

# 目 录

第1章 引言 .....		1
1.1 什么是 USB .....	1	
1.1.1 USB 的孕育 .....	1	
1.1.2 USB 的含义 .....	2	
1.2 USB 的用途 .....	4	
1.3 USB 的布局 .....	5	
1.4 USB 设备 .....	7	
1.4.1 USB 显示器 .....	8	
1.4.2 USB 调制解调器 .....	8	
1.4.3 USB 视频相机和输入设备 .....	9	
1.4.4 USB 键盘、鼠标和游戏杆 .....	9	
1.4.5 USB 集线器 .....	10	
1.4.6 USB 音箱 .....	10	
1.4.7 USB 声卡 .....	11	
1.4.8 USB 扫描仪 .....	11	
1.4.9 USB 打印机 .....	11	
1.4.10 USB 软驱 .....	12	
1.4.11 USB 网卡 .....	12	
1.4.12 USB 转接设备 .....	12	
第2章 计算机总线概论 .....	14	
2.1 总线概念 .....	14	
2.1.1 系统总线 .....	16	
2.1.2 外部总线 .....	16	
2.2 串行总线和并行总线的比较 .....	17	
2.2.1 并行接口 .....	18	
2.2.2 串行接口 .....	18	
2.3 总线标准 .....	19	
2.3.1 PC/XT 总线、ISA(AT)总线及 EISA 总线 .....	20	
2.3.2 PCI 总线 .....	21	
2.3.3 AGP 总线 .....	22	
2.4 流行总线的性能比较 .....	24	
2.4.1 其它几种系统总线 .....	24	

2.4.2 其它几种串行总线.....	25
<b>第3章 USB总线规范 .....</b>	<b>28</b>
3.1 概述.....	28
3.2 应用范围分类.....	29
3.3 USB 的特点 .....	30
3.4 USB 与 IEEE 1394 的比较 .....	32
3.5 有关 USB 的几个重要概念 .....	33
<b>第4章 USB总线体系结构 .....</b>	<b>35</b>
4.1 USB 系统描述 .....	35
4.1.1 总线拓扑结构.....	36
4.2 物理接口.....	37
4.2.1 电气特性.....	37
4.2.2 机械特性.....	37
4.3 电源.....	38
4.3.1 功率分配.....	38
4.3.2 电源管理.....	38
4.4 总线协议.....	38
4.5 稳定性 .....	39
4.5.1 差错检测.....	39
4.5.2 差错控制.....	39
4.6 系统配置.....	39
4.6.1 插入 USB 设备 .....	40
4.6.2 拆除 USB 设备 .....	40
4.6.3 总线枚举.....	40
4.6.4 层间关系.....	40
4.7 数据流类型.....	41
4.7.1 控制信息传输.....	41
4.7.2 批量数据传输.....	41
4.7.3 中断数据传输.....	41
4.7.4 同步传输.....	42
4.7.5 分配 USB 带宽 .....	42
4.8 USB 设备 .....	42
4.8.1 设备特征.....	42
4.8.2 设备描述.....	43
4.9 USB 主机:硬件和软件 .....	44
<b>第5章 USB数据流模型 .....</b>	<b>46</b>
5.1 开发人员观点.....	46

5.2 总线构成	48
5.2.1 USB 主机	48
5.2.2 USB 设备	48
5.2.3 物理总线拓扑结构	49
5.2.4 逻辑总线拓扑结构	50
5.2.5 客户软件	50
5.3 USB 通信流	50
5.3.1 设备端点	52
5.3.2 管道	53
5.4 传输类型	55
5.5 控制传输	56
5.5.1 数据格式	56
5.5.2 分组尺寸限制	56
5.5.3 总线访问限制	57
5.6 同步传输	59
5.6.1 数据格式和方向	60
5.6.2 分组尺寸限制	60
5.6.3 总线访问限制	61
5.6.4 数据顺序	61
5.7 中断传输	61
5.7.1 数据格式	61
5.7.2 方向	62
5.7.3 分组尺寸限制	62
5.7.4 总线访问限制	62
5.7.5 数据顺序	64
5.8 批量传输	64
5.8.1 数据格式	64
5.8.2 方向	64
5.8.3 分组尺寸限制	65
5.8.4 总线访问限制	65
5.8.5 数据顺序	66
5.9 总线传输访问	66
5.9.1 传输管理	67
5.9.2 跟踪处理操作	69
5.9.3 计算总线操作时间	70
5.9.4 计算功能模块/软件中的缓冲区大小	72
5.9.5 回收总线带宽	72
5.10 对同步传输的特殊考虑	73
5.10.1 非 USB 同步应用实例	74
5.10.2 USB 时钟模型	76

5.10.3 时钟同步 .....	76
5.10.4 同步设备 .....	78
5.10.5 数据预缓存 .....	84
5.10.6 SOF 跟踪 .....	85
5.10.7 差错控制 .....	85
5.10.8 为速率匹配而进行缓存操作 .....	86
<b>第 6 章 USB 总线机械规范 .....</b>	<b>88</b>
6.1 机械规范概述 .....	88
6.2 尺寸要求 .....	89
6.3 USB 电缆 .....	89
6.3.1 电缆规范 .....	89
6.3.2 连接器(A 系列) .....	91
6.3.3 连接器(B 系列) .....	96
6.3.4 串行总线图标 .....	96
6.3.5 插头/插座机械和电气要求 .....	100
6.4 电缆压降要求 .....	104
6.5 传播时延 .....	105
6.6 接地技术 .....	105
6.7 信息调整 .....	105
<b>第 7 章 USB 总线电气特性 .....</b>	<b>106</b>
7.1 信号 .....	107
7.1.1 USB 驱动器特性 .....	107
7.1.2 接收器特性 .....	109
7.1.3 信号终端 .....	109
7.1.4 信号电平 .....	109
7.1.5 数据编码/解码 .....	116
7.1.6 比特填充 .....	116
7.1.7 同步方式 .....	118
7.1.8 起始的帧时间间隔和帧调整能力 .....	118
7.1.9 数据信号速率 .....	118
7.1.10 数据信号上升和下降时间 .....	119
7.1.11 数据源信号 .....	119
7.1.12 集线器信号时序 .....	120
7.1.13 接收器数据抖动 .....	122
7.1.14 电缆时延 .....	123
7.1.15 总线转向时间/分组间时延 .....	123
7.1.16 端到端最大信号时延 .....	124
7.2 功率分配 .....	124

7.2.1 设备类型 .....	125
7.2.2 电压下降预算 .....	129
7.2.3 功率控制 .....	129
7.2.4 动态插拔 .....	130
7.3 物理层 .....	131
7.3.1 环境 .....	131
7.3.2 总线定时/电气特性 .....	131
7.3.3 时序波形 .....	135
<b>第8章 协议层.....</b>	<b>137</b>
8.1 比特安排 .....	137
8.2 SYNC 域 .....	137
8.3 分组域格式 .....	138
8.3.1 分组标识域 .....	138
8.3.2 地址域 .....	139
8.3.3 端点域 .....	139
8.3.4 帧标号域 .....	139
8.3.5 数据域 .....	139
8.3.6 循环冗余检验 .....	140
8.4 分组格式 .....	140
8.4.1 令牌分组 .....	141
8.4.2 帧开始分组 .....	141
8.4.3 数据分组 .....	141
8.4.4 握手分组 .....	142
8.4.5 握手响应 .....	142
8.5 处理格式 .....	144
8.5.1 批量处理操作 .....	144
8.5.2 控制传输 .....	145
8.5.3 中断处理操作 .....	147
8.5.4 同步处理操作 .....	148
8.6 数据触发同步和重试 .....	149
8.6.1 通过 SETUP 令牌进行初始化 .....	149
8.6.2 成功的数据处理操作 .....	150
8.6.3 数据被破坏或不能接受 .....	150
8.6.4 破坏了的 ACK 握手分组 .....	151
8.6.5 低速处理操作 .....	151
8.7 差错检测和恢复 .....	153
8.7.1 分组差错分类 .....	153
8.7.2 总线转向时间 .....	153
8.7.3 假 EOP .....	154

8.7.4 串扰和活性损失恢复 .....	155
<b>第9章 USB设备结构 .....</b>	<b>156</b>
9.1 USB设备状态 .....	156
9.1.1 可见的设备状态 .....	157
9.1.2 总线枚举 .....	159
9.2 通用USB设备操作 .....	160
9.2.1 动态连接和拆除 .....	160
9.2.2 地址分配 .....	160
9.2.3 配置 .....	160
9.2.4 数据传输 .....	161
9.2.5 功率管理 .....	161
9.3 USB设备请求 .....	162
9.4 标准设备请求 .....	163
9.5 描述符 .....	169
9.6 标准USB描述符定义 .....	170
9.6.1 设备 .....	170
9.6.2 配置 .....	171
9.6.3 接口 .....	173
9.6.4 端点 .....	174
9.6.5 字符串 .....	175
9.7 设备类型定义 .....	175
9.8 设备通信 .....	176
<b>第10章 USB主机:硬件和软件 .....</b>	<b>181</b>
10.1 USB主设备概述 .....	182
10.1.1 控制机制 .....	185
10.1.2 数据流 .....	185
10.1.3 搜集状态和性能统计信息 .....	185
10.1.4 电气接口考虑 .....	186
10.2 主控制器请求 .....	186
10.2.1 状态控制 .....	186
10.2.2 串行器/解串器 .....	187
10.2.3 帧产生 .....	187
10.2.4 数据处理 .....	187
10.2.5 协议引擎 .....	188
10.2.6 传输差错控制 .....	188
10.3 软件机制概述 .....	188
10.3.1 设备配置 .....	189
10.3.2 资源管理 .....	191

10.3.3 数据传输.....	191
10.3.4 公共数据定义.....	192
10.4 主控制器驱动程序.....	192
10.5 通用串行总线驱动程序.....	193
10.5.1 概述.....	193
10.5.2 USBD 命令机制要求.....	195
10.5.3 USBD 管道机制.....	197
10.5.4 利用 USBD 机制来管理 USB .....	199
10.6 操作系统环境指南.....	201
<b>第 11 章 集线器规范 .....</b>	<b>202</b>
11.1 概述.....	203
11.2 设备特性.....	203
11.2.1 集线器体系结构.....	203
11.2.2 集线器连接.....	204
11.2.3 集线器端口状态.....	205
11.2.4 总线状态鉴定.....	209
11.2.5 全速率和低速率行为比较.....	210
11.2.6 集线器状态操作.....	211
11.3 集线器 I/O 缓冲区要求 .....	213
11.3.1 上拉和下拉电阻.....	214
11.3.2 边沿变化率控制.....	215
11.4 集线器故障恢复机制.....	215
11.4.1 集线器控制器故障恢复.....	215
11.4.2 假 EOP .....	215
11.4.3 中断器故障恢复.....	216
11.4.4 集线器帧定时器.....	216
11.4.5 靠近 EOF 时的集线器动作 .....	217
11.5 挂起和重新开始.....	220
11.5.1 全局挂起和重新开始.....	220
11.5.2 选择性挂起和重新开始.....	223
11.6 USB 集线器复位操作 .....	227
11.6.1 集线器在根端口上接收复位信号.....	227
11.6.2 端口复位.....	228
11.6.3 电源供给和复位时延.....	228
11.7 集线器电源分配要求.....	229
11.8 集线器端点组织.....	230
11.8.1 集线器信息体系结构和操作.....	230
11.8.2 端口变化信息处理.....	231
11.8.3 集线器和端口状态变化位图.....	231

11.9 集线器配置.....	234
11.10 集线器端口电源控制 .....	234
11.11 描述符 .....	234
11.11.1 标准描述符 .....	235
11.11.2 集线器描述符 .....	235
11.12 请求 .....	237
11.12.1 标准请求 .....	237
11.12.2 专用类型请求 .....	237
<b>第 12 章 USB 产品开发和驱动程序设计 .....</b>	<b>246</b>
12.1 Windows 世界中的 USB 设备 .....	246
12.1.1 Windows 95 中的 USB 设备 .....	247
12.1.2 Windows 98 中的 USB 设备 .....	249
12.1.3 Windows 98 环境下的 IEEE 1394 设备 .....	251
12.2 USB 硬件产品开发 .....	252
12.2.1 设计选择.....	254
12.2.2 USB 设备实现举例 .....	258
12.3 USB 设备驱动程序设计 .....	260
12.3.1 Windows USB 驱动程序接口 .....	261
12.3.2 USBDI 的 IOCTL .....	264
12.3.3 USBDI 结构定义 .....	265
12.4 URB 的定义 .....	268
<b>附录 A USB 字汇表.....</b>	<b>270</b>
<b>附录 B 网络资源 .....</b>	<b>279</b>

# 第1章 引言

超星阅览器提醒您：  
使用本复制品  
请尊重相关知识产权！

最初 USB(UNIVERSAL SERIAL BUS)由 COMPAQ、DEC、IBM、INTEL、NEC 和 PHILIPS 等公司联合提出。在当时，市场上已经存在许多种新的外设连接技术，如 IEEE 488 并行总线、RS-232C 串行总线等。然而，这些技术都存在着各自的缺点，如 IEEE 488 总线的带宽较低，RS-232C 总线的连接距离较短等。因此，市场上急需一种新的外设连接技术，以解决这些问题。为此，这些公司联合成立了 USB 工作组，开始研究和制定 USB 技术规范。经过数年的努力，USB 技术终于在 1996 年得到了正式发布。此后，USB 技术得到了广泛的应用，成为了当今最流行的外设连接技术之一。

本章将介绍 USB 总线的产生背景，以及 USB 总线的基本原理和特点。首先，我们将简要回顾一下传统的串行通信技术，包括 RS-232C、IEEE 488 等，并分析它们的优缺点。然后，我们将对比分析 USB 总线与传统串行通信技术的区别，以及它们各自的优缺点。接着，我们将详细介绍 USB 总线的物理层、数据链路层和网络层，以及它们的工作原理。最后，我们将通过一些具体的例子来说明如何使用 USB 总线连接各种外设，以及如何配置 USB 总线的典型配置，从而更容易理解引入 USB 技术的一些基本概念。通过本章的学习，读者将能够对 USB 技术有一个全面而深入的了解。

总线，将串行通信技术推向 21 世纪。该总线应独立于主计算机系统，并在整个计算机系统结构中保持一致。为了实现上述的目标，USB - IF 发布了一种称为通用串行总线的串行技术规范( Universal Serial Bus )，简称为 USB。

## 1.1.1 USB 的孕育

众所周知，对 PC 机进行功能

本章具体包括以下内容：



USB 总线的含义；



USB 总线的用途；



USB 总线的布局；

次发展,今天计算机已经进入了我们的办公室、家庭,成为了一个不可缺少的辅助工具。随着计算机的普及;不了解计算机内部结构的用户日益增多。如何简化外围设备和扩充作业,使之方便易行,这是众多 PC 机厂家面临的重大研究课题。在这背景下 Microsoft 公司于 1994 年提出了 Plug & Play(即插即用)方案,其思想是把 PC 机的外围设备和扩充电路板连接起来,系统可以自动分配中断和端口等资源,而无需用户干预。也就是说在软件方面,增设外围设备和扩充电路板时较以前相对简化了许多。但是在硬件方面,扩充各种扩展卡,例如 PCI 网卡、PCI 声卡等,用户不得不拆开机箱,才能把扩展卡插到主板上的相应扩展槽内,这仍然相当麻烦。

因此,在 1996 年召开的面向 PC 机硬件技术工作者会议上 Compaq、Intel 和 Microsoft 三家厂商提出了设备插架(Device Bay)概念,并于 1997 年 6 月正式公布设备插架标准规格 0.8 版,经过一段时间对 0.8 版的大幅度修正,终于设备插架规格于 1997 年第 3 季度正式确定下来。

设备插架的规格分为台式 PC 机规格和笔记本式 PC 机规格,例如:台式机的设备插架外形尺寸为  $159.4\text{mm} \times 177.8\text{mm} \times 1.9\text{mm}$ ,而笔记本式的设备插架占用面积为  $(130 \times 127)\text{mm}^2$ ,高度为 19mm 或 12.7mm;对于馈电电源的规定如下:台式机设备插架里有 +12V、+5V 和 +3.3V 供电连接器,笔记本机设备插架里只有 +5V 和 +3.3V 供电连接器。同时,设备插架规定了扩充插槽的机械规范和电气规范。事先把要扩充的电路板和外围设备放在规定的外壳内,执行扩充作业时将它插入到设备插架里便可。

这种设备插架有以下 3 大特点:

(1) 设备插架插拔方式与 VTR(磁带录像机)装入盒式录像带方式十分相似,把设备插架盒套对准 PC 机箱上的设备插架入口轻轻一推即可,十分方便。

(2) 利用设备插架实现 PC 机功能扩充时,PC 机并不停电,原来的应用程序照样运行,即后面所提到的动态插拔(热插拔)技术。

(3) 利用设备插架实现 PC 机功能扩充远比 PCI 总线扩充性高。主板内的 PCI 扩展槽只有那么寥寥几个,即使再加上一两个 ISA 扩展槽,也很难适应未来发展的需要。而使用设备插架技术最多可扩充 63 个外围设备。

在设备插架规格里,可以利用串行接口 IEEE 1394(另外一种串行总线)和 USB 连接外围设备。设备插架的连接器分别备有“IEEE 1394”和“USB”(两者的详细资料请见第 2 章)的连接端。究竟利用那种串行接口,由外围设备的使用情况而定。对于数据传送速度为 12Mb/s 以下的低速外围设备,应使用 USB, IEEE 1394 则适用于数据传送速度为 100Mb/s 以上的高速外围设备。

### 1.1.2 USB 的含义

近年来 USB 技术已经成为了计算机领域发展最快的技术之一,并为越来越多的个人计算机界人士所接受。随着 Windows 98 的问世,USB 系统的支持问题得到了解决,各种 USB 设备不断涌现。可以说正是由于 Windows 98 的出现,才带来了 USB 技术的飞速发展和普及。现在,你如果买了一台没有 USB 端口的基于 Intel 生产的芯片的 PC 机,那简直是不可能的。而且 USB 技术并不仅限于 PC 行业,现在每一个计算机硬件的生产商都在尝试在他自己的平台上应用 USB 技术。那么什么是 USB 呢?

读者可能对串口有所了解,我们通常所用的鼠标和调制解调器都是连接在串口上的。但 USB 并不完全是一个串口,它实际上是一种串行总线。这意味着你的机箱后盖上的 USB 端口可以连接许多设备,这些设备可以相互连接在一起。而且不同类型的设备组成可以通过一种称为 USB 集线器的硬件分离开来,这些都是与传统的串口上只能链接一个设备有着本质区别的。正如前面提到的一样,USB 用来把串口、并口等不同的接口统一起来,使用一个 4 针插头作为标准插头。通过这个标准插头,采用菊花链形式(星型结构)可以把所有的外设连接起来,并且不会损失带宽。也就是说,USB 将取代当前 PC 上的串口和并口。所以当我们提到 USB 时,与其将它想象成一个串口,还不如将它想象成一个连接有不同设备的网络,就像我们所熟悉的以太网一样。图 1-1 给出了一个典型的 USB 设备网络配置的示意图。

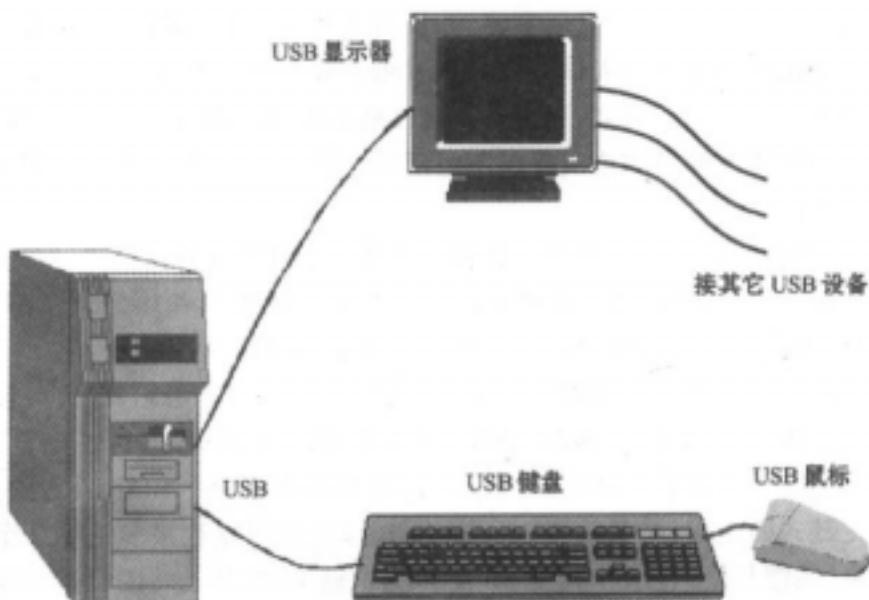


图 1-1 典型的 USB 配置图

但是要想在同一条总线上连接不同的设备并不容易实现,因为这意味着会有许多设备来共享总线上有限的带宽。对于我们所熟悉的 RS - 232 串口通信的标准来说,它的带宽就非常有限,不能用其来与打印机相连。当然,我们也就更不可能利用它来从数字相机上下载图片了。一条 RS - 232 串口通信电缆只能连接一个物理设备,而 USB 上却可以连接最多 127 个外设。所有这些外设都有可能和主机进行通信,USB 不仅要处理好总线竞争问题,还要保证各个设备的正常数据通信要求。因此,相对于 RS - 232 而言,USB 总线的实现机制要复杂得多。

问题的关键在于必须要有一条速度很快的总线。USB 在计算机工业中被认为是一个具有中低速率的总线,它每秒钟可以处理 10Mbit 的信息。而这一速度正是大多数商用计算机网络的速度。但是在与工作速度为 300Mb/s 的“光通道”串行总线和即将问世的专门用于处理音频和视频信号的 IEEE1394 等总线技术而言,USB 的速度就不能称为快了。所以,我们说 USB 是一种中低速的总线,这一点读者要明确。

## 1.2 USB 的用途

如上所述,USB 就是设备插架的一种规范。USB 技术的设计目的是向广大计算机用户提供纯粹的数字视频和音频信号,以实现通信服务。所以 USB 的总线速度要足够用来支持这些类型的设备。

对广大 PC 机用户来说,在连接外设时有一个很大的问题,那就是对于每一种设备而言,都需要在总线上插入一块它自己的适配器,当然还需要占用主板上的一个扩展槽。例如,对于高分辨率的图像支持需要有显卡;为了驱动你的游艺杆需要有游戏卡;驱动音箱还要有声卡;为了将图像信号输入计算机,还要有视频输入卡等等。

随着计算机技术的飞速发展,新的外设不断涌现,留给各种适配卡使用的扩展槽越来越少了。同时,为了适应未来发展的需要,计算机还要包含足够的技术和电源功率,从而可以为将来可能出现的数字输入设备和带宽敏感的外设提供支持。例如在 PC 机上实现电视会议在今天已经成为了现实,而由你的 PC 机来提供环绕立体声将成为一种标准的功能。当然究竟要实现哪些功能现在还不能预计。因此 USB 技术必须着眼于未来,具有良好的可扩充性。

所以,人们正在努力发展一种可以涵盖中低速数字外围设备的需要,同时所占资源又更少的技术。而 USB 正好满足了这些要求。它把所有的输入和输出外设都置于机箱之外,而不使用任何扩展槽。它使设备具有了一些智能,虽然它不是计算机。

在 USB 方式下,所有的外设都在机箱外连接,不必再打开机箱;允许外设热插拔,而不必关闭主机电源。USB 采用“级联”方式,即每个 USB 设备用一个 USB 插头连接到一个外设的 USB 插座上,而其本身又提供一个 USB 插座供下一个 USB 外设连接用。通过这种类似菊花链式的连接(星型结构),一个 USB 控制器可以连接多达 127 个外设,而每个外设间距离(线缆长度)可达 5m。USB 能智能识别 USB 链上外围设备的插入或拆卸,USB 为 PC 的外设扩充提供了一个很好的解决方案。

主机和 USB 设备之间的连接拓扑结构是星形连接。USB 连接器分 A 系列和 B 系列,一般 USB 设备利用 B 系列连接器与主机连接,而键盘、鼠标和扩充集线器等 USB 设备则利用 A 系列连接器与主机实现连接。主机与要求全速传送的 USB 设备连接时,可利用 HUB 级联方法延长连接距离,但最多允许 5 个 HUB 级联,最长扩展连接距离不得超过 30m。

对一般外设而言,USB 有足够的带宽和连接距离来支持它。USB 允许两种数据传送速度。低速传送为 1.5Mb/s,全速传送为 12Mb/s。全速传送时,结点间连接距离为 5m,连接使用 4 芯电缆(电源 2 条,信号线 2 条)。该速率与一个标准的串行端口相比,大约快出 100 倍,与一个标准的并行端口相比,也快出近 10 倍。因此,USB 能支持高速接口(例如 ISDN、PRI、T1),使用户拥有足够的带宽供新的数字外设使用。

有了 USB 技术,外设的设计者就可以很自由地实现其方案——而无需将整个外设的功能分为设备和接口卡两个部分。现在出现的 USB 音箱,就可以认为是一个 USB 声卡和传统音箱的结合体。同时计算机中的内部总线也就不必处理在这些接口卡之间穿梭的信息流了。你会发现,有了这种类型的配置,你可以获得更好的整体系统性能。

现在扬声器的设计者利用 USB 技术实现以前直接和扬声器相连的声卡所实现的功能。从事视频输入的人们正在设计可以插入 USB 总线的视频编码器。甚至显示器的生产厂商也在它们的显示器后放上了 USB 总线接口,从而舍弃了过时的显卡。还有支持高分辨率的数字游戏杆。另外,USB ZIP 的出现大大方便了用户存储数据的要求。更有甚者,还出现了 USB 的调频收音机。可以说,USB 技术的问世改变了传统的 PC 机外设世界。使不同外设与主机之间的接口技术大大简化了。

### 1.3 USB 的布局

由于 USB 总线上连接的外设可以多达 127 个,所以如果没有一个好的配置方案,你的书桌也许会一团糟。如何安排你的 USB 设备,使其看起来更像设置一个计算机网络,毕竟它们之间有许多共同点。

当我们安装一个计算机网络时,只要我们安装了正确的软件并将电缆接好之后,它就可以工作了。而我们配置 USB 设备时,则要比安装一个计算机网络简单。因为我们无须安装许多的软件、协议来支持 USB 设备(当然你必须安装一些驱动程序)。USB 外设的安装十分简单,所有的 USB 外设利用“ONE - SIZE - FITS - ALL”(统一规格)连接器都可简单方便地连接计算机中,安装过程高度自动化,既不必打开机箱插入插卡,又不必考虑资源分配(即支持即插即用功能),也不用关掉计算机电源(热插拔)。

一个 USB 总线的基本组成是:一个主机(host)和若干台从设备。这些从设备都连接在 USB 总线上,它们可以是音箱、显示器,还可以是集线器、鼠标、扫描仪等等。

目前,我们可以在许多机箱后盖上发现除了通常的串口、并口还有 PS/2 端口外,还有两个长方形口,这就是两个 USB 端口。它们在机箱后盖上的布局参见图 1-2。

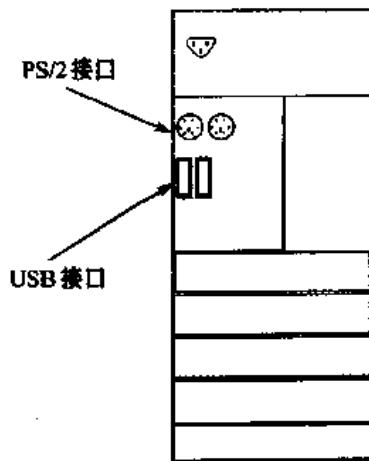


图 1-2 USB 接口

如果我们只有一两个 USB 设备时,那么利用 USB 电缆可以很容易地把 USB 设备与我们的主机相连。但是,当我们的 USB 设备很多时(下一节我们可以看到目前已经有许多种类的 USB 设备),这两个 USB 端口显然是不够用了。最简单的办法是在机箱上多设置几个 USB 端口,使得一个 USB 端口与一个 USB 设备相对应。但是 USB 总线上可以支

持多达 127 个 USB 设备,很难想象在机箱上安装 127 个 USB 端口。这时,就要利用前面提到的集线器(hub)来解决这一问题。USB 集线器与传统以太网的集线器的功能十分相似,难怪我们要将 USB 总线与计算机网相比较了。但两者还是有区别的,在以太网中集线器是一台独立的硬件设备(图 1-3),用来实现网络中的服务器和各个终端设备的互连;而 USB 集线器既可以是一台独立的硬件设备,又可与其它 USB 设备集成在一起,这一类设备我们称之为复合功能设备(多功能设备)。例如,许多 USB 键盘上都集成了一个单端口的集线器,以便连接 USB 鼠标或 USB 游戏杆,参见图 1-4。

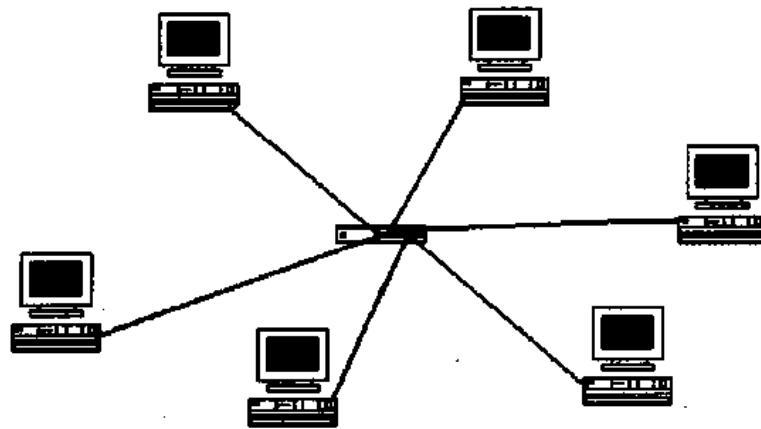


图 1-3 以太网集线器

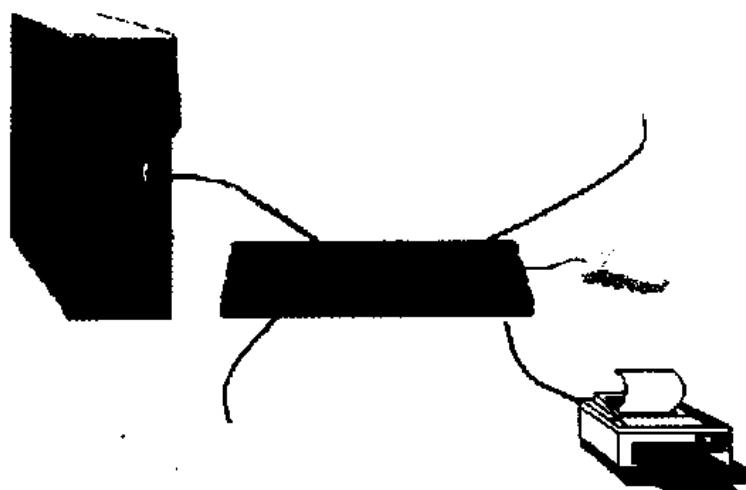


图 1-4 USB 集线器(在键盘上)

目前,几乎所有 USB 显示器上都有两个具有 4 个端口和 8 个端口的集线器。这样一来,USB 显示器就成了所有 USB 设备的核心,机箱上也不用接许多很长的电缆了。图 1-5 说明了这种类型的配置。

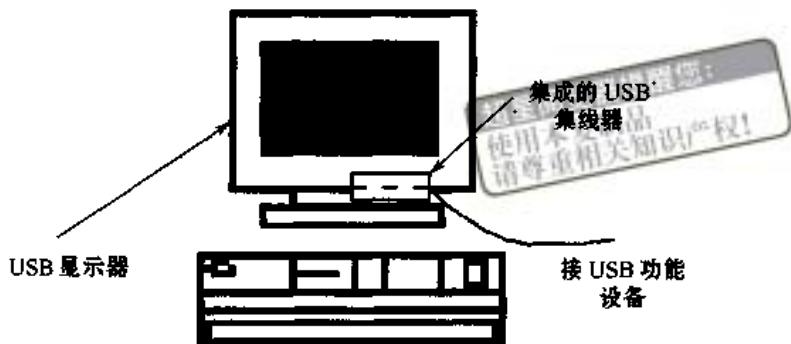


图 1-5 USB 集线器(在显示器上)

还有一类特殊的集线器,那就是在主机上提供了两个 USB 接口的集线器,我们称之为根集线器(Root Hub)。无论是集成在键盘或显示器上的集线器,还是独立的集线器,或者是集成了其它功能模块的集线器,都带有若干个 USB 接口。其中一个用于连接通往主机方向的上行 USB 电缆,剩下的接口就可以用来连接别的 USB 功能设备或是另外的集线器,从而构成一个星型拓扑结构。

在 USB 系统上,一个 USB 设备可以插在任何一个端口上。一个设备接在一台 4 端口的集线器上时的性能,和它直接接到机箱上的根集线器时性能不会有任何差别。问题的关键在于根据你自己的系统环境进行恰当的配置,这一点也体现了 USB 总线技术布线的灵活性。例如,我们拥有了 USB 键盘、游戏杆、鼠标和音箱,那么我们可以把它们都与 USB 显示器上集成的集线器相连;当然,我们也可将其中的某个 USB 设备直接与机箱上的另外一个 USB 端口相连。正是由于这种灵活性,使得对于初学者而言,将 USB 设备与主机相连也是一件很容易的事情,因为不存在连结对错的问题。

PC 机的控制芯片组和操作系统中已经内建了 USB 功能,因此 PC 机在硬件和软件的价格上不会增加。而由于减少了接口插卡和供电电源,使得外设的成本可以降低。此外 USB 的“热插拔”功能允许用户十分方便地连接和拆卸外设,实现外设的共享。这也就降低了这些外设的使用成本。

## 1.4 USB 设备

虽然早在 1997 年,Intel 440 LX 芯片组就提供了 USB 功能,但由于当时支持 USB 的设备极少,这一技术并没有引起用户过多的关注。时至今天,支持 USB 的设备数量已经相当可观,USB 接口的种种优势正在显示出来,在 PC99 规范中,USB 已经是个人电脑的标准设备之一。USB 技术正在不断地发展和完善,因而支持 USB 技术的外设也在不断地涌现。如 USB 键盘、USB 鼠标、USB 调制解调器等。可以预见,以后的主板上将没有 PS/2、COM 等规格不一的烦人的外设接口,取而代之的是数个 USB 接口,所有的外设都通过这一接口连接。本节就来针对这些 USB 设备,作一个简单的介绍,并给出了一些目前市场出现的 USB 设备的概述,希望可以给读者一定的感性认识。

### 1.4.1 USB 显示器

通常 USB 显示器都集成了一个 USB 集线器,这样显示器就可以用来连接其它的 USB 外设了。先前在机箱后面那些混乱的连线也可以消失了,取而代之的仅是一根 USB 电缆。利用 USB 在 USB 显示器和计算机之间实现数字连接,可以使用户在计算机上调整显示器的设置,而不用通过显示器上的控制按钮来调整。

但是,对于 PC 机的视频信号的传输要求来说,USB 规范 1.0 所支持的带宽就显得十分紧张了。所以,目前仍然需要在主机中插入一块显卡来支持视频输出。因此,现在的 USB 显示器并不像 USB 音箱那样真正通过 USB 总线传输数字信号,它还是要使用显卡,所以它只能算作“带 USB 功能的显示器”。

LG795FT Plus 就是这样一款显示器,通过 USB 接口,用户可以使用软件方式调整一些显示器属性,包括屏幕大小、屏幕位置、失真、色彩等,而不必使用 OSD 菜单。使用时,只要选中显示属性的 USB 显示器项,就可以用鼠标拖动滑动条,对大多数原本需要通过 OSD 菜单调整的功能进行设置,非常简单。更为重要的是它还内建了一台 4 口的 USB Hub(集线器),可以减小用户的费用。

### 1.4.2 USB 调制解调器

简单地说,Modem 的作用就是将电脑的数字信号调制为能在电话线上传输的模拟信号,USB Modem 也是如此,但它的结构要比普通外置式 Modem 简单得多。

目前最普遍的 Modem 是通过 RS - 232 口接入计算机的。由于 RS - 232 技术的限制,最大的带宽将限制为 119Kb/s。虽然 119Kb/s 对于目前的拨号 Modem 来说已经足够快了,甚至对于 ISDN 来说也可以运行得很好,但是如果要适应未来通信发展的需要,就不是那么乐观了。

在电信领域,现在人们正在研究快速接入技术,以实现通过有线电视网、DSL(直接用户环路)等接入技术来传输 Internet 和其它通信信息。而在计算机领域,人们已经可以实现 T1(北美,1.554Mb/s)和 E1(欧洲,2.048Mb/s)这样的传输速率了。对于这些传输速率的应用而言,利用 USB 技术就可以直接将你的计算机与 T1、T3 或 E1 链路相连了。因而,USB 技术在这一领域具有很多优势。

例如在市场上可以见到联宝的“小黑猫”56K USB Modem,从外型上看非常小巧,只有一只鼠标大小,和常见的外置式 56K Modem 那庞大的体积相比完全不同。“小黑猫”不需要外接电源,Modem 的两端分别有一个 USB 接口和电话线接口。USB Modem 的表现还不错,连接速度一直稳定在 52000b/s,而且很少出现掉线的情况。但是它的 CPU 占用率要比普通 56K Modem 高一些,特别是在打开网页的时候,CPU 占用率约在 50% 左右。这款 USB Modem 没有 Photo 接口,不能接电话,也就是说如果用户不是使用一条专门的电话线上网,就要将电话线头在电话和 Modem 之间换来换去,没有普通 Modem 方便。

因为 USB Modem 的价格和普通 Modem 相比要便宜,性价比较高,应该是目前意义比较大的 USB 外设之一。

### 1.4.3 USB 视频相机和输入设备

通常我们要把图像存储到计算机内,就需要利用一块视频捕捉卡,由它来完成从模拟图像信号到数字图像信号的转换,并把 A/D 变换后的数字信号存储到计算机内。而且,以前我们所见到的这类视频捕捉卡大多都是 PCI 总线或 ISA 总线的。要安装这些视频卡都需要占用主板上的 PCI 或 ISA 扩展槽,使用起来很不方便。

另外一方面,由于总线的带宽一定,如果同时有多个设备申请总线的访问权,总线控制器就要在不同的设备之间作出仲裁,决定由哪一个设备可以访问总线。例如,如果我们使用的是一个 PCI 视频捕捉卡,而磁盘控制器也是 PCI 总线 IDE 控制器。那么在输入图像信号时,视频捕捉卡就要和 IDE 控制器争夺总线的访问权,这时就出现了总线竞争,从而影响了该卡的性能发挥。

但是,如果我们使用的是 USB 视频相机,就可以减小这些可能出现的麻烦。因为 USB 为同步信号流提供了保留的带宽,从而可以保证这些时延敏感的信号可以在确定的时延要求下,通过 USB 总线(这一点,我们将在以后给予详细说明)。再者,使用数字相机,由于它生成的信号本身就是数字信号,因此就可以减小由模拟信号变成数字信号所带来的信号失真,进一步提高设备性能。

而目前,摄像头的成像质量还不能同数码相机相提并论,最主要的是,图像传输速度较慢,这也是摄像头迟迟不能得到广泛应用的原因。早期的摄像头是通过并行口或串行口与计算机相连,数据传输速度极慢,最糟糕的是它还得从键盘或鼠标口获取电源(+5V)。

摄像头厂商都注意到了这个致命的缺点,纷纷投入研发力量改进自己的产品。不过,在 USB 接口成为流行总线以前,他们取得的成绩并不大,“不合时宜的定格”还是在几乎所有的摄像头产品中存在。1998 年以后,Creative、Logitech、NEC 等厂商推出的摄像头产品都是 USB 接口,同传统的其它接口摄像头相比,USB 接口的摄像头优势明显:支持热插拔,传输数据的速度更快,因为传输速率的提高,图像的分辨率也可以做得更高一些,图像质量的提高也成为可能。

### 1.4.4 USB 键盘、鼠标和游戏杆

键盘和鼠标算得上是从 USB 得益最少的设备,目前也没有什么必要使用 USB 接口,因为现在的主板都提供了键盘和鼠标专用的 PS/2 口。不过 USB 终会取代 PS/2 口,抢先感受一下新产品也不错。

微软的新款人体工程学键盘就采用了 USB 接口,兼容 PC 和 iMac,无论外型、手感都无可挑剔,不过在 DOS 系统下就不能使用了。由于 USB 接口的键盘没有什么出奇的优势,所以也有公司为它增加了一点功能,Cherry 制造的 USB 键盘内建 4 个 USB 接口的 Hub,可以连接 4 个低功率的 USB 设备,如鼠标、飞行摇杆等。

罗技应该算是鼠标厂商中的旗首了,它推出的使用 USB 接口的两款鼠标是银貂和旋貂。这两款鼠标都提供了一个 USB 转 PS/2 的转接头,主板不支持 USB 的用户也可以使用。USB 鼠标和 PS/2 鼠标并没有太大的区别,热插拔的机会也比较少,但罗技银貂和旋貂同时提供 USB 和 PS/2 口,比单一接口有更多选择。

相对于鼠标和键盘而言,游戏杆可以说是从 USB 得益最多的设备了。与传统的模拟游戏杆相比,USB 游戏杆具有惊人的速度和可控性。目前在游戏玩家中,新出现的一种强力反馈型游戏杆已经越来越受欢迎。这种强力反馈型游戏杆可以向游戏杆返回振动和其它一些反馈信息。例如,你在玩 F - 16 这样的空战游戏时,当你正在驾驶飞机飞行时,你就会感觉到来自于游戏杆的飞机的动作和阻力。这样一来,就更增加了游戏的真实性,可以给你带来更大的乐趣。

#### 1.4.5 USB 集线器

在以后我们会看到,USB 系统中的核心元件是 USB 集线器(USB Hub)。USB 集线器用于提供更多的接入 USB 系统的端口。与以太网中的集线器很相似,它通过一条上行线与主机中的根集线器相连(在后面将讲到,USB 根集线器就是集成在主机上的集线器),并具有若干个下行端口供其它 USB 设备与之相连。这样一来,虽然一个主机上的根集线器的 USB 接口通常只有两个,但通过 USB 集线器,主机就可以连接多达 127 个的 USB 设备。

还有一点值得注意的是,USB 集线器本身还可以具有额外的功能模块。例如,一个 USB 集线器上可以集成一个调制解调器(Modem)。这样一个 USB 集线器不仅可以完成上面所提到的基本功能,还可以用于拨号上网。像这样在一个物理设备之内集成了多个功能模块(逻辑上是多个 USB 设备)的设备,我们称之为复合功能设备或多功能设备。除了可以集成一个 Modem 之外,集线器还可以集成声卡、扬声器、网卡等等,从而使集线器成为一个 USB 系统中的核心设备。

#### 1.4.6 USB 音箱

PC 的音频接口目前还是以模拟接口为主,不久模拟将让位于 USB 通用数据总线接口,USB 音箱取代模拟音箱和声卡将是发展趋势。但目前来说,是否使用声卡还有赖于具体的应用。对于专业音频信号处理人士来说,USB 音箱可能还不能满足他们的需要,这时仍然需要一块声卡。但对于大多数普通用户而言,使用一个 USB 音箱不仅可以为自己节省一块声卡,还可以欣赏到一定质量的声音效果,还是很值得考虑的。因为,许多人并不要求能够欣赏高保真的音乐。

一般来说,USB 音箱都嵌入一个自供电方式的功放,这也是一个相当好的功能。现在具有 USB 音箱生产能力的厂家众多,较为有名的有 J - S 淳誉电子的“爵士”音箱、漫步者、麦蓝。而目前市场上较为流行的有 J - S 的 J6502 和漫步者的 USB1900T/USB1000TC。

漫步者的 R1900/R1000 是典型的木质音箱,提供两路模拟信号输入接口和一个 USB 数字信号输入接口。除多了一个 USB 接口外,其它方面没有任何改变。在音箱后座多了一块 USB 控制电路板,从形式上来看,就是把一块 USB 声卡从外部移到了音箱内部。

淳誉电子的 J6502 也是采用 USB 接口的塑料音箱,带有模拟信号输入接口。从声音表现来看,J6502 的中高音细致逼真,低音效果尚可,但其震撼力度不够,不过,对于放在电脑桌上的多媒体音箱来说,其音质表现完全能够满足用户要求。

### 1.4.7 USB 声卡

现在流行的 A3D 和 EAX 环境音效技术及 64 位 DLS 波表让用户非常满意, 而厂商针对声卡发挥想象的空间却越来越小了。不过, USB 的流行却给这些厂商带来新的商机。USB 声卡的出现就是这些厂商的杰作。

USB 声卡可满足那些已经买了传统的模拟音箱但又想体味数字音频技术的用户的味口。由于采用 USB 接口, USB 声卡就成为外置式设备, 这样就可以节省一个主板上的扩展槽。USB 声卡的发声原理很简单: 从 USB 接口获取数字音频, 经过 USB 声卡的 D/A

Canon 公司在新世纪来临之际,推出了一款 BJC - 3000 彩色喷墨打印机。在 USB 接口的设备已经非常流行的今天,佳能打印机也不甘落后。BJC - 3000 打印机除具有并口的连接方式外,还具有 USB 接口的连接方式,给用户以自由选择的空间。使用 USB 接口安装该打印机非常简单,插上 USB 线后,系统提示找到 USB 设备,按照提示插入 BJC - 3000 的驱动光盘后,安装该打印机的驱动程序。然后,在 Windows 98 的“控制面板”中的“系统”选项内,点击“设备管理”菜单项,就可以在“通用串行总线控制器”下看到 USB 打印机了。USB 接口安装比并口方便,而且可以热插拔,当几台没有联网的机器分享一台打印机时,USB 接口的优势显而易见。

由于打印机的速度瓶颈并不在电脑向打印机传输数据部分,所以使用 USB 接口给它带来的速度提升并不大。它的 USB 接口应该说是锦上添花的一项功能。

#### 1.4.10 USB 软驱

3.5 英寸 USB 软驱是国内较少见的外设之一,实用性也并不强,毕竟绝大多数电脑都带有一个内置的软驱。VSTTech 的 3.5 英寸软盘驱动器最初是为 iMac 设计的,但它完全兼容 PC 机的 3.5 英寸盘,只要你的 PC 支持 USB,就可以将它当作普通软驱一样使用。这款软驱使用了和 iMac 一样的半透明外型,非常漂亮,并且传输率提高到了 500KB/s。

ZIP 驱动器在国内比较少见,但在欧美比较普及,它使用专门的 100MB 或 250MB ZIP 磁盘,容量、速度和可靠性都与 3.5 英寸软驱不可同日而语。内置 ZIP 驱动器使用 IDE 接口,显然比 USB 接口更有优势,但它要求机箱中有一个 3.5 英寸驱动器的空位,而多数品牌机并不具备这一条件,所以 ZIP 驱动器以外置式为主,接口包括并行口、Ultra SCSI 和 USB。持续数据传输率分别为 0.8Mb/s、2.4Mb/s 和 1.2Mb/s,我们可以看到 SCSI 接口的驱动器速度最快,但一块 SCSI 接口卡价值不低,从性价比来说,USB 接口的 ZIP 驱动器显然是最好的选择。新款 Iomega ZIP USB 驱动器的平均寻道时间为 29ms,容量 250MB,体积很小,只有手掌那么大,移动起来十分方便,对于那些需要从不同的计算机上拷贝大量数据的应用作用极大。

#### 1.4.11 USB 网卡

USB 网卡的功能还比不上我们常用的 PCI 网卡,价格也不便宜,不过它也是 USB 家族的一员。USB 网卡目前只能提供 10Mb/s 的局域网连接,而较大型的局域网现在都使用 100Mb/s 连接,所以它比较适用于家庭或小型办公室。目前 USB 网卡的价格都很高,甚至比 100Mb PCI 网卡更贵,对国内普通用户来说没有太大的实用价值,但对带有 USB 接口的笔记本电脑用户来说则很有用处。另外,还有一种 USB to USB 对联器,它可以让两台计算机通过 USB 接口互相通信,比串口对联提供更高的连接速度,和 USB 网卡比起来,它可能是一种更为实用的设备。

#### 1.4.12 USB 转接设备

USB 转接设备的种类很多,主要是提供 USB 接口与其它接口的转换功能,包括 USB to ADB、USB to PS/2、USB to SCSI、USB to Serial、USB type A to B 等等。这些转接设备

可以使其它非 USB 接口的外设接到 USB 口上使用,这些设备在今天看来还没有太多用处,但有朝一日主板上只提供 USB 接口时,它们就有了用武之地了。现在比较实用的 USB 转接设备主要有两种,其中之一是 USB to PCI 接口卡,它的功能是为不支持 USB 但支持 PCI 的主板(如 Intel 430TX 等)提供两个 USB 接口;或者为只支持两个 USB 接口的主板(如 Intel 440BX 等)提供额外的两个 USB 接口,不过要占用一个 PCI 插槽。Keyspan 的产品采用 OPTi 的芯片,CableMAX 的产品采用 VIA 的芯片,它们都支持 USB 高速和低速连接,无跳线设计,自动设置 IRQ,用户只需要将卡插到 PCI 插槽并安装驱动软件就可以了。另一种实用的 USB 转接设备是 USB 设备延长电缆。普通 USB 设备的联线距离被限制在 1.8m 之内,超出这一距离,USB 设备可能不能正常工作,而通过 USB 设备延长电缆可以把这一距离延长到 5m。这种电缆内置有 ASIC 芯片,用来缓冲输入和输出的 USB 数字信号,它就像一个单一端口的 USB Hub 一样。这种电缆还可以集联工作,最多支持 5 条电缆串联,将 USB 设备和计算机之间的距离延长到 25m。

除了上述这些设备之外,甚至还出现了 USB FM Radio。而且它的价格也不是很贵,用户使用它可以很方便地在电脑上收听广播。所以说,USB 技术是方兴未艾,新的 USB 设备还在不断涌现,前景喜人。

## 第2章 计算机总线概论



本章将介绍一些有关计算机总线的内容，部分是各个模块之间实现通信的桥梁。目前章的介绍，可以使我们对总线有一个比较全面

总线对于任何计算机系统来说，却很不深刻，甚至会有一些误解。总线、SCSI 总线、ISA 总线、MCA(多通道连接)总线等。面对如此多的名词，会使人觉得总线很神秘。阅读后面的章节时有一个清晰的印象：首先，什么是总线呢？就其

本章具体包括以下内容：



总线的概念；



总线的分类；



系统总线；



外部总线；



串行总线与并行总线的比较；

利用总线,一个设备就可以完成与另外一个设备或多个设备之间的通信。总线技术包括通道控制、仲裁方法和传输方式等内容。

一般来说,计算机总线分为系统总线、外部总线、内部总线三部分。另外外部总线也称做 I/O 总线。为了和系统总线加以区分,也把内部总线和外部总线叫做局部总线。

(1) 片内总线:它位于微处理器芯片内部,用于 ALU(算术逻辑单元)及各种寄存器等功能单元之间进行互连。

(2) 存储总线:具有 32 位地址线(MAB)和 36 位数据线(MDB,包括 4 位奇偶校验位),用来连接存储控制器和 DRAM。

(3) 片总线(又称元件级总线,内部总线):它是一台单板机或一块 CPU 插件板使用的板上总线,用于芯片一级的连接。它是微型机系统的重要总线,在将接口芯片与 CPU 连接时就要与这种总线打交道。它一般是 CPU 引脚的延伸,与 CPU 的关系密切,负责和 CPU 的通信。但板内总线较多时,往往需要增加锁存、驱动等电路,以提高驱动能力。

(4) 系统总线:也称 I/O 通道总线,微型计算机总线或板级总线,用来与 PC 机系统,充槽上的各扩充板卡相连,它是微型机系统的最重要的一种总线。一般谈到微型机总线,指的就是这一种总线。有的系统总线是片总线经过重新驱动和扩展而成,其性能与某种 CPU 性能有关。但有不少系统总线并不依赖于某种型号的 CPU,可为多种型号 CPU 及配套芯片所使用。在用各种插件板组成或扩充微型机系统时,就要选用恰当的系统总线,并按总线的规定来操纵这些插件板。系统总线有多种标准(如 PCI、ISA、EISA 等),其数据地址线不同,以适用于不同的应用系统。

(5) 外部总线(又称通信总线):它用于在系统之间进行互连,如微型机之间,微型机与仪器或其它设备之间。常用的外总线有 RS - 232C、IEEE - 488、VXI 等。我们所使用的 USB 总线,也属于这一类型。

在以上几种总线中,片总线、存储总线、外部总线在系统板上,不同的计算机系统采用的芯片组不同。所以这些总线均不完全相同,也没有互换性问题。

而系统总线则不同,它是与 I/O 扩展插槽相连接的。I/O 插槽中可以插入各种扩充板卡,作为各种外设的适配器与外设连接。因此要求系统必须有统一的标准,以便按照这些标准来设计各类适配卡。

USB 总线是为了解决一些相对而言速度较慢的设备的 I/O 操作的需要;而 PCI 和 ISA 总线则用于提高系统的性能;SCSI 和 IDE 总线则专门用于解决磁盘和主机之间的 I/O 操作;VESA 则用于提高系统的视频性能。总之不同的总线技术都是为了解决某一方面问题而产生的。

每一种总线都不仅包含一些用于数据传输的数据线,还包括一些保证数据传输正确进行的信号线。这些信号线有的用于防止数据丢失,有的用于保证在收发数据时设备已经作好了准备。总之,为了实现不同设备之间的通信,总线技术的规范必须以机械、电气特性和通信协议等方面在插件板级之间建立一个标准,总线上的所有的动作都必须以这个标准为基础,这样才能维持正常的通信。

由于系统总线的重要地位,所以在此我们对系统总线和外部总线展开了比较详细的讨论,一方面使读者可以加深理解,另外一方面也可以为以后几节中 USB 与各类总线的比较提供参考。

### 2.1.1 系统总线

系统总线的信号线大致可分为 5 类：

(1) 数据传输信号线,包括地址线、数据线以及读写控制线等。数据线用于传输数据和代码,一般为双向信号线,可以进行两个方向的数据传输,采用三态逻辑(所谓三态是指除可输出高电平和低电平外,还可以处于高阻状态)。地址线用来传送地址信号,而且均为单向总线(所谓单向是其信息只能向一个方向传送),也采用了三态逻辑。控制线则是用来传送控制信号的总线,用来实现命令、状态传送、中断、DMA(直接存储器存取)的控制,根据不同的使用条件,控制总线有的为单向,有的为双向,即可以采用三态,也可以采用非三态。

- (2) 中断信号线,包括中断请求线、中断认可线等。
- (3) 总线仲裁信号线,包括总线请求线、总线认可线等。
- (4) 其它信号线,包括系统时钟、复位、电源和地线等。
- (5) 备用线,用于扩充功能或用户要求的特殊用途。

系统总线一般都作成多个插槽的形式,各插槽相同的引脚都连接在一起,总线就直接连接在这些引脚上。总线接口引脚的定义、传输速率的设定、驱动能力的限制、信号电平的规定、时序的安排以及信息格式的约定等等,都有统一的标准。

保证信息在总线上高速可靠地传输是系统总线最基本的任务。据此,系统总线最基本的指标主要有:

(1) 总线定时协议:为使信息源与信息接收能同步,在总线上传送信息时必须遵守一定的定时规则,通常有 3 种定时协议:

- 同步总线定时,所有模块都连接到公共时钟上,总线操作都在其同步时钟控制下的固定时间进行。

- 异步总线定时,总线操作由信息源或信息接收器的特定跳变所决定。
- 半同步总线定时,总线操作之间时间间隔按公共周期的整数倍变化。

(2) 总线带宽:总线带宽为总线本身所能达到的最高传输率,其单位是 Mb/s。例如,对 ISA 总线和 EISA 总线,总线时钟频率为 6.0MHz ~ 8.33MHz 时,ISA 总线的最大带宽为 16.66Mb/s,EISA 总线的最大带宽为 33.32Mb/s。总线带宽受三种因素的制约:

- 挂在总线上的模块数目。
- 总线布线长度。
- 总线驱动器和接收器的性能。

随着微电子技术和计算机系统设计技术的迅速发展,系统总线技术也在不断发展和完善,从性能和技术上看,系统总线正在朝着具有强有力的工业输入输出支持,组合灵活,价格低廉,系统处理能力强的方向发展。像我们所熟悉的 PCI 总线,以及笔记本型微机广泛使用的 PCMCIA(个人计算机存储器卡国际协会)总线和工作站中的 M 总线,就是典型的代表。

### 2.1.2 外部总线

如前所述外部总线用于在系统之间实现互连,可以作为可编程仪器、设备与计算机相连接的总线。因而在实现形式上,外部总线接口同系统总线也有很大差别。系统总线一

般都作成多个插槽的形式,集成在主板上,供用户扩展计算机的功能时插入各种适配器,如网卡、声卡、显卡等。而外部总线的接口则大都置于机箱之后,以便于连接各种不同的外围设备。

在第1章中我们已经看到,为了支持多种不同的应用需要,电脑系统接驳外围设备(如键盘、鼠标、打印机等)的输入/输出接口有许多独立运行且互不兼容的标准。如键盘的插口是圆的,连接打印机要用9针或25针的并行接口,以前鼠标则要用9针或25针的串行接口,而为了将机箱后的串口留给其它用途,如Modem之用,IBM推出了PS/2型接口,专门用于连接鼠标和键盘。可以说每一种总线标准都只能用于一个或几个方面,因而外围设备的接口并无统一的标准。所以说机箱后面要留有大小不等、形状各异的多种接口,十分烦琐。这种情况直到USB总线标准提出后,才出现了为PC机连接外设提供一种统一的标准的可能。

外部总线和系统总线一样,一般都拥有数据线、电源线、控制线。由于外部总线的线路不可能像系统总线那样可以多达几十条,分别用于数据传输、地址信号传送和控制信号发送等功能,另外每一种接口往往仅对应于一个设备,如串口(COM1, COM2)可以分别用于鼠标和Modem,并口用于打印机等,因此外部总线的控制操作相对于系统总线要简单一些,设备之间通常是利用一些握手信息来实现通信。而且外部总线无须支持很多外设,直接实现存储器访问等,因此也就没有必要提供像系统总线那样多的地址线来实现寻址功能,所以在很多情况下,我们可以利用片选信号(例如各种准备就绪信号和允许信号)来代替地址总线。

另外,依据数据传输方式的不同,外部总线可以分为串行和并行两大类。依据数据是否可以实现双向传输,可以分为全双工、半双工和单工等工作方式。同样,随着计算机技术的飞速发展,以及人们不断增长的应用需要,外部总线的改进和完善也在不断进行,新的标准也是层出不穷。其中最有代表性的当数本书的重点——USB总线,以及应用于高速数据传输领域的IEEE 1394总线。

## 2.2 串行总线和并行总线的比较

上一节我们已经提到,总线是为了实现计算机各模块之间的通信。而通信就是要在两个模块之间进行数据交互,这就必然涉及一个很基本的问题——那就是数据传输的方式问题。一般而言,数据传输的方式有两大类:串行通信和并行通信。

串行通信,就是所有的比特在总线上一个接一个地顺序传输。如我们所谈的USB使用的就是这一方式。而在并行通信方式中,则是几个比特同时在几条数据线上同时传输,如打印机的并行接口就是以并行方式工作的。比较而言,串行通信实现起来要比并行通信容易,而且它的传输距离也较远。

对于一个8bit的字符,如果使用串行方式,则这8个bit将按顺序传送。这与日常生活中的排队问题很相似。而若采用8条并行线路进行并行传输,即每个线上传输这个字符的1个bit,那么在并行总线上就同时有8个bit在传送。如图2-1所示。显然,在线路传输速率相同的条件下,并行通信比串行通信具有更快的通信速率。

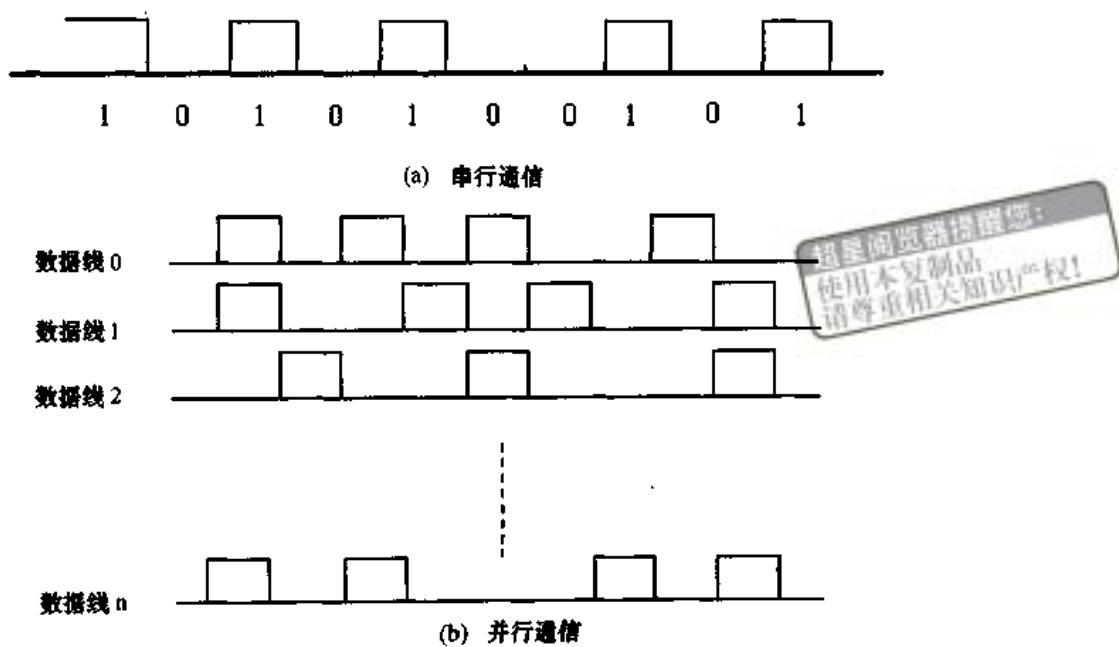


图 2-1 串行通信和并行通信的示意图

正是由于这种速度上的差别,所以在连接鼠标、键盘、调制解调器等低速设备时通常采用串行总线。而对 CPU、内存、磁盘驱动器等工作速度很高的设备而言,就要采用并行通信方式了。

### 2.2.1 并行接口

并行接口有以下几方面特点:

(1) 并行接口是在多根数据线上,以数据字节(字)为单位与输入/输出设备或被控对象传输信息的。如打印机接口,A/D、D/A 转换器接口,IEEE - 488 接口,开关量接口,控制设备接口等。并行接口的“并行”含义不是指接口与系统总线一侧的并行数据线而言(这时采用的都是并行数据线,以提高数据吞吐速率),而是指接口与 I/O 设备或被控对象一侧的并行数据线。

(2) 并行传送的信息,不要求固定的格式,这与串行传送的信息有数据格式的要求不同。例如,异步串行通信的格式是一个数据,它包括起始位、数据位、校验位和停止位。

(3) 从并行接口的电路结构来看,并行口有硬线连接接口和可编程连接接口之分。硬线连接接口的工作方式及功能用硬线连接来设定,用软件编程的方法不能加以改变;如果接口的工作方式及功能可以用软件编程的方法加以改变,则称之为可编程接口。

### 2.2.2 串行接口

串行通信是在一根传输线上一位一位地传送信息,所用的传输线很少,因此,特别适合远距离传输。对于那些与计算机相距不远的人-机交互设备和串行存储的外部设备如终端、逻辑分析仪、磁盘等,采用串行方式交换数据也很普遍。在实时控制和管理方面,采用多台微处理机组成分级分布控制系统中,各 CPU 之间通信采用的方式一般都是串行方式。所以说串行接口是微机应用系统的常用接口。

在并行通信中,传输线数目没有限制,一般除了数据线之外还设有通信联络控制线。例如,在开始传输数据之前,发送方要先询问接收方是否“准备就绪”(READY)或者是否“正忙”(BUSY)。接收方在接收到正确的数据之后,要向发送方返回一个“应答”(ACK)信号,告知发送方它已经成功地接收了数据,发送方可以接着发送下一个数据;如果接收方未正确接收数据,它就不会返回“应答”信号,这时发送方就要重传该数据;另外,如果发送方传输数据时,接收方“正忙”,它就会返回一个“否定应答”(NAK)信号,通知发送方它不能接收其发来的数据,因此发送方必须进行重传操作。但是,在串行通信中,由于信息在一个方向上传输,只占用一根通信线,因此这根线即作数据线又作联络线,也就是说要在一根传输线上既传输数据信息,又传送联络控制信息,这就是串行通信最主要的特点。这一点在我们深入探讨了 USB 规范之后,会对该特性有一个更深刻的了解。

那么如何在一根线上串行传输的数据流中识别出哪一部分是联络信号,哪一部分是数据信号呢?为了解决这一问题,就引入了串行通信的一系列约定。因此,串行通信的第二个特点就是它的信息格式有固定的要求(这一点与并行通信不同),分异步和同步信息格式,与此相对应,就有异步通信和同步通信两种方式。

(1) 异步通信通常采用的是起止式异步协议,其特点是一个字符一个字符传输,并且传输一个字符总是以起始位开始,以停止位结束,字符之间没有固定的时间间隔要求。如果没有字符传输,则信号电平处于空闲位,长度不定。由于采用起始位来表示字符传输的开始,并在每一次传输时都以此来重新核对收发双方的同步。若接收设备和发送设备两者的时钟频率略有偏差,也不会因偏差的累积而导致错位。所以异步串行通信的可靠性较高。但是每传输一个字符都要加上起始位和停止位,因此传输效率比较低,只有约 80%。因此,起止式协议一般用在速率较慢的场合(小于 19.2Kb/s)。对于要求更高速率传输时,一般要采用同步协议。

(2) 同步协议又有面向字符(Character - Oriented)和面向比特(Bit - Oriented)以及面向字节计数三种。这里仅讨论面向字符计数的同步协议,其它两种协议与此类似。这种协议的典型代表是 IBM 公司的二进制同步通信协议(BSC)。它的特点是一次传输由若干个字符组成的数据块,而不是只传送一个字符,并规定了 10 个特殊字符作为这个数据块的开头与结束标志以及整个传输过程的控制信息,它们叫做通信控制字。面向字符的同步协议,不像异步协议那样,需在每个字符前后加上起始和停止位,因此,传输效率提高了。同时,由于采用了一些传输控制字,也增强了通信控制能力和校验功能。但也存在一些问题,如应该如何来处理数据字符代码和特定字符代码的问题。这是因为在数据块中完全有可能出现和特定字符代码相同的数据字符代码,这就会产生误解。因此,协议中还设置了转义字符。用于指示此时应当把特定字符看作数据字符。总之,同步协议虽然传输效率提高了,但随之而来的是控制处理也比异步协议要复杂。

串行通信中的第三个特点就是对信息的逻辑定义与 TTL 不兼容,因此需要进行逻辑电平变换。

### 2.3 总线标准

前面已经提到,系统总线是微机系统中最重要的一类总线。我们所要探讨的 USB 总

线也必须通过系统总线(PCI 到 USB 的主控制器)来实现和主机的通信。因此,我们有必要来详细介绍一些常用的系统总线标准。在介绍之前,让我们先来看看,总线标准的具体定义。

总线标准是指国际工业界正式公布或推荐的连接各个模块的总线标准。它是把各种不同的模块组成计算机系统(或计算机应用系统)时必须遵守的规范。

总线标准为计算机系统(或计算机应用系统)中各模块的互连提供了一个标准界面,该界面对界面两侧的模块是透明的,界面任一方只需根据总线标准的要求来实现接口的功能,而不必考虑另一方的接口方式。按总线标准设计的是通用接口。采用总线标准可以为计算机接口的软、硬件设计提供方便。对硬件的设计,由于总线标准的引入,使各模块的接口芯片的设计相对独立。同时,也便于接口软件的模块化设计。

为了充分发挥总线的作用,每种总线标准都必须有详细和明确的规范说明。一般包括如下几部分:

- (1) 机械结构规范,确定模板尺寸、总线插头、边沿连接器等的规格及位置。
- (2) 功能规范,确定各引脚信号的名称、定义、功能与逻辑关系,对会相互作用的协议进行说明。
- (3) 电气规范,规定信号工作时的高低电平、动态转换时间、负载能力以及最大额定值。

在后面几章中,我们会看到 USB 总线的规范也是按照这样一个提纲来给出的。

### 2.3.1 PC/XT 总线、ISA(AT)总线及 EISA 总线

这几种总线是早期的比较有代表性的几种总线,曾经得到了广泛的使用,其中一些技术为以后的高性能总线的提出奠定了基础。目前,ISA 总线还有应用,读者仍然可以在主板上见到 ISA 扩展槽。

#### 1. PC/XT 总线

PC/XT 总线是最早的 PC 机的系统总线,是 IBM 公司于 1981 年推出的基于准 16 位机 PC/XT 的总线,也称 PC 总线。PC/XT 总线支持 8 位数据传输和 20 位寻址空间。其特点是把 CPU 视为总线的唯一主控设备,其余外围设备均为从属(Slave)设备,包括暂时掌管总线的 DMA 控制器或协处理器。

IBMPC/XT 总线是一种开放的结构总线,在其总线母板上有数个系统插槽,可用于 I/O 设备与 PC 机连接。其价格低、可靠简便、使用灵活、且对插板兼容性好,因此有许多厂家生产该总线的兼容产品,品种范围广泛。起初,IBMPC/XT 总线产品主要用于办公自动化,后来很快扩大到实验室或工业环境下的数据采集和控制。

#### 2. ISA(AT)总线

1984 年 IBM 公司推出了 16 位 PC 机 PC/AT,其总线称为 AT 总线。后来为了统一标准,便将 8 位和 8/16 位兼容的 AT 总线命名为工业标准结构(ISA——Industrial Standard Architecture)。

由于 ISA 是 8 位和 8/16 位兼容的总线,故插槽有 2 种类型,即 8 位和 8/16 位。8 位扩展 I/O 插槽由 62 个引脚组成,用于 8 位的插接板;8/16 位的扩展插槽除了具有 1 个 8 位 62 线的连接器外,还有 1 个附加的 36 线连接器。这种扩展 I/O 插槽既支持 8 位的插

接板,也支持 16 位的插接板。ISA 总线具有以下的主要性能指标:

- (1) I/O 地址空间 0100H~03FFH。
- (2) 24 位地址线,可直接寻址的内存容量为 16MB。
- (3) 8/16 位数据线。
- (4) 62+36 引脚。
- (5) 最大传输率 8Mb/s。
- (6) 中断功能。
- (7) DMA 通道功能。
- (8) 开放式总线结构,能使多个 CPU 共享系统资源。

ISA 总线结构示意图如图 1。

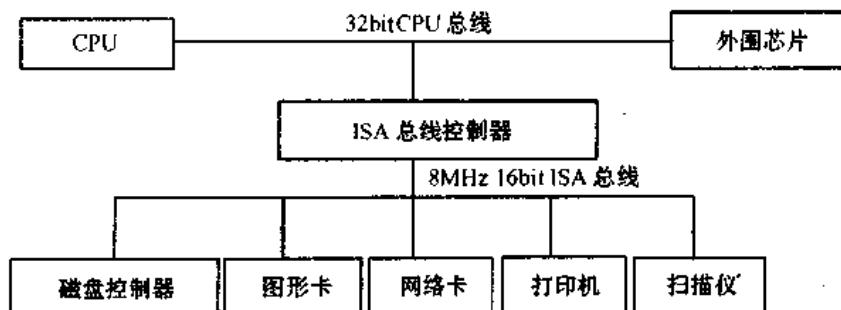


图 2-2 ISA 总线结构示意图

从表面看,ISA(AT)总线是在 PC/XT 总线基础上增加了 1 个 36 线插座形成的。但实际上,ISA(AT)总线比 PC/XT 总线不仅增加了数据线宽度和寻址空间,还加强了中断处理和 DMA 传输能力,并且具备了一定的多主控功能。故 ISA(AT)总线特别适合于控制外设和进行数据通信的功能模块。由于其数据宽度为 16 位,工作频率为 8MHz,数据传输速率最高为 16MB/s,所以 286、386、486 型微机大多采用 ISA 总线。

### 3. EISA 总线

EISA 总线,扩展工业标准结构(EISA—Extend Industrial Standard Architecture)是由 COMPAQ 牵头的一个联合组织共同制定的一组总线标准。EISA 总线是 ISA 总线的扩展,它与 ISA 总线完全兼容。EISA 总线是一种 32 位总线结构,除了 ISA 支持的 8 位或 16 位数据传输外,还支持 32 位地址总线和 32 位数据传输。与 ISA 总线相比,EISA 总线数据传输速度更快,且支持多主控总线功能,可使普通微机的单处理器系统升至多处理器工作状态,使运行达到峰值,其峰值总线数据传输率达 33MB/s。最多可支持 6 个总线主控,是一种支持多处理器的高性能 32 位数据总线。

#### 2.3.2 PCI 总线

进入 1993 年后,由于微处理器的飞速发展,使 ISA、EISA 总线均显落后。微处理器的高速度和总线的低速度不同步,造成硬盘、图形卡和其它外设只能通过 1 个慢速且狭窄的瓶颈发送和接收数据。因此推出了一项新技术——局部总线。

从结构上看,局部总线是在 ISA 总线和 CPU 总线之间增加 1 级总线。这样可将一些高速外设,例如网络适配卡、磁盘控制器等从 ISA 总线上卸下来而通过局部总线直接挂

接至 CPU 总线上,使之与高速的 CPU 总线相匹配。局部总线结构示意图如图 2-3 所示。

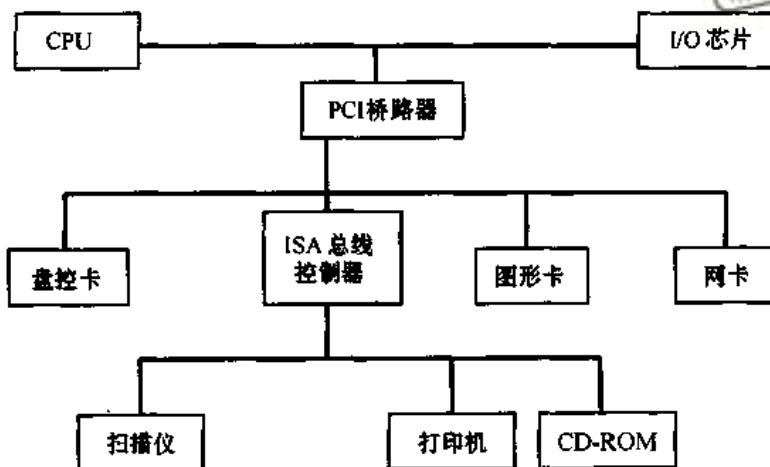


图 2-3 PCI 总线结构示意图

标准的局部总线中典型的是 PCI 局部总线(Peripheral Component Interconnect Local Bus),由 Intel 公司开发。其特点是在 CPU 和外设之间插入 1 个复杂的管理层,以协调数据传输并提供总线接口。由于采用了信号缓冲,PCI 能支持 10 种外设,并在高时钟频率下保持最高的传输速率。

PCI 局部总线允许在 1 个计算机上安装多达 10 个 PCI 附加扩展卡,允许将 ISA、EISA 等扩充总线控制卡安装在上面,以使所有已安装的系统总线更好地同步。PCI 控制器可以用 32 位或 64 位(这取决于设备)与系统的 CPU 交换数据。另外,还允许智能化的 PCI 辅助适配器采用一种称为总线管理(Bus Mastering)的技术协助 CPU 执行各种任务。PCI 规范允许复用,即在 1 个时间内允许有 1 个以上的电信号出现在总线上。PCI 总线较以前的总线具有独特优点:

(1) 高性能的多媒体技术。高性能图形、视像、网络对处理器提出很高的要求。PCI 局部总线提供了很宽的通路,允许这些应用程序相当平滑地执行。

(2) 真正的兼容性。与其它总线标准不同,任何与 PCI 相兼容的机器可用于任何遵从 PCI 的系统而不论其总线类型或所用的处理器。

(3) 与 Intel 处理器一起工作。PCI 总线可与 Intel 处理器协同工作,而不论其兆赫值从 Intel 486SX 处理器直至奔腾及未来的版本。

(4) 增长的余地。PCI 能以 132MB/s 的速率传送数据,远比 ISA 总线的 16MB/s 快,这使 PCI 可处理极高的容量,如用于全屏幕高质量视频的级别。

可见,PCI 局部总线为外设提供了访问微处理器更宽、更快的通路,有效地克服了数据传输的瓶颈现象。在目前一段时间内,PCI 局部总线接口是许多适配器的首选接口,如网络适配器、内置 Modem 卡、声音适配器等,且目前绝大多数主板上均带有 PCI 插槽。

### 2.3.3 AGP 总线

AGP 总线,AGP 加速图形端口(Accelerated Graphics Port)是一种用来连接 CPU 和图

形加速器的比 PCI 更快的总线。它是由主要的图形器件厂商及全球的 PC 机和 OEM 厂商组成的 AGP 执行者论坛(AGP—IF)在 1996 年 8 月提出的。Intel 公司决定将 AGP 技术主要用于基于 Pentium II 处理器的平台上。

AGP 总线特点:AGP 是专门为点对点的图形器件设计的,并非为了替代 PCI 总线,它是 PCI 总线的增强和扩充。综述与评论标准建立在 PCI2.1 标准的基础上。

在 PCI 总线下,许多外设如高速硬盘驱动器、高速网络适配卡、调制解调器、声卡等均需使用 PCI 总线传输数据,而 PCI 的数据带宽有限(峰值传输率为 132MB/s),若再加上 1 个需大量数据传输的 3D 显示卡,PCI 总线就会不堪重负,因此导致了 AGP 总线的产生。故 AGP 的设计核心是在 PC 机上实现高性能的 3D 图像。

#### AGP 性能特点:

- (1) 为 3D 图形显示提供高带宽(为 PCI 的 4 倍),使突发数据传输率达 526MB/s。
- (2) 采用了管线技术、边带寻址,在时钟上升沿和下降沿可同时存取数据。
- (3) 可直接对系统内存中的纹理图像进行处理。因为 AGP 允许图形控制器与系统内存的无缝连接,不必将纹理数据预载到本地显存上。
- (4) 减少了总线阻塞,提高了处理效率。PCI 总线就像一条单行道,其显示卡从内存中取得数据时,如同从显示卡开出一辆空车到内存去装货,必须等到该车将货物运回来后,才能发出下一趟车。这样在发出请求和得到数据之间有很多时间是在等待。AGP 总线对此作了很大的改进,AGP 更像一条高速公路,不但改善了路面质量提高了车速(提高了时钟频率),而且增强了总线控制器的功能,这样使得从 AGP 显示卡可以连续发出多辆车,减少了等待内存的寻址时间,大大提高了处理效率。

AGP 与 PCI 总线相比具有更高的时钟频率和数据传输率。对于 1 个系统基频为 66MHz 的奔腾级计算机来说,PCI 的工作频率为基频的 1/2 即 33MHz,相应的峰值数据传输率为 132MB/s;而 AGP 有 3 种模式: $\times 1$  模式、 $\times 2$  模式和  $\times 4$  模式,工作频率分别达 66MHz、133MHz 和 266MHz,相应的峰值数据传输率为 264MB/s、528MB/s 和 1GB/s。

AGP 有 2 种工作模式:一种是直接内存访问(DMA: Direct Memory Access)模式,另一种是直接内存执行(DIME: Direct Memory Execute)模式。

当 AGP 总线工作在 DMA 模式时,AGP 总线先将系统内存中的纹理和其它数据装载到图形加速器的本地内存中,接着图形加速器的各种处理工作如纹理映射、明暗度调整、Z 向缓冲等都在本地内存中执行。在此模式下,AGP 与基于 PCI 的图形加速器的工作方式大致一样。图形加速器只是拥有了 AGP 总线高速数据传输的优势。当 AGP 总线处于执行模式时,图形的数据可直接在系统内存中执行而不需要将原始数据全部传输到图形控制器。例如 3D 图形的一些特定操作如传输量最大的纹理映射可在系统的主存中直接处理,然后图形控制器将处理过的数据传输到显示缓冲区。这样做好处是可减少主内存和图形控制器之间的数据传输量,同时也节省了图形控制器的本地内存。为实现以上功能,就要求图形控制器必须能访问系统的主内存,能在主内存和显示内存之间传送数据。

限于篇幅,这里就不在对 AGP 进行更详细的介绍了,有兴趣的读者可以参考相关的技术资料。



## 2.4 流行总线的性能比较

总线的标准多种多样,选择哪一种标准,完全取决于用户的实际应用需要,没有任何强制性。这就为标准的多样性敞开了大门。除了上述我们所提到的几种总线之外,还有许多总线标准,我们选择了几种有代表性的,简述如下:

### 2.4.1 其它几种系统总线

(1) VME 总线:由 Motorola 公司推出的一种支持多计算机或多处理器系统的总线,由系统总线、局部总线和串行总线组成。有 4 个子总线,即数据传输总线、优先中断总线、仲裁总线和公用总线。VME 协议有两个层次,一层为底板访问层,由底板接口逻辑、公用总线模块和总线仲裁模块组组成;另一层为数据传输层,由数据传输总线和优先中断总线模块组成。VME 总线支持 8 位、16 位和 32 位的数据传输,数据传送速度为 40MB/s,采用异步传送方式,最大可支持 20 个总线主控,使用非多路复用的数据传输方法,是一种高性能的系统通用总线。

(2) MCA 总线:MCA 总线是 IBM 公司为 IBM PS/2 设计的总线,带有 32 位数据总线和 32 位地址总线,提供突发方式传输,配有总线仲裁机构,可支持 16 个总线主控。IBM 总线与 MCA 总线和 AT 总线都是不兼容的,因此其定义不受原有总线定义的限制。

(3) Multi Bus(多总线):Multi Bus I 是 Intel 公司为提高处理能力、支持多处理器并行运行而推出的 16 位微型计算机总线。它由系统总线、局部总线和板上输入输出扩展总线 SBX、LBX 等多种总线组成。为了适应 32 位微处理器的要求,Multi Bus I 经扩充后形成 Multi Bus II 总线,它除了定义物理层外,还采用 OSI(开放系统互连)通信网络的 7 层协议的办法,定义了链路层和更高层,是一种高性能的 32 位总线标准。

(4) STE 总线:1987 年由 IEEE 标准委员会批准的一种开放式结构的总线,是标准的欧洲卡总线。STE 总线具有多主控能力,有很好的独立性,即可以允许多种处理器接在 STE 总线上。采用异步握手方式,以适应不同速度的设备接口。

(5) STD 总线:1978 年由 Pro - Log 公司推出的 8 位工业标准总线,其特点是小尺寸 (4.5 in × 6.5 in)、高可靠性和低价格。16 位微处理器问世后,STD 总线采用总线复用和周期窃取的方法,可以同所有的 16 位微处理器完全兼容。32 位微处理器发展后,STD 总线一方面可以采用扩展的 STD 32 位总线方式,另外可以采用一体化方式和 32 位微处理器兼容。

(6) VL - Bus 是 VESA(视频电子学会)与 60 多家公司联合推出的一种全开放局部总线标准。VL - BUS 与中央处理机同步工作,总线最大传输率为 132MB/s(32 位数据)。主要特性为:

- 最多支持 3 个总线主控器。
- 支持 3 个 VL - Bus 槽。
- 支持高速视频控制器、硬盘控制器和 LAN 网络适配器等。
- 接口工作频率为 40MHz。
- 支持回写高速缓存。

### 2.4.2 其它几种串行总线

USB 是一种串行总线, 属于外部总线的范畴。目前还有另外几种被认为是串行总线的串行通信技术。每一种都是某种具体的应用范围所定义的。这些串行总线包括:

(1) ADB(Apple desktop bus): 这一专用的低要求的串行接口可以为最多 16 个设备提供简单的读/写控制协议。这种硬件接口的费用很低。ADB 所支持的速率可以达到 90Kb/s, 仅够用于支持键盘、鼠标或其它桌面 I/O 设备的通信需要, 也就是它只能支持低速设备。

(2) A.b(存取总线, Access.bus): 存取总线是由存取总线工业技术小组根据 Philips 提出的 I<sup>2</sup>C 技术和 DEC 软件模型而发展出来的。存取总线的应用范围主要是键盘和定点设备(如鼠标等);但是, 它比起 ADB 来说, 更为通用。该协议对于动态插拔、仲裁、数据分组、配置和软件接口都给出了很好的规范。而且它的地址空间可以支持最多 127 个外设, 实际的负载能力仅受到电缆长度和电源分配考虑的限制。

(3) IEEE 1394: IEEE 1394 是一种高性能的串行总线。它的应用范围主要是那些带宽要求超过 100Mb/s 的硬盘和视频外设。利用同样的四条信号线, IEEE 1394 即可以同步传输, 也可以支持异步传输。这四根信号线分为差模时钟信号线对和差模数据线对。IEEE 1394 规范得到了很好的定义, 而且基于 IEEE 规范的产品也出现在了市场上。目前, IEEE 1394 解决方案的价位认为被可以同 SCSI 磁盘接口相竞争, 但它不适用于一般的桌面连接。对于 IEEE 1394 和 USB 的区别, 我们将在第三章进行更详细的说明。

(4) CHI(Concentration Highway Interface): CHI 是由 AT & T 为终端和数字交换设备而设计的。CHI 是一种全双工, 时分复用的串行接口, 用于通信系统中的数字语音的传输。该协议包含了许多的用于装载语音数据和控制信息的固定时隙。目前, CHI 规范所支持的数据传输速率可以达到 4.096Mb/s。CHI 总线有四根信号线: 时钟、帧同步、数据接收和数据传送。帧同步和时钟信号都是在中心产生的(也就是由 PBX 交换机产生)。

(5) GeoPort: GeoPort 最初是由 Apple 电脑公司为了实现 Macintosh 机的电话应用而发展出来的。目前, GeoPort 规范所支持的数据传输速率可以达到 2Mb/s, 并可以在半径为 4 英尺的范围内提供点到点的连接。标准的 GeoPort 指定了一个 9 针连接器(8 针和一个可选的电源引脚), 并适用于 RS-422 信号。另外, 为了扩展电缆长度, Apple 还规定了一种可供替换的 14 针连接器。GeoPort 协议支持三种不同的工作方式: 信标、TDM 和分组传输方式。Apple 电脑公司目前正在对 GeoPort 规范进行授权工作。

以上是几种目前最新提出的串行总线规范的简述, 目的是使读者对串行总线的发展有所了解。通过以上内容我们可以看到, USB 的速率为 1.5Mb/s 和 12Mb/s 两种, 虽然它的速率比起 IEEE 1394 要低, 但是它们的应用领域并不相同。因此, 读者在选择应该采用哪一种总线技术时, 应当明确各种总线的设计目的, 即它的主要应用领域, 然后根据自己的具体需要, 选择一种总线规范来实现。

尽管各类总线在设计细节上有许多不同之处, 应用范围也不尽相同, 但从总体上看, 它们都必须解决信号分类、传输应答、同步控制、资源共享和资源分配等问题。因此, 它们的主要性能指标是可比较的, 这对于系统设计者是很重要的, 对于接口电路板的设计者来说也是重要的。

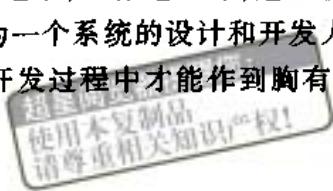
表 2-1 列出了流行总线的性能比较,仅供读者参考。

表 2-1 流行总线性能比较

总线类型	适用机型	工业参考	总线宽度 (bit)	总线工作速率 (MHz)	最大传输速率 (MB/s)
PC/XT	8086 个人计算机	PX/XT	8	4	4
ISA (16位)	80286, 386, 486 系列 PC	Industry Standard Architecture	16	8	16
EISA	286, 386, 586 系列 个人计算机	Extended Industry Standard Architecture	32	8.33	33.3
STD	Z - 80, V20, V40, IBM 系列 PC	STD Bus	8	2	2
MCA (32bit)	IBM PC 机 与工作站	Microchannel Architecture	32	8	33
VL-BUS	I486, PC - AT 兼容机	VESA Local Bus	32	33 40 50	128 ~ 132
PCI	Pentium 系列 PC, Power PC, Alpha 工作站	Peripheral Component Interconnect	32 或 64	33	132 或 264(64 位)
SCSI	Pentium 系列 PC, 工作站 和小型机	Small Computer System Interface			
1394	Pentium 系列 PC 和工作站	FireWire		100 ~ 400	100 ~ 400 Mb/s
USB	Pentium 系列 PC	Universal Serial Bus		12	12 Mb/s
AGP	Pentium 系列 PC	Accelerated Graphics Port	64	66 以上	264 或更高

以上我们从总线的宽度、总线工作频率、最大数据传输速率等三个方面,对几种常用总线进行了比较。由于总线的规范所涵盖的内容很多,如时钟同步/异步,数据总线的多

路复用和非多路复用,负载能力,总线控制方式,信号有效电平,工作电压等,这里就不再一一予以说明了。但是,并不是说这些内容都不重要,作为一个系统的设计和开发人员来讲,必须全面细致地了解总线规范的每一个细节,这样在开发过程中才能作到胸有成竹,有的放矢。



## 第3章 USB总线规范

随着通信融合趋势，  
交换的信息手段。但是，长期以来通信技术与计算机技术是两个完全独立地发展的。那么能诞生一种新的技术来实现计算机和通信设备之间的互连呢？

现在，由于PC机在进行重新配置时缺乏足够的灵活性，已经影响了PC机的进一步普及。随着友好的用户图形界面PCI、PnP ISA 和 PCM CIA 等像当初那么直观了，而且要重新认识PC机上的I/O接口如串行/并行有没有一种可以使用户使用；

随着计算机的普及，人们可能的外设也就不断出现，与之相适应的、廉价的，支持中低速外围设备的进一步发展，而现有的

本章将介绍一些与USB总线规范有关的概念和USB的性能优势，同时就USB和IEEE 1394进行了比较，供广大读者参考。

本章具体包括以下内容。



概述；



应用范围分类；



USB的特点；



USB与IEEE 1394的比较；

的功能的外设,就必须在机箱上增加一个新的接口来对其进行寻址。谁也无法说出来未来会出现什么新的外设以及会出现多少。这样一来,这种机械地增加端口的方法就显得很不方便了。那么,能否利用新的技术来简化端口的扩充,适应未来发展的需要呢?

