4.3. USB Transfer and Transaction

图 4.6. USB Transfer and Transaction

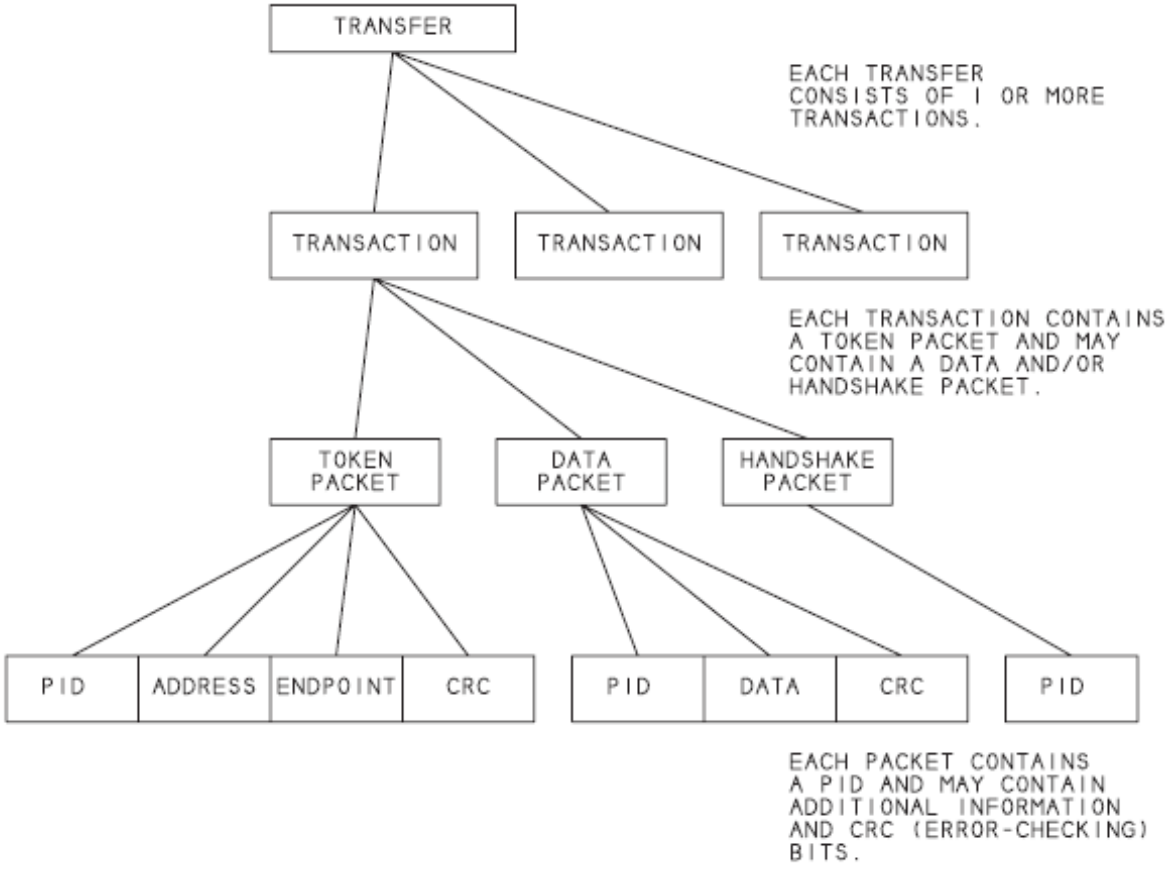


Figure 2-3: A USB transfer consists of transactions. The transactions in turn contain packets, and the packets contain a packet identifier (PID), PID-check bits, and sometimes additional information.

