USB 例程指南（基础篇）

目录

一 USB 基础讲解

1.1 USB 简介

USB ,是英文 Universal Serial Bus（通用串行总线）的缩写，而其中文简称为“通串线”，是一个外部总线标准，用于规范电脑与外部设备的连接和通讯。是应用在 PC 领域的接口技术。与传统计算机接口相比，它克服了对硬件资源独占，限制对计算机资源扩充的缺点，以较高的数据传输速率和即插即用等优点，逐步发展成为计算机与外设的标准连接方案。

USB 接口的优点有如下几点：

（1）USB 为所有的 USB 外设提供了单一的易于使用的标准的连接类型。

（2）整个的 USB 的系统只有一个端口和一个中断节省了系统资源。

（3）USB 支持热插拔(hot plug)和 PNP(Plug-and-Play) 。

（4）USB 在设备供电方面可以通过 USB 电缆供电；也可以通过电池或者其它的电力设备来供电；或使用两种供电方式的组合，并且支持节约能源的挂机和唤醒模式。

（5）为了适应各种不同类型外围设备的要求，USB 提供了四种不同的数据传输类型：控制传输、数据传输、中断数据传输和同步数据传输。

（6）USB 提供全速 12Mbps 的速率和低速 1.5Mbps 的速率来适应各种不同类型的外设，USB2.0 还支持 480Mbps 的高速传输速率，USB3.0 支持超速 5.0Gb/s。

（7）USB 的端口具有很灵活的扩展性，一个 USB 端口串接上一个 USB Hub 就可以扩展为多个 USB 端口。

更多关于 USB 协议的详细内容，可以查看链接：http://www.usb.org

1.2 USB 各版本基本区别

USB 开始的第一版是 USB1.0,经过多次升级更新已经到了 USB3.0,下面给出 USB1.0 USB2.0 USB3.0 之间的比较情况。

表 1-1 USB 版本区别

比较项	USB1.0	USB1.1	USB2.0	USB3.0
最大速度	1.5Mbps	12Mbps	480Mbps	5G-10Gbps
支持设备	低速	低速、全速	低速、全速、高速	低速、全速、高速、超速
输出电流	250mA	250mA	500mA	900mA
推出时间	1996 年 1 月	1998 年 9 月	2000 年 4 月	2008 年 11 月

本文主要以 USB2.0 为讲解基础。

1.3 USB 的拓扑结构及基本概念

1.3.1 USB 拓扑

USB 是一种主从结构系统。USB 系统是通过一根 USB 电缆线将外设与 PC 连接起来，PC 即主机，外设即 USB 设备。主机具有一个或者多个 USB 主控制器和根集线器。主控制器主要负责数据的处理，而根集线器则提供一个连接主控制器与设备之间的接口和通路。图 1-1 给出了 USB 连接拓扑结构。

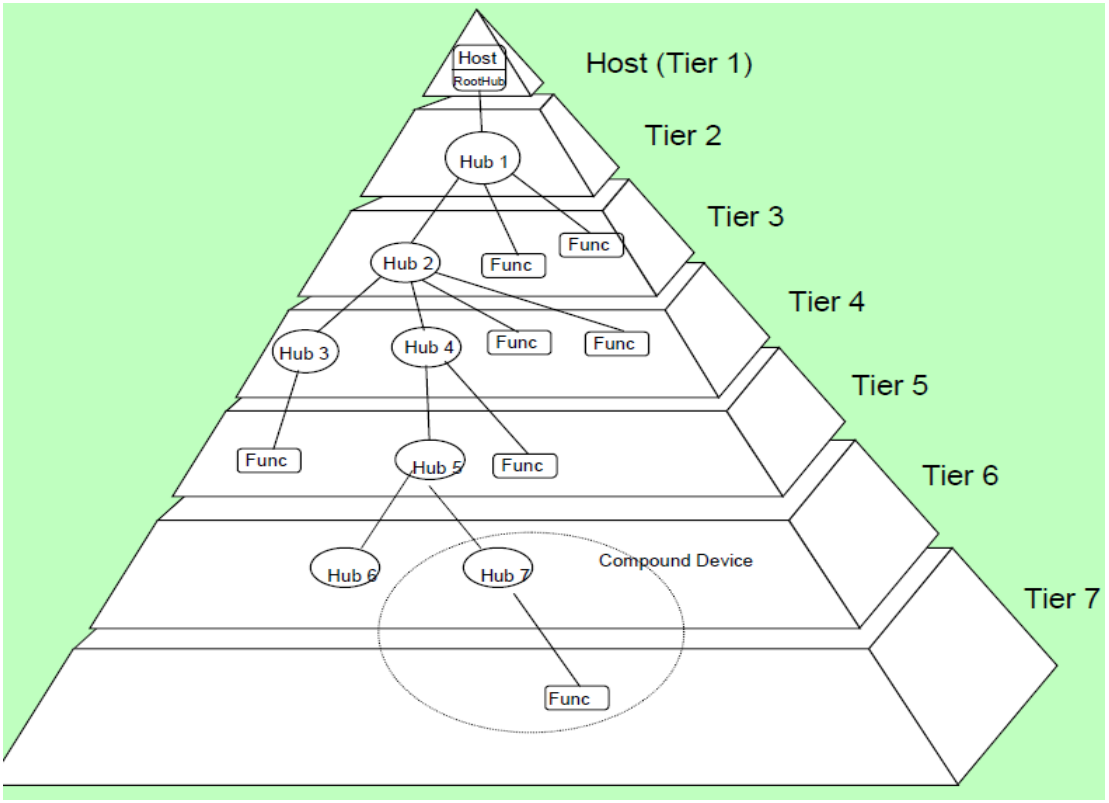


图 1-1 USB 总线拓扑结构

一个 USB 系统中只能有一个主机。主机内设置的根集线器（roothub），提供了外设在主机上的初始附着点。包括根集线器上的一个 USB 端口在内，最多可以级联 127 个 USB 设备，层次最多为 7 层。一个 USB 主控制器最多可以接 127 个设备，是因为协议规定每个 USB 设备具有一个 7bit 的地址，范围为 0-127，而地址 0 是保留给未初始化的设备使用的。

Hub 提供了附加的 USB 节点（又叫端口）。Hub 可以检测出每一个下行端口的状态，并且可以给下端的设备提供电源。在 USB 系统中，将指向 USB 主机的数据传输方向称为上行通信；将指向 USB 设备的数据传输方向称为下行通信。

关于电脑中 USB 主控制器和 USB 根集线器的情况，可以通过“我的电脑”的“设备管理器”查看。如图 1-2 所示，在“通用串行总线控制器”栏目下，可以看到具体的主控制器个数，一个主控制器带有一个根集线器。

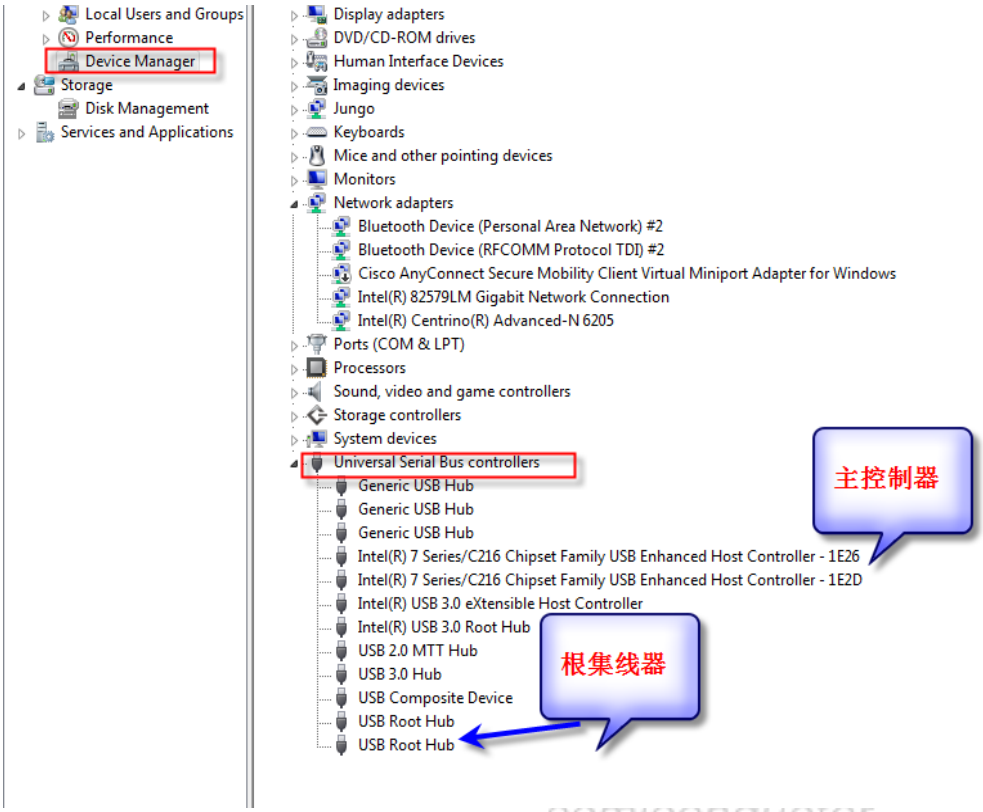


图 1-2 USB 主控制器和根集线器的查看

1.3.2 USB 主机

USB 主机指的是包含 USB 主控制器，并且能够控制完成主机和 USB 设备之间数据传输的设备。主机控制所有的对 USB 的访问。一个 USB 设备想要访问总线必须由主机给予它使用权。主机还负责监督 USB 的拓朴结构。USB 主机可分为三个不同功能的模块，分别为：客户软件、USB 系统软件和 USB 总线接口。

客户软件，一般包括 USB 设备驱动程序和界面应用程序两部分，它负责和 USB 设备的功能单元通信，从而实现特定的功能。客户软件不能直接访问 USB 设备，必须要经过 USB 系统软件和 USB 总线接口模块。

USB 系统软件，一般包括 USB 总线驱动程序和 USB 主控制器驱动程序两部分，它负责和 USB 逻辑设备进行配置同行，并管理客户软件启动的数据传输。

USB 总线接口，包括主控制器和根集线器两部分。主控制器负责完成主机和 USB 设备之间数据的实际传输；根集线器为 USB 系统提供连接起点，用于给 USB 系统提供一个或多个连接点即端口。

1.3.3 USB 设备

USB 设备用于向主机提供一些额外的功能。USB 设备提供的功能是多种多样的，但面向主机的接口却是一致的。所以，对于所有这些设备，主机可以用同样的方式来管理它们与 USB 有关的部分。一个 USB 设备可以分为三个层：最底层是总线接口，用来发送与接收包；中间层处理总线接口与不同的端点之间的数据流；最上层就是 USB 设备所提供的功能。

USB 对一些具有相似特点并提供相似功能的设备进行抽象，进而将 USB 设备分成很多种标准类，包括音频、通信、人机接口设备 HID、显示、大容量存储、电源、打印、集线器设备类等等。

为了帮助主机辨认及确定 USB 设备，这些设备本身需要提供用于确认的信息。在某一些方面的信息，所有设备都是一样的；而另一些方面的信息，由这些设备具体的功能决定。信息的具体格式是不定的，由设备所处的设备级决定.

设备描述符合接口描述符中的类代码、子类代码及协议代码指定了 USB 设备或其接口所属的设备类及相关信息，并定位了合适的设备类驱动程序，关于相关类代码将在下面章节讲解。在设备类中，集线器类，主要是用于为 USB 系统提供额外的外接点，使得一个 USB 端口可以扩展连接多个设备。其余设备类，由于他们一般可以设置为具有特定功能的独立的外部设备，用于扩展主机功能，所以统称为 USB 功能设备类。

1.3.4 USB 端点

每一个 USB 设备在主机看来就是一个端点的集合。主机只能通过端点与设备进行通讯，以使用设备的功能。每个端点实际上就是一个一定大小的数据缓冲区，这些端点在设备出厂时就已定义好。在 USB 系统中每一个端点都有唯一的地址，这是由设备地址和端点号给出的。每个端点都有一定的特性，其中包括传输方式，总线访问频率，带宽端点号，数据包的最大容量等等。端点必须在设备配置后才能生效(端点 0 除外)。端点 0 通常为控制端点，用于设备初始化参数等。端点 1，2 等一般用作数据端点，存放主机与设备间往来的数据。

1.3.5 USB 通信管道

一个 USB 管道是驱动程序的一个数据缓冲区，与一个外设端点的连接。它代表了一种在两者之间移动数据的能力。在数据传输发生之前，主机和设备之间必须先建立一个管道。设备被配置后，端电就可以使用，管道就存在了，如果设备从总线上移除，主机也就撤销了相应的管道。管道有两种类型：数据流管道(其中的数据没有 USB 定义的结构)和消息管道（其中的数据必须有 USB 定义的结构）。管道只是一个逻辑上的概念。

管道和 usb 设备中的端点一一对应，一个 usb 设备含有多少个端点，其和主机进行通信时就可以使用多少条管道，且端点的类型决定了管道中数据的传输类型。每个设备都有一个使用端点 0 的默认控制管道，通过控制管道，可以获取完全描述 USB 设备的信息。端点和各自的管道在每个方向上按照 0-15 编号，因此一个设备最多有 16 个端点，32 个活动管道，即 16 个 IN 管道和 16 个 OUT 管道。

图 1-3 即通信管道图。

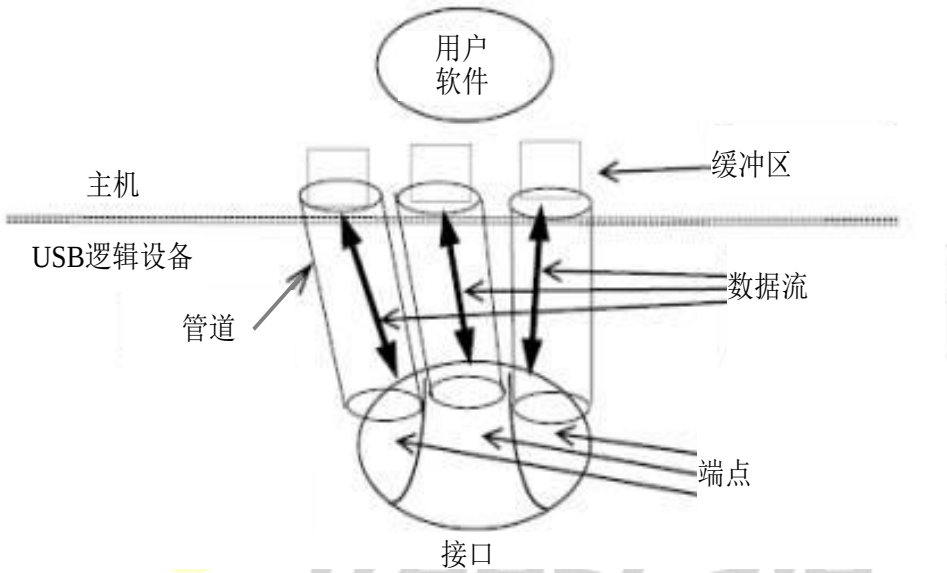


图 1-3 通信管道

1.3.6 USB 应用分类

USB 应用系统可分为三类：

（1） 待开发的 USB 设备作为从机，PC 作为主机；
（2） 待开发的 USB 设备作为主机，其他设备作为从机；
（3） 待开发的 USB 设备可以根据需要在主机和从机两种角色之间进行切换，即所谓的 OTG 技术。

二 USB 的物理特性

2.1 USB 接口定义

USB 作为一种常用的 PC 接口，典型的只有 4 根线连接，其中两根电源线，两根差分信号线，需要注意的是，不能将正负极接反，否则可能会烧坏 USB 设备或者电脑的南桥芯片。现在，在 USB OTG 中，又加了一种 MINI USB 接头，使用的是 5 条线，比标准的 USB 多了一条身份识别线 ID。系统控制器会判断 ID 脚的电平确定是什么样的设备插入，如果是高电平，则是 B 接头插入，此时系统就做主模式

如果 ID 为低，则是 A 接口插入，然后系统就会使用 HNP 对话协议来决定哪个做 Master，哪个做 Slave。下面给出具体引脚的定义。

表 2-1 USB 四线接口示例及引脚定义

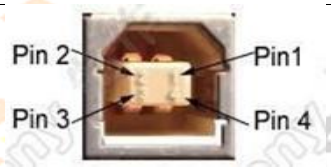
<table>
<tr><th>类型</th><th>图例</th><th>接口管脚定义</th></tr>
<tr><td>A 型 USB 插头
（公口）</td><td> 
</td><td rowspan="4">
<table>
<tr><th>引脚</th><th>功能</th><th>颜色</th><th>备注</th></tr>
<tr><td>1</td><td>V_Bus</td><td>红</td><td>电源
正 5v</td></tr>
<tr><td>2</td><td>Data-</td><td>白</td><td>数据-</td></tr>
<tr><td>3</td><td>Data+</td><td>绿</td><td>数据+</td></tr>
<tr><td>4</td><td>GND</td><td>黑</td><td>地</td></tr>
</table>
</td></tr>
<tr><td>A 型 USB 插座
（母口）</td><td> 
</td></tr>
<tr><td>B 型 USB 插头
（公口）</td><td> 
</td></tr>
<tr><td>B 型 USB 插座
（母口）</td><td> 
</td></tr>
</table>