这三方面的要求是相互联系的,而USB技术就是在这三个方面需求的推动下发展而来的。它是一种快速的、双向的、同步传输的、廉价的、并可以进行热插拔的串行接口,它是与当今计算机的发展趋势相适应的。

总而言之,USB技术是为实现计算机和通信的集成(CTI)而提出的一种用于扩充PC体系结构的工业标准。它的设计目标可以概括如下:

- (1) 简化扩充PC机外设的操作。
- (2) 为对传输速率高达12Mb/s的支持提供了廉价的解决方案。
- (3) 完全支持实时数据(语音、音频和压缩视频信号)的传输。
- (4) 协议灵活,即可以用于同步数据传输,也可以用于异步消息传输。
- (5) 同商用设备技术实现了集成。
- (6) 提供一个标准的接口,可以迅速地应用于生产。
- (7) 可以适应未来发展的需要,连接可以增强PC机功能的新的类型设备。

## 3.2 应用范围分类

图3-1对通用串行总线(USB)可以提供服务的数据通信流量的范围进行了分类。正如我们所看到的一样,12Mb/s总线可以包含中速和低速数据通信范围。最典型的是,

性能	应用	属性
高速率 ■视频、磁盘 ■25Mb/s~500Mb/s	视频 磁盘	带宽要求高 保证时延要求 使用方便

图3-1 应用范围分类

中速数据类型是同步的,而且低速数据是来自于交互的设备。USB 被建议主要用于实现桌面总线,但它却已经可以应用于移动系统中了。通过提供对多个 USB 总线主控制器的支持,软件结构可以允许未来对 USB 进行扩展。

正如第 1 章中所提到的那样,USB 相对于其它高速总线而言,速度并不快。所以,它仅适用于中低速的应用。 $12\text{Mb/s}$  的传输速率对于支持像鼠标、键盘这样的设备是绰绰有余。对于像打印机、扫描仪和调制解调器,也足以应付了。然而,要想支持更高速率的应用,对于 USB 来说,就是勉为其难了。至少目前 USB 规范 1.0 版支持的速率仅为  $1.5\text{Mb/s}$  和  $12\text{Mb/s}$  两种。若要支持更高的数据传输要求,就得求助于其它总线标准了,例如在第 2 章中提到的 IEEE 1394 总线。就目前的多数外设而言, $12\text{Mb/s}$  的带宽已经可以获得前所未有的性能提高了。目前,USB 论坛(USB - IF)正在进行 USB 规范 2.0 版的制定工作,预计很快就会推出,相信它所支持的速率一定会有更进一步的提高,其适用范围也会更加广泛。

### 3.3 USB 的特点

USB 总线具有一些固有的特点,使得它很容易满足中低速外设的需要。USB 易于使用,管理和设计,因而得到了广泛的应用。其特点主要有:

- 对用户而言隐藏了技术细节。
- 具有广泛的应用领域。
- 带宽足以保持多媒体应用的需要。
- 可靠。
- 设备与系统相互独立。

USB 总线的一个显著特点是对用户隐藏了应用细节和大部分的配置操作。USB 端口和电缆都有确定的规格,因而不会在连线时出现混淆。另外,USB 技术还支持即插即用功能(所谓“即插即用”,就是指计算机可以自动发现和配置 USB 设备,而无须用户进行端口地址和中断使用的配置工作)。并可以进行“热插拔”(所谓“热插拔”就是指系统上电后可以自由地插拔 USB 设备,而不会对系统产生任何影响)。

之所以提出 USB 总线技术的主要原因就是想利用单一的总线技术,来满足多种应用领域的需要。在 USB 总线上,可以同时支持低速( $1.5\text{Mb/s}$ )和高速( $12\text{Mb/s}$ )的数据传输;而且可以支持异步(如键盘、游戏杆、鼠标)传输和同步传输(如声音,图像设备)两种传输方式;它还可以同时支持多达 127 个外设。可见 USB 总线技术的提出使人们渴望利用单一的总线技术来实现多种外设同主机互连的梦想得以实现。

对于音频、视频信号这些对带宽要求很高的信号而言,USB 可以为其提供足够的带宽来保证其传输,这称为“同步传输”。而对于其它一些突发性较强的信号,USB 可以利用当前可用的带宽进行传输,这称为“异步传输”。与“同步传输”相比较,“异步传输”没有固定的带宽保证,USB 仅采用“力所能及”的方式来传送信息。如果没有足够的带宽,有些信息就会被丢弃或阻塞。

USB 总线使用起来非常可靠,因为它在协议层提供了很强的差错控制和恢复功能。而且 USB 总线是与系统完全独立的。只要有软件的支持,同一个 USB 设备就可以在任

何一种计算机体系中使用。这种良好的兼容性也使得 USB 技术可以迅速地发展壮大。

根据 USB 总线规范的定义,USB 总线的优点可以详细说明如下:

(1) 方便终端用户的使用

- 电缆和连接器具有唯一的型号。
- 对终端用户隐藏了电气细节,例如总线终结。
- 外设可以自我识别,并可以自动完成配置和到驱动程序的功能映射。
- 支持动态接入,并可以重新进行配置外设。

(2) 工作负荷和应用范围广

- 可以同时支持速度为几 Kb/s 至几 Mb/s 的设备。
- 在同一套总线上可以同时支持同步和异步传输类型。
- 支持多连接,支持对多个设备的同时操作。
- 支持多达 127 个物理外设。
- 在主机和设备上支持对多个数据和消息流的传输。
- 允许使用复合设备,即具有多个功能的外设。
- 具有较小的协议开销,因而总线利用率较高。

(3) 实现费用低廉

- 提供了十分经济的 1.5Mb/s 子通道。
- 为外设和主机硬件中的集成进行了优化。
- 可以用于开发许多价格便宜的外设。
- 所需的电缆和连接器价格低廉。
- 利用了商用技术。

(4) 同步带宽

- 可以为电话,音频信号提供确定的带宽和很小的时延。
- 同步载荷可以使用总线上的全部带宽。

(5) 同 PC 工业协同作用

- 实现和集成时协议简单。
- 符合 PC 即插即用体系结构。
- 对现存的操作系统接口性能产生了巨大影响。

(6) 灵活性

- 可以有很多不同大小的分组,并允许在一定范围内选择设备的缓冲区。
- 通过支持不同的分组缓冲区和时延要求,USB 可以支持许多具有不同的数据速率的设备。

- 在协议中提供了用于控制缓冲区的流控功能。

(7) 稳定性

- 协议中包括了差错控制/缺陷发现机制。
- 可以动态地插入和拔出 USB 设备。
- 支持对缺省设备的识别。

(8) 升级途径

- USB 体系结构可以升级,从而在一个系统中支持多个通用串行总线控制器。

### 3.4 USB 与 IEEE 1394 的比较

在第 2 章计算机总线概述当中,我们已经提到了 IEEE 1394 总线标准。虽然我们还提到了另外的四种总线,但它们同 USB 的差别比较大,在此就不做详细的介绍了。而 IEEE 1394 却和 USB 有许多相似之处,因而有必要对 IEEE 1394 和 USB 这两种先进的串行总线技术进行比较,使读者掌握其异同点,以便在今后的实践当中,根据自己的实际需要,选择一种标准来开展开发工作。

同 USB 相比,IEEE 1394 的速率更高。IEEE 1394 接口标准化工作始于 1986 年,由 IEEE 1394 委员会主持。从 1988 年开始,Apple 公司 Michael Teener 着手研究 IEEE 1394 的基本技术,1992 年 Apple 公司提案被采纳为 IEEE 1394 标准规范。1994 年 9 月成立 IEEE 1394 Trade Association,主持推进以 IEEE 1394 为标准的家庭网络规格普及工作,并推出了用于保证高质量和兼容性的规范。例如:欧洲数字化视频广播 DVB(DIGITAL VIDEO BROADCASTING)联盟决定把 IEEE 1394 作为遥控器及其相关机器的标准总线;由 50 多家企业构成的数字化视频摄像机联盟,把 IEEE 1394 作为数字化视频和音频的标准接口规格;具有决定性意义的是美国硬盘驱动器厂家 Seagate 等公司把 IEEE 1394 作为硬盘接口。

IEEE 1394 具有如下主要特性:

#### 1. 高速数据传送

目前市场的 IEEE 1394 用的 LSI 电路一律支持 400Mb/s 数据传输速度,而且为了适应新的需求,人们仍在研究更高速的传送,如开发 800Mb/s 和 Gb/s 级(千兆比特)的 LSI 电路。电气规范也在扩充,如引进 IP 的规范标准化和进行连接不同网络用规范的标准化。

#### 2. 保证实时性

1394 接口具有高速性和实时性。支持异步传送和同步传送两种模式,而同步传送模式专用于实时地传送视频和音频数据。

#### 3. 高自由度连接/拓扑结构

IEEE 1394 接口允许结点菊花链(Node Daisy Chain)和结点分枝,实现混合连接。同时,通过协议时序优化(Protocol Timing Optimization),可实现更高效率的网络结构。尽管 1394 规范允许 Daisy Chain,但若一味以一线串珠方式的(Straight Line)连接,最多只能连接 16 台设备;只有采取混合连接才能实现额定的 63 台设备连接。并且,当用户连接时,结点超过规定的 4.5m 时,就不能再使用廉价的 6 芯电缆布线了,必须改用新产品 POF(Polymer Optical Fiber)。

#### 4. 带电插拔/即插即用

1394 接口的通信协议已明确规定,当网络上附加结构和撤销结点时,能够自动地实现网络重构和自动分配 ID。因此允许 HotPlugIn(热插拔)和 Plug & Play(即插即用),对用户十分方便。

有一点要注意的是,由于 1394 结构的所有资源,都是以统一存储器编址形式用存储器映射(Memory - Maps)方式(IEEE 1212 规范)识别,实现资源配置和管理。因此,从这

种意义上观察,IEEE 1394 是总线体系结构;它向各设备发送数据时,它就是存取由 IEEE 1212 映射存储空间的总线体系结构;也就是从高层次观察,IEEE 1394 也和 PCI 总线等同样是总线。这也就是有些资料把 IEEE 1394 接口也称为 IEEE 1394 总线的原因。

在上一节中,我们已经详细讨论了 USB 总线技术的特点。因此,综上所述,USB 和 IEEE 1394 串行接口具有以下几个共同特点:

- (1) 信号线条数少,可用细而柔的轻便电缆。
- (2) 电缆细软导致可用小巧的连接器。
- (3) 不需要标识符 ID 设备和终端设定。
- (4) 在不切断电源的情况下,可自由地向系统里接入或切断设备连接。
- (5) 两者都支持同步(Isochronous)传送模式,适合于多媒体数据实时处理,可保证图像等数据显示不间断,提高画面质量和确保实时播放。

表 3-1 给出了 IEEE 1394 和 USB 两种串行总线技术的性能比较表,供广大感兴趣的读者参考。

表 3-1 IEEE 1394 和 USB 性能比较表

性 能	IEEE 1394 - 1995	USB
数据传送速度	100、200、400(Mb/s)	1.5Mb/s, 12Mb/s
可连接结点数	63 个	127 个
结点之间距离	4.5m(可延长至 50m~100m)	5m(对全速率电缆而言)
同步传送模式	支持同步传送	支持同步传送
信号线条数	6 条(电源:2,信号:4)	4 条(电源:2,信号:2)
编码方式	DSTLink	NRZI

IEEE 1394 和 USB 都是新一代的多媒体 PC 机的外设接口。当前 USB 用于连接中低速外设,而 IEEE 1394 则可连接高速外设和信息家电设备(尤其适合连接高档视频设备,当前这一技术已经在 Windows 98 中通过添加卡形式向 PC 提供支持)。从性能上观察,USB 的应用局限于 PC 机领域,而 IEEE 1394 应用领域将扩展到通信和信息家电。

### 3.5 有关 USB 的几个重要概念

USB 总线是一项新技术,因此对于初学者来说有些名词很生疏,而有些在 USB 中已经有了全新的含义。在展开后面内容之前,很有必要介绍一下一些关键性的名词,从而使大家首先树立起某些概念。这些概念对于深入理解 USB 总线技术是十分重要的,为了方便读者查阅,书后附录中列出了所有的名词。这里仅对几个笔者认为最重要的概念加以注释,其它内容请参看附录。

在所有有关 USB 技术的概念中,管道和端点是两个十分基本的概念。在介绍这两个概念之前,我们先来说明主机的概念。在 USB 的具体应用中,是主机和另外的 USB 设备组成了整个 USB 世界,主机负责发现总线上新插入的 USB 设备,并在拔下某一 USB 设备后释放先前为它分配的系统资源等操作。可以说主机是 USB 世界的核心,这里的“主机”(USB Host)是指安装了 USB 主控制器的计算机系统,它不仅包括 CPU 总线等硬件,

还包括当前正在使用的操作系统。其中 USB 主控器是指 USB 总线的主接口。

所有的通信都是在 USB 主机和其它 USB 设备之间进行的(这与以太网是不同的),这时就要涉及前面提示的两个基本概念端点和管道。端点(Endpoint)仅是主机和设备之间一个逻辑的通道。每一个 USB 设备都支持几个确定的端点。而每个端点仅与一个方面的数据传输相对应,所以在 USB 主机和其它 USB 设备之间的双向传输就要有两个“端点 endpoint”相对应。总之,“端点”是所有 USB 设备中唯一可以视为其相同的部分,它与主机和设备之间某一方向的数据传输相对应。

在 USB 主机上的一个软件功能和一个 USB 设备之间建立的一个虚连接称为“管道”。在 USB 技术中有两种类型的管道——流管道和消息管道。“流管道”是指没有确定的总线帧结构而以数据流的方式进行数据传输的“管道”。“消息管道”中的数据具有一定的帧结构。因而其数据传输就可以与所需的带宽、传送类型和端点特征(传送方向和缓冲区大小)相适应。

管道是在一个设备插入系统后,由位于 USB 主机上的软件建立的。所有的 USB 设备都必须支撑“管道 0”从而使 USB 主机可以利用该管道对 USB 设备进行配置。有关管道的更详细的讨论请参见第 5 章,这里仅给出基本概念,以方便读者对后面内容的理解。

## 第4章 USB总线体系结构

本章阐述了 USB 总线的体系结构和工作原理。当一个主机同时连接多个外设时，它会通过轮询的方式来共享 USB 的总线带宽。在主机和其连接的设备之间，可以随时插入或拔下一个新的 USB 设备，这就是我们在前面几章中提到的即插即用特性。

### 4.1

本章具体包括以下内容：

一个 USB 系统可以从三个方面来考虑：  
(1) USB 互联。  
(2) USB 设备。  
(3) USB 主机。



USB 系统描述；



总线拓扑结构；



物理接口；



电源；

USB 互联是指一个 USB 设备与 USB 主机相联并和其通信的方式,它包括:

- (1) 总线拓扑结构:USB 主机和 USB 设备的连接模型。
- (2) 层间关系:USB 在系统中的每一层都要完成一定的任务。
- (3) 数据流模型:USB 系统中信源和信息之间的数据传送方式。
- (4) 任务规划:USB 提供可以共享的互联机制。通过规划对互连机制的访问,可以支持同步数据传输。

在以下几节中将详细讨论 USB 设备和 USB 主机。

#### 4.1.1 总线拓扑结构

USB 设备和 USB 主机通过 USB 总线相连。USB 的物理连接是一个星型结构,集线器(Hub)位于每个星形结构的中心,每一段都是主机和某个集成器,或某一功能设备之间的一个点到点的连接,也可以是一个集线器与另一个集线器或功能模块之间的点到点的连接。参见图 4-1。

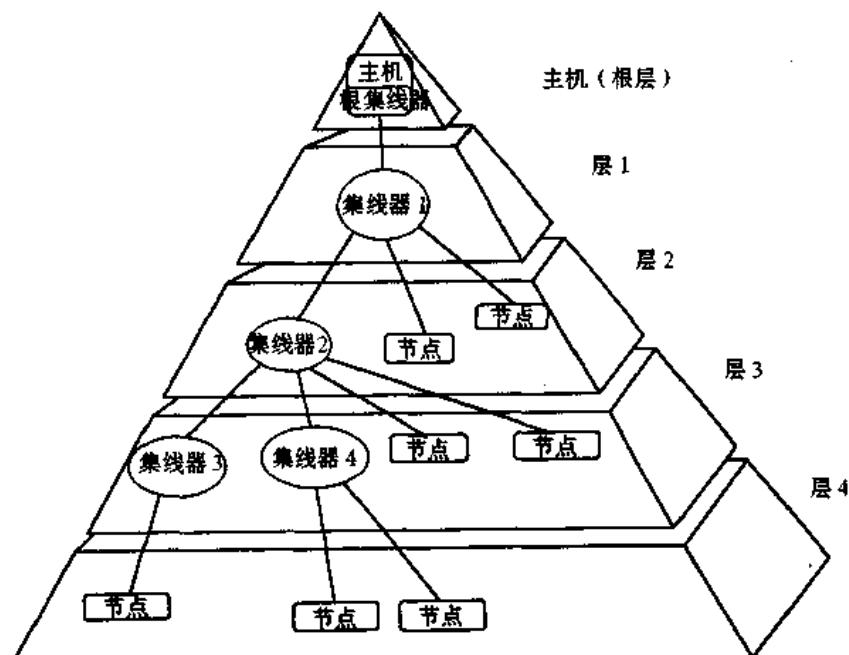


图 4-1 总线拓扑结构

##### 1. USB 主机

在整个 USB 系统中只允许有一个主机。主计算机系统的 USB 接口称之为 USB 主控制器。这里 USB 主控制器可以是硬件、固件或软件的联合体(参见 4.6 节)。而根集线器是集成在主机系统中的,它可以提供一个或更多的接入端口。有关主机的更详细的介绍请参考本章 4.9 小节和第 10 章。

##### 2. USB 设备

在第 1 章已经提到了许多 USB 设备,可以说那些都是 USB 协议的具体实现。这里从协议的角度再来讲述一下 USB 设备,它包括:

- (1) 集线器:提供用以访问 USB 总线的更多的接入点。

(2) 功能部件:向系统提供特定的功能,如 ISDN 连接设备、鼠标、显示器等。

一个 USB 设备要正常工作,必须满足以下条件:

(1) 支持 USB 协议。

(2) 可以对诸如配置和复位等标准的 USB 操作作出响应。

(3) 具有标准的描述消息。

进一步的讨论参见本章第 4.8 小节和第 4.9 节。

## 4.2 物理接口

USB 的物理接口包括电气特性和机械特性两部分。详见第 6、7 章。

### 4.2.1 电气特性

USB 总线中的物理介质由一根 4 线的电缆组成(图 4-2),其中两条用于提供设备工作所需的电源;另外两条用于传输数据。信号线的特性阻抗为  $90\Omega$ ,而信号是利用差模方式送入信号线的。利用这种差模传输方式,接收端的灵敏度可以达到不低于  $200\text{mV}$ 。

USB 支持两种信号速率。USB 的最高速率是  $12\text{Mb/s}$ ,但它可以工作在  $1.5\text{Mb/s}$  的较低速率,而这种较低的传送速率是依靠较少的 EMI 保护而实现的。利用一种对设备透明的方式来实现数据传送模式的切换。同一个 USB 系统可以同时支持这两种模式。 $1.5\text{Mb/s}$  低速率方式用来支持数量有限的像鼠标这样的低带宽要求的设备,这类设备不能太多,因为其数目越多对总线利用率的影响就越大。

时钟信号编码后是同差模数据信号一起在信号线上传输的。时钟信号的编码方式采用 NRZI(非归零)方式,为了保证有足够的跳变沿进行了比特填充。接收器利用每一个分组前的 SYNC 域来同步它的比特恢复时钟。

USB 电缆中还有两条用来向设备提供电源的电源线,即 VBus 和 GND。VBus 在源端的标称值为  $+5\text{V}$ 。通过选择合适的导线规格来匹配特定的 IR 下降以及功率预算和布线的灵活性等其它属性,可以允许 USB 使用不同长度的电缆,最长可达几米。为了提供可靠的输入电压和适当的终端阻抗,在电缆的每一端都有一个带偏压的终端。该终端可以发现任一端口上 USB 设备的“插入”和“拔除”操作,并能区分全速和低速设备。

有关电气规范的详细讨论请参考第 6 章。

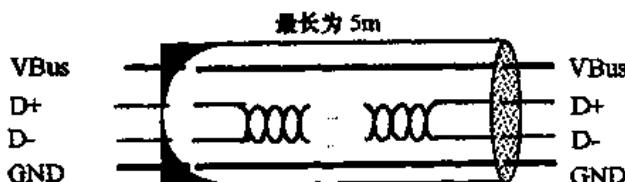


图 4-2 USB 电缆

### 4.2.2 机械特性

对于所有的 USB 设备而言,都有“上行”连接。“上行”(Upstream)和“下行”(Downstream) 连接器在机械方面并不是可以互换的,所以要尽量消除集线器上出现非法的环路。

连接。一条电缆拥有四根导线：一对具有标准规格的双绞信号线，和一对在允许的规格范围内的电源线。每个连接器（无论 A 系列还是 B 系列）都具有四个接触点，并且具有屏蔽的外壳、规定的坚固性和易于插拔的特性。

机械特性规范详见第 7 章。

## 4.3 电 源

有关 USB 电源的规范包括下面两个方面：

- (1) USB 总线上的功率分配问题，这涉及所有 USB 设备如何利用主机所提供的电源功率问题。
- (2) 功率管理，这主要涉及如何使 USB 软件和设备同基于主机的功率管理系统相适应的问题。

### 4.3.1 功率分配

每一个 USB 段上，电源电缆都是提供受限的功率。主机向直接与其相连的 USB 设备提供电源。而且，每一个 USB 设备都可以有它自己的电源。那些完全依赖于 USB 电缆为其供电的 USB 设备称为总线供电设备 (Bus-powered Devices)。与之相对，那些具有可替换电源供应的 USB 设备称为自供电设备 (Self-powered Devices)。一个集线器也可以为其连接的设备供电。

### 4.3.2 电源管理

USB 主机拥有一个独立于 USB 的电源管理系统。USB 系统软件和主机电源管理系统相互作用，以此来控制像挂起或恢复等系统电源事件。另外，USB 设备可以具有 USB 技术规范所规定的电源管理信息，这样一些设备就可以由系统软件或类设备驱动程序来对其进行电源管理。

USB 电源分配和电源管理功能，可以使 USB 技术应用于像由电池供电的笔记本电脑这样对电源敏感的系统。

## 4.4 总 线 协 议

总线上的所有处理都包括最多三个分组的传输。每一次处理操作开始时，都是由 USB 主控制器根据一个计划的步骤，发送一个用于描述处理类型和方向、USB 设备地址、端点 (endpoint) 号的 USB 分组，这一分组被当作令牌分组。被寻址的 USB 设备通过对恰当的地址域进行解码就可以知道这是发给自己的分组。对一个特定的处理操作而言，数据总是由主机传向 USB 设备或者由 USB 设备传向主机。这种数据传送方向在令牌分组中加以规定。然后处理操作的信宿就可以发送数据分组或指出它自己没有数据需要发送。通常情况下，信宿通过“握手分组”来指明这次传送是否成功。

USB 总线上的这种在主机和设备端点之间的数据传输模型称之为“管道”。共有两

种管道类型：流管道和消息管道。流管道中的数据没有确定的USB帧结构，而消息管道中的数据却有。另外，管道还同数据传输带宽、传送服务类型和像传送缓冲区大小这类的端点特性相联系。只要某一USB设备完成了配置之后，就会存在管道0。当一个USB设备上电后，控制管道0这一消息管道就总是存在。因为这一管道要提供对设备配置、状态和控制信息的访问。

对处理操作进行安排可以对一些流模式的管道实现流控功能。对于硬件而言，流控功能可以使用NACK信号来扼制数据速率，以此来防止缓止区溢出情况的发生。当有可以利用的总线时间出现时，系统可以为一个收到否定应答的处理操作重新发出令牌分组。这种流控机制允许建立灵活的操作规划，从而可以服务于许多不同种类的流管道通信。因此，在USB中多个流管道可以拥有大小不同的分组，并可以在不同时间获得服务。

## 4.5 稳定性

利用USB技术，可以使系统运行得更加稳定，其特点概括如下：

- (1) 利用差模驱动器、接收机和屏蔽技术来确保信息的完整性。
- (2) 对控制信号和数据域进行了CRC保护。
- (3) 可以检测出设备的插拔状态，并对资源进行系统级的配置。
- (4) 对丢失或遭破坏的分组利用超时功能，使USB协议具有自愈功能。
- (5) 对数据流进行流量控制，以此确保同步性和对硬件缓冲区的管理。
- (6) 分别建立控制和数据分组，确保不同功能模块之间相反的处理的独立性。

### 4.5.1 差错检测

我们希望USB物理媒介的误比特率与底板的误比特率相近，而且实际上任何假信号脉冲都很可能是暂时的。为了提供对这种暂时性的干扰保护，每一个分组都包含差错保护域。当需要保证数据完整性时，例如对于要求数据无任何丢失的设备，差错恢复过程就会在硬件或软件中启动。

该协议中每一个分组包括了与控制和数据域相分离的CRC域。CRC用于指出受破坏的分组，它可以发现单个或两个比特的错误。

### 4.5.2 差错控制

协议允许在硬件或软件中有选择地实现差错控制功能。硬件控制操作包括对失败的传送进行报告和重试操作。控制器在向客户软件通告错误之前，要进行3次重试操作。客户软件会根据特定的应用而进行恢复操作。

## 4.6 系统配置

USB协议支持在任何位置、任何时间插入USB设备，或从USB总线上移出USB设

备。因此,USB 的物理总线技术必须允许总线上动态的变化。



#### 4.6.1 插入 USB 设备

所有的 USB 设备都通过集线器上特定的端口而接入 USB 总线。集线器在各端口的状态中指出 USB 设备的接入或拆除。主机将向 USB 集线器进行询问来决定其指示的原因。集线器会作出响应,指明已经用于连接 USB 设备的端口。主机通过控制管道,使用缺省的 USB 地址来使这些端口可用,并对 USB 设备进行寻址。所有的 USB 设备在刚接入 USB 总线或复位之后,都要利用 USB 缺省地址来寻址。

主机将决定新接入的 USB 设备是一个集线器还是一个功能模块,并为 USB 设备分配一个唯一的 USB 地址。主机利用已分配的 USB 地址和端点号“0”来建立该 USB 设备的控制管道。

如果新接入的 USB 设备是一个集线器,并且它的端口接有其它的 USB 设备,那么对所有的 USB 设备都会进行上述的操作。

如果新接入的 USB 设备是一个功能模块,那么 USB 软件将向有关的主机软件发出接入指示。

#### 4.6.2 拆除 USB 设备

当从集线器的一个端口移出某一 USB 设备时,集线器会自动废除该端口并向主机发出设备移出指示。而主机则会从主机数据结构中删除该 USB 设备的信息。

如果移出的 USB 设备是一个集线器,则需要对先前接入集线器的所有 USB 执行移出操作过程。

如果移出的 USB 设备是一个功能模块,那么移出信息将会送到相关的主机软件中。

#### 4.6.3 总线枚举

总线枚举是指对总线上接入的 USB 设备进行识别和寻址操作。对许多总线而言,这一工作都是在启动时完成的,并且所搜集的信息也是静态的,而 USB 技术允许在任何时间从 USB 总线上插入或拔除 USB 设备,因此,USB 总线的总线枚举操作是一种持续执行的工作。而且,USB 总线枚举还包括发现和处理设备的移出。这是 USB 与以往一些总线技术的区别之一。

#### 4.6.4 层间关系

USB 设备在逻辑上可以分成一个 USB 设备接口部分,一个设备部分和一个功能部分。而 USB 主机在逻辑上可分成 USB 主机接口部分,集合系统软件部分(USB 系统软件和主系统软件)和设备软件部分。

所有这些定义的部分使得一个具体的 USB 任务仅是一个部分的责任,即只由一个部分负责,而无须其它部分参与。USB 主机和 USB 设备各部分示于表 4-1。

表 4-1 主机和设备的层间关系

USB 主机部分	USB 设备部分
设备软件	功能模块
系统软件	设备
USB 接口	USB 接口

## 4.7 数据流类型

USB 协议支持以单向或双向的方式,在 USB 主机和一个 USB 设备之间交换功能数据和控制信息。USB 的数据传输是在主机软件和一个 USB 设备上的特定端点之间进行的。一个给定的 USB 设备支持多个数据传输端点。USB 主机将分别地处理一个 USB 设备的任一端点和其它端点上的通信。这种主机软件和一个 USB 设备端点之间的联系称为管道。比如,一个给定的 USB 设备可以有一个端点用来支持一个接收数据的管道,同时由另外一个端点支持一个发送数据的管道。

在 USB 技术的体系结构中有四种基本的数据传送类型:

- (1) 控制信息传输:用于在设备接入时对其进行配置,也可用于其它目的。
- (2) 批量数据传输:在数据相对比较多和突发数据量较大时使用,在传输限制方面具有很宽的动态自由度。
- (3) 中断数据传输,这种方式与人类的反应特点很相似。
- (4) 同步或流实时数据传输:占用预先商量好的带宽,并且有预先商量好的发送时延。

任一给定的管道都必须能够支持上述的一种传输方式。在第 5 章中将详细地介绍数据流模型。

### 4.7.1 控制信息传输

控制数据用于在 USB 接入总线时对其进行配置。其它的驱动软件可以根据具体的应用来选择使用控制传输。这种数据传输不会丢失数据。

### 4.7.2 批量数据传输

典型的批量数据包括像使用打印机或扫描仪时所出现的大数据量的数据。这种批量数据是连续的。通过在硬件中实现差错检测功能,并且有选择地进行一定的硬件重试操作,可以在硬件层次上保证数据的可靠交换。而且,批量数据可以占有总线上所有可用的和其它传输类型未使用的带宽。

### 4.7.3 中断数据传输

由设备自发产生的数据传输是中断数据传输。这类数据传输可以由 USB 设备在任意时刻发起,而且 USB 总线以不低于设备说明的速率进行传输。

典型的中断数据包括事件指示、特性等,它们由一个或数个字节组成,例如来自一个指定设备的同步信号。尽管 USB 协议并不需一个明确的时钟速率,但数据交互会有 USB

协议必须支持的反应时间的限制。

#### 4.7.4 同步传输

同步数据在产生、传送和处理过程中是连续的和实时的。在稳定的同步数据发送和接收速率中包含了相应的时钟信息。为了保持定时关系，同步数据必须按照接收的速率进行传输。除了传输速率，同步数据也可能会对传送时延敏感。对同步管道而言，所需的带宽与相应功能设备的抽样特性有关。而时延要求则与每一端点的缓冲能力有关。

同步数据的典型例子是声音信号。如果这些数据流的传送速率不保持一致的话，由于缓冲区或帧结构溢出或欠载工作，数据流会出现假信号。即使数据以适当的速率进行传输，传送时延也会损害一些具有实时性的要求，例如电话会议。

可以保证对同步数据的适时传送，但是要以数据流中存在潜在的数据暂时丢失可能性为代价。换句话说，任何在传输中电气方面出现的错误都不可能用像重试这样的硬件机制来加以纠正。在实际中，USB（在多数情况下）的误比特率很小，而可以不予考虑。协议中允许在整个 USB 带宽中分出一部分专门用于 USB 同步数据流，从而保证了这些数据可以按所需求的速率进行传输。USB 同样可以支持要求时延很小的同步数据传输。

#### 4.7.5 分配 USB 带宽

USB 的带宽要在管道中进行分配。当一个管道建立后，USB 可以为一些管道分配带宽。而 USB 设备则要为数据提供一些缓存能力。这里假定需要带宽越大的 USB 设备可以提供更大的缓冲区。USB 体系结构的设计目标就是将需要缓冲操作来予以减小的硬件时延限制在几个毫秒之内。

USB 的总线带宽可以在许多不同的数据流内进行分配。这允许在 USB 总线上插入范围很广的一些 USB 设备。比如，USB 总线上可以包含从 1B + D 至 T1 这样速率的通信设备。而且，该总线可以同时支持动态范围的，具有不同的比特速率的设备。

另外 USB 带宽分配是需要予以限制的，即如果新分配的管道对已经存在的带宽或时延特性产生了影响，系统就会否决或阻塞对管道的进一步分配。如果一个管道被关闭了，那么为其分配的带宽就会被释放，并可以重新分配给别的管道。

USB 协议规范中对每一种传输类型如何访问总线都作出了相应规定。

### 4.8 USB 设备

USB 设备可以分为集线器、文本设备等几类。集线器指的是专门设计用来提供额外的 USB 接入点的一类 USB 设备（这一点在前面已经提到）。USB 设备必须携带用于自我识别和通常的配置操作的有关信息，而且在任何时候都要表现出与定义的 USB 设备状态相符的操作。

#### 4.8.1 设备特征

所有的 USB 设备都要通过一个唯一的 USB 地址对其进行访问。每一个 USB 设备

还要支持一个或多个端点,用以实现和主机之间的通信。而且所有的USB设备都必须支持一个特殊的端点——端点0,该端点是USB控制管道在USB设备一侧的接入点。

与端点0相对应的是用来完整描述该USB设备的信息。这些信息包括如下几类:

(1) 标准。这是一类对所有的USB设备都通用的定义,并且包含厂商标识,设备类型和电源管理等项目对设备、配置,接口和端点的描述携带了与该设备的配置有关的信息。详细的讨论请参见第7章。

(2) 类型。该信息的定义将根据USB设备的类型而有所不同。

(3) USB设备供应商。USB设备供应商可以自由地提供此处所需的信息。但是,本规范并未对其格式做出规定。

而且,每一个USB设备都携带了USB控制和状态信息。所有的USB设备都支持通过其USB控制管道的通用访问模式。

#### 4.8.2 设备描述

有两种主要的设备类型:集线器和功能设备。只有集线器具有提供额外的USB接入点的能力,而功能设备用于向主机提供附加的功能。

##### 1. 集线器

集线器是USB即插即用体系结构中的关键元件。图4-3给出了一个典型的集线器的示意图。从用户角度而言,集线器用于简化USB的连接,并以较低的费用和复杂性来提供其稳定性。顾名思义,集线器就是电线集中器。它使USB具有多个连接的特性。接入点在这里称为端口。每一个集线器都可以将一个接入点变成多个接入点。这种体系结构支持多个集线器的连接。

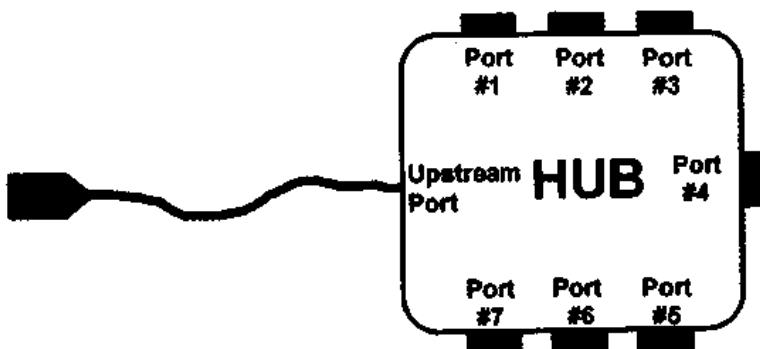


图4-3 一个典型的集线器

上行端口用以连接集线器和主机,另外的下行端口都可以用来连接集线器或功能设备。集线器可以发现下行端口上的设备插入和移出操作,并为下行设备分配电源。每一个下行端口都可以分别将其配置成高速或低速。集线器可以将低速端口与全速率信号隔离开来。

一个集线器包括集线器控制器和集线器中继器。中继器是一个位于上行和下行端口之间的、由协议控制的开关。它也具有复位和挂起/重新开始信令的硬件支持。控制器为主机之间的通信提供了接口寄存器。专用的集线器状态和控制命令允许主机配置一个集线器,并监视和控制其端口。

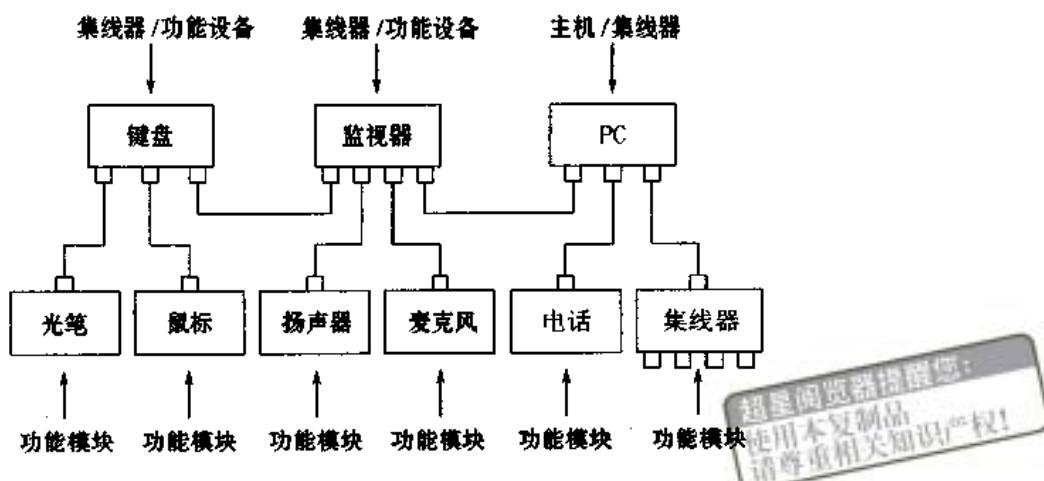


图 4-4 在台式计算机环境中的集线器

## 2. 功能设备

一个功能设备是指一个可以从 USB 总线上接收或发送数据或控制信息的 USB 设备。一个功能设备由一个独立的外围设备而实现, 它通过一根电缆接入集线器上的某一端口。但是, 一个物理组件也可以仅用一根 USB 电缆来连接多个功能设备和一个嵌入的集线器。这称为多功能设备。一个多功能设备对主机而言是一个永远都接着一个或多个 USB 设备的集线器。

每一个功能设备都包含了用来描述其能力和所需资源的配置信息。在使用一个功能设备之前, 必须由主机来对其进行配置。这种配置操作包括分配 USB 带宽和为该功能设备选择特定的配置选项。

下面给出一些功能设备的例子:

- (1) 定位设备: 如鼠标、图形输入板或光笔。
- (2) 人工输入设备(HID): 如键盘。
- (3) 输出设备: 如打印机。
- (4) 电话适配器: 如 ISDN。
- (5) 软盘驱动器: 如 USB ZIP。
- (6) 视频输入设备: 如 USB 数字相机。

## 4.9 USB 主机: 硬件和软件

USB 主机通过主控制器与 USB 设备交互。主机主要负责以下操作:

- (1) 检测 USB 设备的插入和移出。
- (2) 在主机和 USB 设备之间管理数据流。
- (3) 搜集状态信息和活动统计信息。
- (4) 为接入的 USB 设备提供数额受限的功率。

位于主机中的 USB 系统软件用来管理 USB 设备和基于主机的设备软件之间的交互。USB 系统软件和设备软件之间的交互有五个方面, 它们是:

- (1) 设备枚举和配置。

- (2) 同步数据传输。
- (3) 异步数据传输。
- (4) 电源管理。
- (5) 设备和总线管理信息。

只要条件允许,USB 软件就会使用主机系统接口来管理上述的交互操作。例如,如果主机系统使用高级电源管理(APM)来进行电源管理,USB 系统软件就会与 APM 消息广播设备相连,以便截获挂起和重新开始指示。

我们将在第 10 章对 USB 主机进行详细的介绍。

USB 体系结构包含了在主控制器驱动程序和 USB 驱动程序之间的可扩充性。因此,实现由多个主控制器和相应的主控制器驱动程序组成的应用成为可能。

# 第 5 章 USB 数据流模型

超星阅览器提醒您：  
使用本复制品  
请尊重相关知识产权！

本章阐述了对所有开发人员都有意义的基本问题。本章的内容是介绍在系统定义的信道和协议层的信息将在以后的章节中进行详细讨论。本章将为第 8 章的进一步扩展提供一个信息框架。在此我

本章具体包括以下内容：



总线构成；



USB 通信流；



设备端点；



管道；

USB 在主机和接入的 USB 设备之间连接的情形如图 5-1 所示，一个简单的情形如图 5-1 所示，一个复杂的连接情形如图 5-2 所示。图 5-1 中所示的要复杂一些。对于大多数应用来说，连接是完全对称的。对于某些应用来说，连接是不对称的。为了向终端用户提供适当的连接，必须满足一定的要求。

十分重要的概念和特性。USB协议规范是以分层方式加以阐述的,以此来简化对其的解释,并允许特定的USB设备开发人员注意与其产品相关的细节。

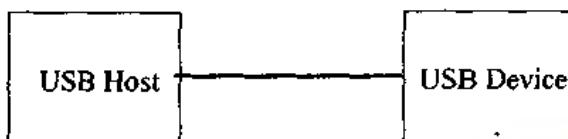


图 5-1 USB 主机/设备简单视图

图 5-2 从更深的层次对 USB 加以描述,说明了系统的不同分层。这些分层我们将在余下的部分对其进行详细的解释。具体地说,这里有四个值得关注的应用领域:

- (1) USB 物理设备:它是执行一些有用的终端用户功能的硬件。
- (2) 客户软件:在主机上执行的对应于一个 USB 设备的软件。
- (3) USB 系统软件:在一个特定的操作系统中用来支持 USB 的软件,通常由操作系统来提供,而与特定的 USB 设备或客户软件无关。
- (4) USB 主控制器(在主机一侧的总线接口):它指的是允许 USB 设备接入主机的硬件和软件。

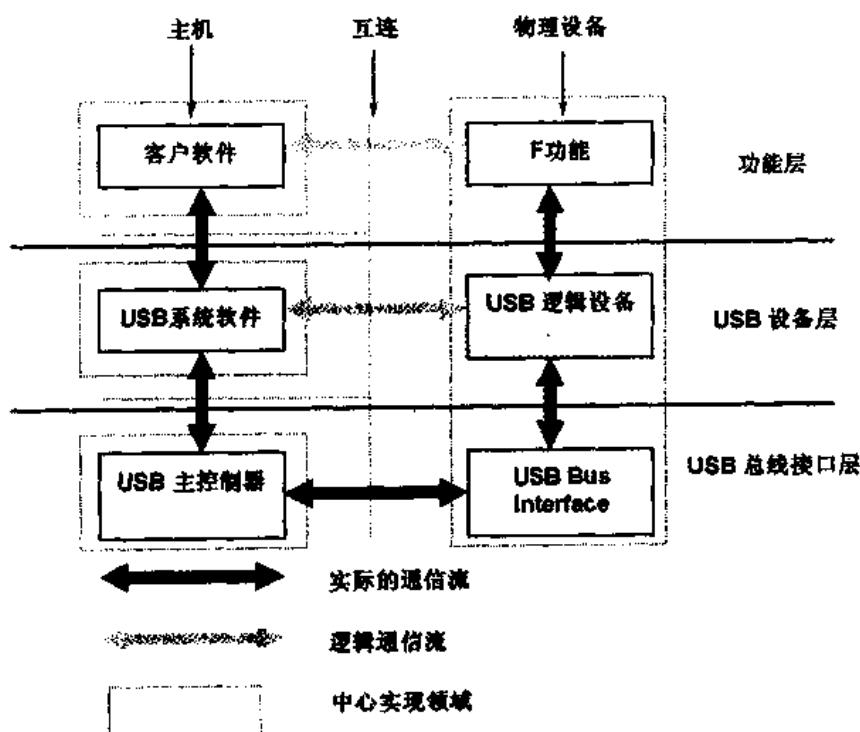


图 5-2 USB 实现领域

四种 USB 系统组件具有相同的权力和义务。下面我们将详细地介绍如何在一个功能模块和它的客户之间支持稳定、可靠的通信流。

正如图 5-2 所示,一个主机和一个 USB 设备之间的简单连接需要一些分层和实体之间的交互。USB 总线接口层为主机和设备提供了物理的/信令/分组连接。USB 设备层是 USB 系统软件中用于对一个 USB 设备执行通常的 USB 操作的部分。而功能层通过一个适当匹配的客户软件层向主机提供一些附加的功能。对于 USB 设备和功能层而

言,这两层都有其层间的逻辑通信,而这种逻辑通信实际上是通过 USB 总线接口层来完成其数据传输的。

下面这些概念对于描述和管理 USB 通信是很重要的:

- (1) 总线拓扑(Topology)构成:5.2 节介绍了 USB 中的主要的物理和逻辑元件,并指出它们之间的相互关系如何。
- (2) 通信流模型:5.3 ~ 5.8 节阐述了在 USB 主机和设备之间,通信数据是如何在 USB 总线中传送的,并定义了 4 种 USB 传输类型。
- (3) 对同步传输的特殊考虑:5.10 节说明了需要进行同步数据传输的 USB 设备的特性。

## 5.2 总线构成

USB 总线由四个主要部分构成:

- (1) 主机和设备。这是 USB 系统中的主要构件。
- (2) 物理构成:USB 元件是如何连接的。
- (3) 逻辑构成:不同 USB 元件的角色和责任,以及从主机和设备的角度出发,USB 所呈现的结构。
- (4) 客户软件:客户软件和 USB 设备上与其相对应的功能接口是如何看待对方的。

### 5.2.1 USB 主机

USB 主机的逻辑构成如图 5-3 所示:

- (1) USB 主控制器。
- (2) USB 整体系统软件(USB 驱动程序,主控制器驱动程序和主机软件)。
- (3) 客户。

USB 主机是 USB 中唯一的一个用于协调工作的实体。除了它的特殊物理位置,对于 USB 和与之相连的设备而言,主机还有一些特殊的责任。对 USB 的访问都由主机控制,只有当主机允许其访问时,一个 USB 设备才能获得对总线的访问权。另外,主机还负责监视整个 USB 的构成情况。

更详细的介绍请参考第 10 章。

### 5.2.2 USB 设备

一个 USB 物理设备的逻辑组成如图 5-4 所示:

- (1) USB 总线接口。
- (2) USB 逻辑设备。
- (3) 功能模块。

USB 物理设备向主机提供了附加的功能。但是这些由 USB 设备所提供的功能相差很大。而所有的 USB 逻辑设备都具有同样的接入主机的基本接口。这样主机就可以通过相同的方式管理不同的 USB 设备中与 USB 有关的问题。

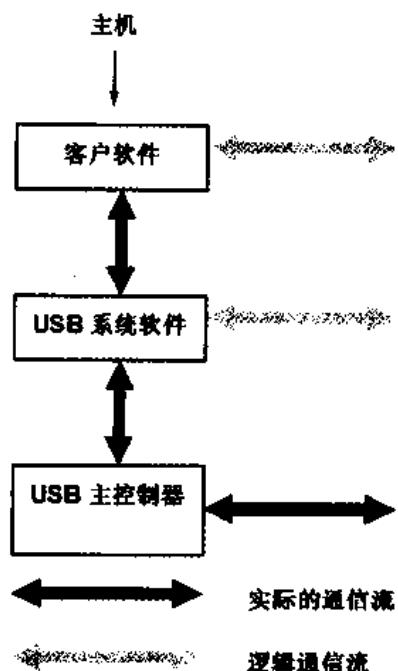


图 5-3 主机结构

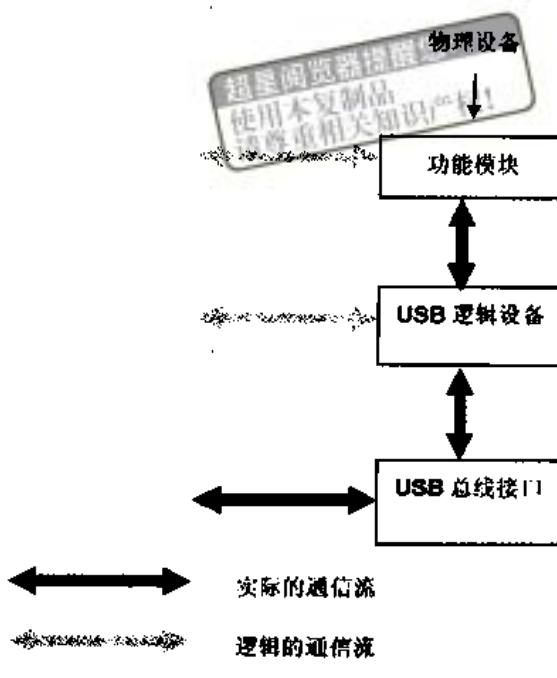


图 5-4 物理设备结构

为了协助主机识别和配置 USB 设备，每一个设备都要携带和报告与配置有关的信息。对于所有的逻辑设备而言，所报告的信息中有一些是通用的。而另外一些则由设备的功能决定。这种信息的详细格式根据设备所处的类型而有所不同。

详细的介绍请参阅第 7 章。

### 5.2.3 物理总线拓扑结构

USB 总线上的设备以星形的拓扑结构实现与主机的物理连接（如图 5-5 所示），USB 的接入点由集线器来提供。这种由集线器提供的额外的接入点称为端口。主机中

图 5-5 USB 物理总线拓扑结构

包含了一个嵌入的集线器,它称之为根集线器。通过根集线器,主机可以提供一个或多个接入点。为主机提供附加功能的设备称为功能模块。为了防止出现环行接入的情况,在 USB 中使用了分层的拓扑结构。这种配置结果具有树形结构。

多个功能模块可以套装在一起成为一个单一的物理设备。例如,一个键盘和一个鼠标可以组合在同一个包装之内。在该包装内,一个集线器上永远都接着不同的功能模块,而且由这一内部集线器实现与主机的互连。当多个功能模块和一个集线器组合在一个包装内时,我们称其为多功能(复合)设备。对主机而言,一个复合设备与一个连有多个功能设备的分离的集线器之间没有什么区别。

#### 5.2.4 逻辑总线拓扑结构

由于 USB 设备以星形拓扑结构与主机相连,所以主机与每一个逻辑设备之间的通信好像是跟直接在根集线器上相连的设备一样。图 5-6 给出了与图 5-5 中的物理拓扑结构相对应的逻辑拓扑结构。集线器也是一个逻辑设备,但在图 5-6 中并未给出。虽然大多数主机和逻辑设备的操作都使用这种逻辑视图,但是为了处理集线器的移出操作,主机仍然需要了解实际的物理构成情况。当拔下集线器后,从逻辑构图来讲,所有的与集线器相连的设备都要从主机上移出。

#### 5.2.5 客户软件

虽然 USB 的物理和逻辑拓扑结构都反映了总线的共享特性,但是客户软件要使用 USB 功能接口且仅对其对应的接口感兴趣。USB 功能设备的客户软件使用 USB 软件编程接口来使用它们的功能,这一点与其它总线技术(即 PCI、EISA、PCMCIA 等)利用存储器或 I/O 访问来使用其功能的方法不同。在操作过程中,客户软件必须不受可能接入 USB 的别的 USB 设备的影响。它允许设备和客户软件的设计者将焦点集中在硬件/软件交互的设计细节上。图 5-7 给出了相对图 5-6 所示的逻辑构图和对设备设计者而言的客户软件和 USB 功能设备之间的关系。

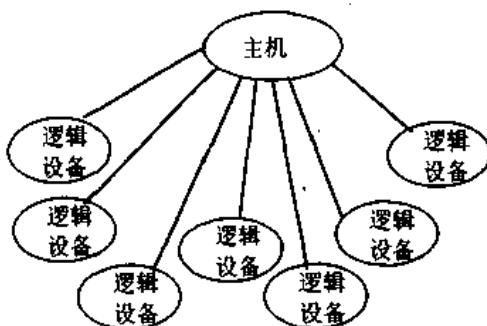


图 5-6 USB 总线逻辑拓扑结构

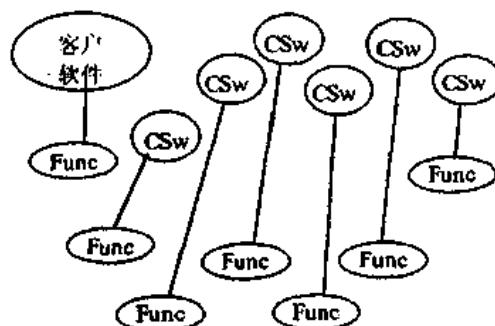


图 5-7 客户软件同功能模块的关系

Func—功能模块;CSw—客户软件。

### 5.3 USB 通信流

USB 为主机上的软件和客户的 USB 功能模块之间提供了通信服务。功能模块会对

通信流有不同的要求,需要不同的客户来实现相互作用。通过允许将不同的USB功能模块的不同通信流分离开来,USB提供了更好的整体总线利用率。每一个通信流都要使用某一总线访问来完成客户和功能模块之间的通信,并且终止于设备上的某一个端点。设备端点用于区别任意的通信流。

图5-8对图5-2进行了更详细的解释。图5-2中一个完整的实际通信流支持逻辑设备和功能层之间的通信流。而这些实际的通信流要经过若干个接口。第5章、第6章中给出了机械、电气和协议接口定义。第9章则阐述了允许从主机一侧的电缆来操纵USB设备的USB设备编程接口。第10章则描述了主机侧的两个软件接口。

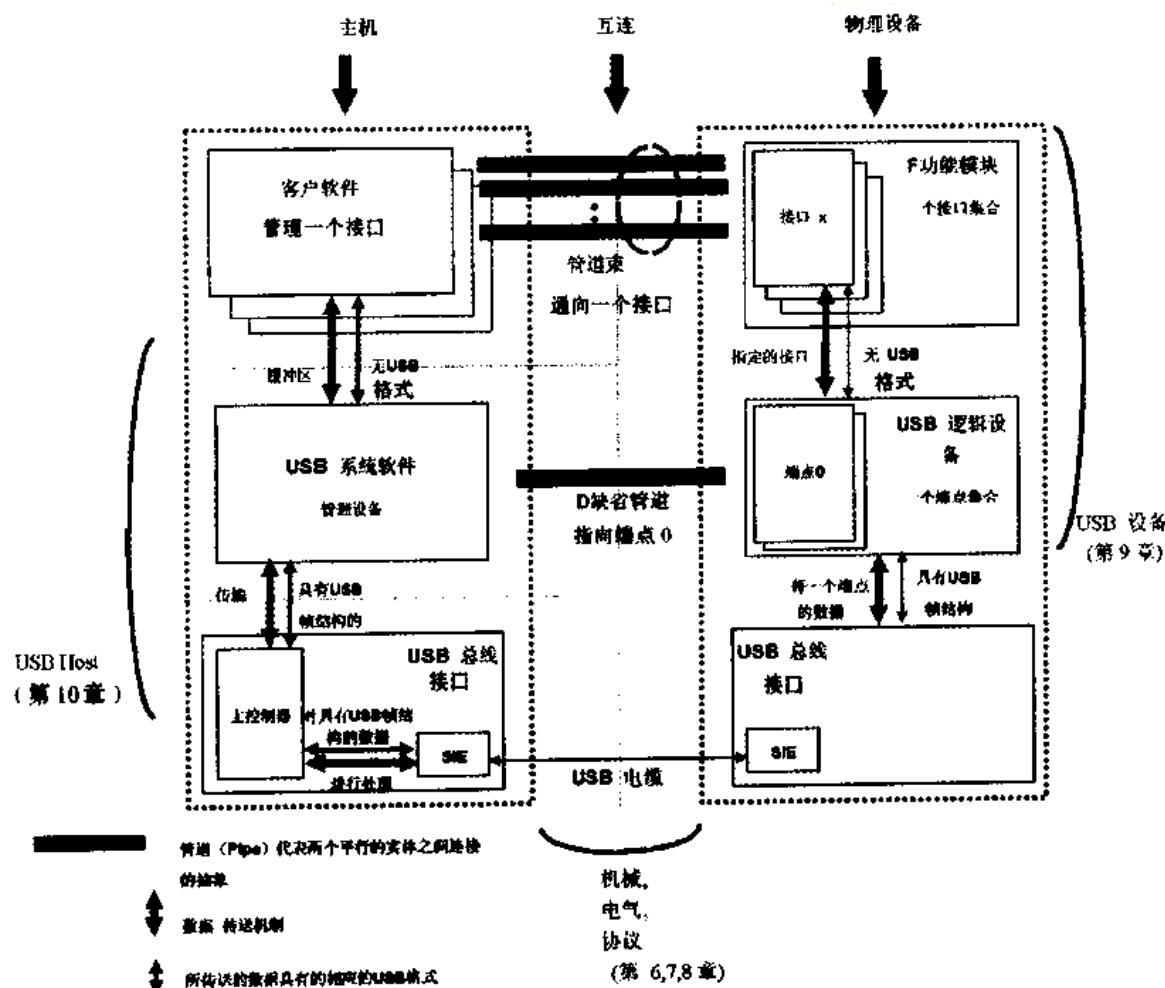


图5-8 USB主机/设备的详细视图

(1) 主控制器驱动程序:(HCD)它是位于USB主控制器和USB系统软件之间的软件接口。该接口可以实现一定范围的主控制器应用,而无须要求所有的主机软件都依靠于某一特殊的应用。无需对一个主控制器应用的专门知识,一个USB驱动程序就能支持不同的主控制器。一个主控制器的开发者要提供用来支持主控制器的HCD应用。

(2) USB驱动程序(USBD):它是位于USB系统软件和客户软件之间的接口。该接口为客户使用USB设备提供了方便。

一个USB逻辑设备对整个系统而言就是一个端点的集合。根据其使用的接口,端点

可以被分成不同的端点集。接口是对功能的认识。系统软件使用缺省管道(与端点 0 对应)来管理 USB 设备。而客户软件使用的是管道束(与一个端点集对应)。客户软件要求数据在主机上的缓冲区和 USB 设备上的端点之间,通过 USB 总线来移动。主控制器(或依赖于传送方向的 USB 设备)对数据进行分组,然后通过 USB 来传送。当使用总线访问来实现 USB 上的数据传输时,主控制器也会协同工作。

图 5-9 说明了在端口和主机一侧的存储器缓冲区之间的管道是如何承载通信流。下面将详细地介绍端点、管道和通信流。

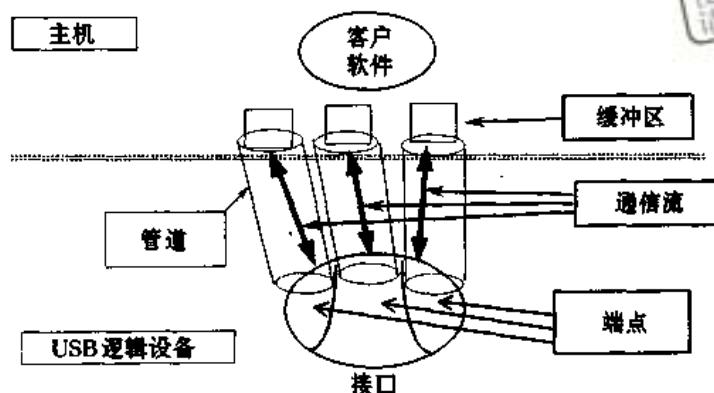


图 5-9 USB 通信流

主机上的软件通过一系列的通信流来与一个逻辑设备进行通信。这一系列的通信流由设备软件/硬件设计者来加以选择,从而使 USB 设备的通信要求与 USB 提供的传送特性更有效地匹配。

### 5.3.1 设备端点

端点是一个 USB 设备唯一可以确认的部分,它是主机和设备之间的通信流终点。每一个 USB 逻辑设备都包含了一个相当独立地进行操作的端点集合。软件只能通过一个或多个端点与一个 USB 设备通信。在设备接入时,每一个逻辑设备都有一个由系统分配的唯一的地址。而一个设备上的任一个端点都有一个由设备而定的(设计时)唯一的标识和端点号。利用设备的地址和端点号就可以唯一地指定任一个端点。

端点可以决定端点和客户软件之间通信所需要的传输服务类型。一个端点由以下内容来描述:

- (1) 总线访问频率/延时要求。
- (2) 带宽要求。
- (3) 端点号。
- (4) 差错控制要求。
- (5) 端点可以接收或传递的最大分组。
- (6) 端点的传送类型。
- (7) 对于同步传送方式而言,还要包括端点和主机之间的数据传送方向。

在对其进行配置之前,端点处于一种不确定的状态。所以只有在对其进行了配置工作之后,主机才能访问某个端点。

### 1. 端点0要求

所有USB设备都要拥有端点0，该端点用于对一个逻辑设备进行初始化和一般的操作（即对一个逻辑设备进行配置）。端点0提供了对设备配置信息的访问权，通过它还允许访问一般的USB状态和控制操作。端点0支持5.5节中所定义的控制传输，并且它总是在设备一经接入和上电时就进行配置。

### 2. 非0端点要求

由于具体应用的需要，功能设备还具有别的端点。除了端点0之外，低速功能设备有两个端点可供选择。而对于全速率设备来说，它的附加的端点数仅受到协议的限制，即最多可有16个输入端点和16个输出端点。

一个端点只在对其进行配置之后才可以使用。包括端点0在内的所有端点，都作为设备配置过程中的普通部分来对其进行配置。

### 5.3.2 管道

一个USB管道是设备上的一个端点和主机上的软件的联合体。管道表示经过一个存储器缓冲区和一个设备上的端点，可以在主机上的软件之间传送数据的能力。有以下两种不同的非斥的管道通信类型：

- (1) 流管道：数据通过管道时不具有确定的USB定义的结构。
- (2) 消息管道：数据通过管道时具有USB定义的某种结构。

USB并不对通过一个管道传递的数据内容进行翻译，即使是消息管道要求根据USB的规定对数据进行打包，USB也不会翻译这些数据的内容。

另外，管道还需要确定以下三个参数：

- (1) 对USB总线访问和带宽使用的声明。
- (2) 传输类型。
- (3) 所对应的端点的特征，如方向和最大数据负载尺寸等。其中数据负载指的是总线操作中，一个数据分组的数据域中所携带的分组。

对一个USB设备进行配置之后，就会形成管道。由于一个USB设备上电后总要对端点0进行配置，所以端点0总是拥有一个管道。该管道称为缺省管道。系统软件利用该管道来确定设备识别和配置要求，并对设备进行配置。当设备配置完毕之后，该设备的专用软件也可以使用缺省管道。但是USB系统软件却也保留对缺省管道的“所有权”，并且由它来协调另外的客户软件对该管道的使用。

一个客户软件通常是通过向一个管道发出IRP(I/O请求分组)来要求传送数据，发出IRP之后客户软件就等待或得到指示。有关IRP定义的细节由具体的操作系统而定。这里仅指一个客户软件要求它(主机)和设备的端点之间以适当的方向进行数据传输时，所发出的可以视为其一样的请求。如果需要的话，一个客户软件可以使管道返回所有的未处理的请求。

当与一个IRP相对应的总线操作完成时，无论其成功或是由于出错而结束，这个客户软件都会被告知IRP已知处理完了。

如果一个管道没有未处理的IRP或有正在进行处理的IRP，那么该管道就处于空闲状态，主控制器不会对该管道进行任何操作，即这一管道的端点没有任何的总线处理指向。

它。只有当这个管道有未处理的 IRP 时,该管道中才会有总线操作。

如果一个非同步管道遇到在 STALL 条件或一个 IRP 的任意分组中出现 3 个总线错误,那么这个 IRP 就会被重试/中止,而所有未经处理的 IRP 也会被重试。并且直至客户软件从这个状况中恢复后,且利用一个 USBD 调用来自回应 STALL 或错误条件之后,所有未处理的 IRP 都会被重试而且不会再接收 IRP。一个适当的状态会向客户软件通告与 STALL 对应的特定的 IRP 错误结果。

为了通过总线来传送数据,一个 IRP 可能需要多个数据负载。对于一个需要多个数据载荷的 IRP 来说,其分组的数据域应该具有最大的尺寸,直到最后一个数据载荷中包含该 IRP 中的余下的部分。对于拥有输入的短分组(即数据小于数据载荷的最大值)的 IRP 而言,其数据不能完全填满该 IRP 的缓冲区,我们可以根据用户的要求而采用两种方法处理。

(1) 一个用户可以要求一个 IRP 具有不同大小的数据量。在这种情况下,一个不能填满一个 IRP 的数据缓冲区的短分组可以被简单地用作指示数据单元结束定界标志。这时该 IRP 就应该被消除,而主控制器将转而处理下一个 IRP。

(2) 一个用户可以要求传送特定数量的数据。在这种情况下,一个不能填满一个 IRP 的数据缓冲区的分组是差错的指示。此时,该 IRP 会被消除,该管道也会被禁用,并且与这个管道相关的任何未经处理的 IRP 都会被消除。

由于主控制器必须在两种情况下进行不同的操作,而且它不会知道对于一个给定的 IRP 应当采用哪种处理方式,所以有必要告诉每一个 IRP 用户所要求的操作方式。

通过发出 NAK 作为回应,一个端点可以告诉主机它正忙。NAK 并不是作为向客户软件返回一个 IRP 时的清除条件来使用的。在处理某一个 IRP 的过程中,会遇到任意多个 NAK。一个操作中返回 NAK 并不会产生错误,也不会将其作为上述三种错误之一而进行计数。

### 1. 流管道

流管道在传送数据时对数据分组没有结构要求。数据流从一端进入总线,并以同样的顺序从另一端流出。而且在通信流中流管道通常不是双向的。

通过一个流管道进行数据传输时,我们总是希望与一个 USB 认为是单个的客户进行交互。USB 系统软件不必在可能使用同一个流管道的多个客户之间提供同步机制。流管道中的数据以顺序的方式经过管道,即先进先出方式。

通往某个设备的一个流管道,在适当的方向一定要与一个唯一的设备端点号相对应(即与协议层所指定的一个 IN 或 OUT 令牌所对应)。用于反方向的设备端点号也可由通向该设备的另外的某个流管道使用。

流管道可以支持批量、同步和中断传输类型。

### 2. 消息管道

消息管道用与流管道完全不同的方式来与端点进行沟通。首先,主机向该 USB 设备发出一个请求;该请求后面是适当方向的数据传输;最后,在后来的某一时刻端点会返回一个状态作为响应。为了适应请求/数据/状态模式,消息管道要求通信流具有一定的结构,这样命令就可以被可靠地识别和通信。虽然通信流多数情况下是一个方向的,但消息管道却允许双向的数据流存在。与端点 0 对应的管道,即缺省管道,总是一个消息管道。

USB系统软件确保多个请求不会同时送到一个端点。在某一时刻一个端点仅需服务于单个消息请求。主机上的多个客户软件可以通过缺省管道发出请求,但是这些请求却是按照先进先出的顺序送至某个端点。根据其对主机操作的响应能力,在数据和状态阶段一个端点可以对信息流进行控制。

通常情况下,当一个端点对当前的消息的处理没有结束时,该端点不会接到由主机发来的下一个分组。但是,有一些错误情况可以使主机禁止该消息传输,并且该端点会提前收到一个新的消息传输。从软件对一个消息管道控制角度来讲,一个IRP内某一个部分的错误会废除当前的IRP和所有排队的IRP。利用一个适当的差错指示,可以告知发出IRP请求的客户软件IRP操作已经完成了。

一个通向一个设备的消息管道要求在两个方向都具有唯一的一个端点号(IN或OUT令牌)。USB不允许在任一方向,一个消息管道与不同的端点号相对应。消息管道支持控制传输类型。

## 5.4 传输类型

USB通过位于同主机上的客户软件相对应的内存缓冲区和USB设备上的端点之间的管道来传送数据。在消息管道中传输的数据具有USB定义的结构,但是USB也允许在消息数据负载中传送某个具有特定设备结构的数据。USB规定对于任一种管道类型(流或消息)数据经过总线时都要进行分组,但是最终对一个总线操作的数据负载中传送的数据分析和解释工作,则要由客户软件和使用该管道的功能模块来完成。但是,USB提供了不同的传输类型,这些类型经过优化从而与使用管道的客户软件和功能设备的要求更接近。一个IRP使用一个或多个总线操作来实现在客户软件和功能模块之间传送信息。

每一个传送类型都决定了数据流所具有的不同的特性:

- (1) USB规定的数据结构。
- (2) 通信流方向。
- (3) 分组大小限制。
- (4) 总线访问限制。
- (5) 所要求的数据顺序。

一个USB设备的设计者要为设备端点选择其所具有的性能。如果为一个端点建立了一个管道,那么大多数的管道传送特性就确定了,并且在该管道的生存期中保持不变。我们将分别介绍每一种传送类型的可以修改的传送特性。

USB定义了以下四种传输类型:

- (1) **控制传输**:主要用于命令/状态操作,由主机软件发起的请求/响应通信过程,具有突发性,非周期的特点。
- (2) **同步传输**:主要用于主机和设备与时间有关的信息传输,具有周期性、连续性的特点。这种传输类型保留了数据中时间压缩的概念。但是,它并不意味着这一类数据传送都是实时的。
- (3) **中断传输**:主要用于向主机通知设备的服务请求,它是由设备发起的通信,具有

数据量小,非周期,低频率,延时一定等特点。

(4) 批量传输:主要用于那些可以利用任何可用的带宽进行传送,或可以延迟到有可以利用的带宽时再进行传送的数据,它具有非周期和突发性强的特点。

下面几节中,我们将详细讨论这四种传输类型。对这四种类型有一个深入的了解,是学习和利用 USB 技术的基础,因此笔者希望读者认真对待以下内容。

## 5.5 控制传输

控制传输允许对一个设备的不同部分进行访问,它用于支持客户软件及其功能模块之间的配置/命令/状态等类型的通信流。一个控制传输由以下几部分构成:总线建立操作,负责将请求信息由主机送至功能模块;零个或多个数据处理操作,传送方向由建立操作指定;状态处理操作,用于功能模块向主机返回状态信息。当一个端点成功地完成了请求操作的处理时,状态处理操作就返回“成功”的状态。在第 6 章中我们将详细讨论用于完成一个控制传输所需的分组、总线操作和操作顺序等细节。第 7 章中将详细讨论 USB 的命令代码。

任何一个 USB 设备都必须支持与一个控制传输类型相对应的端点 0。该端点被 USB 系统软件用于一个控制管道。而控制管道则提供了对 USB 设备配置、状态和控制信息的访问。根据其应用的需要,一个功能模块也可以为更多的控制管道提供相应的端点。

USB 设备结构规定了处理一个设备状态的有关标准、设备类型或某个厂商的要求,并规定了设备上包含有不同信息的描述等。控制传输提供了访问设备描述符的传送机制,并可以请求一个设备来处理其操作。

控制传输只能由消息管道来实现。因此,使用控制传输的数据流必须遵照 5.5.1 节中所定义的数据结构。

USB 系统将尽最大努力来支持主机和设备之间的控制传输。一个功能模块及其软件不能请求某个访问频率或带宽。USB 系统软件将对一个设备所要求的控制传输的总线访问和带宽加以限制。这些限制将在 5.5.3 和 5.5.4 节中讲述。

### 5.5.1 数据格式

建立分组具有一个 USB 确立的结构,它包括了实现主机和一个设备之间通信所需的小指令集。该结构允许厂商对某些设备指令予以扩展。在建立操作后的数据处理操作则没有确定的 USB 结构,而状态处理操作则具有一定的结构。

利用消息管道的双向通信流,可以实现控制传输。

### 5.5.2 分组尺寸限制

与一个控制传送相对应的端点会说明,该端点可以从总线上接收或向总线传送的最大数据负载。USB 规定,对全速率设备而言所允许的最大控制数据负载仅可以是 8、16、32 或 64 字节,而低速设备则限制为 8 个字节。这一最大值仅对跟随着建立分组后的数

据分组的数据域有意义,即所说明的尺寸只对第 3 章中所定义的分组的数据而言,并不包括其它协议所要求的信息。一个建立分组总是 8 字节长。而一个控制端点也总是具有最大的数据负载。

在复位之后,所有的控制端点都必须支持最大为 8 字节的控制数据负载。当然一个端点也可以被设计成可以支持更大的数据负载。这种端点需要在其配置信息中报告它的最大数据负载值。USB 并不要求传送的数据负载具有最大值,即如果一个数据负载小于所允许的最大值,不需将它填充到具有最大的尺寸。

对于全速率控制端点,所有的主控制器都要能支持最大值为 8、16、32 或 64 字节的数据负载。而对于低速率控制端点,主控制器仅需支持最大值为 8 字节的数据负载。而且主控制器不必支持那些最大数据负载大于或小于 8 字节的分组。

在配置过程中,USB 系统软件将了解到该端点的最大数据负载尺寸,并且保证不会向这一端点发送超过其所能支持尺寸的数据负载。主机所使用的最大数据负载尺寸至少应为 8 个字节。

一个端点所传送的数据必须小于或等于该端点 MaxPacketSize 中所规定的值(参见第 7 章)。当一个控制传输包含了超过当前所使用的最大尺寸的数据负载的数据时,除了最后一个数据负载中包含剩余数据之外,所有的数据负载都具有最大值。如果一个端点想传送的数据小于客户软件的预计值,那么主控制器将收到一个数据负载小于最大值的数据分组。这种分组使主控制器进入状态处理操作,而不是下一个数据处理操作,或像 5.3.2 节所描述的那样禁止这个管道。如果接收到的数据负载大于所预期的值,那么该控制传输的 IRP 被中止/撤消,并且该管道会废除以后的 IRP,直到这种状态得到纠正和应答。

### 5.5.3 总线访问限制

全速和低速 USB 设备都可以使用控制传输类型。

一个端点不能指出一个控制管道所要求的总线访问频率。USB 将负责平衡所有的控制管道和某些挂起的 IRP(为了在客户软件和功能模块之间提供最大可能的数据传送能力)对总线的访问请求。

USB 要求在每一帧中都保留一部分专供控制传输使用:

(1) 如果所进行的控制传输(依应用而定)所占用的时间小于帧周期的 10%,则剩余时间可以用来支持批量传输。

(2) 如果一个正在尝试的控制传输需要撤消,可在当前的帧中或以后的帧中进行,即撤消操作不必在同一帧中进行。

(3) 如果控制传输超过了保留时间,而同时总线上还有同步传输或中断传输未使用的可用帧时间,那么主控制器可以进行另外的控制传输。

(4) 如果等待处理的控制传输大大超过了可用的帧时间,将适当地选择控制传输通过总线。

(5) 如果多个端点都有等待处理的控制传输,根据一个与主控制器实际应用有关的公平访问原则,将选择不同的端点所对应的控制传输通过总线。

(6) 对一个经常被撤消的控制传输的操作,不应该让其消耗不公平的帧时间份额。

以上这些要求可以保证主机和设备之间的控制流可以尽量正常地通过 USB 总线。

一个系统中的所有待处理的控制传输都可以竞争使用同一段可用的总线时间。正因为这样,对于某个具体的端点所对应的控制传输而言,USB 系统软件可以自行决定其可用的总线时间。一个端点及其客户软件不能为控制传输假定一个服务速率。当别的 USB 设备插入、移出系统,或另外的设备端点需要控制传输时,用于一个客户软件及其端点的总线时间会发生改变。

总线频率和帧的定排对可以成功进行的控制传输的最大值提出了限制,对于任一 USB 系统,一帧中所包含的数据负载应小于 29 个全速率 8 字节数据负载或 4 个低速率 8 字节数据负载。表 5-1 中列出了有关全速率控制传输的不同尺寸和一帧中可能有的最大传输数的信息。该表是在假定状态处理阶段数据长度为 0 和只有一个数据处理阶段的条件下得到的,它说明了小于或等于允许的最大数据负载大小的两个数据负载的可能的承载能力。

表 5-1 全速率控制传输限制

协议开销(字节)					
45 (9 - syncs, 9 - pids, 6 - EP + CRC, 6 - CRC, 8 - Setup, 数据 7 - byte 分组间延迟(EOP, 等))					
数据负载	最大带宽/字节/秒	每个传输所占地帧带宽	最大传输数	剩余字节数	有用数据/字节/帧
1	32000	3%	32	28	32
2	62000	3%	31	43	62
4	120000	3%	30	30	120
8	224000	4%	28	16	224
16	384000	4%	24	36	384
32	608000	5%	19	37	608
64	832000	7%	13	83	832
最大值	1500000				1500

控制传输要保留 10% 的帧周期,这意味着在一个已经完全分配了总线时间的系统中,所有的全速率控制传输都要在每一帧中竞争名义上的三个控制传输使用权。由于 USB 系统要利用控制传输来进行配置操作,而且别的客户软件也会请求另外的控制传输,所以不能指望某一个客户软件及其功能模块会为了其自己的控制操作而使用全部的带宽。主控制器也可以自由地决定某个控制传输的独立的总线操作是在一帧中或是跨越几个帧而通过总线。一个端点也会注意到在同一帧内或几个不连续的帧之间,对一个控制传输进行的所有总线操作。最后根据实际的应用需要,一个主控制器可以不提供理论上每帧中的最大控制传输数。

全速率和低速率控制传输都可以竞争使用同一段可以利用的帧时间。不过低速率控制传输所用的传送时间要长一些。表 5-2 列出了不同大小的低速率分组和一帧中可能的最大分组数的有关信息。对于这两种速率而言,由于一个控制传输要由几个分组来组成,因此这些分组可以扩展到几个帧中进行传输,从而增加了通过几个帧传输所需的总线时间。

表5-2 低速率控制传输限制

协议开销(字节,byte)					
数据负载	最大带宽(约为)	每一次传输所占带宽	最大传输数	剩余字节数	有用数据/字节/帧
1	3000	25%	3	46	3
2	6000	26%	3	43	6
4	12000	27%	3	37	12
8	24000	29%	3	25	24
最大值	187500				187

如果在先前发起的控制传输结束之前,一个端点收到了一个建立处理操作,设备必须禁止当前的传输/操作,并对新的控制建立操作加以控制。一般来说,在前一个控制传输结束之前,不应该再发出建立处理操作。但是,如果由于总线上的错误一个传输被禁止了,从端点的角度看,主机会提前送出下一个建立处理操作。

在遇到 STALL 条件或主机检测到一个错误之后,允许一个控制端点通过接收下一个建立 PID 而得到恢复。也就是对于控制端点而言,恢复动作不需要另外的某个管道参与。但是对于某些应用来说,却需要这样做。对于缺省管道(端点 0),如果没有收到下一建立 PID,则最终需要一个设备复位(通过 USBD 进行)来清除 STALL 或错误状态。