4.4. USB枚举（Emulation）

关于USB的枚举，是学习USB协议所面临的第一个最基础也是最重要的内容。

4.4.1. 什么是USB枚举

USB枚举，USB Emulation，从字面意思看，就是去列举USB，而列举啥呢，其实就是USB的初始化。

简单来说，USB的枚举，对应的就是USB的Host和Device之间的对话，即Host根据Device所报告上来的参数，得知USB的device是啥类型的，具有啥功能，然后初始化相关参数。

接下来，就USB Device就可以正常工作了。

所以，可以简单的理解为，USB枚举，就是USB设备的初始化（init）。

4.4.2. USB枚举的过程

此处摘录一个，[12]中的关于windows下USB枚举的过程的总结：

1. The host or hub detects the connection of a new device via the device's pull up resistors on the data pair. The host waits for at least 100ms allowing for the plug to be inserted fully and for power to stabilise on the device.
2. Host issues a reset placing the device is the default state. The device may now respond to the default address zero.
3. The MS Windows host asks for the first 64 bytes of the Device Descriptor.
4. After receiving the first 8 bytes of the Device Descriptor, it immediately issues another bus reset.
5. The host now issues a Set Address command, placing the device in the addressed state.
6. The host asks for the entire 18 bytes of the Device Descriptor.
7. It then asks for 9 bytes of the Configuration Descriptor to determine the overall size.
8. The host asks for 255 bytes of the Configuration Descriptor.
9. Host asks for any String Descriptors if they were specified.

   At the end of Step 9, Windows will ask for a driver for your device. It is then common to see it request all the descriptors again before it issues a Set Configuration request.

而相对来说，更加详细一点的解释，可以去看Enumeration: How the Host Learns about Devices[3]

其实，单独看此文字描述，虽然解释的很是详细了，但是还是很难彻底搞懂。

所以，后面会专门通过第 4.4.3 节 “举例详解USB的枚举过程”来彻底的解析，到底USB的枚举的过程如何，以及发送的数据的详细所对应的含义。

4.4.3. 举例详解USB的枚举过程

既然说到举例，那么就要有相应的数据。

此处的USB枚举所涉及到的数据，是之前某次开发过程中，通过第 2.2.3 节 “其他一些USB测试和协议分析等软件”中所介绍的SBAE USB所抓取出来的数据，并且软件可以详细分析每个字节所对应的含义。

4.4.3.1. USB枚举示例数据

抓包工具抓到了共0x42=66字节的数据，其中每个字节对应的十六进制表示是两个数字，所以一共是66x2=132个数字：

```
0902420002010480E10904000002FF0000000921100100012 23F00070501034000010705810340000109040100
```

可被拆分为对应的8组：

- 0902420002010480E1
- 0904000002FF000000
- 092110010001223F00
- 07050103400001
- 07058103400001
- 090401000103000000

[3] http://www.lvr.com/usbcenum.htm

- 092110010001222100
- 0705820340000A

而关于为何可以被分成这8组，此处先解释一下：

如何（解析）看懂USB枚举的数据

对于Configuration，Interface，Endpoint，Class等部分，其数据格式的定义中，首字节，都是表示长度，即，接下来多少个字节，属于当前这部分。

所以，对于上述数据来说，从开始的“09”，我们就知道了，接下来的8个字节的数据“02420002010480E1”，都是属于当前Configuration部分的。

以此接着往下判断，则分别可以判断出对应的每一部分的数据，都是哪些。

而对于这些数据分组的依次顺序，则是USB协议中定义的。详细定义，请参考USB协议。

4.4.3.2. 详细分析USB枚举数据的每个字段的具体含义

接下来，就是分析，每一部分的数据，到底对应的是何种含义。

此处，再把上述数据贴出来，并进行分析：

```
0902420002010480E1❶
0904000002FF000000❷
092110010001223F00❸
070501034000001❹
070581034000001❺
0904010001030000000❻
0921100100012221007❼
0705820340000A❽
```

❶ 此部分内容对应着的是：Configuration

其定义为：

表 4.3. USB Configuration Descriptors

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	Configuration Descriptor (0x02)
2	wTotalLength	2	Number	Total length in bytes of data returned
4	bNumInterfaces	1	Number	Number of Interfaces
5	bConfigurationValue	1	Number	Value to use as an argument to select this configuration
6	iConfiguration	1	Index	Index of String Descriptor describing this configuration
7	bmAttributes	1	Bitmap	D7 Reserved, set to 1. (USB 1.0 Bus Powered) D6 Self Powered D5 Remote Wakeup D4..0 Reserved, set to 0.