表 2-2 USB 五线接口示例及引脚定义

<table>
<tr><th>类型</th><th>图例</th><th>接口管脚定义</th></tr>
<tr><td>A 型 USB 插头
（公口）</td><td> 
</td><td rowspan="3">
<table>
<tr><th>引脚</th><th>功能</th><th>颜色</th><th>备注</th></tr>
<tr><td>1</td><td>V_Bus</td><td>红</td><td>电源
正 5v</td></tr>
<tr><td>2</td><td>Data-</td><td>白</td><td>数据-</td></tr>
<tr><td>3</td><td>Data+</td><td>绿</td><td>数据+</td></tr>
<tr><td>4</td><td>ID</td><td></td><td>A 型：
接地
B 型：
悬空</td></tr>
<tr><td>5</td><td>GND</td><td>黑</td><td>地</td></tr>
</table>
</td></tr>
<tr><td>B 型 USB 插头
（公口）</td><td> 
</td></tr>
<tr><td>USB 插座
（母口）</td><td> 
</td></tr>
</table>

在 USB 的插头里的 4 个触点中，会有 2 个触点比另外 2 个要长。长的两个触点分别为 Vbus 引脚和 GND 引脚，这样做的目的是支持热插拔。当 USB 插入时，先接通 GND 和 Vbus，然后再接通数据线。拔下的时候，先断开数据线再断开 Vbus 和 GND。这就保证了在插拔过程中，不会出现有数据信号而无电源的情况。如果数据线早于电源线接通，则可能会让芯片 I/O 引脚电压比电源电压高，从而导致芯片闩锁现象。芯片一旦进入闩锁状态，轻则不能正常工作，重则使芯片过流、过热而烧毁。闩锁是一种类似可控硅的效应，要解除闩锁状态，必须断开电源重新上电。

2.2 USB 通信信号

数据在 USB 总线上实际传输时，采用的是 NRZI（反向不归零）编码的差分信号，这种信号有利于保证数据的完整性和消除噪声干扰。NRZI 的编码方式是：当数据为 0 时，电平翻转；数据为 1 时，电平不翻转。而 NRZI 的译码则采用相反的操作。图 2-1 为 NRZI 编码的数据情况。

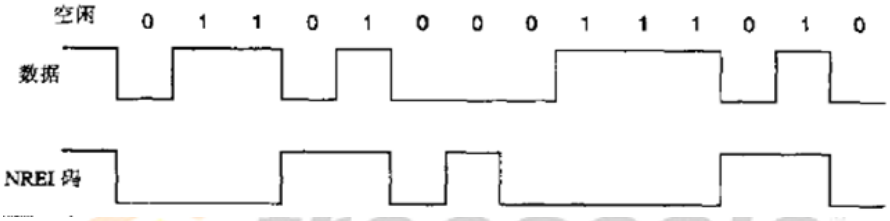


图 2-1 NRZI 编码数据

使用差分传输的原因在于，传统的单线电平传输，是使用信号的电平范围来表示 1 和 0，如果数据在传输过程中受到中低强度的干扰，会导致高低电平突破临界值，从而引起数据传输错误。总线频率越高，线路间的电磁干扰就越厉害，数据传输失败的发生机率也就越高，而差分信号技术能有效克服这种缺点。

差分信号技术最大的特点是：必须使用两条线路才能表达一个比特位，用两条线路传输信号的压差作为判断 1 还是 0 的依据。这种做法的优点是具有极强的抗干扰性。

USB 串行数据使用 NRZI 进行编码，编码过程是在通过 USB 数据线传输之前进行。图 2-2 即通过 SUB 数据线进行信息传输的步骤。NRZI 编码首先由 USB 代理进行，它负责发送信息。接下来，编码后的数据被放入 USB 数据线，这是由差分驱动

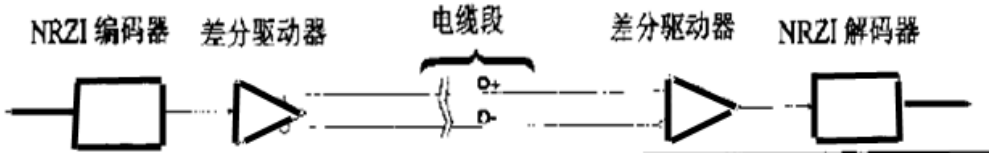


图 2-2 在 USB 上采用 NRZI 编码和差分信号的传输方式

程序完成的。接收器放大传来的差分数据，并把 NRZI 数据发送到解码器，对数据进行编码和采用差分信号进行传输有助于确保数据的完整性和消除噪声干扰。

2.3 USB 总线状态

协议中，给 USB 总线定义了 4 种状态：SE0、SE1、J 和 K 状态。

SE0 状态：两根数据线都被拉低时的状态；

SE1 状态：两根数据线都被拉高时的状态（该状态是非法状态）。

J 状态：当有设备连接到主机，使 D+或 D-被上拉，被上拉的线为高电平而另一根数据线为低电平时，这种状态称为 J 状态（该状态为空闲状态），包被传送之前和之后，数据线上就是该状态。

K 状态：两根数据上的极性都与 J 状态相反的状态（如 J 状态为 D+上拉，D-下拉，则 K 状态指 D+下拉，D-上拉），主机通过 K 和 J 状态测试设备是否支持高速通信。

2.4 USB 设备速度模式检测

当有设备连接以后，电流流过由集线器的下拉电阻和设备在 D+/D-的上拉电阻构成的分压器。由于下拉电阻的阻值是 15KΩ，上拉电阻的阻值是 1.5KΩ，所以在 D+/D-线上会出现大小为（Vcc*15/（15+1.5））的直流高电平电压。如果主机检测到 D+线上为高电压，说明连接上的是高速/全速设备；若检测到 D-线上为高电压，说明连接上的是低速设备。

对于 J 状态而言，全速设备处于差动 1 的状态，低速设各则处于差动 0 的状态；对于 K 状态而言，全速设备处于差动 0 的状态，而低速设各则处于差动 1 的状态。所谓的差动 1，是指 D＋是逻辑高电位，而 D-是逻辑低电位；差动 0 则是刚好相反。

图 2-3 是代表低速状态，图 2-4 是代表全速或者高速状态。

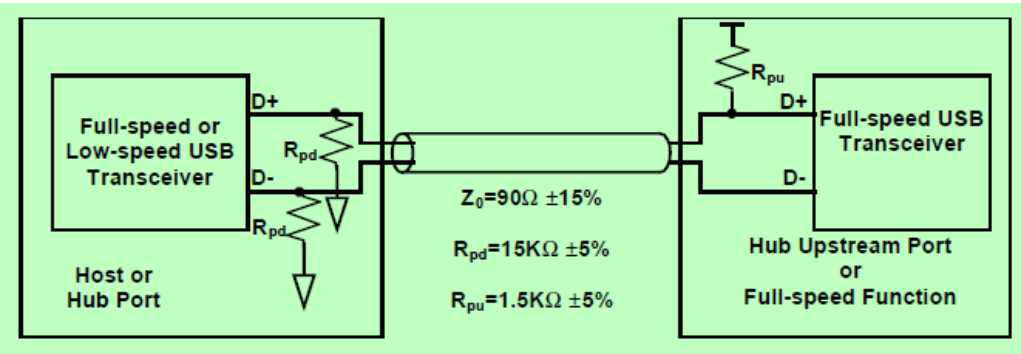


图 2-3 USB 全速或高速的连接方式

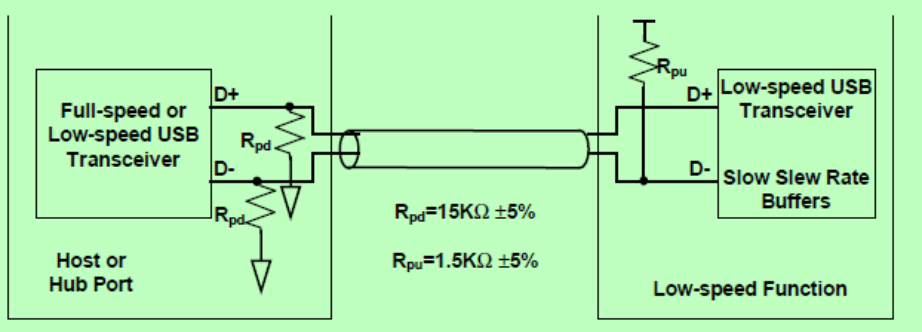


图 2-4 USB 低速的连接方式

在全速的连接方式下，具体是全速还是高速，是靠具有高速能力的 Hub 检测到总线上的 K 状态后，是否响应 K 和 J 的交替状态序列来辨别的。如果能够响应 K 和 J 的交替状态序列，则说明是高速通信，设备将会将全速提高为高速状态。如果 Hub 不响应设备的 K 状态，则设备就运行在全速状态。

USB2.0 支持 3 中传输速度：低速模式（1.5Mb/s）、全速模式（12Mb/s）以及高速模式（48Mb/s）。在 USB 的低速和全速模式中，采用的是电压传输模式；而在高速模式下，则是电流传输模式。当 USB 设备被检测为高速模式，这时要将 D+上的上拉电阻断开。

三 USB 设备的描述符

3.1 描述符定义及作用

所谓描述符，就是用于描述设备特性的具有特定格式排列的一种数据组织结构。

描述符的作用在于设备向主机汇报自己的信息、特征，主机根据这些信息从而加载相应的驱动程序。

USB 只是一个总线，只提供一个数据通路而已。USB 总线驱动程序并不知道一个设备具体如何操作，有哪些行为。具体的一个设备实现什么功能，要由设备自己来决定。那么 USB 主机是如何知道一个设备的功能以及行为的呢？这就要通过描述符来实现了。描述符中记录了设备的类型、厂商 ID 和产品 ID，通常也是利用这些来加载相应的驱动程序，另外还记录了设备的端点情况、版本号等众多信息。

3.2 描述符分类

描述符分为三大类：标准描述符、设备类描述符、厂商描述符。

在标准描述符中，除字符串描述符可选外，任何设备都必须包含剩下的几种标准描述符。规定的 5 种标准的描述符分别为：设备描述符、配置描述符、接口描述符、端点描述符以及字符串描述符。

下面给出三种描述符的类型值，如表 3-1 所示。

表 3-1 USB 描述符分类及描述符类型值

类型	描述符	描述符类型值
标准描述符	设备描述符（Device Descriptor）	0x01
	配置描述符（Configuration Descriptor）	0x02
	字符串描述符（String Descriptor）	0x03
	接口描述符（Interface Descriptor）	0x04
	端点描述符（EndPont Descriptor）	0x05
类描述符	集线器类描述符（Hub Descriptor）	0x29
	人机接口类描述符（HID）	0x21
	报表描述符（HID 相关）	0x22
	实体描述符（HID 相关）	0x23
厂商定义的描述符		0xFF

表中的描述符类型值代表的是具体的描述符类型。

3.3 描述符类代码

设备描述符和接口描述符中的类代码、子类代码以及协议代码指定了 USB 设备或其接口所属的设备类以及相关信息，并定位了合适的设备类驱动程序。设备类驱动程序通常由操作系统提供，开发人员可以直接使用，不需要自己编写。

3.3.1 USB 设备类 DeviceClass

设备类代码是用于确定设备具体属于哪个功能类，设备类代码是放在设备描述符中的第 5 个字节，配合设备类代码的还有设备子类代码以及设备协议代码，关于对应的设备子类代码以及设备协议代码的代码号情况将会在下面的小节讲解，设备子类代码是放在设备描述符中的第 6 个字节，设备协议代码是放在设备描述符中的第 7 个字节。

表 3-2 给出了具体的设备对应的设备类代码。

表 3-2 设备的类别（bDeviceClass）

值（十进制）	值（十六进制）	说明
0	0x00	使用接口描述符中提供的类
2	0x02	通信类（CDC）
9	0x09	集线器类
220	0xDC	用于诊断用途的设备类

224	0xFE	混杂类型设备类
255	0xFF	厂商定义的设备类

3.3.2 USB 接口类 InterfaceClass

接口类代码和设备类代码一样都是是用于确定设备具体属于哪个功能类，接口类代码是放在接口描述符中的第 6 个字节，配合接口类代码的还有接口子类代码以及接口协议代码，关于对应的接口子类代码以及接口协议代码的代码号情况将会在下面的小节讲解，接口子类代码是放在接口描述符中的第 7 个字节，接口协议代码是放在接口描述符中的第 8 个字节。

表 3-2 给出了具体的设备对应的接口类代码，从上节设备类代码可以看到，如果设备类代码选择为 0，则设备的具体类型由接口类代码决定。

表 3-3 USB 协议定义的接口类别（bInterfaceClass）

值（十六进制）	类别
0x01	音频类
0x02	通信类（CDC）
0x03	人机接口类（HID）
0x05	物理类
0x06	图像类
0x07	打印机类
0x08	大数据存储类
0x09	集线器类
0x0A	CDC 数据类
0x0B	智能卡类
0x0D	安全类
0xDC	诊断设备类
0xE0	无线控制器类
0xEF	混杂设备类
0xFE	特定应用类（包括红外的桥接器等）
0xFF	厂商定义的设备

从设备类代码以及接口类代码中可以看到，有一部分的功能其实是重复的，比如通讯类 CDC 功能，在设备类代码中 0x02 就是通讯类 CDC 功能，而在接口类代码中也是 0X02，这是由于设备类代码和接口类代码还存在着交叉和独享的关系，下面将讲解具体的关系。

3.3.3 USB 类的交叉与独享

在描述符中，只有设备描述符和接口描述符中会有类别之分，即只有设备和接口会分类使用，不过有些类别的使用只需经过设备或接口的区分就可彻底清楚明白，这说明在设备类别和接口类别的定义上会有共同的类别名称。而有些类别则是设备或接口独享的，下表是与使用设备相关的类别划分交叉或共享情况：

表 3-4 USB 类的交叉与独享关系表

基类号	用途	描述
00h	设备	使用接口描述符中提供的类
01h	接口	音频类
02h	设备接口通用	通信类（CDC）
03h	接口	人机接口类（HID）
05h	接口	物理类
06h	接口	图像类
07h	接口	打印机类
08h	接口	大数据存储类
09h	设备	集线器类
0Ah	接口	CDC 数据类
0Bh	接口	智能卡类
0Dh	接口	安全类
0Eh	接口	音频类
0Fh	接口	个人医疗类
10h	接口	音频视频类
DCh	设备接口通用	诊断设备类
E0h	接口	无线控制器类
EFh	设备接口通用	混杂设备类
FEh	接口	特定应用类（包括红外的桥接器等）
FFh	设备接口通用	厂商定义的设备

从上表中可以看出：在设备或接口分类上均可彻底分清使用的(设备接口通用)，即在任一处描述符中定义即可的分清楚使用的类的基本类有：02h 通信及 CDC 控制类；DCh 诊断设备类；EFh 混杂设备类；FFh 厂商定义的设备类。

下面分别讲解各个基类号对应的子类代码以及协议代码。

3.3.3.1 基类 00h（设备）

设备代码选择基类 00h 表示使用接口描述符中提供的类，如果接口代码也选择为 00h，表示类型值为空。

表 3-5 基类 00h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
00h	00h	00h	使用接口描述符中提供的类

3.3.3.2 基类 01h（音频类）

基类 01h 表示该音频类设备符合 USB 网站上的音频类设备规范。下表中的 xx 表示任意值。

表 3-6 基类 01h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
01h	xxh	xxh	音频类

3.3.3.3 基类 02h（通信类 CDC）

基类 02h 表示该类设备符合 USB 网站上的通信类设备规范。下表中的 xx 表示任意值。

表 3-7 基类 02h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
02h	xxh	xxh	通信类

3.3.3.4 基类 03h（人机接口类 HID）

基类 03h 表示该类设备符合 USB 网站上的 HID 类设备规范。下表中的 xx 表示任意值。

表 3-8 基类 03h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
03h	xxh	xxh	人机接口类 HID

3.3.3.5 基类 05h（物理类）

基类 05h 表示该类设备符合 USB 网站上的物理类设备规范。下表中的 xx 表示任意值。

表 3-9 基类 05h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
05h	xxh	xxh	物理类

3.3.3.6 基类 06h（图像类）

基类 06h 表示该类设备符合 USB 网站上的图像类设备规范。

表 3-10 基类 06h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
06h	01h	01h	图像类