USB 提供了突发错误检测机制,以及对在控制传输过程中出现差错而进行的恢复/重传机制。根据他们在控制传输中所处的阶段,发射器和接收机可以保持同步,并尽最小的努力来恢复。接收器通过分组中的数据重传指示符,可以发出对数据和状态分组进行的重传操作。通过在分组中返回的交互(握手)信息,发射器可以断定与之对应的接收器已经成功地接收到了所发送的分组。除了控制建立分组之外,协议允许对最初的分组和一个重传的分组加以区别。建立分组会因传输错误而重传,但是建立分组不能指出这个分组是最初的分组还是一个重传的分组。

## 5.6 同步传输

在一个非 USB 环境中,同步传输通常意味着具有恒定速率和容许一定差错的传输。而在 USB 环境中,需要一个同步传输类型具有下列的特性:

- (1) 在一定的时延条件下,保证对 USB 带宽的访问。
- (2) 只要向管道提供了数据,就应当保证通过该管道的数据所要求的恒定的数据传输速率。
- (3) 一旦由于错误而使传送失败,不应利用“重试”来传送数据。

由于 USB 同步传送类型是设计用来支持同步信源和信宿的,它不要求使用这类传输的软件为了利用该传输类型就一定要是同步的。5.10 节将详细讨论如何在 USB 上控制同步数据。



### 5.6.1 数据格式和方向

对同步管道, USB 未对其通信流规定结构。

一个同步管道是一个流管道, 因此通常它是单向的。端点描述符将用来区别一个给定的管道通信流是流向还是流出主机的。如果一个设备需要双向的同步通信流, 就需要使用两个同步管道, 一个方向一个。

### 5.6.2 分组尺寸限制

在一个同步管道的配置操作中, 端点会说明它可以发送/接收的最大数据负载。USB 系统软件正是利用配置操作中的这一信息来保证在每一帧中有足够的总线时间来容纳最大的数据负载。如果有足够的总线时间可用于最大数据负载, 配置操作就会建立, 如果没有, 就不会进行配置操作。USB 系统软件不会像对待控制管道那样, 为一个同步管道调整其最大数据负载的大小。在一个给定的 USB 子系统配置过程中, 一个同步管道只有可以被支持或不能被支持的区别。

USB 将每一个同步管道的最大数据负载限制为 1023 字节。表 5-3 列出了不同大小的同步处理操作和一帧中可能存在的最大操作数的有关信息。

表 5-3 同步操作限制

协议开销(字节, byte)					
9 (2 - syncs, 2 - pids, 2 - EP + CRC, 2 - CRC, 1 个字节的分组间延迟)					
数据负载	最大带宽	每一次传输所占地帧带宽	最大传输数	剩余字节数	有用数据/字节/帧
1	150000	1%	150	0	150
2	272000	1%	136	4	272
4	460000	1%	115	5	460
8	704000	1%	88	4	704
16	960000	2%	60	0	960
32	1152000	3%	36	24	1152
64	1280000	5%	20	40	1280
128	1280000	9%	10	130	1280
256	1280000	18%	5	175	1280
512	1024000	35%	2	458	1024
1023	1023000	69%	1	468	1023
最大值	1500000				1500

任何一个对于同步管道的处理操作都不需要一定的具有为端点确定的最大尺寸。数据负载的大小由发射器来决定(客户软件或功能模块), 并可以根据操作的需要而加以改变。一个端点可以使用可选的 USB 标准样值头来指示在这一样值流中该分组是从何处开始的。它还允许接收器从由于差错而造成的分组丢失中恢复过来。USB 保证任何主控制器声明的尺寸都可以在总线上传递。一个数据负载的真实大小要由数据发射器来决定, 并且可能会小于前面协商好的最大值。总线错误会改变一个接收器所收到数据的真实尺寸。但是, 这些错误却可以通过数据的 CRC 校验或接收器对任一操作的预计尺寸的认识而加以检测出来。



### 5.6.3 总线访问限制

同步传输只能应用于全速率设备。

USB 要求任意帧中为周期性传输(同步和中断传输)所分配的时间不超过 90%。

用于一个同步管道的端点不具有有关总线访问频率的信息。通常,所有的同步管道在每一帧中(即每 1ms)仅传送一个数据分组。总线上的错误或操作系统在调度客户软件时出现的延时都会导致一帧中没有分组进行传输。在这种情况下,一个错误指示将会作为状态返回到客户软件。通过跟踪 SOF 令牌并注意到在一个同步端点的两个 SOF 令牌之间没有插入数据分组,设备也可以发现这种状态。

总线频率和帧时序限制了一帧中能成功进行的同步处理操作的最大值,对任何 USB 系统该值都小于 151 个全速率 1 字节数据负载。最后,根据不同应用的需要,主控制器可能不会使一帧中的同步处理操作具有理论上的最大值。

### 5.6.4 数据顺序

同步传输不支持对总线上的差错做出数据重传响应。一个接收器可以判断是否发生了一个传输错误,低层 USB 协议不允许向一个同步传输类型的发送器返回交互信息。通常情况下,将返回一个分组是否重新接收的握手信息。对同步传输而言,及时性要比准确性/重传更重要。而且在总线上预计差错率较低的条件下,USB 协议已经得到了优化,以保证传输可以成功地进行。同步接收器可以知道在一帧中它是否漏掉了数据。而且,接收器可以确定有多少数据丢失了。5.10 节将详细地讨论这些 USB 机制。

由于没有用于报告 STALL 条件的握手信息,所以用于进行同步传输的某个端点不会被禁止。主机和客户软件不可能遇到这种情况,出现的错误会作为与一个同步传输 IRP 相对应的状态而被报告,但同步管道不会在错误的情况下被禁止。如果检测到一个错误,主机将继续处理下一个传输帧中的数据。由于同步处理操作协议不允许对每个处理操作交换握手信息,所以差错检测功能很有限。

## 5.7 中断传输

中断传输是设计用来支持那些偶然需要对少量数据进行通信,但服务时间却受限的设备。申请一个中断传输类型的管道要提供下列要求:

- (1) 该管道要求保证的最大服务时间。
- (2) 如果由于总线上的错误而出现偶然的传输失败,在下一个阶段将进行重新传递的尝试。

### 5.7.1 数据格式

对于中断管道,USB 未对其通信流规定数据结构。



### 5.7.2 方向

一个中断管道是一个流管道,因此它总是单向的。而且,一个中断管道仅用于向主机输入数据。USB 不支持输出中断管道。

### 5.7.3 分组尺寸限制

用于实现一个中断管道的某个端点要说明它将传输的最大数据负载。对全速率方式,所允许的最大中断数据负载尺寸为 64 字节或更少。而低速设备则限制为 8 个字节或更少。这一最大值将应用于数据分组,也就是说这个尺寸限制只对分组中的数据域有限制(参见第 6 章),并不包括其它协议所要求的信息。USB 并不要求数据分组一定要具有最大的尺寸,即如果一个数据分组小于最大值,我们无须对其进行填充以使其达到最大值。

对全速率中断端点,要求所有的主控制器都可以支持最大为 64 字节的数据负载;对低速率端点,要求所有主控制器能够支持 8 字节或更小的数据负载。USB 不要求主控制器支持更大的数据负载。

在设备配置操作中,USB 系统软件将决定一个中断管道可以使用的最大数据负载尺寸。在一个设备配置期中,该尺寸保证不变。USB 软件利用配置操作中决定的最大数据负载尺寸来保证在为其分配的时间段中有足够的总线时间来容纳具有最大尺寸的数据负载。如果总线时间充足,将建立该管道;如果不充足,就不会建立这一管道。USB 软件不会像对待控制管道那样,为一个中断管道调整其可用的总线时间。在某个 USB 子系统配置过程中,一个中断管道要么可以支持,要么不可以支持。但是,一个数据负载的实际大小仍然要由数据发送器决定,并且可能小于最大值。

一个端点所传输的数据负载其数据域必须小于或等于该端点 MaxPacketSize(参见第 7 章)中所规定的值。一个设备可以通过中断管道传送大于 MaxPacketSize 中规定值的数据。一个客户软件可以通过用于要求多个总线处理操作的中断传输的一个 IRP 来接收数据,而不需要每个处理操作中都有一个 IRP 完整指示。通过声明一个可以存储所需大小数据的缓冲区,我们可以达到这一目的。缓冲区的大小是 MaxPacketSize 规定值的倍数再加上一定的剩余量。除了最后一个处理操作,一个端点必须以 MaxPacketSize 所规定的值来传输任一操作,而最后一个处理操作将传输余下的数据。在为该管道所建立的时间段内,将有多个数据处操作通过总线进行传输。

如果一个中断传输所包含的数据大于当前建立的最大数据负载尺寸,除了最后一个数据负载之外,所有的数据负载都要具有最大的尺寸,而最后一个数据负载用于传输剩余的数据。如果一个端点要传输的数据量小于客户软件的预计值,那么主控制器将收到一个数据负载小于最大值的数据。这一分组将使主机废除当前的 IRP 而转向下一个 IRP,或像 5.3.2 节所描述的那样禁止该管道。如果接收到的数据负载大于所预期的值,那么该中断 IRP 就会被禁止/撤消,并且该管道将禁止后续的 IRP,直到这种状态得到纠正和承认。

### 5.7.4 总线访问限制

全速率和低速率设备都可以使用中断传输。

USB要求任一帧中为周期性传输(同步和中断传输)所分配的时间不超过90%。

总线频率和帧时序限制了一帧中能成功进行的中断处理操作的最大值,对于任何USB系统,该值都不应该小于108个全速率1字节数据负载或14个低速率1字节数据负载。最后,根据不同应用的需要,主控制器可能不会使每一帧中的中断处理操作都具有理论的最大值。

表5-4给出了一帧中,不同大小的全速率中断处理操作和可能进行的最大处理操作数的有关信息。表5-5则列出了低速率中断处理操作的类似信息。

表5-4 全速率中断处理操作的限制

13 (3 - syncs, 3 - pids, 2 - EP + CRC, 2 - CRC, 3个字节分组间延迟)					
协议开销(字节,byte)	最大带宽	每一次传输所占用的帧带宽	最大传输数	剩余字节数	有用数据/字节/帧
1	107000	1%	107	2	107
2	200000	1%	100	0	200
4	352000	1%	88	4	352
8	568000	1%	71	9	568
16	816000	2%	51	21	816
32	1056000	3%	33	15	1056
64	1216000	5%	19	37	1216
最大值	1500000				1500

用于实现一个中断管道的端点将说明其所需要的总线访问周期。一个全速率端点所要求的周期可以从1ms至255ms,低速率端点只能从10ms至255ms。在配置操作中USB软件将利用该信息来决定它所能维持的周期。系统所提供的时间可能小于设备所要求的时间,并可以达到USB所规定的最短访问周期。客户软件和设备只能依赖于这样的事实,那就是主机将保证一个端点的两个无错的处理操作之间的持续时间,不长于所要求的访问周期。注意,总线上的错误会阻碍一个中断处理操作成功地通过总线进行传递,并因此而超过所要求的访问周期。任何两个处理操作尝试之间的时间都会不同;即使在无错的情况下它永远也不会超过所要求的访问周期。另外,只有当客户软件有未处理的中断传输IRP时,该端点才会进行处理操作。如果进行一个中断传输的总线时间已经到了,而此时却没有等待处理的IRP,那么该时刻端点就不会得到进行数据传输的机会。一旦对一个IRP发出了请求,就会在下一个分配的时间段中传输其数据。

表5-5 低速率中断处理操作限制

13					
协议开销(字节,byte)	最大带宽(约为)	每一次传输所占用的帧带宽	最大传输数	剩余字节数	有用数据/字节/帧
1	13000	7%	13	5	13
2	24000	8%	12	7	24
4	44000	9%	11	0	44
8	64000	11%	8	19	64
最大值	187500				187

通过每一个周期对某个中断端点的访问，中断传输就可以通过 USB 了。不对一个端点进行访问和请求一个中断传输，主机就无从知道该端点是否发起了一个中断。如果主机访问时端点没有中断数据要传送，它就会出 NAK 响应。为了防止一个客户软件被错误地告知 IRP 已经结束了，一个端点只能在它有等待处理的中断时，才能提供中断数据。数据负载长度为 0 也是一个有效的传输，并且在某些应用中可能会用到。

### 5.7.5 数据顺序

中断处理即可以选择使数据触发比特只在成功地完成了传输时触发，也可以选择对数据触发比特进行连续地触发。主机在任何情况下都必须假定设备完全遵守了第 6 章所规定的握手/重试原则。一个设备可以选择总是对 DATA0/DATA1 PID 进行触发，从而使它可以忽略来自于主机的握手信息。但是，在这样情况下，由于主控制器会将下一个分组解释成为一个丢失分组的重试操作，在错误发生时客户软件会丢失一些数据分组。

如果由于传输错误而在一个中断管道上检测到了禁止条件或从端点返回了一个 STALL 握手信息，所有未处理的 IRP 都会被撤消。通过在一个分开的控制管道中的软件介入。可以消除 STALL 条件。这种恢复操作也一定会对该端点的 DATA0 中的触发比特进行重置。该客户软件也必须调用一个 USBD 函数来为 DATA0 重置主机数据触发比特，回应并清除主机上的禁止条件。

如果在总线上检测到了会影响某个传输的错误，会对中断处理操作进行重试。

## 5.8 批量传输

批量传输类型是设计用来支持那些在不同的时刻需要传输相对而言数据量较大的数据，并且该传输可以推迟到有可用带宽时再进行的设备。申请一个批量传输类型的管道要提供下列条件：

- (1) 根据可用的带宽对 USB 进行访问。
- (2) 如果由于总线上的错误而出现偶然的传输失败，需要重试。
- (3) 保证对数据的传递，但不对带宽或时延提供保证。

批量传输仅会在有可用带宽的基础上进行。对一个具有大量空闲带宽的 USB 系统来说，批量传输可能相对快一些；如果一个 USB 系统只有很少的带宽可用，那么批量传输就会在一个相对较长的时间内慢慢地进行。

### 5.8.1 数据格式

批量管道中的通信流不具有 USB 规定的数据结构。

### 5.8.2 方向

批量管道是一个流管道，因此一个给定的管道中总是有通信流进入或流出主机。如果一个设备要求双向的批量通信流，就必须使用两个批量管道，一个方向一个。

### 5.8.3 分组尺寸限制

用于批量传输的一个端点要说明它可以从总线上接收或向总线发送的最大数据负载的大小。USB 规定所允许的批量数据负载的最大尺寸只能是 8、16、32 或 64 字节。该值仅适用于数据分组的数据负载。也就是说，所声明的尺寸只对第 6 章中规定的分组的数据域有意义，并不包括其它协议所要求的信息。

批量端点被设计成可以支持最大的数据负载尺寸。一个批量端点将在其配置信息中报告它的最大数据负载的大小。USB 不要求数据负载一定以最大尺寸进行传输，即如果数据负载小于最大尺寸，无须将它填充至最大尺寸。

所有的主控制器都需要为批量端点提供对 8、16、32 和 64 字节的最大分组大小的支持。主控制器无须支持那些最大分组尺寸大于或小于这些值的分组。

在配置过程中，USB 系统软件将读取端点的最大数据负载值，并且保证不会向该端点发送超过其所能支持最大数据负载尺寸的数据负载。

对于一个端点所传送的数据负载其数据域必须小于或等于从该端点的 MaxPacket-Size 中读取的值。当一个批量 IRP 中的数据不能放在一个具有最大尺寸的数据负载中时，除了最后一个数据负载用来装剩余数据之外，其它所有的数据负载都必须具有最大值。如果一个端点要传输的数据量小于客户软件的预计值，那么主控制器将收到一个数据负载小于最大值的数据。这一分组将使主机废除当前的 IRP 而转向下一个 IRP，或像 4.3.2 节所描述的那样禁止该管道。如果接收到的数据负载大于所预期的值，那么该批量 IRP 就会被禁止/撤消。

### 5.8.4 总线访问限制

只有全速率设备可以使用批量传输。

一个端点不能指出一个批量管道所要求的总线访问频率，USB 将负责平衡所有的批量管道和某些未处理的 IRP(为了在客户软件和功能模块之间提供“最大可能”的数据传送能力)的总线访问请求。经过总线进行批量传输，具有比进行批量传输更高的优先权。

对批量传输来说，不会像控制传输那样得到对帧时间的保证。批量传输只有在有可用带宽的条件下才会经过总线进行传递。如果有帧时间没有用于其它的用途，批量传输就可以通过总线。如果帧中没有可用于批量传输的时间，在该帧中就不会进行批量传输。如果可用的帧时间需要用于很多的等待处理的批量传输，就会适当地选择批量传输通过总线。如果多个端点都有等待处理的批量传输，根据一个依赖于主控制器实际应用的公平访问原则，将选择不同端点的批量传输进行处理。

一个系统中，所有未处理的批量传输都可以竞争使用同一段可用的总线时间。正因为这样，对于某具体的端点所对应的批量传输而言，USB 系统软件可以自行决定其可用的总线时间。一个端点及其客户软件不能为批量传输假定一个服务速率。当别的 USB 设备插入移出系统，或另外的设备端点需要批量传输时，用于一个客户软件及其端点的总线时间会发生改变。客户软件不能推测批量传输和批量传输的顺序，即在某些情况下，批量传输可以先于控制传输进行。总线频率和帧的定时对可以成功进行的批量传输的最大值提出了限制。对于任一 USB 系统，一帧中所包含的数据负载应小于 728 字节。表 5-6

中列出了有关批量传输的不同尺寸和一帧中可能有的最大传输数的信息。主控制器可以自由地决定,对于用于某个批量传输的独立的总线处理操作,应该怎样使其在帧内和跨越几帧而进行传输。一个端点可以在同一帧内收到用于某个批量传输的所有处理操作,或者是经过若干个不连续的帧才收到这些处理操作。最后,由于不同的实现方式的原因,一个主控制器可能不能在一帧中提供上述的最大处理操作数。

表 5-6 批量传输限制

协议开销(字节,byte)					
13 (3 - syncs, 3 - pids, 2 - EP + CRC, 2 - CRC, 三个字节分组间延迟)					
数据负载	最大带宽/字节/秒	每一次传输所占用的帧带宽	最大传输数	剩余字节数	有用数据/字节/帧
1	107000	1%	107	2	107
2	200000	1%	100	0	200
4	352000	1%	88	4	352
8	568000	1%	71	9	568
16	816000	2%	51	21	816
32	1056000	3%	33	15	1056
64	1216000	5%	19	37	1216
最大值	1500000				1500

### 5.8.5 数据顺序

批量处理操作的数据触发比特仅在处理操作成功完成时进行触发,利用数据触发比特可以在出现差错后撤消处理操作时,保护发送器和接收器之间的同步关系。在由一个适当的控制传输对某个端点进行配置时,批量处理操作被初始化为 DATA0,主机同样会以 DATA0 开始第一个批量处理操作。如果一个批量管道被禁止了,当客户软件通过一个 USBD 函数对禁止进行回应后,数据触发比特会被重置为 DATA0。利用一个适当的控制传输,该端点也可以消除它自己的禁止条件。这一操作也会将端点的数据触发比特复位为 DATA0。

如果在总线上检测到影响某个处理操作的错误,批量处理操作就会被撤消。

## 5.9 总线传输访问

要完成主机和一个 USB 设备之间的任何数据传输,都需要使用 USB 带宽。为了支持各式各样的同步和异步设备,需要满足每一个设备传输要求。为设备分配总线带宽的过程称为传输管理。在主机上有几个用于协调信息通过 USB 的实体:客户软件,USB 驱动程序(USBD)和主控制器驱动程序(HCD),这些实体的开发人员需要了解与总线访问有关的关键概念:

- (1) 传输管理:指用来支持通信流经过 USB 的实体和对象。
- (2) 跟踪处理操作:在处理操作通过 USB 系统时,用于跟踪这些处理操作的 USB 机

制。

(3) 总线时间:通过总线传递一个分组的信息所用的时间。

(4) 设备/软件缓冲区大小:为了支持总线处理操作需要的空间。

(5) 回收总线带宽:如果为别的传输分配了带宽却没有使用,这时控制和批量传输就可以重新利用这些带宽。

上一节主要讲述客户软件和一个功能模块的关系如何,以及通过这两个实体之间的管道的有哪些逻辑流。本节则主要讲述主机的不同部分,以及它们是怎样彼此联系支持数据通过USB进行传输的。这其中也包含了设备开发人员可能会感兴趣的信息,那就是当一个客户请求一个传输时,主机是怎样处理的以及该传输是怎样提交给设备的。

### 5.9.1 传输管理

为了让处理操作通过总线,传输管理包括几个对不同对象进行操作的实体:

(1) 客户软件:通过调用/回调(Callback)来向USBD接口请求IRP,使用/产生某个来自或送往一个功能端点的功能数据。

(2) USB驱动程序(USBD):通过适当的HCD的调用/回调,对客户IRP中来自或送往设备端点的数据进行变换。

(3) 主控制器驱动程序(HCD):对送往或来自处理操作的IRP进行转换(根据主控制器的应用要求)并且对这些IRP加以组织以便用于主控制器的操作。主控制器驱动程序和它的硬件之间的交互依赖于实际的应用,有关内容不属于USB规范的范畴。

(4) 主控制器:利用分组来进行处理操作并产生总线动作,从而通过总线为每个处理操作传递某个功能数据。

图5-10说明了信息在客户软件和USB之间传输时,各实体是如何组织的。在实体之间的接口处,给出了同每一实体相联系的对象。

#### 1. 客户软件

客户软件决定需要为一个功能模块建立哪一种类型的传输。它使用某个适当的操作系统接口来请求IRP。客户软件只清楚操作它的功能模块所需要的管道集(即接口)。客户软件必须清楚并遵守前面所讲到的对每一种传输类型的总线访问和带宽限制。客户软件的请求要通过USBD接口来提出。

某些客户软件可能利用操作系统所规定的另外的设备类型接口来操作USB功能模块,而且它们自己可能不会直接发出USBD调用。但是,却总是有一些最底层的客户软件会发生USBD调用,向USBD传送IRP。所有提交的IRP都必须遵守先前协商好的对带宽的限制,这一限制在设备接入总线和对其进行配置操作时已经设定好了。如果一个功能模块从一个非USB环境进入到一个USB环境,驱动程序将利用存储器或I/O访问来直接操作功能硬件,它是为了操作其功能模块而与USBD相互联系的最底层的客户软件。

在客户软件为其功能模块请求了一个传输并且该请求得到服务之后,客户软件会得到IRP已经完成的通知。如果传输中包含了向主机传输数据的功能,客户软件就会访问与某个完成了的IRP相对应的数据缓冲区的数据。

USBD接口将在第10章中定义。

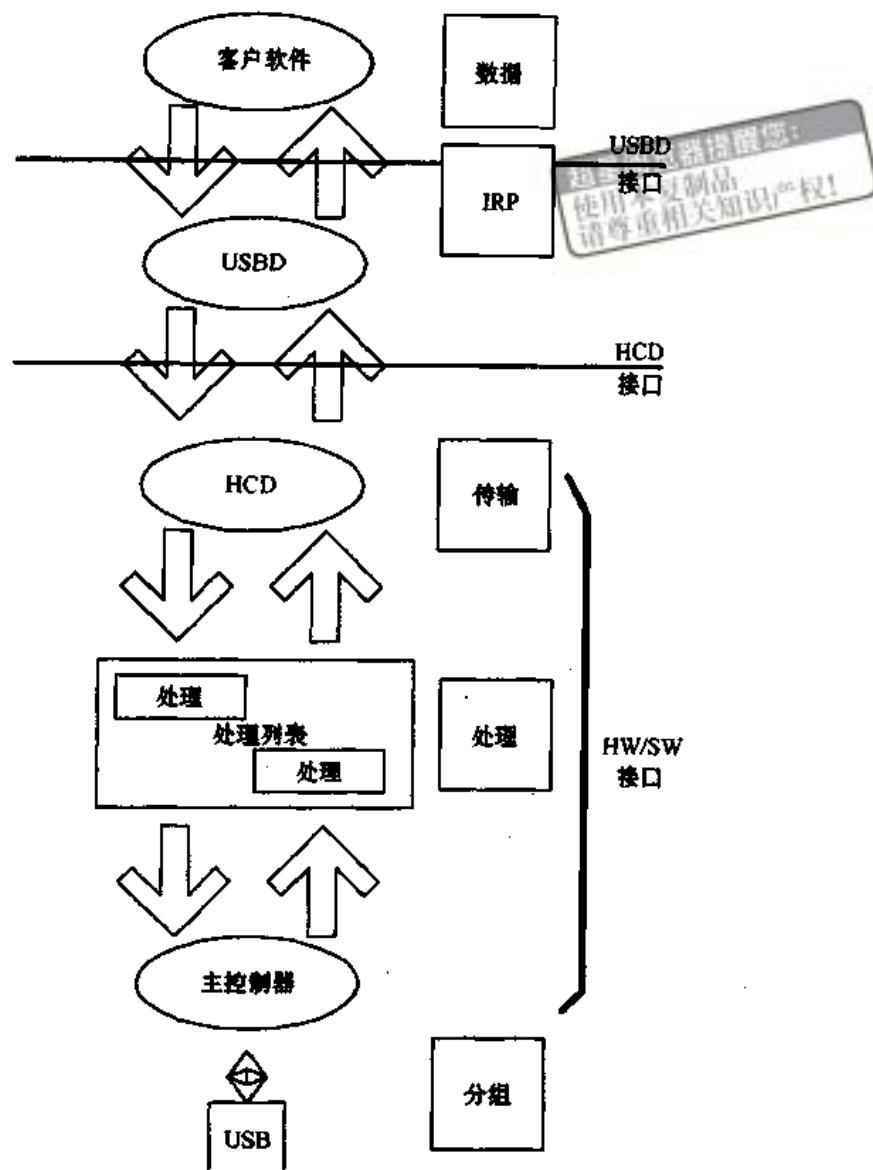


图 5-10 从客户软件至总线的 USB 信息转换

## 2. USB 驱动程序

在一个接入总线的配置操作和正常的传输中, USBD 通常要对总线访问进行两次斡旋。当一个设备插入并对其进行配置时, 需要 USBD 来保证总线上可以容纳所要求的设备配置操作。USBD 从用来说明所需要的设备配置操作的配置软件中接收配置请求, 其内容包括端点、传输类型、传型周期、数据大小等。根据可用的带宽和总线上对请求类型的容纳能力, USBD 将接受或拒绝一个配置请求。如果它接受了该请求, USBD 将为发出请求的设备创建一个所需类型的管道, 该管道具有为这种传输类型所定义的适当的限制。

USBD 配置操作与某个操作系统环境有关。同时为了避免定义额外的(冗余的)接口, 它还要影响操作系统的配置特性。

一旦对一个设备进行了配置, 客户软件就可以为在它和它的功能端点之间传输数据而请求 IRP。

## 3. 主控制器驱动程序

HCD 负责在操作过程跟踪 IRP，并且保证不超过 USB 带宽和帧时间的最大值。当为一个管道建立了 IRP 时，HCD 会将它们填加到处理操作列表中去。如果一个 IRP 完成了，HCD 会通知发出请求的客户软件，该 IRP 已经完成。如果这个 IRP 包含从功能模块至客户软件的数据传输，那么数据将会被放置在客户软件指定的数据缓冲区中。

IRP 定义依赖于所使用的操作系统。

#### 4. 处理操作列表

处理操作列表是一个根据主控制器的应用而作出的说明，它描述了那些需要在总线上运行的，当前仍未完成的总线处理操作集。根据某个与主控制器的应用有关的表示方式，一个典型的处理操作列表包括了一系列的帧说明。只有 HCD 及其主控制器可以访问某个表示方式。每一个帧说明中都包含了处理操作说明，其中指明了像数据大小(字节)、设备地址和端点，以及用来发送或接收数据的存储区域等参数。

一个处理操作列表和位于 HCD 及其主控制器之间的接口都与实现方式有关，USB 规范并没有清楚地定义它们。

#### 5. 主控制器

主控制器访问处理操作列表，并将其转化为总线操作。而且，主控制器提供了一种报告机制，由此可以获得一个处理操作的状态(完成、等待处理、禁止等)。主控制器可以将处理操作转变为与应用有关的适当的操作，从而使得 USB 分组可以通过基于根集线器的总线拓扑结构而进行传输。

主控制器保证协议所规定的总线访问原则得以遵守，例如分组间时序、超时、干扰等。

HCD 接口为控制器提供了一个决定是否允许一个新的管道访问总线的方法。这样做的原因是，对于其支持的总线处理操作组合，主控制器可能会对处理操作之间的最小时间隔有限制/约束。

处理操作列表和主控制器之间的接口隐含在 HCD 和主控制器的应用中。主控制器一般是由硬件来实现的。

#### 5.9.2 跟踪处理操作

一个 USB 功能模块可以允许数据以第 8 章所定义的分组形式通过总线。主控制器使用某种与实现方式有关的表示方法来跟踪那些分组，从而确定什么样的分组在哪一个时刻或以怎样的顺序，向哪一个端点进行传送或来自于哪一个端点。大多数的客户软件都不想处理经过打包的通信流，因为这意味着由此带来的复杂性和互连的依赖性将限制用户的使用。USB 系统软件(USBD 和 HCD)可以将一个用户对数据传输的要求与总线上的分组匹配起来。主控制器硬件和软件使用 IRP 来跟踪那些组合在一起，从而在客户软件和功能模之间传递信息的一个或多个处理操作。图 5-11 总结了对四种传输类型而言，处理操作是如何组合到 IRP 中去的过程。有关每一种传输类型的详细的协议信息请参见第 8 章。

虽然 IRP 跟踪那些为通过 USB 来传送某个数据流而采用的处理操作，但主控制器仍然可以在满足 USB 规定的限制的条件下，自由地选择应该让某个特殊的总线操作怎样地通过总线。例如对同步传输来说，可以使每一帧中恰有一个处理操作。无论如何，一个端点所收到的处理操作的顺序应与一个 IRP 中的顺序完全一样，除非出现了错误。例如，

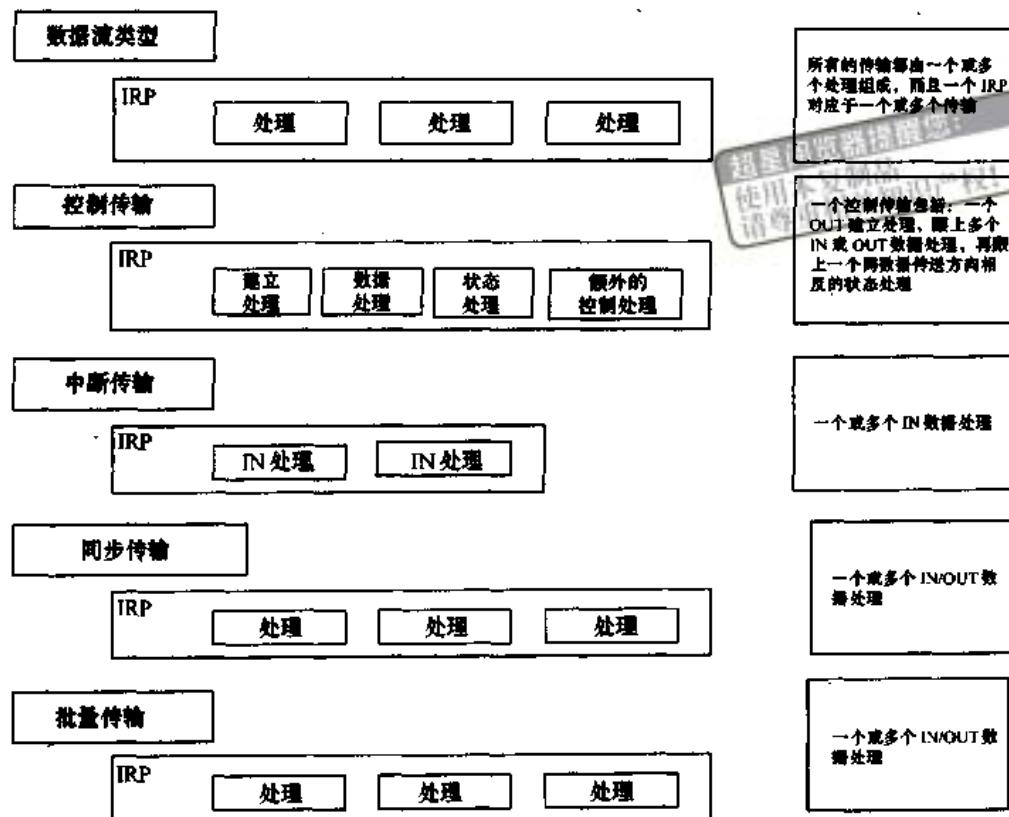


图 5-11 传输通信流

图 5-12 给出了 2 个 IRP, 每个 IRP 都包含了三个处理操作, 分别用于两个管道。对任一种传送类型而言, 在帧 1 中主控制器可以自由地使第二个 IRP 的第一个处理操作跟在第一个 IRP 的第一个处理操作后面而进行传输, 同时在帧 2 中的某处可以利用相反的顺序来传送两个 IRP 中的第二个处理操作。如果是同步传输类型, 那么上述内容就是一个主控制器所具有的唯一的自由度。如果是控制或批量传输, 主控制器就可以进一步在任一帧中, 从任一 IRP 内任意选择处理操作来进行传输。功能模块不能依赖于那些在同一帧中的一个 IRP 中紧接着传输的处理操作。

### 5.9.3 计算总线操作时间

当 USB 系统软件允许为总线建立一个新管道时, 它必须计算某个处理操作所需的总线时间。对总线时间的计算是基于端点所报告的最大分组长度的信息、某个处理操作类型要求的协议开销、比特填充信号的开销、协议所规定的分组间定时要求、处理操作之间的定时等进行的。这些计算值要求保证不超过一帧的可用时间。用于确定处理总线时间的方程如下:

**关键词:**

Data\_bc 数据负载的字节数

Host\_Delay 主机为传输准备的时间或从传输中恢复的时间, 与主控器的实现方式有关

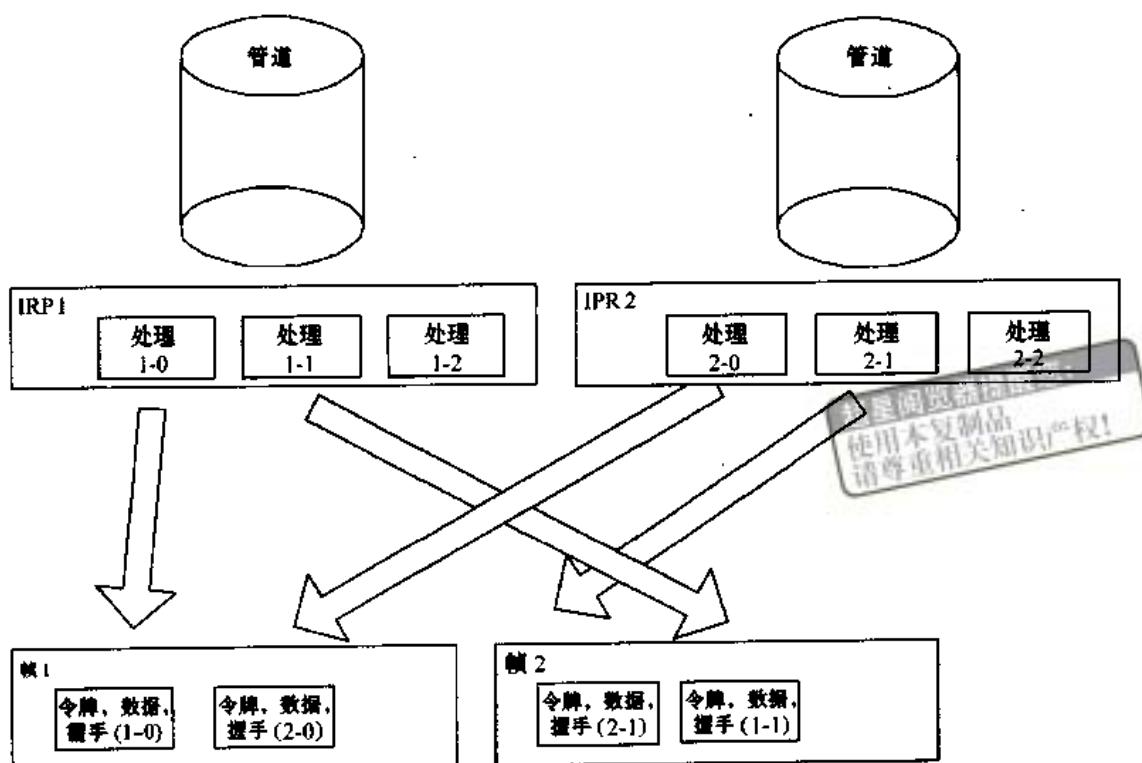


图 5-12 一帧中的处理操作安排

**Floor( )** 结果的整数部分

**Hub\_Ls\_Setup** 主控制器为集线器提供的使其它低速率端口可用的时间。从前一个 PID 结束至低速率 SYNC 开始, 最小值为 4 个全速率比特周期

**BitStuffTime** 用于计算在信号中进行比特填充时, 所需的理论上的附加时间的函数, 最坏为  $(1.1667 \times 8 \times \text{Data\_bc})$

全速率(输入):

非同步传输(包括握手)

$= 91.07 + (83.54 \times \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

$$= 64107 + (2 \times \text{Hub\_LS\_Setup}) + (667.0 \times \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_be}))) + \text{Host\_Delay}$$

上述方程中总线时间的单位是 ns，并且已经考虑了由于设备和主机间的距离而引入的传输时延。这些是用于计算总线时间的典型的方程。但是，不同的应用可以选择使用这些计算结果的粗略的近似值。

某个处理操作所实际占用的总线时间要比计算的结果少，这是因为比特填充开销与数据有关。在最坏情况下，比特填充开销计为原来时间的 1.1667 倍（即方程中 BitStuffTime 函数将 Data\_be 乘以  $8 \times 1.1667$ ）。这意味着在所有正常安排的处理操作完成之后，总是几乎有未使用的时间（按照某个数据模式来说）。如果将所有的批量/控制处理操作都放在一帧末尾进行，所计算的比特填充时间就会比最坏情况时小。这种更进一步的计算所付出的代价是：在某一帧中，因为以前的处理操作由于最坏情况下的比特填充而超过了该帧的规定时间，某个批量/控制传输偶尔会失败。失败的处理操作可以被重试，并且很少会以某个随机数据模式而产生，这样就可以为同步和中断传输时间提供更好的分配方案。无论怎样，由于较少的比特填充而带来的可用总线时间，都可以重新利用，这一点将在 5.9.5 节中讨论。

方程中的 Host\_Delay 因子与主控制器和系统有关，而且它还为主控制器由于访问存储区或其它与应用有关的原因而产生的时延提供了附加的时间考虑。利用 HCD 接口所提供的传输限制管理函数可将这一因子并入到这些方程的应用中去。这些方程一般是由一个相互协调工作的 USBD 和 HCD 软件组合来应用。这些计算结果用来决定在某个 USB 配置操作中，能不能创建一个处理操作或管道。

#### 5.9.4 计算功能模块/软件中的缓冲区大小

客户软件和功能模块都需要为那些在总线上等待的未处理的数据处理操作提供缓冲区。对于非同步管道而言，缓冲区空间只需足够容纳下一个数据分组。如果对于某个端点有多个处理操作请求等待处理，就必须为每一个处理操作提供缓冲处理。一个功能模块可能需要能够用来绝对精确地计算最小缓存时间的方法，这是因为 USB 规范并未对功能模块及其客户软件的交互进行规定。

主控制器应该可以支持无限多的等待总线处理的操作，只要有足够的系统内存可以用于缓冲区和描述符空间等。主控制器可以事先对它允许一个处理操作所请求的帧数目加以限制。

对于同步管道，5.10.4 节详细讲述了对主机和设备有影响的缓存要求。一般来说，缓冲区需要能装载大约 1ms 中可以传输的数据量两倍的数据。

#### 5.9.5 回收总线带宽

USB 带宽和总线访问权的分配，是以最坏情况下对总线传输时间和所需的时延的计算结果为基础而进行的。但是，由于对不同传输类型的限制，以及比特填充所用的总线时间虽然是作为一个常量来计算的，但实际上它却是与数据有关的，所以相对于所计算的帧传输时间来说，在每一帧中经常会有总线时间剩余。为了支持对总线带宽的最有效的利用，控制和批量传输只有在一个总线时间成为可用的条件下才能通过总线。至于一个主

控制器是如何对其提供支持的，则实际上与应用有关。一个主控制器可以依据未处理的IRP传输类型，以及所了解的某个应用的剩余帧时间，来重新利用所回收的带宽。

## 5.10 对同步传输的特殊考虑

支持主机和一个设备之间的同步数据传输是USB所支持的系统功能之一。为了通过USB可靠地传送同步数据，需要认真注意一些细节。有几个USB实体承担了可靠传递数据的责任：设备/功能模块、总线、主控制器和一个或多个软件代理。由于时间是一个同步传输的关键部分，因此理解这些不同的实体在USB中是如何对待总线时间的，对于一个USB设计者来说是十分重要的。

所有的同步设备必须以一定的设备描述符的形式来报告它们的能力。这些能力还要以这样一种形式来说明，从而使得潜在的用户可以以此来决定该设备是否为他的问题提供了解决办法。一个设备的特定的能力会带来价格上的差异。

在任何通信系统中，为了可靠地传递数据，发送器和接收器必须保持足够的同步。而在一个异步通信系统中，为了可靠地传递数据，通过让发送器可以对接收器并未正确接收的一个数据项进行检测并简单地重试对该数据的传输，这样就可以实现数据可靠传送。

在一个同步通信系统中，为了实现数据的可靠传输，发送器和接收器必须保持时钟和数据同步。USB没有为同步数据提供传输重试功能，因而可以为同步传输分配最小的带宽，并且时钟同步不会由于重试时延而丢失。但是，关键在于，无论是在通常的数据传输情况下，还是在总线上出现差错的情况下，一对USB同步发送器/接收器要一直保持同步。

在很多处理同步数据的系统中，系统中的所有实体都使用一个单一的通用时钟来进行同步；例如PSTN——公共交换电话网就是这样。由于各式各样的具有不同频率的设备都可能会接入到USB系统中，因此不可能有这样一个单一的时钟信号，它可以提供为了满足所有的设备和软件的同步要求所需的全部特性，同时还要适应大众市场上对PC产品的成本要求。为此，USB定义了一种时钟模型，它允许大量的设备在总线上共存，并且具有合理的费用。

本节将介绍一些选项或特性，同步端点可以使用这些选项或特性将一个非USB应用的功能模块和一个USB功能模块之间的操作差别减至最小。我们还将举例说明一个非USB和USB功能模块之间的相同点和差别。

本节和剩余部分将说明以下一些关键概念：

- (1) USB时钟模型：一个USB系统中存在的哪些时钟会对同步数据传输产生影响。
- (2) USB帧时钟与功能模块时钟同步选项：USB帧时钟怎么同一个功能模块时钟联系起来。
- (3) 帧跟踪的开始：对于帧起始令牌(SOF)和USB帧，同步端点所具有的责任/机会。
- (4) 数据预缓存：在对数据进行生成/传输/使用之前，存储数据的要求。
- (5) 差错控制：差错控制中有关同步传输的细节。

(6) 为了实现速率匹配进行的缓冲操作: 可以用来计算同步端点所需缓冲区空间大小的方程。

### 5.10.1 非 USB 同步应用实例

我们所使用的例子是一个相当普遍的例子。当然可能还有其它更简单或更复杂的情况。

实例中包括一个 8kHz 的单声道话筒, 它通过一个与之相连的混合驱动器, 向立体声扬声器发送 44kHz 的输入数据流。位于输入和输出端的速率匹配驱动器将抽样速率和编码方式, 从最初的速率和设备最初的编码方式转变为混合器所希望的速率和编码方式。参见图 5-13。

PC 中的主时钟(可以由实际的时钟所驱动的软件来提供)用来唤醒混合器, 使其向输入源请求输入数据并向输出端提供输出数据。在本例中, 假定其每 20ms 被唤醒一次。话筒和扬声器都有它们各自的抽样频率, 在谈到这两者或混合器主时钟时, 它们是不同步的。话筒按照其平常的速率来产生数据(样值为 1 个字节, 每秒钟 8000 次抽样), 而扬声器也以其正常的速率来消耗数据(每个样值为 4 个字节, 每秒钟 44100 次)。当谈到系统中的这三个时钟时, 它们都可能发生漂移和抖动。每一个速率匹配器也可以在不同于混合驱动器、输入源/驱动器、或输出端/驱动器的速率下工作。

速率匹配器还负责监测它的设备中同混合器主时钟相比较而言的长期数据速率, 而且为了将其设备的数据速率调整为混合器的数据速率, 它还可以附加一个额外的样值或将两个样值融合。这种调整可能每 2s 就需要进行一次, 但通常都是偶然才发生的。速率匹配器还提供了额外的缓冲功能来进行速率的匹配。

注意其它一些应用可能不能容忍对样值的调整, 并会需要其它方法来对时钟进行调整, 从而与漂移的设备时钟相配合, 或者需要其它时钟同步方法来保证不会发生频率漂移。

混合器总是希望从它的输入设备中收到恰好为一个服务期的数据(20ms 服务期), 并为其输出设备产生恰好可以占有一个服务期的数据。如果它的输入/输出设备没有可用的数据, 可以推迟对混合器的操作而使服务时间小于一个服务期。混合器假定这种延迟是可累加的。

输入和输出设备及其驱动器希望在它们的混合器处理完一个服务期的数据后, 可以放置/获得数据来作为对来自于 DMA 控制器的硬件中断响应。它们希望获得/放置的数据恰好可以占有一个服务期。输入设备在每个 20ms 服务期中可以生成 160 字节(10 个样值)。而输出设备在每个 20ms 服务期中可以消耗 3528 个字节的数据(882 个样值)。DMA 控制器可以在设备和主机缓冲区之间, 以比任一设备的抽样速率都要快得多的速率来传送单个样值。

输入和输出设备提供了占有两个服务期的系统缓冲操作。其中一个缓冲区一直由 DMA 控制器来处理, 而另一个缓冲区要保证其在当前的缓冲区用完之前已经处于就绪状态。要是当前的缓冲区已经清空了, 硬件中断会提醒设备驱动器, 而设备驱动器则会向速率匹配器请求为它分配一个缓冲区。在当前的缓冲区用完之前, 设备驱动器会发送一个新的缓冲区 IRP 请求。

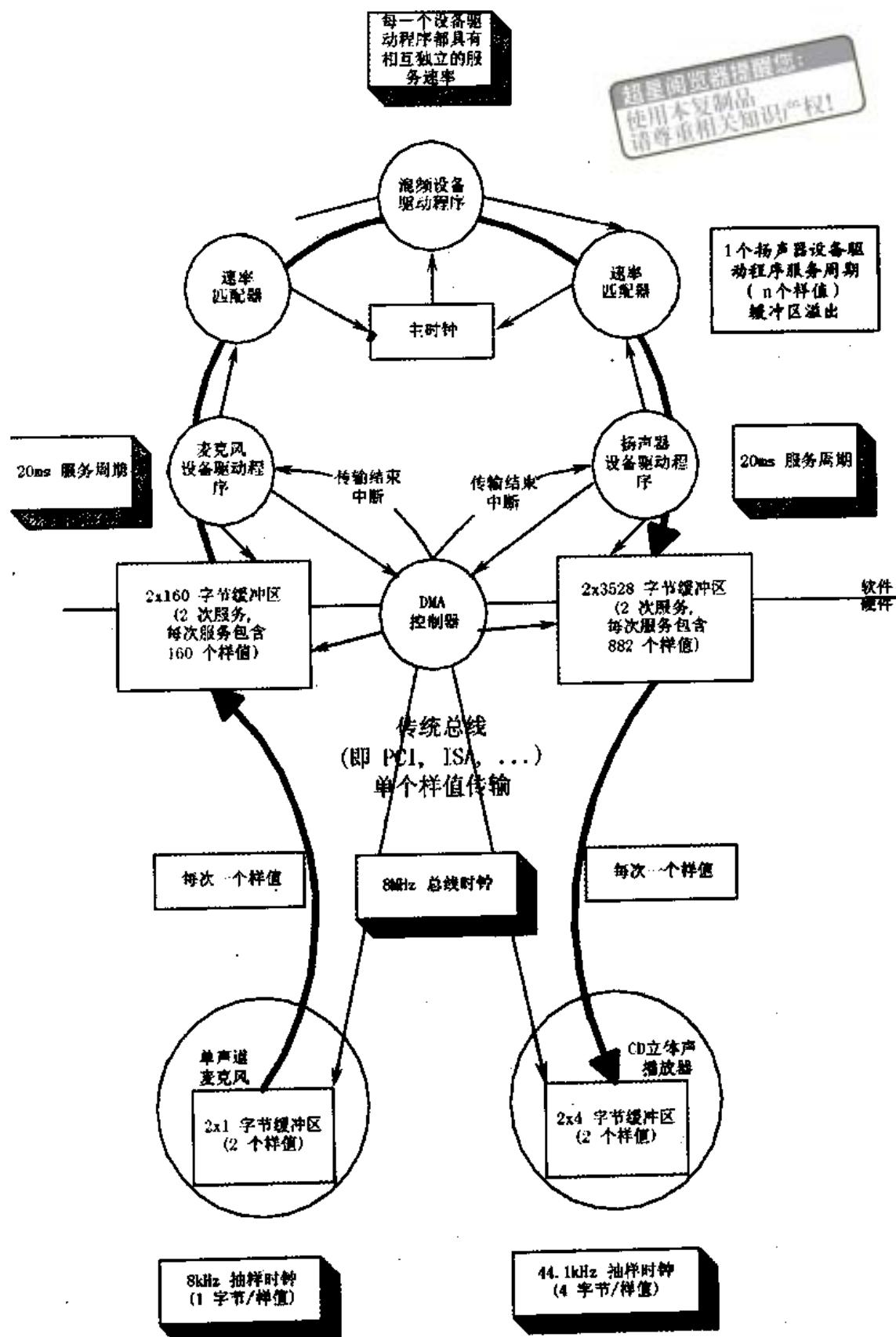


图 5-13 非 USB 同步应用实例

设备可以对两个样值数据进行缓冲,从而保证在系统对前一个/后一个样值作出反应时,设备在下一个抽样期间总是有一个样值可以供其处理。

可以选择对驱动器的服务时间,从而使其不受操作系统环境中可能存在的中断时延变化的影响。为了进行可靠的操作,不同的操作系统会需要不同的设备服务时间。由于系统中的其它软件也可能会请求处理时间,所以要对服务时间进行选择,使得系统的中断负载最小。



### 5.10.2 USB 时钟模型

可以利用时钟来作为 USB 系统中的时间基准。实际上,在一个 USB 系统中存在着多个时钟,我们必须加以了解。

(1) 抽样时钟:该时钟决定了主机中的客户软件和功能模块之间所传输的样值和一般数据速率。在非 USB 和 USB 应用中,该时钟无须变化。

(2) 总线时钟:该时钟周期为 1.000ms(频率为 1kHz),并由总线上的帧开始(SOF)分组速率来指示。它同非 USB 实例中的 8MHz 时钟有些类似。在 USB 环境下,总线时钟通常具有比抽样时钟更低的频率;而在非 USB 环境下,总线时钟频率几乎总是比抽样频率要高。

(3) 服务时钟:该时钟由客户软件为了服务于在执行操作之间所存储的 IRP,所需要的运行速率决定。在 USB 和非 USB 环境下,这一时钟也可以相同。

在今天存在的大多数操作系统环境中,如果每一个设备都要被高抽样速率下的每一个抽样值所中断,那就不可能支持大量的同步通信流。因此,如果不是多个分组,而是多个样值,就不会被客户软件处理,而是把它交给主控制器,然后根据预先商量好的总线访问要求而使其顺序通过总线。

图 5-14 给出了一个实例,它具有一个合理的 USB 时钟环境,这一点与图 5-13 所给出的非 USB 实例相同。

图 5-14 显示了信息从一个作为输入设备的话筒,到一个作为输出设备的扬声器所经过的一条典型的环路。其中出现的时钟、分组和缓冲操作都给出了。在以下几节中将对图 5-14 进行更详细的介绍。

给出这个实例的目的是为了在与前面介绍的非 USB 实例比较时,对 USB 所带来的差异予以区别。这些差异位于缓冲操作,在给定了一个 USB 总线时钟条件下的同步操作和时延等方面。位于设备驱动器之上的客户软件在多数情况下可以不受影响。

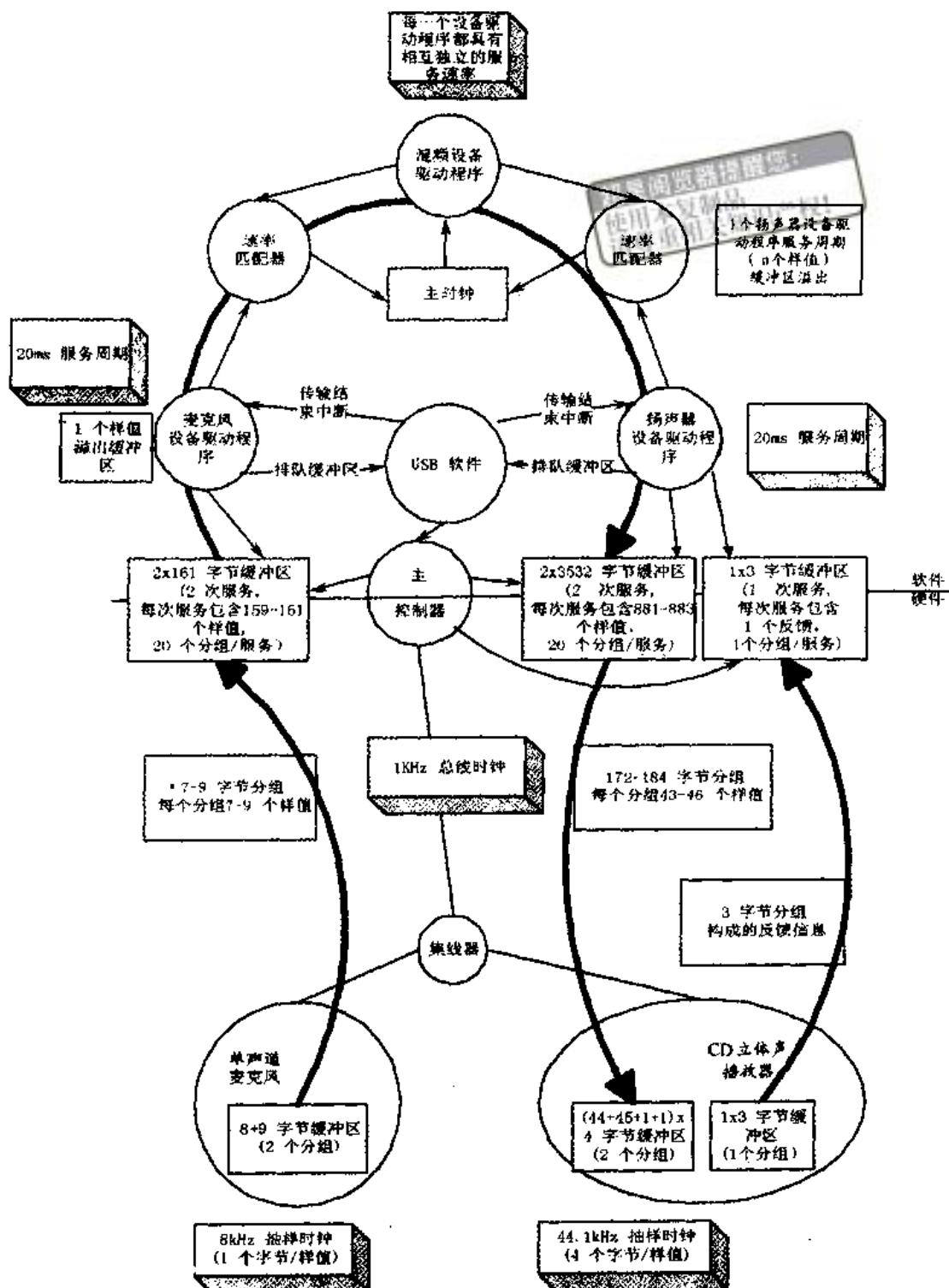


图 5~14 USB 同步应用

这样在实际传送的数据和所预计的数据比较时,也可能会有变化。

(3) 时钟相位差异:如果两个时钟相位没有锁定,由于经过一段时间时钟频率节拍会有所不同,从而会在不同的时间点上出现不同数量的数据。这将导致与量化/抽样有关的人为影响。

总线时钟提供了一个中心时钟,USB 硬件和软件可以利用它来实现某种程度上的同

步。但是,通常来说,在大多数 PC 操作系统中为实时操作系统调度的支持所提供的支持能力,还不能使软件实现同总线时钟相位或频率精确锁定。但是,在主机中运行的软件可以知道数据是经打包之后才通过 USB。对于同步传输类型而言,每帧恰好传输一个单分组数据,并且帧时钟是相当精确的。为软件提供了一个信息,可以允许它对其处理的数据量加以调整,使得它与实际通过的帧时间相适应。

#### 5.10.4 同步设备

USB 包括了一个用于同步设备的框架,它规定了同步类型、同步端点如何提供数据速率反馈,以及它们是如何连接在一起的。同步设备中包括了模拟抽样设备(例如音频和电话设备)以及同步数据设备。同步类型根据一个端点将其数据速率和端点所连接的数据速率进行同步的能力,来对端点进行分类。通过指出相对于 SOF 频率而言实际所要求的数据速率是什么,就可以提供反馈信息。连接能力依赖于所要求的连接质量,端点同步类型以及进行连接的主机的应用能力。根据应用的需要,可能还会需要额外的某类设备的信息。

注意这里“数据”一词使用得非常普遍,它可以指那些代表抽样的模拟信息数据或者更为抽象的信息。“数据速率”指的是模拟信息的抽样速率,或对数据进行计时的速度。

为了决定怎样连接同步端点,需要下列信息:

1) 同步类型。

- (1) 异步型:未实现同步,虽然接收端提供了数据速率反馈。
- (2) 同步型:与 USB SOF 同步。

(3) 自适应型:利用反馈或前馈数据速率信息进行同步。

2) 可用数据速率。

3) 可用数据格式。

如果在信源和信宿之间需要数据速率的精确匹配,或者存在一个允许信源同信宿相连接的可以接受的转换过程,就需要确定同步类型和数据速率的有关信息。应用程序负责决定在可用的处理资源和其它限制条件下(如时延),能否支持这一连接。某个 USB 设备类型会确定如何来描述同步类型和数据速率信息。

一个连接同样需要对数据格式进行匹配和转换,但是对于同步连接而言,这不是一个惟一的要求。格式转换细节请参考与某种格式相关的其它文档。

1. 同步类型

定义了三种不同的同步类型,表 5-7 简要列出了信源和信宿端点所具有的同步特性。我们按照能力不断增强的顺序来说明这些类型。

表 5-7 同步特性

	信 源	信 宿
异步	频率( $F_s$ )可以自由运行 提供隐含的前馈信息(数据流)	频率( $F_s$ )可以自由运行 提供明确的反馈信息(中断管道)
同步	频率( $F_s$ )同 SOF 锁定 使用隐含的反馈信息(SOF)	频率( $F_s$ )同 SOF 锁定 使用隐含的反馈信息(SOF)

(续)

	信 源	信 宿
自适应	频率( $F_s$ )同信宿锁定 使用明确的反馈信息(控制管道)	频率( $F_s$ )同数据流锁定 使用隐含的前馈信息(数据流)

### 1) 异步方式

异步端点不能同 USB 范围内的 SOF 或其它时钟相同步。它们在发送或接收一个异步数据流时,要么以一个确定的数据速率(单一频率端点)和限定的几种数据速率(32kHz, 44.1kHz, 48kHz 等)进行,或者以一个可连续编程来改变的数据速率进行。如果数据速率是可编程的,那它就是在对异步端点进行初始化的过程中设定的。异步设备必须在其设备类型规范中所描述的专用类端点描述符中,报告它的可编程能力。数据速率将会同一个 USB 外部时钟或一个自由运行着的内部时钟相锁定。这些设备会对在 USB 环境中其它地方进行匹配的数据速率加以限制。在异步信源端点每帧所产生的样值数中,包含了其数据速率的隐含信息。异步信宿端点必须向一个自适应驱动器提供明确的反馈信息(参见 5.10.4 节中的“反馈”小节)。

可以举一个异步信源实例,那就是 CD 音频播放器,它根据一个内部时钟或振荡器来提供数据。另外一个例子是一个数字音频广播(DAB)接收器或一个数字卫星接收器(DSR)。同样,这里抽样速率在广播时已经确定了,而不受 USB 控制。

异步信宿端可以是费用较低的扬声器,按照其内部抽样时钟来工作。

当 USB 中的两个或更多的设备为了像同步设备一样工作而需要具有控制 SOF 产生的控制权时,就会产生另一种情况。如果有两个电话机,每一个都同一个不同的外部时钟锁定,就可能出现这种情况。其中一个电话机可以同另一个未与 ISDN 同步的 PBX 进行数字连接,而另外一个电话直接连向 ISDN。每一个设备都会以一个外部驱动的速率,从网络一侧接收或向网络一侧发送数据。由于只有一个设备可以获得 SOF 的控制权,所以另一个只能以同 SOF 异步的速率发送或接收数据。这个例子说明任何一个可以获得 SOF 控制权的设备,都可能会被迫像一个异步设备一样进行工作。

### 2) 同步方式

同步端点可以通过 SOF 同步,对其时钟系统(其时间基准)进行外部控制。这些端点必须进行以下工作:

- (1) 使其抽样时钟由 1ms SOF 所支配(利用一个可编程 PLL)。
- (2) 控制 USB、SOF 产生速率,从而使其数据速率自动锁定为 SOF。一旦这些端点没有获得对 SOF 的控制权,它们必须进行异步模式操作(参看异步方式实例)。

同步端点在发送或接收一个同步数据流时,要么以一个确定的数据速率(单一频率端点)和限定的几种数据速率(32kHz, 44.1kHz, 48kHz 等)进行,或者以一个可连续编程来改变的数据速率进行。如果数据速率是可编程的,那它就是在对同步端点进行初始化的过程中设定的。在一系列 USB 帧中产生的样值数或数据单元数具有确定性和周期性。同步设备必须在其设备类型规范所描述的专用类端点描述符中,报告它的可编程能力。

一个同步信源实例可是一个数字话筒,它可以由 SOF 来合成其抽样时钟,并在每个 USB 帧中产生确定数目的音频样值。另一种可能是来自于一个 ISDN 调制解调器(Modem)的 64Kb/s 比特流。如果 USB SOF 产生的频率锁定为 PSTN 时钟频率(可能是通过

同一个 ISDN 设备), 数据的产生频率也将会锁定为 SOF, 并且该端点可以产生一个稳定的 64 Kb/s 数据。

### 3) 自适应方式

自适应端点是所有端点中能力最强的端点。它们能在其工作范围内, 以任意速率发送或接收数据。自适应信源端点按照信宿所控制的速率产生数据。信宿向信源提供反馈, 使信源了解信宿所要求的数据速率。自适应端点可以同所有类型的信宿端点进行通信。对于自适应信宿端点而言, 有关数据速率的信息隐含在了数据流内。在一定的平均时间内, 所接收到的样值数决定了瞬时的数据速率。如果这个数值在运行过程中发生了变化, 数据速率就会相应地得到调整。

数据速率的工作范围可以以一个速率为中心(例如 8kHz), 也可以在 N 个可编程的或自动检测到的数据速率(32kHz、44.1kHz、48kHz 等)之间进行选择, 或者是一个或多个范围之内(例如, 从 5kHz 至 12kHz, 44kHz 至 49kHz)。自适应设备必须在其设备类型规范中所描述的专用类端点描述符中, 报告它的可编程能力。

可以举一个自适应信源实例, 那就是一个 CD 播放器, 它含有一个完全自适应抽样速率转换器(SRC), 它使得输出抽样频率不必再为 44.1kHz, 而是在 SRC 工作范围内的任意值。自适应信宿(接收器)包括高性能数字扬声器、耳机等等之类的端点。

## 2. 反馈

通过指出相对于 USB SOF 频率而言, 它要求的数据速率的准确值( $F_f$ )是什么, 一个异步信宿可以向一个自适应信源提供反馈信息。为了能够产生高质量的信源速率并经得起反馈环路带来的时延和差错影响, 对所需的速率值的抽样频率最好要高于 1 秒钟一次(1Hz)。

$F_f$  值包括一个小数部分, 使用它是为了获得 1kHz 帧频中所要求的分辨率。还有一个整数部分, 用来给出每一帧中的最小样值数。在 1kHz 帧频内, 一个样值需要分解为 10 比特。 $(1000/2^{10} = 0.98)$  这一部分占 10 比特, 以无符号定点小数 0.10 格式来表示。整数部分需要 10 比特来为一帧中最多为 1023 个的字节样值进行编码。这 10 个整数比特以无符号定点小数 10.0 格式来表示。合成的  $F_f$  值可以以无符号定点小数 10.10 这样的格式进行编码, 放在 3 个字节中(24 比特)。由于整数部分的最大值限定为 1023, 因此需要从左边开始在 24 比特中确定 10.10 格式所代表的数值, 因而实际上它具有 10.14 这样的格式。我们仅仅需要小数点后的前 10 个比特。低 4 位比特可以选择用来扩展  $F_f$  可达到的精确度, 否则应该置为 0。第 8 章在对其它多字节域定义之后, 给出了比特和字节顺序。

在每一帧内, 自适应信源都要将上一帧中剩余样值小数部分与  $F_f$  相加, 并将样值放入结果的整数部分, 同时为下一帧的样值计数保留小数部分。如果需要的话, 信源可以经过许多帧来观察  $F_f$  的表现, 以此来得到更为精确可靠的速率。

通过在  $2^{(10-P)}$  个帧周期内, 以  $FS \times 2^P$  的频率来计算一个时钟内的周期数, 信宿可以确定  $F_f$ , 其中 P 是一个整数。P 实际上被限制在了[0, 10]这一范围之内, 因为使用低于  $FS$  的时钟和试图使一帧中的更新次数多于一次都是没有意义的。每  $2^{(10-P)}$  帧, 计数器就会将结果读到  $F_f$  中去, 并进行重置。只要没有时钟周期被漏掉, 经过很长时间计数值也会是精确可靠的。一个端点只需要实现其  $F_f$  最大时所需要的计数比特就可以了。

例如,一个数字电话端点通常利用对 64kHz 时钟分频的办法来得到其 8kHz 的  $F_s$ , 并使用 64kHz 的时钟来产生连续的数据流。64kHz 时钟相位也可以给出额外的比特的精确度, 这时实际上  $P = 4$ 。这样一来每  $2^{(10-4)} = 64$  帧,  $F_f$  就会更新一次。需要一个 13 比特长的计数器来保存  $F_f$ , 其中 3 比特用于表示每帧 8 个样值, 10 个比特用于小数部分。这 13 比特实际上是在 10.14 格式的  $F_f$  值中提供了一个格式为 3.10 的区域, 其余的比特应置为 0。

选择什么样的  $P$  值与端点有关, 并且要位于 1 和 9(包括在内)之间。建议使用较大数值的  $P$ , 因为它可以减小帧计数器的尺寸并增加  $F_f$  的更新速率。更新频率越高, 对信源数据速率的控制就越紧密, 这样可以减小用于控制  $F_f$  变化的缓冲区空间。 $P$  应该小于 10, 这样至少经过两帧  $F_f$  值就会得到平均, 从而减小 SOF 抖动的影响。 $P$  不应该为 0, 因为我们要在丢失  $F_f$  值的情况下, 保证所发送的样值数的误差小于 1。

中断传输用来从反馈寄存器中周期性地读取  $F_f$ 。所需要的对反馈信息的报告速率应该为  $2^{10-P} \text{ms}$ (帧)。每一个更新周期内,  $F_f$  就几乎会被报告一次。如果每一个更新周期内对同样的  $F_f$  值的报告多于 1 次, 不会有什么收获。端点仅会选择前一个  $F_f$  值改变之后, 已经更新了的  $F_f$  值进行报告。

由于错误或在测量  $F_f$  的过程中所积累下来的不正确性, 信源可能会在经过一段较长的时间之后传送了过多或过少的样值。信宿必须有足够容量的缓冲区来容纳它。当信宿识别出这情况时, 它应该对报告的  $F_f$  值进行调整来纠正它。这对于补偿相对时钟漂移也是必要的。对这一纠错处理的应用与端点有关, 并未给以说明。

一个自适应信源可以从一个自适应信宿那里获得有关信宿数据速率的有关信息, 而这些信息与信宿都锁定于同样的时钟。这就像一个两路通话连接的情形。在这种情况下, 就不需要反馈了。

### 3. 连接

为了完整地描述信源与信宿连接的过程, 这里提出了一个互连模型。该模型指明了使用的不同元件, 以及它们是如何组合在一起组成连接的。

该模型用于多信源/多信宿情况之下。图 5-15 说明了一个典型的情况(非常简略和不完整)。一个物理设备通过 USB 规范中所描述的不同硬件和软件层次, 与一个主机应用软件相连。在客户接口层, 应用软件所面对的是虚拟化了的设备。从应用软件的立足点出发, 系统中只存在虚拟设备。应该由设备驱动程序和客户软件来决定物理和虚拟设备之间所具有的确切关系。

设备生产商(或操作系统供应商)必须提供必要的设备驱动程序软件和客户接口软件, 从而将他们的设备从物理实现方式转变为一个符合 USB 的软件实现方式, 即虚拟设备。正如前面所声明的, 依赖于装入软件的能力, 虚拟设备可以显示出不同于物理设备的同步行为。但是, 同步分类方法对物理和虚拟设备都同样适用。所有的物理设备都属于三种可能的同步类型中的一种。因此, 置入设备驱动程序和/或客户软件中的功能同一个物理设备所具有的功能是相同的。所以“应用软件”一词必须改为“设备驱动程序/客户软件”。在一个物理信源和虚拟信源连接时, “虚拟信源设备”必须改为“物理信源设备”, 而“虚拟信宿设备”也必须改为“虚拟信源设备”。在一个虚拟信宿和物理信宿连接时, “虚拟信源设备”必须改为“虚拟信宿设备”, 而“虚拟信宿设备”也必须改为“物理信宿设备”。

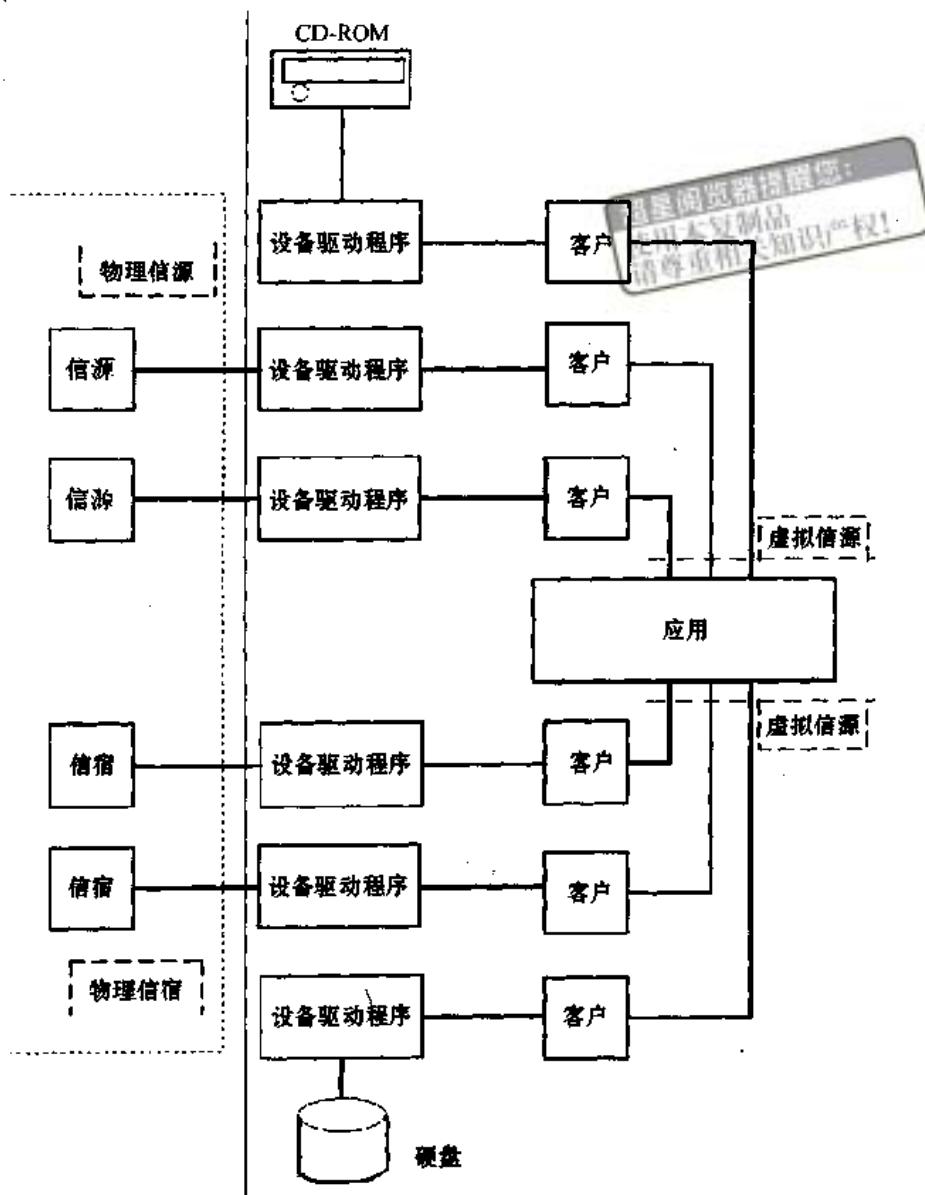


图 5-15 信源信宿连接实例

将速率适配功能放到设备驱动程序/客户软件层具有明显的好处,它可以隔离所有的应用程序,使用那些与速率适配问题有关的设备。那些本该服务于多速率的应用程序就可以简化为更简单的单速系统。

注意该模式并不仅限于 USB 设备。例如,一个 CD - ROM 驱动器含有 44.1kHz 音频信号,就可以表现为一个异步信源,或者是一个同步信源,或者一个自适应信源。异步操作意味着该 CD - ROM 按照它从光盘上读取数据的速率来填充其缓冲区,而驱动器则根据它的 USB 服务时段来清空缓冲区。同步操作则表示驱动器使用 USB 服务时段(例如 10ms)和标称的数据抽样速率(44.1kHz)来决定,在每一个 USB 服务时段内输出 441 个样值。

自适应操作将建立一个抽样速率变换器,来对 CD - ROM 输出速率和不同的信宿抽

样速率进行匹配。

利用这一参照模型,可以确定在不同的信源和宿主之间建立连接时,哪些操作是必要的。另外,该模型指出了在哪一层必须或可以实现这些操作。首先,要有一个将物理设备映射到虚拟设备上的阶段,反之亦然。这由驱动程序和/或客户软件来完成。依据软件所含有的功能,一个物理设备可以转换为一个具有完全不同的同步类型的虚拟设备。第二个阶段是应用软件利用虚拟设备。将速率适配功能放在软件堆中的驱动程序/客户软件层,可以将与虚拟设备通信的应用程序,从为每一个与之相连的设备都进行速率匹配的压力下解放出来。一旦决定了虚拟设备的特性,那么实际的设备特性并不比另一个驱动程序的真实物理设备特性更令人感兴趣。

举例来说,我们来看看一个混合器应用,它在源端连接了不同的信源,每一个信源都工作在自己的频率和时钟下。在混合操作开始之前,所有的流必须转变为一个共同的频率,并且按照一个通用的参考时钟进行锁定。这一工作可以在物理设备到虚拟设备的映射层来进行,或者由应用程序本身对每一个信源设备进行独立地管理。宿主端必须进行类似操作。如果应用程序将混合过的数据流发送给不同的宿主设备,它要么亲自为每一个设备进行速率匹配,要么就是在可能的情况下,依靠驱动程序/客户软件来完成这一工作。

表5-8指出了在交叉点,应用程序必须进行什么样的操作来把一个信源端点连接到一个宿主端点上去。

如果需要RA但却不能提供时,通过进行丢弃/填充样值可以模拟速率调整过程。这时连接仍然可以建立,但很可能会出现一个对质量过低的警告,否则,就不能建立连接。

表5-8 连接要求

信源端点			
宿主端点	异步	同步	自适应
异步	异步信源/宿主RA 参见注释①	异步SOF/宿主RA 参见注释②	数据+反馈帧通(Feedthrough) 参见注释③
同步	异步信源/SOF RA 参见注释④	同步RA 参见注释⑤	数据帧通+应用反馈(Feedthrough) 参见注释⑥
自适应	数据帧通(Feedthrough) 参见注释⑦	数据帧通(Feedthrough) 参见注释⑧	数据帧通(Feedthrough) 参见注释⑨

注释:

① 应用中的异步RA。利用数据流内所隐藏的前馈信息, $F_{S_i}$ 由信源来决定,接收器决定 $F_{S_o}$ ,这是根据来自于接收器的反馈信息而得到的。如果名义上 $F_{S_i} = F_{S_o}$ ,要是由于同步性能缺陷所造成的节略/填充可以忍受的话,这个过程就会简化为一个直通的连接。在音频应用当中,这些节略/填充会对信号带来可以觉察到的影响。

② 应用中的异步RA。 $F_{S_i}$ 由信源决定,但是要同SOF锁定。根据来自于接收器的反馈信息, $F_{S_o}$ 可以由接收器来决定。如果名义上 $F_{S_i} = F_{S_o}$ ,要是由于同步性能缺陷所造成的节略/填充可以忍受的话,这个过程就会简化为一个直通的连接。在音频应用当中,这些节略/填充会对信号带来可以觉察到的影响。

③ 如果 $F_{S_o}$ 落在了自适应信源可以锁定的范围之内,就可以建立一个直通的连接。 $F_{S_i} = F_{S_o}$ ,且根据来自于接收器的反馈信息, $F_{S_i}$ 和 $F_{S_o}$ 都由异步接收器来决定。如果 $F_{S_o}$ 落在了自适应信源可以锁定的范围之外,自适应信源会切换到同步模式,并会使用注释2所给出的方法。

续

④ 应用中的异步 RA。 $F_{S_i}$  由信源决定。 $F_{S_o}$  由信宿决定，并且同 SOF 相锁定。如果名义上  $F_{S_i} = F_{S_o}$ ，要是由于同步性能缺陷所造成的节略/填充可以忍受的话，这个过程就会简化为一个直通的连接。在音频应用当中，这些节略/填充会对信号带来可以觉察到的影响。

⑤ 应用中的同步 RA。 $F_{S_i}$  由信源决定并且要同 SOF 锁定。根据来自于接收器的反馈信息， $F_{S_o}$  可以由接收器来确定。如果  $F_{S_i} = F_{S_o}$ ，该过程就会退化为一个无损耗的直通连接。

⑥ 应用程序会提供反馈，以实现信源和 SOF 的同步。然后自适应信源就会表现为一个同步端点，并出现注释中提到的情况。

⑦ 如果  $F_{S_i}$  落在了自适应信宿可以锁定的范围之内，就可以建立一个直通的连接。

$F_{S_i} = F_{S_o}$ ，并由信源来决定，且和信源锁定。

如果  $F_{S_i}$  处于自适应接收器的可锁定的范围之外，那么主机就会进行同步 RA 操作，以便提供一个处于这个自适应接收器可锁定落围之内的频率  $F_{S_o}$ 。

⑧ 如果  $F_{S_i}$  落在了自适应信宿可以锁定的范围之内，就可以建立一个直通的连接。 $F_{S_i} = F_{S_o}$ ，并由信源来决定，且和 SOF 锁定。

如果  $F_{S_i}$  处于自适应接收器的可锁定的范围之外，那么主机就会进行同步 RA 操作，以便提供一个处于这个自适应接收器可锁定落围之内的频率  $F_{S_o}$ 。

⑨ 当连接建立后，应用程序将使用反馈控制来设置自适应信源所具有的  $F_{S_o}$ 。要是缺少正在传送的反馈信息，自适应信源就会作为一个异步信源来工作，此时正如注释 7 所述的情况。

### 1) 音频连接

当上述内容应用于音频数据流时，速率调整过程就会被抽样速率转换过程所取代，该过程实际上是速率调整的特殊形式。我们使用某种形式的抽样插值法(Sample Interpolation)来代替差错控制，进行输入和输出抽样速率的匹配。依据所使用的抽样插值技术，转换操作所得到的音频性能(失真，信噪比)等会有显著差别。一般来说，质量越高所需的处理能力就越强。

### 2) 同步数据连接

对于同步数据，要使用速率调整。对于许多使用了某种差错控制方法的应用程序来说，偶然的节略/填充是可以接受的。差错控制包括差错检测和丢弃、差错检测和重传，或前向纠错。对速率的节略/填充与信源和信宿之间时钟不匹配有关，并且可能成为信道中主要的差错来源。如果差错控制能力是充分的，就自然可以实现差错纠正。

## 5.10.5 数据预缓存

USB 要求在处理/传送数据之前，设备应预先对数据进行缓存，从而在每一个管道处理操作一帧一帧地通过总线时，使主机管理起来具有更大的灵活性。

对于由功能模块到主机这样的传输，端点必须在帧 X 期间保存样值，直到它收到了 X + 1 帧的帧开始(SOF)令牌分组。它会把数据从帧 X 移到其分组缓冲区中，并准备在 X + 1 帧内将包含了那些样值的分组发送出去。在一帧中何时发送数据仅由主控制器来决定，并且帧与帧之间可能会有所不同。

对于从功能模块到主机这样的传输，在 Y 帧内的某个时刻端点会接收主机发出的数据。当它接收到 Y + 1 帧的 SOF 时，它就可以开始处理 Y 帧中收到的数据。

当主控制器通过总线传输分组时,这种方法允许一个端点把 SOF 令牌作为一个具有很小的抖动/漂移的稳定时钟来使用;它还允许当分组确实要通过总线时,主控制可以在一帧中精确地变化。同总线访问都是在距离 SOF 有相等时间偏数量的同一时刻一帧一帧进行的情况相比,这种预缓存操作会在一个端点处出现可用的样值和该样值通过总线的时刻之间引入的某个附加的时延。

图 5-16 给出了从一个功能模块到主机的数据传输的时序(输入过程),在  $F_i$  帧内的  $T_i$  时刻,数据  $D_0$  被保存,并在  $F_{i+1}$  帧内送往主机。类似地,对于一个从主机到功能模块的传输(输出过程),数据  $D_0$  在  $F_{i+1}$  帧内被端点接收,并在  $F_{i+2}$  帧内得到处理。

时间:	$T_i$	$T_{i+1}$	$T_{i+2}$	$T_{i+3}$	...	$T_m$	$T_{m+1}$
帧:	$F_i$	$F_{i+1}$	$F_{i+2}$	$F_{i+3}$	...	$F_m$	$F_{m+1}$
总线上的数据:		$D_0$	$D_1$	$D_2$	...	$D_0$	$D_1$
输出处理:			$D_0$	$D_1$	...		$D_0$
输入处理:	$D_0$	$D_1$	...	$D_0$	...		