Offset	Field	Size	Value	Description
8	bMaxPower	1	mA	Maximum Power Consumption in 2mA units

其每个字节对应的含义为：

图 4.7. Configuration Descriptor: 09024200020104B0E1

Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor is 9 bytes
1	bDescriptorType	1	0x02	CONFIGURATION Descriptor Type
2	wTotalLength	2	0x0042	Total length of data returned is 66 bytes
4	bNumInterfaces	1	0x02	Number of interfaces supported by this speed configuration is 2
5	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request is 1
6	iConfiguration	1	0x04	Index of string descriptor describing this configuration is 4
7	bmAttributes	1	0x80	Configuration characteristics: D7: Bus powered D6: No self powered D5: No remote wakeup D4..0: Reserved (reset to zero)
8	bMaxPower	1	0xe1	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational is 450 mA

❷ 此部分内容对应着的是：Interface

其定义为：

表 4.4. USB Interface Descriptors

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (9 Bytes)
1	bDescriptorType	1	Constant	Interface Descriptor (0x04)
2	bInterfaceNumber	1	Number	Number of Interface
3	bAlternateSetting	1	Number	Value used to select alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints used for this interface
5	bInterfaceClass	1	Class	Class Code (Assigned by USB Org)
6	bInterfaceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
7	bInterfaceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
8	iInterface	1	Index	Index of String Descriptor Describing this interface

其每个字节对应的含义为：

图 4.8. Interface Descriptor: 0904000002FF000000

Interface Descriptor				
Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor is 9 bytes
1	bDescriptorType	1	0x04	INTERFACE Descriptor Type
2	bInterfaceNumber	1	0x00	Number of this interface is 0
3	bAlternateSetting	1	0x00	Value used to select this alternate setting is 0
4	bNumEndpoints	1	0x02	Number of endpoints used by this interface is 2
5	bInterfaceClass	1	0xff	The device class is Vendor-Specific
6	bInterfaceSubClass	1	0x00	Reserved for future standardization
7	bInterfaceProtocol	1	0x00	Reserved for future standardization
8	iInterface	1	0x00	No string descriptor is available

❸ 此部分内容对应着的是：Class

由于其前面的❷中bInterfaceClass=0xFF，对应着表 4.1 “USB Class表” 中的vendor-specific，所以此部分的值的含义，是针对特定厂家的特定的含义，因此此处就不具体解释了。

❹ 此部分内容对应着的是：Endpoint

其定义为：

表 4.5. USB Endpoint Descriptors

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (7 bytes)
1	bDescriptorType	1	Constant	Endpoint Descriptor (0x05)
2	bEndpointAddress	1	Endpoint	Endpoint Address Bits 0..3b: Endpoint Number Bits 4..6b: Reserved. Set to Zero Bits 7: Remote Wakeup D4..D0: Direction 0: Out 1: In (Ignored for Control Endpoints)
3	bmAttributes	1	Bitmap	Bits 0..1: Transfer Type 00: Control 01: Isochronous 10: Bulk 11: Interrupt Bits 2..7: are reserved. If Isochronous endpoint: Bits 3..2: Synchronisation Type (Iso Mode)

<table>
<tr><th>Offset</th><th>Field</th><th>Size</th><th>Value</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td>
<table>
<tr><td></td><td>
<table>
<tr><td>00</td><td>No Synchonisation</td></tr>
<tr><td>01</td><td>Asynchronous</td></tr>
<tr><td>10</td><td>Adaptive</td></tr>
<tr><td>11</td><td>Synchronous</td></tr>
</table>
</td></tr>
<tr><td>Bits 5..4</td><td>Usage Type (Iso Mode)
<table>
<tr><td>00</td><td>Data Endpoint</td></tr>
<tr><td>01</td><td>Feedback Endpoint</td></tr>
<tr><td>10</td><td>Explicit Feedback Data Endpoint</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>wMaxPacketSize</td><td>2</td><td>Number</td><td>Maximum Packet Size this endpoint is capable of sending or receiving</td></tr>
<tr><td>6</td><td>bInterval</td><td>1</td><td>Number</td><td>Interval for polling endpoint data transfers. Value in frame counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1 to 255 for interrupt endpoints.</td></tr>
</table>