3.3.3.7 基类 07h（打印机类）

基类 07h 表示该类设备符合 USB 网站上的打印机类设备规范。下表中的 xx 表示任意值。

表 3-11 基类 07h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
07h	xxh	xxh	打印机类

3.3.3.8 基类 08h（大数据存储类）

基类 08h 表示该类设备符合 USB 网站上的大数据存储类设备规范。下表中的 xx 表示任意值。

表 3-12 基类 08h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
08h	xxh	xxh	大数据存储类

3.3.3.9 基类 09h（集线器类）

基类 09h 表示该类设备符合 USB 网站上的大数据存储类设备规范。下表中的 xx 表示任意值。此类仅仅用于设备描述符。

表 3-13 基类 09h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
09h	00h	00h	全速集线器
		01h	全速集线器带有单端 TT
		02h	全速集线器带有多端 TTs

3.3.3.10 基类 0Ah（CDC 数据类）

基类 0Ah 表示该类设备符合 USB 网站上的 CDC 数据类设备规范。下表中的 xx 表示任意值。

表 3-14 基类 0Ah 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
0Ah	xxh	xxh	CDC 数据类

3.3.3.11 基类 0Bh（智能卡类）

基类 0Bh 表示该类设备符合 USB 网站上的智能卡类设备规范。下表中的 xx 表示任意值。

表 3-15 基类 0Bh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
0Bh	xxh	xxh	智能卡类

3.3.3.12 基类 0Dh（安全类）

基类 0Bh 表示该类设备符合 USB 网站上的安全类设备规范。

表 3-16 基类 0Dh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
0Dh	00h	00h	安全类

3.3.3.13 基类 0Eh（智能卡类）

基类 0Eh 表示该类设备符合 USB 网站上的音频类设备规范。下表中的 xx 表示任意值。

表 3-17 基类 0Eh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
0Eh	xxh	xxh	音频类

3.3.3.14 基类 0Fh（个人医疗类）

基类 0Fh 表示该类设备符合 USB 网站上的个人医疗类设备规范。下表中的 xx 表示任意值。

表 3-18 基类 0Fh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
0Fh	xxh	xxh	个人医疗类

3.3.3.15 基类 DCh（诊断设备类）

基类 DCh 表示该类设备为诊断设备类。

表 3-19 基类 DCh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
DCh	01h	01h	诊断设备类

3.3.3.16 基类 E0h（无线控制器类）

基类 E0h 表示该类设备为无线控制器类。

表 3-20 基类 E0h 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
E0h	01h	01h	蓝牙编程接口
		02h	超宽频无线控制接口
		03h	远程 NDIS
		04h	蓝牙放大控制器
	02h	01h	蓝牙主机适配器控制数据接口
		02h	蓝牙设备适配器控制数据接口
		03h	蓝牙适配器同步接口

3.3.3.17 基类 EFh（混杂设备类）

基类 EFh 表示该类设备为混杂设备类。

表 3-21 基类 EFh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
EFh	01h	01h	主动同步设备
		02h	掌上同步
	02h	01h	接口关联描述符
		02h	无线适配器多接口编程接口
	03h	01h	以电缆为基础的关联框架

3.3.3.18 基类 FEh（特定应用类）

基类 FEh 表示该类设备为特定应用类。

表 3-22 基类 FEh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
FEh	01h	01h	设备固件升级
	02h	00h	红外线桥设备
	03h	00h	USB 测试和测量设备
		01h	遵守 USBTMC 协议的 USB 测试和测量设备

3.3.3.14 基类 FFh（厂商定义的设备）

基类 FFh 表示该类设备为厂商定义的设备,下表中的 xx 表示任意值。

表 3-23 基类 FFh 对应子类代码以及协议代码

基类号	子类代码	协议代码	意义
FFh	xxh	xxh	厂商定义的设备

3.4 标准描述符结构

通过 3.2 节描述符的分类，可以知道，标准描述符共具有 5 种标准，他们分别为：设备描述符、配置描述符、接口描述符、端点描述符以及字符串描述符。

首先来讲解一下设备、配置接口等之间的关系，实际上，USB 设备是由一些配置、接口和端点等组件构成的，即一个 USB 设备可以包含一个或者多个配置，在每个配置中可以包含一个或多个接口，在每个接口中可包含若干个断点。其中，配置和接口是对 USB 设备功能的抽象，实际的数据传输时由端点来完成的。接口是端点的集合，配置是接口的集合，设备是配置的集合。

一个 USB 设备只有一个设备描述符。设备描述符里决定了该设备有多少中配置，每种配置都有一个配置描述符；而在每个配置描述符中又定义了该配置里有多少个接口，每个接口都有一个接口描述符；在接口描述符中又定义了该接口有多少个端点，每个端点都有一个端点描述符；端点描述符定义了端点的大小、类型等。如果有类特殊描述符，它会跟在相应的接口描述符之后。所以描述符之间的关系是层次关系。在主机获取描述符时，首先获取设备描述符，接着再获取配置描述符，然后根据配置描述符中的配置集合的总长度，一次将配置描述符、接口描述符、类特殊描述符、端点描述符一次读回。对于字符串描述符，是单独获取的。主机通过发送获取字符串描述符的请求以及描述符的索引号、语言 ID 来获取对应的字符串描述符。

下面具体讲解标准描述符的结构。

3.4.1 设备描述符

设备描述符用于说明设备的总体信息，包括描述符设备的类型、设备支持的协议类型、供应商 ID（VID）、产品 ID（PID）、设备版本号、供应商名称、产品名称及设备所致贺词的配置数等，目的是让主机获取插入的 USB 设备的属性，以便加载合适的驱动程序。一个 USB 设备只能有一个设备描述符，固定为 18 字节的长度，它是主机向设备请求的第一个描述符。

设备描述符 18 个字节中每个字节对应的含义如表 3-24 所示。

表 3-24 设备描述符结构

偏移	域	字节数	值	描述
0	*bLength*	1	数字	此描述符的字节数
1	*bDecriptorType*	1	常量	描述符的类型（设备描述符：0x01）
2	*bcdUSB*	2	BCD 码	USB 版本号（BCD 码）
4	*bDeviceClass*	1	设备类	设备类码： bDeviceClass = 0 ,表明设备类型使用接口描述符中定义的类型，且各个接口独立工作。 bDeviceClass = FFh，表明设备类是由厂商自定义的。bDeviceClass = 1~FEh，查表可得对应设备类值，该设备在不同的接口上支持不同的类。且这些接口可能不能独立工作。此值指出了这些接口集体的类定义。
5	*bDeviceSubClass*	1	设备子类	设备子类码： 这些码值的具体含义根据 bDeviceClass 域来看。 如 bDeviceClass 域为零，此域也须为零 如 bDeviceClass 域为 FFH，此域的所有值保留。
6	*bDevicePortocol*	1	设备协议	协议码 这些码的值视 bDeviceClass 和 bDeviceSubClass 的值而定。 如果设备支持设备类相关的协议，此码标志了设备类的值。如果此域的值为零，则此设备不支持设备类相关的协议，然而，可能它的接口支持设备类相关的协议。如果此域的值为 FFH，此设备使用厂商定义的协议。
7	*bMaxPacketSize0*	1	数字	端点 0 的最大包大小（仅 8,16,32,64 为合法值）
8	*idVendor*	2	ID	厂商标志（由 USB-IF 组织赋值）
10	*idProduct*	2	ID	产品标志（由厂商赋值）
12	*bcdDevice*	2	BCD 码	设备版本号（BCD 码）
14	*iManufacturer*	1	索引	描述厂商信息的字符串描述符的索引值。
15	*iProduct*	1	索引	描述产品信息的字串描述符的索引值。
16	*iSerialNumber*	1	索引	描述设备序列号信息的字串描述符的索引值。
17	*bNumConfigurations*	1	数字	可能的配置描述符数目

下面以 KL25 的 HID 代码为例，给出它的设备描述符定义：

```
uint_8 USB_DESC_CONST g_device_descriptor[DEVICE_DESCRIPTOR_SIZE] =// DEVICE_DESCRIPTOR_SIZE=18
{
  DEVICE_DESCRIPTOR_SIZE,          // bLength ， 描述符长度 18 个字节
  USB_DEVICE_DESCRIPTOR,           // bDecriptorType，0x01 为设备描述符，该代码可查看描述符分类
  0x00, 0x02,                      // bcdUSB，USB 规范为 2.0,BDC 码
  0x00,                            // bDeviceClass ，设备类代码位 0x00，说明外设由接口描述符决定
  0x00,                            // bDeviceSubClass，设备子类代码
  0x00,                            // bDevicePortocol，设备协议代码
  CONTROL_MAX_PACKET_SIZE,         // bMaxPacketSize0，端点 0 支持的最大数据包长度，高速 64 字节，否则 16 字节
  0xA2,0x15,                       // idVendor，供应商 ID VID
  0x00,0x01,                       // idProduct，产品 ID  PID
  0x02,0x00,                       // bcdDevice，设备版本号，BCD 码
  0x01,                            // iManufacturer，供应商的字符串描述符索引：1
  0x02,                            // iProduct，产品的字符串描述符索引：2
  0x00,                            // iSerialNumber，设备序列号的字符串描述符索引
  0x01                             // bNumConfigurations，该 USB 设备支持的配置数目：1
};
```

3.4.2 配置描述符

配置描述符用于说明 USB 设备中各个配置的特性，包括配置信息的总长度、配置所支持接口的个数、配置值、配置的属性及设备可以从总线提取的最大电流等。一个 USB 设备可以包含一个或多个配置，每一个配置都对应一个配置描述符，长度固定为 9 字节。在使用 USB 设备前，主机必须为其选择一个合适的配置。

配置描述符 9 个字节中每个字节对应的含义如表 3-25 所示

表 3-25 配置描述符结构

偏移量	域	大小	值	描述
0	*bLength*	1	数字	此描述表的字节数长度。
1	*bDescriptorType*	1	常量	配置描述表类型（此处为 0x02）
2	*wTotalLength*	2	数字	此配置信息的总长（包括配置，接口，端点和设备类及厂商定义的描述符），即：将要返回的配置信息总长度。
4	*bNumInterfaces*	1	数字	此配置所支持的接口个数
5	*bCongfigurationValue*	1	数字	在 SetConfiguration（）请求中用作参数来选定此配置。
6	*iConfiguration*	1	索引	描述此配置的字串描述符的索引
7	*bmAttributes*	1	位图	配置特性： D7： 保留（设为 1）

				D6： 自给电源 D5： 远程唤醒 D4..0：保留 一个既用总线电源又有自给电源的设备会在 MaxPower 域指出需要从总线取的电量。并设置 D6 为 1。运行时期的实际电源模式可由 GetStatus(DEVICE) 请求得到。
8	*MaxPower*	1	mA	在此配置下的总线电源耗费量。以 2mA 为一个单位。

下面以 KL25 的 HID 代码为例，给出它的配置描述符定义：

```
uint_8 USB_DESC_CONST g_config_descriptor[CONFIG_DESC_SIZE] =
{
  CONFIG_ONLY_DESC_SIZE,  // bLength , 配置描述符的长度：9 个字节
  USB_CONFIG_DESCRIPTOR,  // bDescriptorType，0x02 为配置描述符，该代码可查看描述符分类
  CONFIG_DESC_SIZE, 0x00, // wTotalLength，配置信息的总长度为 34 个字节
  1,                      // bNumInterfaces，该配置所支持的接口数为 1
  1,                      // bCongfigurationValue，配置值为 1
  0,                      //iConfiguration， 配置字符串描述符索引 0
  BUS_POWERED|SELF_POWERED|(REMOTE_WAKEUP_SUPPORT<<REMOTE_WAKEUP_SHIFT),
  // bmAttributes， 配置的属性，非自供电，支持远程唤醒
  0x32,                   // MaxPower，设备从总线提取的最大电流以 2mA 为单位：50*2mA=100mA
}
```

在使用 USB 设备前，主机必须为其选择一个合适的配置。主机根据设备描述符所支持的配置数按照顺序找所有的配置描述符，然后查找接口描述符以及端点描述符，知道查找到主机所支持的配置。

3.4.3 接口描述符

接口描述符用于说明 USB 设备中各个接口的特性，包括接口号、接口使用的端点数、所属的设备类及其子类等。一个配置可以包含一个或多个接口，每个接口都必须有一个接口描述符。接口是端点的集合，可以包含一个或多个可替换设置，用户能够在 USB 处于配置状态时，改变当前接口所含的个数和特性。

接口描述符长度固定为 9 个字节，每个字节对应的含义如表 3-26 所示

表 3-26 接口描述符结构

偏移量	域	大小	值	描述
0	*bLength*	1	数字	接口描述符的字节数大小
1	*bDescriptorType*	1	常量	接口描述符的类型编号
2	*bInterfaceNumber*	1	数字	接口的编号
3	*bAlternateSetting*	1	数字	可替换的接口描述符编号。实际就是接口的描述符的编号。
4	*bNumEndpoints*	1	数字	该接口使用的端点数，不包括端点 0
5	*bInterfaceClass*	1	设备类	接口类
6	*bInterfaceSubClass*	1	设备子类	接口子类
7	*bInterfaceProtocol*	1	设备协议	接口遵循的协议
8	*iInterface*	1	索引	描述该接口的字符串索引值

下面以 KL25 的 HID 代码为例，给出它的接口描述符定义

```
IFACE_ONLY_DESC_SIZE, // bLength,接口描述符长度：9 个字节
USB_IFACE_DESCRIPTOR,// bDescriptorType，0x04 表示本描述符为接口描述符
0x00,                // bInterfaceNumber，接口号
0x00,                // bAlternateSetting，接口的可替换设置值
HID_DESC_ENDPOINT_COUNT,// bNumEndpoints，接口的端点数（除了端点 0）：1
0x03,                // bInterfaceClass，接口所属的 USB 设备类，03 表示为 HID 接口
0x01,                // bInterfaceSubClass 接口所属的 USB 设备子类，0 无引导，1 支持引导
0x02,                // bInterfaceProtocol 接口采用的 USB 设备类协议，1：键盘接口，2 鼠标接口。
0x00,                // iInterface 接口字符串描述符的索引
```

接口描述符中用到接口编号 bInterfaceNumber，以区分在同一配置下的不同的接口。同时还有该接口描述符的索引 iInterface，这意味着将为其准备一个字符串描述符。

可替换的接口描述符编号 bAlternateSetting，表示对某一接口进行描述的描述符编号。虽然，USB 设备的配置与配置描述符是一一对应的，即一个配置只能由一个配置描述来描述它，但一个接口却允许有多种描述符来描述它，尽管接口描述符的编号还是唯一一个。说白了就是：一个接口有唯一的一个接口编号，但一个接口却可以有多个不同的描述符编号，而这些不同的接口描述符的编号值就是 bAlternateSetting。所以，通过 bInterfaceNumber 可以选定一个唯一的接口，然后再通过 bAlternateSetting 选择想要的对该接口的描述。主机通过 GetInterface 可以获取当前正在使用的接口及接口描述，通过 SetInerface 可以选定某接口及其使用的描述符。

3.4.4 端点描述符

端点是设备与主机之间进行数据传输的逻辑接口，所有的传输都是传送到设备端点，或是从设备端点发出，这种端点实际上就是一个能够存储多个字节的缓冲器。除配置使用的端点 0（控制端点，一般一个设备只有一个控制端点）为双向端口外，其它均为单向。端点描述符描述了数据的传输类型、传输方向、数据包大小和端点号（也可称为端点地址）等。

端点描述符长度固定为 7 个字节，每个字节对应的含义如表 3-27 所示

表 3-27 端点描述符结构

偏移量	域	大小	值	描述
0	*bLength*	1	数字	端点描述符的字节数大小
1	*bDescriptorType*	1	常量	端点描述符的类型编号
2	*bEndpointAddress*	1	端点	端点地址以及方向 Bit 3-0：端点号 Bit 6-4：保留，为 0 Bit 7：方向，0=OUT;1=IN
3	*bmAttributes*	1	数字	描述了该端点的传输特性： Bit 0-1:定义了传输类型,00=控制传输、01=同步传输、10=批量传输、11=中断传输 如果不是同步端点，bit5-2 保留，为 0；如果是同步端点，定义如下： Bit 3-2： 同步类型 00=非同步；01=异步；10=适配；11=同步 Bit 5-4: 使用类型 00=数据端点；01=反馈端点；10=隐式反馈数据端点；11=保留
4	*wMaxPacketSize*	2	数字	最大包长度
6	*bInterfaceSubClass*	1	数字	定义了该端点被主机访问的周期，此域值对于批量传输和控制传输毫无意义。对于同步传输，其值必须为 1，即 1ms 为标准的同步帧周期。对于中断传输，该值为 1~255，即 1ms~255ms