图 5-16 数据预缓存

### 5.10.6 SOF 跟踪

支持同步管理的功能模块必须能接收和理解 SOF 令牌,来支持前面提到的预缓存操作。某个 SOF 可能会被破坏,所以一个设备必须能做好准备从一个破坏了的 SOF 中恢复出来。这些要求仅限于那些通往全速率设备的同步传输,因为低速率设备不能识别总线上的 SOF。另外,因为在传输过程中 SOF 分组可能被破坏,所有支持同步传输的设备,需要能对一个由于总线错误而不能被识别的 SOF 进行合成。

同步传输需要将适当的数据放在相应的帧中进行传输。USB 要求当向主控制器发出了一个同步传输时,主控制器要识别第一帧的帧标号。在收到所指示的帧标号之前,主控制器不能传递第一个处理操作。在该 IRP 中的每一个后续的处理操作,必须在接下来的帧中传输。如果没有处理操作等待当前的帧进行传输,那么主控制器就不能为一个同步管道传送任何信息。如果指明的帧标号已经过去了,主控制器必须跳过(即不传输)那些在与当前所接收到的帧相对应的处理操作之前的处理操作。

### 5.10.7 差错控制

同步传输没有为数据分组提供重试功能(即一个接收器不会向一个发送器返回握手信息),因此数据传递的及时性不会受到影响。但是,对于那些负责数据传输的部件,了解差错是何时发生的以及差错对通过信流影响如何仍然很重要。尤其是对于一定顺序的分组(A,B,C,D),USB 提供了充足的信息,从而可以发现一个丢失的分组(A,\_,C,D)并且不会让其在不知不觉中变成错误数据或时序(A,C,D 或 A,\_,B,C,D)。协议提供了四种机制来支持这一功能:每帧一个分组、SOF、CRC 和总线操作超时。

对于一般的操作,同步传输每帧需要一个数据处理操作。USB 不能要求在每一帧中传输什么样的数据。数据发送器/信源决定应该提供什么样的数据。每帧中的这种平常的数据,为检测数据丢失错误提供了一个基本的框架。在总线上的传输过程中,一个处

理操作上的任一个阶段都会受到破坏。第 6 章讲述了每一种差错情形是如何影响协议的。

因为每一帧前面都有一个 SOF 分组，并且接收器可以识别总线上的 SOF，因此一个接收器可以确定在两个 SOF 之间有没有产生它所希望的处理操作。另外，因为一个 SOF 分组甚至也会被破坏，所以一个设备必须能对一个丢失了的 SOF 进行如 5.10.6 节所述的重建操作。

一个数据分组在总线上可能会被破坏，因此 CRC 保护允许由一个接收器来确定其接收的数据分组是否受到了损坏。

最后，协议还规定了一些细节，从而允许一个数据接收器根据总线处理操作超时来确定：在其成功地识别了令牌分组之后，它不会再接收数据分组。

一旦一个接收器确定它没有收到一个数据分组，它可能需要了解丢失数据的大小，从而根据其能力来恢复错误。如果每一帧中的数据流都具有同样的数据尺寸，那么该尺寸通常是一个已知的常数。但是，在某些情况下，数据大小会一帧一帧地变化，这时，接收器和发送器就可以利用一个同应用有关的机制，来确定丢失分组的大小。

总而言之，不管一个处理操作事实上是否成功地通过了总线，发送器和接收器总是继续传输其数据/缓冲区流，每帧一个处理操作，以保持数据同步。上述的详细机制允许对破坏了的处理操作进行检测、跟踪和报告，使得一个功能模块或其客户软件能以一个功能模块所具有的适当方式，对破坏作出反应。有关功能模块/应用程序的某个反应的细节，不在 USB 规范所描述的范围内。

### 5.10.8 为速率匹配而进行缓存操作

由于在 USB 系统中，有多个时钟会影响同步通信流，因此需要利用缓存操作来对通过 USB 的通信流进行速率匹配。在设备的每个端点和代表客户软件的主机一侧，都应该有可供利用的缓冲区空间。这些缓冲区为数据提供了存储空间，直到一个传输应当通过 USB 时为止。给出该设备的一般数据速率，就可以计算出通过总线的数据分组的最大尺寸。图 5-17 给出了用于确定设备区和主机上的缓冲区大小和为了支持所需的数据速率所要求的最大分组尺寸的方程。这些方程允许一个设备和客户软件来设计由服务时钟速率（变量 X），抽样时钟速率（变量 C）和样值大小（变量 S）所确定的时间。USB 只允许每个总线时钟内有一个处理操作。这些方程应该提供一些设计信息，用于选择一个端点将在其特征信息中报告的合适的分组尺寸和设备/端点及其客户软件对缓冲区的适当要求。图 5-17 说明了一个典型的同步实例中，缓冲区、分组和时钟所具有的真实值。

USB 数据模型假定设备具有某个正常的样值大小和速率。USB 支持对大小为样值整倍数的分组进行传输，从而使得当同步处理操作在总线上被破坏时，更易于控制差错恢复。如果一个设备不具有正常的样值大小，或者它的样值大于一个分组，它会说明样值为 1 个字节大小。如果一个样值在一个数据分组内被分开了，那么当任意一个处理操作丢失时，差错恢复可能会变化更加困难。在某些情况下，除非接收器知道每一部分的样值是在哪一帧中传输，否则就可能会丢失同步。进而，如果样值数可能会由于时钟纠正而发生变化（例如一个不是来自于设备的时钟），这时要想知道一个部分的样值是什么时候传输的，可能会比较困难或者效率很低，因此，USB 不会跨越分组而将样值分开。

利用缓存操作实现  
同步速率(时钟)匹配!

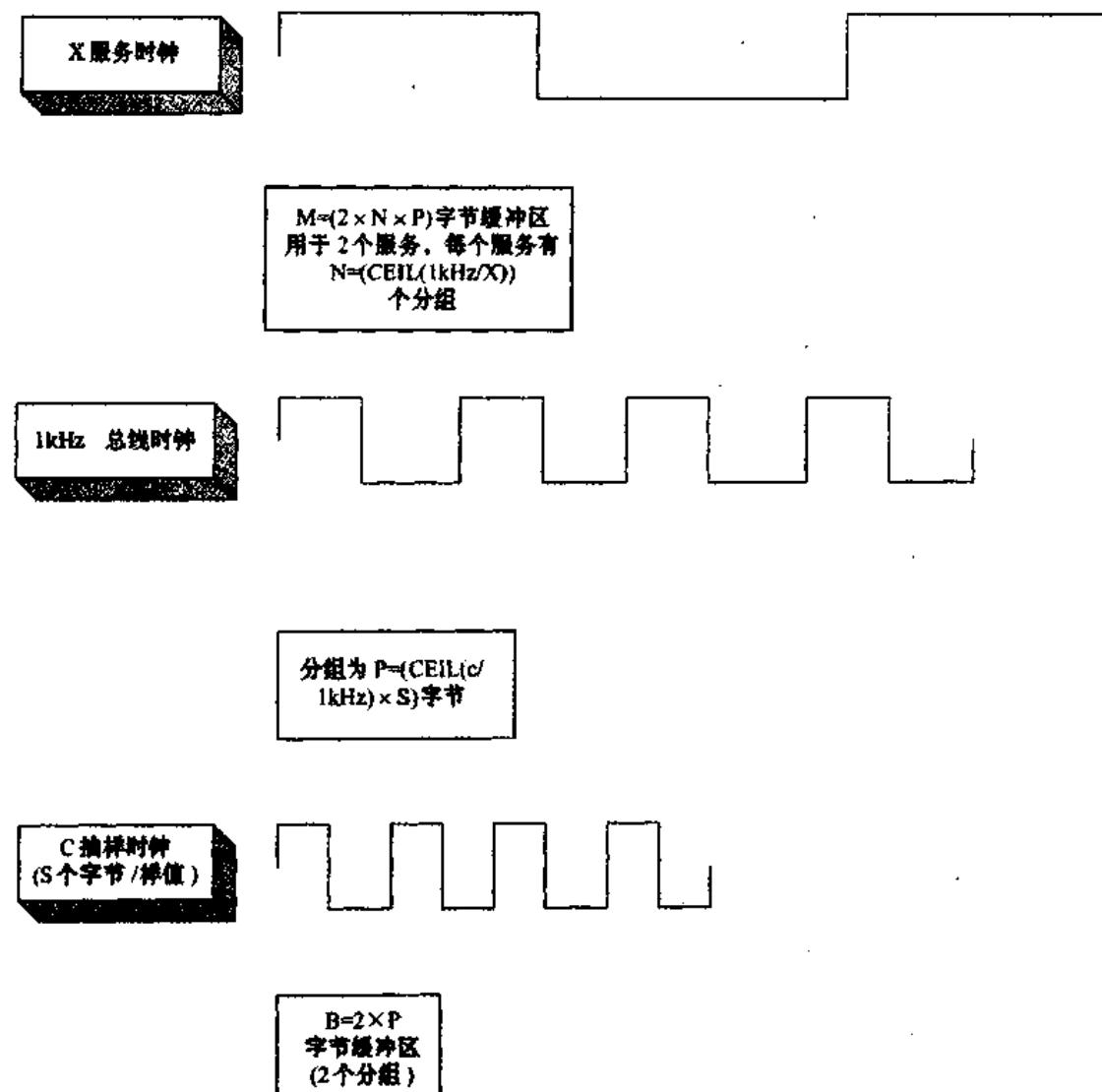


图 5-17 分组和缓冲区尺寸计算方程(适用于速率匹配的同步传输)

## 第6章 USB 总线机械规范

本章将介绍 USB 总线规范中的机械规范。有关 USB 集线器、功能模块和主机的连接器和电缆和可靠性的要求。而电气规范则描述了 USB 速率分配和物理层规范。

### 6.1 机械规范概述

一个 USB 信道的物理拓扑模块或主机之间的连接。每一条内部具有两条电源线和 1.5 Mb/s 的子信道，规范允许仅有两种类型的插头和插座：那些总是接有外部电缆的 USB

本章具体包括以下内容：



机械规范概述；



尺寸要求；



USB 电缆；

那些利用连接器才能实现 USB 布线可分离的设备,例如打印机,扫描仪和调制解调器。有时可能需要内部连接器来满足 USB 规范的电气要求,但是有关内部连接器的机械特性并未包含在 USB 规范当中。所有具有 A 系列或 B 系列连接器的电缆应该满足全速率信道的结构要求,并具有最大功率标准的导线。

系列 A 和系列 B 连接器不能互换,所以系统的完整性不会受到损害。

## 6.2 尺寸要求

缺省的容差列于表 6-1 中,除非另外声明,单位是毫米。

表 6-1 缺省容差

1~5	5~30	30~100	100~300	300~1000	1000~3000	3000~5000
±0.3	±0.4	±0.6	±0.8	±1.6	±2.5	±10

## 6.3 USB 电缆

本规范中定义的所有集线器和功能模块都具有永远与之相连的 USB 电缆或 B 系列连接器。

标准的 USB 电缆包括一对用于电源分配的 20~28 AWG 规格的线对和一对 28 AWG 规格的双绞线(AWG:平均线规),并具有屏蔽和完整的保护层。它适用于工作在 12Mb/s 信号速率下的典型的 USB 外设。

另外一种同样规格的可以替换的电缆没有绞缠在一起的线对和屏蔽,因而只能应用于 1.5Mb/s 信号速率下。在不需要更多的带宽的子通道应用中可以使用这种电缆。在其它方面,子通道和全速率信道具有同样的机械规范。下面我们来详细地介绍 USB 电缆的有关规范。

### 6.3.1 电缆规范

该规范规定了用于全速率设备的双绞线对、28 AWG、PVC 和用于子通道的四线电缆的详细要求。

#### 1. 应用文档

##### 美国标准测试材料(ASTM)

ASTM - D - 4565 通信线缆的外层保护套,以及绝缘材料的物理和环境性能,标准测试方法。

ASTM - D - 4566 通信线缆的绝缘材料和外层保护套的电气性能,标准测试方法。

#### 2. 要求

##### 1) 机械要求

材料方面的要求是:

- (1) 外层保护套:聚氯乙烯(PVC)。
- (2) 颜色:乳白色。

请尊重相关知识产权！  
使用本复制品  
请尊重相关知识产权！

(3) 导线绝缘材料:电源线采用半硬性的 PVC,信号线采用聚乙烯材料(也可选用泡沫塑料)或满足表 6-4 中给出的要求(仅对 12Mb/s 全速率线而言)。

(4) 导线:电源线规格参考表 6-2;信号线的规格是 28 AWG。

电缆结构方面的要求是:

(1) 全速率:整根电缆由四根导线构成;一对 28 AWG 规格的双绞线(数据线对),一对具有完整外保护套非双绞线对(电源线对)。其中双绞线要每 6~8cm 绞一圈。

(2) 子通道:整根电缆由四根导线构成;一对 28 AWG 规格的线对(数据线对)和一对具有完整外保护套的电源分配线对。

(3) 外保护套具体是:

A. 外直径:全速率和子通道,3.4 至 5.3 毫米。

B. 颜色:建议为乳白色。

(4) 导线绝缘材料:

A. 外直径:参见表 6-2。

B. 颜色:参见表 6-5。

(5) 导线:

全速率:

A. 28 AWG 双绞线对。

B. 非双绞线对,可以从表 6-2 中的线束中选出一种用于直流电压分配。

C. 屏蔽:用于屏蔽电磁干扰。

(6) 子通道:

A. 28 AWG 规格线对。

B. 非双绞线对,可以从表 6-2 中的线束中选出一种用于直流电压分配。

(7) 断裂强度:利用 ASTM - D - 4565 标准测试时,最少为 45 牛顿。

## 2) 电气要求

(1) 电压额定值:最大为 30V。

(2) 导线阻抗:根据 ASTM - D - 4565 标准测试时,结果示于表 6-3。

表 6-2 电源线对

规格和导线外直径
28 AWG - .84 ± .05 mm
26 AWG - 1.00 ± .05 mm
24 AWG - 1.10 ± .07 mm
22 AWG - 1.30 ± .07 mm
20 AWG - 1.50 ± .08 mm

表 6-3 导线阻抗

规格	DC 阻抗(最大值)
28	0.232 Ω/m
26	0.145 Ω/m
24	0.0909 Ω/m
22	0.0574 Ω/m
20	0.0358 Ω/m

(3) 不平衡阻抗:根据 ASTM - D - 4566 标准测试时,两根导线之间的不平衡阻抗不应超过 5%。

(4) 长度:对于子通道而言,电缆的最大长度不应超过 3m;全速率电缆不应超过 5m。该要求仅适用于全速率电缆。

(5) 衰减：表 6-4 给出了根据 ASTM - D - 4566 标准测试时，信号线对衰减额定值。

表 6-4 信号衰减

频率(MHz)	衰减(最大值)dB/305 m	频率(MHz)	衰减(最大值)dB/305 m
0.064	4.80	4.000	24.0
0.256	6.70	8.000	35.0
0.512	8.20	10.000	38.0
0.772	9.40	16.000	12.0
1.000	12.0		

(6) 特性阻抗：在 1 ~ 16MHz 频率范围内根据 ASTM - D - 4566 标准测试时，信号线对的特性阻抗为  $(90 \pm 15\%) \Omega$ 。在 1 ~ 16MHz 频率范围内使用时，应用于全速率条件下的信号线对的传播时延必须等于或小于 30ns。注意对于一条全长为 5m 的电缆，信号线对需要聚乙烯绝缘材料来保证传播时延的要求。

(7) 相位偏移：在 10kHz 条件下，两条信号线之间的最大相位偏移应小于 65.6 ps/m。

### 3) 环境要求：

- (1) 额定温度：可在 -40℃ 至 60℃ 时存放；工作温度为 0℃ 至 40℃。
- (2) 易燃性：所用的塑料材料必须满足 NEC 800 号文中有关易燃性的要求。
- (3) 标记：必须具有厂商名称或符号清晰的、永久性标记。
- (4) 限制条件：如果需要的话，所有的供应商都必须有能力提供适当的文件，以此来表明其符合本章所提出的要求。

USB 电缆中的四条导线具有不同的颜色，如表 6-5 所列。

表 6-5 导线颜色

导线	颜色	导线	颜色
+ 数据	绿色	VCC	红色
- 数据	白色	地线	黑色

## 6.3.2 连接器(A 系列)

图 6-1 至图 6-7 给出了 A 系列连接器的图示。

### 1. 插头(A 系列)

与 5.3 节中提到的一样，A 系列的 USB 插头是具有屏蔽外壳的四针插头。

在生产过程中，导线连接点之间已经恰当地连接好了。

### 2. 插座(A 系列)

对于通常的使用情况，有四种插座可供使用。但是，只要满足规范对接口提出的要求，具体采用哪一种形式的插座将由开发人员决定。内部的塑料应该采用乳白色或类似的颜色。

### 3. 连接器连接的特点(A 系列)

### 4. 插座 PWB 管脚图(A 系列)

的镀金  
的基座  
镀金  
的基座  
的镀金  
的基座

R0.64±0.13  
TYP

4.5±0.1  
X

郑重提醒您  
使用本复制品  
请尊重相关知识产权!

卷之三

(1) 尺寸单位为 mm  
(2) 电键:

插头接触点(采用下列方式之一):

在接触的地方，覆盖在量少为 $1.25\mu\text{m}$ 厚的镍上的镀金

• 在接触的地方，覆盖在量少为 1.25um 厚的镍上的量薄至少应为 0.5um。

为 $0.75\mu\text{m}$ 的钯上，还要有量少为 $0.05\mu\text{m}$ 厚的镍金。

在接触的地方，镀层厚度为 $0.75\mu\text{m}$ 的钯合金上，还要有量少为 $0.05\mu\text{m}$ 厚的镍金

外套：至少应有2.5um的锡或以锡为主的合金覆层

(3) 在一个连接器内的四个连接点必须处在同一个 $0.2$ 的区域内

(4) 材料：插座外套——磷青铜，UN

(5) 美標 ANSI Y14.5M-1982 標準圖

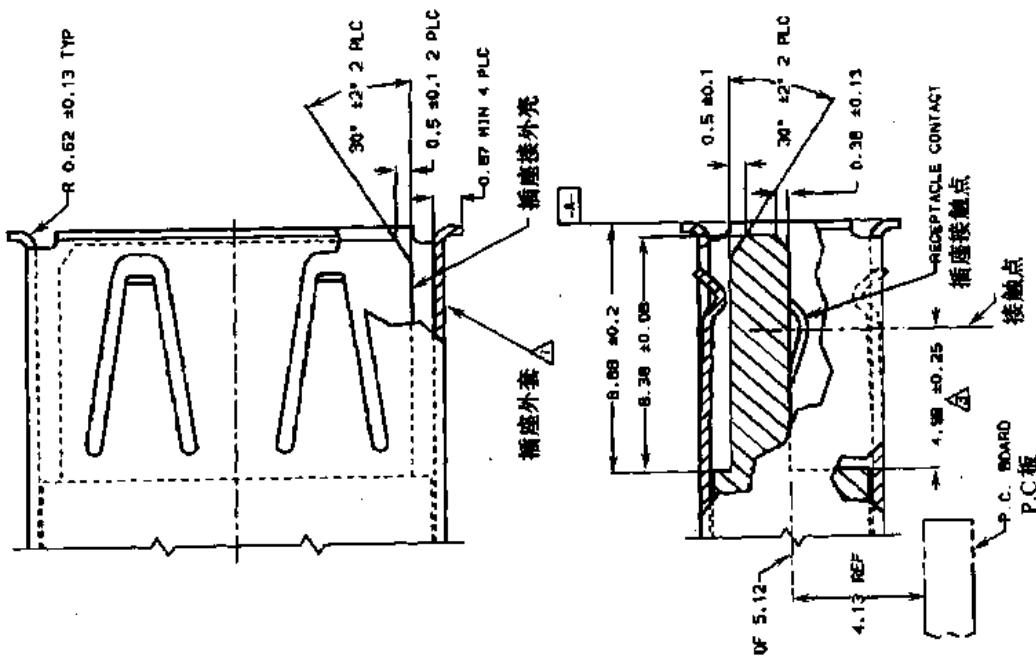


图 6-2 插座(A 系列)

超量浏览器提醒您：  
使用本复制品  
请尊重相关知识产权！

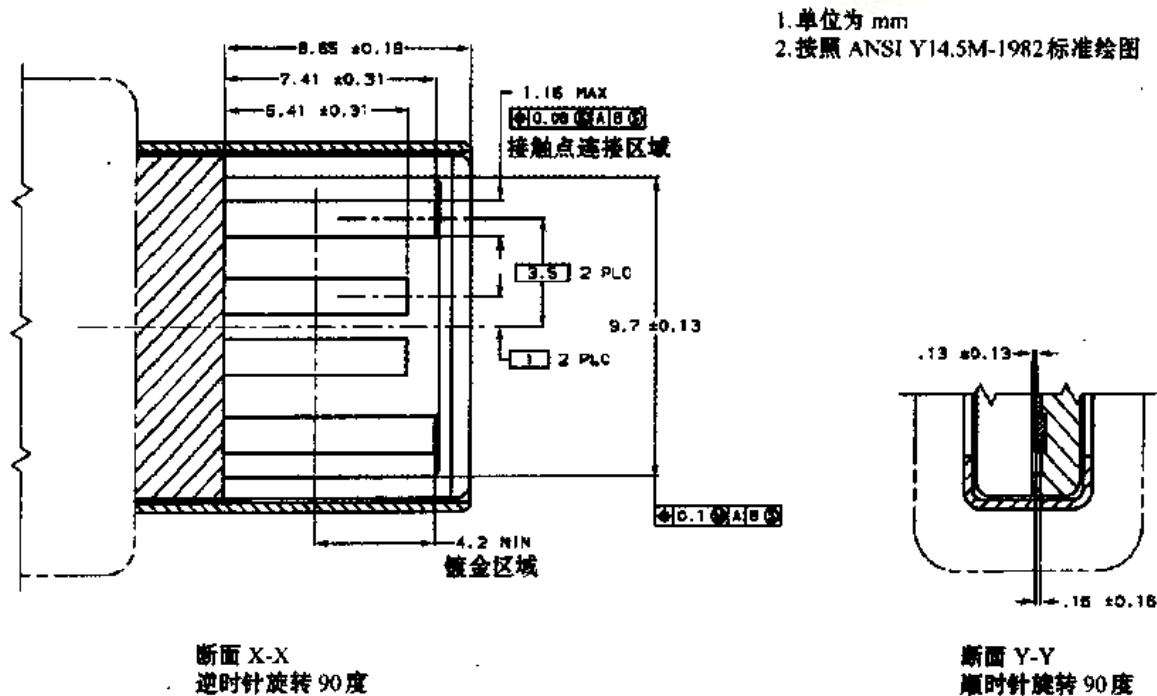
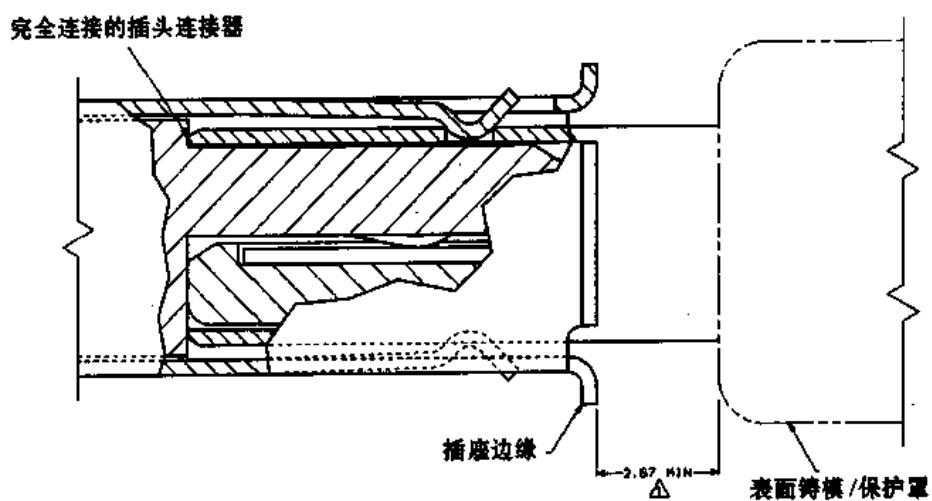


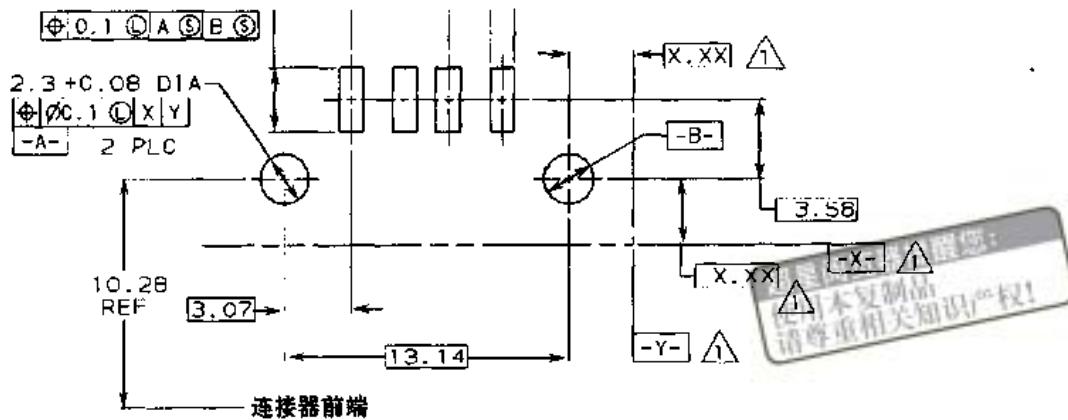
图 6-3 插头连接点细节(A 系列)



注：

- (1) 在图 6-1 和图 6-3 中所描述的尺寸中，允许插座边缘和表面铸模/保护罩。
- (2) 之间有一个最小为 2.67mm 的缝隙。

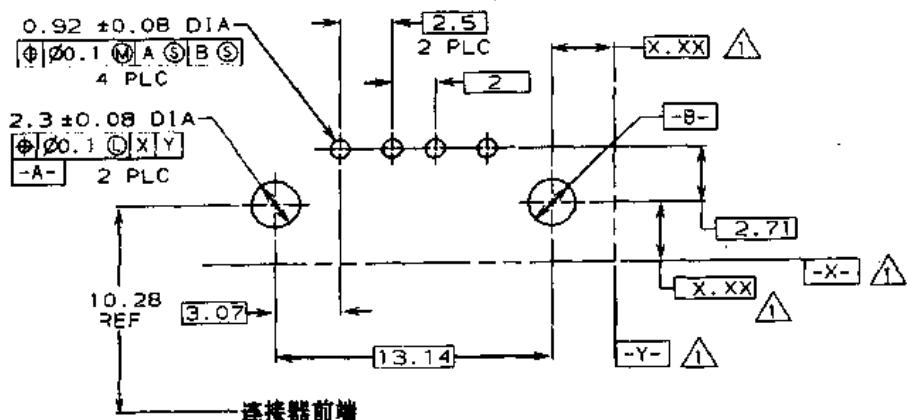
图 6-4 连接器的连接特征



注：

- (1) △代表资料和基本尺寸由用户建立。
- (2) 建议PC板厚度为1.57mm。
- (3) 按照ANSI Y14.5M-1982标准绘制。

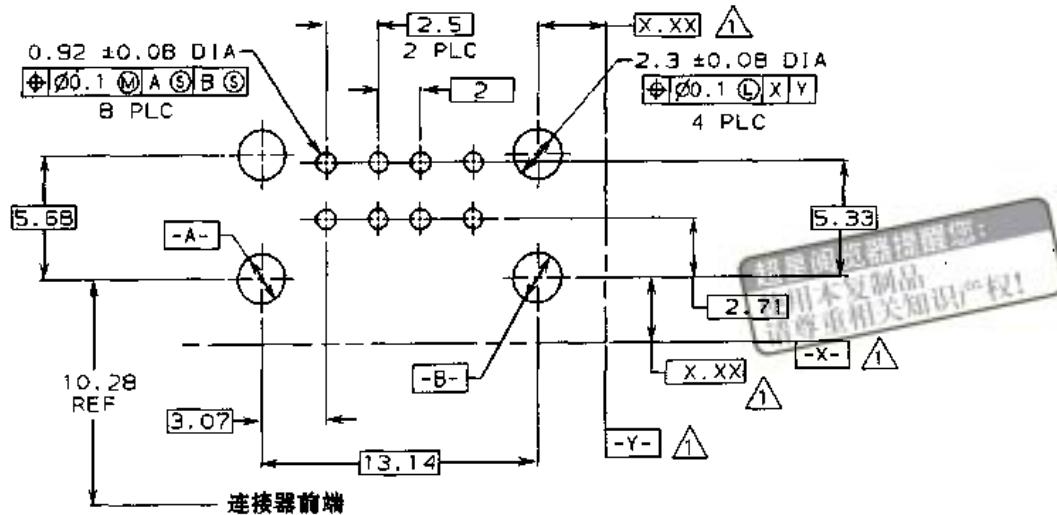
图 6-5 插头 PWB 管脚图, SMT(A 系列)



注：

- (1) △代表资料和基本尺寸由用户建立。
- (2) 建议PC板厚度为1.57mm。
- (3) 按照ANSI Y14.5M-1982标准绘制。

图 6-6 插头 PWB 管脚图, 通孔(A 系列)



注：

- (1) △代表资料和基本尺寸由用户建立。
- (2) 建议 PC 板厚度为 1.57mm。
- (3) 按照 ANSI Y14.5M-1982 标准绘制。

图 6-7 插头 PWB 管脚图, 直角重叠(A 系列)

### 6.3.3 连接器(B 系列)

图 6-8 至图 6-12 给出了 B 系列连接器的图示。

#### 1. 插头(B 系列)

与 6.3 节中提到的一样,B 系列的 USB 插头是具有屏蔽外壳的四针插头。下面所采用的方针将确保其互连能力。建议外保护套采用乳白色, 内部可以采用乳白色或类似的颜色。

#### 2. 插座(B 系列)

具体内容请参见图 6-9。

#### 3. 连接器连接特性(B 系列)

图 6-11 说明了 B 系列连接器的连接特性。

#### 4. 插座 PWB 管脚图(B 系列)

图 6-12 说明了 B 系列插座的 PWB 管脚图。

### 6.3.4 串行总线图标

图 6-13 给出了 USB 图标, 应该将其烧铸到连接器内, 并放置到产品上以便于识别 USB 端口。在连接插头和插座时, 建议使产品上的图标和插头上的图标彼此相邻。A 系列和 B 系列都可以使用该图标。在插头上, USB 图标的周围应该是一个 0.635mm 的矩形凹槽, 从而使得图标更加清晰。

4.3 REF 整体电缆连接器装配

注：(1)尺寸单位为mm  
(2)由算

- 在接触的地方，是在量少为  $1.25\mu\text{m}$  厚的镍上的镍金，至少应为  $0.75\mu\text{m}$ 。
  - 在接触的地方，是在量少为  $1.25\mu\text{m}$  厚的镍上的镍金，为  $0.75\mu\text{m}$  的钯上，还要有量少为  $0.05\mu\text{m}$  厚的镍金。
  - 在接触的地方，是在量少为  $1.25\mu\text{m}$  厚的镍上的镍金，为  $0.75\mu\text{m}$  的钯合金上，还要有量少为  $0.05\mu\text{m}$  厚的镍金。
  - 在接触点采用下列方式之一：

(3) 按照 ANSI Y14.5M-1982 标准绘图。

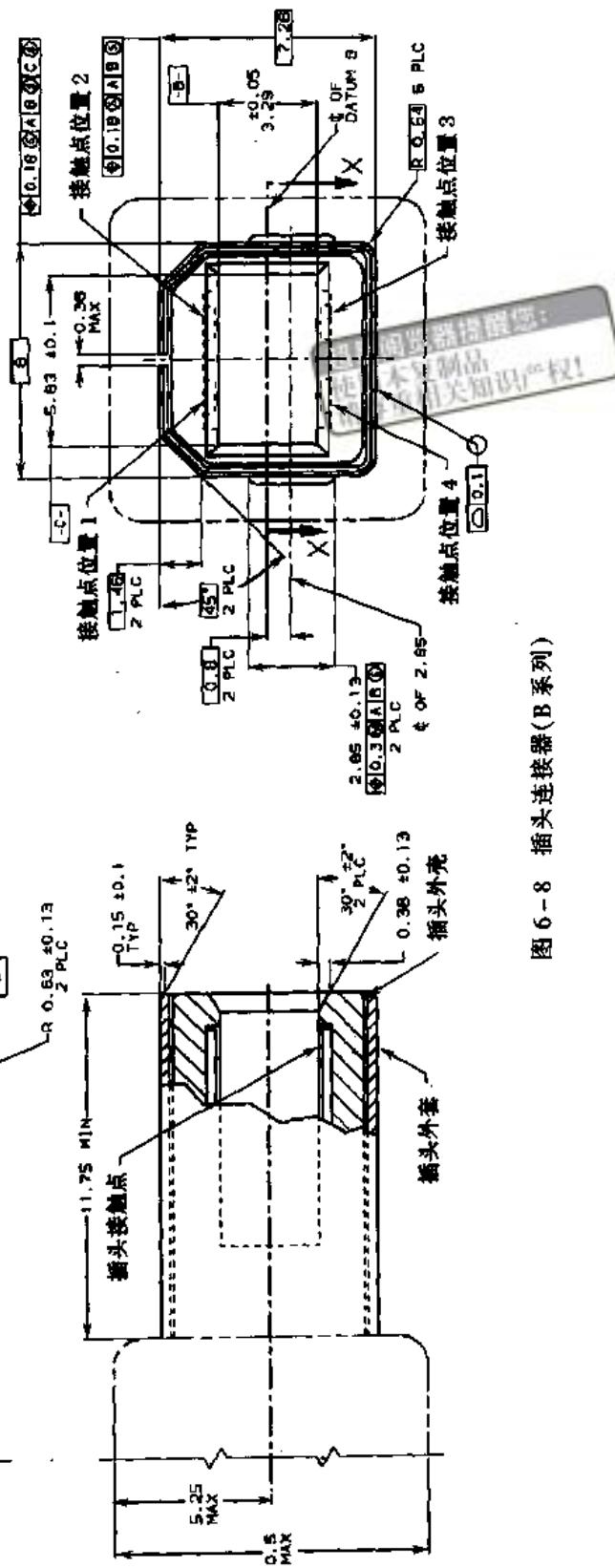


图 6-8 插头连接器(B 系列)

- 注：
- (1) 尺寸单位为 mm
  - (2)  $\triangle$  代表电镀：
    - 插头接触点(采用下列方式之一)：
      - 在接触的地方，覆盖在最少为 1.25μm 厚的镍上的镍金，至少应为 0.75μm
      - 在接触的地方，覆盖在最少为 0.05μm 厚的镍上的镍金，为 0.75μm 的钯上，还要有量少为 0.05μm 厚的镍金
      - 在接触的地方，覆盖在最少为 1.25μm 厚的镍上的镍金，为 0.75μm 的钯镍合金上，还要有量少为 0.05μm 厚的镍金
      - 插头外套：至少应有 2.5μm 的镍或以镍为主的合金覆盖，在适当的底板上
    - (3)  $\triangle$  代表在一个连接器内的四个连接点必须处在一个 0.2 的区域内
    - (4)  $\triangle$  代表材料：插座外套——磷青铜，UNS C5110，或其它合适的材料
    - (5) 按照 ANSI Y14.5M-1982 标准绘图

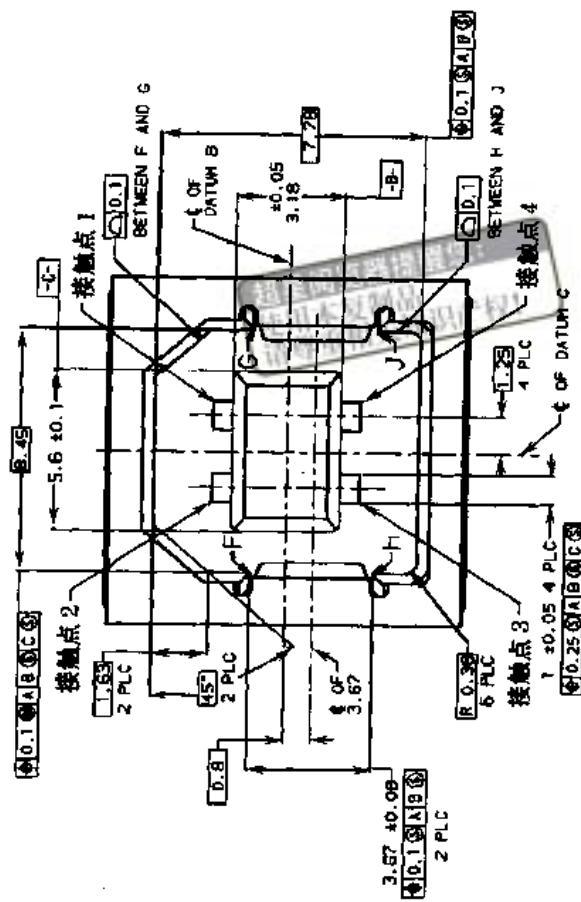
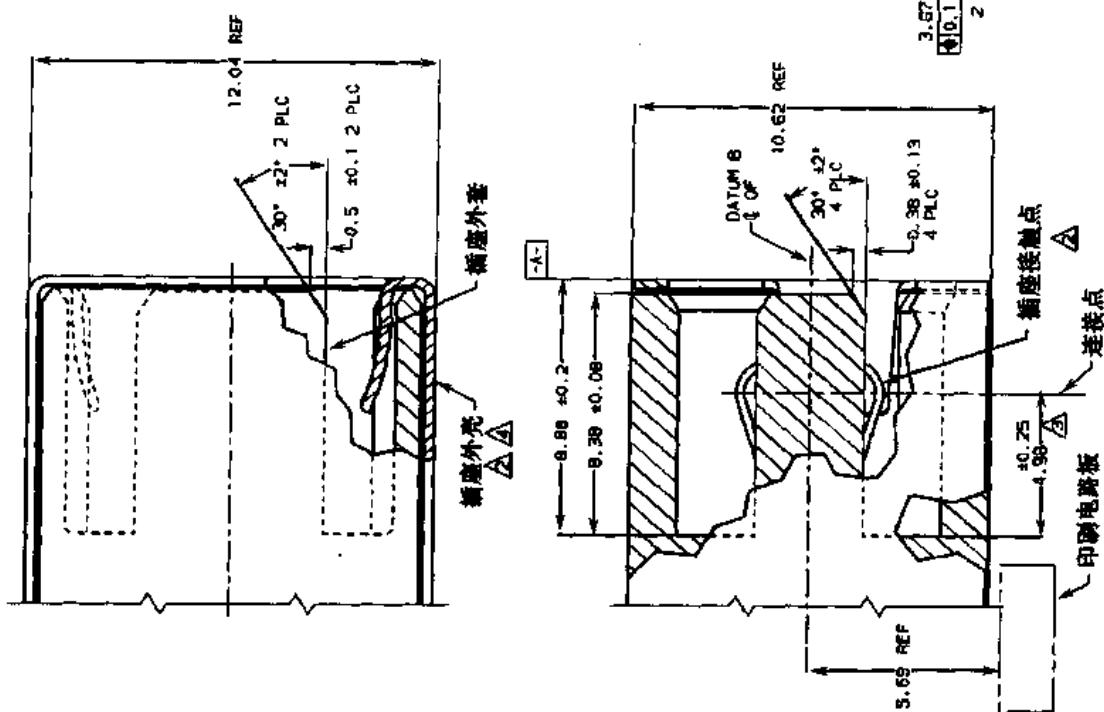


图 6-9 插座(B 系列)

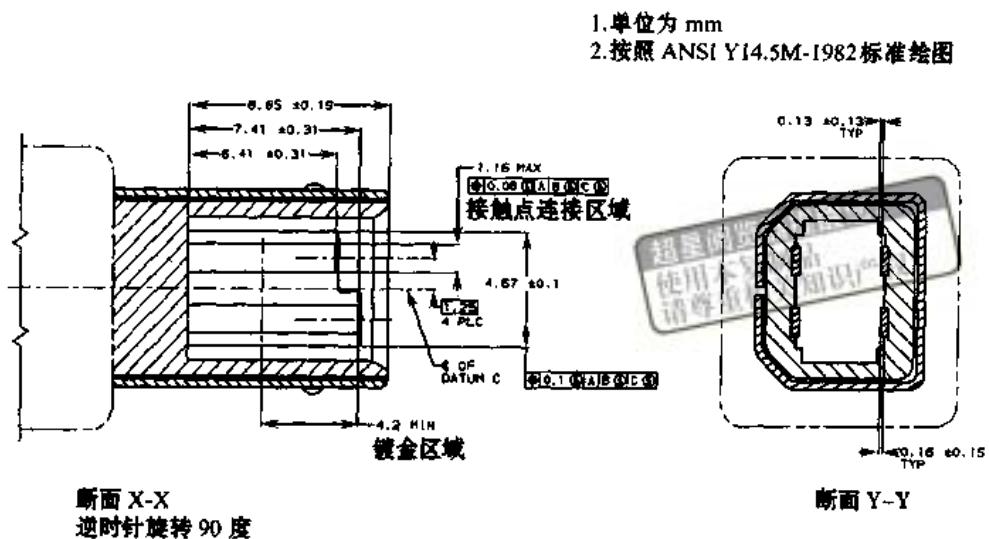


图 6-10 插头连接点细节(B 系列)

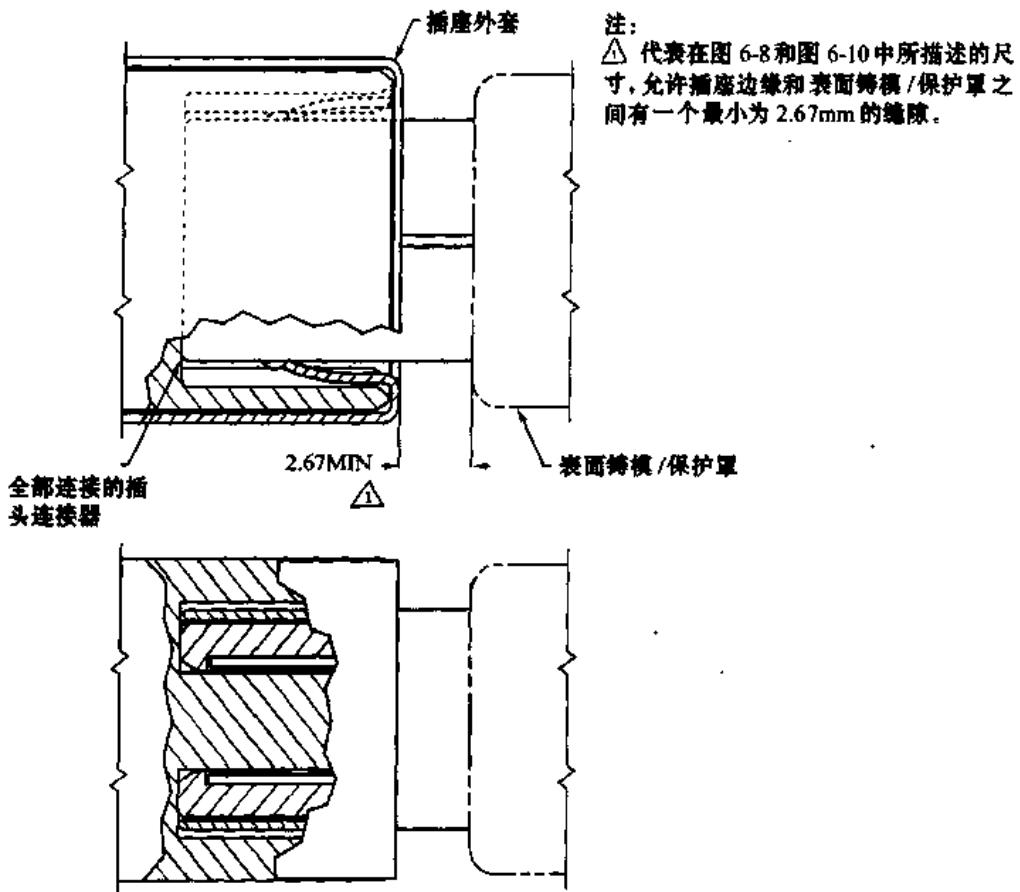
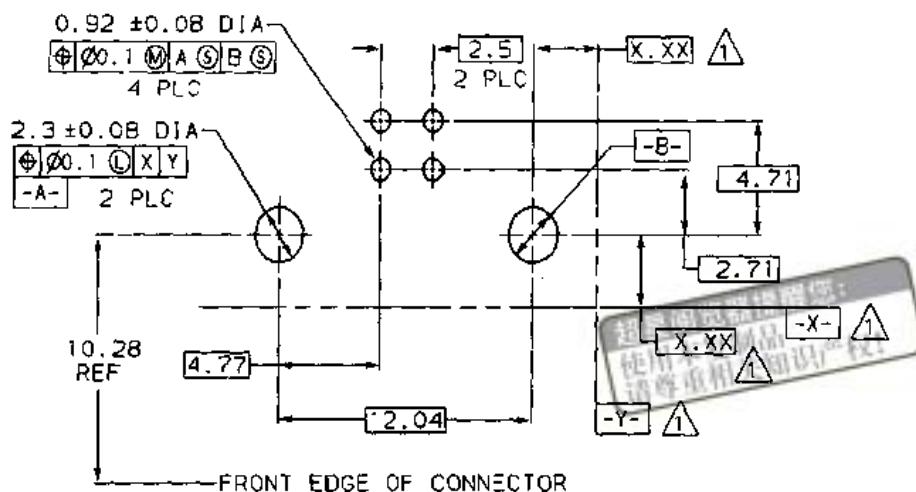


图 6-11 连接器连接特性(B 系列)



注：

- (1) △代表资料和基本尺寸由用户建立。
- (2) 建议 PC 板厚度为 1.57mm。
- (3) 按照 ANSI Y14.5-1982 标准绘制。

图 6-12 插座 PWB 管脚图,通孔(B 系列)



图 6-13 USB 图标原图

### 6.3.5 插头/插座机械和电气要求

#### 1. 连接编号方式

连接编号方式如表 6-6 所列。

表 6-6 连接编号方式

接点编号	信号名称	注释
1	VCC	电源线
2	- Data	
3	+ Data	
4	Ground	地线

#### 2. 额定值

额定值如下：

- (1) 电压: 30V<sub>ac</sub> (rms)。

(2) 电流:每个接点的最大电流为1A,温度升高不能超过30℃。

(3) 温度:储藏温度-40℃至60℃;工作温度0℃至40℃。

### 3. 性能和测试说明

必须按照表6-7中列出的电气、机械和环境性能要求来设计产品。除非另外声明,所有的测试都必须在一定的周围环境条件下进行。测试报告中必须包含测试所使用的电缆结构和/或部分编号。

表6-7 测试要求和过程概要

测试说明	要求	测试过程
产品检测	满足5.3节中的要求	符合视觉、空间和功能的要求
电气方面		
终端阻抗	最大为30 mΩ	EIA 364-23 对于外壳中装配好的已连接的接点,使其开路时最大电压为20mV,最大电流为100mA。参见图6-14
绝缘电阻	最小为1000 MΩ	EIA 364-21 在未连接和已连接的连接器部件的相邻连接点之间的测试结果
绝缘体抗压能力	在海平面为750 Vac (均方根值)	EIA 364-20 在未连接和已连接的连接器部件的相邻连接点之间的测试结果
电容	最大为2pF	EIA 364-30 在1kHz时,在未连接的连接器中相邻的电路之间的测试结果
机械方面		
随机振动	中断持续时间不能超过1μs或更长。 参见注释	EIA 364-28 条件V,测试号A。将已连接的连接器置于均方根为5.35G的环境中。在三个相互垂直的平面中每一个都进行15分钟。参见图6-15
物理振动	中断持续时间不能超过1μs或更长。 参见注释	EIA 364-27 条件H。将已连接的连接器置于持续时间为11ms的30个半正弦振动脉冲的环境中。沿着三个相互垂直的平面,每一个方向都进行三次振动。关于测试装备请参见6-15
持久性	参见注释	EIA 364-09 以每小时最大为200次的速率,对连接器部件进行1500次的连接和拆除操作
连接力	最大为35N	EIA 364-13 以每分钟12.5 mm的最大速率,测量连接器部件进行连接时所必须使用的力
拔除力	最小为10N	EIA 364-13 以每分钟12.5 mm的最大速率,测量拔除连接器部件时所必须使用的力
电缆保持力	在对电缆进行弯曲时,电缆不应该发生移位	对电缆沿轴向施加25N的力

(续)

测试说明	要 求	测 试 过 程
环 境 方 面		
热抖动	参见注释	EIA 364 - 32 测试条件 I, 使已连接好的连接器经受五次从 -55°C 至 85°C 的温度变化
湿度	参见注释	EIA - 364 - 31 方法 II, 测试条件 A。将已连接的连接器放置在温度为 40°C, 相对湿度为 90% 至 95% 的环境中 96 个小时
温度寿命	参见注释	EIA - 364 - 17 测试条件 3, 方法 A。使已连接的连接器在 80°C 环境中的温度寿命达到 250 小时

注: 要满足直观要求, 没有出现物理损坏, 并且应该满足表 6-8 中说明的测试顺序中给出的其它测试要求。

表 6-8 产品限定测试顺序

测试或检验	测试组 (a)		
	1	2	3
测试顺序 (b)			
产品检验	1, 10	1, 5	1, 9
终端阻抗	3, 7	2, 4	
绝缘阻抗			3, 7
绝缘体抗压			4, 8
电容			2
振动	5		
物理振动	6		
持久性	4		
连接力	2		
拔除力	8		
热抖动			5
湿度			6
电缆保持力	9		
温度寿命		3(c)	

注:

- (a) 请参考下面第 4 小节“实例选择”。
- (b) 数字指出了进行测试的顺序。
- (c) 前提是经过 10 个周期考验后所得的样值。

#### 4. 样品选择

应该根据正确的厂商说明来准备样品, 图 6-14 和图 6-15 分别给出了有关终端阻抗的测量点, 以及用于固定被测样品的夹具的示意图。而且可以从当前的产品中随机地

选出。测试组1、2和3应该最少包含8个连接器，并且每一个测试组最少应该挑选和标识出30个接触点。除非另外声明，这些接触点都会用于所有的测量。

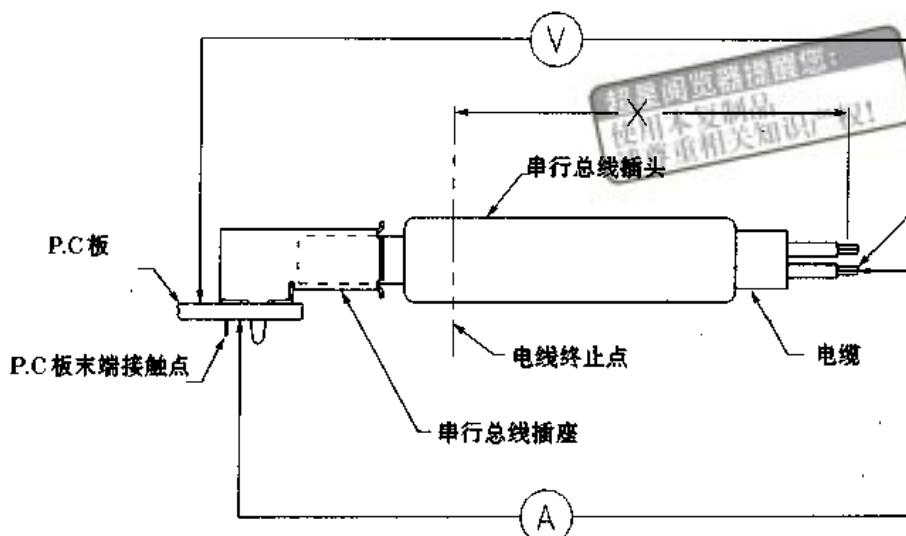


图 6-14 终端阻抗测量点

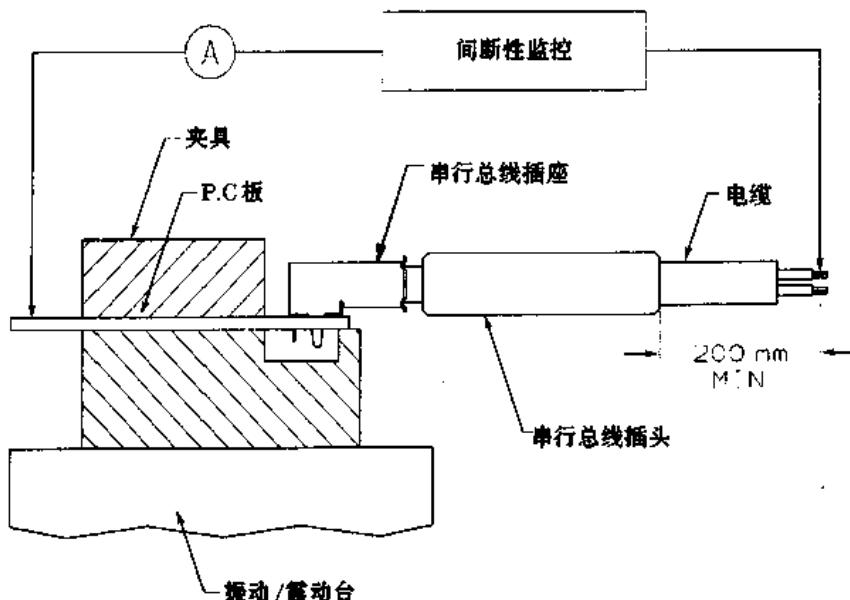


图 6-15 振动和物理振动固定夹具

## 5. 其它要求

其它方面的要求包含以下几个方面。

- (1) 易燃性：所使用的塑料材料按照 UL - STD - 94，应该可以达到 94V - 0 这一等级。
- (2) 标记：在插头上应该有图 6-13 所示的标记。建议 OEM 在最终产品上，靠近插座的可能的或实用的地方加上一个标记。
- (3) 限定：在需要的时候，所有的供应商都应能提供适当的资料来显示其符合了本

章所提出的要求。

## 6.4 电缆压降要求

USB 物理层协议要求进行最大配电时,两个集线器之间或集线器和功能设备之间下降的电压最大应为 350mV。表 6-9 列出了对于每一种规格的导线配电电缆线路的标称长度。下面是一个用于计算到达一个未得到电源供应的集线器的压降的公式。

$$V_{unpowered\_hub} = V_{switch} + 4 \times V_{connector} + 2 \times V_{cable}$$

其中:  $V_{switch} = I_{max} \times (\text{接线板阻抗和 FET 阻抗}) = 100 \text{ mV}$  (由定义所得的最大值)

$$V_{connector} = I_{max} \times 30 \text{ m}\Omega (\text{连接器阻抗}) = 15 \text{ mV}$$

$$V_{cable} = I_{max} \times \text{电缆阻抗}$$

$$I_{max} = 500 \text{ mA}$$

由上述公式,假定在电缆装置中使用了两个连接器,则可以计算出  $V_{cable} \approx 95 \text{ mV}$ 。

对于在 20℃ 条件下使用铜线而产生的 95 mV 压降,表 6-9 列出了电流为 500mA 时电缆的长度。图 6-16 说明了压降分布情况。

表 6-9 电缆长度(与规格相对应)

规 格	阻 抗	长 度(最大值)
28	0.232 Ω/m	0.81 m
26	0.145 Ω/m	1.31 m
24	0.091 Ω/m	2.08 m
22	0.057 Ω/m	3.33 m
20	0.036 Ω/m	5.00 m

注:本表没有包括额外的温度影响(约为 10%)。

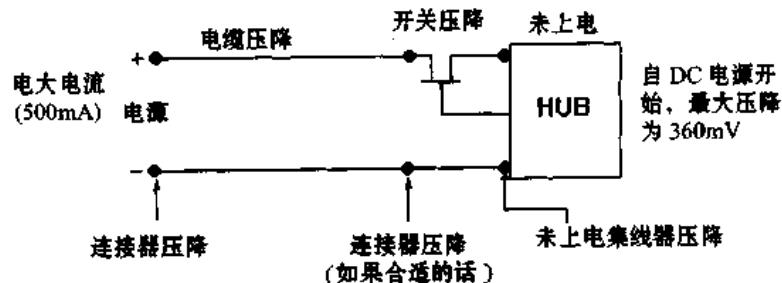


图 6-16 电缆和连接器压降分布

建议每一个单独的 USB 电缆生产商都应该核实正确的 DC 压降要求。如果生产商使用了不同于前面所提及的材料,他就应负责在未得到电源供应的集线器上能够实现正确的 DC 电压。

为了满足本规范对电缆最大长度为 5m 的要求,DC 配电导线应该处于 20AWG 至 28AWG 范围之内。

注释:对于不需要 500mA 的某个功能模块而言,按照压降要求使用规格更小的导线会比较恰当。

## 6.5 传播时延

如果你选择的敷设电缆不能满足第 6.3.1 小节中的第 2 小节中的要求,那么就需要使用表 6-10 所列的内容来限制用于全速率信道的电缆长度。

表 6-10 传播时延(与电缆长度相对应)

电缆传播时延规范	最大电缆长度
9.0 ns/m	3.3 m
8.0 ns/m	3.7 m
7.0 ns/m	4.3 m
6.5 ns/m	4.6 m

注:实际应用必须使用可以满足 6.3.1 小节中第 2 小节,第 6.4 和第 6.5 节中所要求的最短的电缆。

## 6.6 接地技术

对于完整的装置而言,屏蔽必须到连接器插头才能终止。在主机端,屏蔽、DC 电源和机箱地线应该焊接在一起。整个总线在主机端应该只有一个 DC 接地点。其它所有的设备都不应该连接屏蔽或向机箱地线返回 DC。这样可以阻止低频电流通过。但是,为了服从 EMI 要求,可以允许 AC 耦合。耦合阻抗在 60Hz 时必须小于  $250\text{k}\Omega$ ,而在 3Hz 到 30Hz 之间时不应大于  $15\Omega$ 。电容器的电介质额定电压必须为 250V<sub>ac</sub>(均方根值)。

## 6.7 信息调整

经适当的局部调整来安装这一电缆线路的有关建议和原则,都应该由 OEM 来提供。建议像 EIA CB8-198[4]和 ANSI/NFPA 70-1984 这样的准则以及局部代码和规则都应该得到满足。

## 第7章 USB总线电气特性

本章说明了USB电气规范。它包括信息、功率分配和物理层规范。

本章具体包括以下内容：

- USB总线信号；
- USB驱动器特性；
- 接收器特性；
- 信号电平；
- 数据编码/解码；
- 比特填充；
- 同步方式；
- 起始的帧时间间隔和帧调整能力；
- 数据信号速率；
- 数据信号上升和下降时间；
- 数据源信号；
- 集线器信号时序；
- 接收器数据抖动；
- 电缆时延；
- 总线转向时间/分组间时延；
- 端到端最大信号时延；
- 功率分配；
- USB设备类型；
- 物理层。

## 7.1 信 号

下面几小节中将讨论 USB 信号规范。

### 7.1.1 USB 驱动器特性

USB 使用一个差模输出驱动器来向 USB 电缆传送 USB 数据信号。在低输出状态，驱动器稳态输出的变化幅度必须使 VOL 小于 0.3V，此时要有  $1.5k\Omega$  负载加到 3.6V 电源上；在高输出状态，驱动器稳态输出的变化幅度必须使 VOH 大于 2.8V，此时在地上有  $15k\Omega$  负载，如 7.3.2 节中的表 7-4 所示。差模高输出状态和低输出状态之间输出的变化幅度必须很好地进行平衡，从而将信号偏差减至最小。另外，还需要驱动器上的摆动速率控制功能把辐射噪声和串话减至最小。驱动器输出必须支持三态操作，以此来进行双向半双工通信。同时还需高阻抗来将那些正在进行热插入操作或已经连接了但电源却没有接通的下行设备同端口隔离开来。相对于设有损坏的局部参考地而言，驱动器必须能承受信号管脚上的 -0.5V 至 3.8V 的电压。当驱动器处于工作状态和正在驱动信号时，它必须能够承受这一电压达 10.0ms，并且能够承受当驱动器处于高阻状态时所产生的不确定条件。

#### 1. 全速率(12Mb/s)驱动器特性

一个全速率连接可以利用一个屏蔽的双绞线对来实现，该电缆特性阻抗  $Z_0$  为  $(90 \pm 15\%) \Omega$ ，最大长度为 5m。每一个驱动器的阻抗必须位于  $19\Omega$  和  $44\Omega$  之间。数据信号上升和下降时间必须在 4ns 和 20ns 之间，平稳地上升或下降(单调的)，并且能得到很好地匹配，以便将 RFI 辐射和信号偏差降至最小。

如果使用的是 CMOS，驱动器阻抗一般应由一个阻抗小于该阻抗值的 CMOS 驱动器，以及起补偿平衡作用的一系列离散电阻来实现。图 7-1 给出了一个全速率驱动器实例，它使用了两个完全相同的 CMOS 缓冲器，这些 CMOS 缓冲器的输出阻抗位于  $3\Omega$  和  $15\Omega$  之间，并接有两个  $27\Omega$  电阻。图 7-2 给出了全速率驱动器信号波形。

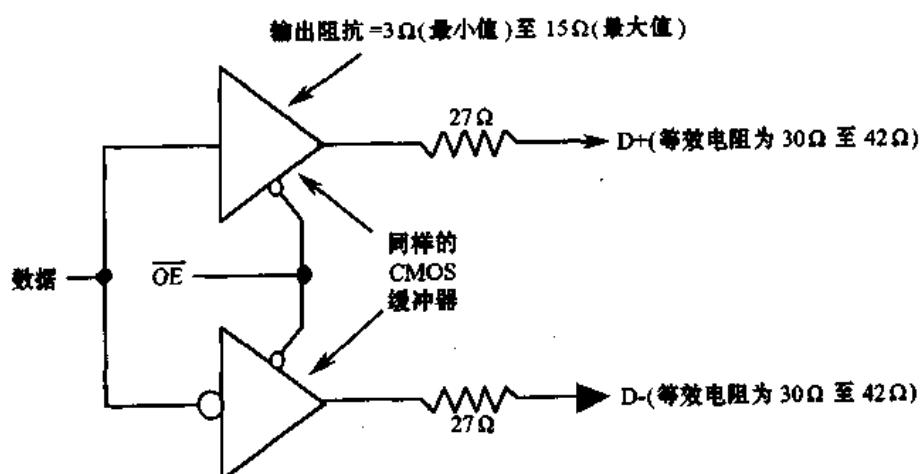


图 7-1 全速率 CMOS 驱动器电路实例

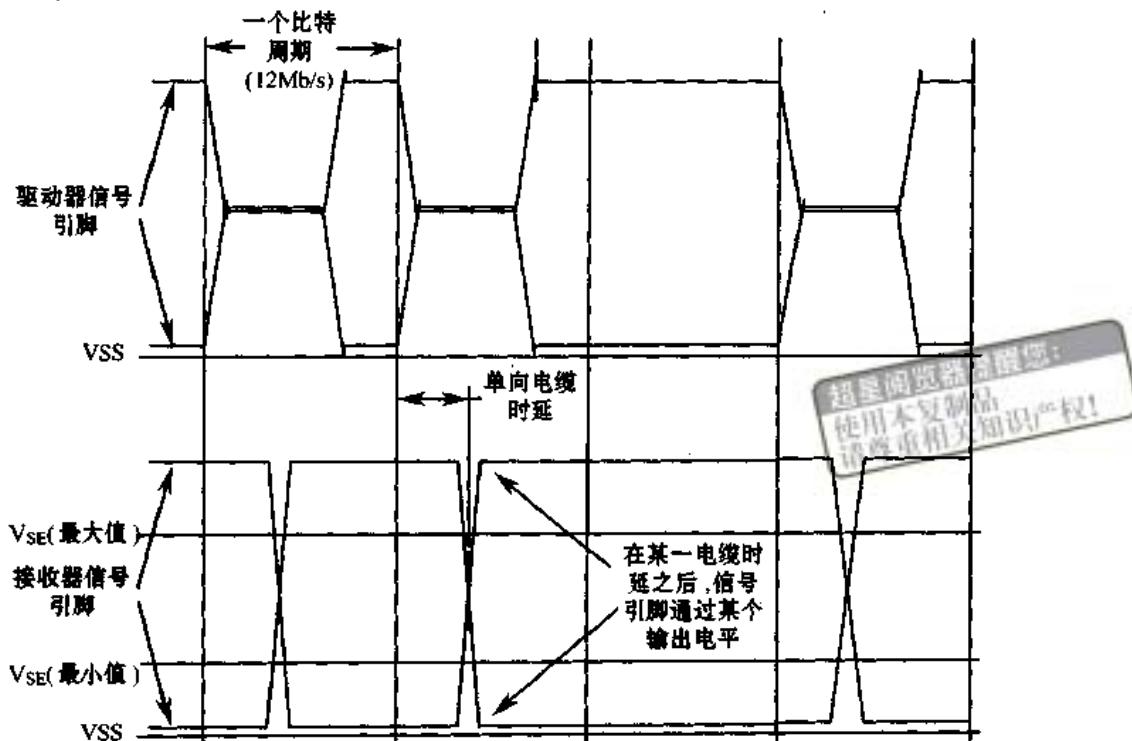


图 7-2 全速率驱动器信号波形

## 2. 低速率(1.5Mb/s)驱动器特性

一个低速率连接可以利用一个非屏蔽的非双绞线对来实现，其最大长度为 3m。该电缆上的上升和下降时间必须大于 75ns，以使 RFI 辐射低于 FCC 类型 B 规定的限制；还要小于 300ns，以限制时序延迟和信号偏差及变形。在驱动电缆时，驱动器必须以平滑的上升和下降时间、微小的反射和阻尼振荡来达到所要求的稳态信号电平（参见图 7-3）。该电缆只能在位于低速率设备和与之相连的端口之间所形成的网络部分使用。

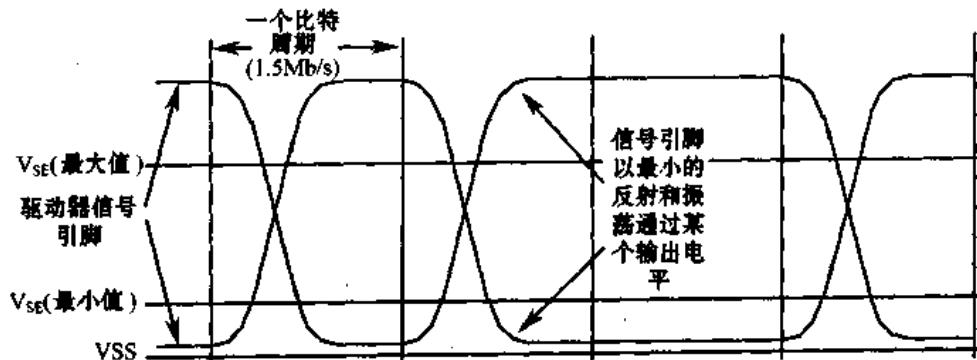


图 7-3 低速率驱动器信号波形

## 3. 驱动器用法

全速率缓冲器应用在所有集线器和全速率功能设备的上行端口（通向主机）上。所有带有集线器的设备都必须是全速率设备。一个全速率驱动器既可以以全数据速率，也可以以低数据速率来发送数据。但是，信号却总是使用全速率信号约定（参见表 7-1）和边缘变化率。以低速数据速率运行并不会改变设备的特性。

低速率缓冲器应用于低速率功能设备的上行端口。所有集线器上的下行端口都要符合两种驱动器的特性，从而使得任一类型的设备都可以插入这些端口（如图7-5和图7-6所示）。低速率设备使用低速信号约定（参见表7-1）和边缘变化率，只能以低速数据速率发送数据。

超星阅读器专用  
使用本资源请  
尊重相关知识产权！

### 7.1.2 接收器特性

接收USB数据信号时必须利用一个差模输入接收器。当两个差模数据输入以地电位作为参考，并且处于至少为0.8V至2.5V这样的范围之内时，接收器具有的灵敏度至少应为200mV，这称为共模输入电压范围。当差模信号线不在共模范围之内时，也要求能进行正确的数据接收，如图7-4所示。如果在没有损坏并以本地地电位作为参考的条件下，接收器所能承受的稳态输入电压应该位于-0.5V至3.8V之间。另外对不同的接收器而言，每一条信号线都必须有一个单端接收器。这些接收器必须具有一个位于0.8V和2.0V之间（TTL输入）这样的开关阈值电压。建议单端接收器具有某些滞后作用，以减小其对噪声的灵敏度。

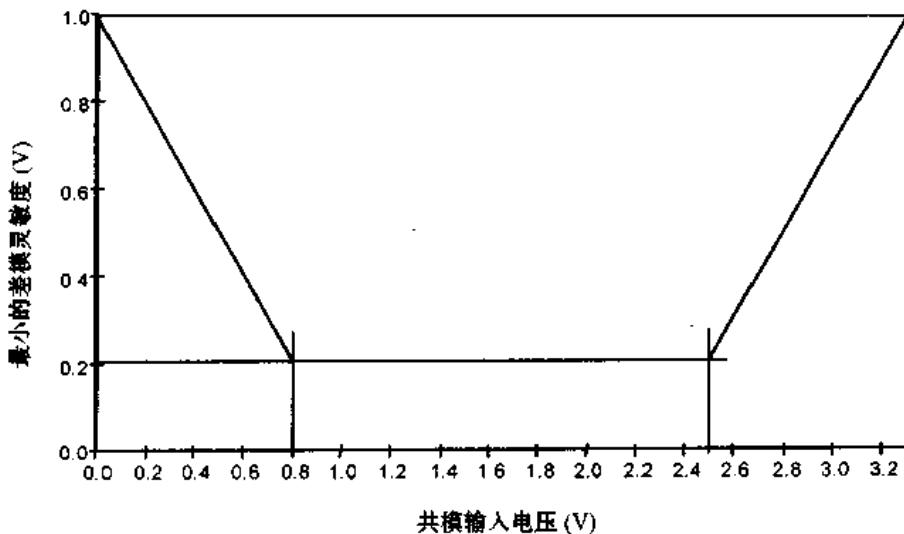


图7-4 差模输入灵敏度（超过了整个共模范围）

### 7.1.3 信号终端

USB是在集线器和功能模块端终止，如图7-5和图7-6所示。全速率和低速率设备的区别在于，位于电缆下行端点处的上拉电阻的位置不同。全速率设备中上拉电阻位于D+线上，如图7-5所示，而低速设备的上拉电阻位于D-线上，如图7-6所示。

这种上拉终端是一个与相对于本地参考地电位，处于3.0V和3.6V这样的范围之内的电压源相连，电阻值为 $1.5\text{k}\Omega \pm 5\%$ 的电阻。而下拉终端是与其地线相连的电阻值为 $15\text{k}\Omega$ 的电阻。

### 7.1.4 信号电平

表7-1列出了USB信号电平。本节和以后几节我们都将用来讨论信号电平。

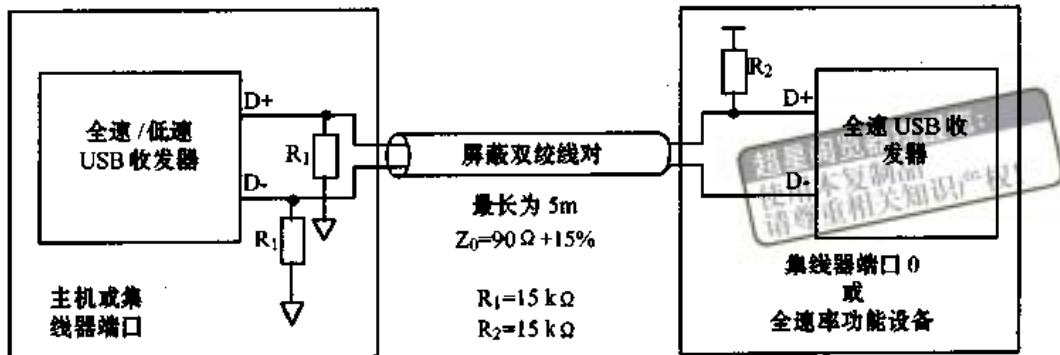


图 7-5 全速率设备电缆和电阻连接

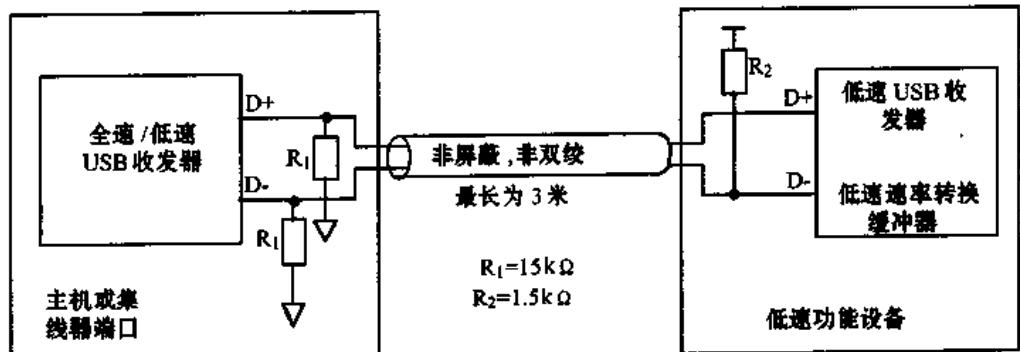


图 7-6 低速率设备电缆和电阻连接

J 和 K 数据状态是系统中用于差模数据通信的两个逻辑电平。差模信号是在数据线信号交叉点处进行测量的。只要信号的交叉电平位于共模范围之内，差模数据信号就与信号的交叉电平无关。当总线不处在差模信号方式时，数据线必须处在 V<sub>SE</sub> 开关阈值范围之外才有效，就像在空闲和重新开始状态时一样。注意，空闲和重新开始状态在逻辑上分别等同于 J 和 K 状态。

表 7-1 信号电平

总线状态	信 号 电 平	
	始发端驱动器	在接收器端
差模“1”	$(D+) - (D-) > 200\text{mV}$ 并且 $D+ > V_{SE}$ (最小值)	
差模“0”	$(D+) - (D-) < -200\text{mV}$ 并且 $D+ < V_{SE}$ (最小值)	
数据 J 状态：		
低速率	差模“0”	
全速率	差模“1”	
数据 K 状态：		
低速率	差模“1”	
全速率	差模“0”	
空闲状态：		
低速率	差模“0” 和 $D- > V_{SE}$ (最大值) 并且 $D+ < V_{SE}$ (最小值)	
全速率	差模“1” 和 $D+ > V_{SE}$ (最大值) 并且 $D- < V_{SE}$ (最小值)	
重新开始状态：		
低速率	差模“1” 和 $D+ > V_{SE}$ (最大值) 并且 $D- > V_{SE}$ (最小值)	
全速率	差模“0” 和 $D- > V_{SE}$ (最大值) 并且 $D+ > V_{SE}$ (最小值)	

(续)

总线状态	信号电平	
	始于发端驱动器	在接收器端
分组开始(SOP)	数据线由空闲状态变换为K状态	
分组结束(EOP)	D+和D- $< V_{SE}$ (最小值)应持续2个比特周期 <sup>①</sup> ,再跟上一个比特周期的空闲状态	D+和D- $< V_{SE}$ (最小值)应至少持续1个比特周期 <sup>②</sup> ,再跟上一个J状态
断开 (仅对上行方向而言)	(n.a.)	D+和D- $< V_{SE}$ (最大值)应至少持续2.5μs
连接 (仅对上行方向而言)	(n.a.)	D+或D- $> V_{SE}$ (最大值)持续时间 $\geq 2.5\mu s$
复位 (仅对下行方向而言)	D+和D- $< V_{SE}$ 持续时间 $\geq 10ms$	D+和D- $< V_{SE}$ (最小值)持续时间 $\geq 2.5\mu s$ (必须在5.5μs之内识别出来) <sup>③</sup>

① EOP的宽度由和数据传输速率有关的比特周期来决定。  
 ② EOP由和接收EOP的设备的类型有关的比特周期来决定。  
 ③ 这些周期适用于一个不处在挂起状态的有效设备。

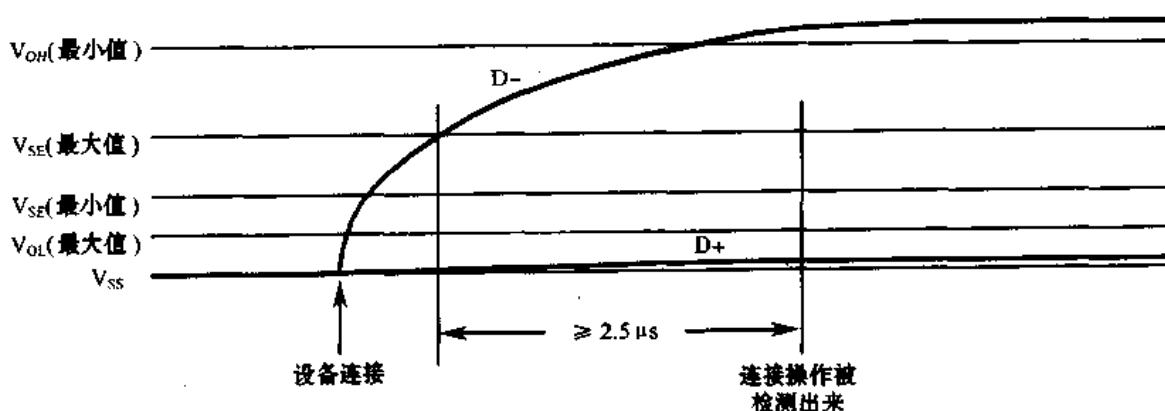
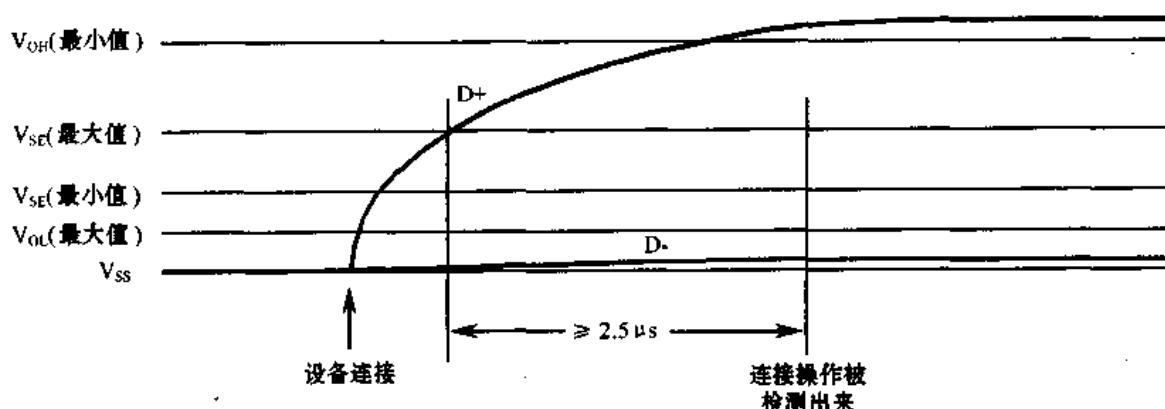
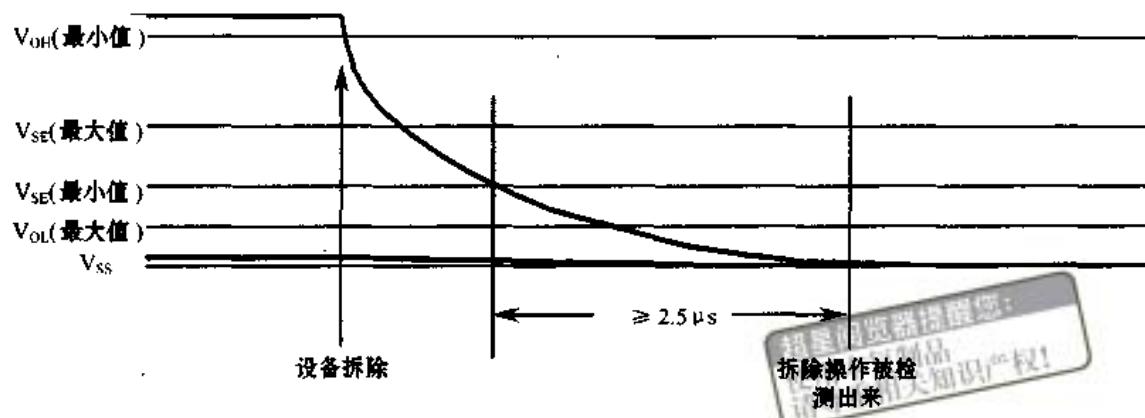
表7-1所列的全速率的J和K状态正好与低速率的状态相反。对数据空闲和重新开始信号的识别是由接入一个端口设备的类型所决定的。如果是一个全速率设备接入了一个端口,即使数据是以低数据速率通过信号线进行传送的,USB网络段也会使用全速率信号约定(和快速上升和下降时间)。表7-1所示的低速信号约定仅在一个低速设备和它所连接的端口之间使用。下节就介绍用于确定设备类型的方法,以及由此而使用的信号约定。

### 1. 连接和断开信号

在主机或集线器下行方向的所有端口,在D+和D-线上都有上拉电阻。所有的设备其上行端口数据线中的一根具有上拉电阻。设备类型将决定哪一条数据线具有上拉电阻。全速率设备的D+线上有一个上拉电阻(参见图7-5),而低速率设备的D-线上具有上拉电阻(参见图7-6)。当一个设备插入了集线器或主机,但数据线却并未被驱动时,这些电阻可以在信号线上产生一个静止的偏压条件,使得带有上拉电阻的信号线高于2.8V,而另一条线的电压接近于地电位。这称之为**空闲状态**。

当没有功能设备接入主机或集线器的下行端口时,或者接入设备的上拉电阻没有得到供电,下拉电阻可以使D+和D-信号线上的电压降至主机或集线器端口的单端低阈值。这时将在下行端口上产生一个称之为**单端0**的状态。如果SE0在一个下行端口上持续时间超过了2.5μs(30个全速率比特周期),就产生一个**断开指示**。注意断开信号只能应用于上行方向(参见图7-7)。

当一个设备同主机或集线器相连时,就会检测到一个连接条件,并且其中一条数据线将被拉至单端高阈值之上的电平超过2.5μs(30个全速率数据比特周期)。当端口状态从断开变为连接时,处于高电位的数据线将把这段总线置为空闲状态,并确定所连接的设备是一个全速率设备还是一个低速率设备。一旦确定了空闲状态,那么表7-1中给出的所有信号电平都是为这个网段所设(并且只为这个网段)。图7-8给出了一个全速率设备的连接顺序,图7-9给出了一个低速率设备的连接顺序。



当所有的集线器端口得到供电时, 这些端口都是从一个隐含的断开状态开始工作。如果一个设备连接到了该端口, 这一端口就会按照上述的连接顺序来检测设备的类型和对端口的信号特性进行设置。

## 2. 数据信号

一个分组中的数据传输是由差模信号来实现的。在接收器看来, 如果 D+ 至少比 D- 高 200mV 就代表一个差值“1”, 而一个差模“0”则由 D- 至少比 D+ 高 200mV 来表

示。信号的交叉点必须位于 1.3V 和 2.0V 之间。

通过将 D+ 和 D- 信号线从空闲状态驱动到其相反的逻辑电平 (K 状态), 发送端口可以发出分组开始(SOP)信号。这一电平变化代表了 Sync(同步)域中的第一个比特。当 SOP 的传输时间小于 5ns 时, 集线器必须对跟在 SOP 后的第一个比特的宽度畸变加以限制。通过对经过集线器的标称数据时延和集线器输出有效的时延进行匹配, 可以将畸变降至最小。

单端 0 状态用来发出分组结束信号(EOP)。该状态由 D+ 和 D- 都低于 0.8V 表示。EOP 信号是通过将 D+ 和 D- 都驱动为单端 0 状态达两个比特周期, 然后再将信号线都驱动为空闲状态并持续一个比特周期来发送的。从单端 0 到空闲状态的变化标示了一个分组的结束。空闲状态保持 1 个比特周期, 然后 D+ 和 D- 驱动器都被置为高阻状态。总线终端电阻将使总线保持在空闲状态。图 7-10 给出了一个分组开始和结束时的信号。