其每个字节对应的含义为：

图 4.9. Endpoint (Interrupt Out) Descriptor: 07050103400001

Interrupt OUT Endpoint 1 Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x07	Size of this descriptor is 7 Bytes
1	bDescriptorType	1	0x05	ENDPOINT Descriptor Type
2	bEndpointAddress	1	0x01	Bit 3..0: The endpoint number is 1 Bit 6..4: Reserved, Reset to zero Bit 7: Direction of this endpoint is OUT
3	bmAttributes	1	0x03	Bits 1..0: Transfer type is Interrupt Bits 7..2: Reserved, reset to zero
4	wMaxPacketSize	2	0x0040	Bits 10..0: Maximum packet size is 64 bytes Bits 12..11: Number of additional transaction opportunities per microframe is 0 Bits 15..13: Reserved, reset to zero
6	bInterval	1	0x01	Interval for polling endpoint for data transfers is 1 Frames / MicroFrames

❺ 此部分内容和❹类似，也是对应着：Endpoint

对应定义为：表 4.5 “USB Endpoint Descriptors”

其每个字节对应的含义为：

图 4.10. Endpoint (Interrupt In) Descriptor: 07058103400001

Interrupt IN Endpoint 1 Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x07	Size of this descriptor is 7 Bytes
1	bDescriptorType	1	0x05	ENDPOINT Descriptor Type
2	bEndpointAddress	1	0x81	Bit 3..0: The endpoint number is 1 Bit 6..4: Reserved, Reset to zero Bit 7: Direction of this endpoint is IN
3	bmAttributes	1	0x03	Bits 1..0: Transfer type is Interrupt Bits 7..2: Reserved, reset to zero
4	wMaxPacketSize	2	0x0040	Bits 10..0: Maximum packet size is 64 bytes Bits 12..11: Number of additional transaction opportunities per microframe is 0 Bits 15..13: Reserved, reset to zero
6	bInterval	1	0x01	Interval for polling endpoint for data transfers is 1 Frames / MicroFrames

❻ 此部分内容对应着的是：Interface

其定义为参考：表 4.4 “USB Interface Descriptors”

其每个字节对应的含义为：

图 4.11. Interface Descriptor: 090401000103000000

Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor is 9 bytes
1	bDescriptorType	1	0x04	INTERFACE Descriptor Type
2	bInterfaceNumber	1	0x01	Number of this interface is 1
3	bAlternateSetting	1	0x00	Value used to select this alternate setting is 0
4	bNumEndpoints	1	0x01	Number of endpoints used by this interface is 1
5	bInterfaceClass	1	0x03	The interface class is HID (Human Interface Device)
6	bInterfaceSubClass	1	0x00	The interface subclass is No Subclass
7	bInterfaceProtocol	1	0x00	The interface protocol is NONE
8	iInterface	1	0x00	No string descriptor is available

❼ 由于❻中bInterfaceClass=0x03，对应着表 4.1 “USB Class表” 中的HID，所以，此部分内容的解析，依赖于对应的HID中的定义。

可在官方的1.1版本的HID协议：HID1_11.pdf[4]中的“6.2.1 HID Descriptor”部分找到对应的定义：

表 4.6. USB HID Descriptors

Part	Offset/ Size(Bytes)	Description
bLength	0/1	Numeric expression that is the total size of the HID descriptor.
bDescriptorType	1/1	Constant name specifying type of HID descriptor.
bcdHID	2/2	Numeric expression identifying the HID Class Specification release.