下面以 KL25 的 HID 代码为例，给出它的端点描述符定义

```
ENDP_ONLY_DESC_SIZE,          //端点描述符字节数大小，固定为 7 个字节
USB_ENDPOINT_DESCRIPTOR,    //端点描述符类型编号，为 0x05 表示本描述符为端点描述符
HID_ENDPOINT|(USB_SEND << 7), //端点地址及输入输出属性，0x81，输入方向，端点号为 1
USB_INTERRUPT_PIPE,           //端点的传输类型属性，传输方式为中断传输
HID_ENDPOINT_PACKET_SIZE, 0x00,  //端点收、发的最大包大小，0x08, 0x00, 包长度为 0.
0x0A                  //对周期性端点的访问间隔，  间隔为 10ms
```

端点的传输特性还决定了其与主机通信时所采用的传输类型，如控制端点只能使用控制传输。根据端点的不同用途，可将端点分为两类：0 号端点和非 0 号端点。0 号端点比较特殊，它有数据输入 IN 和数据输出 OUT 两个物理单元，并只支持控制传输。所有的 USB 设备都必须含有 0 号端点，用作缺省控制管道。USB 系统软件就是使用该管道和 USB 逻辑设备进行配置通信的。0 号端点在 USB 设备上电后就可以使用，非 0 号端点必须要配置之后才可以使用。对于低速设备，非 0 端点数最多 4 个，范围 0-3；对于全速/高速设备，非 0 端点数最多 16 个，范围 0-15。

3.4.5 字符串描述符

字符串描述符是一种可选的 USB 标准描述符，描述了如制商、设备名称或序列号等信息。如果一个设备无字符串描述符，则其它描述符中与字符串有关的索引值都必须为 0。字符串使用的是 Unicode 编码。

字符串描述符是用字符的形式描述设备、配置、接口、端点等信息。

字符串描述符以一种格式 2 类符值的方式存在：

1） 显示语言的字符串描述符：该字符串描述符表明了设备支持哪几种语言。

2） 显示信息的字符串描述符：用于描述具体的信息。

字符串描述符长度为 bString 的字节+2 个字节，每个字节对应的含义如表 3-28 所示

表 3-28 端点描述符结构

偏移量	域	大小	值	描述
0	bLength	1	数字	此描述表的字节数（bString 域的数值 N+2）
1	bDescriptorType	1	常量	描述符类型（此处应为 0x03）
2~N	Strings	N	数字	字符串

显示语言的字符串描述符与显示信息的字符串描述符的区别在于 Strings 项的不同，对于显示语言的字符串描述符来说 Strings 项由多个 wLANGID[n]数组元素组成，每个 wLANGID[n]是一个双字节的代表语言的 ID 值。而对于显示信息的字符串描述符而言，Strings 则是描述信息后的一组 UNICODE 编码。

下面以 KL25 的 HID 代码为例，给出它的字符串描述符定义

```
uint_8_ptr const g_string_descriptors[USB_MAX_STRING_DESCRIPTORS+1] = {
(uint_8_ptr) USB_STR_0, //索引 0

(uint_8_ptr) USB_STR_1, //索引 1

(uint_8_ptr) USB_STR_2, //索引 2

(uint_8_ptr) USB_STR_n//索引 n

};
```

以其中的索引 1 为例，具体内容如下：

```
uint_8 USB_DESC_CONST USB_STR_1[USB_STR_1_SIZE+USB_STR_DESC_SIZE]
                = {  sizeof(USB_STR_1),  //bLength, 描述符表的长度
                   USB_STRING_DESCRIPTOR,//描述符类型 3，代表是字符串描述符
                   'F',0,   //以下是具体的信息字符。
                   'R',0,
                   'E',0,
                   'E',0,
                   'S',0,
                   'C',0,
                   'A',0,
                   'L',0,
                   'E',0,
                   ' ',0,
                   'S',0,
                   'E',0,
                   'M',0,
                   'I',0,
                   'C',0,
                   'O',0,
                   'N',0,
                   'D',0,
                   'U',0,
                   'C',0,
                   'T',0,
                   'O',0,
                   'R',0,
                   ' ',0,
                   'I',0,
                   'N',0,
                   'C',0,
                   '.',0
                };
```

3.4.6 HID 相关描述符

HID 设备的描述符除了 5 个 USB 的标准描述符（设备描述符、配置描述符、接口描述符、端点描述符、字符串描述符）外，还包括 3 个 HID 设备类特定描述符：HID 描述符、报告描述符、实体描述符。

HID 设备类描述符并不是说仅用这一个描述符就可描述清楚这类设备，而是指 HID 设备除包含所有的标准描述符外，还需这个 HID 设备来补充描述。也就是说，在使用一般的设备时，只需使用标准的描述符就可描述清楚，而若使用 HID 设备时，除了要使用全部的标准的描述符外还需 HID 描述符来补充描述，具体的流程如图 3-1 所示。

除了 HID 的三个特定描述符组成对 HID 设备的解释外，5 个标准描述符中与 HID 设备有关的部分有：设备描述符中 bDeviceClass、bDeviceSubClass 和 bDeviceProtocol 三个字段的值必须为零。
接口描述符中 bInterfaceClass 的值必须为 0x03，bInterfaceSubClass 的值为 0 或 1，如果为 0 则只有在操作系统启动后才能识别并使用您的 HID 设备）此时 bInterfaceProtocol 无效，置零即可。为 1 表示 HID 设备符是一个启动设备（Boot Device，一般对 PC 机而言才有意义，意思是 BIOS 启动时能识别并使用您的 HID 设备，且只有标准鼠标或键盘类设备才能成为 Boot Device。此时 bInterfaceProtocol 的取值含义如下表所示：

表 3-29 HID 接口描述符中 bInterfaceProtocol 的含义

bInterfaceProtocol 的取值（十进制）	含义
0	NONE
1	键盘
2	鼠标
3-255	保留

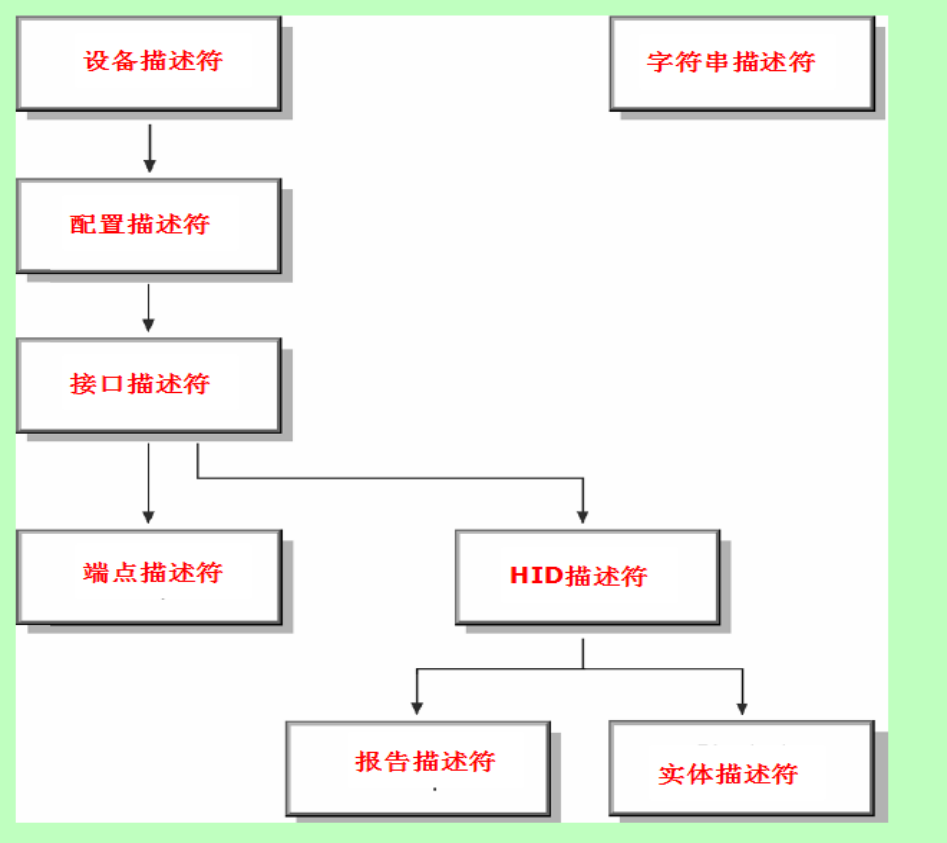


图 3-1 HID 描述符流程图

下面分别对这 3 个 HID 设备类特定描述符进行说明。

3.4.6.1 HID 描述符

HID 描述符关联于接口描述符，因而如果一个设备只有一个接口描述符，则无论它有几个端点描述符，HID 设备只有一个 HID 描述符。HID 设备描述符主要描述 HID 规范的版本号、HID 通信所使用的额外描述符、报表描述符的长度等

HID 描述符长度,在有一个以上描述符的设备的时候，为 12 个字节，如果只有一个描述符设备的时候，只有 9 个字节，每个字节对应的含义如表 3-30 所示

表 3-30 HID 描述符结构

偏移量	域	大小	值	描述
0	bLength	1	数字	此描述符的长度（以字节为单位）
1	bDescriptorType	1	常量	描述符种类（此处为 0x21 即 HID 类描述符）

2	bcdHID	2	数字	HID 规范版本号（BCD 码），采用 4 个 16 进制的 BCD 格式编码，如版本 1.0 的 BCD 码为 0x0100,版本为 1.1 的 BCD 码为 0x0110
4	bCountryCode	1	数字	硬件目的国家的识别码（BCD 码）（见表 3-31）
5	bNumDescritors	1	数字	支持的附属描述符数目
6	bDescriptorType	1	常量	HID 相关描述符的类型，即补充描述符码
7	wDescriptorLength	2	数字	报告描述符总长度
9	【bDescriptorType】	1	常量	用于识别描述符类型的常量，使用在有一个以上描述符的设备
10	【wDescriptorLength】	2	数字	描述符总长度，使用在有一个以上描述符的设备

表 3-31 HID 硬件目的国家识别码

识别码（十进制）	国家和地区	识别码（十进制）	国家和地区
00	不支持	18	Netherlands/Dutch
01	Arabic	19	Norwegian
02	Belgian	20	Persian (Farsi)
03	Canadian-Bilingual	21	Poland
04	Canadian-French	22	Portuguese
05	Czech Republic	23	Russia
06	Danish	24	Slovakia
07	Finnish	25	Spanish
08	French	26	Swedish
09	German	27	Swiss/French
10	Greek	28	Swiss/German
11	Hebrew	29	Switzerland
12	Hungary	30	Taiwan
13	International (ISO)	31	Turkish-Q
14	Italian	32	UK
15	Japan (Katakana)	33	US
16	Korean	34	Yugoslavia
17	Latin American	35	Turkish-F
		36—255	Reserved

下面以 KL25 的 HID 代码为例，给出它的 HID 描述符定义

```
HID_ONLY_DESC_SIZE, //此描述符的长度为 9 个字节
USB_HID_DESCRIPTOR, //描述符种类（此处为 0x21 即 HID 类描述符）
0x00,0x01,          // HID 规范版本号（BCD 码），为版本 1.0 的 BCD 码
0x00,               //硬件目的国家的识别码（BCD 码）
0x01,               //支持附属描述符个数为 1
0x22,               //补充描述符的类型为报表描述符
0x34,0x00,          //报告描述符总长度为 0X34
```

3.4.6.2 报告描述符

报告描述符的语法不同于 USB 标准描述符，它是以项目（items）方式排列而成，无一定的长度。HID 的报告描述符已经不是简简单单的描述某个值对应某个固定意义了，它已经能够组合出很多种情况，并且需要 PC 上的 HID 驱动程序提供 parser 解释器来对描述的设备情形进行重新解释，进而组合生成出本 HID 硬件设备独特的数据流格式，所以可以把它理解为“报告描述符脚本语言”更为贴切。我们使用“报告描述符”专用脚本语言，让用户来自己定义他们的 HID 设备都有什么数据、以及这些数据各个位（bit）都有什么意义。

有关报告描述符的详细信息可参考 USB HID 协议，USB 协会提供了一个 HID 描述符编辑工具称作 HID Descriptor Tool，用它可方便生成我们的报告描述符。

3.4.6.3 实体描述符

实体描述符被用来描述设备的行为特性。实体描述符是可选的描述符，HID 设备可以根据其本体的设备特性选择是否包含实体描述符。表 3-32 为 HID 实体描述符的结构。

表 3-32 HID 实体描述符的结构

偏移量	域	大小	说明
0	bDesignator	1	用来指定本体的哪一部分影响项目（含义见表 3-33）
1	bFlags	1	位指定标志 位 0~4：Effort 位 5~7：Qualifier（含义见表 3-34）

表 3-33 bDesignator 取值含义表

bDesignator 取值	含义	bDesignator 取值	含义
0x00	无	0x15	小指
0x01	手	0x16	头
0x02	眼球	0x17	肩
0x03	眉	0x18	腰骨
0x04	眼皮	0x19	腰
0x05	耳	0x1A	大腿
0x06	鼻	0x1B	膝盖
0x07	嘴	0x1C	小腿
0x08	上唇	0x1D	足
0x09	下唇	0x1E	脚
0x0A	颚	0x1F	脚跟
0x0B	颈	0x20	拇指
0x0C	上臂	0x21	大拇指
0x0D	手肘	0x22	第二指
0x0E	前臂	0x23	第三指
0x0F	手腕	0x24	第四指
0x10	手掌	0x25	小拇指
0x11	拇指	0x26	眉
0x12	食指	0x27	脸
0x13	中指	0x28~0xFF	保留
0x14	无名指		

表 3-34 Qualifier 取值含义

Qualifier 取值	含义	Qualifier 取值	含义
0x00	无	0x04	其中之一
0x01	右	0x05	中间
0x02	左	0x06	保留
0x03	两者同时	0x07	保留

四 USB 的通信协议

4.1 USB 的包结构及分类

要想了解 USB 的通信协议，就必须从 USB 的信息传输单元包及其数据开始。USB 的包是 USB 通信的基础单位。所有的包都是由域构成的。但是不同的包，包含了不同的域。一般情况下，包由同步域开始；紧跟包标示符 PID；以包结尾符 EOP 结尾。此外，不同的包具有不同的中间数据。下面详细介绍包的结构以及分类。

4.1.1 USB 的包结构

USB 协议中，包（Packet）是 USB 系统中信息传输的基本单元，所有数据都是经过打包后在总线上传输的。组成包的字段主要有：同步字段、包标识符字段、地址字段、端点字段、帧号字段、数据字段、循环冗余校验字段和包结尾字段。包的一般格式为：

图 4-1 USB 包的一般格式

同步字段（SYNC）	PID 字段	数据字段	CRC 字段	包结尾字段（EOP）

在 USB 的数据传输中，所有的传输包都始于 SYNC，接着是 PID 字段，后面是包中所包含的数据信息，然后是循环冗余校验信息，最后是包结尾字段。根据不同的包类型，包结构有相应的具体格式，将在包分类中具体讲解。

数据在总线传输的时候，每个字段的 LSB 在前。

下面介绍包中对应的每个字段。

4.1.1.1 同步字段 SYNC

所有的包都从同步（SYNC）字段开始的，同步字段是产生最大的边缘转换密度（Edge Transition Density）的编码序列 .输入电路用它将输入数据和本地时钟对齐. 同步字段作为空闲状态出现在总线上，后面跟着以 NRZI 编码的二进制串“KJKJKJKK” .全速/低速的 SYNC 字段被定义为 8 位,高速的 SYNC 字段被定义为 32 位.接收到的 SYNC 字段只是作为一个同步机制,并不在接着的包图中表现出来,SYNC 字段的最后两个位是一个识别 SYNC 字段的结束或者以此推断 PID 的开始.

全速/低速包的 SYNC 数值宽度为 8bit，值固定为 00000001B。

高速包的 SYNC 宽度为 32bit，值固定为 31 位 0+1 位 1.

4.1.1.2 包标示符字段 PID

包标识符字段（PID）紧随在 SYNC 字段后面，用来表示包的类型。PID 长度为一个字节（8 个数据位），由 4 位包类型字段和 4 位校验字段构成,如表 4-1 所示。

PID 是 USB 包类型的唯一标志，USB 主机和 USB 设备在接收到包后，首先必须对包标识符解码得到包的类型再决定下一步动作。

包标识符指出了包的类型，并由此隐含地指出了包的格式和包上所用错误检测的类型。包标识符的 4 位的校验字段可以保证包标识符译码的可靠性，这样包的余项也就能被正确地解释。包标识符的校验字段通过对包类型字段的二进制的求反码产生的。如果 4 个 PID 检验位不是它们的各自的包标识符位的补，则说明存在 PID 错。

表 4-1 PID 格式

D7	D6	D5	D4	D3	D2	D1	D0
PID3	$\overline{\text{PID2}}$	$\overline{\text{PID1}}$	$\overline{\text{PID0}}$	PID3	PID2	PID1	PID3

由 PID 可以将包分为令牌、数据、握手和特殊包四种封包类型。这四种封包类型是由 PID[1-0]来决定的，此外每种封包的封包类型都可以通过 PID[3-2]来定义出不同的封包格式，具体情况，请查看图 4-2.