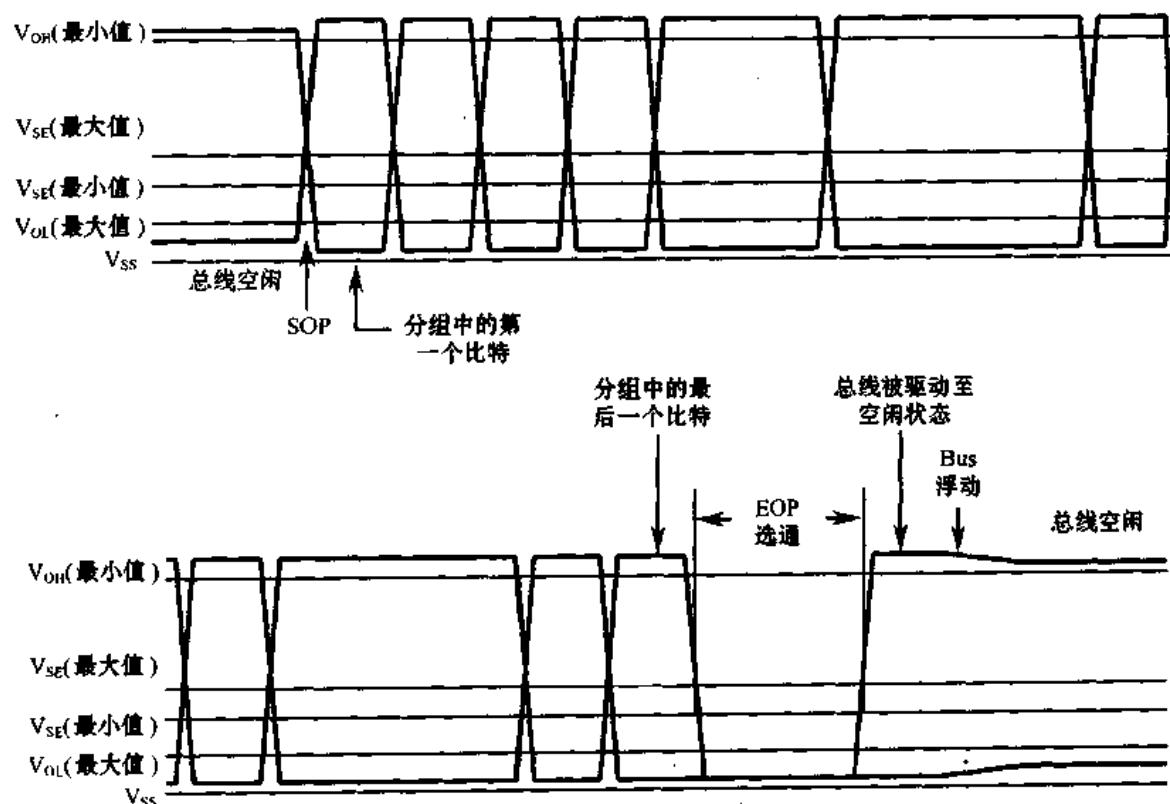


图 7-10 分组处理操作电平值

### 3. 复位信号

通过在一个设备的上行端口上实现一个延长了的 SEO 状态, 就可以在一个集线器端口上向下发送一个复位信号。当复位信号消除之后, 设备就处于连接状态, 而不是处于寻址或配置状态(参见 7.1 节)。注意复位信号仅适用于下行方向。

复位信号可以通过主机命令而在任一个集线器或主控制器端口上产生。复位信号必须至少维持 1ms。在复位操作结束时, 产生复位信号的端口会变为逻辑上的断开状态。如果设备接入了该端口, 总线上拉电阻将决定设备的类型(低速率或全速率设备), 而且该

端口会处于无效状态。

一个有效的设备(有电并且不处于挂起状态)在其上行端口上识别到了一个大于 $2.5\mu s$ 的单端0状态,就会将其视为一个复位信号,但必须在 $5.5\mu s$ 之内将该信号解释为复位信号。一个可以从持续时间位于32和64个全速率比特周期之间或4个和8个低速率比特周期之间SE0状态中识别出复位信号的设备能满足这些要求。复位信号可以通过位于信号发送端口下行方向的任一集线器的有效端口进行传播,但不会通过那些无效的端口传播。一个总线供电集线器如果在其根端口上接收到了一个复位信号,就会对它的所有下行端口断电。在复位信号撤消之后,所有接收到复位信号的设备都会被设置为具有它们的缺省地址,并处于一个未经配置的状态,接收到一个复位信号的集线器上的所有端口都会变为无效状态。

集线器必须能够建立连接,并且所有的设备都必须在复位信号消除之后的10ms之内,通过一个SET\_ADDRESS命令来接收设备地址(参见7.4节)。如果建立连接和接收一个地址的操作失败,就会使得该设备不能被USB枚举器(enumrator)所识别。对于一个集线器的情况,它可能会使与该集线器相连的所有设备都不能被识别。除了SETUP分组之外(参见8.4.5节中的第4小节),所有对数据或服务的其它要求都可能会得不到应答(NAK),最长可达5.0s,这之后该设备就会被宣布为有故障或不能识别。

复位操作可以将一个设备从挂起模式中唤醒。建议设备在接收复位信号之前,应该等待时钟稳定下来,以避免假“单端0”状态使设备复位的事件发生。一个设备从挂起状态苏醒过来需要最长为10ms的时间。

#### 4. 挂起

所有的设备都必须支持挂起模式。设备可以在任何上电状态下进入挂起状态。当一个设备在总线上识别出一个持续时间超3.0ms的稳定空闲状态时,它就会进入挂起状态。任一个总线操作都可以使设备从挂起状态中解放出来。应该保证每一帧中都产生一次SOF分组,以使在正常的总线操作时,全速率设备都保持清醒状态。未处于挂起状态的集线器,通过在连接了低速率设备的有效端口上产生一个低速率EOP,可以使低速率设备保持清醒状态。当一个设备处于挂起状态时,它从总线上吸收的电流小于 $500\mu A$ 。即使挂起的设备至多可以从总线上拉下 $500\mu A$ 的电流,当集线器处于挂起状态时,集线器端口也必须能为下行设备提供其最大电流。这对于支持如7.2.3节所述的远程唤醒操作,是十分必要的。

所有的设备都可以通过将总线状态切换至重新开始状态、正常的总线操作或发送一个复位信号,而从挂起状态中苏醒过来。某些设备具有能从与其内部功能模块的相关动作中苏醒,即而在共上行连接上产生用于唤醒或警告系统进行复位操作的信号的功能。这一特性被称作远程唤醒,在7.1.4节的第5小节中将给予介绍。

##### 1) 全部挂起

当总线上没有任何一处需要进行通信和网络被置于挂起状态时,需要使用全部挂起。主机通过停止它的所有处理操作来发出挂起开始信号(包括SOF令牌)。当总线上的每一个设备都认识到总线上没有活动和总线处于空闲状态已经有了相当长的时间时,它就会进入其自己的挂起状态。当集线器进入挂起状态时,它将停止在任一低速率配置的下行端口上发送低速率EOP。

## 2) 选择挂起

系统软件可能希望通过挂起拓扑结构中的某一段,而继续其余段上的正常工作来节省电源。通过对某一网段相连的集线器端口进行挂起,网络的各段就可以有选择地进入挂起状态。被挂起的端口会封闭总线上这一段的操作,而与之相连的设备在经过了上述的适当延迟之后也会进入挂起状态。

任何一种非集线器设备都可以以这种方式而被挂起。对于没有参与将余下的设备连接至主机的任一个集线器,可以通过将它所连的端口置为无效状态而挂起它。被选择挂起的设备仍然可以利用一个远程唤醒信号来向系统发出警告,虽然过程可能会有些不同。对挂起端口状态的描述可以在 11.2.3 节找到,而选择挂起则会在 11.5.2 节进一步阐述。

## 5. 重新开始(resume)

只要一个设备处于挂起状态,通过从总线上接收一个非空闲信号就可以重新开始它的操作,或者如果它具有远程唤醒功能,它就可以向系统发出继续操作信号。重新开始信号状态,由具有远程唤醒功能的主机或者一个设备在唤醒系统时使用。在重新开始信号的传播和产生过程中,集线器起着重要的作用。下列阐述的内容是有关一个一般的整体重新启动顺序的概要。有关重新开始的顺序、由选择挂起产生的特殊情况和集线器的作用的完整描述将在 11.5 节中给出。

在主机把总线置为挂起模式之后,它可以任意时刻发出重新开始的信号。主机所发送的重新开始信号必须至少持续 20ms,并以一个标准的低速率 EOP(持续时间为两个低速率比特周期的 SE0,后跟一个对空闲状态的处理操作)结束。20ms 长的重新开始信号保证网络中的所有能识别该信号的设备都会被唤醒。EOP 将拆除由重新启动信号所建立的连接,并使集线器为正常的操作做好准备。在重新启动了总线之后,主机必须在 3ms 之内开始传送总线数据(至少是 SOF 令牌),以防止系统重新进入挂起状态。

一个具有远程唤醒功能的设备,在发送远程唤醒重新开始信号之前,必须在总线处于空闲状态之后等待至少 5ms。这样可以允许集线器进入其挂起状态,并为传播重新开始信号做好准备。远程唤醒设备必须将重新开始信号保持并且不超过 15ms。在重新开始信号结束时,设备应将其数据线置为高阻状态。

具有远程唤醒设备的集线器的上行线将向其根端口和它的所有有效的下行端口,包括开始时发出重新启动信号的端口,传送重新启动信号。集线器必须在接收到原始重新开始信号后 50 $\mu$ s 之内,开始重新广播该重新启动信号。这一重新开始信号将以这种方式,向上行方向传播,直至其到达主机(或一个被挂起的集线器端口),而主机则会在它的有效端口沿下行方向返回该重新开始信号,其它任意一个集线器也会这样做。同时,向主机发送重新开始信号的集线器会在 10ms 之内苏醒过来。经过 1ms 的延迟,集线器会将它的根端点的连接方向由上行转变为下行,并将其根端口上的信号状态返回至其有效的下行端口上。这样一来主机就控制了重新开始信号,随后就如上述步骤而继续进行。

连接的端口和未连接的端口也能使受其影响的集线器发送一个重新开始信号并激活系统。

 **注意** 主机通过复位操作,可以激活整个总线。这样就要求必须对整条总线重新进行枚举和配置。

有关挂起和重新启动期间对功率控制的描述请参考 7.2.3 节。



### 7.1.5 数据编码/解码

在传输分组时, USB 应用了 NRZI 编码方式。在 NRZI 编码方式中, “1”由不出现电平变化来表示, 而“0”由电平发生变化来表示。图 7-11 给出了一个数据流和等同的 NRZI 码流。其中高电平代表数据线上的 1 状态, 而相应的图示则说明了 NRZI 编码过程。一串连“0”会使得 NRZI 数据每比特周期都会出现跳变。而一串连“1”则使得数据中长时间不会出现变化。

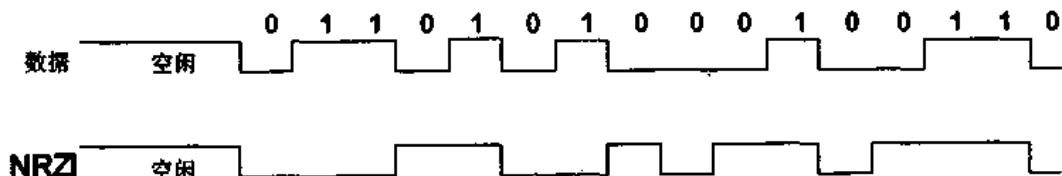


图 7-11 NRZI 数据编码

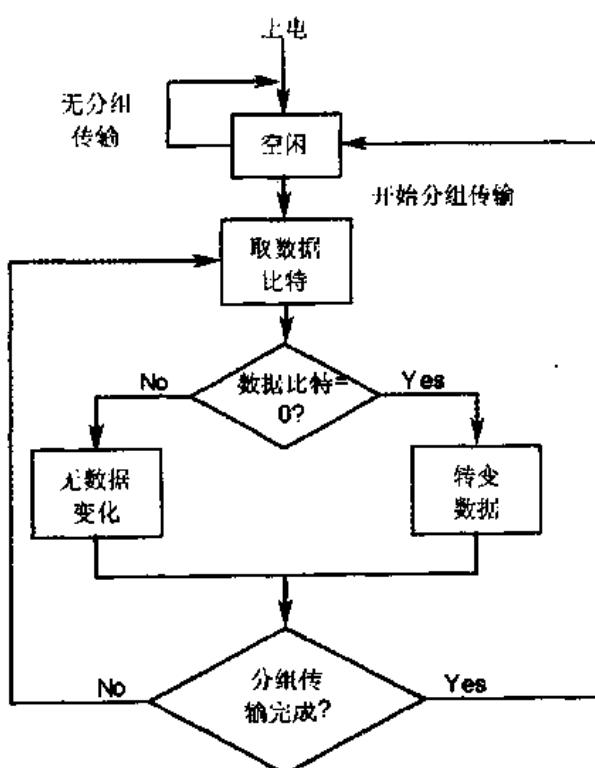


图 7-12 NRZI 编码的流程图

### 7.1.6 比特填充

为了保证信号有足够的变化, 在 USB 上发送一个分组时, 传输设备要进行比特填充(参见图 7-13 和图 7-14)。对数据进行 NRZI 编码之前, 在数据流内每 6 个连“1”之后都应该插入一个“0”。从而在 NRZI 数据流中强制加入一个变化。这样在逻辑上至少每

数据编码顺序：

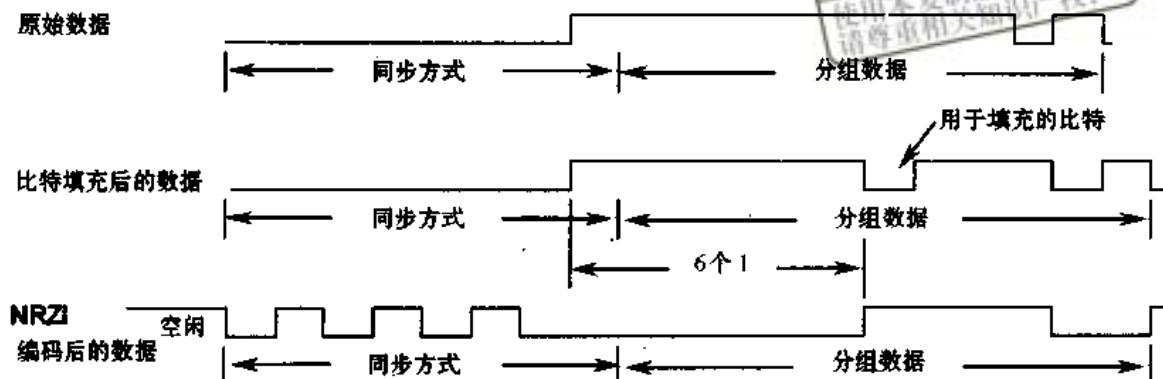


图 7-13 比特填充

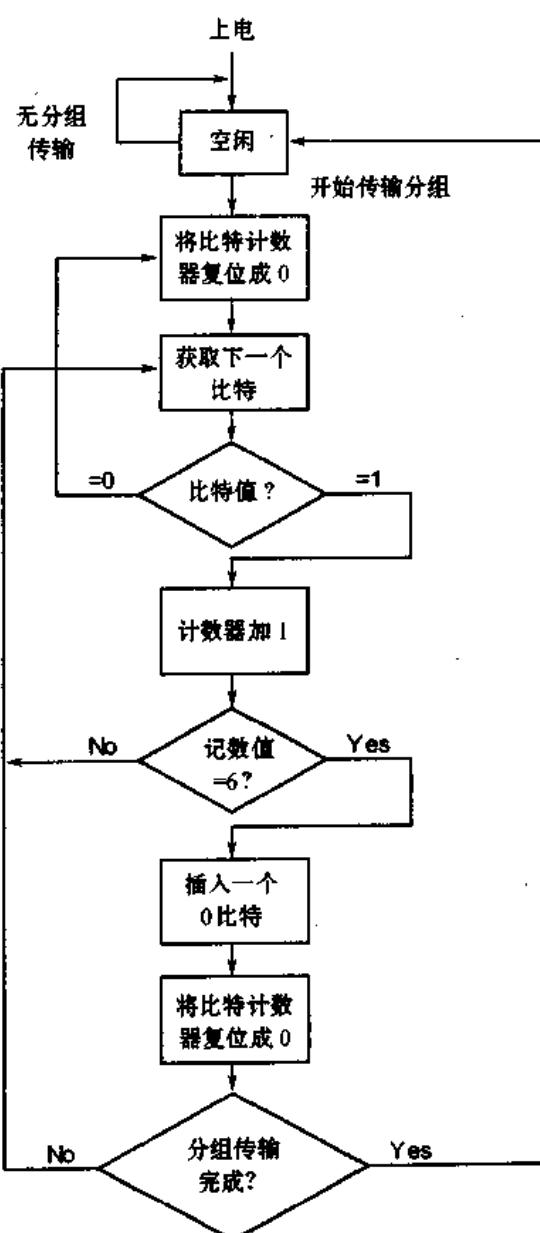


图 7-14 比特填充流程图

7个比特周期,接收器就会收到一个数据变化,以保证数据和时钟相互锁定。接收器必须对 NRZI 数据进行解码,识别填充比特并丢弃它们。比特填充以 sync(同步)模式开始进行,并且贯穿于整个传输过程之中。用于终止该 sync 方式的数据“1”,将作为一个序列中的第一个数据而加以计数。比特填充总是被强制执行,不会有例外。如果比特填充原则需要的话,即使该比特是分组结束(EOP)信号之前的最后一个比特,也会在后面插入一个“0”比特。

### 7.1.7 同步方式

图 7-15 所示的 NRZI 比特模式作为一个同步方式标记来使用,并加在每一个分组之前。这一方式同七个“0”再加一个“1”的数据方式(0x80)相同。



图 7-15 同步方式

### 7.1.8 起始的帧时间间隔和帧调整能力

USB 规定一个帧的时间间隔为 1.00ms。这一帧时间间隔是从一帧中的帧开始(SOF)PID 开始,到下一帧中的 SOF 令牌内的同一点结束而进行测量的。对起始的帧时间间隔所允许的误差为  $\pm 0.5\%$ (500ppm)。这一误差包括下列原因所产生的不准确性:最初的频率准确度、晶体容性负载、提供给振荡器的电压、温度和老化。

主控机器必须能对帧时间间隔进行调整。有两种元件可以对帧时间间隔进行调整。如果主机数据时钟速率不是严格地等于 12.00Mb/s,那么通过从 12000 开始对每一帧中的缺省比特数进行微小地调整,就可以满足对起始的帧时间间隔  $\pm 0.05\%$  的精确度要求(主控制器元件具有一定范围的可能的时钟值,可能必须使最初的帧计数值成为一个可对其进行编程的数值。7.1.9 节给出了这些值的范围)。为使主机同一个外部参考时钟同步,还需要额外的能进行  $\pm 15$  个全速率比特周期调整的能力。在正常的总线操作期间,每一次调整时对帧时间间隔进行重新编程后的改变量,都不能超过一个全速率比特周期。

集线器和某一全速率设备需要跟踪帧时间间隔。它们也同样需要充足的帧期调整能力来补偿其自身的频率容差,并可以跟踪主机的  $\pm 15$  个全速率比特周期的调整范围。

### 7.1.9 数据信号速率

全速数据速率的标称值是 12Mb/s。对于主机,集线器和全速率功能设备的数据速率容差为  $\pm 0.25\%$ (2500ppm)。为了符合帧时间间隔精确度的需要,主控制器数据速率的精确度必须保证其好于  $\pm 0.05\%$ (500ppm)。这一误差包括下列原因所产生的不准确性:最初的频率准确度、晶体容性负载、提供给振荡器的电压、温度和老化。

低速数据速率名义上为 1.5Mb/s。对于低速率功能模块而言,所允许的频率容差为  $\pm 1.5\%$ (15000ppm)。这一误差包括下列原因所产生的不准确性:最初的频率准确度、晶

体容性负载、提供给振荡器的电压、温度和老化。低速数据速率的抖动应该小于 10ns。这一容差要求允许在低速率设备中使用价格较低的振荡器。

超链接文字  
使用本复制品  
请尊重相关知识产权!

### 7.1.10 数据信号上升和下降时间

输出的上升时间和下降时间是在整个信号幅值的 10% 和 90% 处进行测量的(参见图 7-16)。在具 50pF 电容负载(CL)条件下测得的全速率数据信号上升和下降沿的边沿变化时间为 4ns(最小值)和 20ns(最大值)。上升和下降时间必须能很好地匹配。在负载电容为 50 pF 时,低速率信号的上升和下降时间为 75ns(最小值);当负载电容变为 350pF 时,上升和下降时间为 300ns(最大值)。在这两种情况下,驱动电缆时都应该使上升和下降沿平滑变化(单调)以避免产生过高的 EMI。(对全速率信号,驱动器和电缆阻抗不匹配,所以对于从接收器电缆端反射回来的信号应该予以考虑)。

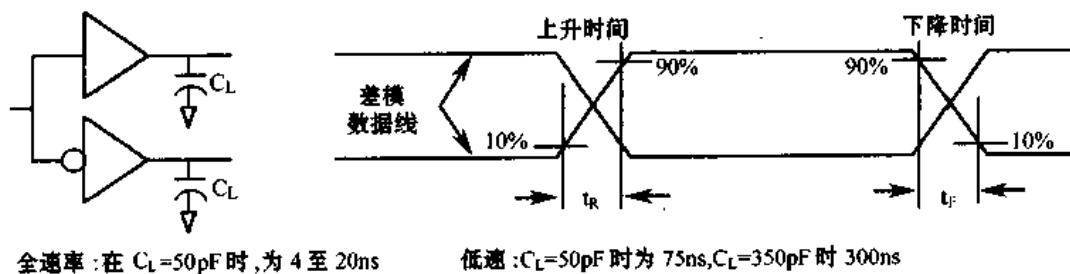


图 7-16 数据信号的上升和下降时间

### 7.1.11 数据源信号

本节讨论了由一个设备(数据源)产生并发送出来的数据的时序特征。7.1.12 节则讨论了通过一个集线器和中继部分,而进行传输的数据所具有的时序特征。在这一节中,定义 TPERIOD 作为数据的真实周期,它具有 7.1.9 节所定义的范围。

#### 1. 数据源抖动

数据源所传送的数据的边缘时序可能会有一些变化(抖动)。任意的数据变化点之间间隔的时间都表示为  $N \times T_{\text{PERIOD}} \pm$ (抖动时间),其中 N 是变化点之间所有的比特数。对数据抖动的测量时所用的负载电容,与测量最大上升和下降时间时使用的负载电容相同,而且也是在数据线上的交叉点进行测量的,如图 7-17。

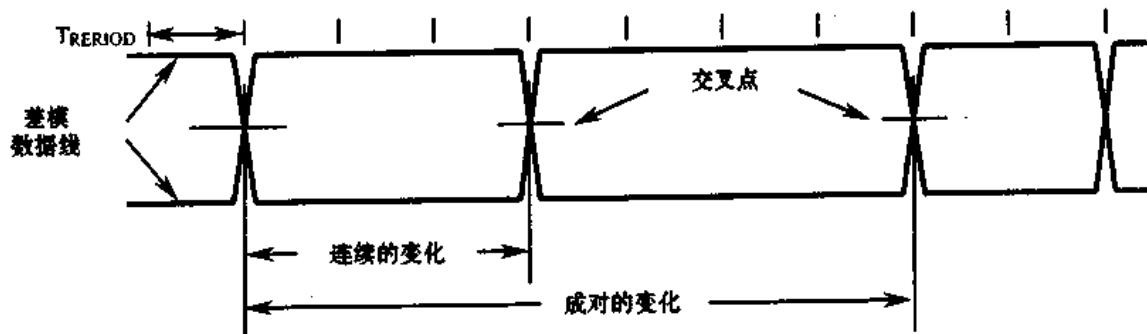


图 7-17 数据抖动

对于全速率传输,任意连续的差模数据变化的抖动时间必须在  $\pm 2.0\text{ns}$  之内,而对于任意一组成对的差模数据变化则应该在  $\pm 1.0\text{ns}$  之内。对于低速率传输,如果是任意连续的数据变化,抖动时间应该在  $\pm 25\text{ns}$  之内;而如果是任一组成对的差模数据变化,抖动时间应该在  $\pm 10\text{ns}$  之内。这一抖动数值包括由于差模缓冲器时延和上升/下降时间不匹配,内部时钟源抖动和噪声以及其它随机影响所带来的定时变化。

## 2. EOP 带宽

EOP 中的 SEO 持续宽度约为  $3 \times T_{\text{PERIOD}}$ 。测量 EOP 宽度时,所使用的电容负载与测量最大上升和下降时间时所使用的电容负载相同,并且是同在数据线上的差模信号交叉点相同的电平上进行测量的(参见图 7-18)。

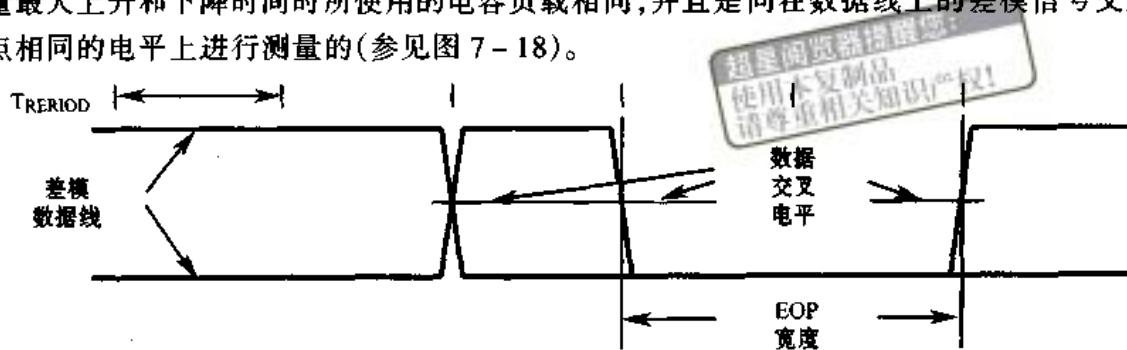


图 7-18 EOP 定时宽度

对于全速率传输,从发送器发出的 EOP 宽度必须在  $160\text{ns}$  和  $175\text{ns}$  之间。而对于低速率传输,发送器传送的 EOP 宽度必须在  $1.25\mu\text{s}$  和  $1.50\mu\text{s}$  之间。这些范围包括由于差模缓冲器时延和上升/下降时间不匹配,内部时钟源抖动和噪声以及其它随机影响所带来的定时变化。

全速率接收器必须将后跟一个 J 变化的一个  $82\text{ns}$  宽的 SEO 作为一个有效的 EOP 来予以接收。一个小于  $40\text{ns}$  的 SEO 或任何一个没有跟有一个 J 变化的 SEO 必须作为一个 EOP 而被全速率接收器丢弃。根据与抽样和同步有关的应用的约定,接收器可能丢弃或接收一个从  $40\text{ns}$  到  $82\text{ns}$  宽的 EOP。低速率接收器必须将后跟一个 J 变化的一个  $330\text{ns}$  宽的 SEO 作为一个有效的 EOP 来接收。一个小于  $330\text{ns}$  的 SEO 或任何一个没有跟有一个 J 变化的 SEO 必须作为一个 EOP 而被丢弃。和前面提到的一样,接收器可能丢弃或接收一个从  $330\text{ns}$  到  $670\text{ns}$  宽的 EOP。对于这一原则来说,主机是一个例外,它总是在接收时利用全速率 EOP 时序原则和任一数据速率来进行工作。(这样一来就有必要支持帧结束差错恢复了)。任何持续时间为  $2.5\mu\text{s}$  或更长的 SEO 都会自动作为一个复位或断开信号来处理(与方向有关)。

### 7.1.12 集线器信号时序

一个全速率差模数据信号经过集线器的传播如图 7-19 所示。下行信号是在端口没有接电缆,并且负载电容为  $50\text{pF}$  的条件下进行测量的。经过电缆和集线器后的全部时延的最大值应为  $70\text{ns}$ 。如果集线器具有一个符合 USB 标准的可拆除电缆,那么经过集线器的最大时延应为  $40\text{ns}$ ,以允许最坏情况下电缆时延可以为  $30\text{ns}$ 。通过集线器的时延在上行的下行方向都进行了测量,如图 7-19(b)所示从输入端口上的数据线交叉点一直到

输出端口上的交叉点。

对于差模信号的低速率传播时延,我们使用了与全速率信号的情况相同的测量方法。低速率集线器时延最大为300ns。这允许低速率缓冲区传播时延和上升以及下降时间可以具有更大的值。

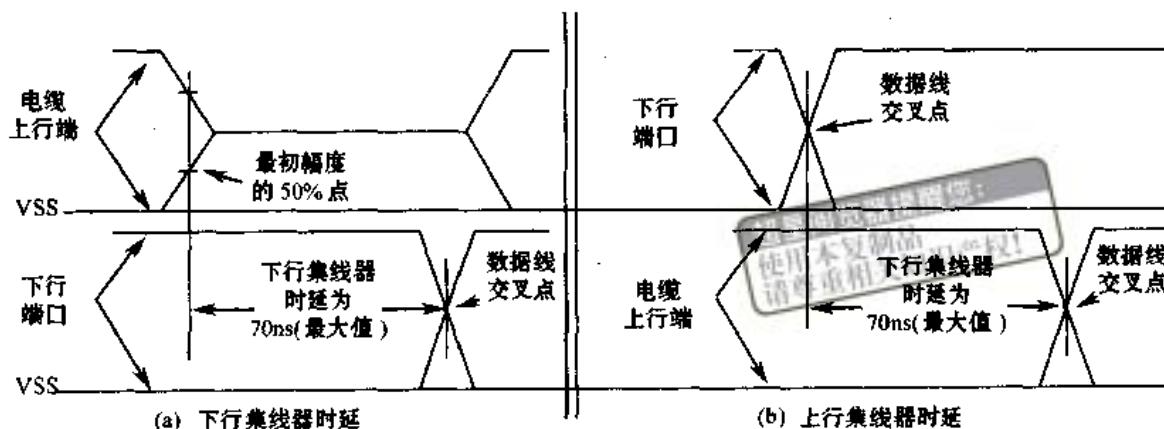


图7-19 全速率差模信号的集线器传播时延

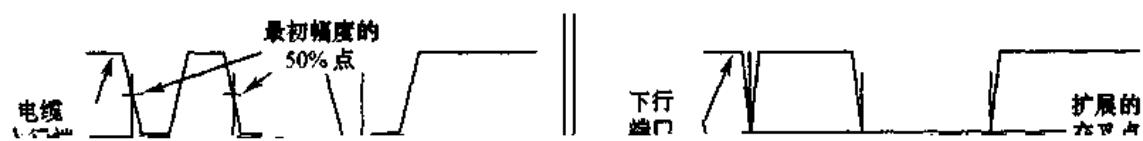
如果集线器是作为一个中继器而使用时,它必须能够在其输出端真实地再生出所接收到的信号。这就意味着对于差模信号而言,一个J状态到K状态变化的传播时延,必须同一个K状态到J状态变化的传播时延紧密匹配。而对于一个集线器接入电缆而言,在这两个时延之间所允许存在的最大差别为 $\pm 3.0\text{ns}$ (与图7-19中所示的测量方法相同)。类似地,在通过一个集线器后,任意两个J至K或K至J的状态变化之间所存在的时延差应该小于 $\pm 1.0\text{ns}$ 。

一个例外的情况是在SOP由空闲状态变化至K状态时可能会出现的偏差(参见7.1.4节中的第2小节)。在这种情况下,向反方向端口的传送时延中包含了激活输出缓冲器所需的时间。但是,这一时延必须能够与正常情况下的集线器时延紧密匹配,并且在经过一个正常的J至K状态变化之后,所产生的最大附加时延差应为 $-3.0\text{ns}$ 至 $+5.0\text{ns}$ 之间。这样一来就对分组中第一个比特可能会出现的最大畸变进行了限制(注意:由于SOP变化的畸变与下一个K至J的状态变化有关系,所以不应该使用第一个Sync域来对接收器和数据流进行同步)。

EOP必须以和差模信号相同的方式通过一个集线器进行传播。识别一个SEO所需的传播时延必须不小于J至K或K至J差模数据时延中较大的一个(以避免漏掉分组中最后的数据),但是它也不能比全速率情况下较大的差模时延大出15ns以上,对于低速率的情况这一要求应该为200ns(以避免在分组结束时产生一个比特填充错误)。图7-20所示的是EOP时延。

由对SEO状态的识别电平不是位于信号变化的中点,所以SEO状态的宽度在它经过每一个集线器之后都会发生改变。一个集线器对一个全速率SEO状态的持续长度的改变不能超过 $\pm 15\text{ns}$ (根据EOP-的EOP+的时延差所得的测量值,参见图7-20)。一个来自于一个低速率设备的SEO具有较长的上升和下降时间,并且其畸变会更大,但是这种仅存在于从低速率设备到它所连接的端口的这一段电缆上。此后,信号发送将使用全速率缓冲区和它具有的更快的上升和下降时间。当一个来自于低速设备的SEO,当它通

过一个与该设备相连的集线器时,其变化不能超过 $\pm 300\text{ns}$ 。这一时间可以使得在低连端口上的某些信号调整,降低其对噪声的灵敏度。



超星浏览器提醒您：  
使用本复制品  
请尊重相关知识产权！

表 7-3 低速率抖动预算表

集线器数	5	全速率频率的变化(总计)	0.0025
比特/跃变的最大值	7	低速率频率的变化(总计)	0.015
低速率 - 上行			
抖动源		下一个跃变	
		每个(ns)	总计(ns)
功能设备驱动器抖动		25.0	25.0
功能设备频率(总计)——最坏情况下	10.0/bit	70.0	10.0/bit
信源(功能设备)抖动合计		95.0	150.0
集线器/低速率设备抖动	45.0	45.0	45.0
其余的(全速率)集线器抖动	3.0	12.0	1.0
抖动规定		152.0	200.0
主机频率(总计)	1.7/bit	12.0	1.7/bit
主机接收器抖动预算		164.0	225.0
主机驱动器抖动	2.0	2.0	1.0
主机频率(总计)——最坏情况下	1.7/bit	12.0	1.7/bit
信源(主机)抖动合计		14.0	25.0
集线器/低速率设备抖动	45.0	45.0	15.0
其余的(全速率)集线器抖动	3.0	12.0	1.0
抖动规定		75.0	45.0
功能设备频率(总计)	10.0/bit	70.0	10.0/bit
功能设备接收器抖动预算		150.0	185.0

注:本表对通过了最多的集线器,使用全速率信号而以低速数据速率向一个低速率设备进行传输的主机,给予了说明。当主机直接同低速设备相连时,它会使用低数据速率和低速信号约定,并且必须满足在“抖动说明”一行中所列出的对信源抖动的要求。

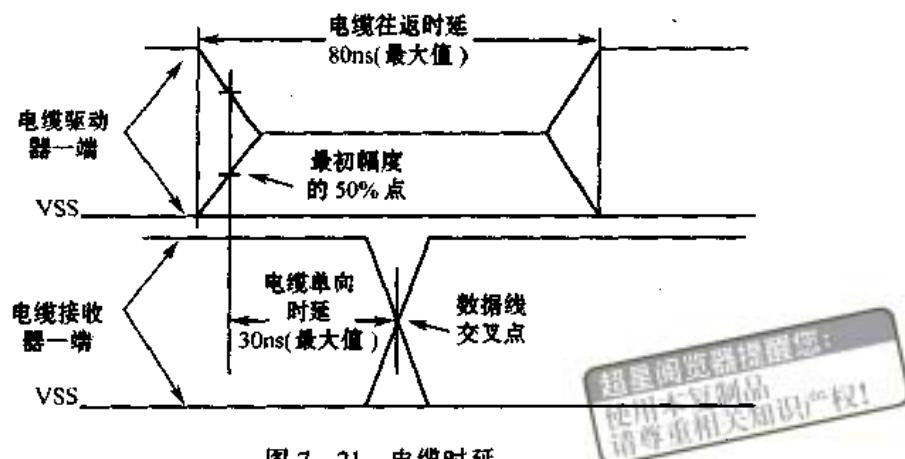
由于低速率设备所连接的集线器同其它所有设备在数据通路上所引入的抖动有所不同,所以低速率设备抖动预算表中有一行是附加的。其余的设备都以全速率信号约定来工作(即使是在低速数据速率下)。

### 7.1.14 电缆时延

在一个周期内,USB上只允许有一个数据变化。一个全速率信号边沿必须在一个全速率比特周期之内完成变化,向电缆远端传播,返回和稳定下来。因此,所允许的最大电缆时延为30ns。与电缆上的实际传输速率无关,对于全速率设备,其电缆最大长度为5.0m;而对于低速率设备,电缆最长为3.0m。一个电缆上对单向数据时延的测量如图7-21所示。

### 7.1.15 总线转向时间/分组间时延

直到前一个设备完成了EOP序列并禁止了其驱动器后,一个新的设备才可以驱动总线。只有当一个新设备检测到在EOP内的SE0之后,总线处于J状态至少两个比特周期,它才能驱动总线,从而保证了上述条件的执行。最小值为两个比特周期的要求适用于所有的设备,包括背对背的主机分组传输。



如果一个功能模块希望对一个主机传输做出响应,那么在电缆上行端出现 EOP 之后,总线返回 J 状态的 7.5 个比特周期之内,它必须使响应出现在电缆上行终点处。对于一个没有完整电缆的功能模块或集线器来说,最大总线转向时间为 6.5 个比特周期。在一个层次最多的拓扑结构中(参见 7.1.16 节),这个最大总线转向时间将阻止一个全速率接收主体设备,在其作出响应 16 个比特周期之后就出现超时。不过,这些时序要求适用于所有全速率和低速率设备。

一个主机必须对一个由功能模块发出的一个数据分组做出响应的最大时延为 7.5 个比特周期(如果需要进行握手的话)。这是在主机端口引脚上所得到的测量结果。对于分离的处理操作之间,没有最大时延要求。

### 7.1.16 端到端最大信号时延

在一个传输结束后(在 EOP 内,SE0 变化至 J 状态之后),如果一个等待传输响应的设备,在超时期限之内没有检测到分组开始(SOP)变化,它就会把该处理操作标记为无效。这种情况可以在一个 IN 令牌和后续的数据分组之间发生,或者在一个数据分组和握手分组之间发生(参见第 6 章)。我对设备的数据引脚进行测量后发现,在 EOP 结束之后,等待响应的设备不会在 16 个比特周期之前和 18 个比特周期之后出现超时。主机在检测到一个未被响应的分组结束之后,它不会在 18 个比特周期之前(在其数据引脚进行测量)开始传输下一个处理操作令牌。

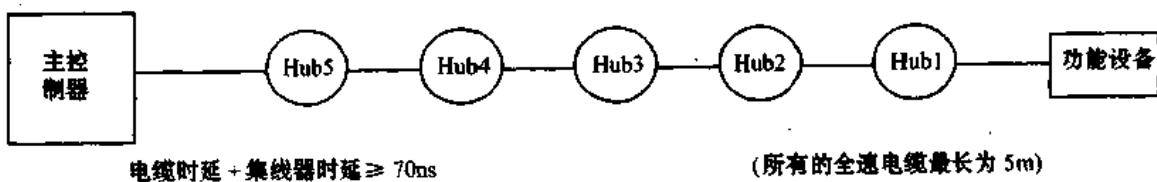


图 7-22 最坏情况下的端到端信号时延模型

## 7.2 功率分配

本节描述了 USB 功率分配规范。

### 7.2.1 设备类型

通过引入单元负载这一概念,可以简化对不同设备类型的电源传输和使用要求。一个单元负载规定为 100mA。

USB 所支持的设备,可以根据其电源使用情况进行分类,它们包括:

(1) 总线供电集线器:从 USB 连接器的电源引脚,向任一内部功能模块和下行端口提供电源。当设备上电时可以获得一个单元负载,总数为 5 个单元负载,并由内嵌的功能模块和内部端口来分享。不管集线器上其它端口所获得的电流有多少,一个总线供电集线器的外部端口仅能为每个端口提供一个单元负载。当集成器处于活动或挂起状态时,它必须能够提供这一端口电流。

(2) 自供电集线器:内部功能模块和下行端口的电源供应不是来自于 USB。虽然 USB 接口可能会从其上行连接中获得一个单元负载的电流,以允许该接口进行工作,而该集线器的其余部分没有电源供应。即使是在挂起状态,集线器也必须能够在其所有的外部下行端口上最多提供 5 个单元负载。

(3) 低功率,总线供电功能设备:这些设备的所有电源供应都来自于 USB 连接器。在任何时刻,它所获得的供应不能超过 1 个单元负载。

(4) 大功率,总线供电功能模块:这些设备的所有电源供应都来自于 USB 连接器。它在上电时所获得的电源供应必须不超过 1 个单元负载,并且在配置之后可以最多获得 5 个单元负载。

(5) 自供电功能设备:当集线器的剩余部分没有电源供应时,为了使接口得以工作,它可以将其上行连接最多获得 1 个单元负载的电流。其它所有电源供应都来自于一个外部(对于 USB 而言)电源。

在一个台式计算机内,主机上的集线器是一个自供电集线器。而在一个笔记本电脑内。同样的集线器既可以规定为一个自供电集线器,也可以规定为总线供电集线器。

无论它是总线供电或是自供电,所有的只能从总线上获得(接收)电流。它们不能向主机或集线器端口提供上行电流。上电时,所有的设备都要保证它们的上行端口处于禁止状态,从而使设备可以接收复位信号,并且使一个设备所获得的最大电流为 1 个单元负载。如果一个设备由总线获得电源供应,那么由  $V_{bus}$  所得到的内部电源供应必须在  $V_{bus}$  达到 4.4V 的 100ms 之内,达到稳定。所有从总线获得电源供应的设备必须能进入挂起状态,并将它们从  $V_{bus}$  上所消耗的电流减小至 500uA 以下(参见 7.1.4 节中第 4 小节和 9.2.5 节中第 1 节)。

#### 1. 总线供电集线器

利用一个如图 7-23 所示的功率控制电路,就可以满足上述对总线供电集线器的要求。总线供电集线器通常都至少含有一个内嵌的功能模块。主控制器一直处于有电状态,从而允许主机在枚举过程中访问电源管理和其他配置寄存器。一个内嵌的功能模块可能需要改变其电源供应,从而使整个设备上电时(集线器和内嵌的功能模块)所获得的电流不超过 1 个单元负载。关闭任一内嵌的功能模块的电源供应既可以通过断掉电源,也可以通过关闭时钟来实施。如果内嵌的功能模块和集线器控制器所获得的全部电源供应小于 1 个单元负载,就不需要对内嵌的功能模块予以关闭。一个总线供电设备所获得

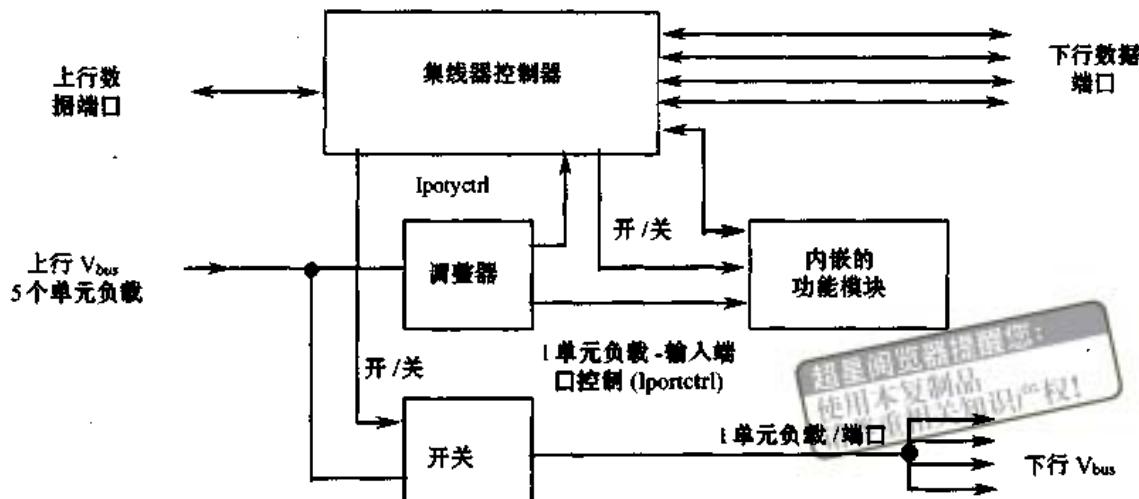


图 7-23 复合总线供电集线器

的全部电流是流向集线器控制器、任一内嵌的功能模块和下行端口的电流总和。

图 7-23 根据最大上行电流为 5 个单元负载, 给出了电源分配情况: 一个单元负载用于集线器控制器和内嵌的功能模块, 每个下行端口使用一个单元负载。因此所支持的下行端口最大数量限制为 5 个。如果需要更多的端口, 那么集线器就要自己供电。如果内嵌的功能模块和集线器控制器所获得的电流超过了一个单元负载, 那么端口数据就要相应地减少。对总线供电集线器的功率控制可能需要一个调节器。如果说有的话, 该调节器总是可以为集线器控制器提供电源, 调节器也可以为内嵌的功能模块供电。对流入的电流限制必须融入到调节器子系统当中。

对下行的端口的电源供应必须能进行改变。集线器控制器可以提供一个来自于主机的, 由软件进行控制的开/关信号。当一个设备上电后或在发出重启信号后, 对下行端口的电源供应会处于“关闭”状态, 当变换为“开通”状态时, 这个开关将实现一个软开通功能, 阻止从上行端口获得短暂的过量电流。通过上行电缆、连接器和一个总线供电集线器开关之后, 下降的电压在最大电流情况下不应超过 350mV。

## 2. 自供电集线器

自供电集线器具有一个本地电源, 它可以向任一内嵌的功能模块和所有的可行端口供电, 如图 7-24 所示。但是, 集线器控制器的电源供应不是来自于上行端口, 就是来自于本地电源。由上行电源供应来为集线器控制器供电的好处是: 即使设备电源供应中断主机仍然可以进行通信, 这样就可以对一个未连接的设备和一个上电设备加以区别。

本地电源对所有的下行端口进行供电。可以支持的端口数仅受到本地电源供应能力和安全考虑的限制。一个比较合理的最大数量是支持 7 个端口。每一个端口都必须能够提供至少 5 个单元负载。但是, 为了满足通常的安全限制, 单个端口所传送的电流不能超过 5.0A。为了向所有的端口提供足够的电流供应, 并且不超过通过一个端口的可用电流值, 对下行端口的限制电流可能需要分割成两个或更多的端口子组。如果所有 7 个端口都是并行排列, 那么某个端口处的可用电流应为  $7 \times 500\text{mA} = 3.5\text{A}$ 。这一数值已经非常接近安全极限了。通过实现两个电流限制电路, 在操作中所需提供的最大电流就可以减小到 1.5A 至 2.0A 之间, 从而为安全限制提供了足够的空余量。

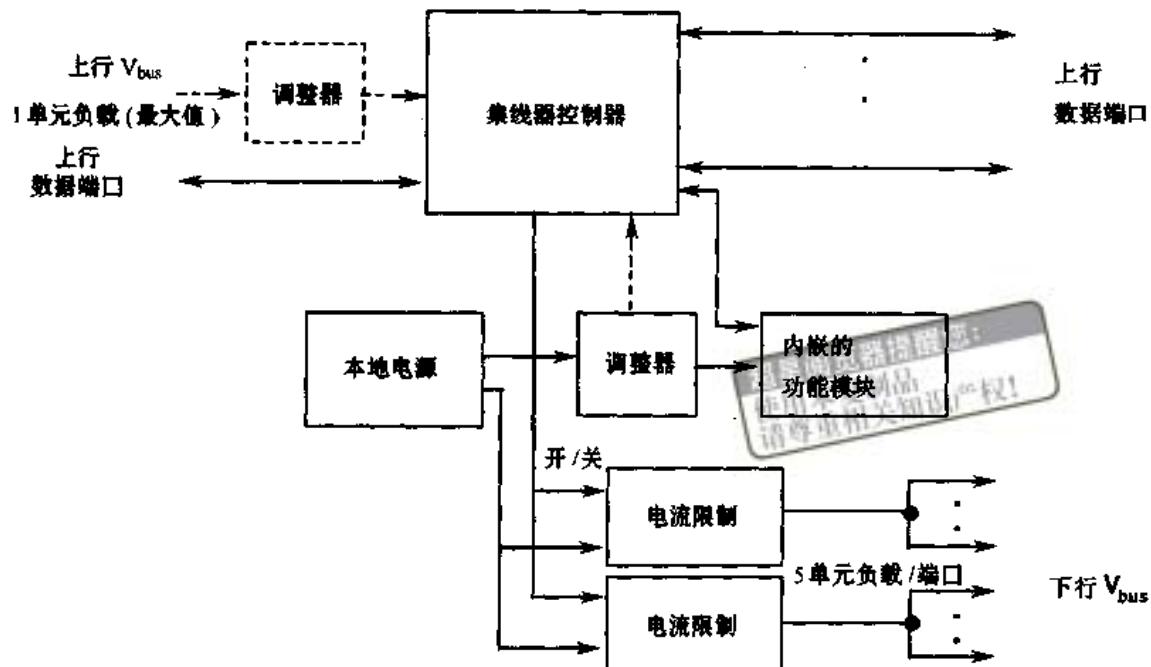


图 7-24 复合自供电集线器

### 1) 过电流保护

由于安全原因，主机和所有的自供电集线器必须实施过电流保护，并且它们必须有办法来检测过电流情况并向 USB 软件报告。要是一组下行端口所获得的全部电流超过了预先设定的值，那么过电流保护装置就会对所有的下行端口断电并且通过集线器向主控制器报告这一情况。预先设定的电流值不能超过 5.0A，并且同所允许的最大端口电流相比应该足够大，从而使上电或动态接入时出现的瞬时电流不会触发过电流保护装置。如果在任一端口上出现了过电流情况，USB 的后续操作就不会得到保护，并且一旦消除了这一情况，还有必要像在上电时所进行的操作那样，对总线重新进行初始化。实现过电流限制的方法包括多个熔丝、标准保险丝或使用某一类型的固态开关。唯一的要求是每一个端口的电流要限制为 5 个单元负载，并且要向主机通报过电流情况。

由于在大功率功能模块和总线供电集线器中采用了电源开关这一保护措施，即使配置了非法的拓扑结构，也不应该发生电流限制。相反，过电流保护电路应该对严重的设备操作失败、当超过了电流预算值时打开设备的软件错误和诸如将连接器引脚短路等用户操作加以保护。

### 2) 电源隔离

图 7-24 是基于这样一个假设而得到的，那就是本地电源和上行及下行端口具有一个公共地线。但是，其  $V_{bus}$  却和上行端口的  $V_{bus}$  相互绝缘。此外，还有一个额外的要求，就是要在自供电集线器的机箱地线和 USB 信号地之间实现 DC 隔离。机箱地线同具有一个 120V<sub>ac</sub> 电源电缆的地相连，并且不保证来自于两个不同输出的 AC 地具有同样的电位。如果没有对这一点采取预防措施，就会导致大量低频电流通过 USB 地线通路。

### 3. 低功率，总线供电集线器

一个低功率设备是指当其全负荷工作时，它从 USB 电缆上获得的电流小于 1 个单元

负载。调节器模块必须既能对输入电流加以限制,又能为正确的信号电平提供必要的电压。图 7-25 给出了一个典型的总线供电低功率功能设备的示意图,例如一个鼠标。低功率调节功能可以集成到功能芯片中。对于更高的电流,即位于 20~100mA 范围之内的电流,可能需要使用一个 IC 线性调节器。低功率功能模块必须能在  $V_{bus}$  输入电压为 4.40V 这样低的条件下进行工作(在电缆插头端所得到的测量结果)。

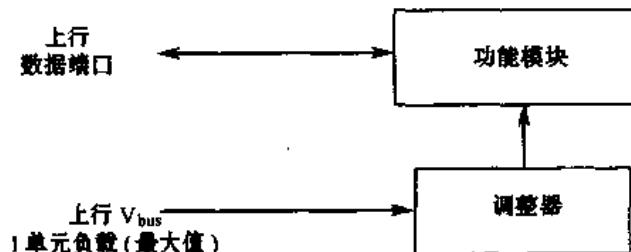


图 7-25 低功率功能设备

#### 4. 大功率,总线供电功能设备

如果一个设备在其全部上电后,它从 USB 电缆上获得的电流大于 1 个单元负载,最大可以为 5 个单元负载,那么该设备就称之为大功率设备。一个大功率功能设备需要分阶段地改变电源供应。它必须首先从电流小于 1 个单元负载的低功率状态下启动。在总线进行枚举操作时,它的全部功率要求必须在同可用的功率预算比较后,再作出决定。如果功率预算中有足够的功率可供分配,那么该功能设备的剩余部分就会上电。如果没有足够的功率可供利用,那么该功能模块的剩余部分就不能得到电源供应,并且还要向客户发出一个功率受限的警告消息。图 7-26 给出了一个大功率功能设备的示意图。其中功能电路被分成两个部分:功能控制器包含了允许枚举和功算预算操作所必需的最少电路;功能电路的剩余部分都包括在功能模块内。大功率功能设备必须能够在输入电压为 4.40V 这样低的条件下,工作于低功率(电流为 1 个单元负载)模式,从而使得即使将其插入了一个总线供电集线器后,它仍然可以工作。另外,它也必须能够在  $V_{bus}$  输入电压为 4.75V 条件下,以全负荷方式(最大电流为 5 个单元负载)进行工作(在电缆上行插头端测量所得的结果)。

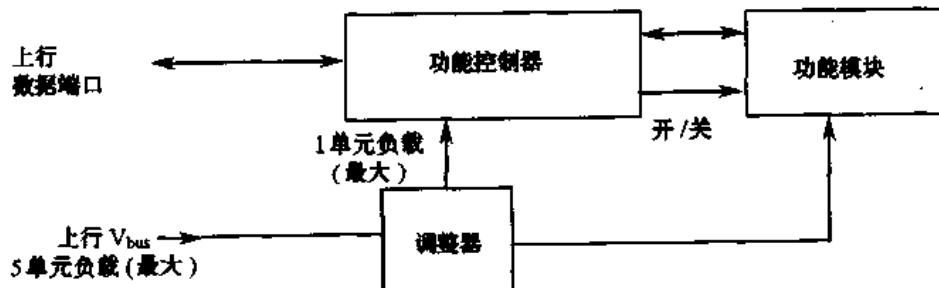


图 7-26 大功率,总线供电功能设备

#### 5. 自供电功能设备

图 7-27 给出了一个自供电功能设备的示意图。功能控制器的电源供应要么是经过一个低功率调节器而由上行总线获得,要么就是来自于本地电源。使用前一种方案的好处是:它允许对一个本地电源关闭了的自供电设备进行检测和枚举操作。当一个功率控

制器是由外界来对其供电时,它所获得的最大上行电流为1个单元负载,另外调节模块还必须对输入电流加以限制。而功能模块所获得的电源供应量仅受到本地电源供应能力的限制。因为本地电源无需为每一个下行总线端口供电,所以它不需要实现电流限制、软开关或电源变换等功能。

自供电功能设备必须遵守同自供电集线器相同的地线和 $V_{bus}$ 相互隔离原则。

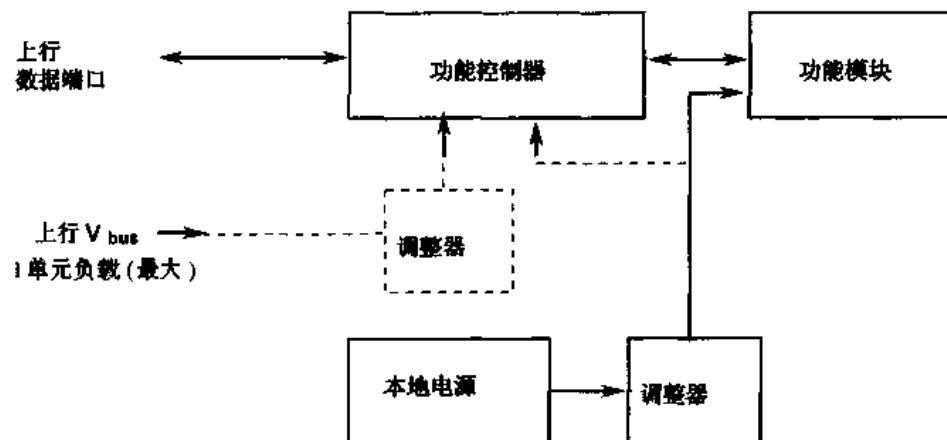


图 7-27 自供电功能设备

### 7.2.2 电压下降预算

针对电压下降所做出的预算由以下几方面决定:

- (1) 主机或上电后的集线器端口所提供的电压位于4.75V至5.25V之间。
- (2) 总线供电集线器,从其与电源相连接的电缆插头到其提供电源供应的输出端口连接器之间电压最大跌幅为350mV。
- (3) 在上行电缆的连接器端,所有的集线器和功能模块必须能够以4.40V这样低的电压来提供配置信息。只有低功率功能模块需要能够以最小电压来进行全负荷工作。
- (4) 如果该功能模块获得的电流超过了1个单元负载,那么它必须能够在其上行电缆的连接器端,最小输入电压为4.75V的条件下进行工作。

图 7-28 给出了由一个总线供电集线器来驱动一个总线供电功能设备时,在所构成的最坏情况下的拓扑结构内,所允许的最小电压。

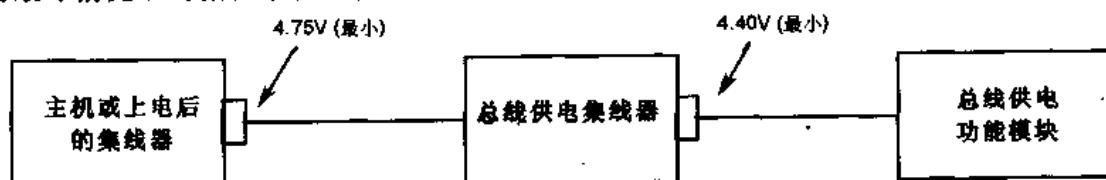


图 7-28 电压下降示意图(最坏情况下)

这些要求对电缆和连接器 IR 所引入的压降(参见 6.4 节)和端口交换所引起的压降都提出了严格的限制。

### 7.2.3 功率控制

当一个设备进入挂起状态时,它从总线上获得的电流必须小于  $500\mu A$ 。这一电流包

括了通过总线上拉和下拉电阻所获得的电流值。对于总线供电集线器,这一电流没有包括同集线器下行端口相连的已通电设备所获得的电流。

当一个集线器处于挂起模式时,它必须能够为每一端口提供最大的电流(对总线供电集线器而言,每个端口为一个单元负载;对自供电集线器而言,每个端口为五个单元负载)。这对于支持那些在系统的其余部分仍处于挂起状态时可以上电工作,能够实现远程唤醒功能的设备是很有必要的。

当设备要么由其自身,要么通过识别重新开始信号而被激活后,它必须对  $V_{bus}$  上的输入电流进行限制。在集线器中的  $V_{bus}$  上所达到的最大电压下降值应为 330mV,或是来自于功能模块的标称信号幅值的 10%。设备必须具有充足的旁路电容或一个受控的通电顺序,从而使设备被激活时的任一时刻,集线器所获得的电流不会超过该端口的最大电流承载能力。

#### 7.2.4 动态插拔

插入或下一个集线器或功能设备的操作,不应该对位于该网络上其它网段的设备的性能产生影响。拔下一个功能设备会停止在功能设备和主机之间进行的处理操作。但是,连接该功能设备的集线器会从这一状态中恢复出来,并且警告主机该端口已经拆除了连接。

##### 1. 涌入电流限制

当一个功能模块或集线器插入网络时,在  $V_{bus}$  和地之间就会存在一定的电容。而且,只要调节器通电后,它就会向其输出旁路电容和功能模块提供电流。所以,如果不采取措施来阻止它,就会有大量的电流涌人设备,足够将集线器的  $V_{bus}$  拉低至最小工作电平以下。当一个大功率功能设备切换到大功率工作模式时,也会出现涌人电流。这一问题必须通过限制涌人电流,在每一个集线器内提供充足的电容来阻止提供给其它的端口的电流超过容限等手段来解决。对涌人电流加以限制的另外一个主张是将接触电弧放电降至最低,因而可以延长连接器的使用寿命。

集线器上的  $V_{bus}$  所出现的最大压降应为 330mV 或约为来自于功能设备的标称信号幅值的 10%。为了达到这一目标,必须满足下列条件:

(1) 一个电缆的下行端所带的最大负载应为  $10\mu F$  再并联一个  $44\Omega$  电阻。 $10\mu F$  电容代表任一在功能模块内直接跨接在  $V_{bus}$  线上的旁路电容器,再加上经过设备中的调节器后任意的可觉察到的电容影响。 $44\Omega$  电阻代表在连接阶段,由设备所产生的一个单位电流负载。

(2) 如果由于负载电流具有较大的幅值而需要在设备中实现更大的旁路电容,那么设备就必须因电缆中的电流而融入某种形式的输入电流限制功能,从而使其适应上述负载所具有的特性。

(3) 集线器端口  $V_{bus}$  电源线必须用不小于  $120\mu F$  的钽电容器来进行旁路。应该使用良好的标准旁路方法来把旁路电容和连接器之间存在的电感降至最低限度。旁路电容器自身应该有较低的损耗因子,以允许在较高的频率工作时可以由它来完成去耦功能。

一个集线器的上行端口也需要满足上述要求。另外,一个总线供电集线器还必须在其对下行端口提供电源时,以一种软启动电路的方式来提供额外的电涌限制。

在动态插入设备期间,通过在连接器内使信号线休眠而使电源线首先进行连接,可以

保护信号引脚免受过电流影响。这样就可以保证在信号线连接之前,向下行设备所输送的电源已经确定了。而且,在连接操作期间,集线器端口信号线被禁止并处于高阻状态,不允许为达到标准信号电平而提供电流。

## 2. 动态拆除

当一个设备从网络上拆下,而电缆中仍有电源供应时,电缆所具有的电感就会在设备电缆的开路端产生一个大的回程电压。这一回程电压并没有什么坏处。在集线器端口上采用适当的旁路措施就可以抑制任何耦合噪声。相反,这种噪声的频率还与电缆的长度有关。对于一条1m长的电缆而言,最高频率为60MHz。这就需要较低电容,而在每一个集线器端连接器上则需要具有极低感应系数的旁路电容器。回程电压和由其产生的噪声影响也可以通过设备端的电缆上的旁路电容而得到缓和。另外,在设备端的电缆上必须有一个极小的电容,以确保在电缆开路端所产生的感应回程电压,不会使设备端的电压出现极性反转。建议采用一个最小值为 $1.0\mu F$ 的电容跨接在 $V_{bus}$ 上对其进行旁路。

另外,在动态拆除操作期间,通过使信号线在连接器内处于休眠状态,而使其早于电源引脚断开连接,可以保护信号引脚免受过电压影响。

## 7.3 物理层

在下面几节中将介绍物理层规范。

### 7.3.1 环境

USB工作的环境温度应该为0℃至70℃。

USB必须满足下列规则要求:

EMI:

FCC部分15,B类

EN55022:1994(基于CISPR-22:1993)

EN5082-1:1992(通用抗扰性标准)

VCCI(CISPR-22日本版)

安全性(Safety):

UL, CSA

### 7.3.2 总线定时/电气特性

表7-4、表7-5、表7-6和表7-7分别列出了DC条件下的电气特性以及全速率信源、低速率信源和集线器/中继器的定时/电气特性参数。

表7-4 DC电气特性

参数	符号	条件(注释1,2)	最小值	最大值	单位
电压供应:					
上电后的(主机后集线器)端口	$V_{bus}$		4.75	5.25	V
总线供电集线器端口	$V_{bus}$		4.40	5.25	V

(续)

参数	符号	条件(注释 1,2)	最小值	最大值	单位
电流供应:					
上电后的主机/集线器端口(输出)	I <sub>CCPRT</sub>		500		mA
总线供电集线器端口(输出)	I <sub>CCUPT</sub>		100		mA
大功率功能设备(输入)	I <sub>CCHPF</sub>			500	mA
低功率功能设备(输入)	I <sub>CCLPF</sub>			100	mA
未配置的功能设备/集线器(输入)	I <sub>CCINIT</sub>			100	mA
被挂起的设备	I <sub>CCS</sub>			500	μA
漏电流:					
高-Z 状态数据线漏电流	I <sub>L0</sub>	0 V < V <sub>IN</sub> < 3.3 V	-10	+10	μA
输入电平:					
差模输入灵敏度	V <sub>DI</sub>	((D+) - (D-)), 和图 7-4	0.2		V
差共模范围	V <sub>CM</sub>	包括 V <sub>DI</sub> 范围	0.8	2.5	V
单端接收器阈值	V <sub>SE</sub>		0.8	2.0	V
输出电平:					
稳定的输出低电平	V <sub>OL</sub>	1.5 kΩ 电阻负载接至 3.6V		0.3	V
稳定的输出高电平	V <sub>OH</sub>	15 kΩ 接至 GND	2.8	3.6	V
电容:					
接收器电容	C <sub>IN</sub>	引脚和 GND 之间		20	pF
下行集线器端口旁路电容	C <sub>HPC</sub>	V <sub>bus</sub> 和 GND 之间	120		μF
根端口旁路电容	C <sub>RPC</sub>	V <sub>bus</sub> 和 GND 之间注释 9	1.0	10.0	μF
终端:					
在根端口处的总线上拉电阻	R <sub>P0</sub>	(1.5 ± 5%) kΩ	1.425	1.575	kΩ
在下行端口处的总线下拉电阻	R <sub>PD</sub>	(15 ± 5%) kΩ	14.25	15.75	kΩ
电缆阻抗和时序:					
电缆阻抗(全速率)	Z <sub>0</sub>	(45 ± 15%) kΩ	38.75	51.75	Ω
电缆时延(单向)	T <sub>CBL</sub>	图 7-21		30	ns

表 7-5 全速率信源电气特性

参数	符号	条件(注释 1,2,3)	最小值	最大值	单位
驱动器特性:					
变化时间:					
上升时间	T <sub>R</sub>	注释 5.6 和图 7-16 CL = 50 pF	4	20	ns
下降时间	T <sub>F</sub>	CL = 50 pF	4	20	ns
上升/下降时间匹配	T <sub>RFM</sub>	(TR/TF)	90	110	%
输出信号交叉电压	V <sub>CBS</sub>		1.3	2.0	V
驱动器输出阻抗	Z <sub>DRV</sub>	稳定状态驱动	28	43	Ω

(续)

参数	符号	条件(注释 1,2,3)	最小值	最大值	单位
数据源时序:					
全速数据速率	T <sub>D RATE</sub>	平均比特速率 ( $12 \pm 0.25\%$ ) Mb/s	11.97	12.03	Mb/s
帧时间间隔	T <sub>FRAME</sub>	$1.0\text{ms} \pm 0.05\%$	0.9995	1.0005	ms
信源差模驱动器抖动 至下一个变化 对于成对变化而言	T <sub>DJ1</sub> T <sub>DJ2</sub>	注释 7,8 和图 7-29	-3.5 -4.0	3.5 4.0	ns ns
信源 EOP 宽度	T <sub>E OPT</sub>	注释 8 和图 7-30	160	175	ns
差模至 EOP 变化偏移	T <sub>DEOP</sub>	注释 8 和图 7-30	-2	5	ns
接收器数据抖动容限 至下一个变化 对于成对变化而言	T <sub>JR1</sub> T <sub>JR2</sub>	注释 8 和图 7-31	-18.5 -9	18.5 9	ns ns
接收器端的 EOP 宽度 必须作为 EOP 而丢弃 必须作为 EOP 而接受	T <sub>EOPR1</sub> T <sub>EOPR2</sub>	注释 8 和图 7-30	40 82		ns ns

表 7-6 低速率信源电气特性

参数	符号	条件(注释 1,2,4)	最小值	最大值	单位
驱动器特性:					
变化时间: 上升时间	T <sub>R</sub>	注释 5,6 和图 7-16 CL = 50pF CL = 350pF	75	300	ns ns
变化时间: 下降时间	T <sub>F</sub>	CL = 50pF CL = 350pF	75	300	ns ns
上升/下降时间匹配	T <sub>RFM</sub>	(TR/TF)	80	120	%
输出信号交叉电压	V <sub>CROS</sub>		1.3	2.0	V
数据源时序					
低速数据速率	T <sub>D RATE</sub>	平均比特速率 ( $1.5 \pm 1.5\%$ ) Mb/s	1.4775	1.5225	Mb/s
信源差模驱动器抖动 主机(下行方向): 至下一个变化 对于成对变化而言	T <sub>D DJ1</sub> T <sub>D DJ2</sub>	注释 7,8 和图 7-29	-75 -45	75 45	ns ns
功能设备(上行方向): 至下一个变化 对于成对变化而言	T <sub>UDJ1</sub> T <sub>UDJ2</sub>		-95 -150	95 150	ns ns
信源 EOP 宽度	T <sub>E OPT</sub>	注释 8 和图 7-30	1.25	1.50	μs
差模至 EOP 变化偏移	T <sub>DEOP</sub>	注释 8 和图 7-30	-40	100	ns
接收器数据抖动容限 在主机端(上行方向): 至下一个变化 对于成对变化而言	T <sub>JR1</sub> T <sub>JR2</sub>	注释 8 和图 7-31	-152 -200	152 200	ns ns

(续)

参数	符号	条件(注释 1,2,4)	最小值 使用本章相关知识 请参考	最大值	单位
在功能设备端(下行方向): 至下一个变化	$T_{DJR1}$		-75	75	ns
对于成对变化而言	$T_{DJR2}$		-45	45	ns
接收器端的 EOP 宽度 必须作为 EOP 而丢弃	$T_{EOPR1}$	注释 8 和图 7-30	330		ns
必须作为 EOP 而接受	$T_{EOPR2}$		675		ns

表 7-7 集线器/中继器电气特性

参数	符号	条件(注释 2)	最小值	最大值	单位
集线器特性(全速率):					
驱动器特性 (参见表 7-5)		根端口和以低速 率配置的下行端口			
集线器差模数据时延 (有电缆) (无电缆)	$T_{HDD1}$ $T_{HDD2}$	注释 3, 7, 8 和图 7-32		70 40	ns ns
集线器差模驱动器抖动(包括电 缆)		注释 3, 7, 8 和图 7-32			
至下一个变化	$T_{HDJ1}$		-3	3	ns
对于成对变化而言	$T_{HDJ2}$		-1	1	ns
SOP 后的数据比特宽度畸变	$T_{SOP}$	注释 3,8 和图 7-32	-5	3	
相对于 $t_{HDD}$ 而言, 集线器 EOP 时延	$T_{EOPD}$	注释 3,8 和图 7-33	0	15	ns
集线器 EOP 输出宽度偏移	$T_{HESK}$	注释 3,8 和图 7-33	-15	15	ns
集线器时序(低速率):					
驱动器特性(参见表 7-6)		以低速率配置 的下行端口			
集线器差模数据时延	$T_{LHDD}$	注释 4,7,8 和图 7-32		300	ns
集线器差模驱动器抖动(包括电 缆)		注释 4,7,8 和图 7-32			
下行方向: 至下一个变化	$T_{LDHJ1}$		-45	45	ns
对于成对变化而言	$T_{LDHJ2}$		-15	15	ns
上行方向: 至下一个变化	$T_{LUHJ1}$		-45	45	ns
对于成对变化而言	$T_{LUHJ2}$		-45	45	ns
SOP 后的数据比特宽度畸变	$T_{SOP}$	注释 4,8 和图 7-32	-60	45	
相对于 $t_{HDD}$ 而言, 集线器 EOP 时延	$T_{LEOPD}$	注释 4,8 和图 7-33	0	200	ns
集线器 EOP 输出宽度偏移	$T_{LHESK}$	注释 4,8 和图 7-33	-300	+300	ns
注					
① 除非另外声明,所有的电压都是根据本地地电位进行测量的。					
② 除非另外声明,所有的定时都使用对地为 50pF 的电容负载(CL)。					
③ 在 D+ 数据线上,所有的全速率定时方法都使用了 1.5kΩ 电阻接至 2.8V。					

(续)

- ④ 在 D+ 数据线上,所有的低速率定时方法都使用了  $1.5k\Omega$  电阻接至 2.8V。
- ⑤ 从数据信号的 10% 至 90% 的位置进行测量。
- ⑥ 上升和下降沿应该平滑地过渡(单调的)。
- ⑦ 在差模数据信号之间出现的定时差别。
- ⑧ 在差模数据信号的交叉点进行测量。
- ⑨ 最大负载说明指的是,在满足目标集线器的  $V_{bus}$  压降为 300mV 条件下的最大有效电容。

### 7.3.3 时序波形

图 7-29 至图 7-33 给出了几种情况下的时序波形,供读者参考。

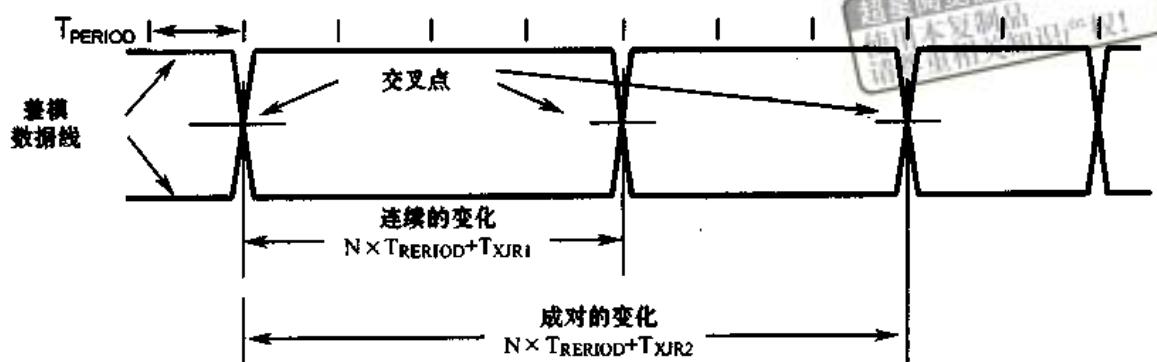


图 7-29 差模数据抖动

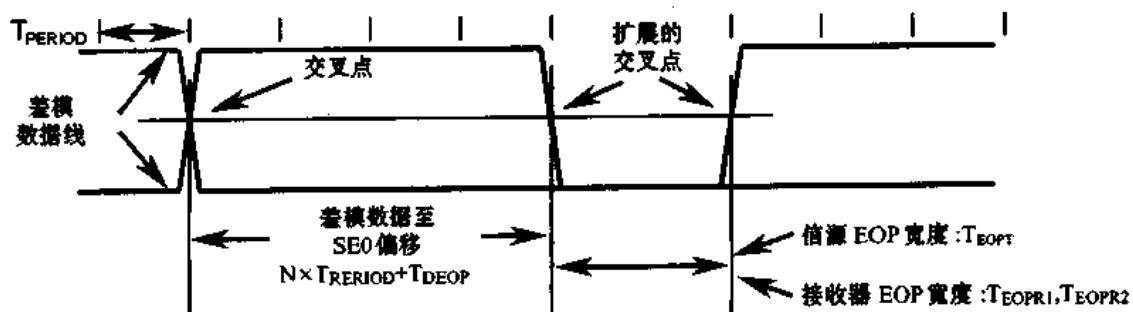
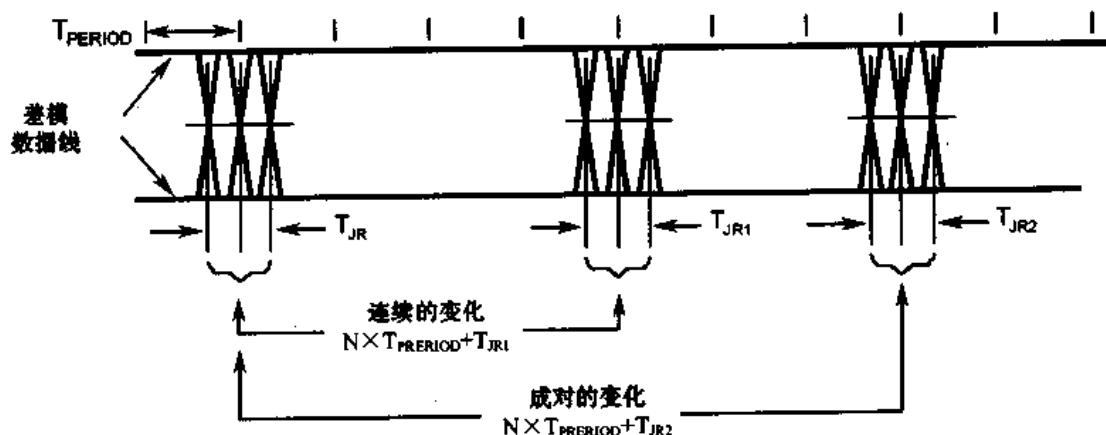
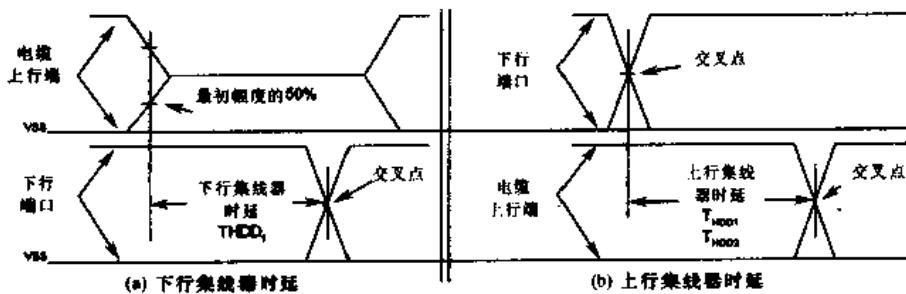


图 7-30 差模数据至 EOP 转变的相位偏移和 EOP 的宽度



注:请参见 7.1.11 节中有关  $T_{PERIOD}$  的定义。

图 7-31 接受器抖动容限



集线器差模抖动:

$$T_{HDV} = T_{HDDx}(J) - T_{HDDx}(K) \text{ 或 } T_{HDDx}(K) - T_{HDDx}(J) \text{ 连续的变化}$$

$$T_{HDK} = T_{HDDx}(J) - T_{HDDx}(J) \text{ 或 } T_{HDDx}(K) - T_{HDDx}(K) \text{ 成对的变化}$$

SOP 后的比特宽度畸变: (与对于 SOP 和接下来的 J 变化的数据抖动相同)

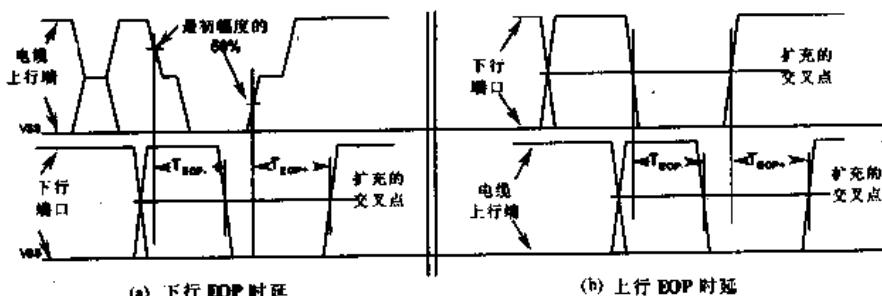
$$T_{HDJ} = T_{HDDx}(SOP) - T_{HDDx}(next J)$$

同样可以定义低速率定时如下:

$$T_{LHDD}, T_{LHDK}, T_{LDHDD}, T_{LDHDK}, T_{LSOP} \text{ 和 } T_{LSOP}.$$

超星浏览器提醒您:  
使用本资源制品  
请尊重相关知识产权!