[4] http://www.usb.org/developers/devclass_docs/HID1_11.pdf

Part	Offset/ Size(Bytes)	Description
bCountryCode	4/1	Numeric expression identifying country code of the localized hardware.
bNumDescriptors	5/1	Numeric expression specifying the number of class descriptors (always at least one i.e. Report descriptor.)
bDescriptorType	6/1	Constant name identifying type of class descriptor. See Section 7.1.2: Set_Descriptor Request for a table of class descriptor constants.
wDescriptorLength	7/2	Numeric expression that is the total size of the Report descriptor.
[bDescriptorType]	9/1	Constant name specifying type of optional descriptor.
[wDescriptorLength]	10/2	Numeric expression that is the total size of the optional descriptor.

其每个字节对应的含义为：

表 4.7. USB HID Descriptor: 090401000103000000

Offset	Field	Size	Value	Description
0	bLength	1	09	the total size of the HID descriptor =9 bytes
1	bDescriptorType	1	21	descriptor constant, HID = 0x21
2	bcdHID	2	1001	HID Class Specification release 0x0110= 1.10
4	bCountryCode	1	00	No country code of the localized hardware
5	bNumDescriptors	1	01	the number of class descriptors = 1
6	bDescriptorType	1	22	Class descriptor constant, 0x22 = Report descriptor
7	wDescriptorLength	2	2100	the total size of the Report descriptor =0x0021=33 bytes
9	[bDescriptorType]	2		
10	[wDescriptorLength]			

❽ 此部分内容和❺类似，也是对应着：Endpoint

对应定义为：表 4.5 “USB Endpoint Descriptors”

其每个字节对应的含义为：

图 4.12. Endpoint (Interrupt In 2) Descriptor: 0705820340000A

Interrupt IN Endpoint 2 Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x07	Size of this descriptor is 7 Bytes
1	bDescriptorType	1	0x05	ENDPOINT Descriptor Type
2	bEndpointAddress	1	0x82	Bit 3..0: The endpoint number is 2 Bit 6..4: Reserved, Reset to zero Bit 7: Direction of this endpoint is IN
3	bmAttributes	1	0x03	Bits 1..0: Transfer type is Interrupt Bits 7..2: Reserved, reset to zero
4	wMaxPacketSize	2	0x0040	Bits 10..0: Maximum packet size is 64 bytes Bits 12..11: Number of additional transaction opportunities per microframe is 0 Bits 15..13: Reserved, reset to zero
6	bInterval	1	0x0a	Interval for polling endpoint for data transfers is 10 Frames / MicroFrames