表 4-2 各种包的协议与规范

包类型		PID[3：0]	含义
令牌包	OUT	0001B	主机向设备发送数据
	IN	1001B	主机向设备要数据
	SETUP	1101B	主机到设备，用于控制传输
	SOF	0101B	帧的起始标记，标志着一个时间片的开始，表面接下来安排数据传输
数据包	DATA0	0011B	第偶数次发送的包
	DATA1	1011B	第奇数次发送的包
	DATA2	0111B	一个事务中的高速、高带宽同步传输数据包
	MDATA	1111B	分裂高带宽同步传输的高速数据包
握手包	ACK	0010B	接收器收到无错误的包
	NAK	1010B	接收器无法接受数据或发射器无法送出数据
	STALL	1110B	端点产生停滞的状况
	NYET	0110B	接收器还没有准备好接收下次数据
特殊包	PRE	1100B	使能下游端口的 USB 总线的数据传输切换到低速的设备，只是用在全速中，通知集线器打开低速端口
	PING	0100B	等待设备返回 ACK 或 NAK,判断设备是否能够进行传输
	SPLIT	1000B	通知集线器将高速数据包转化为全速或低速

			数据包下发
	ERR	1100B	分裂事物中的错误表示
	保留	0000B	保留未用

接下来分别讲解各个包对应的字段格式。

（1） OUT、IN 以及 SETUP 包格式

IN 令牌包 用于通知设备返回一个数据包；OUT 令牌包用于通知设备将要输出一个数据包；SETUP 令牌包用在控制传输中。以上三种令牌包的结构如下：

表 4-3 OUT, IN, SETUP 令牌包结构

	8 位	7 位	4 位	5 位	
同步字段 (SYNC)	包标示符字段（PID）：$\overline{PID}$ + PID	地址字段 (ADDR)	端点字段（ENDP）	CRC5 校验	包结尾字段（EOP）

（2） SOF 包格式

SOF 包在每帧开始时以广播方式发送，发送帧号，不跟随数据，该令牌包的结构如下：

表 4-4 SOF 令牌包结构

	8 位	11 位	5 位	
同步字段 (SYNC)	包标示符字段（PID）：$\overline{PID}$ + PID	帧序列号	CRC5 校验	包结尾字段（EOP）

（3） 数据包格式

用于传输数据。当设备或主机端成功发送或接收数据时，会切换数据包的类型。有 DATA0, DATA1, DATA2, MDATA 类型。结构为：

表 4-5 数据包结构

	8 位	8 位	8 位		8 位	16 位	
同步字段 (SYNC)	包标示符字段（PID）：$\overline{PID}$ + PID	字节 0	字节 1	……	字节 N	CRC16 校验	包结尾字段（EOP）

（4） 握手包以及特殊包格式

握手包数据结构简单， 用于表示对方是否确认传输。

表 4-5 握手包和数据包结构

	8 位	
同步字段(SYNC)	包标示符字段（PID）： $\overline{PID}$ + PID	包结尾字段（EOP）

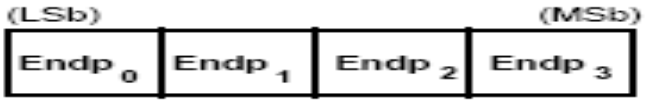
4.1.1.3 地址字段 ADDR

设备地址由 7 位组成，共有 128 个地址值，由 ADDR<6：0>指定。ADDR 规定在 IN,SETUP 和 OUT 令牌以及 PING 和 SPLIT 特许令牌中使用.根据定义,每个 ADDR 值都定义了一个功能设备。地址 0 作为缺省地址，不能分配给 USB 设备，因此只有 127 个可分配的地址值。在 USB 设备上电的时候，主机用缺省地址 0 与设备通信。当 USB 上电配置完成后，主机重新为 USB 设备分配一个 USB 地址。

4.1.1.4 端点字段 ENDP

端点字段由 4 位组成，可寻址 16 个端点， 如图 4-2 所示。ENDP 字段仅用在 IN、OUT 与 SETUP 令牌包中。(ENDP)字段在功能部件需要一个以上端口时候允许更灵活的寻址。除了端口地址 0 之外，端口个数是由功能部件决定的。端口字段只对输入，建立和输出标记 PID 有定义。所有的功能部件都必须在端口 0 提供一个控制管道（缺省控制管道）。对于低速(Low Speed)设备，每个功能部件最多提供 3 个管道：在端口 0 的控制管道加上 2 个附加管道（或是 2 个控制管道，或是 1 个控制管道和 1 个中断端口，或是 2 个中断端口）。全速(Full Speed)功能部件可以支持最多可达 16 个的任何类型的端口。

表 4-2 4 位端点字段结构



4.1.1.5 帧序列号字段

帧序列号长度为 11 位，从 0 开始，每发送一个 SOF，其值自动加 1，最大数值为 0x7FF（十进制值 2047），当超过最大值时自动从 0 开始循环。

4.1.1.6 数据字段

数据字段的最大长度为 1024 字节，而且必须是整数个字节，在数据传输的时候，首先传输低字节，然后传输高字节。对于每一个字节，先传输字节的低位，再

传输字节的高位。实际的数据字段长度，需根据 USB 设备的传输速度（低速、全速或高速）以及传输类型（中断传输、批量传输、实传输）而定。

4.1.1.7 循环冗余校验字段 CRC

循环冗余校验（CRC）被用来在标记和数据包中保护所有的非 PID 字段。在上下文中，这些字段被认为是保护字段。PID 不在含有 CRC 的包的 CRC 校验范围内。在位填充之前，在发送器中所有的 CRC 都由它们的各自的字段产生。同样地，在填充位被去除之后，CRC 在接收器中被译码。标记和数据包的 CRC 可 100 %判断单位错和双位错。失败的 CRC 指出了保护字段中至少有一个字段被损坏，并导致接收器忽略那些字段，且在大部分情况下忽略整个包。为了 CRC 的发生和校检，发生器和检验器里的移位寄存器置成为全 1（All-ones）型。对于每个被发送或者被收到的数据位，当前余项的最高一位和数据位进行异或（XOR），然后，余项是左移 1 位，并且，最低一位置零。如果异或的结果是 1，余项和生成多项式作异或。当检查的字段的最后的一位被发送的时候，发生器里的 CRC 被颠倒，再以最高位（MSb）在前发给检验器。当检验器收到 CRC 的最后的一位,且不发生错误的时候，余项将等于多项式的的剩余。如果剩余与包接收器中最后计算出的检验和余项（Checksum remainder）不匹配，则存在 CRC 误差。对于 CRC，必须满足位填充的要求，且如果前 6 位都是 1 的话，这包括在 CRC 的最后插入零。

在令牌包中，一般采用 5 位循环冗余校验；在数据包中，采用 16 位循环冗余校验。

（1） 令牌 CRC

标记使用了 5 位的 CRC 字段，它覆盖了 IN，SETUP 和 OUT 令牌的 ADDR 和 ENDP 场或 SOF 标记的时间场。生成多项式如下：

$$G(X) = X^5 + X^2 + 1$$

用二进制的位方式表示多项式表示这个多项式就是 00101B.如果令牌的所有位都被正确接收,那么接收器的 5 位余数是 01100B.

（2） 数据 CRC

数据 CRC 是一个 16 位的多项式，应用在数据包的数据场。其生成的多项式如下：

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

这个多项式的二进制位组合是 1000000000000101B。如果全部的数据和 CRC 位被准确无误地收到，16 位剩余将是 1000000000001101B.

4.1.1.8 包结尾字段 EOP

包结束 EOP（End of Packet）字段，每个包的 EOP 为 3 位(全速/低速)或 8 位(高速)，低速/全速 EOP 在物理上表现为差分线路的两根数据线保持 2 比特低位时间和 1 比特空闲位时间。对于高速设备的 EOP，使用故意的位填充错误来表示。USB 主机根据 EOP 判断数据包的结束。

4.1.2 USB 的包分类

通过 4.4.4.2 节，可以指定 USB 包有多种分类，主要有令牌包、数据包、握手包以及特殊包四大类。每类中还分为具体的子类，通过 PID 识别具体的包类型。下面讲解 USB 各类的具体情况。

4.1.2.1 令牌包

令牌包用来启动一次 USB 传输。因为 USB 是主从结构的拓扑结构，所以所有的数据传输都是由主机发起的，设备只能被动的接收数据（唯一例外的是支持远程唤醒的设备能够主动改变总线的状态让集线器感知到设备的唤醒信号，但是这个过程并不传送数据，只是改变总线的状态）。

令牌包有四种，分别为输入（IN）、输出（OUT）、建立（SETUP）以及帧起始（SOF）。

输出令牌包用于通知设备将要输出一个数据包；

输入令牌包用于通知设备返回一个数据包；

建立令牌包用在控制传输中，它跟输出令牌包的作用一样，也是通知设备将要输出一个数据包，两者的区别在于：SETUP 令牌包后只使用 DATA0 数据包，并且只能发到设备的控制端点，设备必须要接收，即使设备正在进行数据传输操作的过程中也要相应 SETUP 令牌包，而 OUT 令牌包没有这些限制。SETUP 包是 OUT 包的一个特殊形式，是高优先级的，SETUP 包总是指向端点 0 的。

帧起始包在每帧（或微帧）开始时发送，它以广播的形式发送，所有 USB 全速设备和高速设备都可以接收到 SOF 包。在低速、高速模式下，主机每间隔 1ms（这个 1ms 称为一帧，允许误差 0.005ms）发送一个帧开始包 SOF。在高速模式下，主机每间隔 125us（即为一微帧，允许误差 0.0625us）发送一个帧开始包 SOF。包括集线器的所有功能部件都可以收到 SOF 包，SOF 标记不会使得接收功能部件产生返回包，所以不能保证主机发送的 SOF 包都能被功能部件接收到。事实上，每两个 SOF 包之间的时间段为事务处理时间。

4.1.2.2 数据包

数据包就是用来传输数据的，在 USB1.1 协议中，只有两种数据包 DATA0 包和 DATA1 包。在 USB2.0 中又增加了 DATA2 和 MDATA 包，主要用在高速分裂事务和高速高带宽同步传输中。

不同类型的数据包是用于握手包出错时纠错的，并支持数据切换同步。主机总是通过配置事件初始化总线传送的首个数据包为 DATA0，下一个数据包使用 DATA1，并且以后的数据传送轮流切换，即 DATA0->DATA1->DATA0->DATA1…

下面讲解如何利用数据包切换来确认数据是否出错的：

当数据包成功发送或者接收的时候，数据包类型切换。当检测到对方使用的数据包类型不对的时候，USB 系统认为发生了错误，并试图从错误中恢复。数据包类型不匹配主要发生在握手包被损坏的情况。当一端已经正确接收到数据并返回确认信号时，确认信号却在传输过程中被损坏。这时，另一端就无法知道刚刚发送的数据是否成功，则保持数据包类型不变。如果下一次接收到数据包类型和现在的类型不一致，则认为发送成功。否则，认为没有发送成功。

4.1.2.3 握手包

握手包用来表示一个传输是否被对方确认。握手包只有同步字段、PID 和 EOP，是一种最简单的数据包。

握手包有 ACK、NAK、STALL 和 NYET 四种。

ACK 表示正确接收数据，并且有足够的空间来容纳数据。主机和设备都可以用 ACK 来确认，而 NAK、STALL、NYET 只有设备能够返回，主机不能使用这些握手包。

NAK 表示没有数据需要返回，或者数据正确接收但是没有足够的空间来容纳。当主机收到 NAK 时，知道设备还没有准备好，主机会在以后重新传输。

STALL 表示设备无法执行这个请求，或者端点已经被挂起，它表示一种错误状态。设备返回 STALL 后，需要主机进行干预才能解除这种 STALL 状态。

NYET 只有 USB2.0 的高速设备输出事务中使用，它表示设备本次数据成功接收，但是没有足够的空间来接收下一次数据。主机在下一次输出数据时，将先使用 PING 包试探设备是否有足够空间接收数据。

4.1.2.4 特殊包

特殊包是一些特殊的场合使用的包，总共有四种：PRE、ERR、SPLIT 和 PING。其中 PRE、SPLIT 和 PING 是令牌包，为 USB2.0 新增。ERR 是握手包。

PRE 令牌包是通知集线器打开其低速端口的一个前导包。PRE 只使用在全速模式中，结构和握手包一样。

PING 令牌包和 OUT 令牌包有同样的结构，PING 令牌包不发数据，只是等待设备返回 ACK 或者 NAK，以判断设备是否能够传送数据。该令牌包只在 USB2.0 中使，只被使用在批量传输和控制传输的输出事务中。

SPLIT 令牌包是高速事务分裂令牌包，通知集线器将高速数据包转化为全速或低速数据包发送给其下面的端口。

ERR 握手包是在分裂事务中表示错误使用。

4.2 USB 通信中的事务处理

什么是事务呢？事务通常由两个或者三个包组成：令牌包、数据包和握手包。

令牌包用来启动一个事务，总是由主机发送，令牌包是必需的；

数据包传输数据，可以从主机到设备，也可以从设备到主机，方向由令牌包来决定。

握手包通常由数据接收端发送，当数据接收正确后，发送握手包。设备也可以使用 NAK 握手包表示数据还未准备好。

下面具体讲解三种典型的事务处理：SETUP 事务处理、IN 事务处理和 OUT 事务处理。

4.2.1 SETUP 事务处理

SETUP 事务处理是一种特殊的 USB 事务处理，它只在 USB 控制传输中使用，用于对 USB 设备进行配置。SETUP 的数据传输方向是主机到设备。一个完整的成功的 SETUP 事务处理流程如下所示：

表 4-3 SETUP 事务处理

1：主机->设备（令牌包）	SYNC	SETUP	ADDR	ENDP	CRC5	EOP
2：主机->设备（数据包）	SYNC	DATA0	数据		CRC16	EOP
3：主机<-设备（握手包）	SYNC	ACK	EOP			

4.2.2 IN 事务处理

IN 事务用于实现 USB 主机向 USB 设备要数据，一个完整的成功的 IN 事务处理流程如下所示：

表 4-4 IN 事务处理

1：主机->设备（令牌包）	SYNC	IN	ADDR	ENDP	CRC5	EOP
2：主机<-设备（数据包）	SYNC	DATA0/DATA1	数据		CRC16	EOP
3：主机->设备（握手包）	SYNC	ACK	EOP			

在实际数据传输过程中，可能会出现错误，设备和主机会根据具体情况，采用不同的处理方法。

（1）对于 USB 设备，当主机向设备发送的 IN 令牌包在传输过程中被损坏时，设备接收不到正确的 IN 令牌包，就会忽略该令牌包，不应答。

（2）对于 USB 设备，如果设备接收到正确的 IN 令牌包，但设备的 IN 端点被停止，无法向主机发送数据，此时设备向主机发送 STALL 握手包。

（3）对于 USB 设备，如果设备接收到正确的 IN 令牌包，但设备由于某种原因无法向提供发送数据，此时设备向主机发送 NAK 握手包。

（4）对于 USB 主机，当 USB 设备向主机发送的数据包在传输过程中被损坏，主机接收不到正确的数据包时，主机将忽略该出错的数据包，不做响应。

4.2.3 OUT 事务处理

OUT 事务用于实现 USB 主机到设备的数据传输，一个完整的成功的 OUT 事务处理流程如下所示：

表 4-4 OUT 事务处理

1：主机->设备（令牌包）	SYNC	OUT	ADDR	ENDP	CRC5	EOP
2：主机->设备（数据包）	SYNC	DATA0/DATA1	数据		CRC16	EOP
3：主机<-设备（握手包）	SYNC	ACK	EOP			