图 7-32 集线器差模时延, 差模抖动和 SOP 失真



EOP 时延

$$T_{EOP} = T_{crossover}$$

EOP 偏移

$$T_{HEOP} = T_{EOP} - T_{EOP}$$

同样可以定义以下的低速率定时:

$$T_{LEOP} \text{ 和 } T_{UEOP}$$

图 7-33 集线器 EOP 时延和 EOP 相位偏移

# 第 8 章 协议层



本章将从域和分组定义开始,来说明  
作类型的分组处理格式。我们还要提到  
有关重试同步,串扰和总线操作恢复的内

首先送到总  
MSB 比特。在表  
取顺序)来显示分组的。

本章具体包括以下内容:



比特安排;



SYNC 域;



分组域格式;



分组格式;



处理格式。

所有的分组都由一个同步域  
连续变化的密度。SYNC 域是以  
形式出现在总线上的。其编码方  
号来排列输入数据。SYNC 域的  
有在下一节的表 8-1 中给出。  
第一个比特。分组中的所有后续

### 8.3 分组域格式

后面几节中将描述令牌、数据和握手分组的域格式。分组比特定义以未编码的数据格式给出。为了清晰起见,清除了 NRZI 编码和比特填充所产生的影响。所有的分组都具有清楚的分组开始和结束界限。分组开始(SOP)是 SYNC 域的一部分,而分组结束(EOP)分界则在第 5 章中进行了介绍。

#### 8.3.1 分组标识域

一个分组标识(PID)紧跟在每一个 USB 分组中的 SYNC 域之后。一个 PID 包括了一个 4bit 的分组类型域,再加一个 4bit 的校验域,如图 8-1 所示。PID 指出了分组类型,并且可据此推断出分组格式和该分组所应用的检错方式。PID 中的 4bit 校验域可以确保对 PID 进行可靠的解码,从而正确地翻译分组中的其余部分。通过对分组类型域内实现一个的数值取 1 的补码,就可以得到 PID 校验域。

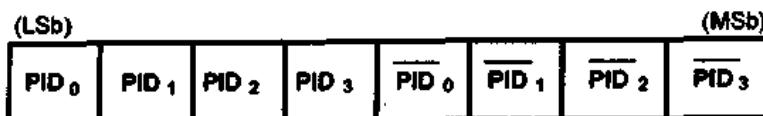


图 8-1 PID 格式

主机和所有的功能设备必须能对所有接收到的 PID 域进行完整的解码。如果任意一个接收到的 PID 的检验域出现错误或其解码后为一个未定义的值,就认为其受到了破坏。那么该 PID 及分组的其余部分都将被分组接收器所忽略。如果一个功能设备收到了一个正确的 PID,但其操作操作类型或方向却不能被这个设备所支持,该功能设备就不能做出响应。例如,一个仅用于输入的端口就必须忽略一个 OUT 令牌。PID 类型、编码和有关说明都列在了表 8-1 中。

PID 可以分成四个码组:令牌、数据、握手和特殊类型,其中所传输的前两个 PID 比特用于指示码组类型。这就解释了 PID 编码的分配问题。

表 8-1 PID 类型

PID 类型	PID 名称	PID[3:0]	说 明
令牌	OUT	b0001	地址 + 主机中的端点号 → 功能设备处理
	IN	b1001	地址 + 功能设备中的端点号 → 主机处理
	SOF	b0101	帧标记和帧标号开始
	SETUP	b1101	地址 + 主机中的端点号 → 用于一个控制端点建立的功能设备处理
数据	DATA0	b0011	数据分组偶 PID
	DATA1	b1011	数据分组奇 PID
握手	ACK	b0010	接收器接受无错的数据分组
	NAK	b1010	接收设备不能接受数据或发送设备不能发送数据
	STALL	b1110	端点被禁止
特定	PRE	b1100	主机发出前同步信号。激活至低速率设备的下行总线数据流



### 8.3.2 地址域

可以利用两个域对功能设备端点进行寻址：功能设备地址域和端点域。一个功能设备需要能对地址和端点域进行完全的解码。不允许将地址或端点混淆。任一域内出现的不匹配都会使这一令牌被忽略。同样，访问一个未初始化的端口也会使令牌被忽略。

地址域到底是什么意思呢？

功能设备地址(ADDR)域通过其地址而对功能设备加以区别，它可以利用令牌 PID 的值来说明该功能设备是一个数据分组的发送者还是接收者。如图 8-2 所示，ADDR <6:0> 可以说明全部 128 个地址。ADDR 域可以指定用于 IN、SETUP 和 OUT 令牌。根据规定，每个 ADDR 值都定义了一个功能设备。当复位和上电时，设备地址缺省为 0，并且必须在枚举过程中由主机来对其进行编程。缺省地址 0 为缺省状态所保留，不能将其分配给一般的操作使用。

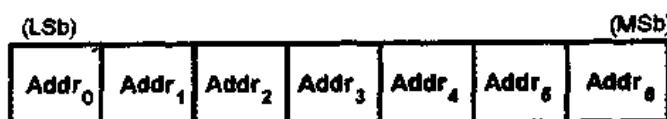


图 8-2 ADDR 域

### 8.3.3 端点域

如图 8-3 所示，一个附加的 4 比特端点(ENDP)域允许对那些需要不只一个子信道的功能模块进行更灵活地寻址。端点号视功能设备而定。规定端点域只能供 IN、SETUP 和 OUT 令牌的 PID 使用。所有的功能设备必须支持一个控制端点 0。低速率设备中每个功能模块最多可以支持两个端点地址；0 再加上另外一个端点。全速率功能模块最多可以支持 16 个端点。

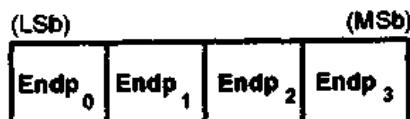


图 8-3 端点域

### 8.3.4 帧标号域

帧标号域有 11bit 长，每过一帧主机都要对其加 1。当达到了它的最大值  $0 \times 7FF$  后，帧标号域就开始循环，而且只在一帧开始时向 SOF 令牌发送。

### 8.3.5 数据域

数据域的范围可以从 0 到 1023B，并且必须具有整数个字节。图 8-4 给出了多个字节的格式。每一个字节中的数据比特都是首先移出 LSB 比特。

数据分组尺寸将根据第 5 章所讲述的传输类型而变化。

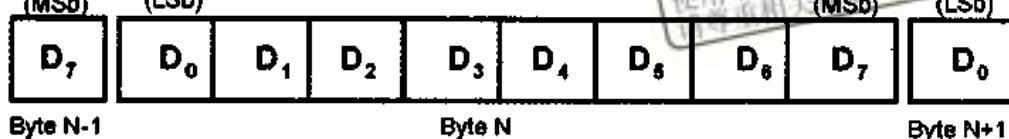


图 8-4 数据域格式

### 8.3.6 循环冗余检验

循环冗余校验(CRC)用于对令牌和数据分组中的所有非 PID 域进行保护。在本文中,这些域被认为是受到保护的域。PID 不包括在一个含有 CRC 的分组中的 CRC 校验中。在进行比特填充之前,发送器内所有的 CRC 都会通过其各自的域而产生。类似地,在接收器内,只有在去除了填充比特之后才能进行 CRC 解码。令牌和数据分组 CRC 提供了对所有单个和双比特错误的 100% 的覆盖。一个 CRC 错误被认为是一个或多个受保护的域遭到了破坏,从而使接收器忽略这些域,并且在多数情况下是忽略整个分组。

对于 CRC 生成和校验来说,在生成器和校验器内的移位寄存器都是以全 1 方式开始的。对于每一个接收的或发送的数据比特,都需要将当前剩余的高阶比特同数据比特进行异或(XOR),然后将余数左移 1 位并且将低阶比特置为 0。如果异或后的结果为 1,那么余数就要与生成多项式进行异或。

当校验域内的最后一个比特发送之后,生成器内的 CRC 将会倒换位置,并首先向校验器发送 MSB 比特。当一个校验器接收到 CRC 的最后一个比特并且没有出现错误时,余数将等于多项式余数。

CRC 也必须满足比特填充的要求,这意味着如果前面 6 个 bit 为全 1,就需要在 CRC 末尾插入一个零。

#### 1. 令牌 CRC

一个 560 bit 的 CRC 用于令牌,它涵盖了 IN、SETUP 和 OUT 令牌内的 ADDR 和 ENDP 域,或者是一个 SOF 令牌中的时间标记。其生成多项式为:

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

用二进制比特方式表示这一多项式,就应该是 00101。如果接收到的全部令牌比特都无误,那么接收器所得到的 5bit 余数为 01100。

#### 2. 数据 CRC

对一个数据分组的数据域,我们采用了一个 16bit 的多项式。其生成多项式为:

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

用二进制比特方式表示这一多项式,就应该是 1000000000000101。如果接收到的全部令牌比特都无误,那么接收器所得到的 16bit 余数为 1000000000001101。

## 8.4 分组格式

本节给出了一个令牌、数据和握手分组的分组格式。我们按照图中所示的,比特移出总线的顺序来显示一个分组中的多个域。



### 8.4.1 令牌分组

图 8-5 给出了一个令牌分组的域格式。一个令牌包括一个用于区别是 IN、OUT 或 SETUP 分组类型的 PID 域、ADDR 及 ENDP 域。对于 OUT 和 SETUP 处理操作而言，地址和端点域可以唯一地标识一个将接收后续数据分组的端点。对于 IN 处理操作而言，这些数据域将唯一地标识一个即将发送一个数据分组的端点。只有主机才能发送令牌分组。INPIN 定义了一个由一个功能模块至主机的数据处理操作。OUT 和 SETUPPIP 则定义了由主机至一个功能模块的处理操作。

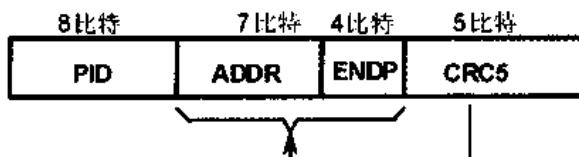


图 8-5 令牌格式

如上图所示，令牌分组还包括了一个 5 比特的 CRC 域，涵盖了地址和端点域。CRC 没有包括 PID，因为 PID 有其自己的校验域。令牌和 SOF 分组利用位于三个字节的分组数据之后的 EOP 来进行定界。如果一个分组解码为另外一个有效令牌或 SOF，但是却没有以三个字节之后的一个 EOP 来终止，就会认为该分组不正确，并被接收器所忽略。

### 8.4.2 帧开始分组

帧开始分组(SOF)由主机以每  $1.00 \pm 0.05\text{ms}$  一次的标称速率进行发送。如图 8-6 所示，SOF 分组包含一个用于指示分组类型的 PID，后面还跟有一个 11 比特的帧标号域。



图 8-6 SOF 分组

SOF 令牌由仅对令牌进行处理的操作组成，它在依据每一帧的开始而精确定时的时间间隔内分配一个帧开始标记和帧标号。所有的全速率功能设备，包括集线器，都必须对 SOF 分组进行接收和解码。SOF 令牌不会使任何进行接收的功能设备产生一个返回分组。因此，不能保证 SOF 可以传送到某个给定的功能模块。SOF 分组可以传递两条定时信息。当一个功能设备检测到 SOF PID 时它就被告知帧已经开始了。对帧定时功能设备只需对 SOF PID 进行解码，而无须了解帧标号，它可以忽略帧标号及其 CRC。如果一个功能设备需要跟踪帧标号，它就必须理解 PID 和时间标记。

### 8.4.3 数据分组

如图 8-7 所示，一个数据分组包括一个 PID，一个数据域和一个 CRC 域。不同的 PID 可以区别两种类型的数据分组：DATA0 和 DATA1。USB 规范定义了两个数据分组

PID 来实现数据触发同步(参见 8.6 节)。

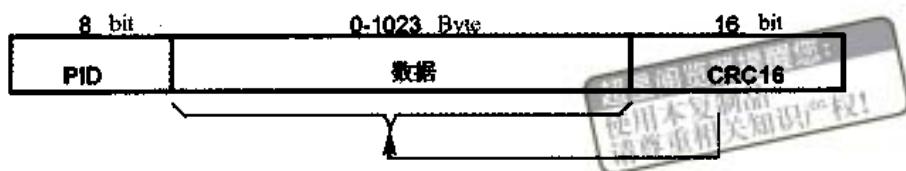


图 8-7 数据分组格式

数据必须以整数个字节进行发送。数据 CRC 仅对分组中的数据域进行计算,而不包括 PID,因为 PID 有自己的校验域。

#### 8.4.4 握手分组

如图 8-8 所示,握手分组仅包含一个 PID。握手分组用于报告一个数据处理操作的状态,并能够返回数值来指示数据的成功接收、流量控制和停止条件。只有支持流量控制的处理操作类型才能返回握手信息。握手分组总是在一个数据处理操作中的握手阶段返回,而且可以代替数据在数据阶段返回。握手分组由位于一个字节的分组域之后的一个 EOP 来定界。如果一个分组解码为另外一个有效的握手分组,但是却没有以一个字节之后的一个 EOP 来终止,就会认为该握手分组不正确,并被接收器所忽略。

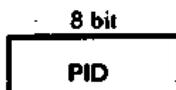


图 8-8 握手分组

总共有三种类型的握手分组:

(1) ACK 说明接收的数据分组在整个数据域没有比特填充或 CRC 错误,而且数据 PID 也是正确地接收。在比特序列匹配并且接收器能够接收数据时,或者是在比特序列不匹配并且发送器和接收器必须互相进行重新同步时,可以发送 ACK 分组。一个 ACK 握手分组只在那些已经发送了数据或者期待握手分组的处理操作中才适用。ACK 既可以由主机为了 IN 处理操作而返回,也可以因为 OUT 处理操作而由一个功能设备返回。

(2) NAK 说明一个功能设备不能从主机(OUT)接收数据或一个功能设备没有数据可以向主机(IN)传输。NAK 只能由功能设备在 IN 处理中的数据阶段或 OUT 处理中的握手阶段返回。主机永远不能发送一个 NAK 分组。NAK 用于实现流量控制,指出一个功能设备暂时不能传输或接收数据,但是它最终可以进行数据传输或接收操作而无须主机干涉。NAK 也可以由中断端点用来指明没有中断等待处理。

(3) STALL 由一个功能设备对一个 IN 令牌作出响应时,或在一个 OUT 处理之后的数据阶段返回(参见图 8-9 和图 8-13)。STALL 说明一个功能设备不能传输或接收数据,该功能设备必须继续返回 STALL 分组,直到通过主机的干涉而清除了产生停止的条件。在任何情况下都不允许主机返回一个 STALL 分组。

#### 8.4.5 握手响应

完成传输和接收功能的功能设备,必须根据表 8-2 至表 8-4 所说明的先后次序来

返回握手分组。不是所有的握手分组都被允许使用。这依赖于处理操作类型和该握手分组是由一个功能设备还是由主机发出的。

表 8-2 功能设备对 IN 处理的响应

接收的令牌分组被破坏	功能设备发送端点被禁止	功能设备可以发送数据	采取的操作
是	无关	无关	不返回任何响应
否	是	无关	发送 STALL 握手
否	否	否	发送 NAK 握手
否	否	是	发送数据分组

### 1. 功能设备对 IN 处理的响应

表 8-2 给出了一个功能设备对一个 IN 令牌作出响应时,可能使用的响应。如果该功能设备由于一个停止或流量控制条件而不能发送数据。它就会分别发送一个 STALL 或 NAK 握手分组。如果该设备可以发送数据,它就不会进行上述操作。如果接收到的令牌已经被破坏,功能设备就不应作出任何响应。

### 2. 主机对 IN 处理的响应

表 8-3 给出了主机对一个 IN 处理的响应。主机只能返回一种类型的握手分组,即一个 ACK,如果主机收到了一个遭破坏了的数据分组,它会抛弃数据并且不发送任何响应。如果主机不能从一个功能设备接收数据(由于像内部缓冲区溢出这样的问题),这一情况就会被认为是一个错误,而主机则不会返回任何响应。如果主机可以接收数据,并且所接收到的分组没有任何错误。那么主机就会接受该数据并发送一个 ACK 握手分组。

表 8-3 主机对 IN 处理的响应

数据分组被破坏	主机可以接受数据	主机返回的握手
是	N/A	丢弃数据,不返回任何响应
不	不	丢弃数据,不返回任何响应
不	是	接受数据,发送 ACK

### 3. 功能设备对一个 OUT 处理的响应

对于一个 OUT 处理的握手响应示于表 8-4。一个功能设备,当其接收到一个数据分组时,可以返回三种握手类型中的任意一种。如果数据分组被破坏,该功能设备不会返回任何握手分组。如果收到的数据分组没有错误,而该功能设备的接收端点被禁止,它就会返回一个 STALL 握手分组,如果该处理操作保持了比特序列同步并且检测到了一个不匹配之处(详细内容请参考 5.6 节),那么该功能模块就会返回 ACK 并丢弃这个数据。如果一个功能设备可以接收数据,并且它所接收到的数据没有错误,它就会返回一个 ACK 握手分组。如果由于流量控制的原因使一个功能设备不能接收数据分组,它就会返回一个 NAK 分组。

### 4. 功能设备对一个 SETUP 处理的响应

创建操作定义了由主机至功能设备的一种特殊类型的处理操作,这种处理操作允许主机将一个端点的同步比特初始化为主机的同步比特。一旦收到了一个创建处理操作,功能设备必须接受该数据。创始处理操作不能被禁止或不予回应。进行接收的功能设备

必须接受创建操作所传送的数据。如果一个非控制端点收到了一个 SETUP PID, 它必须忽略该处理操作并且不能返回任何响应。

表 8-4 功能设备对一个 OUT 处理的响应(按照先后顺序排列)

数据分组被破坏	接收器被禁止	序列比特不匹配	功能设备可以接受数据	功能设备返回的握手
是	N/A	N/A	N/A	无
不	是	N/A	N/A	STALL
不	不	是	N/A	ACK
不	不	不	是	ACK
不	不	不	不	NAK

## 8.5 处理格式

根据端点的类型, 分组处理格式会有所不同。总共有四种端点类型: 批量、控制、中断和同步。

### 8.5.1 批量处理操作

批量处理操作类型的特性是: 通过差错检测和重试方法, 它可以保证在主机和一个功能设备之间进行无差错的数据传输。批量处理操作利用了一个具有三个阶段的处理操作, 包括令牌、数据和握手分组, 如图 8-9 所示。在一定的流量控制和停止条件下, 数据阶段可以被一个握手分组所取代, 从而产生一个只有两个阶段的处理操作, 其中没有数据被传送。

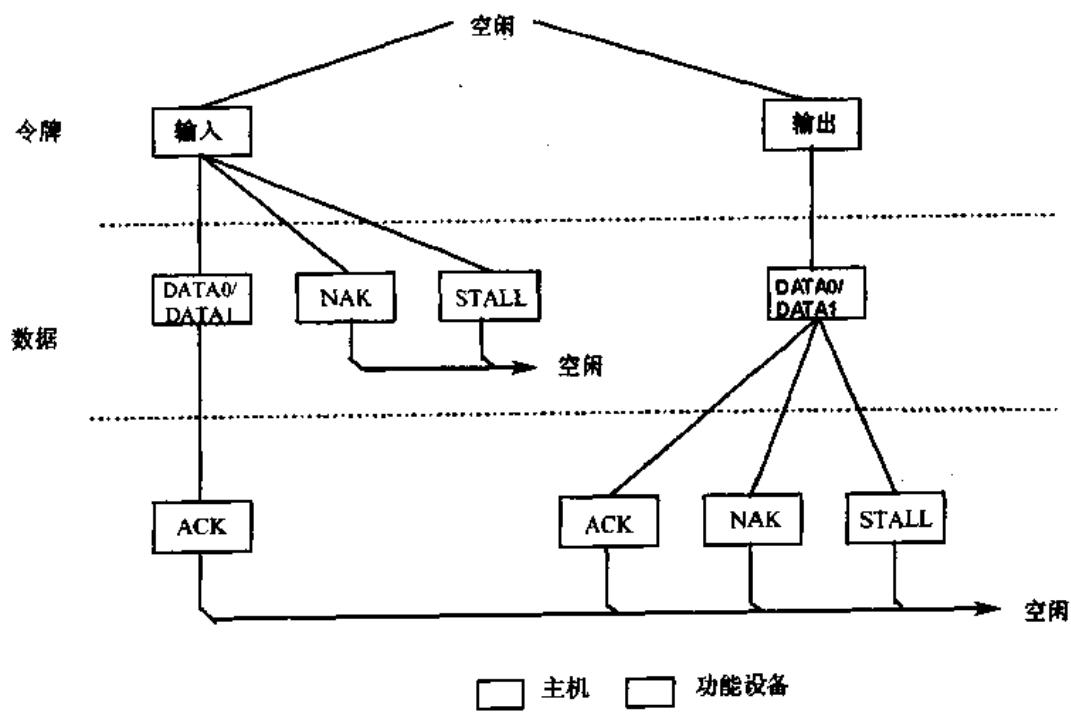


图 8-9 批量处理格式

当主机希望接收批量数据时,它就会发出一个 IN 令牌。功能设备端点要么返回一个数据分组,或者要是它不能返回数据的话,就返回一个 NAK 或 STALL 握手分组来作为响应。一个 NAK 说明该功能设备暂时还不能接收数据,而一个 STALL 则说明该端点已经被永远禁止,需要主机软件干预。如果主机收到了一个正确的数据分组,它就会以一个 ACK 握手来响应。如果主机在接收数据时检测到了一个错误,它就会向功能设备返回一个握手分组。

当主机希望传输批量数据时,它首先发出一个跟有一个数据分组的 OUT 令牌。功能设备会返回三个握手分组中的一个分组。ACK 说明数据分组已经被无误地接收,并通知主机它可以按顺序发送下一个分组。NAK 说明接收到的数据没有错误,但是由于该功能设备暂时处于一种阻止其在该时刻接受数据的状态,所以主机应该重新发送这一数据(如果缓冲区被填满)。如果端点被禁止,就会返回 STALL 分组来指出;由于功能设备上存在一个错误,主机不应对该传输进行重试操作。如果接收到的数据分组存在一个 CRC 或比特填充错误,就不会返回任何握手分组。

图 8-10 给出了用于批量读和批量写的比特序列和数据 PID。通过使用数据序列触发比特和 DATA0/DATA1 PID,可以获得数据分组同步。批量端点必须有其自己的经过一个分离的控制端点而初始化的触发比特序列。

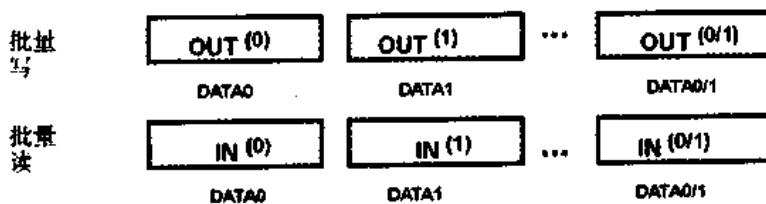


图 8-10 批量读写

主机总是将一个总线传输的第一个处理操作初始化为 DATA0 PID。第二个处理操作使用一个 DATA1 PID,接下来的数据传输交替使用余下的批量传输。一旦接收到了 ACK,数据分组发送器就触发,而接收器在收到并接受了一个正确的数据分组时触发(参见 8.6 节)。

### 8.5.2 控制传输

控制传输最少具有两个处理阶段:创建和状态。一个控制传输可以选择在创建和状态阶段之间再包含一个数据阶段。在创建阶段,一个创建处理操作用于向一个功能设备的控制端点传送信息。创建处理操作所具有的格式与一个 OUT 处理很相似。但是它使用的是 SETUP 而不是 OUT PID。图 8-11 给出了创建处理操作格式。一个创建处理操作总是对其数据域使用一个 DATA0 PID。接收一个创建处理的功能设备必须接受创建处理数据并用一个 ACK 握手分组来响应,或者如果这一数据被破坏了,它必须丢弃该数据并且不返回任何握手分组。

如果一个控制传输内有数据阶段的话,那么这个数据阶段就包含了一个或多个 IN 或 OUT 处理,并且要遵循和批量传输一样的协议原则。在数据阶段所有的处理操作都必须具有同样的方向,即全部为 IN 或全部为 OUT。在数据阶段所传送的数据量及其方

超量阅读  
使用本复制品  
请尊重相关知识产权!

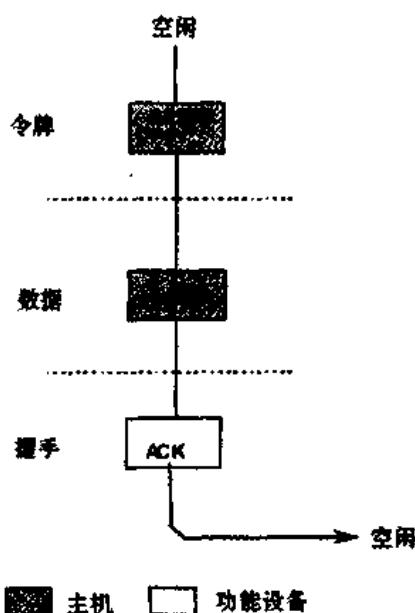


图 8-11 控制创建处理操作

向都是在创建阶段加以说明了的。如果数据量超过了预先确定的数据分组大小,那么数据就会在多个装载了最大分组大小的处理操作内传送(IN 或 OUT)。任何剩余的数据都将作为剩余量而在最后一个处理操作内发送。

一个控制传输中的状态阶段是该序列中的最后一个操作。一个状态阶段由相对于前一个阶段数据流方向的变化来描述,并且总是使用一个 DATA1 PID。例如,如果一个数据阶段包含有 OUT,那么状态就是一个单一的 IN 处理操作。如果一个控制序列没有数据阶段,那么它就会包括一个创建阶段,再跟上一个包含了一个 IN 处理的状态阶段。图 8-12 说明了处理顺序、数据序列比特值和用于控制读和写序列的数据 PID 类型。序列比特在括号内给出。

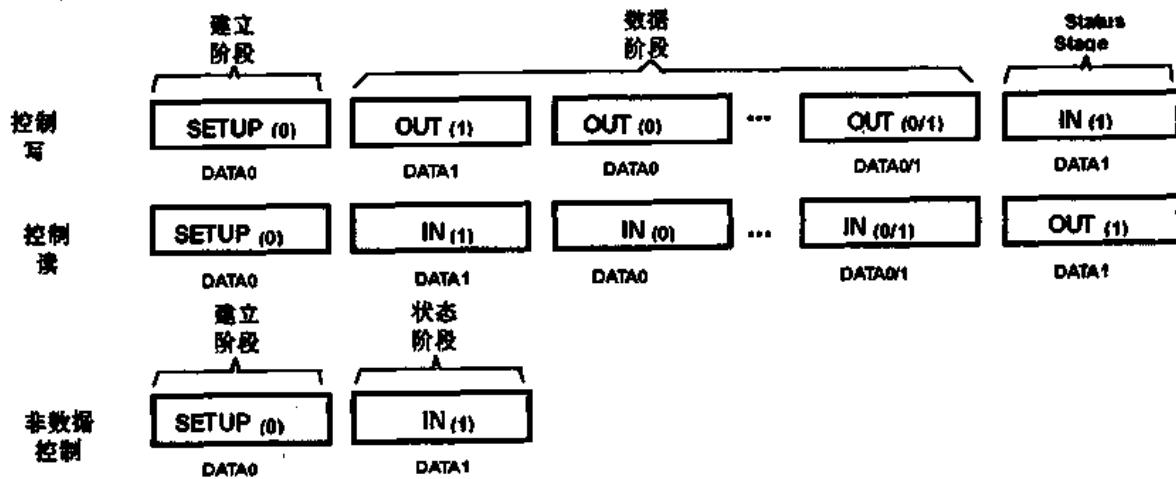


图 8-12 控制读写顺序

### 1. 报告状态结果

状态阶段向主机报告该传输中前一个创建和数据阶段的输出。可能返回的三种结果是：

- (1) 命令序列成功地完成

- (2) 命令序列未能完成  
 (3) 功能设备正在忙着完成命令

报告状态总是具有从功能设备到主机的方向。表 8-5 对每一种结果所要求的响应类型作了总结。控制写传输在该传输中的数据阶段返回状态信息。在前一个数据阶段内主机发送了一个长度为 0 的数据之后, 控制读传输将在握手阶段返回状态信息。

对于控制读, 主机向控制端点发送一个零长度的数据分组。端点作出握手响应说明状态阶段结束。NAK 说明该功能设备仍在处理命令, 而主机应该继续状态阶段。ACK 说明该功能设备已经完成了这一命令, 并为接收一个新的命令作好了准备。而 STALL 则说明该功能设备出现了一个阻碍其完成命令的错误。

表 8-5 状态阶段响应

状态响应	控制写传输(在数据阶段发送)	控制读传输(在握手阶段发送)
功能设备完成	0 长度数据分组	ACK 握手
功能设备出错	STALL 握手	STALL 握手
功能设备正忙	NAK 握手	NAK 握手

对于控制写而言, 功能设备不是用一个握手, 就是用一个 0 长度的数据分组来响应, 以指出其状态。一个 NAK 说明该功能设备仍在处理命令, 而主机应该延续状态阶段。返回一个 0 长度分组则说明命令已经正常的完成, 而 STALL 则说明该功能设备出现了一个阻碍其完成该命令的错误。在数据阶段返回一个 0 长度数据分组的控制写传输, 会使主机向该功能设备返回一个 ACK 握手分组。

在一个数据或状态阶段, 如果向一个命令端点发送了或要求一个命令端点返回多于在创建阶段所指明的数据, 它就应该返回一个 STALL。如果一个控制端点在数据阶段返回了 STALL, 那么该控制传输就没有状态阶段。

## 2. 对最后一组数据处理的差错控制

如果一个 IN 处理上的 ACK 握手分组被破坏, 功能设备和主机对于该处理是否成功的看法会暂时不一致。如果这个处理还跟有另外一个 IN 处理, 触发重试机制会检测到不匹配之处并从错误中恢复出来。如果 ACK 位于一个控制传输的最后一个 IN 处理之内, 就不能使用触发重试机制, 而必须采用另一方案。

主机如果成功地接收到了最后一个 IN 处理内的数据, 它就会发出一个 OUT 创建传输。而且功能设备一旦发现令牌方向发生了改变, 它就会将这一动作解释为主机已经成功地接收到了该数据。换句话来说, 功能设备会把令牌方向的变化, 作为主机已经成功地接收到了最后一个 ACK 握手的确凿证据来对待。因此, 当功能设备发现了一个 OUT 建立操作时, 它就会进入状态阶段。

控制写没有这一模糊的意义。主机通过接收握手分组, 可以确定最后一个处理操作是否成功地进行了。如果一个 OUT 处理上的一个 ACK 握手分组被破坏, 主机就不会进入状态阶段, 而会转而重试最后一个数据, 8.6.4 节将详细地分析重试原则。

### 8.5.3 中断处理操作

中断处理操作仅包括一个 IN 处理, 如图 8-13 所示。一旦收到了一个 IN 令牌, 一个功

能设备就可以返回数据、NAK 或 STALL。如果该端点没有新的中断信息可供返回,即没有中断等待处理,该功能设备就会在数据阶段返回一个 NAK 握手分组。如果一个被禁止了的端点被永远禁止并且需要主机来进行软件干预,它就会使功能设备返回一个 STALL 握手分组。如果一个中断等待处理,那么该功能设备就会把中断信息作为一个数据分组返回。作为对接收到的数据分组的响应,如果收到的主机没有错误,主机就会发送一个 ACK 握手分组,或者如果所接收的数据分组被破坏,主机就不会返回任何握手分组。

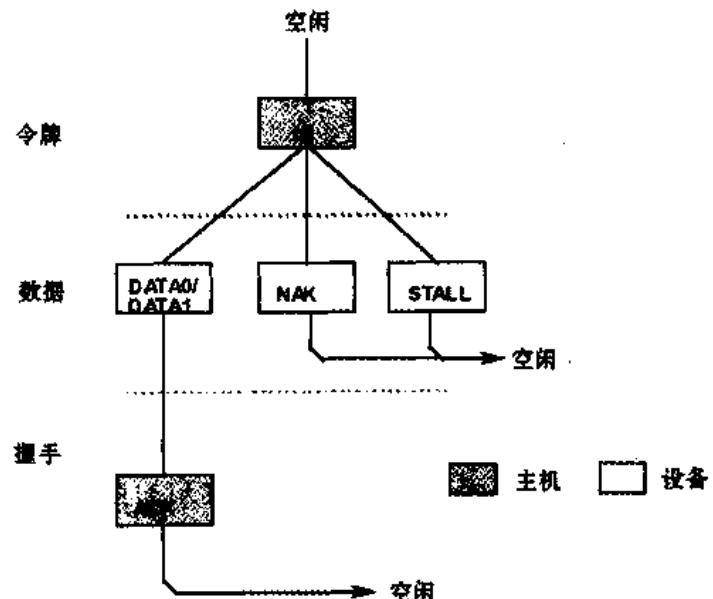


图 8-13 中断处理格式

当一个端点对实际的中断数据使用中断传输机制后,它就必须满足数据触发协议。这允许该功能设备知道数据是否已经被主机成功接收,并且该事件条件可以被清除。这样一种得到保证的传输可以允许功能设备在主机接收到数据之后才发送中断信息,而不是在每一次允许该功能设备传输和直到主机软件清除了中断条件之后才发送中断数据。如果在触发模式中使用,中断端点就应该被初始化为 DATA0 PID,并且像图 8-10 所示的批量 IN 处理一样工作。

一个中断端点也可以用于为一定类型的同步功率设备进行速率反馈信息的通信。当应用于这一模式时,在每一个数据分组发送到主机之后,数据触发比特就应该改变,而不考虑握手分组的存在或类型。

#### 8.5.4 同步处理操作

ISO 处理操作有一个令牌和数据阶段,但没有握手阶段,如图 8-14 所示。主机不是发送一个 IN 令牌,就是发送一个 OUT 令牌,后面还跟有一个数据阶段。在这一阶段端点(对于 IN 处理而言)或主机(对于 OUT 处理而言)将进行数据传输。ISO 处理操作不支持一个握手阶段或重试功能。

ISO 处理操作不支持触发排序,所以数据 PID 总是 DATA0。分组接收器不会检测数据 PID。

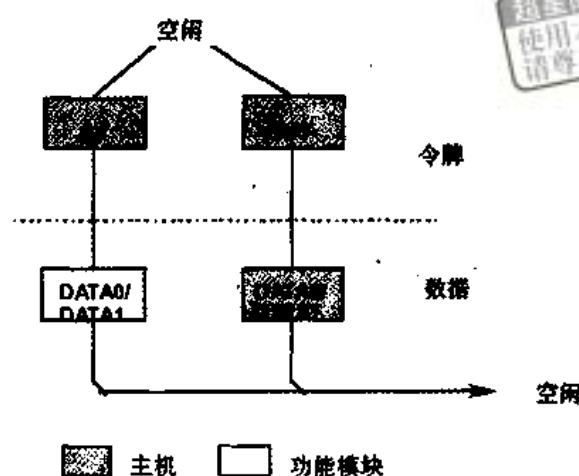


图 8-14 同步处理格式

## 8.6 数据触发同步和重试

USB 提供了一个机制来保证经过了多个处理操作之后，在数据发送器和接收器之间仍能保持数据序列同步。这一机制提供了一个方法来保证发送器和接收器都能正确地解释一个处理操作内的握手阶段。通过使用 DATA0 和 DATA1，以及用于数据发送器和接收器的分离的数据触发比特序列，可以实现同步。只有当接收器可以接受数据并且接收到的数据分组具有正确的数据 PID 时，接收器比特序列才会触发。只有当数据发送器接收了一个有效的 ACK 握手分组之后，发送器比特序列才能触发。数据接收器和发送器必须在一个处理操作开始时，实现其比特序列的同步。所采用的同步机制根据处理操作类型而发生变化。ISO 传输不支持数据触发同步。

### 8.6.1 通过 SETUP 令牌进行初始化

控制传输利用 SETUP 令牌来初始化主机和功能设备的比特序列。图 8-15 说明了

主机                  设备

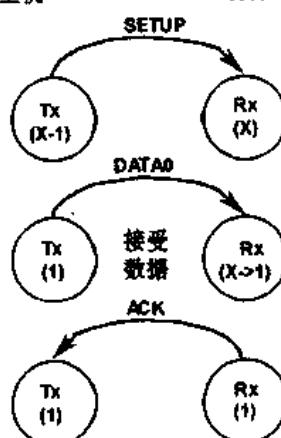


图 8-15 SETUP 初始化

主机向一个功能设备发送了一个 SETUP 分组,后面跟有一个 OUT 处理的情况。圆圈内的数字代表发送器和接收器的比特序列。该功能设备必须接受这一数据,并对该处理操作做出应答(ACK)。当功能设备收到了处理操作时,它必须重新设置它的比特序列,从而使主机和该功能设备在 SETUP 处理操作结束时,其比特序列都等于 1。

### 8.6.2 成功的数据处理操作

图 8-16 说明了进行了两次成功的处理后的情况。对于数据传送器而言,这意味着它在收到一个 ACK 之后就触发其比特序列。只有当接收器接收了一个正确的数据分组,并且分组数据 PID 与接收器比特序列相匹配时,它才会触发其比特序列。

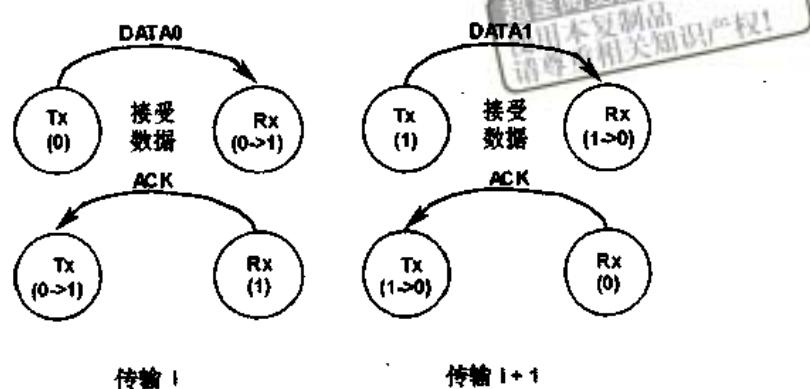


图 8-16 连续的处理操作

在每一个处理操作期间,接收器都会将自己的接收比特序列同发送器比特序列(在数据分组 PID 中进行编码,不是 DATA0 就是 DATA1)。如果不能接受数据,接收器必须发送一个 NAK 分组。如果可以接受数据,并且接收器序列比特与 PID 序列比特相匹配,那么数据就会被接受。只有当一个数据分组被传输时,序列比特才可以变化。一个没有数据分组的两阶段处理操作,会使发送器和接收器序列比特保持不变。

### 8.6.3 数据被破坏或不能接受

如果数据不能被接受或接收到的数据被破坏了,接收器就会发送一个 NAK 或 STALL 握手或超时,这将根据环境而定,而且接收器不会触发其序列比特。图 8-17 说

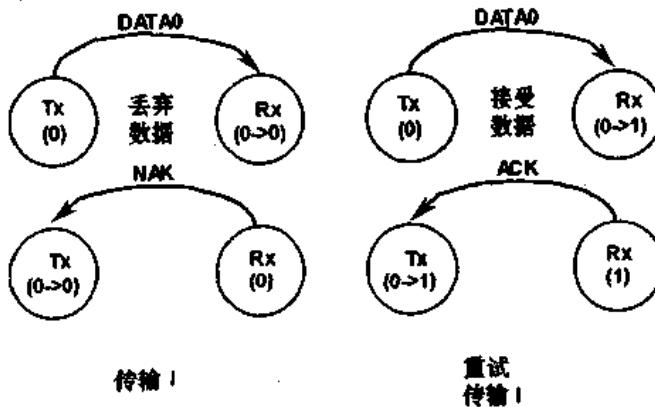


图 8-17 否定应答处理操作和重试操作

明了一个处理操作被否定,然后被重试的情况。任何一个否定应答(NAK)握手或超时都会产生类似的重试操作。一个没有收到一个ACK握手分组的发送器不会触发其序列比特。其结果是,一个失败了的数据分组处理操作并不会使发送器和接收器序列比特同步被触发。然后,该处理操作会被重试,并且如果成功的话,会使发送器和接收器的序列比特都被触发。

#### 8.6.4 破坏了的 ACK 握手分组

由于发送器可以接收一个ACK握手分组,所以它是最后一个并且是唯一一个可以确定处理操作是否成功进行的主体。一个丢失了的或被破坏了的ACK握手分组会使发送器和接收器暂时失步,如图8-18所示。这里发送器发送了一个有效的数据分组,并被接收器成功获得;但是,ACK握手分组却被破坏了。

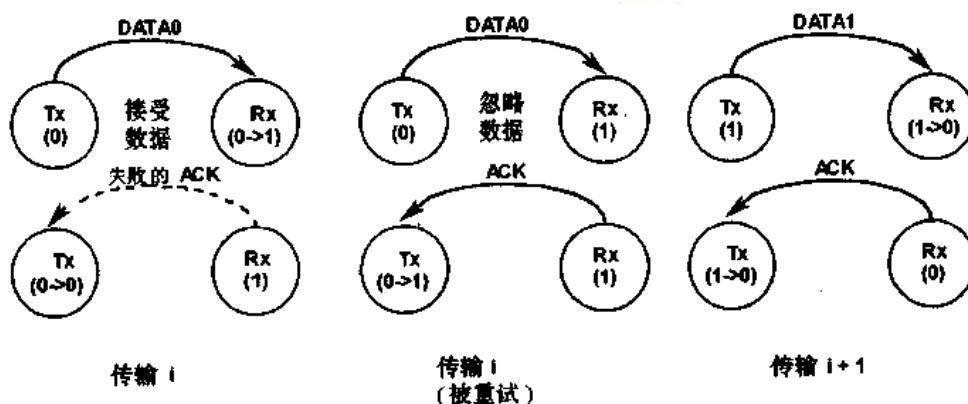


图8-18 破坏了的ACK握手分组和重试操作

在处理操作*<i>*结束时,发送器和接收器会暂时失去相干性,这可以由它们各自的序列比特之间存在失配来证明。接收器已经收到了完好的数据,但发送器不知道它是否成功地发送了数据。在下一个处理操作,发送器会重新发送上一个数据,并使用前一个DATA0 PID。接收器序列比特和数据PID不匹配,所以接收器就知道先前它已经接受了该数据。因此,它就会丢弃这一输入数据,而且不会触发其序列比特。然后,接收器会发送一个ACK分组,这会使发送器认为重试的处理操作已经成功地进行了。收到了ACK就会使发送器触发其序列比特。在*<i+1>*处理操作开始时,序列比特已经触发过了,并再次实现了同步。

数据发送器必须保证任一被重试的数据分组和原来的处理操作内发送的数据分组具有同样的长度。如果由于出现像缓冲区欠载运行这样的问题,而使数据发送器不能传送同最初的数据分组数量相同的数据,那它必须通过产生一个比特填充违规错误来终止该处理操作。这将在接收器上产生一个可检测到的错误,并且保证一个部分分组不会被解释为一个完好的分组。发送器不能试图通过发送一个已知的坏CRC而在接收器上强加一个错误。一个错误分组和一个错误CRC域的混合可能会被接收器解释为一个完好的分组。

#### 8.6.5 低速处理操作

USB支持以两种速率发送信号:全速率信号为12Mb/s,而低速率信号为1.5 Mb/s。

在发送全速率下行信号期间,集线器会对那些流向所有接有低速率设备的端口的下行总线业务量予以禁止。这即是 EMI 原因所决定的,也是阻止产生任何可以使一个低速率设备将一个下行全速率信号错误地解释成对其进行寻址的信号的可能性所要求的。图 8-19 给出了一个 IN 低速率处理操作,其间主机发送了一个令牌和握手分组,并接收了一个数据分组。

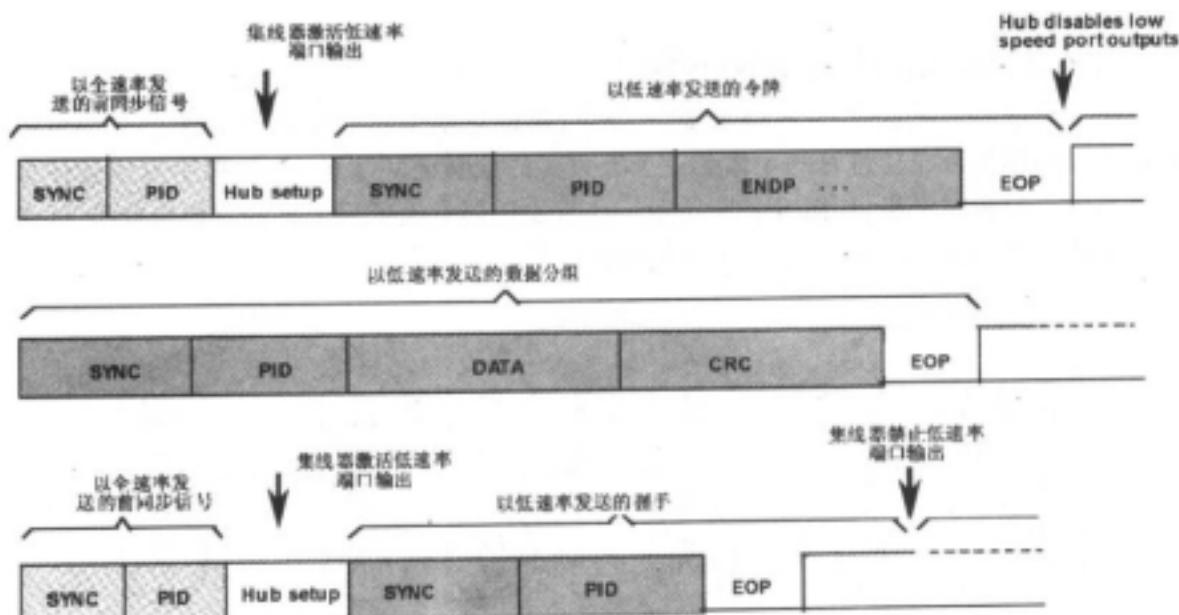


图 8-19 低速率处理操作

所有传向低速率设备的下行分组都需要有一个前同步信号。该前同步信号包括了一个跟有一个 PID 的 SYNC,而且两者都是以全速率发送的。集线器必须理解 PRE PID;而其它所有 USB 设备必须忽略它并把它作为未定义的标记来对待。在一个前同步信号 PID 结束之后,主机必须等待至少 4 个全速率比特周期,这期间集线器必须完成对其中继部分的配置处理以接受低速信号。在该集线器进行创建操作期间,集线器必须将它的全速率和低速率端口全部都驱动到各自的空闲状态。在集线器创建阶段结束之前,集线器必须为接受来自于主机的低速率信号做好准备。低速率连接原则总结如下:

- (1) 在连接过程中低速率设备被识别,并且与之相连的集线器端口也作为低速率装置来加以标识。
- (2) 所有的下行低速率分组必须以一个前同步信号为开始(以全速率发送),它可以打开低速率集线器端口上的输出缓冲区。
- (3) 当接收到一个 EOP 时,低速率集线器端口上的输出缓冲区就会被关闭,并且直到检测到了一个前同步信号 PID 后才能打开。
- (4) 上行连通性不会受一个集线器是全速率还是低速率的影响。

低速率信号以主机用低速率发送 SYNC 为开始,接下来是该分组的其余部分。分组末尾以分组结束(EOP)来标识,在这一时刻所有的集线器都会拆除连接并禁用任一个连接了低速设备的端口。集线器不会为上行信号而改变端口;对于低速率和全速率信号,在上行方向低速率端口都保持有效。

低速率和全速率处理操作保持了高度的协议公共性。但是,低速率信号确实有某些

限制,包括:

- (1) 数据负载限制为最大 8 个字节。
- (2) 低速率只支持中断和控制传输类型。
- (3) 低速率设备不能接收 SOF 分组。

## 8.7 差错检测和恢复

在物理信号发送层出现了差错的情况下,USB 仍然允许端到端的可靠通信。这包括了可以可靠地检测出绝大多数可能的错误的能力,以及根据处理操作类型从差错中恢复的能力。例如,控制处理操作需要高度的数据可靠性;利用差错检测和重试,它可以支持端到端的数据完整性。ISO 处理操作,由于其对带宽和时延的要求,不允许进行重试并且必须承受较高的未被纠正的差错发生的概率。

### 8.7.1 分组差错分类

USB 使用了三种差错检测机制:比特填充违规、PID 校验比特和 CRC。当在物理的 D+ 和 D- 信号线上检测时,如果在一个分组开始和结束之间,一个分组接收器检测到了在 7 个或更长的连续比特周期之内没有差动变化,就会出现一个比特填充违规错误。如果 4 个 PID 校验比特不是其各自的分组标识符比特的补码,就会出现一个 PID 错误。如果在接收完一个分组后所计算的校验和余数不为 0,就说明存在一个 CRC 错误。

除了 SOF 令牌外,任一个被接收的出错分组都会使接收器忽略它,并丢弃该分组所携带任何数据或其它域。表 8-6 列出了差错检测机制,所采用的分组类型和适当的分组接收器响应。

表 8-6 分组差错类型

域	错 误	动 作
PID	PID 校验, 比特填充	忽略分组
地址	比特填充, 地址 CRC	忽略令牌
帧标号	比特填充, 帧标号 CRC	忽略帧标号域
数据	比特填充, 数据 CRC	丢弃数据

### 8.7.2 总线转向时间

主机和 USB 功能设备需要跟踪从发送器发送完一个分组直到其接收到一个返回分组之间所用的时间。该时间指的就是总线转向时间并由分组发送器总线转向计时器来跟踪。该计时器在 EOP 选通脉冲内的 SE0 至 IDLE 变化处开始计数,当检测到一个 IDLE 至 K SOP 变化时停止记数。设备和集线器主机都需要转向计时器。设备总线转向时间定义为最坏情况下的往返时延再加上最大设备转向时延(参见 7.1.14 节)。在前一个 EOP 结束之后,USB 设备超时不能早于 16 个比特周期,并且必须处于超时状态 18 个比特周期。如果主机想通过一个超时来指示一个错误,那么在它发送下一个令牌之前必须等待至少 18 个比特周期,以保证所有的下行设备都处于超时状态。

如图 8-20 所示,设备在令牌和数据阶段或数据和握手阶段之间使用其转向计时器。而主机则在数据和握手阶段或令牌和数据阶段之间使用其计时器。

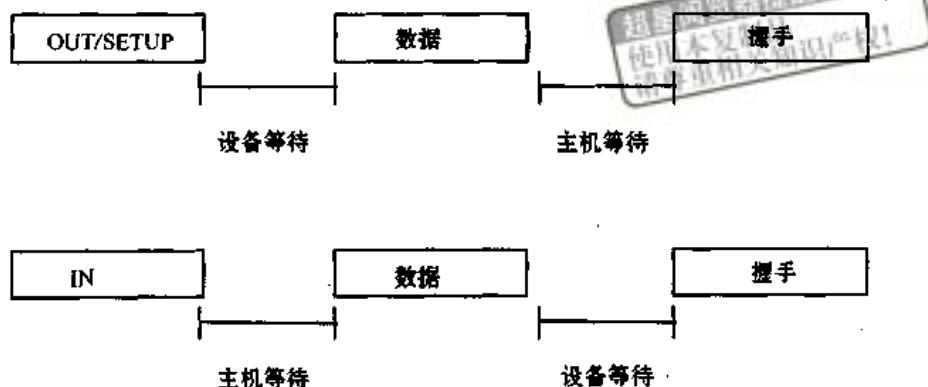


图 8-20 总线转向计时器的使用

如果主机收到了一个被破坏了的数据分组,它必须等待才能发出下一个令牌。这一等待间隔可以保证主机不会试图在一个假 EOP 之后立即发送一个令牌。

### 8.7.3 假 EOP

对于假 EOP 来说,我们必须采用某种方式来对其进行控制,这样才可以保证目前正在运行的分组可以在主机或其它任意一个设备试图传送一个新分组之前完成。但是如果真的出现了这一情况,它就会延续一个总线冲突并破坏最多两个连续的处理操作。对于假 EOP 的检测要依赖于这样一个事实,那就是一个假 EOP 所插入的分组会表现为一个具有 CRC 错误而且是截短的分组(该分组的最后 16 个比特为一个正确的 CRC 的概率很小)。利用这一特征就可以实现对假 EOP 分组的成功检测,从而实现对假 EOP 分组的有效控制,提高传输可靠性。

主机和设备控制假 EOP 情况所采用的方法不同。当一个设备检测到一个被破坏了的数据分组时,它不会对其作出任何响应,而是等待主机发送下一个令牌。这一方案可以确保当主机可能仍在传输一个数据分组时,该设备不会试图返回一个握手分组。如果出现了一个假 EOP,主机数据分组会最终结束,并且设备可以检测到下一个令牌。如果设备发送了一个数据分组,而该数据分组被破坏并且有一个假 EOP 时,那么主机就会忽略该分组并且不会向 USB 设备发送握手分组。而此时期望收到一个来自于主机的握手分组的 USB 设备就会超时。

如果设备收到了一个被破坏了的数据分组,它就会认为一个假 EOP 出现了,并会等待 16 个比特周期来看看是否还有任何后续的上行信息。如果在这 16 个比特间隔内没有检测到任何总线处理操作,并且总线仍停留在 IDLE(空闲)状态,主机可能会发送下一个令牌。否则,主机将等待设备完成发送其余下的分组。这里,等待 16 个比特周期可以保证两个条件。第一个条件是确认设备已经发送完了它的分组。这可以由一个大于在最坏情况下出现的 6 个比特周期的比特填充时间间隔的超时阶段来予以保证。第二个条件是发送设备的总线转向计时器必须可以确认其终止了。注意,超时时间间隔是对处理速度敏感的。主机对于不同的传输速率,它所等待的时间间隔略有不同。对于全速率处理操

作,主机必须等待 16 个全速率比特周期;而对于低速率处理操作,主机必须等待 16 个低速率比特周期。

如果主机收到了一个具有有效 CRC 的数据分组,它会认为分组已经结束并且无须推迟发送下一个令牌。而如果所接收到的数据分组的 CRC 被破坏或丢失,那么主机就会相应地推迟发送下一个令牌。

#### 8.7.4 串扰和活性损失恢复

USB 必须能够检测并从那些使其无限期地等待一个分组结束(EOP),或在一帧结束时使总线不是处于空闲状态的条件中恢复出来。活性损失(LOA)以 SOP 后没有任何总线操作(总线停留在 J 或 K 状态),并且在一帧结束时没有 EOP 为特征。串扰的标志是一个 SOP 之后,还会跟有一个在一帧结束后出现的总线操作。LOA 和串扰都有潜在的可能性会引起总线死锁或强行发送出下一帧的开始标志(SOF)。任何一种情况都是不能接受的,并且都必须阻止其发生。作为一个控制连接的 USB 元件,集线器负责对串扰/LOA 进行检测和恢复。所有在一帧结束之后还不能完成其传输的 USB 设备,都会被集线器阻止其在一帧之后继续传输。这一操作可以通过禁止连接这一出错设备的最近的集线器端口来实现。

# 第9章 USB设备结构



一个USB设备可以分成三层。最低一层中间一层则用于控制总线接口和设备上的各个端点是数据的最终使用者或提供者，可以认为是串行总线设备所提供的功能模块；例如，一个鼠

本章描述了一个USB设备的中间层的共部分将使用这些属性和操作，通过总线接口并最终

本章具体包括以下内容：

一个USB设备具有几个可



USB设备状态；



可见的设备状态；



USB总线枚举；



设置USB设备操作

而其它一些状态仅在 USB 设备内部使用。本节将描述这些状态。

### 9.1.1 可见的设备状态

本节描述了在内部可见的 USB 设备状态(参见图 9-1)。注意,USB 设备执行复位操作,以此来响应来自于主机向上行端口所发出的复位请求。当复位信号结束后,USB 设备就会复位。

#### 1. 连接

一个 USB 设备可以接入 USB,也可以从 USB 上拆除。当一个 USB 设备从 USB 上拆除后所处的状态,本规范并未定义。规范中只覆盖了当设备连接 USB 时所需的操作和属性。

#### 2. 上电

USB 设备可以从一个外部电源或通过与其连接的集线器而从 USB 获得电源供应。外部供电的 USB 设备称之为自供电设备。这些设备在接入 USB 之前可能已经上电了。一些设备可以同时支持自供电和总线供电配置。某些设备配置只支持一种电源。而其它一些设备配置只有当设备是外部供电时才可以进行。当前的电源会作为一个设备状态的一部分而加以报告。设备可以在任意时刻改变其电源。例如,由自供电改为 USB 供电。如果一个配置可以支持两种电源模式,那么该配置所报告的最大功率就是该设备在任一模式中所获得的最大值。不管其模式如何,该设备必须遵守这一最大值限制。如果一个配置仅支持一种供电模式而设备的电源改变了,该设备会释放当前的配置,寻址并返回到连接状态。

为了检测端口状态的变化,一个集线器端口必须处于通电状态,包括连接和拆除操作时间。直到对其进行了配置之后,集线器才能提供下行电源供应,这时它是根据其配置和电源所允许的条件来提供电源供给。从开始供电开始,一个 USB 设备必须能在一定的时间间隔被寻址(参见第 7 章)。在检测到一个端口的连接动作之后,主机应该激活该端口,而且会复位同该端口相连接的设备。

#### 3. 缺省

在设备上电之后,直到它从总线上接收了一个复位信号后,它才能对任一总线处理操作做出响应。在设备收到了一个复位信号后,设备可以用缺省地址来对其进行寻址。

#### 4. 地址分配

当设备刚上电或复位之后,所有的设备都使用缺省地址。主机在连接或复位操作之后,会为每一个 USB 设备分配一个唯一的地址。当一个 USB 设备被挂起时,它会保留为其分配的地址。

无论设备目前已被分配了一个唯一的地址,还是使用缺省地址,它都能在其缺省管道对请求做出响应。

#### 5. 配置完成

在 USB 设备可以使用之前,必须对该设备进行配置。在设备看来,配置就是向其配置寄存器内写入一个非零值。对一个设备的配置或对一个可替代的设置的改变,会使与受影响的接口内的端点有关的所有状态和配置值都设置为其缺省值。这包括利用数据触发器,将任一端点的数据触发值设置为 DATA0。

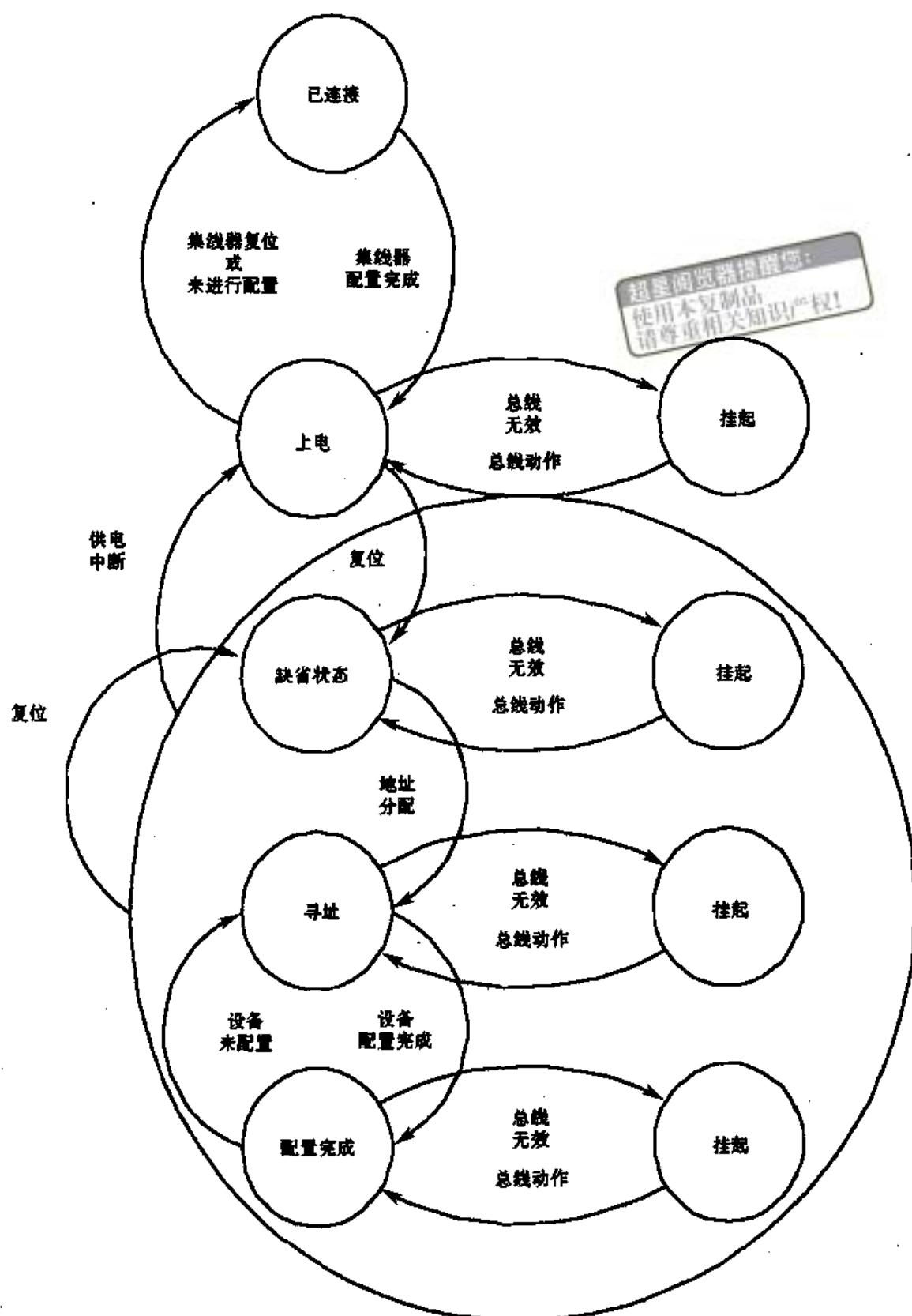


图 9-1 设备状态图

### 6. 挂起

为了节省电源,当设备在一个特定周期内(参见第7章)没有观测到任何总线操作,它就会自动进入挂起状态。当被挂起时,USB设备保留了包括其地址和配置信息在内的任何一个内部状态。

如果在第5章所说明的时间长度之内没有发现任何总线操作,所有的设备都必须挂起。已连接的设备必须在其上电后的任一时刻为挂起作好准备,不管它是否已被分配了一个非缺省地址或已经完成了配置。由于主机进入了其自己的挂起模式,因而总线操作也可以停止。而且,当一个USB所接入的集线器端口被禁止之后,它也会进入挂起状态。这就是选择性挂起。

当出现总线操作时,一个USB设备会退出挂起状态。利用电子信号来指示远端唤醒,一个USB设备也可以请求主机退出挂起模式或一个选择性挂起状态。一个设备可以发送远端唤醒信号的能力是可选的。如果一个USB设备可以发送远端唤醒信号,设备必须有支持由主机来激活和禁止该功能的能力。

表9-1就可能出现的状态组合进行了简要说明。

表9-1 可见的设备状态

连接	上电	缺省	寻址	配置完成	挂起	状态
否	—	—	—	—	—	设备未连接至USB。其他属性无意义
是	否	—	—	—	—	设备已接入USB,但是未上电。其它属性无意义
是	是	否	—	—	—	设备已接入USB,并上电,但是未被复位
是	是	是	否	—	—	设备已接入USB,并上电,而且已被复位,但是没有为其分配一个唯一的地址。设备对缺省地址作出响应
是	是	是	是	否	—	设备已接入USB,并上电,而且已被复位,也为其分配了一个唯一的地址。设备未被配置
是	是	是	是	是	否	设备已接入USB,并上电,而且已被复位,也为其分配了一个唯一的地址和进行了配置,同时未被挂起。此时,主机可以利用该设备提供的功能
是	是	是	是	是	是	设备至少已经接入USB,而且已被复位,并处在最低的挂起电平上。它也可能拥有一个唯一的地址和完成了配置。但是,由于该设备被挂起,主机不能利用设备提供的功能

#### 9.1.2 总线枚举

当一个USB设备接入USB或从USB上拆除时,主机使用一个称之为总线枚举的过程来识别和管理必要的设备状态变化。当一个USB设备接入后,将采取下列动作:

- (1) 现在该USB设备所接入的集线器,通过一个其状态变化管道上的回应,可以向主机报告该事件。这时,USB设备处于连接状态,而连接它的端口则被禁用。
- (2) 主机通过询问集线器来确定变化的真实性质。
- (3) 既然主机已经知道了新的设备所接入的端口,它就会向该端口发送一个端口激

活和复位命令。

(4) 集线器将保留发往该端口的复位信号 10ms。当复位信号释放后,被激活的端口和集线器将向 USB 设备提供 100mA 总线电流。现在 USB 设备就处于上电状态。它的所有寄存器和状态都被重新设置了,而且它可以对缺省地址做出响应。

(5) 在为该 USB 设备分配了一个唯一的地址之前,利用缺省地址仍然可以访问其缺省管道。主机通过读取该设备的描述符,来确定这一 USB 设备的缺省管道实际可以使用的最大数据负载尺寸。

(6) 主机向 USB 设备分配一个唯一的地址,使设备进入寻址状态。

(7) 主机通过从 0 至 n 读取每一个配置,它可以从设备中读出配置信息。这一过程可能需要几帧来完成。

(8) 基于配置信息和如何来使用这一 USB 设备,主机可以向设备分配一个配置值。这时设备就处于配置完成状态,并且在这一配置中的所有端点都具有其描述的特征。现在 USB 设备就可以获得其配置描述符中所描述的  $V_{bus}$  电流量。从设备的观点来看,它已经为使用做好了准备。

当 USB 设备被拆除后,集线器也会向主机发送一个指示。拆除一个设备会使该设备所接入的端口被禁用。一旦收到了拆除指示,主要会更新它的本地拓扑结构信息。

## 9.2 通用 USB 设备操作

所有的 USB 设备都支持一个共同的操作集,本节就描述了这些操作。

### 9.2.1 动态连接和拆除

可以在任意时刻连接或拆除 USB 设备。由提供接入点或端口的集线器负责报告端口状态的任何变化。

一旦检测到了一个连接操作,主机会激活连接了这一设备的集线器端口,而且会使设备复位。一个复位后的 USB 设备具有下列特征:

- (1) 对缺省 USB 地址作出响应。
- (2) 未被初始化。
- (3) 不会首先被挂起。

当一个设备从一个集线器端口上拆除时,主机会被告知这一拆除操作。主机通过禁用设备所连接的集线器端口做出响应。

### 9.2.2 地址分配

当一个 USB 设备连接后,主机会在设备被主机复位并且设备所连接的集线器端口被激活之后,负责向该设备分配一个唯一的地址。

### 9.2.3 配置

一个 USB 设备必须在完成配置之后其功能才能被使用。主机负责对一个 USB 设备

进行配置。主机总是从 USB 设备获得配置信息,从而确定设备的功能。

作为配置过程的一部分,主机将启动设备配置操作,并且如果必要的话,还会为需要这一限制的端点设置一个最大分组尺寸。

在单个配置操作内,一个设备可以支持多个接口。一个接口就是一个相应的端点集,它代表了该设备向主机提供的一个单一的特性或功能。用于同这一相应的端点集通信的协议和接口中的每一个端点的用途,都可以作为一个设备类型或某个厂商类型定义的一部分来加以说明。

而且,位于一个配置内的一个接口可以有可供替换的设置,可以对序号或相应的端点的特征进行重新定义。如果是这种情况,设备就会支持获得(Get)接口和设置(Set)接口请求,并为某个接口报告或选择某个可选的设置。

在每一个配置内,每个接口描述符都包含了可以对该接口号和可替换的设置进行识别的域。接口号可以从 0 开始,直到比当前该配置所支持的接口数少一。可替换的设置的计数范围是,从 0 到比某个接口的可替换设置数少一。当一个设备最初被配置时,其缺省设置是可替换设置 0。

为了支持可以管理一个相应的 USB 设备组的自适应设备驱动程序,设备和接口描述符包括了类型,子类和协议域。这些域用来标识一个 USB 设备所提供的功能和同设备上的该功能模块通信所使用的协议。为一个相应的设备类型分配的类型代码已经作为 USB 规范的一部分而被定义。一个设备类型可以进一步化分为子类,而且在一个类型或子类中,一个协议代码可以规定软件是如何同设备通信的。

#### 9.2.4 数据传输

数据可以以四种方式在一个 USB 设备端点和主机之间传输。请参照第 5 章中有关四种传输类型的定义,某些端点可能适合不同类型的数据传输。但是,一旦配置完成,一个 USB 设备端点只能使用一种数据传输模式。

#### 9.2.5 功率管理

在 USB 设备上进行功率管理,包括下列几节所描述的一些问题。

##### 1. 功率预算

... 从电源管理的角度看,USB 设备的功耗主要由两个方面决定:一是由 USB 总线供电的功耗,二是由设备内部电源管理的功耗。

部事件来触发挂起的 USB 设备,使其向主机发送信号。一个 USB 设备会在一个配置描述符内报告其支持远端唤醒功能。如果一个设备支持远端唤醒,它也必须允许可以利用标准的 USB 请求来激活和禁用该项能力。

远端唤醒使用了本文中其它地方所说明的电气信号才得以实现。

### 9.3 USB 设备请求

所有的 USB 设备都会对设备缺省管道上来自于主机的请求做出响应,这些请求是利用控制传输而产生的。请求和请求参数则是在建立分组中传送到设备的。主机将负责建立要在下列域中传送的数值。每一个建立分组都有 8 个字节,如表 9-2 所列。

表 9-2 USB 设备请求

偏移量	域	尺寸	数值	说 明
0	bmRequestType	1	位图	请求特性: D7 数据传输方向 0 = 主机至设备 1 = 设备至主机 D6..5 类型 0 = 标准 1 = 类型 2 = 供应商 3 = 保留 D4..0 接受器 0 = 设备 1 = 接口 2 = 端点 3 = 其他 4..31 = 保留
1	bRequest	1	数值	特殊请求(参见表 9-3)
2	w Value	2	数值	根据不同的请求,以字节为单位来定义的域会有所差别
4	wIndex	2	偏移量索引	根据不同的请求,以字节为单位来定义的域会有所差别 ——最典型的是用来传递索引或偏移量
6	wLength	2	记数	如果存在一个数据阶段,指出要传输的字节数

#### 1. bmRequestType

比特映射域说明了某个请求的特征。特别地,该域指明了在控制传输的第二阶段中数据传输的方向。如果 WLength 域为 0,就意味着没有数据阶段,方向比特的状态就会被忽略。

USB 规范定义了一系列所有的设备都必须支持的标准请求。而且,一种设备类型可以定义额外的请求。一个设备供应商也可以定义该设备所支持的请求。

请求可以指向设备、设备上的一个接口,或设备上的某个端点。这个域也说明该请求的接收者。当一个接口或端点被说明后,wIndex 域就会标识出接口或端点。

#### 2. bRequest

这个域说明了一个特定的请求。bmRequestType 域中的类型比特可以修改这个域的含义。本规范仅定义了当比特被重置时 bRequest 域的值,以指示一个标准的请求(参见表 9-3)。

### 3. w Value

这个域中的内容根据请求而变化。它用于向该请求指定的设备传递一个参数。

### 4. wLength

这个域说明了在控制传输中的第二个阶段所传输的数据长度。数据传输的方向(主机至设备或设备至主机)由 bRequest 域中的方向比特指定。如果该域为 0,说明没有数据传输阶段。

## 9.4 标准设备请求

本节描述了为所有 USB 设备定义的标准设备请求(参见表 9-3)。表 9-4 则给出了标准设备请求对应的代码。

表 9-3 标准设备请求

bmRequestType	bRequest	w Value	wIndex	wLength	Data
0000000B 0000001B 00000010B	CLEAR_FEATURE	特性选择符	接口端点 0	零	无
1000000B	GET_CONFIGURATION	零	零	1	配置数值
1000000B	GET_DESCRIPTOR	描述符类型和描述符索引	零或语言 ID	描述符长度	描述符
10000001B	GET_INTERFACE	零	接口	1	可替换的接口
1000000B 10000001B 10000010B	GET_STATUS	零	接口端点 0	2	设备, 接口, 或端点状态
0000000B	SET_ADDRESS	设备地址	零	零	无
0000000B	SET_CONFIGURATION	配置值	零	零	无
0000000B	SET_DESCRIPTOR	描述符类型和描述符索引	零或语言 ID	描述符长度	描述符
0000000B 00000001B 00000010B	SET_FEATURE	特性选择符	接口端点 0	零	无
00000001B	SET_INTERFACE	交替设置	接口	零	无
10000010B	SYNCH_FRAME	零	端点	2	帧标号

USB 设备必须对标准设备请求做出响应,不管该设备是否已经被分配了一个非缺省地址或该设备目前正在被配置。

表 9-4 中, wValue 所代表的描述符类型共用 5 种,如表 9-5 所列。在 9.5 节和 9.6

节中,我们将详细讨论描述等。

表 9-4 标准设备请求代码

bRequest	数 值
GET_STATUS	0
CLEAR_FEATURE	1
留作以后使用	2
SET_FEATURE	3
留作以后使用	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

表 9-5 描述符类型

描述符类型	数 值
设备	1
配置	2
字符串	3
接口	4
端点	5

特性选择符在为某个特定的设备,接口或端点而激活或设置其特性(例如远端唤醒)时使用。特性选择符的值如表 9-6 所列。

表 9-6 特性选择符

特性选择符	接收者	数 值
DEVICE_REMOTE_WAKEUP	设备	1
ENDPOINT_STALL	端点	0

如果把一个不支持的或不正确的请求送到了一个 USB 设备,那么该设备就会在这一请求所使用的管道上指示出一个禁止条件,以此做为响应。控制管道,包括缺省管道,即使它们被禁用了,也必须接受一个建立处理操作。ClearStall 用来清除一个被禁用的管道。在主机清除了禁止条件之后,系统软件可以继续对控制管道的正常访问。如果由于任何原因,设备因为一个差错情况而不能通过其缺省管道进行通信,设备必须被复位,以清除这一情况并重新启用缺省管道。

### 1. 清除特性(Clear Feature)

本请求用于清除或禁用一个特定的特性。对应于该请求的建立分组(参见 9.3 节表 9-2)中各个域所表示的值如表 9-7 所列。

表 9-7 清除特性请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
00000000B	CLEAR_FEATURE	特性选择符	接口端点 0	0	无
00000001B					
00000010B					

w Value 中的特性选择符的数值必须同接收者相适应。当接收者是一个设备时,只有设备特性选择符数值可以使用;当接收者是一个接口时,只有接口特性选择符数值可以使用;当接收者是一个端点时,只有端点特性选择符数值可以使用。

对于为每个接收者所定义的特性选择符数值请参考本节中的有关定义。

一个 ClearFeature 请求要指向的特性如果不能被消除或根本就不存在,会引起一个停止条件。

## 2. 获得配置信息(Get Configuration)

该请求返回当前的设备配置信息。

对应于该请求的建立分组(参见 9.3 节表 9-2)中各个域所表示的值如表 9-8 所列。

表 9-8 获得配置信息请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
10000000B	GET_CONFIGURATION	0	0	0	配置值

如果返回值为零,说明设备未被配置。

## 3. 获取描述符(Get Descriptor)

本请求返回某个描述符,如果该描述符存在的话。

对应于该请求的建立分组(参见 9.3 节表 9-2)中各个域所表示的值如表 9-9 所列。

表 9-9 获取描述符请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
10000000B	GET_DESCRIPTOR	描述符类型和 描述符索引 (参见 9.6.5)	0 或语言 ID	描述符长度	描述符

w Value 域在高字节中说明了描述符类型,而在低字节中说明了描述符索引(参见表 9-5)。w Index 域可以为字符串描述符指出语言 ID,或者为其它描述符而重置为 0。w Length 域说明了要返回的字节数。如描述符长于 w Length 域,只可能返回描述符中最初的 n 个字节。如果该描述符比 w Length 域短,设备可以通过在请求更多的数据时发送一个短分组,以此来指出控制传输的结束。一个短分组定义为一个比最大负载尺寸短的分组或一个 NULL 数据分组(参见第 5 章)。

指向一个设备的标准请求支持三种类型的描述符:DEVICE、CONFIGURATION 和 STRING。对一个配制描述符的请求可以返回配置描述符,所有的接口描述符和单个请求中的所有接口的端点描述符。第一个接口描述符紧跟在配置描述符之后。用于第一个接口的端点描述符跟在第一个接口描述符之后。如果有另外的接口,它的接口描述符和端点描述符跟在第一个接口端点描述符之后。

所有的设备必须提供一个设备描述符和至少一个配置描述符。如果一个设备不支持所请求的描述符,它会通过停止该请求所使用的管道来响应。一个描述符的第一个字节为非零值说明缓冲区包含了一个有效的描述符。

#### 4. 获得接口配置(Get Interface)

本请求返回为某个接口所选定的可替换设置。对于该请求的建立分组中各个域所表示的值如表 9-10 所列。

表 9-10 获得接口配置请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
10000001B	GET_INTERFACE	0	接口	1	可供替换的设置

某些设备的配置中接口具有相互排斥的设置。本请求允许主机确定当前选定的可替换的设置。

#### 5. 获取状态(Get Status)

本请求为某个接收者返回一个状态。

表 9-11 获取状态请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
10000000B					
10000001B	GET_STATUS	0	接口端点 0	2	设备, 接口或端点状态
10000010B					

bRequestType 域中的接收者比特说明了所需的接收者。返回的数据是某个接收者的当前状态。

对一个设备的 GetStatus 请求以 Little-endian 的顺序(首先发送 LSB 比特)返回下列信息,如表 9-12 所列。

表 9-12 获取状态请求返回的信息(设备)

D7	D6	D5	D4	D3	D2	D1	D0
保留(复位为 0)						远程唤醒	自供电
D15	D14	D13	D12	D11	D10	D9	D8
保留(复位为 0)							

SelfPowered 域指出目前设备是总线供电还是自供电。如果 D0 被重置为 0,设备就是总线供电。如果 D0 设为 1,该设备就是自供电。SelfPowered 域不可能被 SetFeature 或者 ClearFeature 请求所改变。

Remote Wakeup 域指出目前设备是否可以请求远端唤醒。对于支持远端唤醒功能的设备,其缺省模式是禁用。如果 D1 复位为 0,那么设备发送远端唤醒信号的能力就被禁用。如果 D1 设置为 1,那么设备发送远端唤醒信号的能力就被激活。利用 DEVICE\_REMOTE\_WAKEUP 特性选择符,Remote Wakeup 域可以由 SetFeature 和 ClearFeature 请求来改变。当设备复位时,该域会重置为 0。

向一个接口发出的 GetStatus 请求以 little-endian 的顺序返回下列信息,如表 9-13

所列。

表 9-13 获取状态请求返回的信息(接口)

D7	D6	D5	D4	D3	D2	D1	D0
保留(复位为0)							
D15	D14	D13	D12	D11	D10	D9	D8
保留(复位为0)							

如果该请求指向一个端点,那么端点必须在 wIndex 域内对其进行说明。wIndex 的高位字节复位为 0,而低位字节则包括了端点号,如表 9-14 所列。

表 9-14 wIndex 域的值

D7	D6	D5	D4	D3	D2	D1	D0
方向	保留(复位为0)			端点号			

对于 IN 端点,D7 设置为 1。对于 OUT 端点,D7 复位为 0。

可指向一个端点的一个 GetStatus 请求将返回下列信息,如表 9-15 所列。

表 9-15 获取状态请求返回的信息(端点)

D7	D6	D5	D4	D3	D2	D1	D0
保留(复位为0)							禁止
D15	D14	D13	D12	D11	D10	D9	D8
保留(复位为0)							

如果该端点当前被禁用,Stall 域会置为 1。否则,Stall 域复位为 0。利用具有 ENDPOINT\_STALL 特性选择符的 SetFeature 和 ClearFeature 请求,可以改变 Stall 域。当由 SetFeature 请求设置时,该端点会表现出同如果是一个硬件条件来设置该域时相同的停止操作。如果引起一个停止的条件被清除,对 Stall 域的清除操作会使端点不再返回一个停止状态。对于一个使用数据触发器的端点而言,清除一个被禁用的端点的操作会使数据触发器被重新初始化为 DATA0。在 SetConfiguration 或 SetInterface 请求之后,该域会复位为 0。

#### 6. 设置地址(Set Address)

本请求为以后的所有设备访问设置设备地址。

对应于该请求的建立分组中各个域所代表的值,如表 9-16 所列。

表 9-16 设置地址请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
0000000B	SET_ADDRESS	设备地址	0	0	无

wValue 域说明了用于所有的后续访问的设备地址。

正如在别处所提到的一样,请求实际上可以最多产生三个操作阶段。在第一个阶段,

建立分组被发送到设备。在可选的第二个阶段，数据在主机和设备之间传输。在最后一个阶段，在主机和设备之间传输状态。数据和状态传输的方向依赖于是主机向设备发送数据还是设备向主机发送数据。状态阶段传输总是具有和数据阶段相反的方向。如果没有数据阶段，状态阶段传输方向就是由设备到主机。

在最初的建立分组之后的阶段将同样一个设备地址看作是创建分组。直到这一请求的状态阶段成功地完成之后，USB 设备才能改变其地址。注意本请求和其它所有请求有所不同。对于其它所有请求，所指出的操作必须在状态阶段之前完成。

#### 7. 设置配置操作 (Set Configuration)

本请求用于设置设备配置操作。

对应于该请求的建立分组中各个域所代表的值如表 9-17 所列。

表 9-17 设置配置操作请求

bmRequestType	bRequest	w Value	wIndex	wLength	数据
00000000B	SET_CONFIGURATION	配置值	0	0	无

w Value 域说明了所需的配置操作。该值必须为 0 或同来自于一个配置描述符的配置数值相匹配。如果该值为 0，设备就处于未经配置的状态。

#### 8. 设置描述符 (Set Descriptor)

本请求是可选的。如果一个设备支持这一请求，已经存在的描述符可能会被更新或可能会加入新的描述符。

对应于该请求的建立分组中各个域所代表的值如表 9-18 所列。

表 9-18 设置描述符请求

bmRequestType	bRequest	w Value	wIndex	wLength	数据
00000000B	SET_DESCRIPTOR	描述符类型 和描述符索引	语言 ID (参见 9.6.5)	描述符长度	描述符

w Value 域在高位字节及低位字节分别说明了描述符类型和描述符索引。wIndex 域为字符串描述符说明了语言 ID，或者为其它的描述符而复位为 0，wLength 域说明了由主机向设备传输的字节数。

#### 9. 设置特性 (Set Feature)

本请求用于设置或激活某个特定的特性。

对应于该请求的建立分组中各个域所代表的值如表 9-19 所列。

表 9-19 设置特性请求

bmRequestType	bRequest	w Value	wIndex	wLength	数 据
00000000B					
00000001B	SET_FEATURE	特性选择器	接口端点 0	0	无
00000010B					

w Value 中的特性选择符的数值必须同接收者相适应。当接收者是一个设备时，只有设备特性选择符数值可以使用；当接收者是一个接口时，只有接口特性选择符数值可以使用；当接收者是一个端点时，只有端点特性选择符数值可以使用。

有关为每个接收者所定义的特性选择符的数值请参考 9.4 节。

一个 ClearFeature 请求要指向的特性如果不能被设置或根本就不存在,会引起一个停止条件。

#### 10. 设置接口(Set Interface)

本请求允许主机为某个接口选择一个可替换的设置。

对应于该请求的建立分组中各个域所代表的值如表 9-20 所列。

表 9-20 设置接口请求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE	可替换的设置	接口	0	无

某些设备的配置中接口具有相互排斥的设置。本请求允许主机选择所需的可替换的设置。

#### 11. 同步帧(Synch Frame)

本请求用于设置和报告一个端点的同步帧。

对应于该请求的建立分组中各个域所代表的值如表 9-21 所列。

表 9-21 同步帧请求

bmRequestType	bRequest	wValue	wIndex	wLength	数据
00000010B	SYNCH_FRAME	0	端点	2	帧序号

当一个端点支持同步传输时,该端点也可以依据某一方式,要求每一帧传输都可以改变大小。主机和端点必须就重复模式在哪一帧开始达成一致。本请求使端点开始监测 SOF 帧数,以跟踪处于其模式中的一个给定的帧的位置。该模式开始时的帧的序号将返回给主机。这一帧序号是在该模式的第一个帧之前,由最后一个 SOF 传送到端点的那一个帧的序号。

该数值仅用于使用隐含的同步模式的同步数据传输。如果该端点不是同步的或未使用此方法,那么该端点就不支持这一请求,并且会返回一个停止条件。

由一个设备复位,或者通过一个 SetConfiguration 或一个 SetInterface 请求对端点进行配置,可以使起始帧复位为 0。

## 9.5 描述符

USB 设备利用描述符来报告其属性。一个描述符是具有一个确定格式的一个数据结构。每个描述符都以一个一字节宽并包含了该描述符中的所有字节数的域为开始,再跟上一个说明描述符类型的一字节宽的域。

使用描述符可以对单独的配置的属性进行简洁地存储,这是因为每个配置都可以重用来自那些具有相同特征的其它配置的描述符或部分描述符。在这种方式下,描述符与一个关系数据库中的单独的数据记录相类似。

在适当的地方,描述符包括了字符串描述符引用,它提供了可显示的信息,以一种人们可以读懂的形式来描述一个描述符。是否包括字符串描述符是可选的。但是,描述符的引用域是必备的。如果一个设备不支持字符串描述符,字符串引用域必须复位为 0,以

指出没有字符串描述符可用。

如果一个描述符长度域内的返回值小于本规范定义的值,该描述符就是不正确的,并且应该由主机来丢弃。如果一个描述符长度域内的返回值大于本规范定义的值,主机会忽略多余的字节,但是下一个描述符却是利用返回的长度而不是所期望的长度来进行定位的。

类型和供应商专用描述符可以用两种方式之一返回。与标准描述符有关的类型和供应商专用描述符,是在一个同标准的描述符相同的数据缓冲区之内,紧跟着相应的标准描述符而返回的。

例如,如果某一个类型或供应商描述符同一个接口描述有关,有关的类型或供应商专用描述符会放在缓冲区内的接口描述符和接口端点描述符之间,作为对一个 GET\_CONFIGURATION\_DESCRIPTOR 请求的响应而返回。一个标准描述符的长度不会增加用来容纳设备类型或某个供应商描述符的扩展。类型或供应商专用描述符遵循同标准描述符一样的格式,像某个描述符的前两个字节一样其前两个字节也分别用作长度和类型域。

与一个标准描述符无关的类型或供应商专用描述符,可以利用类型供应商特定的请求而返回。

## 9.6 标准 USB 描述符定义

### 9.6.1 设备

设备描述符如表 9-22 所列。

表 9-22 设备描述符

偏移量	域	尺寸	数值	说 明
0	bLength	1	数字	按字节记,描述符的大小
1	bDescriptorType	1	常数	设备描述符类型
2	bcdUSB	2	BCD	USB 规范发布号用二进制编码小数表示(即 2.10 代表 0x210)。该域指出设备及其描述符所遵照的 USB 发布规范
4	bDeviceClass	1	类型	类型代码(由 USB 指定): 如果该域复位为 0,在一个配置中的每一个接口都要说明它的类型信息,并且不同的接口相互独立工作; 如果该域设置为 1 和 0xFF 之内的某个数值,该设备就可以在不同的接口上支持不同的类型规范,而且接口可能不会相互独立地进行工作。该数值用来识别用于全部接口的类型定义(例如,一个具有音频和数字数据接口的 CD-ROM 设备要求传输控制信息来推出 CD 或使 CD 旋转)
5	bDeviceSubClass	1	子类型	子类型代码(由 USB 分配); 这些代码由 bDeviceClass 域内的数值来限定; bDeviceClass 域复位为 0,该域也必须复位为 0; 如果 bDeviceClass 域未被置为 0xFF,所有的数值都被保留用于 USB 分配

(续)

偏移量	域	尺寸	数值	说 明
6	bDeviceProtocol	1	协议	协议代码(由 USB 分配): 这些代码由 bDeviceClass 和 bDeviceSubClass 域内的数值进行限定。如果一个设备支持基于设备的某种类型而不是基于一个接口的某种类型,这一代码可以标示出该设备所使用的由设备类型规范所定义的协议。 如果该域复位为 0,那么该设备就不是使用基于一个设备的某一类型的协议。 如果该域被置为 0xFF,该设备所使用的就是基于一个设备的由供应商特定的协议
7	bMaxPacketSize0	1	数字	用于端点 0 的最大分组尺寸(只有 8, 16, 32, 或 64 有效)
8	idVendor	2	ID	供应商 ID(由 USB 分配)
10	idProduct	2	ID	产品 ID(由厂家分配)
12	bcdDevice	2	BCD	二进制编码小数形式的设备发行号
14	iManufacturer	1	索引	用于描述厂商的字符串描述符的索引
15	iProduct	1	索引	用于描述产品的字符串描述符的索引
16	iSerialNumber	1	索引	用于描述设备序列号的字符串描述符的索引
17	bNumConfigurations	1	数字	可能的配置数

一个设备描述符描述了有关一个 USB 设备的通用信息。它包括了可以同时应用于设备和所有的设备配置的信息。一个 USB 设备只有一个设备描述符。设备描述符的结构如表 9-22 所示。

所有的 USB 设备都具有一个缺省管道所使用的端点 0。一个设备的端点 0 所允许的最大分组尺寸将在设备描述符中说明。对应于一个配置及其接口的端点将在配置描述符内说明。一个配置及其接口不包括一个用于端点 0 的端点描述符。除了最大分组尺寸之外,端点 0 的特征都由本规范来规定,并且对于所有的 USB 设备都相同。

BNumConfigurations 域标出了该设备所支持的配置数。

### 9.6.2 配置

配置描述符描述了有关某个设备配置的信息。这个描述符包括了一个具有一定数值的 bConfigureValue 域,当其作为一个提供给 Set Configuration 请求的参数使用时,它可以使设备采用所描述的配置方法。其结构如表 9-23 所列。

表 9-23 配置描述符

偏移量	域	尺寸	数值	说 明
0	bLength	1	数字	按字节记,该描述符的大小
1	bDescriptorType	1	常数	配置
2	wTotalLength	2	数字	为该配置所返回的数据的整个长度。其中包括为该配置所返回的所有描述符(配置,接口,端点和类型或具体的供应商)的联合长度
4	bNumInterfaces	1	数字	该配置所支持的接口数