4.5. USB OHCI学习笔记

此处记录之前学习USB OHCI过程中所记录的一些心得：

1. OHCI主要应用于嵌入式等系统，因为其特点就是用硬件实现了尽可能多的功能，而使得USB软件驱动部分，相对要容易很多，相对更加统一，使用统一的接口。

   所以，OHCI spec里面写道，其目的就是为了推广USB，让大家更加接受USB。

2. 端点描述符（ED），一个放在内存里的数据结构，描述了主机控制器（HC）和设备之间如何交互。

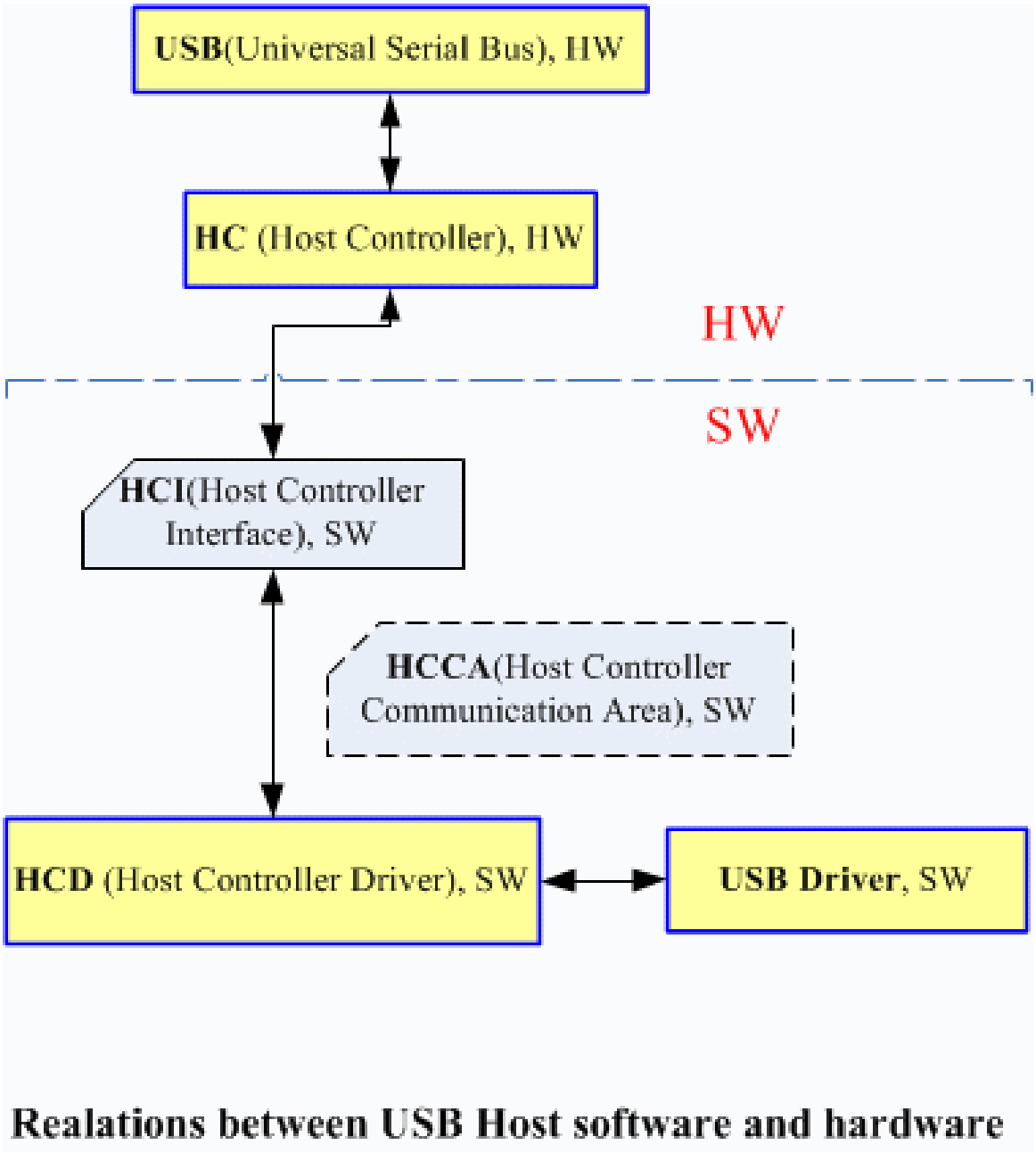
   ED里面包含了一个 TD（Transfer Endpoint，传输描述符）指针

3. 常见的缩写：

   - HC（Host Controller）
   - HCCA（Host Controller Communication Area）
   - HCD（Host Controller Driver）
   - HCDI（Host Controller Driver Interface）
   - HCI（Host Controller Interface）