如果主机在向设备传送数据的过程中发生错误，设备会有如下响应：

（1）当 OUT 令牌包在传输的过程中被损坏，设备接收不到正确的令牌包是，设备会忽略该令牌包，不做应答

（2）如果设备接收到正确的 OUT 令牌包，但是数据包在传输的过程中被损坏，设备将忽略该数据包，不应答；

（3）如果设备接收到正确的 OUT 令牌包，但是设备的 OUT 端点被停止，无法接收主机发来的数据，那么设备向主机发送 SALL 握手包。

（4）如果设备接收到正确的 OUT 令牌包，但是设备由于某种原因无法接收主机发送的数据，那么设备将向主机发送 NAK 握手包。

（5）如果设备的数据触发位和接收到的数据包的触发位不一致，那么设备将丢弃该数据包，然后向主机发送 ACK 握手包。

4.3 USB 的传输类型

在 USB 传输中，制定了 4 种传输类型：批量传输、中断传输、同步传输以及控制传输。

批量传输用于传输大量数据，要求传输不能出错，但对时间没有要求，适用于打印机、存储设备等。

中断传输总是用于对设备的查询，以确定是否有数据需要传输。因此，中断传输的方向总是从 USB 设备到主机，是用于 USB 鼠标、键盘灯设备。

同步传输要求数据以固定速率抵达或在指定时刻抵达，可以容忍偶尔错误的数据，一般用于麦克风、喇叭等设备。

控制传输主要用于传输少量的数据，对传输时间和传输速率没有要求，它是 USB 传输中最重要的传输，只有正确的执行完控制传输，才能进一步正确的执行其他传输模式。

当主机上电检测到 USB 设备时，首先经历的是控制传输。在该过程，主机要读取设备相关描述符，确定该设备的类型以及操作特性。另外，主机还对该设备进行相应的配置。每个 USB 设备都有一个默认的 0 号控制端点。

4.3.1 控制传输

控制传输是一种可靠的双向传输，一次控制传输可分为三个阶段：设置阶段、数据传输阶段（无数据控制就没有此阶段）以及状态阶段。

控制传输通过控制管道在应用软件和 Device 的控制端点之间进行，控制传输过程中传输的数据是有格式定义的，USB 设备或主机可根据格式定义解析获得的数据含义。其他三种传输类型都没有格式定义。

控制传输对于最大包长度有固定的要求。对于高速设备该值为 64Byte；对于低速设备该值为 8；全速设备可以是 8 或 16 或 32 或 64。

高速端点的控制传输不能占用超过 20%的微帧，全速和低速的则不能超过 10%的帧。在一帧内如果有多余的未用时间，并且没有同步和中断传输，可以用来进行控制传输。

下面详细讲解控制传输的三个阶段。

4.3.1.1 设置阶段

第一阶段为设置阶段，即从 HOST 到 Device 的 SETUP 事务传输，这个阶段指定了此次控制传输的请求类型。USB 设备在正常使用之前，必须先通过端点 0 进行配置。在本阶段，主机将会对 USB 设备的请求信息（如：读设备描述符）传送给 USB 设备，从 USB 设备获取配置信息后确定此设备有哪些功能。作为配置的一部分，主机还会设置设备的配置值。设置阶段包含了 SETUP 令牌包、紧随其后的 DATA0 数据包（该数据包里的数据即为设备请求，固定为 8 个字节）以及 ACK 握手包，关于设备请求，将会在下面的章节讲解，图 4-2 给出了 SETUP 事务建立的流程图。

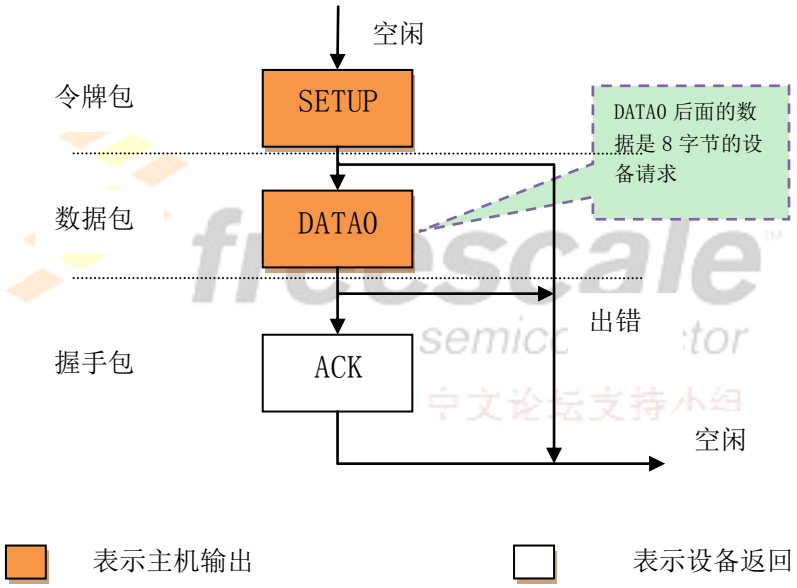


图 4-2 SETUP 事务建立流程图

4.3.1.2 数据传输阶段

第二阶段为数据传输阶段，这个阶段是可选的，即一个控制传输可能没有数据过程。如果有，一个数据过程可以包含一笔或者多笔的数据事务。需要注意的是，在数据传输过程中，所有的数据事务必须是同一个传输方向的，一旦数据方向改变，就认为进入到了状态阶段，数据过程中的第一个数据包必须是 DATA1 包，然后每次正确传输一个数据包后就在 DATA0 和 DATA1 之间交替。

根据数据传输的方向，控制传输又可以分为 3 种类型：控制读取（读取 USB 描述符）、控制写入（配置 USB 设备）以及无数据控制。

控制读取是将数据从设备读到主机，读取的数据包括 USB 设备描述符等内容。该过程先由主机向设备发送 IN 令牌包，表明接下来主机要从设备读取数据。然后，设备向主机发送 DATA1 数据包。最后，主机将以下列的方式加以响应：若数据被正确接收，主机发送 ACK 数据包；若主机正在忙碌，则发送 NAK 握手包；若发生了传输错误，主机发送 STALL 握手包。

控制写入则是将数据从主机传到设备上，所传的数据即为对 USB 设备的配置信息，在该过程中，主机将会发送一个 OUT 令牌包，表明主机将要把数据发送到设备。紧接着，主机将数据通过 DATA1 数据包传递至设备。最后，设备将以下列方式加以响应：若数据被正确接收，设备发送 ACK 握手包；若设备正在忙碌，则发送 NAK 握手包；若发生了传输错误，设备发送 STALL 握手包。

图 4-3 给出了几种数据传输的实例。

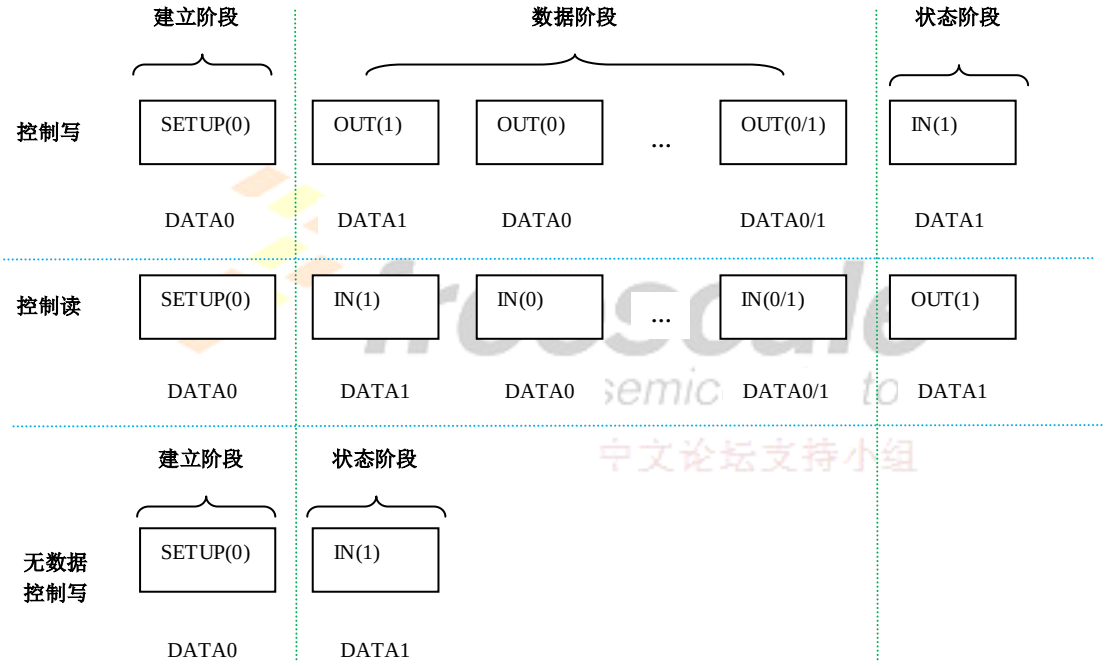


图 4-3 几种控制传输的例子

4.3.1.3 状态阶段

第三阶段为状态阶段，表示整个控制传输的过程已经完全结束，它的传输方向和数据阶段相反，即若数据阶段是 IN 令牌包，则状态阶段为 OUT 令牌包；若数据阶段为 OUT 令牌包，则状态阶段为 IN 令牌包。状态阶段只使用 DATA1 包。

对于控制读取，在状态阶段，主机会发送 OUT 令牌包，其后跟数据为 0 长度的 DATA1 数据包。而此时，设备也会做出相应的动作，向主机发送 ACK 握手包，NAK 握手包或 STALL 握手包。

对于控制写入，在状态阶段，主机会发送 IN 令牌包，然后设备发送一个数据为 0 长度的 DATA1 数据包，主机再做出相应的动作：发送 ACK 握手包、NAK 握手包或 STALL 握手包。

4.3.2 批量传输

批量传输是一种可靠的单向传输，但延迟没有保证，它尽量利用可以利用的带宽来完成传输，适合数据量比较大的传输。

低速 USB 设备不支持批量传输，高速批量端点的最大包长度为 512，全速批量端点的最大包长度可以为 8、16、32、64。

批量传输在访问 USB 总线时，相对其他传输类型具有最低的优先级，USB HOST 总是优先安排其他类型的传输，当总线带宽有富余时才安排批量传输。

高速的批量端点必须支持 PING 操作，向主机报告端点的状态，NYET 表示否定应答，没有准备好接收下一个数据包，ACK 表示肯定应答，已经准备好接收下一个数据包。

批量传输使用批量事务传输数据。一次批量事务有三个阶段：令牌包阶段、数据包阶段和握手包阶段。批量传输分为批量读和批量写，批量读使用批量输入事务，批量写使用批量输出事务，图 4-4 为批量事务流程图，其中 IN 为批量输入事务，OUT 为批量输出事务，PING 令牌包是用于 USB2.0 高速设备中的。

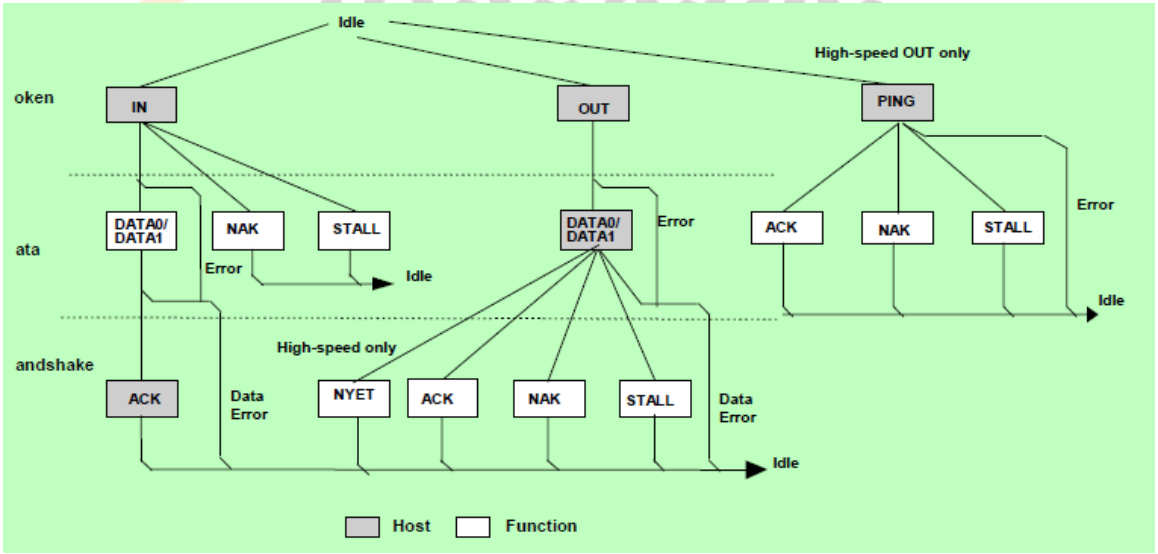


图 4-4 批量事务流程图

批量输出事务，主机先发一个 OUT 令牌吧，这个令牌包中包含了设备地址、端点号。然后，再发送一个 DATA 包，具体的包类型看数据切换，这是地址和端点匹

配的设备就会收下这个数据包。然后主机切换到接收模式，等待设备返回握手包。设备也会做出相应的动作，向主机发送 ACK 握手包，NAK 握手包或 STALL 握手包。

批量输入事务，主机首先发出一个 IN 令牌包，同样，IN 令牌包中包含了设备地址和端点号。然后主机切换到接收数据状态，等待设备返回数据。最后主机再做出相应的动作：发送 ACK 握手包、NAK 握手包或 STALL 握手包。

PING 事务是用于 USB2.0 高速设备中的，它不发出数据，直接等待设备的握手包。因此 PING 事务只有令牌包和握手包。

表 4-5 和 4-6 给出了一次正确的批量输入事务和输出事务，以 2 个字节为例，其实就是之前讲过的 IN 事务和 OUT 事务。

表 4-5 一次正确的批量输入事务

主机发送	SYNC	IN	ADDR	ENDP	CRC5	EOP
设备返回	SYNC	DATA0/DATA1	数据（字节 0+字节 1）		CRC16	EOP
主机应答	SYNC	ACK	EOP			

表 4-6 一次正确的批量输出事务

主机发送	SYNC	OUT	ADDR	ENDP	CRC5	EOP
主机发送	SYNC	DATA0/DATA1	数据（字节 0+字节 1）		CRC16	EOP
设备应答	SYNC	ACK	EOP			

4.3.3 中断传输

中断传输是一种轮询的传输方式，是一种单向的传输，HOST 通过固定的间隔对中断端点进行查询，若有数据传输或可以接收数据则返回数据或发送数据，否则返回 NAK，表示尚未准备好。中断传输的延迟有保证，但并非实时传输，它是一种延迟有限的可靠传输，支持错误重传。

对于高速/全速/低速端点，最大包长度分别可以达到 1024/64/8 Bytes。高速中断传输不得占用超过 80%的微帧时间，全速和低速不得超过 90%。中断端点的轮询间隔由在端点描述符中定义，全速端点的轮询间隔可以是 1~255mS，低速端点为 10~255mS，高速端点为(2*interval-1)*125uS，其中 interval 取 1 到 16 之间的值。除高速高带宽中断端点外，一个微帧内仅允许一次中断事务传输，高速高带宽端点最多可以在一个微帧内进行三次中断事务传输，传输高达 3072 字节的数据。

除了在对端点查询的策略上不一样之外，中断传输和批量传输的结构基本上是一样的，只是中断传输中没有 PING 和 NYET 两种包。中断传输使用中断事务，流程图如图 4-5 所示。

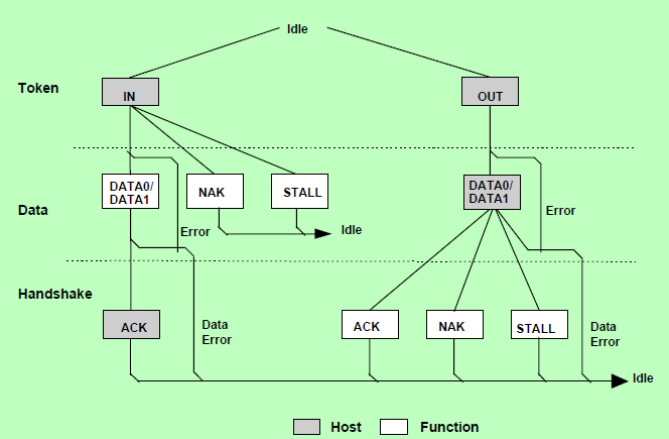


图 4-5 中断事务处理流程

4.3.4 同步传输

同步传输是一种实时的、不可靠的传输，不支持错误重发机制，当数据错误的时候，并不进行重传操作，因此同步传输也没有应答包，数据的正确与否完全靠 CRC 校验。只有高速和全速端点支持同步传输，高速同步端点的最大包长度为 1024，全速的为 1023。

除高速高带宽同步端点外，一个微帧内仅允许一次同步事务传输，高速高带宽端点最多可以在一个微帧内进行三次同步事务传输，传输高达 3072 字节的数据。全速同步传输不得占用超过 80%的帧时间，高速同步传输不得占用超过 90%的微帧时间。 同步端点的访问也和中断端点一样，有固定的时间间隔限制。在主机控制器和 USB HUB 之间还有另外一种传输——分离传输（Split Transaction），它仅在主机控制器和 HUB 之间执行，通过分离传输，可以允许全速/低速设备连接到高速主机。分离传输对于 USB 设备来说是透明的、不可见的。