(续)

偏移量	域	尺寸	数值	说 明
5	bConfigurationValue	1	数字	作为一个用于设置配置的自变量而使用的数值,以选择这一配置
6	iConfiguration	1	索引	用于描述该配置的字符串描述符的索引
7	bmAttributes	1	位图	<p>配置特性:</p> <p>D7 总线供电  D6 自供电  D5 远程唤醒  D4..0 保留(复位为0)</p> <p>一个既从总线又从本地电源获得电源供应的设备配置,要对 D7 和 D6 都进行置位。而在运行时所使用的实际电源,则可以通过 Get Status(获得状态)请求来决定  如果该设备配置支持远程唤醒功能,D5 应该设置为1</p>
8	MaxPower	1	mA	<p>当设备是完全可操纵时,在这一具体的配置内 USB 设备从总线上所消耗的最大功率。用 2mA 为单位来表示(即,50 = 100mA)</p> <p>注意:一个设备配置要报告该配置是总线供电,还是自供电。而设备状态则报告设备当前是否是自供电。如果一个设备从它的外部电源上拆除下来,它就会更新其状态来指出它已经不再是自供电了</p> <p>一个设备在它不采用外部电源对其供电之后,不能增加其从总线上获得的电源功率,从而超过在其配置中所报告的数量</p> <p>如果一个设备在不采用外部电源对其供电之后仍能继续工作,它就会继续进行工作。如果设备不能继续进行工作,它就会让那些它不能再予以支持的操作失败。主机软件可以通过检查状态和注意到缺乏设备电源来确定操作之所以失败的原因。</p>

该描述符描述了配置所能支持的接口数。每一个接口都可以独立地工作。例如,一个 ISDN 设备可以配置成具有两个接口,每个接口都提供 64KB/s 双向信道,该信道在主机上拥有分离的数据源或接收器。另外一种配置可能把这个 ISDN 设备配置为单个接口,将两个信道合为一个 128KB/s 的双向信道。

当主机请求配置描述符时,所有相关的接口和端点描述符都会被返回(参见 9.4.2 节)。

一个 USB 设备具有一个或多个配置描述符。每一个配置都有一个或多个接口,并且每个接口都有一个以上的端点。

在单个配置中,一个端点不会在接口之间共享,除非该端点被同一个接口的可替换的设置所使用。端点可以在那些没有这一限制的具有不同配置的一部分的接口之间共享。

一旦完成了配置,设备还可以支持对配置进行有限的调整。如果某个接口有可替换

的设置，在配置之后就会选择这个可替换的设置。在一个接口中，一个同步端点所允许的最大分组尺寸也可以被调整。

### 9.6.3 接口

该描述符描述了由相应的配置所提供的某个接口。一个配置提供了一个以上的接口，每一个接口都有自己的端点描述符来描述该配置内的一个唯一的端点集。当一个配置支持的接口多于一个时，在 Get \_ Configuration 请求所返回的数据中，某个接口的端点描述符紧跟在接口描述符之后。一个接口描述符总是作为一个配置描述符的一部分而返回的。它不能用一个 Get \_ Descriptor 或 Set Descriptor 请求来直接访问。其结构如表 9 - 24 所列。

表 9 - 24 接口描述符

偏移量	域	尺寸	数值	说 明
0	bLength	1	数字	按字节记，该描述符的大小
1	bDescriptorType	1	常数	接口描述符类型
2	bInterfaceNumber	1	数字	接口数目。非 0 值指出在该配置所同时支持的接口阵列中的索引
3	bAlternateSetting	1	数字	用于为上一个域所标识的接口选择可供替换的设置
4	bNumEndpoints	1	数字	该接口所支持的端点数(不包括端点 0)。如果该值为 0，那么该端点就只能利用端点 0
5	bInterfaceClass	1	类型	类型代码(由 USB 分配): 如果该位复位为 0，那么该接口就不属于任何一种 USB 所规定的设备类型。 如果该位设置为 0xFF，接口类型就是由供应商所特定的。 其他所有的数值都保留而由 USB 进行分配
6	bInterfaceSubClass	1	子类型	子类型代码(由 USB 分配): 这些代码由 bInterfaceClass 域内的数值来限定。 如果 bInterfaceClass 域复位为 0，那么该域也必须复位为 0。 如果 bInterfaceClass 域未被设置为 0 0xFF，所有的数值都保留而由 USB 进行分配
7	bInterfaceProtocol	1	协议	协议代码(由 USB 分配): 这些代码由 bInterfaceClass 域和 bInterfaceSubClass 域内的数值来加以限定。 如果一个接口支持特定类型的请求，该代码标识出该设备所使用的由设备类型规范所定义的协议类型。 如果该域复位为 0，那么在该接口上该设备就没有使用一个特定类型的协议。 如果该域被置为 0xFF，那么在该接口上该设备就使用了一个由供应商指所定的协议
8	iInterface	1	索引	用于描述接口的字符串描述符的索引

一个接口可以包括可交替的设置，这样可以允许端点和/或其特性在设备被配置之后

发生改变。对于一个接口来说,缺省设置总是可交替的设置 0。SetInterface 用于选择一个可交替的设置或返回到缺省设置。Get Interface 请求返回所选择的可交替的设置。

交替的设置允许设备配置的一部分发生变化,而同时其它接口可以继续工作。如果一个配置具有交替的设置可用于一个或多个接口,那么每个设置都包括了一个分离的描述符和其相应的端点。

如果一个设备配置用两个交替的设置来支持单个接口,那么配置描述符后会跟上一个 bInterfaceNumber 和 bAlternateSetting 域复位为 0 的接口描述符,然后是用于该设置的端点描述符,再跟上另外一个接口描述符及其相应的端点描述符。第二个接口描述符内的 bInterfaceNumber 域也应该置为 0,但是第二个接口描述符内的 bAlternateSetting 域应该置为 1。

如果一个接口只使用端点 0,那么接口描述符后就不会跟有端点描述符,并且该接口将标识一个请求接口,它使用同端点 0 相连的缺省管道。在这种情况下,bNumEndpoint 也会被置为 0。一个接口描述符决不会在端点号内包含端点 0。

#### 9.6.4 端点

用于一个接口的每一个端点都有自己的描述符。该描述符包括了主机确定每一个端点的带宽请求所要求的信息。一个端点描述符总是作为一个配置描述符内的一部分而返回。不能利用 Get \_ Descriptor 或 Set Descriptor 请求来直接对其访问。对于端点 0 来说,它永远不会具有一个端点描述符。其结构如表 9-25 所列。

表 9-25 端点描述符

偏移量	域	尺寸	数值	说明
0	bLength	1	数字	按字节记,该描述符的大小
1	bDescriptorType	1	常数	端点描述符类型
2	bEndpointAddress	1	端点	USB 设备上的端点地址由该描述符来描述。 该地址的编码如下所述: Bit 0..3: 端点号 Bit 4..6: 保留,应复位为 0 Bit 7: 方向,对于控制端点 应忽略 0 输出端点 1 输入端点
3	bmAttributes	1	位图	当该域被配置成可以使用 bConfiguration Value 时,它可以描述该端点的 属性 Bit 0..1: 传输类型 00 控制 01 同步 10 批量 11 中断 其它所有比特均保留

(续)

偏移量	域	尺寸	数值	说 明
4	wMaxPacketSize	2	数字	当选择这一配置时,该端点所能发送和接收的最大分组尺寸 对于同步端点而言,该数值用来保留每一帧的数据负载所要求的预算中的总线时间。对于一个正在运行的情况而言,该管道实际所使用的带宽可能小于所保留的数值。如果必要的话,该设备可以利用标准的,非USB定义的机制来报告它实际使用的带宽 对于中断,批量和控制端点而言,可能会发送更小的数据负载,但是这样会终止这一传输,并且可能需要或不需要主机干预来重新启动。 更详细的内容请参考第5章的介绍
5	bInterval	1	数字	用于轮询,用来进行数据传输的端点所需的时间间隔。这里该时间间隔用毫秒(ms)来表示 对于批量和控制传输端点而言,应当忽略该域。而对于同步端点而言,该域必须置为1。 对于中断端点,该域的取值范围是1至255

### 9.6.5 字符串

字符串描述符是可选的。正如前面提到的那样,如果一个设备不支持字符串描述符。在设备、配置和接口描述符内的所有字符串描述符引用都必须复位为0。其结构如表9-26所列。

表9-26 字符串描述符

偏移量	域	尺寸	数值	说 明
0	bLength	1	数字	按字节记,该描述符的大小
1	bDescriptorType	1	Constant	字符串描述符类型
2	bString	N	数字	UNICODE编码的字符串

字符串描述符使用的是 Unicode Standard 标准所定义的 Unicode 编码方式(统一的字符编码标准,采用双字节对字符进行编码)。一个 USB 设备内的字符串可以支持多种语言。当请求一个字符串描述符时,请求者使用由 Microsoft 为 Windows 所定义的一个 16 比特的语言 ID(LANGID)来说明所需要的语言,这一语言 ID 在 Microsoft 出版社出版的《开发 Windows 95 和 Windows NT 通用软件》一书中有阐述。对于所有的语言,字符串索引 0 都将返回该设备所支持的一列两字节的 LANGID 代码。一个 USB 设备可以省略所有的字符串描述符。

UNICODE 字符串描述符不是以 NULL 为终止。字符串长度是通过从该描述符内的第一个字节的数值中减去二得到的。

## 9.7 设备类型定义

所有的设备都必须支持上述寄存器和描述符定义。多数设备都提供了额外的寄存

器,并且可能的话,还提供了用于某一设备扩展的描述符。而且,设备可以提供对于一组设备来说是通用的扩展服务。为了定义一种设备类型,必须提供下列信息来完整定义该类型设备的表现和操作。

### 1. 描述符

如果该类型需要对标准描述符的特殊定义,那么在类型定义中必须将这些请求作为类型定义的一部分而包括进去。而且,如果该类型定义了一个标准的描述符扩展集,它们必须在类型定义中完全定义。任一扩充的描述符应该遵循标准描述符所使用的方法;例如,所有的描述符都应该以一个长度域为开始。

### 2. 接口和端点使用

当一种类型的设备被标准化之后,设备所使用的端点,包括如何使用这些端点都应该包括在设备类型定义内。设备可以进一步用独特的特性来扩展一个类型定义,只要这些特性满足该类型的基本定义。

### 3. 请求

专门用于该类型的所有请求都必须被定义。

## 9.8 设备通信

USB 通信模型描述主机和一个给定的设备之间,通过 USB 互联而传输的数据和控制信息量。主机和设备可以分成不同的层次,如图 9-2 所示。

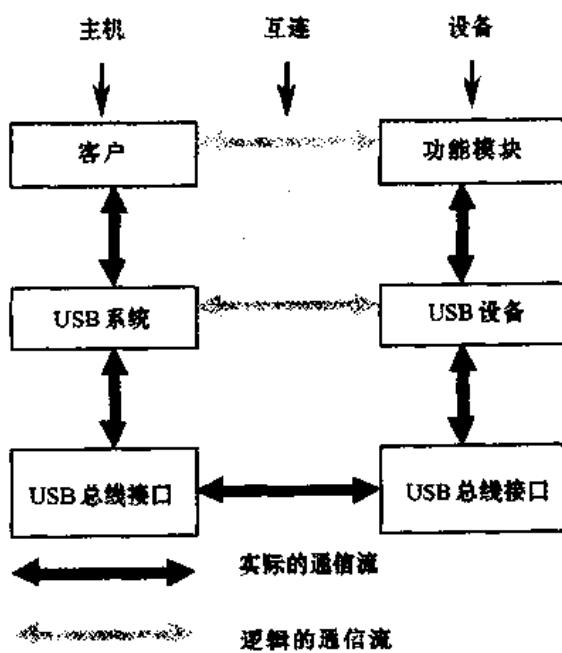


图 9-2 层间通信模型

在主机上进行的实际的通信,由垂直箭头指出,是通过 SPI 而进行的。设备上的层间关系与应用有关。在主机和设备之间,所有的通信必须最终在物理的 USB 电线上实现。但是,在水平的层次之间,有逻辑的主机设备接口。在驻留在主机上的客户软件和设备提

供的功能模块之间,基于当前正在使用设备和设备所提供的功能的应用的需要而进行的联系,是最典型的通信。这种客户和功能模块交互提出了对下面所有层次及其接口的需求。

本节从设备及基层次的观点出发,描述了通信模型。图9-3基于第8章所介绍的整体示意图给出了有关设备与主机通信的设备端的示意图。

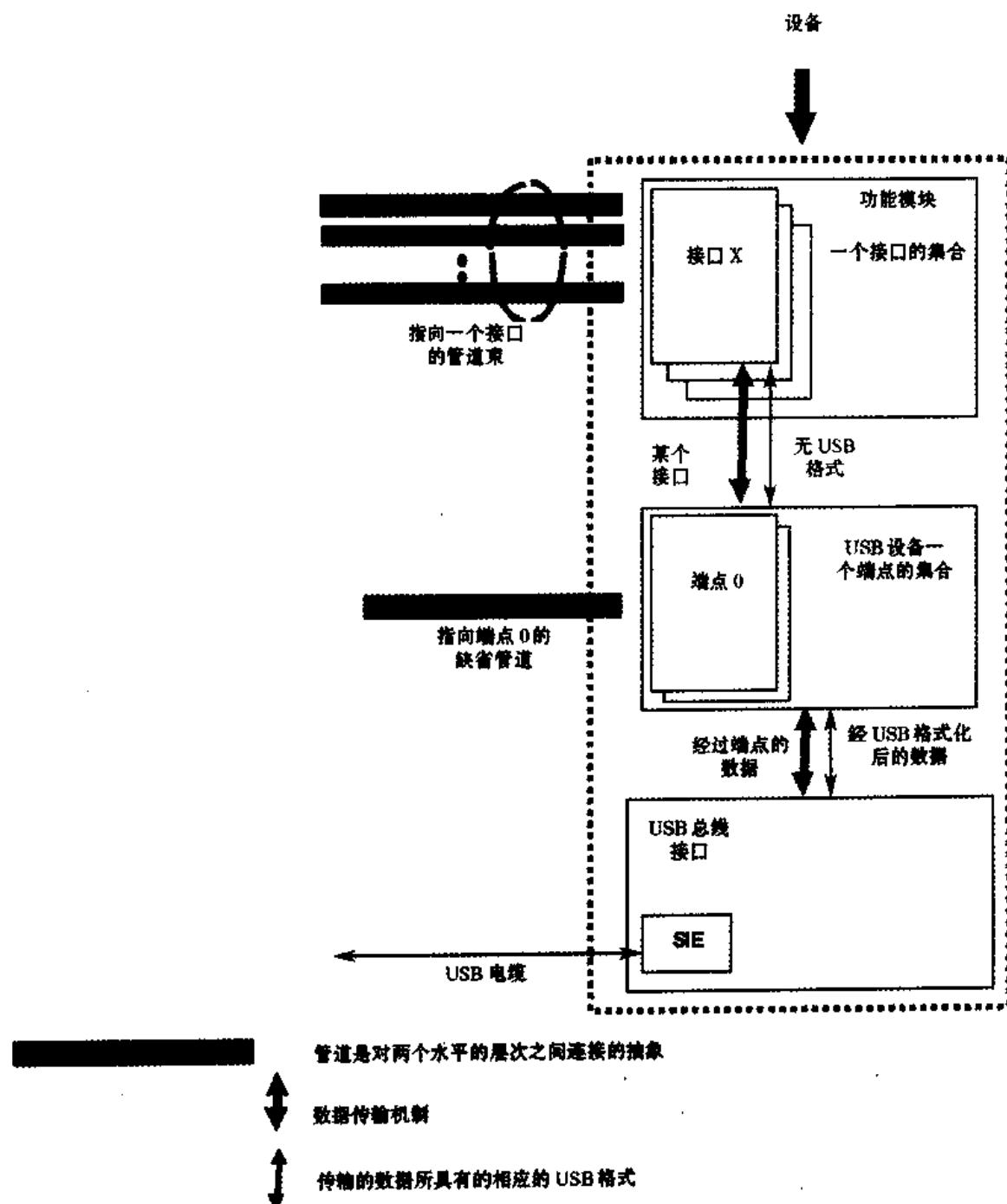


图 9-3 设备通信

USB总线接口在电气和协议层之间控制信息交互(参见第5章和第8章)。USB设备层代表了USB设备对于主机来说的一个统一的抽象。这里主要讨论这一层。融入到

一个给定的接口中的功能层,使用 USB 设备层所提供的功能来支持一个基于主机的应用的要求。

一个 USB 设备表现为一个端点集,每个设备都能支持不同类型的管道。每一个管道在某一时刻可以支持下列传输方式中的一种:

- (1) 控制;
- (2) 同步;
- (3) 中断;
- (4) 批量。

这些传输方式在第 4 章中已经作了详细的介绍。但是,每一种传输都要求相应的端点以某种方式来工作。一个给定的端点可以支持不同的传输类型。但是,一旦一个管道对应于一个端点之后,该端点只能使用单一的传输类型。在本节中,当提到用于某个传输类型的端点操作时,已经假定该端点同支持这一传输类型的一个管道合并了。端点所使用的基本通信机制为:

- (1) 管道模式;
- (2) 帧开始(SOF)同步;
- (3) 握手;
- (4) 数据触发。

一个管道的模式指出经过管道的数据流是流模式还是消息模式。设备可以使用主机所产生的 SOF 来同步其内部时钟。设备可以使用握手和数据触发器来实现差错和流量控制。

一个客户和一个功能模块之间存在的信息量需要一定的传输速率。客户、USB 和功能模块最好都使用差别很小的时钟速率。为了保证所有要求的数据都能以最小的缓存来进行传递,不同的时钟必须实现同步。对于同步选项的讨论请参考第 4 章。另外,为了支持时钟同步所隐含的恰能及时传递信息的能力,主机和设备之间所传输的数据分组的大小应该实行标准化,从而使经过一段时间后分组大小的变化降至最小。为了支持那些只要丢失的数据可以正确地传达,数据丢失就是可以接受的数据流,主机和设备会使用样值头来传送预计的传输容量。对于样值头的定义请参考第 4 章。

下面将从设备的观点出发,详细地描述这些基本的通信机制。每一种不同的传输类型都会以不同的方式来使用这些基本的通信机制。

本节将更详细地描述 USB 设备层所支持的基本通信机制。

### 1. 管道模式

一个管道不是支持流模式,就是支持消息模式的传输。在流模式中,数据流被看作一个单向的串行样值流。在消息模式中,数据以一个相关的字节集合来传输。消息模式管道总是被认为可以成为双向的管道。

一个流模式管道总是单向的。当其为流模式时,端点将收到一个要么是请求该端点发送数据,要么就是警告它有数据要发送给它的令牌。所发送的数据量总是等于或小于当前 Max PacketSize 域中为该端点所设置的数值。

消息传输以一个由主机至设备的命令为开始。设备可以用数据来响应该命令,主机可以在命令后加上用于该设备的数据,或者该命令不要求传输任何数据,在这种情况下,

它会发送一个 NULL 数据分组。在消息模式中,一个端点必须跟踪操作处于该模式所定义的哪一个阶段。一个端点希望一个消息序列中的第一个处理操作是用于后续的通信的一个建立操作。在收到了一个建立分组之后,一个端点通常期待着收到一个请求该端点发送数据(IN令牌)或警告其将所有数据传送给它(OUT令牌)的令牌。端点根据发起这一系列处理的建立命令,可以知道接下来的处理的传送方向是什么。某些建立命令不要求后续的处理传向或来自于一个端点。建立处理分组的长度总是八个字节或更少。后续的处理总是小于或等于当前该端点 MaxPacketSize 所规定的值。

## 2. 同步

主机以均匀定时的间隔来向总线提供一个特殊的 SOF 令牌。在差错容限内,SOF 之间间隔 1ms。端点可以利用接收该令牌来使其相应的时钟同步于 USB 时钟。这样可以使端点将其数据消耗或产生速率同主机速率相匹配。

不是所有的端点都需要 SOF 同步。某些需要同步的端点所具有的时钟不能同步于 USB 所提供的 1ms 总线时钟。这些设备有两个选择。它们可以试着让整个 USB 和它们同步,或定期地调整其传输速率来补偿 USB 时钟和它们的时钟之间所存在的差别。

在一个 USB 实例当中,USB 最多可以提供一个客户来调整主机 SOF 的产生。该客户利用相应的设备所提供的反馈来进行调整。但是,SOF 令牌产生的速率保持为 1ms。对这一调整机制的完整讨论请参考第 8 章。如果一个端点没有配置成可以让其使用 SOF 握手分组来调整 USB 时钟,或者该端点不能进行这样的时钟调整,那么该端点就必须不断地调整其数据流。

因此,正如以上所述,对于 SOF,一个端点可以使用三种可能的同步交互类型。这些端点可以:

- (1) 使其时钟正确地同步于存在的 USB 时钟。
- (2) 调整总线时钟。
- (3) 通过调整其数据流来和主机同步。

重要的是要注意一个需要同步的端点,如果不能实现(1)中所描述的同步类型,但却可以实现(2)中所描述的同步类型,那么它也必须实现(3)中所描述的同步类型。这是因为像这样的一个端点,不能保证会选择它来调整总线时钟。而在整个 USB 系统中只能利用一个设备来调整 SOF。

## 3. 握手

端点使用 USB 处理中的握手阶段来向主机传送差错和数据流要求。端点也可以接收来自于主机的传送差错条件的握手信息。根据所支持的传输类型的不同,端点所使用的握手类型也会有差别。这些握手信息在第 8 章中有详细地描述。

## 4. 数据触发器

当出现差错或流量控制情况时,某些管道可以被允许跳过产生该情况的那一帧;并且在下一个帧中传输为这一帧所安排的数据。在某些情况下,可能会出现这样的情况:数据的接收者已经向发送者指出数据已经被成功地接收了,但是由于一个总线差错,发送者认为该数据并未成功地被接收。那么该发送者就要重传同样的数据。接收者则需要某一机制来确定它现在所接收的数据是重传的数据,而这一数据它已经收到了,现在所接收的数据并不是一个新的数据。

USB 利用数据触发器来提供这一信息,它是用于处理中的数据阶段的 PID。依据传输类型,端点需要理解数据触发器的工作方式和相应地产生或处理 PID。对数据触发器的更完整的讨论请参考第 8 章。

表 9-27 总结了 USB 通信机制。

表 9-27 USB 通信机制

传输类型	控制	同步	中断	批量
管道模式	消息	流	流	流
同步方式	无	总线,外部或软件	无	无
握手	是	未使用	是	是
数据触发	是	忽略	是	是
必需的缓冲区	最低为 8 个字节	帧通信量的两倍	单个处理	单个处理
差错和状态控制	予以保证的传输仅对严重的差错进行报告	报告丢失或遭破坏的数据 不进行重试操作	予以保证的传输仅对严重的差错进行报告	予以保证的传输仅对严重的差错进行报告

# 第 10 章 USB 主机：硬件和软件

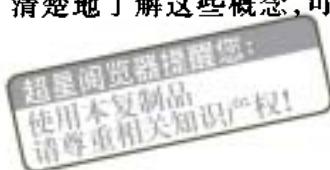
USB 互连支持在一个主机和一个 USB 设备之间传输数据。本章阐述了为了简化驻留在主机上的一个客户软件和在一个设备上实现的一个功能模块之间所进行的 USB 通信，所必需的主机接口。

本章具体包括以下内容：

- USB 主设备概述；
- 控制机制；
- 数据流；
- 搜集状态和性能统计信息；
- 电气接口考虑；
- 主控制器请求；
- 状态控制；
- 串行器/解串器；
- 锁产生；
- 数据处理；
- 协议引擎；
- 传输差错控制；
- 软件机制概述；
- 设备配置；
- 资源管理；
- 数据传输；
- 公共数据定义；
- 主控制器驱动程序；
- 通用串行总线驱动程序。

在前面我们已经提到, USB 系统中的通信要由主机来控制。USB 主机的概念同以往我们所接触到的以太网中的主机概念有一些差别。在 USB 系统中, USB 主机指的是计算机软件和硬件的集合, 而不是单纯指硬件; 而在以太网中, 主机是指一台负责控制网络中的通信, 为网络中的终端提供服务的核心计算机。因此, USB 规范中专门列出了一章来讨论 USB 主机。其中的许多概念, 例如主控制器驱动程序和 USB 总线驱动程序对于开发 USB 功能设备的应用软件和设备驱动程序都很重要。清楚地了解这些概念, 可以使我们从整体上把握和深入理解 USB 系统和主机的通信。

下面我们就来讨论 USB 主机。



## 10.1 USB 主设备概述

图 10-1 给出了 USB 通信模型的基本通信流和相互关系。

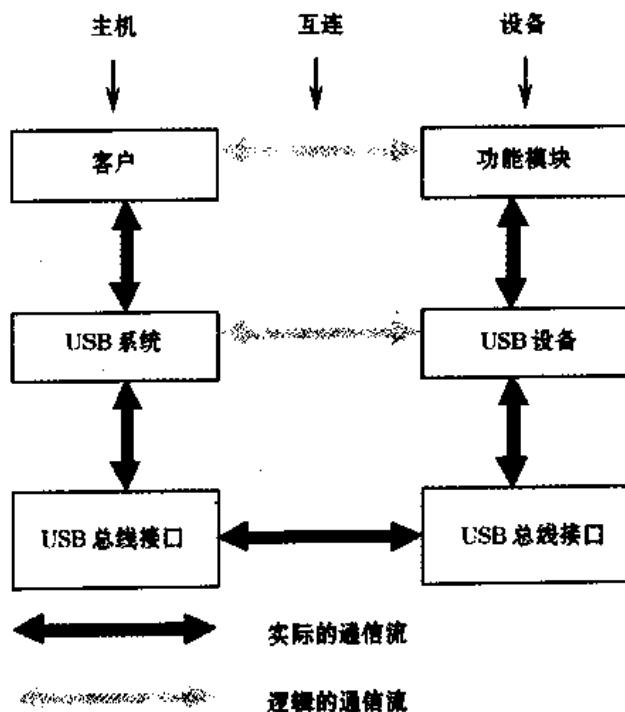


图 10-1 层间通信模型

主机和设备可以分成图 10-1 中所描述的清晰的层次。主机上的实际通信用垂直箭头指出。在设备上的对应接口与某个实现有关。在主机和设备之间进行的通信, 最终都必须出现在物理的 USB 电缆上。但是, 在水平方向上的各层之间都有主机和设备的逻辑接口。驻留在主机上的客户软件和设备所提供的功能模块之间存在的这些通信, 都是以根据当前正在使用设备和设备所提供的功能的应用的需要而进行的联系为代表。

这一客户和功能模块的交互提出了对下面所有层次和接口的需求。

本章所阐述的模型是从主机和其分层的观点出发而给出的。图 10-2 是根据第 5 章所介绍的整体观点, 给出了主机对它和设备的通信所持的看法。

每一个 USB 中只有一个主机。它的主要分层是:

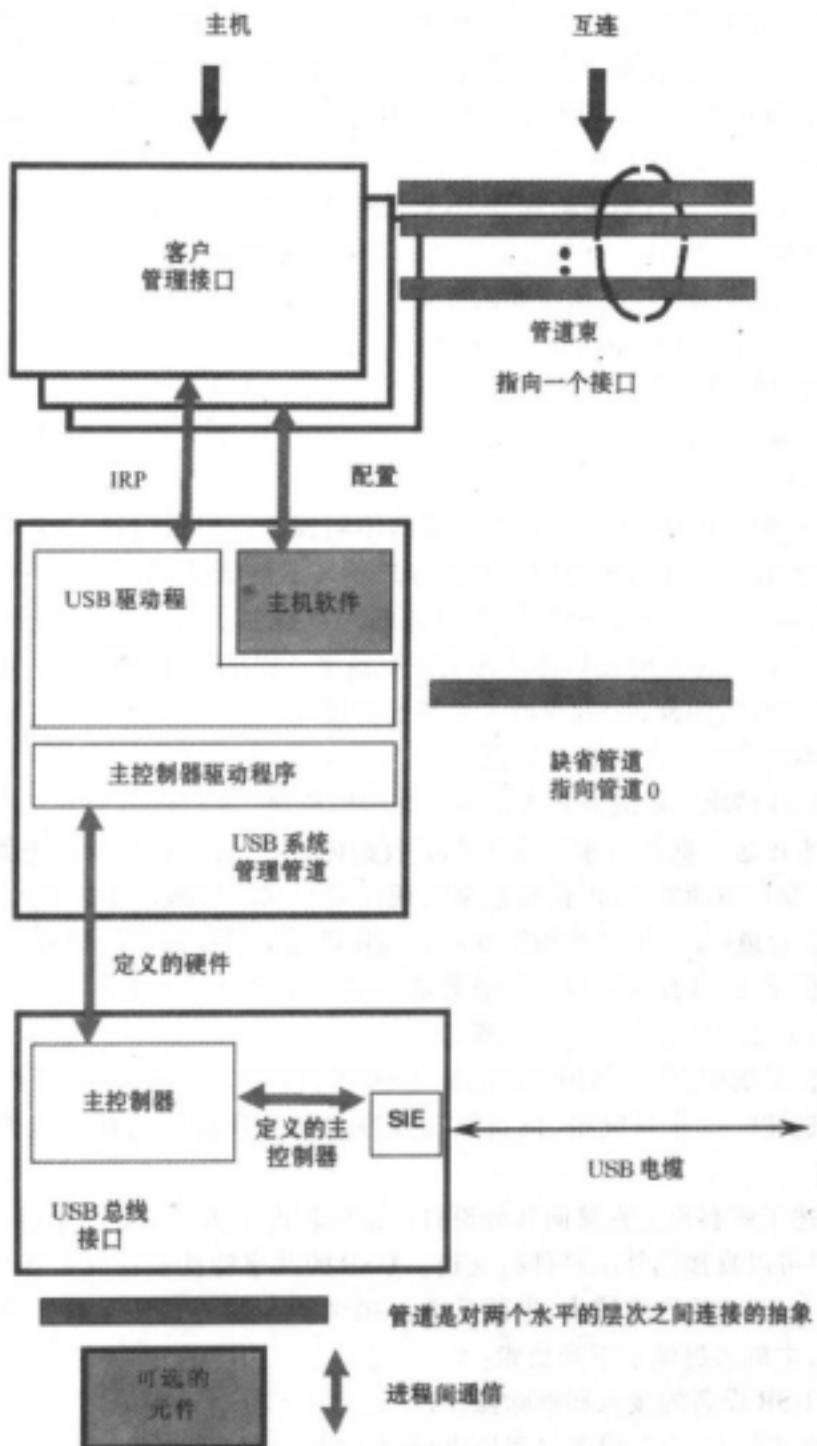


图 10-2 主机通信

- (1) USB 总线接口;
- (2) USB 系统;
- (3) 客户。

USB 总线接口控制了电气和协议层(参见第 7 和第 8 章)的交互。从互连的角度看,一个类似的 USB 总线接口由主机和设备一起提供,如串行接口引擎(SIE)。但是在主机

上,由于在 USB 内主机所拥有的唯一性,USB 总线接口具有额外的责任并且以主控制器的形式而实现。主控制器也由一个集成的或根集线器来向主机提供连接点。

USB 系统利用主控制器来控制主机和 USB 设备之间所进行的数据传输。USB 系统和主控制器之间所存在的接口依赖于主控制器的硬件定义。USB 系统同主控制器一起,对互连时所出现的客户对数据传输和 USB 处理操作所持的不同看法进行转换。这包括了支持任何附加的 USB 特性,例如协议封装。USB 系统同时还负责管理 USB 资源:即使客户对 USB 的访问成为可能时所需要的带宽!

USB 系统有三个基本的组件:

- (1) 主控制器驱动程序;
- (2) USB 驱动程序;
- (3) 主机软件。

主控制器驱动程序(HCD)可以使将可能的不同控制应用映射到 USB 系统的工作变得更加简单,从而使与一个设备进行交互的客户不须了解该设备是同哪一个主控制器相连。USB 驱动程序(USBD)为一个 USB 设备的用户提供了基本的主机接口(USBDI)。HCD 同 USBD 之间所存在的接口称之为控制驱动程序接口(HCDI)。该接口永远不能直接供用户来使用,因此,USB 规范未对其进行定义。但是,某个 HCDI 由每一个支持了不同主控制器应用的操作系统来定义。

USBD 以 I/O 请求分组的方式来提供数据传输机制,而 I/O 请求分组内包括了一个通过某个管道来传送数据的请求。除了提供数据传输机制之外,USBD 还负责向其客户提供一个 USB 设备的抽象,可以控制它来实现配置和状态管理。作为该抽象的一部分,USBD 拥有缺省管道(参见第 5 章和第 9 章),利用该管道可以通过实现对所有 USB 设备的访问来进行标准 USB 控制。这一缺省管道代表了 USBD 同一个如图 10-2 所示的一个 USB 设备抽象之间所存在的一个逻辑通信。

在某些操作系统中,可以利用附加的非 USB 主机软件来向设备驱动程序提供配置和装载机制。在这样的操作系统中,设备驱动程序将使用所提供的接口以取代直接访问 USBDI 的机制。

客户层描述了所有负责直接同其外设打交道的软件实体。当每一个设备分别接入系统时,这些客户可以直接同外围硬件打交道。USB 的共享特性被存放在客户和其设备之间所存在的一个 USB 软件堆栈内;也就是说,一个客户不能直接访问设备硬件。

总的来说,主机层提供了下列功能:

- (1) 检测 USB 设备的接入和拆除操作;
- (2) 管理在主机和 USB 设备之间所进行的 USB 标准流量控制;
- (3) 管理主机和 USB 设备之间所存在的数据流;
- (4) 搜集状态和性能统计信息;
- (5) 控制主控制器和 USB 设备之间存在的电气接口,还包括提供有限的电源。

下列几节将更详细地描述 USBDI 的责任和对它的要求。某个主机平台和操作系统的组合所使用的实际接口将在适当的操作系统环境指南中阐述。

所有的集线器都提供了一个状态管道,在该管道上可以报告集线器和其端口的变化。这包括对一个设备何时接入其端口,或从其端口上拆除的指示。一个 USBD 客户,通常



为集线器驱动程序,将作为集线器状态变化管道的所有者来接收这些指示。对于设备的接入,集线器驱动程序会初始化该设备的配置过程。在某些系统中,集线器驱动程序是操作系统为管理设备而提供的主机软件一部分。

### 10.1.1 控制机制

在主机和一个 USB 设备之间,可以利用带内或带外信令来传送控制信息。带内信令在主机不知晓的情况下,将控制信息和管道内的数据相混合。带外信令则把控制信息放在一个分离的管道内。

每一个连接的 USB 设备都有一个称之为缺省管道的消息管道。这种在一个主机和一个 USB 设备之间所形成的逻辑联系,可以用于像设备枚举和配置这样的 USB 标准控制操作,缺省管道向所有的 USB 设备提供了一个标准的接口。缺省管道也可以用于某个设备的通信,而由掌管着所有 USB 设备的缺省管道的 USBD 来进行中介。

某个 USB 设备可能允许使用其它的消息管道来传输某个设备的控制信息。这些管道使和缺省管道一样的通信协议,但是所传输的信息由 USB 设备而定,而没有被通用串行总线规范来标准化。

USBD 支持对缺省管道的共享,它可以和其用户一起拥有和使用该管道。它也提供了同该设备相对应的任意其它管道的访问。

### 10.1.2 数据流

主控制器负责在主机和 USB 设备之间传输数据流。这些数据传输将被作为一个连续的字节流来对待。USB 支持四种基本的数据传输类型:

- (1) 控制传输;
- (2) 同步传输;
- (3) 中断传输;
- (4) 批量传输。

有关这些传输类型的其它信息请参看第 5 章。

每一个设备都实现了一个或多个接口,一个客户利用这些接口来同设备通信。每一个接口都包括 0 个或更多的管道,这些管道可以在客户和设备上的某个端点之间分别传输数据。USBD 会在主机软件的明确要求下创建接口和管道。当发出配置请求时,主控制器根据主机软件所提供的参数来提供服务。

依据将要传输的数据的传递要求,一个管道有几种特性。这些特性包括:数据所要传输的速率,是以稳定的速率或是以变化的速率来提供数据,在传送之前数据可以被延迟多久和所传输数据的丢失是否会是灾难性的。

一个 USB 设备端点需要说明某个管道所要求的特性。因而端点被作为一个 USB 设备特征信息的一部分来加以说明。有关细节请参考第 9 章。

### 10.1.3 搜集状态和性能统计信息

作为用于主机和 USB 设备之间形成的所有控制和数据传输的一个公共通信器件,USB 系统和主控制器被用来跟踪状态和性能信息。一旦向主机软件发出了请求,就可以

提供这些信息,以允许软件来管理状态和性能信息。

#### 10.1.4 电气接口考虑

主机为同根集线器相连的 USB 设备提供电源。一个端口所提供的电源功率在有关集线器的章节中给出。

### 10.2 主控制器请求



在所有应用当中,对于 USB 和与之相连的设备而言,主控制器肩负着同样的基本职责。下面将讨论这些基本职责。

主机和 USB 都可以向主控制器发送请求。下面是对所提供的功能的一个简要描述(对于每一个功能在后续的几个小节中都将详细地予以讨论):

- (1) 状态控制:作为主机的一个元件,主控制器报告和管理它的状态。
- (2) 串行器/解串器:对于由主机传输的数据,主控制器将协议和数据信息,从其原有的模式转变为在 USB 上传输的一个比特流。对于主机将要接收的数据,控制器进行相反的工作。
- (3) 帧产生:主控制器以 1ms 为周期产生 SOF 令牌。
- (4) 数据处理:主控制器处理向主机发送的数据传输的请求以及来自于主机的对数据传输的请求。
- (5) 协议引擎:主控制器支持 USB 所说明的协议。
- (6) 传输差错控制:当已定义的差错类型进行检测和作出响应时,所有的主控制器都要执行同样的操作。

下面几小节将就主控制器所需实现的功能进行更详细地讨论。

#### 10.2.1 状态控制

作为一个主机中的一个普通元件,主控制器拥有一系列由 USB 系统管理的状态。而且,主控制器拥有与 USB 相关的两个方面所具有的状态:

- (1) 根集线器;
- (2) 状态变化的传送。

根集线器向集线器驱动程序发送和其它 USB 设备相同的标准状态。主控制器为集线器提供了对这些状态和其出现的支持。有关 USB 状态的详细讨论,包括其相互关系和变化请参考第 9 章。

主控制器的整体状态同根集线器和整个 USB 的状态是密不可分的。对于所连接的设备而言,它所能识别的任一主控制器状态变化必须在相应的设备状态变化中反映出来,从而使主控制器和设备的状态保持一致。

USB 设备通过使用重新开始信号来请求一个唤醒操作(参见第 11 章),该操作会使集线器拆除连接,并使设备返回其配置完成的状态。主控制器自己可以通过同样的发送信号的方法来产生一个重新开始事件。主控制器必须通过一种机制,或一种由系统应用

而特定的机制向主机的其余部分告知这个重新开始事件。

### 10.2.2 串行器/解串器

通过物理的 USB 而进行的实际数据传输是以一个串行比特流形式而出现的。一个串行接口引擎(SIE),无论它是作为主机的一部分还是一个 USB 设备的一部分而实现的,都要由它来控制 USB 传输的串行和解串操作。在主机上,SIE 是主控制器的一部分。

### 10.2.3 帧产生

主控制器负责把 USB 时间分成 1ms 的时间段,称之为“帧”。它是通过以 1.0ms 的间隔发送帧开始(SOF)的方法来建立帧的,如图 10-3 所示。SOF 令牌是一个帧周期内的第一个被传输的内容。在发送了一个 SOF 令牌之后,在帧周期所剩余的时间内,主控制器可以自由地发送其它处理操作。当主控制器处于其正常的工作状态时,不管有无其它的总线操作,SOF 令牌必须以 1ms 的间隔速率连续地产生。如果主控制器进入了一个它不能在总线上提供电源供应的状态,它就不能产生 SOF。而且,如果主控制器不再生成 SOF,它可以进入一个低功耗状态。

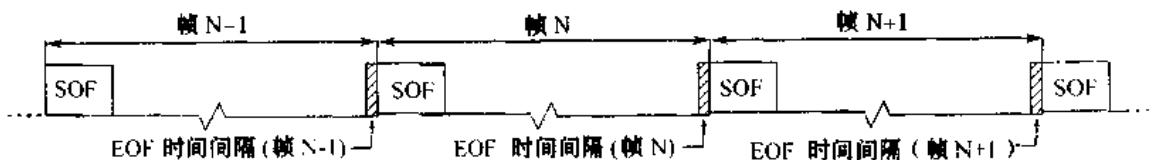


图 10-3 帧产生

SOF 令牌具有访问总线的最高优先权。集线器中的串扰电路在帧结束(EOF)期间,从电气上对任意一个有效的发送器进行隔离,为 SOF 的传输提供一条空闲的总线。

(1) 主控制器必须允许 USB 的帧长可以在 ±1 个比特周期的范围内进行调整(参见第 10.5.3 节中的有关部分)。主控制器保留了当前的帧标号,因为 USB 系统有可能会读取该标号。当前的帧标号用于唯一地将该帧同其它帧区别开来。

(2) 在每一帧周期结束时增加。

(3) 经过下一帧后有效。

主机在每一个 SOF 令牌传输中会传送当前帧标号的低 11 位比特。当主控制器发出请求时,当前的帧标号就是在该请求完成时已经存在的帧标号。由主机(主控制器或 HCD)所返回的当前帧标号至少为 32 比特,尽管主控制器自身并不需要保留超过 11 位比特的标号。

在 EOF 期间,主控制器可以停止传输。当 EOF 间隔开始时,专门为刚刚通过的那一帧所安排的任一处理操作都会被重试。如果在遇到一个 EOF 时间间隔时主控制器正在执行一项处理操作,主控制器将终止该项处理。

### 10.2.4 数据处理

主控制器负责接收来自于 USB 系统的数据并将其发送至 USB,还负责接收来自于 USB 的数据并且将其发送到 USB 系统。对于 USB 系统和主控制器之间所存在的通信而

言,所使用的某种格式依赖于特定的实现方式,并且要在第 5 章中所提到的对传输行为所规定的原则范围之内进行。

### 10.2.5 协议引擎

主控制器管理 USB 协议层接口。它在发送出的传输中加入适当的协议信息。它也会适当地去除输入的协议信息。



### 10.2.6 传输差错控制

主控制器必须能够检测出从主机的角度出发而定义的下列传输差错情况:

- (1) 在一个主机传送完令牌或分组后出现的超时情况。这一错误当被寻址的端点没有响应,或当传输的内容的结构受到严重的破坏而使目标端点无法识别它时发生。
- (2) 数据错误会导致丢失传输或不正确的传输。
- (3) 主控制器发送或接收的一个分组比该传输所要求的分组短;例如,一个跨过 EOF 的传输或主控制器缺乏可用资源。
- (4) 在接收的数据分组上的一个无效的 CRC 域。
- (5) 协议错误。
- (6) 一个无效的握手 PID。例如,一个畸形的或不恰当的握手 PID。
- (7) 一个假 EOP。
- (8) 一个比特填充错误。

对于每一个批量、命令和中断处理而言,主机会保留一个错误次数记录。错误是由上述几种情况产生的,而不是因为一个端点未对一个请求作出响应而产生的结果。这一记录的数值反映了该处理所遇到的传输错误的次数。如果对于一个给定的处理而言,其错误次数记录达到了 3 次,主机就会重试该传输。而当一个传输由于过多的错误而重试时,最后一个错误类型会被指出。无论结果如何,同步处理只会被尝试一次,因此对于该类型没有保留差错记数。

## 10.3 软件机制概述

HCD 和 USBD 代表了基于对不同层次的抽象的软件接口。而且,它们要能以某种方式共同工作,以满足 USB 系统的整体要求(参见图 10-2)。对于 USB 软件堆栈所提出的要求主要以对 USBDI 的要求来表述。在 USBD 同 HCD 之间如何进行任务化分并没有被定义。但是,在某个操作系统环境中,一个必须满足的对 HCDI 的要求是支持主控制器的多个定义。

HCD 提供了对主控制器的抽象和在主控制器看来经过 USB 的数据传输的抽象。而 USBD 则提供了对 USB 设备的抽象和 USBD 上的客户同 USB 设备上的功能模块之间所存在的数据传输的抽象。总的来说,USB 软件堆栈作为在客户和功能模块之间简化数据传输的装置,以及一个用于 USB 设备的某个 USB 接口的控制点来使用。由于要简化数据传输,USB 软件提供了缓冲区管理能力,并可以根据客户和功能模块的需要来对数据

发送者进行同步。

对 USBDI 的具体的要求将在本章中的后面几节来讨论。用来满足这些要求的实际函数将在有关 HCDI 和 USBDI 的相应的操作系统指南中进行说明。下面我们将通过 USBDI 来完成数据传输所包含的过程。



### 10.3.1 设备配置

不同的操作系统环境使用不同的软件元件和不同的事件序列来进行设备配置。USB 系统不会假定某个操作系统所使用的方法。但是,任一 USB 系统的实现都必须遵循一些基本的要求。

在某些操作系统中,由已经存在的主机软件来满足这些要求。在其它环境中,则由 USB 系统来提供这些能力。

USB 系统采用了一个称为集线器驱动程序的一个 USBD 的专门用户,由它作为一个用于向某个集线器添加设备和从某个集线器拆除设备的处理中心。一旦集线器驱动程序接到了这样的指示,它就会以某个操作系统所特定的方式,利用另外的主机软件和其它 USBD 客户来识别和配置设备。图 10-4 所示的这一模型是下列讨论的基础。

当一个设备接入时,集线器驱动程序将接收到检测这一变化的集线器所发来的指示信息。集线器驱动程序利用集线器所提供的信息,来请求一个来自于 USBD 的设备标识。接着 USBD 为该设备建立一个缺省管道并向集线器驱动程序返回一个设备标识。

现在设备已经为配置作好了准备。对于每一个设备而言,必须在三种类型的配置操作完成之后,它才能为使用做好准备:

(1) 设备配置:这包括建立设备的所有 USB 参数和使用所有该设备可见的主机资源。这些工作是通过在设备上设置配置值而完成的。不用对设备全部重新进行配置,也可以允许像使用交替的设置这样的有限的配置变化存在。一旦一个设备被配置之后,从它的观点来看它已经为使用作好了准备。

(2) USB 配置:为了实际建立一个准备由一个客户来使用的 USBD 管道,客户还必须说明另外的 USB 信息,而这些信息对于设备来说是不可见的。这一信息称为管道原则,描述客户会如何来使用管道。它包括了客户利用一个 IRP 所能传输的最大数据量,客户可以使用的最大服务时间间隔和客户的指示标识等项。

(3) 功能配置:一旦配置类型 1 和 2 完成之后,从 USB 的观点来看管道已经完全为使用做好了准备。但是,在客户可以实际使用管道前,可能还要额外地由供应商或设备类型而定义的建立操作。这一配置是在设备和客户之间所进行的独立操作,并未由 USBD 对其进行标准化。

接下来的几段将描述设备和 USBD 的配置要求。

实际的设备配置由负责配置的软件来完成。根据某个操作系统实现,负责配置的软件包括:

- (1) 集线器驱动程序。
- (2) 其它主机软件。
- (3) 设备驱动程序。

配置软件首先读取设备描述符,然后会要求对每一个可能的配置加以说明。它可能

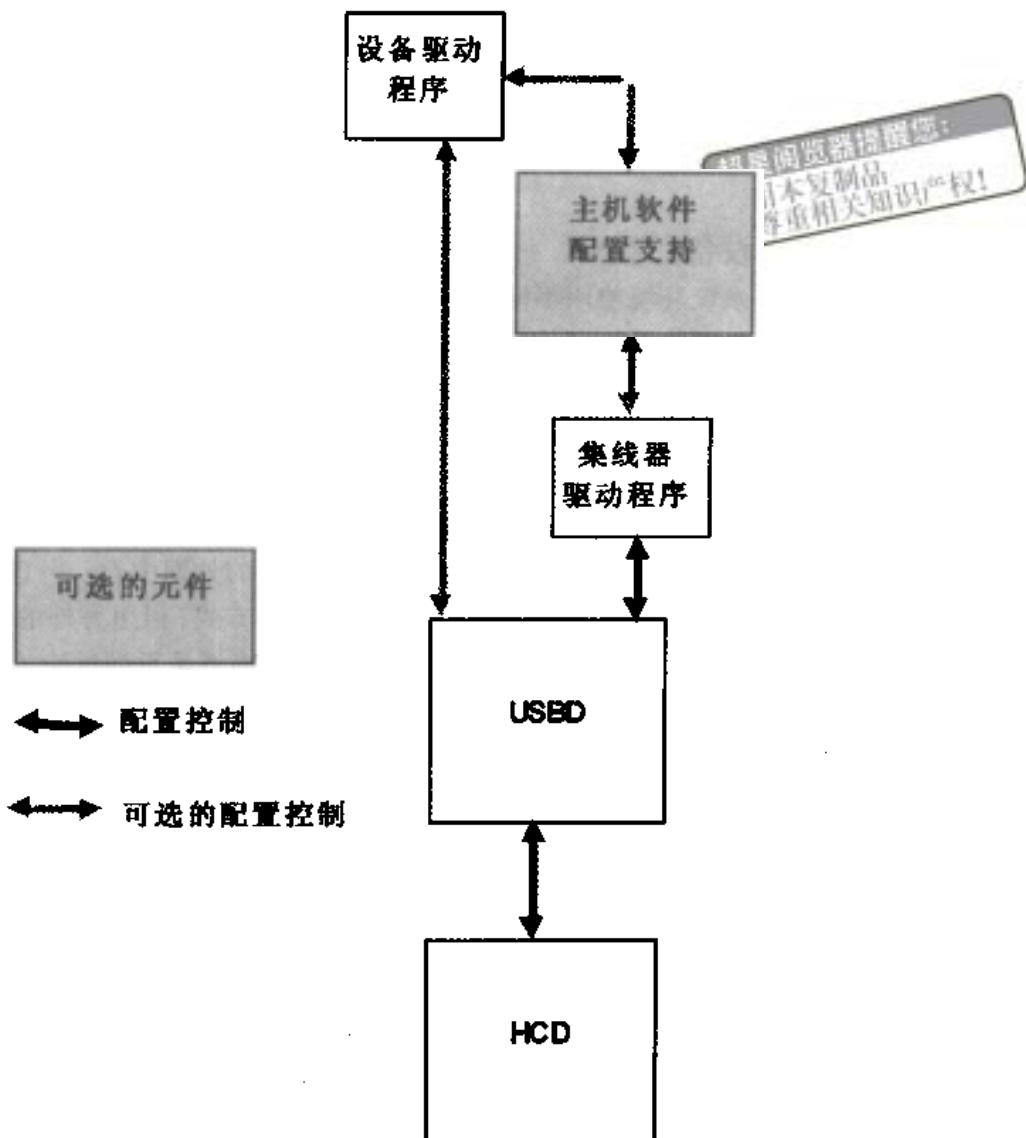


图 10-4 配置交互

使用所提供的信息来装载一个客户,例如一个设备驱动程序,最初要由它来和设备进行联系。配置软件也许还要加上来自于该设备驱动程序的输入,将为该设备选择一个配置方案。启动一个设备的配置操作会建立该设备上的所有端点,并返回一个用于 USBD 客户的数据传输的接口集。每一个接口都是单个客户所拥有的管道的集合。

最初的配置操作会使用接口的缺省设置和每一个端点的缺省带宽。另外,当选择最初配置方案时,一个 USBD 实现可以允许客户说明可以替换的接口。USB 系统可以保证有可用的资源来支持该端点,并且如果是这样的话,就会为其分配所需要的带宽。有关资源管理的讨论请参考 10.3.2 节。

现在,设备已经配置完成了,但是所创建的管道并未为使用作好准备。当客户利用设置一个原则来说明它是如何同管道进行交互的方法来初始化每一个管道时,USB 配置操作就可以完成了。说明的信息中有客户的最大服务时间间隔和指示信息。作为设置一个策略后的结果,USB 系统所采取的操作之一是确定所要求的缓冲区工作空间比客户所提

供的数据缓冲区空间要大多少。所要求的缓冲区尺寸根据客户所选择的使用方式和 USB 系统的每一个传输要求为基础。

当 IRP 完成后,客户可成功地收到指示,或者是出现差错。客户也可以独立地唤醒 USB 标志信息来检测未处理的 IRP 的状态。

客户也可以选择进行诸如为一个接口启动一个可交替的设置或改变为某个管道所分配的带宽等等配置修改操作。为了实现这些改变,管道或接口必须分别处于空闲状态。

### 10.3.2 资源管理

只要 USBD 为一个给定的端点建立了一个管道,USB 系统必须决定它是否能够支持该管道。USB 系统是根据端点描述符中描述的要求来作出决定。为了为一个端点建立一条管道而必须支持的端点要求就是用于该端点传输所必需的带宽。检查可用的带宽分为两个步骤。首先,计算出一个处理的最大执行时间。然后,会检查帧进度表以确定是否可以适应所指出的处理操作要求。

为同步和中断管道分配保证的带宽,以及确定某个控制或批量处理是否可以装入一个给定的帧中,可以由 USB 系统中的试探软件来确定。如果在主控制器中实际的处理执行时间超过了预先确定的时间,主控制器负责保证帧的完整性(参见 10.2.3 节)。下列讨论将说明对 USB 试探系统的要求。

为了确定是否可以进行带宽分配,或者一个处理是否可以放在某一帧内进行传输,就必须计算出处理的最大执行时间。而要计算出这一最大处理操作执行时间,则需要提供下列信息(注意,某些信息是由一个代理而不是由一个客户来提供):

- (1) 所传输的数据字节数 (MaxPacketSize);
- (2) 传输类型;
- (3) 拓扑结构深度。如果允许较小的精确度,可以假定拓扑结构的最大深度。

所得的计算结果必须包括比特传输时间,通过拓扑结构后的信号传播时延和像主控制器所要求的准备或恢复时间这样的由实现方式而定的时延。请参考第 5 章中给出的这些计算可以利用的公式。

### 10.3.3 数据传输

所有客户和功能模块的基础是接口:与某个 USB 设备相对应的一个相关管道的集合。

某一个给定的接口实际上由主机上的一个客户来控制。该客户通过为这一管道设置操作策略来初始化一个接口中的每个管道。这包括每个 IRP 可以传输的最大数据量和用于该管道的最大服务时间。一个服务时间段用毫秒(ms)来描述,它描述的是为一个同步管道传输一个 IRP 数据所需要的时间间隔。当某个请求完成之后,客户会被告知。客户管理每一个 IRP 的大小,从而可以保持其任务周期和时延限制。其它策略信息还包括向客户提供的指示信息。

客户提供了存储发送数据所要求的缓冲区空间。USB 系统利用策略信息来确定它可能要求的额外的工作空间。

客户将其数据视为一个连续的串行流,它利用一个同其它总线技术所提供的那些数

据流管理方法相类似的方法来进行管理。在内部,USB 系统可以根据它自己的策略和任一主控制器限制,而将客户请求分拆成更小的请求通过 USB 进行传输。但是,只要 USB 系统决定进行这样的分割就必须满足两个要求:

(1) 将数据流分成更小的分支,对于客户而言是不可见的。

(2) USB 样值在经过总线处理之后不会分离。有关 USB 样值的定义及其同管道一般样值大小的关系请参考第 9 章。

当一个客户想传输数据时,它会向 USBD 发出一个 IRP。依据数据传输的方向,会提供一个满的或空的数据缓冲区。当该请求结束时(成功地完成或者是由于一个错误情况而中止),IRP 和其状态都应该返回客户。相类似的是,这一状态也是根据每一个处理而提供的。

#### 10.3.4 公共数据定义

为了允许客户可以尽可能直接地从其设备接收请求结果,就要求将处理数量和设备同客户之间所要求的复制操作减至最低。为了简化这一过程,IRP 的某些控制方面得到了标准化,从而使客户所提供的信息可以由堆栈中的不同层次来直接使用。用于这一数据的特殊格式依赖于操作系统中 USBDI 的实现。实际上某些数据单元对客户而言根本不可见,但却是作为客户请求的结果而产生的。

接下来的数据单元为一个请求定义了有关的信息:

- (1) 对与这一请求相对应的管道的标识。标识一个管道也包括对用于这一请求的传输类型等信息的说明。
- (2) 对某个客户的指示标识。
- (3) 用于传输或接收的数据缓冲区的位置和长度。
- (4) 提供该请求的完整状态。根据要求,简要状态和详细的有关每个处理的状态都应提供。
- (5) 工作空间位置和长度。这与实现方式有关。

用于将请求传送至 USBD 的实际机制与操作系统有关。但是,除了上述对必须可以使用的与请求有关的信息所提出的要求之外,还有对如何来处理这些请求的要求。基本的要求已经在第 4 章中说明了。另外,USBD 提供了一种机制来选择一组同步 IRP,而使每一个 IRP 中的第一个处理传输都在同一帧内进行。USBD 也提供了一种机制来为一个缺省管道选择一个不会被中断的、由供应产或类型而定的请求集。对于这样一个不会被中断的请求集,其它任何请求,包括标准、类型或供应商请求都不会被插入到操作流中。如果这一请求集中的任何一个请求失败,那么整个请求集都会被重试。

### 10.4 主控制器驱动程序

主控制器驱动程序是一个对主控制器硬件和主控制器对通过 USB 进行的数据传输的看法的抽象。HCDI 需满足下列要求:

- (1) 提供了对主控制器硬件一个抽象。
- (2) 提供了一个对通过 USB 互连而由主控制器进行的数据传输的抽象。

(3) 提供了对主控制器资源分配(和不予分配)的抽象,从而支持对 USB 设备的服务保证。

(4) 按照集线器类型定义来说明根集线器及其操作。这包括为使集线器驱动程序同根集线器可以完全像它同其它任一集线器一样进行交互而向根集线器提供的支持。特别地,即使一个根集线器可以用一个硬件和软件的组合来实现,刚开始时根集线器也会对缺省设备地址作出响应(从用户的角度而言),返回描述符信息,支持对设备地址进行设置和其它集线器类型请求。但是,在给定了主控制器和根集线器之间可能具有的紧密集成性的条件下,可能需要,也可能不需要为完成这一操作而产生总线处理。

HCD 提供了一个软件接口(HCDI),它实现了所要求的抽象。HCD 的功能模块提供了一个可以隐藏主控制器硬件细节的抽象。位于主控制器硬件下方的是物理 USB 和连接的所有 USB 设备。

HCD 位于 USB 软件堆栈中的最低层。而且 HCD 只有一个客户:通用串行总线驱动程序(USBD)。USBD 将来自于许多客户的请求映射到适当的 HCD。一个给定的 HCD 可以管理许多主控制器。

对一个客户而言,HCDI 是不能直接访问的。因此,对于 HCDI 的某个接口要求不会在此进行讨论。

## 10.5 通用串行总线驱动程序

一个通用串行总线驱动程序(USBD)提供了操作系统组件,主要是设备驱动程序,可以用来访问 USB 设备的一个机制集合。对 USB 设备的访问由 USBD 来提供。而且 USBD 的实现由操作系统而定。USBD 所提供的机制是通过对其运行的操作系统环境所提供的机制进行适当地利用和必要的加强来实现的。下列讨论将以所有的 USBD 实现所要求的基本能力为中心。有关某个环境中的对 USBD 操作所进行的说明,请参考有关的操作系统环境指南对 USBD 的说明。USBD 的单个实现指挥着对一个或多个 HCD 的访问,而这些 HCD 则依次同一个或多个 USB 主控制器相连。如果允许的话,USBD 的实现是如何管理的可以根据操作系统环境而决定。但是,从客户的观点来看,与它进行通信的 USBD 将负责管理所有的已经连接的 USB 设备。

### 10.5.1 概述

USBD 的客户可以向设备发出命令,或者向管道传输数据流或从管理那里接收数据流。USBD 为客户提供了两组软件机制:命令机制和管道机制。

命令机制允许客户对 USBD 操作进行配置和控制,也允许对一个 USB 设备进行配置和通常的控制。更为特殊的是,命令机制提供了对设备缺省管道的所有访问。

管道机制允许一个 USBD 客户来管理某个设备数据和控制传输。管道机制不允许一个客户直接对设备的缺省管道进行寻址。

图 10-5 给出了 USBD 的结构概述。

#### 1. 初始化

某个 USBD 初始化过程是因操作系统而异的。当对 USBD 所管理的某个 USB 进行

初始化时,也会同时建立用于该 USB 的管理信息。这一管理信息的一部分为具有缺省地址的设备及其缺省管道。

一旦一个设备接入了 USB,它会对一个称之为缺省地址的特殊地址作出响应,直到集线器驱动程序为其分配了一个唯一的地址。为了使 USB 系统能和新设备进行交互,用于某个 USB 的具有缺省地址的设备及其缺省管道,在设备接入 USB 后的任一时刻都必须可以被集线器驱动程序所使用。

具有缺省地址的设备及其缺省管道是在一个新 USB 被初始化的过程中由 USB 系统建立的。

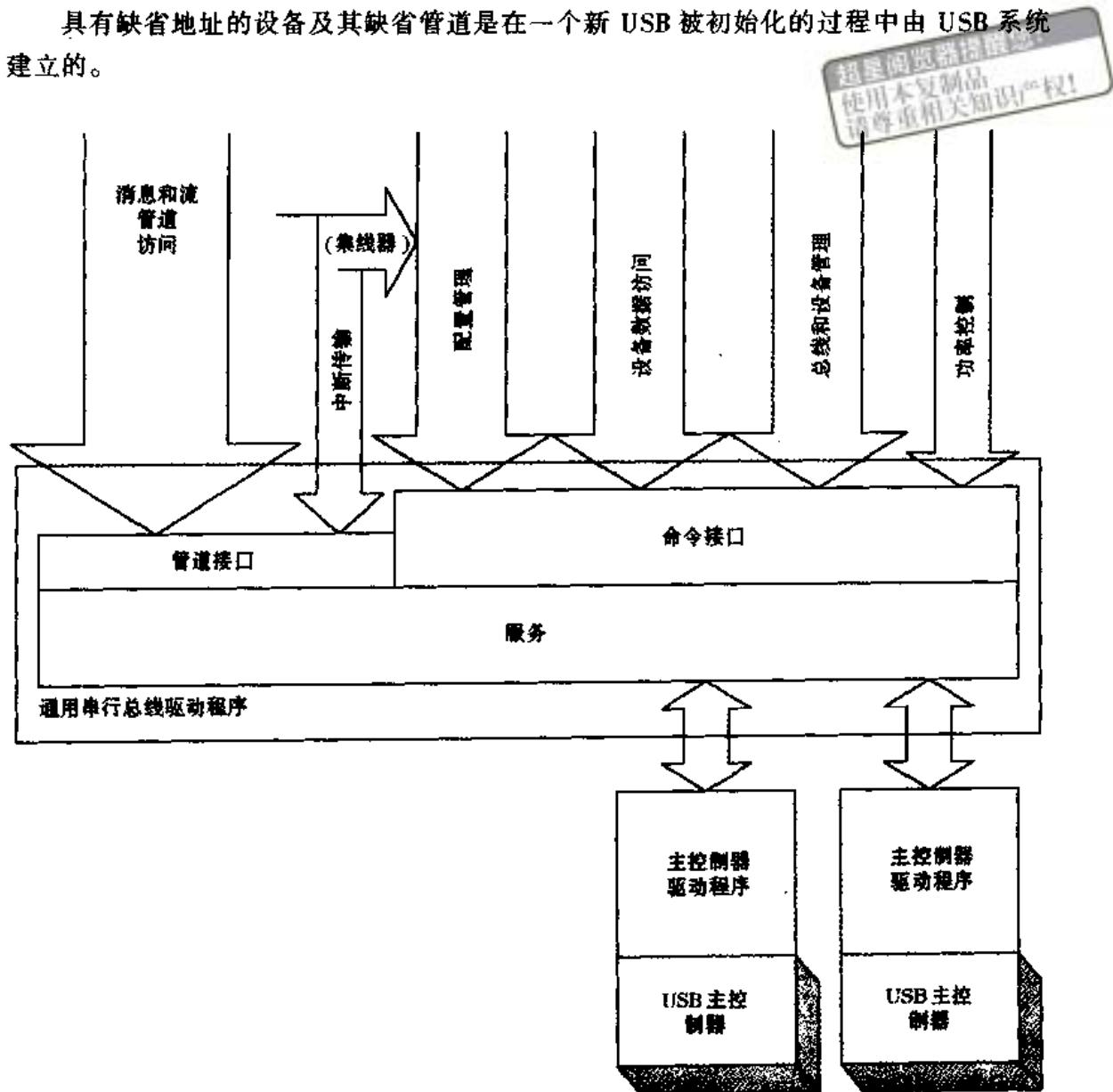


图 10~5 通用串行总线驱动程序结构

## 2. USBD 管道使用

管道是一个设备端点同一个主机软件实体相联系的方法。而管道实际由主机上的一个这样的实体所拥有。尽管无论所有者是谁,一个管道的基本概念都是相同的,但是在两组管道之间还会出现为 USBD 客户提供的功能的某些差别:

- 缺省管道,由 USBD 拥有并进行管理。
- 其它所有管道,由 USBD 的客户拥有并进行管理。

尽管缺省管道经常被用来实现由命令机制来进行中继的用户请求的某些部分,但是它却不能被用户直接访问。

(1) 缺省管道。USBD 负责分配和管理适当的缓冲操作,以支持缺省管道上的传输,而这些操作同设置一个设备地址一样,对于用户而言是不可见的。对于那些用户可以直接见到的传输来说,例如发送供应商和类型命令或读取一个设备描述符,该用户必须提供所需的缓冲作用。

(2) 客户管道。任何一个不为 USBD 所拥有和管理的管道都可以由一个 USBD 客户来拥有和管理。从 USBD 的观点来说,管道只会为单个客户所拥有。实际上,当使用该管道时,只要用户可以像一个协调的组织一样运转,一个合作用户组就可以管理该管道。

用户负责在其可以获得的服务时间内,提供为该管道的数据传输速率服务所需的缓存操作。对工作空间内的其它缓存要求则由 USB 系统来说明。

### 3. 服务能力

USBD 提供了下列类型的服务:

- (1) 通过命令机制进行配置。
- (2) 利用命令和管道机制来提供传输服务。
- (3) 时间指示。
- (4) 状态报告和差错恢复。

#### 10.5.2 USBD 命令机制要求

USBD 命令机制允许对一个 USB 设备的一般用户访问。一般来说,这些命令允许用户对潜在的设备数据和控制空间中的一个进行读或写访问。该客户仅提供一个设备标示符和相关的数据或空的缓冲区指针。

USBD 命令传输不要求对 USB 设备进行配置。USBD 提供的许多设备配置组件都是命令传输。

下面是对所提供的命令机制的专门要求。

##### 1. 接口状态控制

USBD 用户必须可以为任一可置位的管道状态设置一个专用接口。设置一个接口状态会使该接口中的所有管道都进入这一状态。另外,一个接口中的所有管道都可以被复位或禁止。

##### 2. 管道状态控制

USBD 管道状态具有两个组成构件:

- (1) 主机状态。
- (2) 端点反射的状态。

只要报告了管道状态,两个构件的数值都可以识别出来。由端点反射回来的管道状态是处于某个状态的端点的结果。USBD 客户将像对待 USBD 所报告的状态一样,对管道状态进行管理。对于由端点反射回来的任一管道状态,客户也必须同端点进行交互来改变这一状态。

一个 USBD 管道实际上处于下列状态之一：

(1) 有效。已经设置了管道策略，并且该管道可以进行数据传输。客户可以就对于某个管道而言是否有任何未解决的 IRP 进行询问。只要管道可以接收新的 IRP，我们仍然认为没有未解决的 IRP 等待其处理的管道还处于有效状态。

(2) 停止。在该管道上出现了一个错误。这一状态也可以是对设备上相应的被禁止的端点的反映。

(3) 空闲。管道不会再接受 IRP。端点也必须处于空闲状态。

USBD 客户必须能够从上述的任意一种状态，将某个管道设置为空闲或有效状态。客户也必须可以将一个端点设置为有效或空闲状态之一。另外，客户通过下列方式来处理管道状态：

(1) 终止一个管道。为一个管道安排的所有 IRP 被立即撤消，并且向客户返回一个状态来指出它已经被禁止了。主机状态和管道反射回来的状态都不会受影响。

(2) 复位一个管道。管道 IRP 被终止。主机状态变为有效。如果反射回来的端点状态需要被改变，则必须由 USBD 客户明确地给出命令。

### 3. 获得描述符

USBDI 提供一种机制来检索标准设备、配置和字符串描述符以及类或供应商专用描述符。

### 4. 获得当前的配置设置

USBDI 必须能为任何一个指定的设备返回当前的配置描述符。如果设备未进行配置，就不会返回任何配置描述符。这一操作同通过请求某个配置描述符，而返回当前配置的配置描述符等效。但是，它不要求用户知道当前配置的标识符。这样就可以返回所有的配置信息，包括：

- (1) 存储在设备上的所有配置描述符，包括用于所有接口的所有可替换的设置。
- (2) 用于指出有效接口的可替换的设置的指示符。
- (3) 接口处于有效的替换设置时，用于端点的管道句柄。

(4) 接口处于有效的替换设置时，端点的实际 MaxPacketSize。另外，对于某个管道而言，USBDI 必须提供一个工具来返回管道当前正在使用的 MaxPacketSize。

### 5. 添加设备

USBDI 必须为集线器驱动程序提供一个机制来通知 USBD 某一 USB 上添加了一个新设备和检索这一新的 USB 设备的 USBD ID。USBD 的任务包括分配设备地址和准备供使用的缺省管道。

### 6. 拆除设备

USBDI 必须为集线器驱动程序提供一个机制来通知 USBD 某个设备已经被拆除了。

### 7. 管理状态

USBDI 在一个设备、接口或管道的基础上，必须提供一种机制来获得和清除基于设备的状态。

### 8. 发送类型指令

一个用户，最典型的是某个类或自适应驱动程序，可以利用 USBDI 机制来向一个设备发送一个或多个专用的类型指令。

### 9. 发送供应商指令

一个用户可以利用 USBDI 机制来向一个设备发送一个或多个专用的供应商指令。

### 10. 建立替换设置

USBDI 必须提供一个机制来为某个接口改变其可替换的设置。因此，前一个设置的管道句柄会被释放并且会为该接口返回一个新的管道句柄。为了使这一请求可以完成，接口必须处于空闲状态；即接口中的所有管道必须处于空闲状态。

### 11. 创建一个配置

配置软件通过向 USBD 传送一个包含有一个配置描述符的缓冲区来请求一个配置操作。在配置过程中 USBD 会为端点请求资源，而且如果所有的资源请求都得以满足，USBD 会启动设备配置并返回用于所有有效端点的接口句柄和相应的管道句柄。缺省数值可以为用于接口的所有可替换的设置和用于端点的 MaxPacketSize 所使用。随后也可以修改这些缺省数值。

注意：实现某个接口的配置可能需要标识出专门的替换设置。

### 12. 设置描述符

对于支持这一操作的设备而言，USBDI 允许更新现存的描述符和增加新的描述符。

### 13. 为一个管道设立最大分组尺寸

USBDI 必须提供一个机制来改变一个管道传输的特性。USBD 调整所要求的资源，并且每进行一个总线操作都会为指定的管道设置当前的最大数据负载尺寸。这一服务只能应用于一个由创建配置操作所建立的管道和当前处于空闲状态的管道。

## 10.5.3 USBD 管道机制

USBDI 的这一部分可以向用户提供具有可能的最高速率和最低开销的数据传输服务。

通过将某些管道管理任务由 USBD 转移至客户，可以获得更高的性能。因此，管道机制需要在比 USBD 命令机制提供的数据传输服务更基本的层次上实现。管道机制不允许访问一个设备的缺省管道。

只有在设备和 USB 配置都成功地完成之后，USBD 管道传输才可以利用。在设备完成配置的时刻，USBD 将请求所需要的资源以支持配置过程中的所有设备管道。允许客户修改配置，但要受到指定的接口或管道是否处于空闲状态的限制。

客户向输出管道提供填满了的缓冲区，并检索一个请求完成后的传输状态信息。返回的有关一个输出管道的传输信息，允许客户确定该传输是成功了，还是失败了。

客户向输入管道提供空的缓冲区，并检索一个请求完成后来自与输入管道的传输状态信息。返回的有关一个输入管道的传输信息，允许客户确定所接收到的数据的数量和性质。

### 1. 所支持的管道类型

基于四种传输类型，对所支持的管道类型阐述如下：

#### 1) 同步数据传输

要求排队等待一个同步管道的每个缓冲区都可以看作一个样值流。同所有的管道传输一样，客户会为使用该同步管道设立一个策略，包括用于这一客户的相应的服务时间

段。在输入端检测到的字节丢失或遗漏和在输出端所注意到的传输问题都要向用户指出。

客户将对一个起始缓冲区进行排队,以启动一个管道流服务。为了保持在所有的同步传输中所使用的连续流传输模型,在当前的缓冲区退役之前客户对另外一个缓冲区进行排队。

USBD 应该可以将一个客户的数据流视为一个样值流。换而言之,利用客户所指定的同步方法,对数据的精确分组操作由客户隐藏了。另外,在某个客户数据缓冲区中总是完整地保留了一个给定的处理操作。

对于一个输出管道而言,客户提供了一个数据缓冲区。利用客户所选择的同步方式,USBD 可以为服务周期分配跨帧传输的数据。

对于一个中断管道而言,客户必须提供一个空的缓冲区,它应足够存储用户设备在服务周期内可能传输的最大数量的字节。USB 系统会从数据流中去除 USB 定义的封装信息,从而使客户缓冲区内的字节是连续的。在指出了字节遗漏或无效的地方,USBD 会在缓冲区这些字节应该占据的位置留下空格并标识出这一错误。不使用同步方式的结果之一就是保留的空格应假设为最大分组大小。当 IRP 完成后,就会产生一个缓冲区收回指示。注意当返回至客户时,输入缓冲区不应该是填满的。

### 2) 中断传输

中断传输在一个 USB 设备中产生并传递至 USB 驱动程序的一个用户。

用户会对一个足够用来存储中断传输数据的缓冲区进行排队(最典型的是单个 USB 处理操作)。当所有的数据都传输完毕之后或者如果超过了错误阈值,IRP 就会返回至该用户。

### 3) 批量传输

批量传输既可以由设备,也可以由用户来发起。它没有任何周期性或得到保证的时延。

当所有的数据都传输完毕之后或者如果超过了错误阈值,IRP 就会返回至该用户。

### 4) 控制传输

所有的消息管道都是进行双向的数据传输。在所有的情况下,都是由用户向设备端点输出一个建立阶段。可选的数据阶段既可以是一个输入,也可以是一个输出,并且在逻辑上最后一个状态总是又提交给主机。有关规定的消息协议的细节请参考第 8 章。

用户提供了一个用于指定指令阶段或任意一个可选的数据阶段的缓冲区,或者是空的缓冲区空间。当控制传输的所有阶段都完成时,用户会收到一个缓冲区收回指示,或者如果由于传输错误而终止了传输时,它就会收到一个错误指示。

## 2. USBD 管道机制要求

我们提供了下列管道机制。

### 1) 终止 IRP

USBDI 必须允许用于某个管道的 IRP 可以被禁止。

### 2) 对帧开始进行调整

USBDI 必须允许一个主客户来改变一个 USB 帧中所包含的比特数。一个希望对 SOF 进行调整的用户必须已经接收到了一个来自于 USBD 的主客户状态(参考本小节第

5 条内容)。如果激活这一变化的频率大于每 6ms 一次的话, 就会产生未定义的结果。

### 3) 管理管道的策略

USBDI 必须允许一个用户可以为一个单独的管道或整个接口设置或清除策略。在成功地设置一个策略之前, 由客户所产生的任一 IRP 都会被 USBD 丢弃。

### 4) IRP 排队

USBDI 必须允许客户对用于一个给定的管道的 IRP 进行排队。当 IRP 返回至该客户时, 请求状态也会被返回。USBD 提供了一个机制来标识其第一个处理操作都是在同一帧中出现的一组同步 IRP。

### 5) 成为一个主客户

USBDI 必须允许一个客户可以请求成为一个给定的 USB 的主客户, 并且在它不再要求时可以释放该状态。只有主客户才可以为一个给定的 USB 调整 SOF。

一个要求主状态的客户可以通过一个用于其控制的设备的接口句柄来标识自己。

## 10.5.4 利用 USBD 机制来管理 USB

利用所提供的 USBD 机制, 任一 USB 系统都可以支持下列的通用功能。

### 1. 配置服务

配置操作是以每个设备为基础而进行的。USBD 在配置软件的指挥下进行设备配置操作。在设备配置过程中, 一个集线器驱动程序起着一种特殊的作用, 并且至少提供了下列功能:

- (1) 设备连接/拆除识别, 由集线器驱动程序所拥有的一个中断管道来驱动。
- (2) 设备复位, 由集线器驱动程序通过对设备的集线器上行端口进行复位来完成。
- (3) 指挥 USBD 来完成设备地址分配。

另外, USBDI 还提供了下列配置功能, 这些功能可以为主机上的集线器驱动程序或其它可用的配置软件来使用:

- (1) 设备标识和对配置信息的访问(通过访问主机上的描述符)。
- (2) 通过命令机制进行设备配置。

当集线器驱动程序通知 USBD 一个设备已经接入时, USBD 会为这一新的设备建立缺省管道。

### 1) 配置管理

配置管理服务主要是作为一个可以在缺省管道上发起 USB 处理的特定的接口指令集而提供的。值得注意的例外是, 如果使用的另外一个中断管道, 会直接向集线器驱动程序传递集线器状态。

当集线器端口之一出现状态变化时, 每个集线器都会初始化一个中断传输。一般来说, 端口的状态变化是一个下行 USB 设备的接入或拆除。

### 2) 初始设备配置

当一个集线器通过其状态变化管道报告一个新的 USB 设备已经接入时, 设备配置过程就开始了。

配置管理服务允许配置软件从设备中所列的配置集合中挑选出一个 USB 设备配置方式。在启动设备配置之前, USBD 将核实, 在配置过程中向所有的端点提供的数据传输

速率不超过根据当前的进度表所确定的 USB 能力。

### 3) 修改一个设备配置

配置管理服务允许配置软件从设备中所列的配置集合中挑选出另外一个配置方案来取代一个 USB 设备配置。如果在新的配置过程中向所有的端点提供的数据传输速率同根据当前的进度表所确定的 USB 能力向适应,该操作就可以成功完成。如果新的配置被丢弃,会保留前一个配置。

配置管理服务允许配置软件使一个 USB 设备返回到未经配置的状态。

### 4) 设备拆除

当一个集线器通过其状态变化管道报告同一个 USB 设备的通信已经停止时,差错恢复和/或设备拆除处理就会开始。

## 2. 总线和设备管理

总线和设备管理服务允许一个客户成为一个 USB 上的主客户,并且可以允许其作为主客户来调整这一总线上的一帧所包含的比特周期数。一个主客户可以明确地释放主控状态,或者在包含了所引用的接口的设备复位或拆除时自动放弃客户控制状态。在一个 USB 上至多可以有一个主客户。当对 SOF 进行调整时,会给新的主客户一个专用的控制句柄以供使用。

一个主客户可以在当前的 USB 帧中增加或减去一个比特周期。客户必须引用一个接口来标识被调整过的 USB 并允许对客户主控状态加以证实。如果调整 SOF 过于频繁,超过了每 6ms 一次,会产生未定义的结果。

### 3. 功率控制

USB 系统以一个系统特定的方式来提供对功率的管理。对 USB 系统的要求不会超过第 9 章中所描述的那些。

没有可以由所有 USB 设备使用的标准指令。个别的设备可以选择通过供应商特定的控制来提供一个功率管理接口。设备类型可以确定该类型所特有的功率管理功能。

但是所有的 USB 设备都必须支持节电状态(请参考第 9 章)。提供给一个设备的基本功率控制是利用对设备所连接的集线器端口的控制来实现的。

### 4. 事件指示

通过许多来源,USBD 客户可以接收到以下几种事件指示:

(1) 由一个设备指出的操作完成。

(2) 通过流管道的中断传输可以直接将设备事件指示传递给 USBD 客户。例如,集线器使用一个中断管道来传送相当于集线器状态变化之类的设备事件。

(3) 在数据流中由设备所嵌入的事件数据。

(4) 标准设备接口指令、设备类型指令、供应商特定的指令和经过消息管道的一般控制传输都可以由于事件情况而用其来查询设备。

### 5. 状态报告和差错恢复服务

命令和管道机制都可以在其被激活和完成的情况下,应单独的请求而提供状态报告。

另外,利用命令机制能使 USB 设备状态可供 USBD 客户使用。

USBD 通过允许管道被复位或禁用,向客户提供了管道差错恢复机制。

## 10.6 操作系统环境指南

如前面所提到的一样,USB 软件和主机软件之间存在的实际接口是某个主机平台和操作系统专用的。每一个支持 USB 的平台和操作系统的组合都需要有一个规范手册。这些规范描述了用来将 USB 集成到主机的专用接口。每一个支持 USB 软件的操作系统提供商都应标识出一个其兼容的通用 USB 规范版本。

## 第 11 章 集线器规范

本章将介绍 USB 集线器的体系结构要述：集线器中继器和集线器控制器。另外本重新开始的集线器操作。本章的第二部分则器描述符。

在集线器规范中给出了实现集线器所以开发出完全符合 USB 规范的 USB 集线器

本章具体包括以下内容：

在第 1 章中，我们已经提器。在 USB 系统中，集线器用



设备特性；



集线器 I/O 缓冲区要求；



集线器故障恢复机制；

器在所有 USB 外设中处在核心的位置。目前,只有利用集线器才可以实现 USB 规范 1.0 所支持的 127 个 USB 设备。这是因为通常情况下,一个主机只提供了两个 USB 接口(由根集线器来提供),另外还涉及对下行 USB 设备进行供电等问题。所以了解 USB 集线器的工作原理和其必须遵循的原则对于希望深入了解 USB 规范,并利用该总线技术来开发其专用产品的开发人员来说,是非常重要的。

下面我们就来详细地讨论 USB 集线器。

## 11.1 概 述

集线器在主机和 USB 设备之间提供了一个电气接口,并且由其直接负责支持许多使 USB 用户使用起来更友好的属性,以及对用户隐藏 USB 协议的复杂性。下面列出了 USB 集线器所必须支持的主要的 USB 功能:

- (1) 连接性能;
- (2) 电源管理;
- (3) 设备连接/拆除检测;
- (4) 总线故障检测和恢复;
- (5) 支持全速率/低速率设备。

一个集线器包括两个主要的元件:集线器中继器和集线器控制器。集线器中继器负责连接的建立和拆除。同时它也支持像总线故障检测和恢复以及设备连接/拆除检测等额外的控制操作。而集线器控制器为主机到集线器的通信提供了一种机制。另外,集线器特定的状态和控制命令允许主机对一个集线器进行配置,并监视和控制该集线器单独的下行端口。

## 11.2 设 备 特 性

### 11.2.1 集线器体系结构

图11-1说明了一个集线器以及其根端口和下行端口的位置。一个集线器包括一个

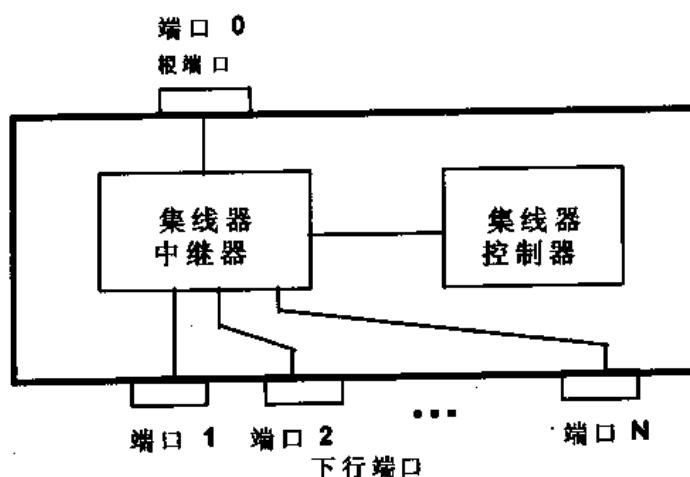


图 11-1 集线器体系结构

集线器中继器部分和一个集线器控制器部分。集线器中继器部分负责基于每个分组来管理连接的建立和拆除,而集线器控制器部分则提供了状态和控制并允许主机来对集线器进行访问。

### 11.2.2 集线器连接

根据其是否传播分组通信或重新开始信号,或它是否处在空闲状态,集线器可以表现出不同的连接性能。

#### 1. 分组信号连接

集线器中继器包含了一个必须总是连接上行方向的端口(就是我们所说的根端口)和一个或多个下行端口。所定义的上行连接是指向主机的,而下行连接则是指向一个 USB 设备的。图 11-2 说明了在上行和下行方向用于集线器的分组信号连接行为。一个集线器在其没有进行连接时,也可以进入一个空闲状态。当集线器处于空闲状态时,所有的端口都处于接收状态,等待下一个分组的开始。

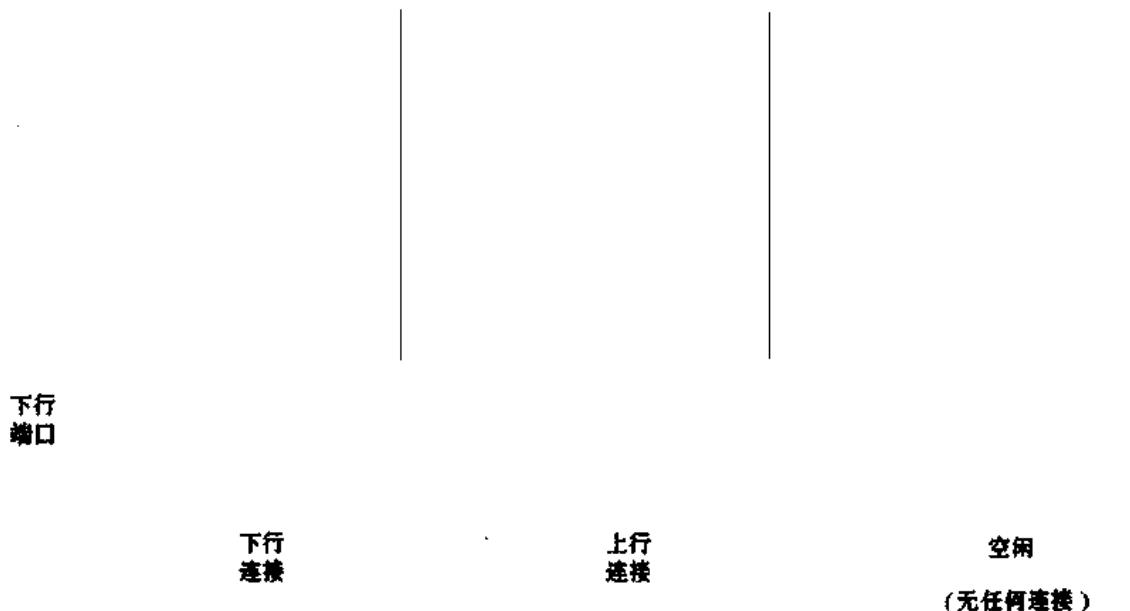


图 11-2 集线器连接

如果一个下行集线器端口的状态),并且集线器在该端口向该集线器根端口的连接,但是设备或一个集线器向上传送一能见到该分组。而当集线器在会被锁住。这样就可以保证集帧结束时集线器超时。

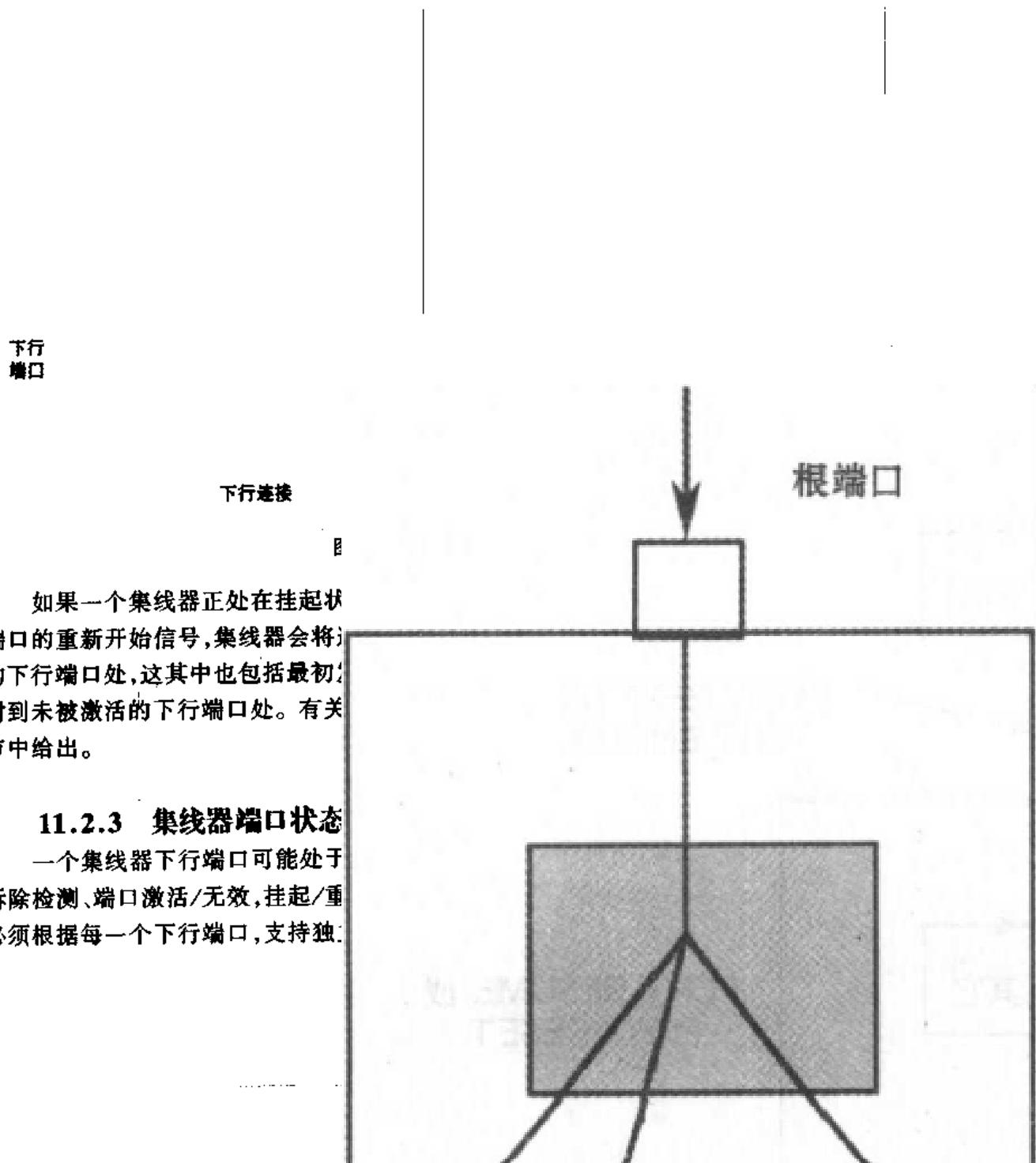
传播信  
立一个  
着当一  
集线器  
有端口  
直到在

在下行方向,集线器采用了广播方式。当一个集线器在其根端口处检测到了一个SOP时,它会为所有被激活的下行端口建立连接。如果一个端口未被激活的话,那么它就不会收到来自于根端口的任何分组传输活动,并且这一端口也不会向下行方向继续传播该分组。

## 2. 重新开始连接

集线器对于指向上行方向和下行方向的重新开始信号采用了不同的连接操作。一个处于挂起状态的集线器,会将重新开始信号从它的根端口反射回其所有被激活的下行端口处。图 11-3 说明了集线器上行和下行重新开始连接是如何进行的。从图中我们可以清楚地看到两者的差别。

赵星原创  
使用本复制品  
请尊重相关知识产权!