4. 一些关系：

图 4.13. USB主机中软件和硬件之间的关系



5. Phase，意思可以理解为阶段。

   一个完整的数据传输(Transaction)分三个阶段。

   对应的，USB传输完成后的结果表示中，关于错误，就有个Phase Error，意思应该就是，其中某个阶段出错了。

6. 队列：传输描述符(TD，Transfer Descriptor)的列表

7. SOF，Start Of Frame。

   SOF的作用是让端点能够识别帧的开始，和同步内部端点的始终，使其和Host一致。

8. OHCI里面，数据传输分两类：周期性和非周期性。

周期性的包含，中断传输和等时传输，因为他们是以固定的间隔调度执行。

非周期性传输即是控制传输和大块(Bulk)传输。

9. HC和HCD之间通信的通道有2个：一堆寄存器和[可选的]HCCA：

图 4.14. USB Communication Channel

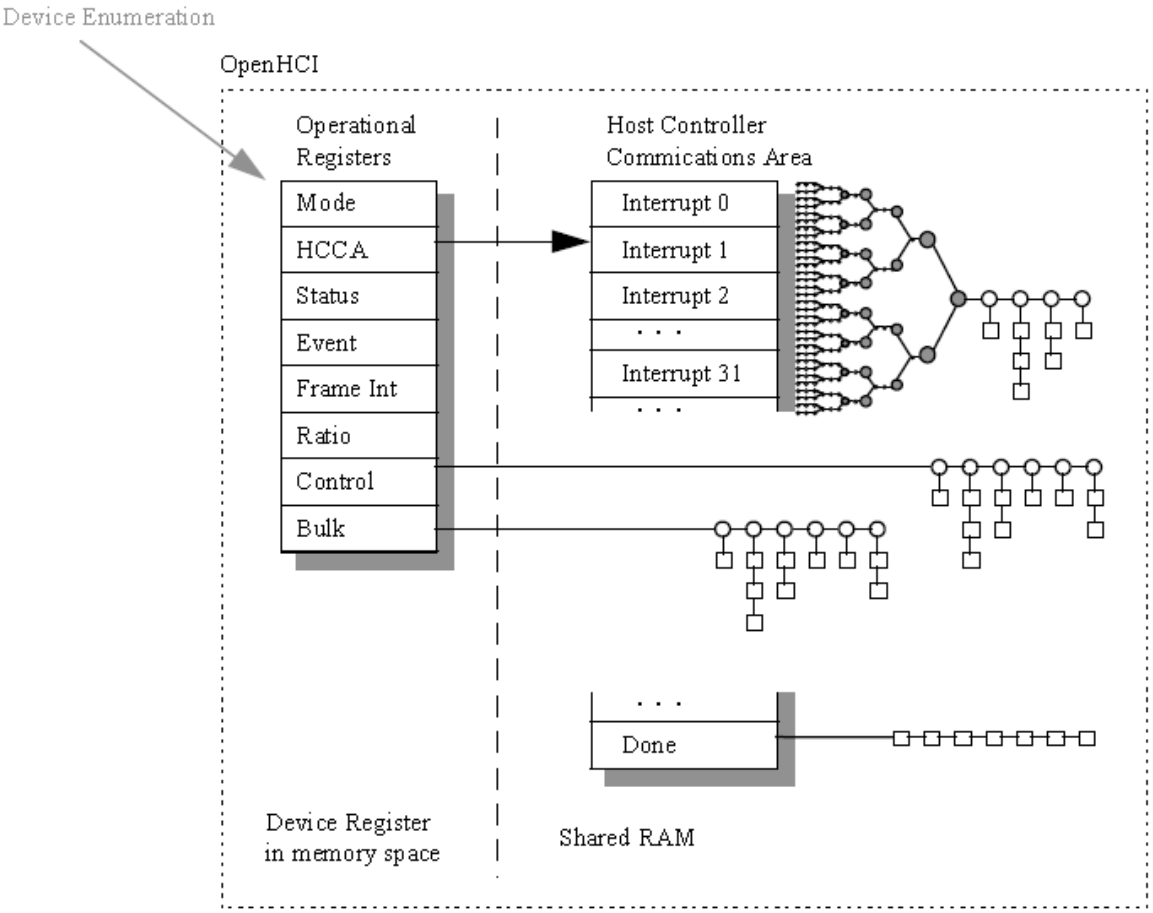


Figure 3-2: Communication Channels

10. HCCA里有个头指针，指向一个ED链表。其中每一个ED里面，包含一个TD链表，有一个或多个要被处理的TD。处理原则是，谁先到，先处理谁。

图 4.15. USB Typical List Structure

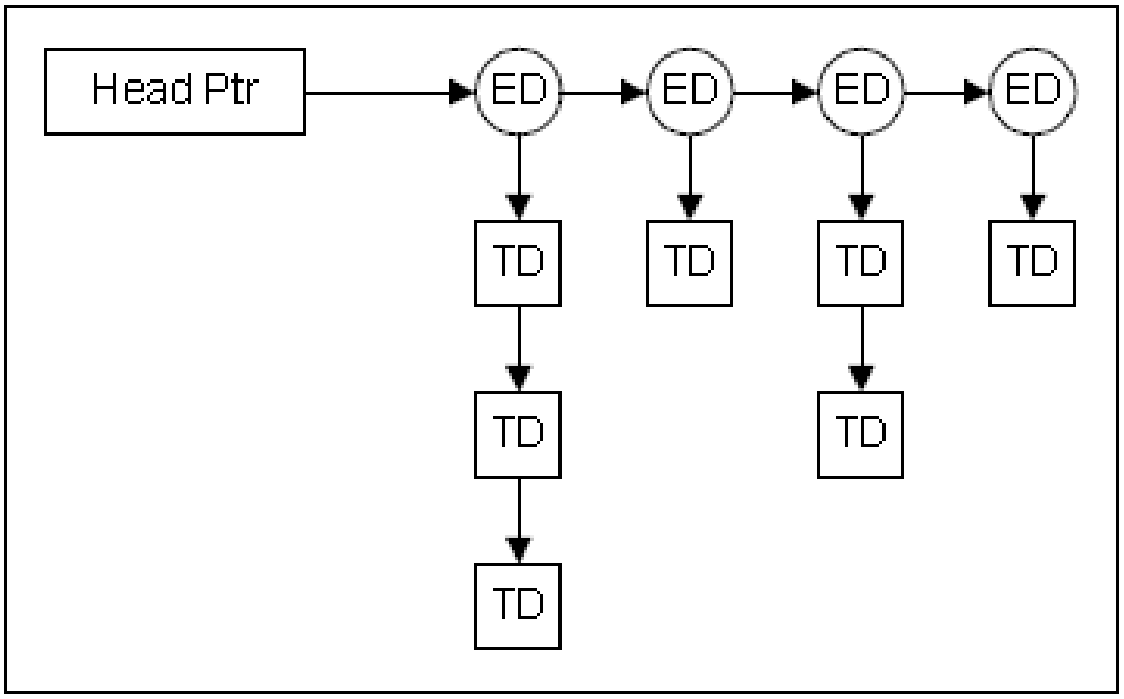


Figure 3-3: Typical List Structure

11. 大块传输和控制传输的ED链表的头指针，是放在HC的操作寄存器里面的；

    中断传输的ED链表的头指针是放在HCCA里面的。

    没有相应的等时传输的ED链表的头指针。第一个等时传输的ED链表，直接指向最后一个中断传输的ED。

参考书目

[1] 官方的USB 2.0规范[1]

[2] USB in a Nutshell - Making sense of the USB standard[2]

[3] USB Complete[3]

[4] 【很好的学习Linux驱动的教材】Linux那些事儿系列[全][pdf][4]

[5] USB接口[5]

[6] USB[6]

[7] USB 2.0 A型、B型、Mini和Micro接口定义及封装[7]

[8] Wireless USB Documents[8]

[9] USB的NRZI编码[9]

[10] USB 3.0[10]

[11] USB in a NutShell - The Setup Packet[11]

[12] Windows enumeration steps[12]

[1] http://www.usb.org/developers/docs/usb_20.zip
[2] http://www.beyondlogic.org/usbnutshell/usb1.shtml
[3] http://www.lvr.com/usbc.htm
[4] http://bbs.chinaunix.net/thread-1977195-1-1.html
[5] http://zh.wikipedia.org/zh/USB
[6] http://en.wikipedia.org/wiki/USB
[7] http://www.metsky.com/archives/474.html
[8] http://www.usb.org/developers/wusb/docs/
[9] http://galeki.is-programmer.com/posts/10054.html
[10] http://en.wikipedia.org/wiki/USB_3.0
[11] http://www.beyondlogic.org/usbnutshell/usb6.shtml
[12] http://www.8052.com/users/knowledgebase/usb-in-a-nutshell.pdf