下面给出同步传输的事务流程。

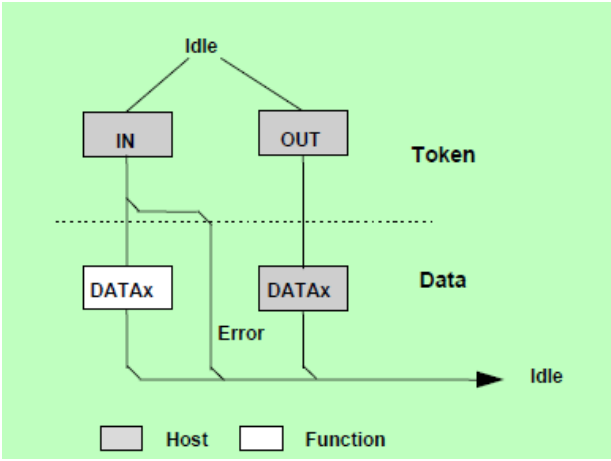


图 4-6 同步事务处理流程

4.3.5 端点类型与传输类型的关系

一个具体的端点，只能工作在一种传输模式下。通常，我们把工作在什么模式下的端点就叫做什么端点，比如控制模式下的端点就叫控制端点。

端点 0 是每个 USB 设备都必须具备的默认控制端点，它一上电就存在并且可用。设备的各种描述符以及主机发送的一些命令，都是通过端点 0 传输的。其他端点是可选的，需要根据具体的设备来决定。非 0 端点只有在 Set Config 之后才能使用。

4.4 USB 的设备请求

在 USB 协议中，主机对 USB 设备的各种配置操作是通过设备请求来实现的。设备请求是在控制传输的 SETUP 阶段由主机发往设备的，通常在默认控制管道上传输，它的各个字段由主机定义，表达了每一次控制传输的目的。

4.4.1 设备请求格式

一个设备请求有 8 个字节，这 8 个字节的数据是在控制传输的建立过程中通过控制端点 0 发出，包含了数据过程所需要的传输数据的方向、长度以及数据类型等信息。它的一般格式如表 4-7 所示。

表 4-7 设备请求格式

偏移量/字节	域	字节数	取值	描述
0	bmRequestType	1	位图	请求的特性 D7：数据传输方向 0=主机到设备 1=设备到主机 D6~5：请求的类型 0=标准 1=类 2=厂商 3=保留 D4~0：请求的接受者 0=设备 1=接口 2=端点 3=其他 4~31=保留
1	bRequest	1	数值	请求代码

2	wValue	2	数值	由具体的请求决定
4	wIndex	2	索引	由具体的请求决定
6	wLength	2	字节数	数据过程所需要传输的字节数，0 表示无数据阶段

4.4.1.1 bmRequestType 域

这个域表明此请求的特性。特别地，这个域表明了第二阶段控制传输方向。如果 wLength 域被设作 0 的话，表明没有数据传送阶段，那 Direction 位就会被忽略.

D[6：5]表示了该请求所属的类型，USB 标准中定义了所有 USB 设备必须支持的标准请求。一个设备类可定义更多的请求。设备厂商也可定义设备支持的请求.

D[4：0]表示接收端，请求可被导引到设备，设备接口，或某一个设备端结点(endpoint)上。当指定的是接口或端结点(endpoint)时，wIndex 域指出那个接口或端节点

4.4.1.2 bRequest 域

这个域表示特定请求，下面讲解 USB 的标准请求，即 bmRequestType 的 D6~D5 位为 00 的请求。USB 协议定义了 11 种标准请求，通过请求代码 bRequest 来确定，表 4-8 即各标准请求的名字以及请求代码。标准请求的功能主要得到设备描述、设置地址、得到配置描述等。所有 USB 设备均应支持这些请求。HOST 通过标准请求来识别和配置设备。

表 4-8 标准请求及其请求代码

请求名字（bRequest）	请求代码
GET_STATUS	0
CLEAR_FEATURE	1
SETFEATURE	3
SETADDRESS	5
GETDESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

4.4.1.3 wValue 域

此域用来传送当前请求的参数，随请求不同而变。

4.4.1.4 wIndex 域

wIndex 域用来表明是哪一个接口或端点，图 4-7 表明 wIndex 的格式(当标识端点时)。Direction 位在设为 0 时表示输出端点点，设为 1 时表示是输入端点，Endpoint Number 是端点号。图 4-8 表明 wIndex 用于标识接口时的格式。

D7	D6	D5	D4	D3	D2	D1	D0
Direction	Reserved (Reset to zero)			Endpoint Number			
D15	D14	D13	D12	D11	D10	D9	D8
Reserved (Reset to zero)							

图 4-7 端点时 wIndex 格式

D7	D6	D5	D4	D3	D2	D1	D0
Interface Number							
D15	D14	D13	D12	D11	D10	D9	D8
Reserved (Reset to zero)							

图 4-8 接口时 wIndex 格式

4.4.1.5 wLength 域

这个域表明第二阶段的数据传输长度。传输方向由 bmRequstType 域的 Direction 位指出。wLength 域为 0 则表明无数据传输。在输入请求下，设备返回的数据长度不应多于 wLength，但可以少于。在输出请求下，wLength 指出主机发出的确切数据量。如果主机发送多于 wLength 的数据，设备做出的响应是无定义的 .

4.4.2 标准请求结构及讲解

从 4.4.1.2 节可以知道，标准请求一共有 11 种，下面具体讲解各请求的结构以及内容。

不同的请求对于其接受者、wValue 和 wIndex，其各字段的意义都是不一样的。表 4-9 是各个标准请求的机构以及数据过程需要传输的数据。其中第一列有的有多个，主要是最低 5 位不同，即表示接受者不同。有的请求只能发送到设备，而有的请求可以发送到设备、接口和端点。常用的请求有：GET_DESCRIPTOR、SET_ADDRESS 和 SET_CONFIGURATION。

表 4-9 各标准请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000000B	CLEAR_FEATURE (0X01)	特性选择	0	0	无
00000001B			接口号		
00000010B			端点号		
10000000B	GET_CONFIGURATION （0X08）	0	0	1	配置值
10000000B	GET_DESCRIPTOR （0X06）	描述符类型和索引	0 或者语言 ID	描述符的长度	描述符
10000001B	GET_INTERFACE （0X0A）	0	接口号	1	备用接口号
10000000B	GET_STATUS （0X00）	0	0	2	设备、接口或者端点状态
10000001B			接口号		
10000010B			端点号		
00000000B	SET_ADDRESS （0X05）	设备地址	0	0	无
00000000B	SET_CONFIGURATION （0X09）	配置值	0	0	无
00000000B	SET_DESCRIPTOR （0X07）	描述符类型和索引	0 或者语言 ID	描述符的长度	描述符
00000000B	SET_FEATURE （0X03）	特性选择	0	0	无
00000001B			接口号		
00000010B			端点号		
00000001B	SET_INTERFACE （0X0B）	备用接口号	接口号	0	无
10000010B	SYNCH_FRAME （0X0C）	0	端点号	2	帧号

4.4.2.1 GET_DESCRIPTOR 请求

GET_DESCRIPTOR 获取描述符请求是在枚举过程中用的最多的一个请求，如表 4-10。用于获取描述符，IN 传输，wValue 高字节为描述符类型，例如，设备描述符、配置描述符等等，具体请查看表 3-1，wValue 低字节用于选择特殊的描述符，例如设备有多个配置描述符、或者多个字符串描述符时需要使用该参数去指定索引，

描述符索引值的范围从 0 开始到设备使用该类型描述符的数量减 1，wIndex 为字符串描述符中的 LanguageID 号，wLength 为请求的数据长度，例如，读取设备描述符请求字段如下：

0x80　　0x06　　0x0100　　0x00　　0x0012

表 4-10 GET_DESCRFIPTOR 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
10000000B	GET_DESCRIPTOR（0X06）	描述符类型和索引	0 或者语言 ID	描述符的长度	描述符

设备在收到获取描述符的请求后，应该按照所请求的描述符类型编号，在数据传输阶段返回相应的描述符。对于全速模式和低速模式，获取描述符的标准请求只有三种：获取设备描述符、获取配置描述符和获取字符串描述符。另外的接口描述符和端点描述符时跟随配置描述符一并返回，不能单独请求返回，如果单独返回，主机无法确认它们属于哪个配置。

在请求结构中，wValue、wIndex、wLength 这三个域是两个字节，USB 协议中规定的是小端结构，即低字节在前，高字节在后。

各状态下，该请求的情况：

缺省状态：此请求合法。

地址状态：此请求合法。

配置状态：此请求合法。

4.4.2.2　SET_ADDRESS 请求

设置地址请求 SET_ADDRESS 用于给 USB 设备设置地址，从而可以对该 USB 设备进行进一步的访问，该请求无数据阶段。设置地址请求的 bmRequsetType 和 bRequest 的值分别为 00000000B 和 05H，wValue 的值为新的设备地址，如表 4-11 所示。该请求与其他请求有一个重要的不同点，即使在该请求下，USB 设备一直不改变它原来的地址，知道该请求的状态阶段被成功的完成，而其他请求的操作都是在状态阶段之间完成。wIndex 和 wLength 的值一般为 0，若给定的设备地址大于 127，或者 wIndex 或 wLength 为非 0 值，那么该请求不执行。

各状态下，该请求的情况：

缺省状态：如果地址值非 0，那设备将进入地址状态，否则地址仍留在缺省态(此非出错状态)。

地址状态：如果新地址值为 0，进入缺省态，否则仍留在地址状态但使用新地址。

配置状态：在此状态下设备对此请求的响应无定义 。

表 4-11 SET_ADDRESS 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000000B	SET_ADDRESS（0X05）	设备地址	0	0	无

4.4.2.3 SET_CONFIGURATION 请求

设置配置请求 SET_CONFIGURATION 用于为 USB 设备设置一个合适的配置值。该请求的 bmRequestType 和 bRequest 值分别为 00000000B 和 09H，如表 4-12 所示，wValue 域的低字节表示设置的配置值，该值为 0 或配置描述符中的配置值相匹配，如果该值与某配置描述符中的配置编号一致时，表示选中该配置，该值通常为 1，因为大多数 USB 设备只有一种配置，配置编号为 1；如果该值为 0，则会让设备进入设置地址状态。设备只有在收到非 0 的配置之后，才能启用他的非 0 端点。wValue 域的高字节保留。wIndex 和 wLength 的值一般为 0，如果这两个域有一个为非 0 值，那么该请求不执行。设置配置请求无数据阶段。

各状态下，该请求的情况：

缺省状态：设备响应无定义

地址状态：如果所指的配置为 0，设备停留在地址状态。如果所指的配置与描述表中的一个值相匹配，那个配置就被选中，设备转到配置状态。否则，返回请求错误

配置状态：如果配置值为 0，设备进入地址状态。如果配置值非 0 并与描述表中的一个配置相匹配则设备仍留在配置态，但采用新的配置值，否则返回请求错误。

表 4-12 SET_CONFIGURATION 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000000B	SET_CONFIGURATION（0X09）	配置值	0	0	无

4.4.2.4 CLEAR_FEATURE 请求

这个请求用来清除或使否某个指定的属性。wValue 必须根据接收者来选择特定的值。接收者是设备就用设备描述符，是接口就用接口描述符，是端口就用端口描述符。一个 Clear Feature 请求一个不能被清除或不存在的特性，或指的是一个不存在的接口或端口的时候，将返回一个 Request Error。如果 wLength 不为 0，则设备响应无定义。

各状态下，该请求的情况：

缺省状态下，此时对请求的响应无定义。

地址状态下，此时对请求的响应有定义，但是不能是对接口或者端口的请求，只能 是端口 0，否则响应为 Request Error。

配置状态下，此时对请求的响应有定义。

表 4-13 CLEAR_FEATIRE 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000000B	CLEAR_FEATURE	特性选择	0	0	无
00000001B	(0X01)		接口号		
00000010B			端点号		

4.4.2.5 GET_CONFIGURATION 请求

这个请求用来得到当前设备的配置值。如果 wValue,wIndex,wLength 不是表 4-14 中定义的默认值，那么设备响应无定义。如果返回 0 值表明设备未配置。

各状态下，该请求的情况：

缺省状态下，设备响应无定义。

地址状态下，返回 0 值。

配置状态下，非 0 的 bConfiguration 值被返回。

表 4-14 GET_CONFIGRATION 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
10000000B	GET_CONFIGURATION (0X08)	0	0	1	配置值

4.4.2.6 GET_INTERFACE 请求

这个请求返回选定接口的可选配置。有些 USB 有接口间互斥设置的配置。这个请求使得主机决定设置。wValue 和 wLength 不是表 4-15 定义的默认数值，设备响应无定义。如果所指的接口不存在，就会返回请求错误。

各状态下，该请求的情况：

缺省状态下，此时设备对此请求响应无定义。

地址状态下，设备返回请求错误。

配置状态下，请求合法。

表 4-15 GET_INTERFACE 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
10000001B	GET_INTERFACE（0X0A）	0	接口号	1	备用接口号

4.4.2.7 GET_STATUS 请求

这个请求返回选定接收者的状态。bmRequestType 中的 Recipient 位指明接收者。返回值是指定接收者的状态。如果 wValue 和 wLength 不是表 4-16 中规定的默认值或者 wIndex 在取设备状态时，设备响应无定义。

各状态下，该请求的情况：

缺省状态下，设备响应无定义。

地址状态下，如果所指的是接口或者一个非 0 的端口，设备返回请求出错。

配置状态下，如果所指的接口或端口不存在，返回请求错误。

表 4-16 GET_STATUS 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
10000000B	GET_STATUS（0X00）	0	0	2	设备、接口或者端点状态
10000001B			接口号		
10000010B			端点号		

4.4.2.8 SET_DESCRIPTOR 请求

设置描述符。此请求可选，以用于添加或更新新的描述表。wValue 的高字节指明描述符种类，低字节指明描述符索引。描述符索引用来给出正确的描述符。因

为一个设备可以有不同的描述符。对于可以通过 SetDescriptor()设定描述表的，其描述符索引一定为 0。描述符索引的范围从 0 到描述符范围值减 1。wIndex 指出字符串描述符 Language ID，对于其他描述符设置为 0。wLength 指明传输数据长度。设备支持的描述符有设备，配置和字符串这 3 种描述符。如果设备不支持该请求，返回一个请求错误。

各状态下，该请求的情况：

缺省状态下，设备对请求响应无定义。

地址状态下，如果支持请求，则合法。

配置状态下，如果支持请求，则合法。

表 4-17 SET_DESCRIPTOR 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000000B	SET_DESCRIPTOR（0X07）	描述符类型和索引	0 或者语言 ID	描述符的长度	描述符

4.4.2.9 SET_FEATURE 请求

这个请求用于设置或使能一个指定的特性，结构如表 4-18。 wValue 域中的特性选择符必须跟接收者相配。当接收器是设备时，只能使用设备特性选择符的值；当接收器是接口时，只能使用接口特性选择符；当接收器是端点时,只能是使用端点符特性选择符。各种接收器定义的特性选择符的值参考表 4-19。

表 4-18 SET_FEATURE 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000000B	SET_FEATURE（0X03）	特性选择	0	0	无
00000001B			接口号		
00000010B			端点号		

表 4-19 标准特性选择。

特性选择符	接收端	值
DEVICE_REMOTE_WAKEUP	设备	1
ENDPOINT_HALT	端点	0
TEST_MODE	设备	2

TEST_MODE 特性只在设备接收器定义(即 bmRequest Type=0)，而且 wIndex 的低字节必须为 0。置位 TEST_MODE 特性会使设备的上行口进入测试模式。如果请求包含无效的测试选择符，设备将用请求错误响应。切换到测试模式必须在请求状态阶段结束后的 3ms 内完成。直到请求的状态阶段结束后，上行口才能切换到测试模式。设备必须重启电源才能退出设备上行口的测试模式。设备必须在默认，寻址或配置的高速状态中支持 TEST_MODE 特性。

SetFeature 请求如果指出一个不存在的特性会使得设备在交换状态阶段返回 STALL 信号。

如果选择特性是 TEST_MODE，那么 Windex 的最高字节用于指明待定的测试模式。SetFeature(TEST_MODE…)的接收器必须是设备，即 wIndex 的低字节必须是 0，而且 bmRequestType 也必须设置为 0。设备必须重启电源以退出测试模式。有效的测试模式选择符在表 4-20 列出.

表 4-20 测试模式选择

Value	Description
00H	Reserved
01H	Test_J
02H	Test_K
03H	Test_SE0_NAK
04H	Test_Packet
05H	Test_Force_Enable
06H-3FH	Reserved for standard test selectors
3FH-BFH	Reserved
C0H-FFH	Reserved for vendor-specific test modes.

如果 wLength 非 0，那么设备的行为不确定；如果指定的端点或接口不存在，则设备会以请求错误响应。

各状态下，该请求的情况：

默认状态，当处于默认状态时，设备必须能接收 SetFeature（ TEST_MODE ，TEST_SELECTOR）请求.设备处于默认状态时接收到其他 SetFeature 请求,设备的行为不确定。

寻址状态，如果指定了一个接口或端点(端点 0 出外),设备用请求错误响应。

配置状态，当设备处于配置状态时,这是一个有效的请求。

4.4.2.10 SET_INTERFACE 请求

这个请求允许主机为指定的接口选择另一个设置。

如有 USB 设备接口配置中有互斥设置，此请求让主机选择所要的设置。如果设备的接口只支持缺省设置，在状态交换阶段设备返回 STALL。这个请求不能用于改变已配置的接口的集合(必须使用 SetConfiguration()请求)。

如果接口或者备用设置不存在,则设备必须以请求错误响应.如果 wLength 的值非 0，那么设备的行为不确定。

各状态下，该请求的情况：

默认状态，当设备处于默认状态时接收到这个请求,设备的行为不确定。

寻址状态，设备必须以请求错误响应 。

配置状态，当设备处于配置状态时,这是一个有效的请求。

表 4-21 SET_INTERFACE 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
00000001B	SET_INTERFACE（0X0B）	备用接口号	接口号	0	无

4.4.2.11 SYNCH_FRAME 请求

这个请求用于设置和报告端点的同步帧

如果一个端节点支持同步传输，端节点可能会根据某一特点的模式来以变长方式传送每一帧。主机与端节点必须在什么时候出现重复模式的第一帧出现上达成一致。模式开始帧的序号由设备返回给主机。

如果高速设备支持同步帧请求，它必须在内部与第 0 号微型帧同步，并且对标准帧有时间概念。只有帧号码用于且由设备端点报告(即无微型帧号码)。端点必须与第 0 号微型帧同步。

这个值仅用于隐式模式的同步数据传输。如果 wValue 非 0 或 wLength 非 2，设备响应无定义。

如果所指的端节点不支持此请求，设备返回一个请求错误。

各状态下，该请求的情况：

缺省状态，设备响应无定义 。

寻址地址，设备返回请求错误 。

配置状态，此请求合法 。

表 4-22 SYNCH_FRAME 请求结构

bmRequestType	bRequest	wValue	wIndex	wLength	数据过程
10000010B	SYNCH_FRAME（0X0C）	0	端点号	2	帧号

4.5 USB 设备枚举过程

4.5.1 枚举过程中的状态

当 USB 设备插上主机时，主机就通过一系列的动作来对设备进行枚举配置(配置是属于枚举的一个态，态表示暂时的状态)，这些态如下：

（1）接入态(Attached)：设备接入主机后，主机通过检测信号线上的电平变化来发现设备的接入;

（2）供电态(Powered)：就是给设备供电，分为设备接入时的默认供电值，配置阶段后的供电值(按数据中要求的最大值，可通过编程设置)

（3）缺省态(Default)：USB 在被配置之前，通过缺省地址 0 与主机进行通信;

（4）地址态(Address)：经过了配置，USB 设备被复位后，就可以按主机分配给它的唯一地址来与主机通信，这种状态就是地址态;

（6）配置态(Configured)：通过各种标准的 USB 请求命令来获取设备的各种信息，并对设备的某此信息进行改变或设置。

（7）挂起态(Suspended)：总线供电设备在 3ms 内没有总线操作，即 USB 总线处于空闲状态的话，该设备就要自动进入挂起状态，在进入挂起状态后，总的电流功耗不超过 280UA。

4.5.2 枚举过程讲解

当有 USB 设备连接到主机时，主机自动对设备进行枚举。

枚举过程如下：

（1）用户将一个 USB 设备连接到 USB 端口，或者系统带着一个 USB 设备上电启动。该 USB 端口可能是主机上的根 Hub 也可能是一个连接到主机下行端口上的 Hub。Hub 向主机汇报 Hub 状态改变，且会为端口提供电源，使设备得到稳定的 Vbus，而进入上电状态。此时该 Hub 端口是未使能的。

（2）HUB 检测设备。Hub 检测它的每个端口的数据线上的电压。平时两根数据线都被拉低，处于 SE0 状态。当有设备连接上之后，设备会将数据 D+或 D-上的电压拉高（当连上全速或高速设备，D+被拉高，当连上低速设备，D-被拉高）。这样

Hub 就能检测到有设备接入。检测到设备后，Hub 继续向设备提供电源，但是不传输数据。

（3）主机检测到新设备。Hub 会使用它的中断端点向主机报告，是否有一个端点发生事件。主机得知事件发生，主机发送 Get_status 到根集线器来获得当前端口的状态。USB2.0 要求在 Hub 复位设备之前，检测该设备是低速、全速或高速设备。

（4）HUB 复位设备。当主机检测到一个新设备时，主机发送 Set_Feature,让根集线器复位端口,使得端口上的设备处于复位状态。Hub 使得设备的 USB 数据线保持复位条件（SE0 状态）至少 10ms。

（5）主机检测是否全速设备支持高速。在复位器件，支持高速的设备发送一个 K 状态，具有高速能力的 Hub 检测到该状态并相应一个 K 和 J 的交替状态序列。设备就将提高为全速，并且以后都用高速通信。如果 Hub 不响应设备的 K 状态，那么设备就是全速通信。

（6）HUB 为设备建立一条连向 USB 总线的通路。主机发送 Get_status 检测端口的复位是否完成,如果完成,设备现在处于默认状态,并准备好使用端点 0 进行控制传输,这时候设备的地址为 0，同时从 Vbus 汲取至少 100mA 的电能。

（7）主机发送 Get_Descriptor 请求，以获得设备描述符（第 1 次），旨在得到端点 0 所支持的最大数据包长度（设备描述符的第 8 个字节）。这是一个 SETUP 事务-》IN 事务-》OUT 事务过程，OUT 事务一结束，主机就要求 Hub 复位设备。

（8）主机给从机分配地址。主机向设备发送 Set_Address 来给设备设定一个新的地址。设备使用默认地址完成状态阶段后，以后的所有通信都使用新地址。该新地址一直有效，直到设备移除、端口复位或系统重启。

（9）主机获悉设备功能。主机向设备发送 Get_Device_Descriptor 获得完整的设备描述符。此后，主机继续请求在设备描述符中指定的配置描述符。对于配置描述符的请求实际上是对配置描述符本身及其下层的描述符（包括接口描述符、端点描述符）的请求。主机先获得 9 个字节的配置描述符本身，其中包括了配置描述符以及下层的所有描述符的总长度。之后，将再次请求配置描述符，这一次将获得完整的配置、接口和端点描述符。

（10）主机弹出消息，显示“产品字符串”，然后分配并加载设备驱动。

（11）主机设备驱动为设备选择一个合适的配置值。在通过描述符获悉设备状况之后，设备驱动程序发送一个带有所需配置号的 Set_Configuration 请求。设备督导请求并使能所要求的配置。这是，设备就处于配置状态，并且设备接口被允许。

4.6 USB 分析仪查看总线数据包

开发 USB 设备，使用 USB 协议分析仪可以更高效的帮助开发人员调试。本文使用的 USB 分析仪是 LeCroy 的 Advisor T3。下面讲解如何使用 Advisor T3 实现 USB 数据的抓取以及分析。

4.6.1 USB 分析仪接线

USB 分析仪其实是串联在所测试的设备以及电脑之间，接线如下图所示：

图 4-9 USB 分析仪测试连接

4.6.2 USB 分析仪软件抓取配置

首先需要配置软件，点击下面红框中的工具。

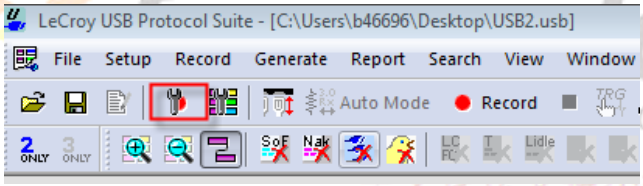


图 4-10 抓取配置按钮

接下来，配置具体的选项，选择抓取通道为 USB2.0，产品型号选择为 advisor T3，其余默认。

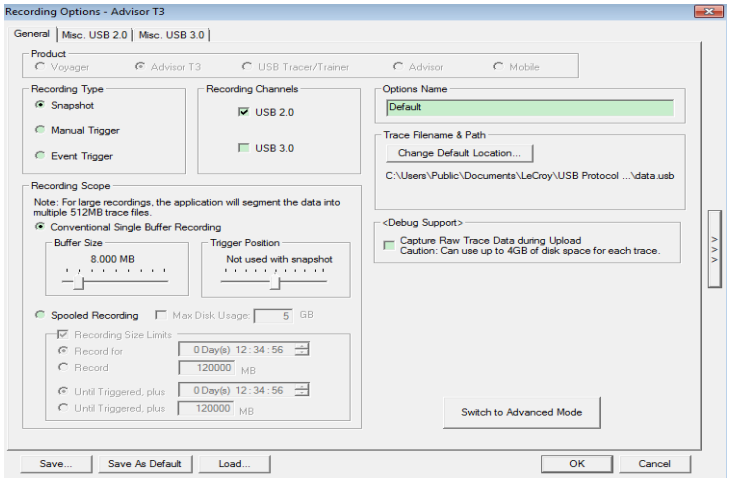


图 4-10 抓取配置

最后，连接好所测设备后，点击 record，开始抓取

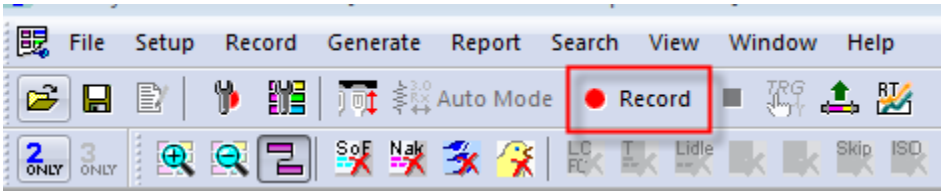


图 4-11 点击采集按钮

需要结束抓取，可以点击 Record 右边的方框实现停止采集。

4.6.3 USB 分析仪数据分析

下面以分析仪抓取到的第一个控制传输流程为例，说明抓取到的数据结构。

图 4-12 即为实际抓到的数据情况。

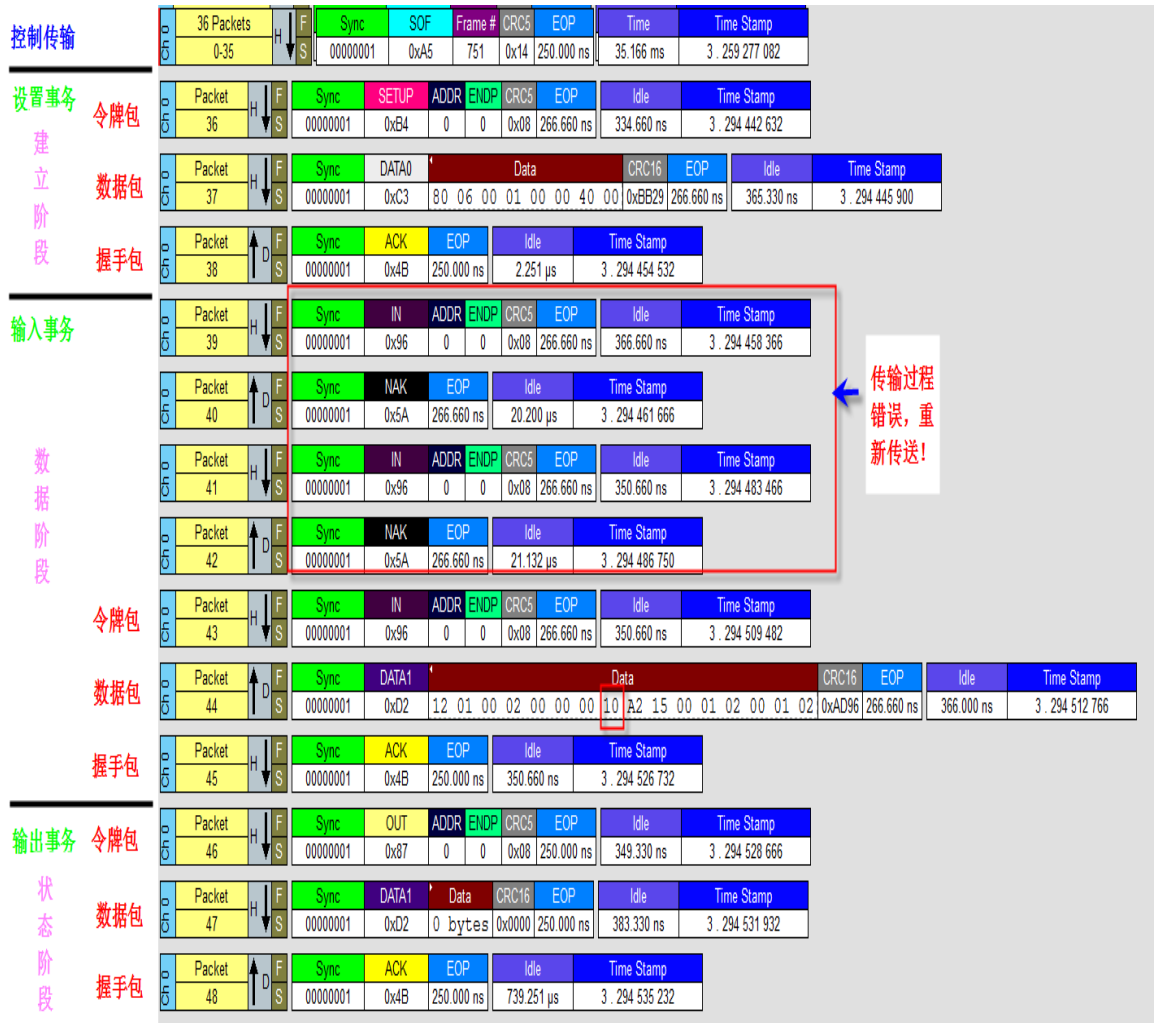


图 4-12 实际抓取 USB 数据情况

上面的数据过程是在设备上电配置完成后，第一次发送获取设备描述符请求的控制传输过程。一共分为三个阶段：建立阶段，数据阶段，状态阶段。这个结构和控制传输过程讲的一致。在每个阶段中，又是由具体的事务处理完成。

建立阶段也是 SETUP 事务处理过程，首先主机给设备发送 SETUP 令牌包，然后主机给设备发送 8 个字节的设备描述符请求包，由数据可以看到数据阶段的方向是设备到主机，即接下来的数据为 IN 包，而请求为 0x06 表示为获取描述符，第 4 个字节为 wValue 的高字节，值为 0x01，表示将获取的是设备描述符，最后设备接收到成功后，会给主机发送一个握手包，表示接受完成。

数据阶段也是 IN 事务处理过程，可以看到在前两次的传输 IN 令牌包后，设备由于某种原因没有能够给主机提供主句，所以返回了 NAK 包，于是主机又重新发送 IN 令牌包，知道第三次发送 IN 令牌包，设备成功返回了数据给主机。这里返回了 16 个字节的数据，大家可能比较异或，为什么设备描述符是 18 个字节，只返回了 16 个字节就不返回了。原因是在初始获取设备描述符的命令时，主机只关心数据的第 8 个字节，即端点 0 所支持的最大数据包长度，这里数据为 0x10，表示端点 0 所支持的最大数据长度为 16 个字节，即每次数据传输都是 16 个字节，如果不够，继续发令牌包获取。而其他的数据主机并不关心，所以并没有再发 IN 令牌包去获取剩余的 2 个字节。当主机成功获取了数据之后，给设备发送了握手包，表示接受数据完成。

状态阶段也是 OUT 事务处理过程，这个时候方向和 IN 包相反，说明传输过程已经到了状态阶段，这时候主机首先发送一个 OUT 包给设备。然后主机再发送数据长度为 0 的数据包给设备，等到设备成功接收到之后，会给主机传回一个握手包，表示接收成功。

以上即为图 4-12 的整个过程描述，这里需要注意的是在 USB 传输的过程中，是遵循小端传输的，即低字节在前，高字节在后。而每个字节中，遵循的是低位在前，高位在后，所以不难看出每个包标示符和 USB2.0 的协议是吻合的。这里以 SETUP 包为例，协议规定的是 PID[3：0]=1101b，而我们抓出来的数据 SETUP=0XB4=bit[0,1,2,3,4,5,6,7]=10110100b，最左边的 1 为最低位 bit0，最右边的 0 为最高位 bit7，高 4 位 bit[7:4]为低四位 bit[3:0]的补码。所以，通过分析可知和实际协议一致。

参考文献：

1. 王宜怀，朱仕浪等.嵌入式技术基础与实践（第 3 版）.清华大学出版社.
2. 刘荣. 圈圈教你玩 USB（第 2 版）.北京航空航天大学出版社.
3. Universal serial bus specification.pdf
4. Device class definition for Human interface devices.pdf
5. USB2.0 协议深入解读.ppt
6. USB2.0 协议中文详解.ppt