源开关的集线器的状态,而图 11-5 则说明了用于一个实现了电源开关的集线器的状态。集线器状态仅适用于下行端口。

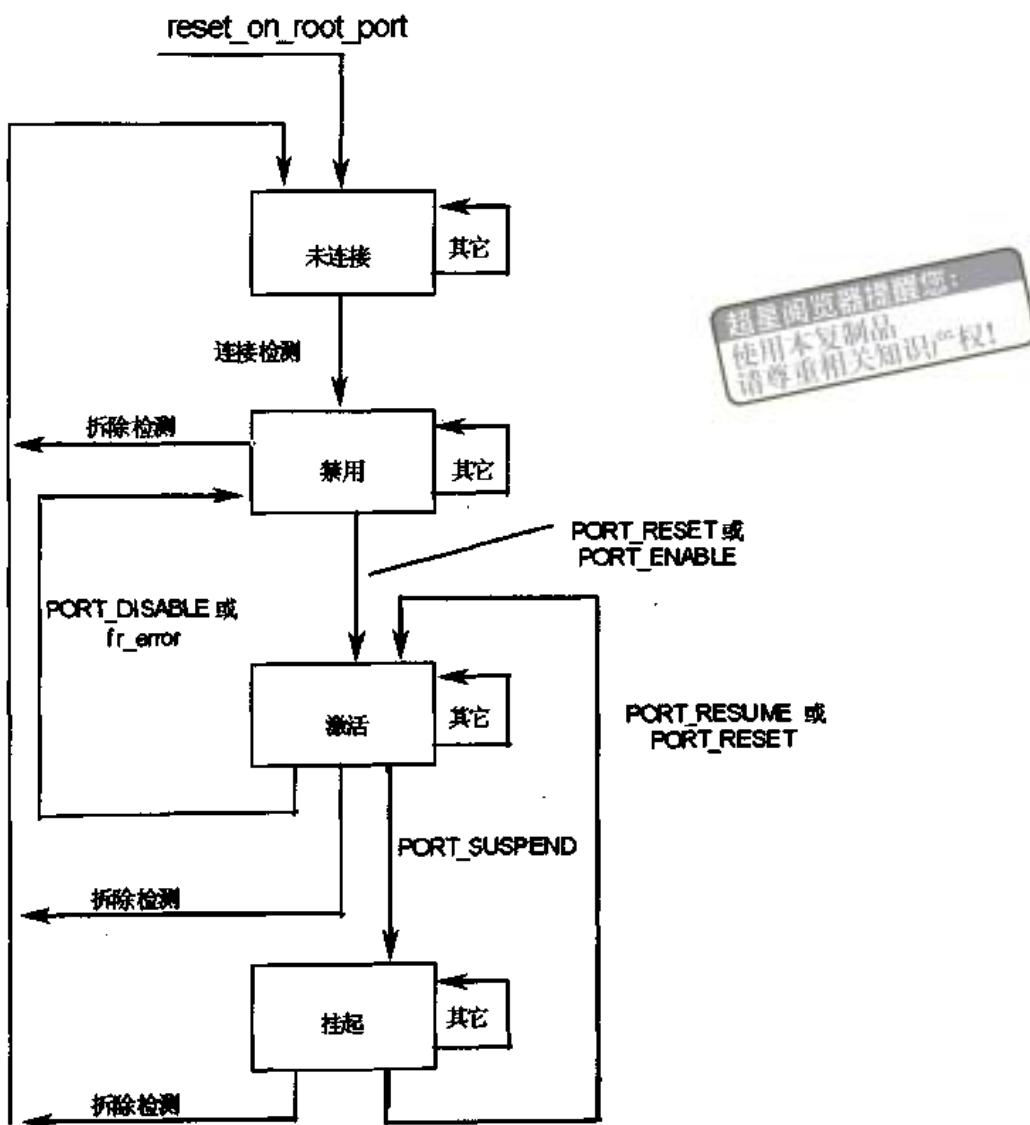


图 11-4 无电源开关的集线器端口状态

每一个状态都可以描述如下：

(1) 掉电(Power Off):只有那些如图 11-5 所示,具有电源开关的端口才能支持这一状态。当集线器收到了一个 ClearPortFeature(PORT\_POWER)请求时,或集线器检测到了根端口复位信号时,一个端口才可以变换为它的掉电状态。一个集线器下行端口在刚把电源加至集线器时,也会进入掉电状态。一个处于掉电状态的端口不会向下提供任何电源,并且它的信号输出缓冲器会被置于 Hi-Z 状态。另外,一个处于掉电状态的端口会忽略在这一端口上出现的所有指向上行方向的总线活动。

(2) 未连接(Disconnected):在利用一个 SetPortFeature(PORT\_POWER)请求而把电源加至集线器时,支持电源开关的集线器的一个下行端口就会从掉电状态转变为未连接状态。如果一个集线器不支持对电源进行开关操作,那么它就会在刚一上电时或接收

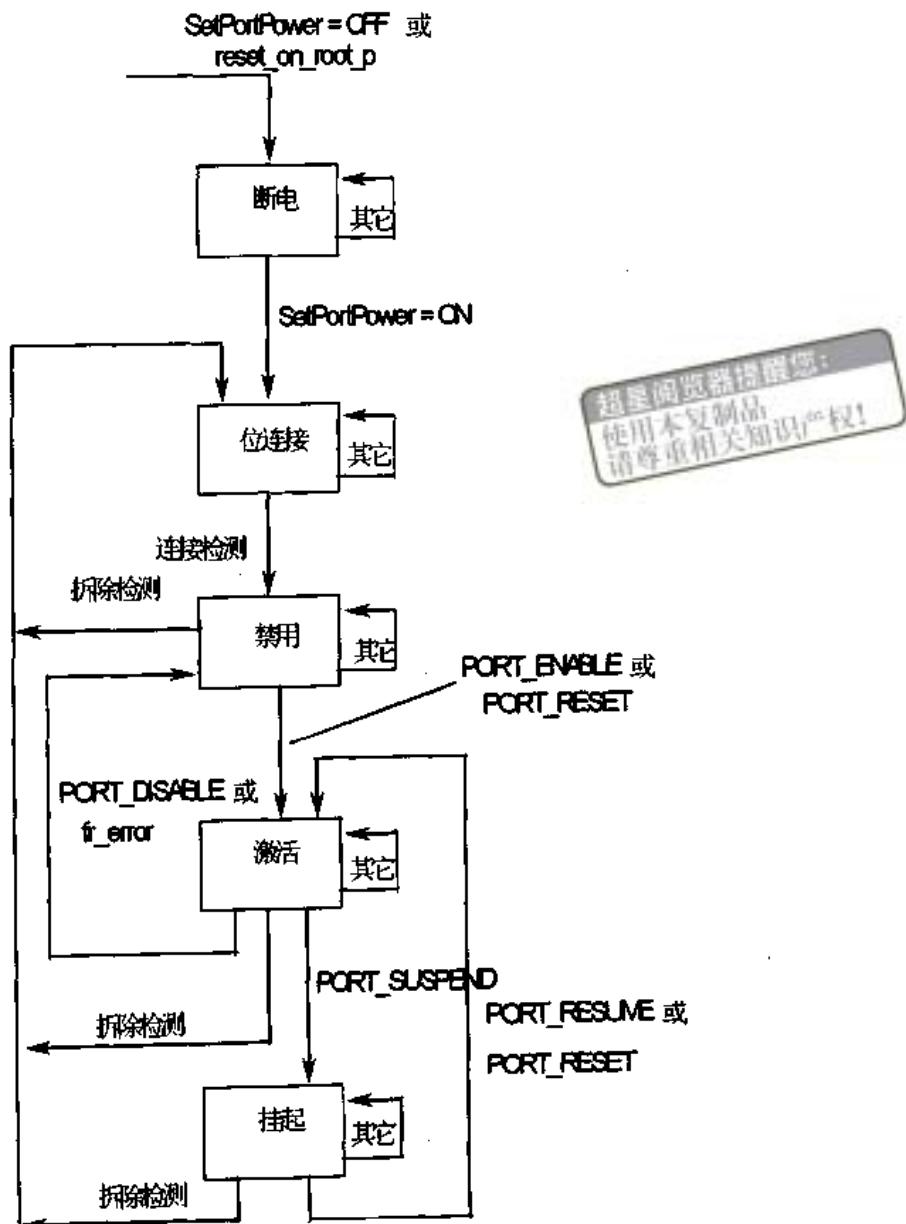


图 11-5 有电源开关的集线器端口状态

到一个根端口复位信号时就进入拆除状态。在未连接状态，一个端口就不能够向上行方向或下行方向传播任何信号。但是，该端口却可以检测到一个连接事件(`conn_det`)，这一事件可以设置在集线器控制器中的一个状态域，并会使该端口的状态转变为未激活的状态。

如果一个下行端口处于未连接状态并且在总线上检测到了一个连续  $2.5\mu s$  的非 SE0 信号，它就会声明它检测到了一个连接事件(`conn_det`)。当第一次声明一个 `conn_det` 时，总线空闲的状态就会由一个低速率或全速率设备来进行驱动，并可能处于一个 DIFF 0 或 DIFF 1 状态。通过确定是 D+ 或是 D- 信号线被拉至高电平，该集线器可以自动确定这个设备的类型(低速率或全速率 USB 设备)。在集线器变换至未被激活的状态之前，

确定设备工作速率的操作就要完成。

(3) 禁用(disabled):一个端口当其检测到一个连接事件(conn\_det)时,它就会从未连接状态转变为禁用状态。如果这个集线器支持电源开关,这就首先要求该端口的电源开关必须闭合。一个处于禁用状态的端口只能传播由一个SetPortFeature(RESET)请求所发起的指向下行方向的信号重新开始。在所有其它的时刻,该端口的输出缓冲区都是处在Hi-Z状态。而在上行方向,当集线器苏醒时,一个处于禁用状态的端口不可能通过集线器向根端口传播任何信号。但是,某些事件,例如拆除操作就会在集线器处于挂起状态时而使上行的重新开始信号传播至根端口。一个拆除事件(disc\_det)会使这一端口返回到未连接状态并将集线器控制器内的一个状态域置位。当该端口没有传送下行通信量时,只要它检测到了一个持续 $2.5\mu s$ 的SE0,就会声明检测到了一个拆除事件(disc\_det)。在安排一个拆除事件之前,被集线器挂起的集线器必须首先被唤醒。

(4) 激活(enabled):一旦一个端口收到了一个SetPortFeature(PORT\_ENABLE)或SetPortFeature(PORT\_RESET)请求之后,它就会转移到激活状态。在激活状态,该端口可以传送所有的下行信号,全速率分组通信流和复位信号重新开始而低速率分组通信流在其前面具有预同步PID时也可以向下行方向传送。在激活状态,该端口可以传播所有的上行信号,包括全速率和低速率分组以及复位信号。当一个端口接到了一个ClearPortFeature(PORT\_ENABLE)请求或出现了一个帧差错(fr\_error)时,它就会进入禁用状态。ClearPortFeature(PORT\_ENABLE)请求可以由主机在任何时刻发出,而且集线器必须立即将该端口置为禁用状态来作为对PORT\_ENABLE请求的响应。如果一个拆除检测产生后,这个端口就应该转移到未连接状态。

(5) 挂起(suspended):当一个集线器接收到了一个SetPortFeature(PORT\_SUSPEND)请求之后,该集线器就可以有选择地挂起在一个端口下行方向的所有USB设备。在一个处理操作正在进行当中,该请求必须不能使当前要被挂起的端口停止传输重新开始,即只有在当前的处理操作结束之后,该端口才能进入挂起状态。根据一个集线器是否被唤醒或处于自我挂起状态,一个端口可以表现出不同的挂起连接操作。然而,如果集线器处于挂起状态,在端口上出现的一个空闲(idle)至SE0或一个空闲(idle)至重新开始(resume)的变化都会作为一个空闲(idle)至重新开始的变化而反射到其它所有非禁用的端口。这一操作可以允许挂起一系列的集线器,而且还可以使处于最低层的一个USB设备唤醒整个USB总线。

如果一个集线器被挂起,并且在一个挂起的端口上出现了总线活动,该集线器就会首先被唤醒。指向端口的一个重新开始请求结束后可以使集线器内的状态域置位。作为响应,主机会查询该集线器来读取状态域的变化和确定究竟在哪一个端口上产生了重新开始信号。当重新开始信号结束之后,该集线器端口就会转移至激活状态。有关选择挂起的集线器信号的细节将在11.5.2节中加以描述。

一个被挂起的集线器上出现的一个拆除事件会使集线器转移至未连接的状态,并会设置集线器控制器的状态域来指明一个拆除操作发生了。由于该端口永远不会退出未连接的状态,所以不可能直接把一个处于未连接状态的集线器放置到挂起模式。

表11-1总结了在不同的端口状态下,对于不同类型的信号集线器端口所采取的操作。当集线器自身也处于挂起状态时,在重新开始信号期间集线器的行为是一种特殊的

情况,我们将在 11.5.2 节中予以介绍。

表 11-1 端口对信号的动作

信号\状态	断电	未连接	禁用	激活	挂起
在根端口上复位(集线器具有电源开关)	停留在断电状态	进入断电状态	进入断电状态	进入断电状态	进入断电状态
在根端口上复位(集线器不具有电源开关)	不予应答	进入未连接状态	进入未连接状态	进入未连接状态	进入未连接状态
ClearPortFeature (PORT_POWER)(集线器具有电源开关)	停留在断电状态	进入断电状态	进入断电状态	进入断电状态	进入断电状态
SetPortFeature (PORT_POWER)(集线器具有电源开关)	进入未连接状态	不予应答	不予应答	不予应答	不予应答
SetPortFeature (PORT_RESET) 请求	停留在断电状态	进入激活状态	进入激活状态	停留在激活状态	进入激活状态
SetPortFeature (PORT_ENABLE) 请求	忽略	忽略	进入激活状态	停留在激活状态	忽略
ClearPortFeature (PORT_ENABLE) 请求	忽略	忽略	停留在禁用状态	进入禁用状态	忽略
下行分组通信流量(集线器启动)	不进行传播	不进行传播	不进行传播	传送通信流量	不进行传播
上行分组通信流量(集线器启动)	不进行传播	不进行传播	不进行传播	传送通信流量	设置状态域, 不进行传播
SetPortFeature (PORT_SUSPEND) 请求	忽略	忽略	忽略	进入挂起状态	忽略
ClearPortFeature (PORT_SUSPEND) 请求	忽略	忽略	忽略	忽略	进入重新开始状态
拆除检测	忽略	忽略	进入未连接状态	进入未连接状态	进入未连接状态
连接事件	忽略	进入禁用状态	不予应答	不予应答	不予应答

通过在一个下行端口上检测一个至少持续  $2.5\mu s$  的 SEO, 集线器就可以检测到一个拆除事件。作为对一个拆除事件的响应, 该集线器会把它的端口放置到未连接的状态并把其输出缓冲区浮动到 Hi-Z 状态。只有当总线上没有下行通信流量时, 才可以检测到设备拆除。如果从一个端口上移走了电源, 它必须通过记录一个拆除事件来作为响应并使该端口处于掉电状态。这样可以保证在一个设备拆除而有一个新的 USB 设备接入时, 接入 USB 设备的事件可以得到应答。

#### 11.2.4 总线状态鉴定

总线状态鉴定操作是在一帧的末尾进行的, 并且可以区别出 SEO, 差模 1 和差模 0 状

态。当没有设备接至一个下行集线器端口时,它的下拉电阻会将 D+ 和 D- 都拉至 V<sub>SE(min)</sub> 以下。

连接/拆除操作检测只有在 V<sub>bus</sub> 加至下行端口后才能进行(这一要求仅对那些下行端口实现了电源开关的集线器有影响)。当一个设备连接之后,总线状态就会从未连接状态转移到连接操作检测状态。低速率设备会把 D- 拉高至 SE1 并使 D+ 停留在 SE0。而全速率设备会把 D+ 拉高至 SE1 并使 D- 停留在 SE0。一旦一个 USB 设备接入之后,每一个下行端口必须能够检测和区别出低速率和全速率设备连接。在一个设备接入并确定了该设备的速率之前,差模 J 和 K 状态是不明确的。

当一个连接或拆除操作发生之后,在该事件发生那一帧的末尾,该事件必须反映在集线器的状态中,除非集线器处于复位或挂起模式。处于挂起状态的集线器可以由一个连接或拆除事件所唤醒,并且必须能够在重新开始操作结束时报告该事件。一旦退出了复位模式,集线器就必须检测究竟是哪一个下行端口接有 USB 设备。连接和拆除变化是根据每个端口为基础而进行报告的。

### 11.2.5 全速率和低速率行为比较

当一个设备连接至 USB 总线或在其上电之后,集线器必须能够识别出全速率设备和低速率设备。而当一个集线器端口由其未连接状态转移至其禁用状态之后,集线器就可以检测出这一个设备是全速率设备或是低速率设备。通过检测出哪一根信号线(D+ 或 D-)被拉至高电平,就可以确定接入一个集线器的设备是全速率设备还是低速率设备。低速率设备会将 D- 信号线拉至高电平;而全速率设备则将 D+ 信号线拉至高电平。必须禁止将全速率信号传送到低速率设备。如果这样做,就会使低速率设备对全速率信号作出错误的响应并引起一个总线冲突。主机和集线器控制器之间进行的通信总是利用全速率信号来实现。

如果检测出一个设备为低速率设备,那么该集线器端口的输出缓冲区就会被配置成工作在慢转变速率(75 ~ 300 ns),而且该端口不会传播任何指向下行方向的通信,除非其前面有一个预同步 PID。低速率信号紧跟在 PID 之后,并传播至所有的全速率和低速率设备。由于没有任何一个低速率数据方式可以产生出一个有效的全速率 PID,所以全速率设备从来不会错误地解释低速率通信量。当启用低速率信号时,一个集线器会持续地向下行方向发送信号至所有的被激活的端口,直到检测到了一个下行的 EOP。此时,用于低速率端口的输出驱动器会被关闭,并且直到该集线器接收到了另外一个预同步 PID 之后才能被重新打开。如果一个端口被禁用,就不会有任何信号传播至该端口。集线器必须在接收预同步 PID 的最后一个比特的四个全速率比特周期内激活它的低速率端口驱动器,而此时它还必须将一个低速率 J 信号驱动到总线上。

如果一个下行端口被启用之后,无论该端口是否被配置成按低速率或全速率方式来工作,它都会传播指向上行方向的总线信号。集线器只在其下行端口上的下行方向上实现了可供选择的转变速率输出缓冲区;在上行方向,它会利用快速(4 ~ 20 ns)的边沿变化率来传送低速率和全速率信号。低速率设备不会将一个预同步 PID 添加到它的上行通信流当中。

在传送下行低速率通信流时,集线器必须能够对两个低速率比特周期宽的 EOP 或两

个全速率比特周期宽的 EOP 都作出适当的响应。前者在低速率设备产生 EOP 的正常操作中产生。而后者在一个下行集线器遇到了一个帧结束串扰或 LOA 条件时产生，并且会产生一个上行的 EOP 作为响应(参见 11.4.5 节)。

所有接有低速率设备的集线器端口必须能够产生一个低速率保持选通脉冲，该选通脉冲在一帧开始时产生，它包括一个两个低速率比特周期的 SOF 并跟有一个至少为 0.5 比特周期的 J 状态。低速率设备要利用该选通脉冲来阻止其在缺乏低速率总线通信流量时进入挂起状态。集线器中继器由它内部的 SOF 计数器来产生这一保持选通脉冲，而且该保持选通脉冲开始时必须不早于第二个 EOF 点，并且其完成的时刻必须不迟于令牌分组的 EOP。

低速率保持选通脉冲必须每帧产生一次，而且它必须跟踪 SOF 令牌从而可以满足下列原则。

- (1) 当集线器进入挂起状态时，低速率保持选通脉冲必须保证在最后一个 SOF 之前不能够送出。
- (2) 在接收到最后一个 SOF 之后，允许一个集线器在仅仅三个低速率保持选通脉冲内进行同步。
- (3) 低速率保持选通脉冲必须在不迟于恢复 SOF 之后的那一帧产生。

### 11.2.6 集线器状态操作

集线器状态操作如图 11-6 所示。一旦集线器退出复位状态或上电之后，该集线器就会从 WFSOP 状态开始其工作。此时集线器将等待在其根端口或任一激活的下行端口

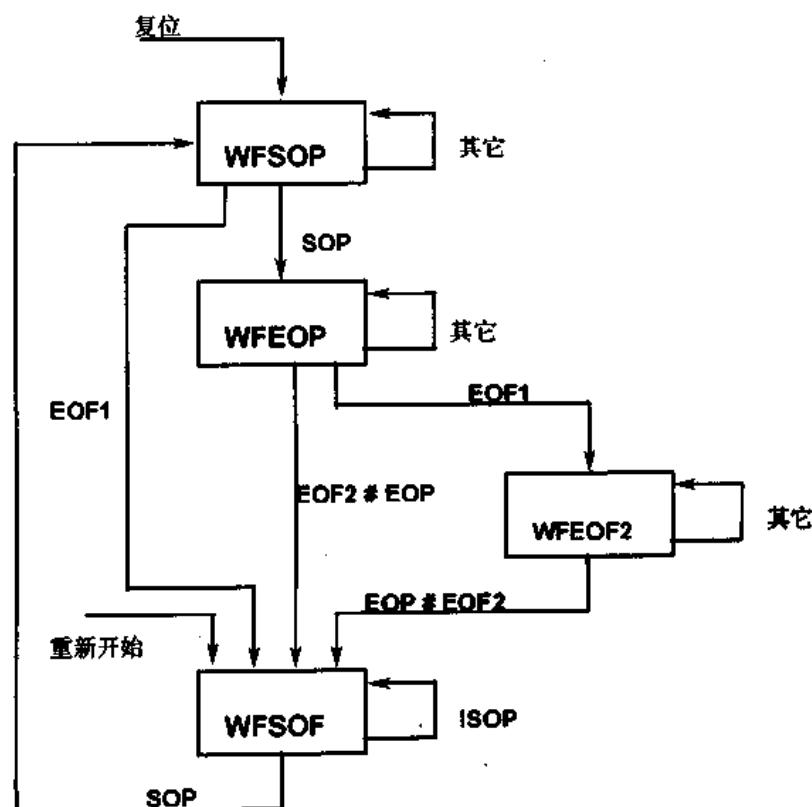


图 11-6 集线器中继器状态

上检测到一个分组开始(SOP)信号。如果检测到了一个 SOP 信号,该集线器就会建立一个来自于产生 SOP 的那个端口的连接,并且转移至 WFSOP 状态。集线器会一直停留在这一状态,直到它遇到了一个分组结束(EOP)或产生了一个帧结束信号。在正常的环境下并且不接近一帧的结尾时,一个集线器中继器就会在 WFSOP 和 WFEOP 两个状态之间来回转移。

处于空闲(WFSOP)状态的集线器通过变换为 WFSOF 状态来响应帧结束(EOF1)点。如果在 EOF1 点该集线器处于 WFSOP 状态,它就会变换为 WFSOF 状态。由 WFSOP 和 WFEOP 至 WFSOF 状态的变换不是出现了一个错误,而仅仅是指出该集线器已经接近了一帧的末尾,并且直到下一帧开始时它才能建立连接。

WFEOF2 是一个特殊的状态,只有在接近一帧的末尾时检测到了一个串扰或总线活性损失(LOA)并且已经建立了一个上行连接之后,集线器才会进入该状态。如果在遇到了一个 EOF1 点时,一个集线器中继器仍然处于 WFEOP 状态(即它没有接收到一个 EOP),该集线器就会转变为 WFEOF2 状态。集线器会一直处于该状态,直到产生了它的 EOF2 点或一个上行 EOP 信号,此时它就会转换为 WFSOF 状态并等待下一个下行 SOP(DSOP)。通常情况下,这个 SOP 还带有一个 SOF 分组以指出下一帧的开始。当产生了一个 DSOP 之后,集线器就会返回到 WFEOP 状态并等待该分组的结束。如果出现了一帧的结束标志并且集线器仍然维持了下行连接,那么该集线器就不可能退出 WFEOP 状态;相反它会等待下一个下行 EOP(一般是 SOF 令牌分组的 EOP),该分组可以使这一集线器进入 WFSOP 状态。

如果一个集线器在 EOF2 出现时仍然处在 WFEOF2 状态,无论总线状态如何,建立了上行连接的端口必须被禁用。当出现了 EOF2 时,一个集线器正处于 WFSOF 状态并正在驱动上行连接,而且它的总线正处于空闲状态,那么先前连接的端口状态必须保持不变。如果该集线器在 EOF2 点正处于 WFSOF 状态并驱动了上行连接,但是总线没有处于空闲状态,那么该端口就会被禁用。

一旦一个集线器退出了重新开始状态,它就会转变为 WFSOF 状态。这样做可以保证在集线器帧计数器至少检测到了一个 SOF 之前,不会向上行方向(并且很可能锁住总线)传播任何通信流。有关帧计数器和集线器中继器状态机之间所进行的互连的有关细节,我们将在 11.5.1 节介绍。

集线器中继器可以在跨越集线器所检测并再生的每一个分组时保持其状态不变。集线器中继器状态所跟踪的分组不必超过一个单一的分组,并且不必在一个处理中跨越多个分组进行跟踪。图 11-7 说明了在一个正常的分组传输过程中,集线器的状态是如何进行变化的。

对于上行连接而言,一个集线器中继器会在四种状态之间变化:等待分组开始(WFSOP)、等待分组结束(WFEOP)、等待 EOF2 点(WFEOF2)和等待帧开始(WFSOF)。EOF1 和 EOF2 点将在 11.4.5 节予以讨论。下面我们将讨论这四种状态。对于下行连接而言,集线器将在 WFSOP、WFEOP 和 WFSOF 之间变化。

### 1. 等待分组开始(WFSOP)

等待分组开始(WFSOP)是当前没有任何分组通过该集线器传播或传送至该集线器时,一个集线器所占据的状态。集线器一旦脱离了重新开始状态就会变换为 WFSOP 状态。

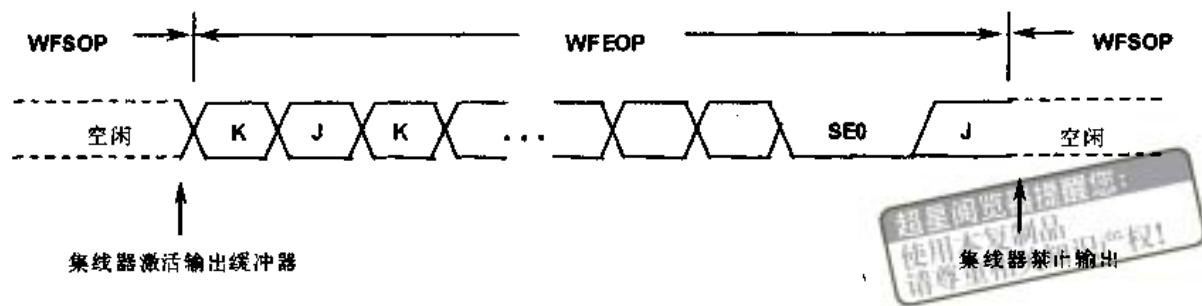


图 11-7 跨越一个分组的集线器状态

态。在集线器处于 WFSOP 状态时,一个集线器的所有端口都要处于高阻状态,而所有被激活的端口则都会处于接收模式并将其输出缓冲区置于 Hi-Z 状态。如果根端口或任何一个被激活的端口检测到了一个 SOP,该集线器就会建立连接并进入等待分组结束(WFEOP)的状态。

#### 2. 等待分组结束(WFEOP)

在集线器处在等待分组开始(WFSOP)状态期间,集线器会建立它的连接并在它的一个端口上接收分组通信流。集线器会在上行方向或下行方向透明地传送该通信流。连接会一直保持到集线器脱离这一状态。当一个集线器检测到了一个 EOP 或如果它遇到了一个帧结束(EOF1)点(参见 11.4.4 节)时,该集线器就会退出 WFEOP 状态。EOP 检测会使该集线器变回到 WFSOP 状态,这是一个标准的操作顺序。如果检测到了一个 EOF1 信号,该集线器就会转变为 WFEOF2 状态。

#### 3. 等待 EOF2 点(WFEOF2)

只有当一个集线器检测到了它的 EOF1 点,并且它仍然在等待一个来自于一个下行端口的 EOP 信号,它才能进入 WFEOF2 状态。这一情况暗示着出现了串扰或总线下行损失(LOA)。一个集线器中继器在检测到了一个 EOP 信号或出现了它的 EOF2 点之前,会一直保持在 WFEOF2 状态。

#### 4. 等待帧开始(WFSOF)

当检测到了一个 EOF1 并且集线器正处在 WFSOP 状态(帧操作通常的结束位置),或者当该集线器处在 WFEOF2 状态并且检测到了一个 EOP 或一个 EOF2 信号时,一个集线器中继器就会进入等待帧开始(WFSOF)状态。

### 11.3 集线器 I/O 缓冲区要求

所有的集线器端口必须能够产生并检测到 J、K 和 SEO 总线信号状态。这就要求该集线器端口可以独立地监视它们的 D+ 和 D- 信号线的输出。每一个集线器端口在 D+ 和 D- 信号线上都要有一个单端接收器,同时还要有一个差模接收器。有关电压所处的电平和驱动要求的详细讨论在第七章中已经提到了。图 11-8 给出了一个典型的集线器端口的 I/O 电路。

表 11-2 列出了集线器 I/O 部分输入和输出的信号。

D+ 和 D- 数据线是连接至 USB 物理媒质的 I/O 线。当把它们置为 Hi-Z 状态时,它们会被集线器和设备上的电阻拉至接近地电位或 Vcc。RxD 是利用差模接收的数据。

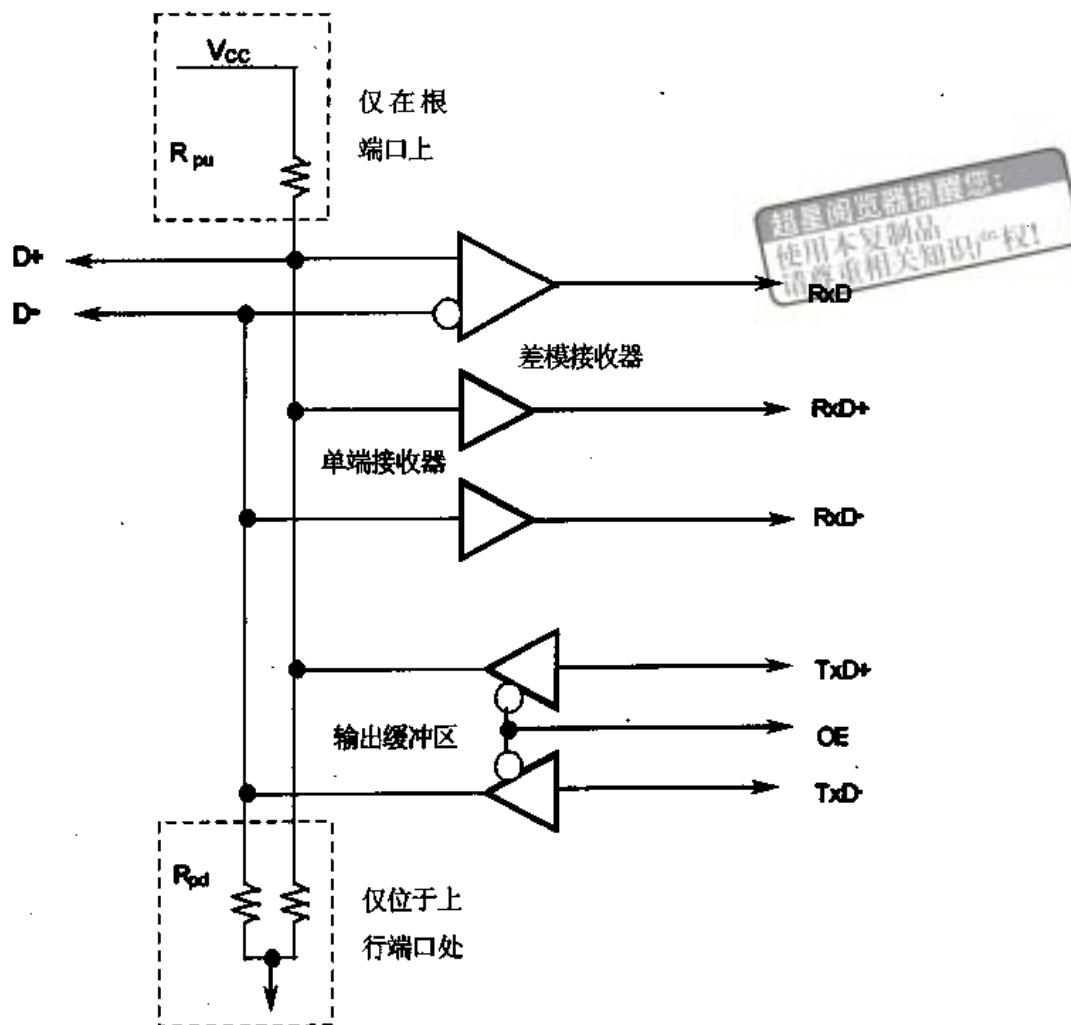


图 11-8 集线器端口 I/O 驱动器和接收器

RxD+ 和 RxD- 是所接收到的单端数据。TxD+ 和 TxD- 则用于发送差模数据和单端复位和 EOP 信号。OE 会禁用输出驱动器。

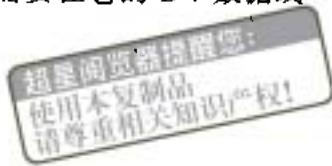
表 11-2 集线器 I/O 部分信号

信号名称	方向	说 明
D+, D-	I/O	外部 USB 数据线
RxD	O	接收的差模信号
RxD+	O	在 D+ 数据线上接收的单端信号值
RxD-	O	在 D- 数据线上接收的单端信号值
TxD+	I	传输的数据值
TxD-	I	传输的数据值
OE	I	用于输出缓冲区的输出激活/禁用信号

### 11.3.1 上拉和下拉电阻

集线器和它所连接的设备利用上拉和下拉电阻的组合，在缺乏对 D+ 和 D- 数据线

的正确驱动时实现对其的控制。这些电阻可以建立用于表示连接和拆除操作所需的电平,而且可以在数据线缺乏正确驱动时使其保持在空闲状态。每一个集线器下行端口都需要在每根数据线上连接一个下拉电阻( $R_{pu}$ );而集线器根端口则需要在它的 D+ 数据线上连接一个上拉电阻。 $R_{pu}$ 和  $R_{pd}$  电阻的值在第 7 章中有述。



### 11.3.2 边沿变化率控制

下行集线器端口必须能够同时支持低速率和全速率边沿变化率。全速率信号规定了 4~20 ns 的一个上升/下降时间。而低速率上升/下降时间必须位于 75~300ns 的范围之内。根据检测到的一个下行设备是全速率或是低速率设备,一个下行端口上的边沿变化速率必须是可选的。集线器根端口总是使用全速率信号,而且它的输出缓冲区总是工作在全速率边沿变化率。

## 11.4 集线器故障恢复机制

集线器是在主机和其它设备之间实现连接的关键 USB 元件。对于任何连接故障,尤其是那些可能造成死锁的故障,重要的是要能检测出它们并可以阻止其发生。只有当集线器处于中继器模式时,它才需要控制连接故障。集线器也必须能够检测并从寻址到集线器控制器的分组丢失或损坏情况下恢复过来。由于实际上集线器控制器是另外一个 USB 设备,所以它必须遵循和我们在第 8 章中所提到的 USB 设备必须遵循的同样的超时原则。有关细节请参考第 8 章。

### 11.4.1 集线器控制器故障恢复

一个集线器控制器可以对分组传输和破坏进行响应,并从中恢复出来。这其中包括了丢失或被破坏的令牌、数据和握手分组。表 11-3 列出了集线器控制器可能检测出来的域水平的错误和它的响应。

表 11-3 分组差错类型

域	差 错	动 作
PID	PID 校验, 比特填充	忽略分组
地址	比特填充, 地址 CRC	忽略令牌
数据	比特填充, 数据 CRC	丢弃数据

### 11.4.2 假 EOP

集线器根据其是作为一个中继器来工作,还是作为一个 USB 设备而被访问,它对假 EOP 的控制还会有所差别。作为一个中继器来工作的集线器透明地传播信号,并且不可能区别出一个“好”EOP 或一个“假”EOP。如果出现了任何一个 EOP,该集线器都会拆除连接并等待下一个 SOP 的出现。如果分组发送器继续发送分组,集线器会在下一个 J 至

K 变化时重新建立连接。从一个集线器的角度来看,一个假 EOP 会使一个单一的分组看起来像是两个分离的分组。当其工作在中继器模式时,该集线器不会参与假 EOP 差错检测或恢复过程。

当一个集线器作为一个 USB 设备而被访问时,集线器控制器可以像我们在 8.7.3 节中提到其它任意一个 USB 设备一样,检测出假 EOP 信号并且可以从假 EOP 信号中的错误中恢复出来。

使用本教材  
请尊重相关知识产权

### 11.4.3 中继器故障恢复

集线器应该可以检测出并从使其无限期地等待一个分组结束(EOP)的条件或在一帧结束时总线处在空闲状态之外的某个状态(即一个没有 EOP 的 SOP)中恢复过来。总共有两种集线器故障情况:活性损失(LOA)和串扰。总线活性损失(LOA)的特征是在所检测到的一个分组开始(SOP)之后缺乏总线活动(总线停留在空闲或 K 状态),和在一帧结束时没有分组结束(EOP)信号。串扰则以 SOP 后出现跨过一帧的终点而继续进行的总线活动为特征。集线器不了解所分配的带宽并且必须依赖帧定时器来检测 LOA 和串扰条件。恢复机制要求集线器跟踪帧定时器(Frame Timer)并能够在下一帧开始之前恢复过来。

集线器故障恢复仅在上行方向进行。主机负责检测它的下行方向的错误并从中恢复出来。集线器故障恢复必须满足下列要求:

- (1) 设备在一帧结束时驱动的非法状态,必须通过禁用该设备所连接的下行方向的集线器端口而予以隔离。
- (2) 如果先前在上行方向已经建立了一个连接,集线器就必须在下一帧开始之前使总线返回到空闲状态。

在无差错的条件下,这些要求可以依靠一个集线器连同每一个分组而接收一个 EOP 信号,并且不会在一帧结束时产生任何的总线操作来得到保证。在描述集线器是如何实现故障能够恢复之前,我们先来讨论集线器帧。

### 11.4.4 集线器帧定时器

每一个集线器都有一个集线器帧定时器,它的定时是来自于由主机所产生的 SOF 令牌,并且在阶段和周期内都会跟踪主机的 SOF 分组。集线器帧定时器在每一次检测到了一个 SOF 时都会复位,并由它来负责产生帧结束(EOF)点。集线器帧定时器必须跟踪主机所产生的 SOF 令牌,并且对于丢失至多两个连续的 SOF 令牌的情况它必须有能力保持和主机 SOF 的同步。所有的集线器都必须实现一个 EOF 定时器并用它来区别时间上的两个点:

- (1) 集线器必须终止其上行连接的时间点(EOF1)。
- (2) 总线必须见到上行通信流结束的时间点(EOF2)。

EOF1 和 EOF2 之间存在的时延与主机和集线器之间定时的偏差和产生 EOP 所需要的时间有关,如图 11-9 所示。

帧定时器在最坏情况下的容限和主机同集线器之间存在的偏移出现时,它必须仍然能够和主机的帧定时实现同步。偏移量是用来适应集线器振荡器的容差的,利用它并有

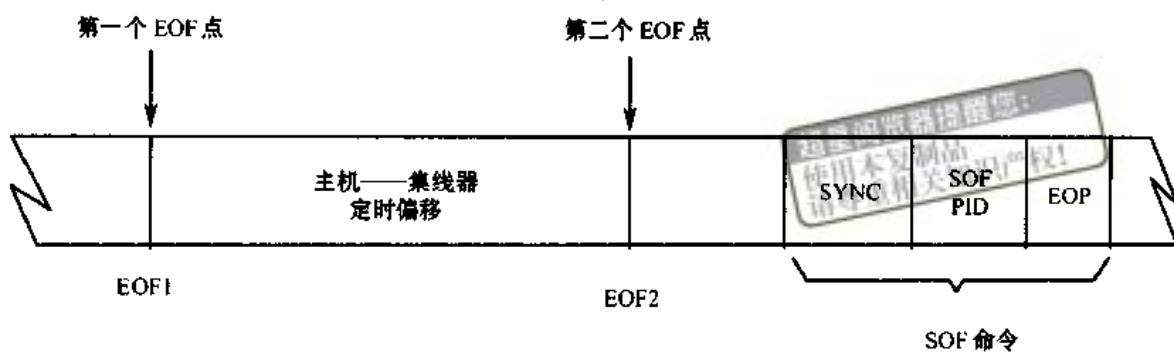


图 11-9 主机—集线器 EOF 偏移

目的地引入在集线器同一个外部信号源相锁定时所出现的来自于 1.000ms 的帧定时误差。上述参数所要求的最小的帧长度(FLmin)为 11985 字节,而最大的帧长度(FLmax)为 12015 字节。

帧定时器的时钟信号来自于集线器的时钟源,利用 SOF 分组它可以实现和主机帧定时的同步。当一个集线器首次接入主机时,对于主机而言帧定时器并未实现同步。当检测到了第一个 SOF 令牌之后,帧定时器会复位并且每 12MHz 时钟周期(时钟频率为 12MHz)记数一次;它会连续记数,直到检测到了下一个 SOF 令牌。在帧定时器复位时,先前的记数值会作为前一个帧长度(PFL)的计数而予以保存。这一数值会因为在其中检测到了一个 SOF 信号的每一帧而更新,并在集线器时钟计数内代表了主机帧时间。在 PFL 计数值和当前的计数值之间存在的差别小于 2 时,就可以认为帧定时器和主机实现了锁定,而且 PFL 位于 FLmin 和 FLmax 之间。

如果接收一个 SOF 令牌的操作失败了,集线器会使用前一个帧长度的计数值(PLF)来作为对当前的帧长度的计数值最好的近似。集线器帧定时器在至多丢失了两个连续的 SOF 令牌时,它仍然可以保证和主机的同步。集线器必须有能力在检测到第一个 SOF 分组(假定没有错过 SOF 分组)之后的三帧时间之内实现和主机的同步。当一个集线器摆脱了重新开始状态之后,它的帧定时器并不会实现同主机的帧定时器相互同步。在这期间,当集线器处于有效状态时,在集线器帧定时器同主机同步之前必须阻止该集线器建立上行连接。

集线器差错恢复将在 11.4.5 节中予以介绍。一旦一个集线器帧定时器被锁住之后,集线器差错恢复就会起作用。如果一个集线器帧定时器未被锁住,它就会使用 FLmin 来作为缺省的帧长度。

#### 11.4.5 靠近 EOF 时的集线器动作

靠近 EOF 时的集线器动作如图 11-10 所示。总共有两个帧定时标记,EOF1 和 EOF2,分别对应于第一个和第二个 EOF 点。所有接收上行通信流量的集线器一旦检测到了 EOF1,它就会向上行方向传输一或两个全速率比特周期长的 EOP 信号,将总线驱动到空闲状态一个比特周期并浮动该总线。从 EOF1 开始时,不允许集线器建立上行连接,直到下一个下行分组,通常是下一个 SOF 令牌分组的末尾。

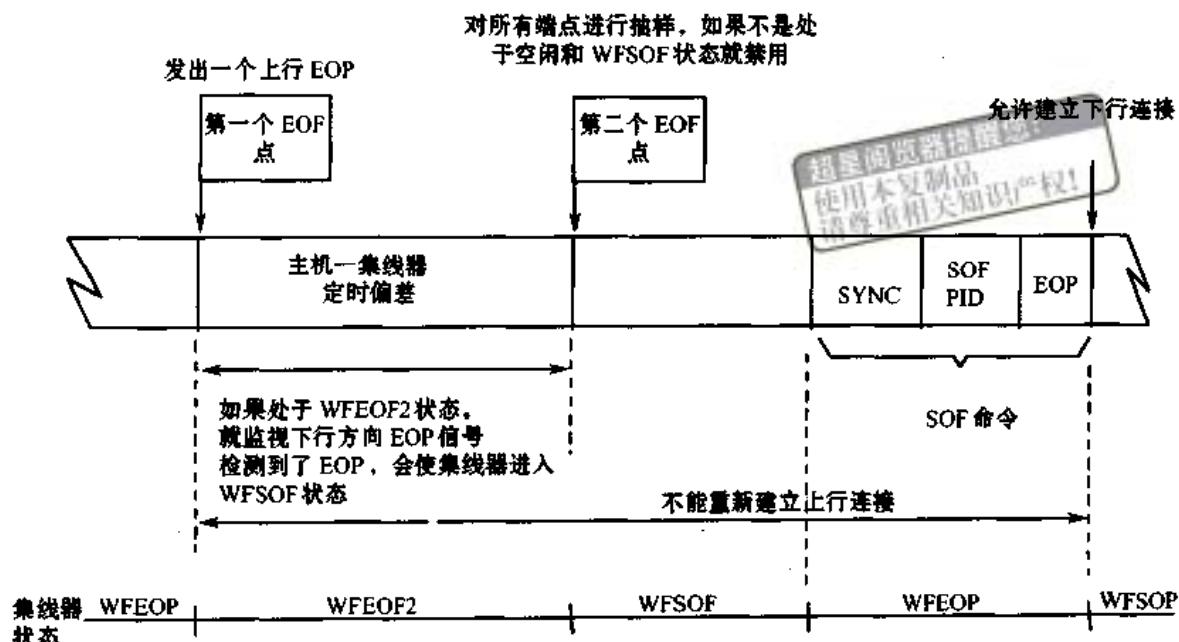


图 11-10 接近帧末尾时的集线器动作

靠近 EOF 时的集线器动作可总结如下：

- (1) 集线器恢复操作仅适用于上行通信流量。如果一个集线器遇到了帧的末尾并且已经建立了下行的连接，它就会保持这一连接并等待下一个下行 EOP 的出现，那时它会拆除连接并返回到 WFSOP 状态。
- (2) 在 EOF1 点，所有的集线器都会传送一个跟有一个空闲状态的上行 EOP 并浮动该总线，除非在下行方向已经建立了连接。EOP 必须不能够截短或延长任何已经进行的上行方向的 EOP。
- (3) 集线器在 EOF1 之后，不会允许在上行方向进一步建立连接。
- (4) 在 EOF1 点处于 WFEOP 状态的集线器必须注意在建立了连接的下行端口上的 EOP 信号。它们必须从 EOF1 至 EOF2 一直监视总线。
- (5) 如果在 EOF1 和 EOF2 之间所形成的窗口内，处于 WFEOF2 状态的一个集线器检测到了一个来自于上行方向的 EOP，该集线器就会进入 WFSOF 状态并可以使它的端口上出现空闲状态。
- (6) 在 EOF2 点，会对所有的端口的状态进行采样，而且如果一个端口没有处于正常的状态(参见表 11-4)，该端口就会被禁用。在 EOF2 点，仍然处于 WFEOF2 状态的集线器会进入 WFSOF 状态。直到接收到了一个来自于主机的分组之后，才允许建立下行连接。

表 11-4 在 EOF2 点集线器的活动

项 目	非空闲状态	空闲状态
WFSOF	禁用端口	不会禁用端口
WFEOF2	禁用端口	禁用端口

### 1. 偏移要求

主机和集线器在它们都同主机的 SOF 实现了同步时,会服从一定的时滞偏移,这可以指出 EOF 点,接近 EOF 点时主机的活动和下一个 SOF 之间所存在的时间间隔长度。图 11-11 说明了帧定时点的临界端点位置。

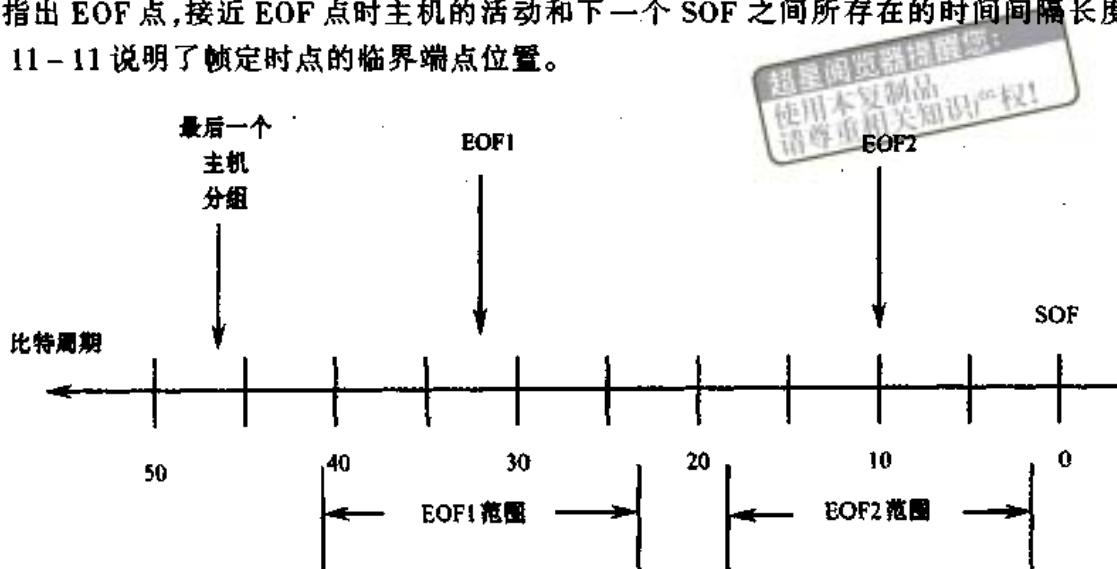


图 11-11 EOF 定时点

### 2. 主机—集线器偏移

通过要求集线器跟踪主机,可以使主机 SOF 点和集线器 SOF 定时器之间存在的偏移量降至最低限度。造成偏移的因素包括这样一个事实:集线器会错过 SOF 而且主机的帧计数器会调整为跟踪一个外部的主时钟。12MHz 时钟是实际所说明的唯一的时钟,所以它是规范所能利用的最佳的间隔粒度。假定一个固定的主机 SOF 定时和两个连续的 SOF 可以被错过,在没有主机定时偏移的情况下累积的最大主机—集线器定时偏移为  $\pm 3$  个时钟。假定主机的时钟每一帧至多可以调整一个比特周期,那么主机的定时就可以超过集线器的定时  $1 + 2 + 3 = 6$  个时钟。最大主机—集线器定时偏移是两个因素的累加和或者是  $\pm 9$  个时钟。

第二个 EOF 点必须实现和 SOF 点足够分离,这样可以使恢复并准备好接收来自于主机的 SOF 令牌。一个集线器必须在它所接入的集线器抵达第二个 EOF 点之前完成发送它的 EOP 信号。这意味着所有集线器 EOF2 点必须至少比主机发出 SOF 之前的时刻早一个比特周期产生。所有集线器的 EOF2 点必须位于一个  $\pm 9$  个比特周期的窗口之内;因此,EOP 信号必须处在这一窗口之外,并且要至少要比主机 SOF 早  $2 \times 9 + 1 = 19$  个比特周期而完成。

下一步就是计算要用多长时间来产生 EOP 并在 SOF 之后多长时间进行该操作。传输 EOP 需要 4 个比特周期。因此,一个集线器必须在 SOP 之前不迟于  $19 + 4 = 23$  个比特周期的时刻开始发送它的 EOP 信号。为了使一个集线器确保它不迟于  $19 + 4 = 23$  个比特周期就开始发送它的 EOP 信号,它必须在此之前 9 个比特周期或在 32 比特周期内开始发送,这一数值正好是第一个 EOF 点的位置。一个集线器最早可以在第一个 EOF 点之前 9 个比特周期或在 41 比特周期开始发送 EOP。

必须使一个集线器在该集线器抵达它的第一个 EOF 点之后就检测到一个来自于主

机的分组已经开始了。集线器传播时延必须计人时延预算之内。每一个集线器时延大约为一个比特中期；所以对从主机开始有 6 个集线器的最坏情况下的拓扑结构来说，总共要有 6 个比特周期的额外的时延。因此，相对于主机的 SOF 定时器，主机的 EOF 传送点应该是距离 SOF  $41 + 6 = 47$  比特周期的那一点。如果主机在 47 比特时仍然在传送并且不能在 SOF 之前完成传输的话，它必须利用一个跟有一个 EOP 的比特填充违规（至少为 8 个连 1）来强制产生一个错误。如果主机在 47 比特时仍然在接收一个分组或一个 EOP 信号，它就会将这一分组作为有错误的情况来对待。

表 11-5 总结了集线器和主机的 EOF 定时点。

表 11-5 集线器和主机的 EOF 定时点

说 明	自 SOF 开始后所经过的比特周期数	注 释
EOF1	32	帧结束点 #1
EOF2	10	帧结束点 #2
主机使全速率发送分组无效	47	主机开始一个全速率分组的最后时刻
主机使低速率发送分组无效	184	主机开始一个低速率分组的最后时刻（扩展至最近的比特周期）
主机使接收分组无效	41	主机会将在 41 比特周期内仍在接收的任一分组作为错误分组来对待

## 11.5 挂起和重新开始

无论集线器是作为一个设备、还是用来传播挂起和重新开始信号，它都必须能够支持挂起和重新开始。集线器可以同时支持全局挂起和重新开始信号以及选择性挂起和重新开始信号。选择性挂起和重新开始可以通过激活/禁用每个端口来实现。全局性挂起由主机通过集线器的根端口来实现。而全局性重新开始则是由主机或一个集线器的下行端口来发起的。

### 11.5.1 全局挂起和重新开始

全局性挂起由主机关闭了通往整个总线的下行通信流来发起。如果一个集线器在其根端口上检测到了一个至少持续 3.0ms 的连续的空闲状态，那么该集线器就会进入挂起状态。当一个集线器被置为挂起状态时，它会把它的中继器置为 WFSOP 状态，浮动它的所有输出驱动器，保持它的所有控制和状态比特的稳定值并为它的每一个下行端口保留当前的状态信息。一个被挂起的集线器会关闭它的时钟，所以它没有任何时间观念，而且只能对总线处理作出响应。

集线器重新开始是由一个集线器根端口或处于激活状态的一个集线器下行端口上的任一个总线处理来发起。集线器重新开始也可以由处于未连接、禁用或挂起状态的一个

集线器下行端口上的USB设备的连接/拆除操作来发起。如果一个总线处理操作出现在了一个被激活的集线器下行端口上，那么该集线器会立即把一个由空闲变化至重新开始的总线处理操作反映到那个端口、其它所有的处于激活状态的端口以及它的根端口上。图11-12说明了在一个设备向一个集线器发起一个唤醒操作的重新开始时序期间，所存在的定时关系。

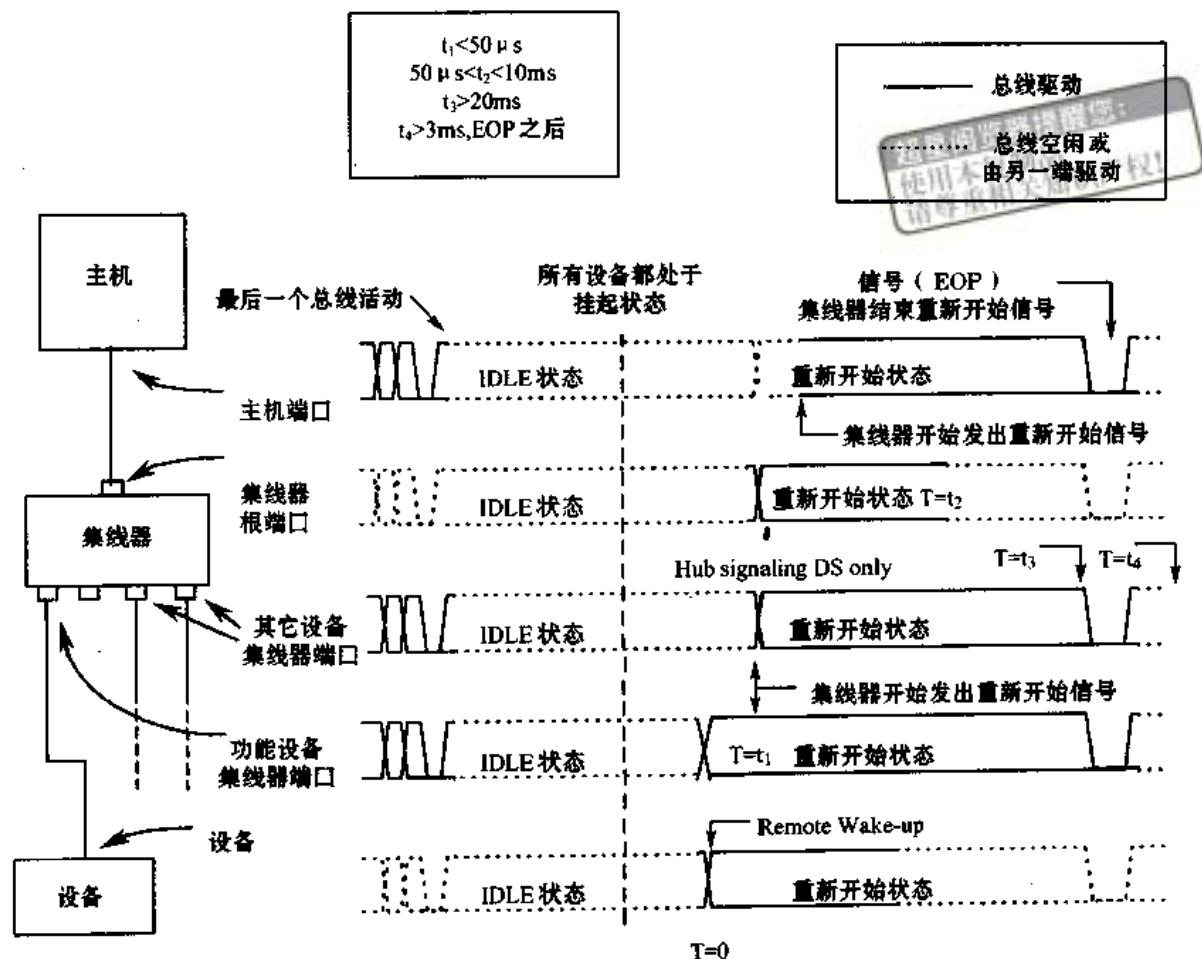


图11-12 重新开始信号

如图11-12所示，一个集线器必须满足四个时间参数。 $T=0$ 代表重新开始信号到达该集线器端口的时刻。 $t_1$ 是一个集线器必须作出响应来把重新开始信号驱动到上行端口和所有的被激活的集线器下行端口的时刻。 $t_2$ 则是集线器向其根端口驱动一个由上行方向发起的重新开始信号，并将目前驱动到其根端口的总线状态反映到它的下行端口上的时刻。 $t_3$ 时间间隔则是一个集线器产生一个下行的K信号送往发起该重新开始操作过程的那个设备的时间。 $t_4$ 则代表了集线器在检测到了一个下行低速率EOP之后，直到它设置了它的中断比特以指出在一个选择性挂起之后出现了一个重新开始操作，该集线器所必须等待的时间。

当一个设备将一个由空闲至重新开始的变化向上传送至一个集线器时，该集线器会通过将重新开始(K)状态驱动到总线上，从而将起送到根端口和包括发起该重新开始信号的那个端口在内的所有下行被激活的下行端口上去(将一个总线段的两端都驱动到相

同的状态是可以接收的)。在驱动了一个重新开始信号之后,该集线器开始一个返回到完全启动状态的过程(例如重启时钟信号)。当一个集线器启动之后,它会将连接反向,从而是集线器根端口上的 K 状态在其下行端口上可以保持在 K 状态上(这一情况假定了在我们所讨论的那个集线器上的一个集线器已经接收到了一个重新开始信号并将其驱动到了下行方向)。一个集线器在收到了一个来自于下行方向的重新开始信号之后,它的连接反向操作不会快于  $50\mu s$ ,也不会慢于  $10ms$ 。时间参数也暗示了一个集线器在接收到了一个重新开始请求之后,该集线器应该不迟于  $10ms$  就进入完全启动状态。

重新开始信号会一直向上行方向进行传送,直到该信号到达了主机。主机会向下行方向反射该重新开始信号至少  $20ms$ ,这样可以保证所有的设备可以有时间来启动并检测下行的重新开始信号。主机通过驱动一个两个低速率比特周期长的 EOP 信号来终止这一重新开始时序。EOP 会作为一个有效的分组结束而被解释,从而使所有的集线器拆除掉他们的连接并通知总线上的所有设备该重新开始时序已经结束了。而发起该重新开始时序的那个设备在它检测到了一个 EOP,从而确定该重新开始时序已经结束了之前,它必须处于等待状态。

主机必须能够在重新开始时序结束之后立即传送下行通信流量,以防止下行方向的设备重新进入挂起状态。集线器控制器必须能够在重新开始时序结束之后的  $10ms$  之内接收分组通信。注意:一个远程唤醒的设备在最后一个总线活动过去  $5ms$  之前不能够开始一个重新开始时序。这样做可以允许集线器进入到挂起状态,以使其重新启用所有的端口而不仅仅是全速率端口。表 11-6 总结了集线器对主机发起的、下行方向的信号所采取的响应。

表 11-6 在全局挂起状态内集线器的活动

端口状态和信号类型	下行端口响应
激活端口,接收到重新开始信号(K)	向下行方向发出重新开始信号
端口被禁用,接收到重新开始信号(K)	无任何操作
端口被挂起,接收到重新开始信号(K)	向下行方向发出重新开始信号

当一个集线器由挂起状态变换为启动状态时,它的帧定时器不能操作并且集线器 EOF 定时电路只有在接收到了第一个 SOF 之后才能工作。为了阻止在一个刚刚被重新启用的总线段上的一个串扰的下行设备锁住总线,有必要在集线器处于启动状态时阻止其传播上行通信流,直到集线器帧定时器开始工作之后。这可以通过使集线器中继器状态机在其一退出重新开始状态之后就变换至 WFSOF 状态来实现。当集线器中继器处于 WFSOF 状态时,所有的上行通信流都会被忽略。中继器状态机会一直保持在 WFSOF 状态,直到检测到了一个 SOF 信号,这时它就会转变为 WFEOP 状态。下行通信流不会受到帧定时器状态的影响。为了防止出现不必要的超时操作,建议主机在一个刚刚被重新启用的总线段上不要发送任何一个寻址到设备的分组,直到主机至少向刚刚被重新启用的集线器或集线器发出了 SOF 令牌才可以进行。这个条件可以通过这样一个事实来予以保证,那就是一个集线器只有在一个重新开始时序完成后的  $3.0ms$ ,它才能向主机报告

它检测到了一个重新开始时序。

### 11.5.2 选择性挂起和重新开始

选择性挂起和重新开始提供了一种将单个设备或一个总线段置为低速率状态的方法。选择性挂起依赖于一个集线器利用一个 SetPortFeature(PORT\_SUSPEND)请求来有选择地挂起个别的端口,这样可以将一个集线器(指的是一个被禁用的集线器)上的一个端口置为挂起状态(参见图 11-4 和图 11-5 中的状态图)。在挂起状态,一个端口会被阻止其向下行方向传播任何任何总线活动(除了端口复位请求之外),而且该端口仅能通过集线器状态比特来反映上行总线的状态变化;即一个启动后的集线器不能从一个被挂起的端口向其根端口传播上行通信流量。该集线器也必须保证通过端口挂起请求而访问的端口不会在一个分组处理操作期间被挂起。作为对一个端口挂起的响应,当总线处于空闲状态时,位于目标端口下行方向上的所有设备在 3.0ms 之内没有检测到总线的活动,就会进入挂起状态。端口挂起请求只能为一个启动后的集线器所理解。如果主机希望向一个被挂起的集线器发送请求,它必须首先唤醒该集线器,然后才能发送出所需要的请求。

#### 1. 对启动后的集线器的选择性挂起

图 11-13 给出了一个设备和一个启动后的集线器(集线器 Y)之间所传送的信号。如果一个集线器端口被有选择地挂起,而且此时该集线器处于启动状态,那么该集线器必须阻止被挂起的端口上的任一总线活动反射到该集线器上的其它被激活的端口上去,因为这些端口可以传送总线通信流量。另外,端口 B 上的重新开始信号不能够传播至其它被挂起的端口(例如端口 C)。其中实线代表重新开始信号的范围,它在集线器 Y 的端口 B 处结束。重新开始信号既可以由一个直接重新激活该集线器端口(下行方向的重新开始信号)的一个主机请求来产生,也可以由被挂起的总线段上(上行方向的重新开始信号)的一个设备来产生。

上行方向的重新开始信号可以由一个被挂起的集线器端口上的一个空闲至重新开始(设备发出的重新开始信号),一个空闲至 SEO(设备拆除)或一个 SEO 至空闲(设备接入)的总线状态转变来发起。集线器 Y 上的端口 B 在接收到重新开始信号的 50 $\mu$ s 之内反映出重新开始信号,以通知远程唤醒的设备(设备 B)已经检测到了它的重新开始信号。如果端口 B 上的变化不是一个空闲至重新开始(J 至 K)变化,那么就不会向下行方向反射任何信号,而是继续将其输出缓冲器保持在 Hi-Z 状态。后面一种情况覆盖了连接或位连接事件。

集线器 Y 上的端口 B 接收到重新开始信号后会使下列顺序的时间产生:

- (1) 集线器 Y 的端口 B 在检测到重新开始信号的 50 $\mu$ s 内,应该向下行方向反射回该重新开始信号。
  - (2) 集线器 Y 至少应该将下行的重新开始信号保持 20ms。
  - (3) 在 20ms 结束时,集线器会用一个低速率 EOP 来终止重新开始信号(此时下行通信流会流过该端口)。
  - (4) 在 EOP 结束后的 3.0ms 时间内,集线器控制器内的一个中断会置位。
- 集线器负责保持所有的重新开始定时参数。主机不必跟踪任何定时,它可以利用一

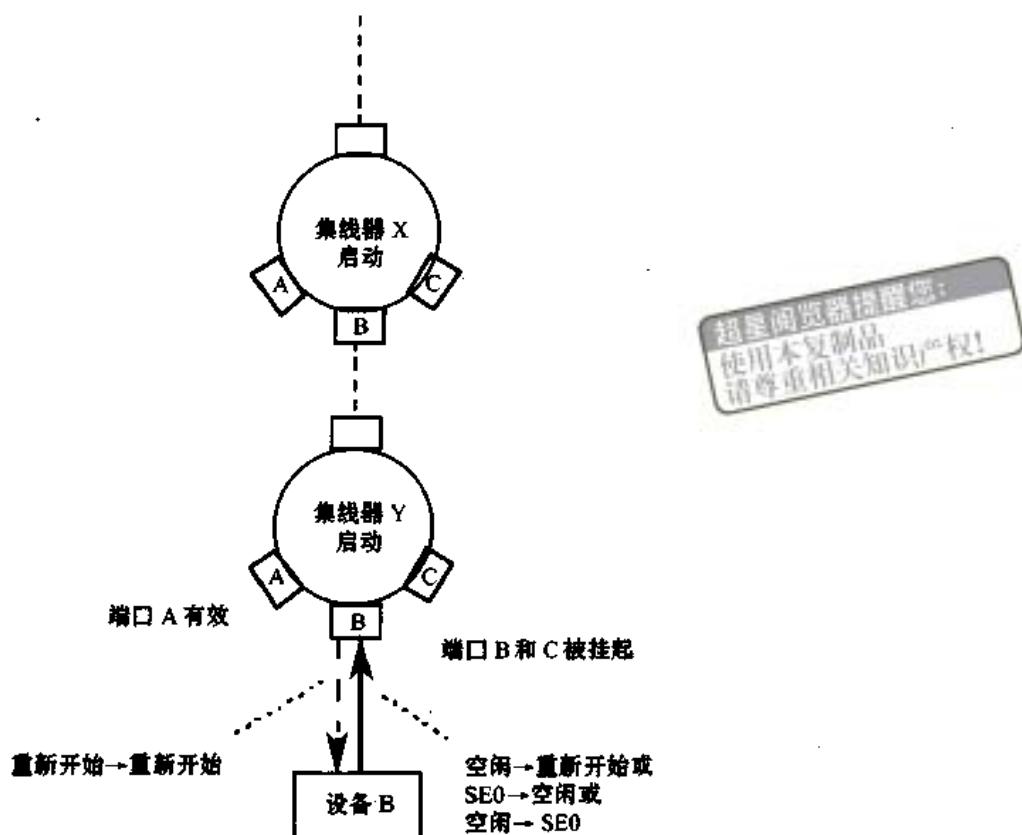


图 11-13 在一个启动的集线器上的选择性挂起信号

个 GetPortStatus 请求来查询该集线器，从而确定发生了一个重新开始事件。由于集线器会独立地处理选择性重新开始操作，所以主机不会在不是由主机发起的一个选择性重新开始时间中接收到直接的信号。对于主机来说很有必要来查询集线器（作为其状态轮询的一部分），以此来确定一个选择性重新开始事件已经结束了。在选择性重新开始信号的终点和对中断比特的置位之间所存在的 3.0ms 的时延，可以保证产生任何一个和新近被重新启动的设备有关的分组通信流，直到新近被重新启动的设备有时间实现它的帧定时器和 SOF 的同步；但是，下行设备可以见到不是专门对其进行寻址的通信流和 SOF 令牌。当一个重新开始信号是由一个下行设备发起时，主机会将对选择性重新开始的查询作为它的状态查询工作中的一部分来进行。集线器也必须保证在一个分组处理操作正在进行期间，该端口不会被重新激活。

下行选择性重新开始也可以由一个 ClearPortFeature(PORT\_SUSPEND) 请求来发起。这一请求会使被禁用的端口将重新开始信号驱动到总线上至少 20ms，后面再跟上一个低速率 EOP。同设备发起的集线器重新开始一样，集线器控制器内的中断比特发出集线器重新开始操作结束信号的过程只有在低速率 EOP 结束后的 3.0ms 才可以进行。主机必须查询集线器中断来确定什么时候集线器重新开始信号完成了。表 11-7 是一个启动后的集线器在出现下行方向发起的集线器重新开始信号的情况下，它所采取的操作。

表 11-7 在重新开始期间启动了的集线器的动作

端口状态和信号类型	收到信号的端口的响应	邻近的被激活的端口的响应	邻近的被禁用的端口的响应	邻近的被挂起的端口的响应
端口被挂起, 接收到了重新开始信号	在收到信号的端口上向下行方向反射 K 信号。发起端口唤醒操作。设置状态比特和中断	无任何操作	无任何操作	无任何操作
端口被激活、禁用或挂起, 并且接收到了拆除信号	设置端口未连接, 禁用和改变状态比特。设置中断	无任何操作	无任何操作	无任何操作
端口被禁用并且接收到连接信号	设置中断	无任何操作	无任何操作	无任何操作

## 2. 对一个被挂起的集线器的选择性重新开始操作

主机可以挂起一个集线器端口, 进而挂起整个集线器。在这种情况下, 集线器的连接在集线器一进入挂起状态时就立刻发生变化。图 11-14 给出了设备为在一个被挂起的集线器上的被挂起的端口而发起的连接的示例。设备 B 可以向集线器 Y 上的端口 B 发送连接、拆除或集线器重新开始信号。作为对其的响应, 无论设备 B 向上行方向发送的

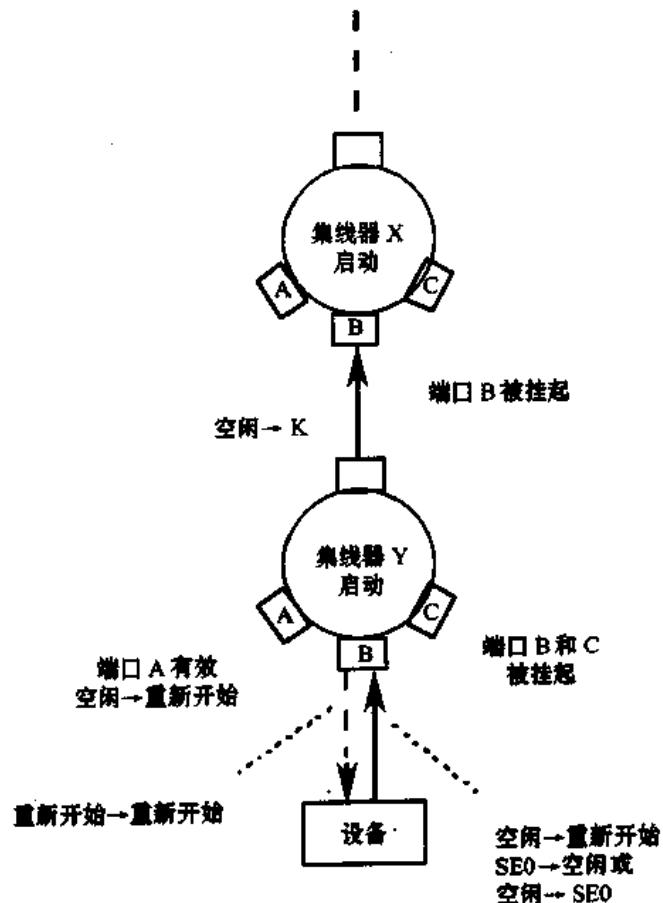


图 11-14 由设备发起的用于被挂起的集线器的重新开始操作

信号是什么,集线器 Y 都必须将该信号转变为一个 J 至 K 的变化。同时该集线器会将集线器重新开始状态驱动到根端口和其它所有被激活的端口(端口 A)上。如果设备将总线驱动到了集线器重新开始状态,那么集线器 Y 就必须将端口 B 也驱动到集线器重新开始状态。如果端口 B 检测到发生了一个连接或拆除事件,该集线器就不会将一个集线器驱动到下行方向发出信号的那个端口,而是将该端口的输出缓冲器保持在 Hi-Z 状态。表 11-8 总结了被挂起的集线器在重新开始期间所进行的活动。

表 11-8 在重新开始期间被挂起的集线器活动

端口状态和信号类型	收到信号的端口的响应	邻近的被激活的端口的响应	邻近的被禁用的端口的响应	邻近的被挂起的端口的响应
端口被激活, 接收到了重新开始信号(K)	向上行方向和下行方向反射重新开始(K)信号。不设置状态比特	向下行方向发出重新开始(K)信号	无任何操作	无任何操作
端口被禁用, 接收到了重新开始信号(K)	无任何操作	无任何操作	无任何操作	无任何操作
端口被挂起并且接收到了重新开始信号(K)	在根端口上向上行方向, 并在收到信号的端口上向下行方向反射重新开始(K)信号	向下行方向发出重新开始(K)信号	无任何操作	无任何操作
端口被激活、禁用或挂起, 并且接收到了拆除信号	向上行方向反射重新开始信号。开始集线器唤醒操作。更新端口连接和变化状态比特。设置中断	向下行方向发出重新开始(K)信号	无任何操作	无任何操作
端口被禁用, 并且接收到了连接信号	向上行方向反射重新开始信号。开始集线器唤醒操作。设置端口连接和变化状态比特。设置中断	向下行方向发出重新开始(K)信号	无任何操作	无任何操作

主机所发出的用于被挂起的集线器的信号如图 11-15 所示。唤醒一个位于一串集线器最低层上的一个设备需要多个步骤来完成。下面我们就来讨论这一唤醒一个 USB 设备的过程。

(1) 主机通过向集线器发送出一个端口集线器重新开始请求,可以使集线器 X 上的端口 B 脱离挂起状态。作为对这个集线器重新开始请求的响应,集线器 X 会通过将一个至少长 20ms 的 K 信号并跟上一个低速率 EOP 驱动到它的端口 B 上来发起重新开始信号。集线器 Y 检测到该集线器重新开始信号并开始它的唤醒过程。EOP 说明在 EOP 被集线器 X 送出后的 3.0ms,该集线器重新开始时序已经完成了。集线器 X 控制器内的集线器重新开始完成状态比特会置位,以此来指示集线器重新开始时序已经结束了。在集线器 X 的这一阶段结束时,端口 B 处于激活状态,而且集线器 Y 也启动了。

(2) 下一步是要将集线器 Y 上的端口 B 退出挂起状态,而使其进入激活状态。除了请求是送往集线器 Y 上的端口 B 而不是集线器 X 之外,该过程同上面所描述的步骤一模一样。在该步骤的结束时刻,集线器 Y 上的端口 B 被激活,而且设备 B 收到了集线器重新开始信号并处于启动状态。此时,主机就可以同设备 B 进行通信了。这样的集线器重新开始策略就可以允许总线顺序挂起,同时允许在一个被挂起的总线段上的任一个设备唤醒该总线。它也允许在一个时刻唤醒一个嵌套的集线器。

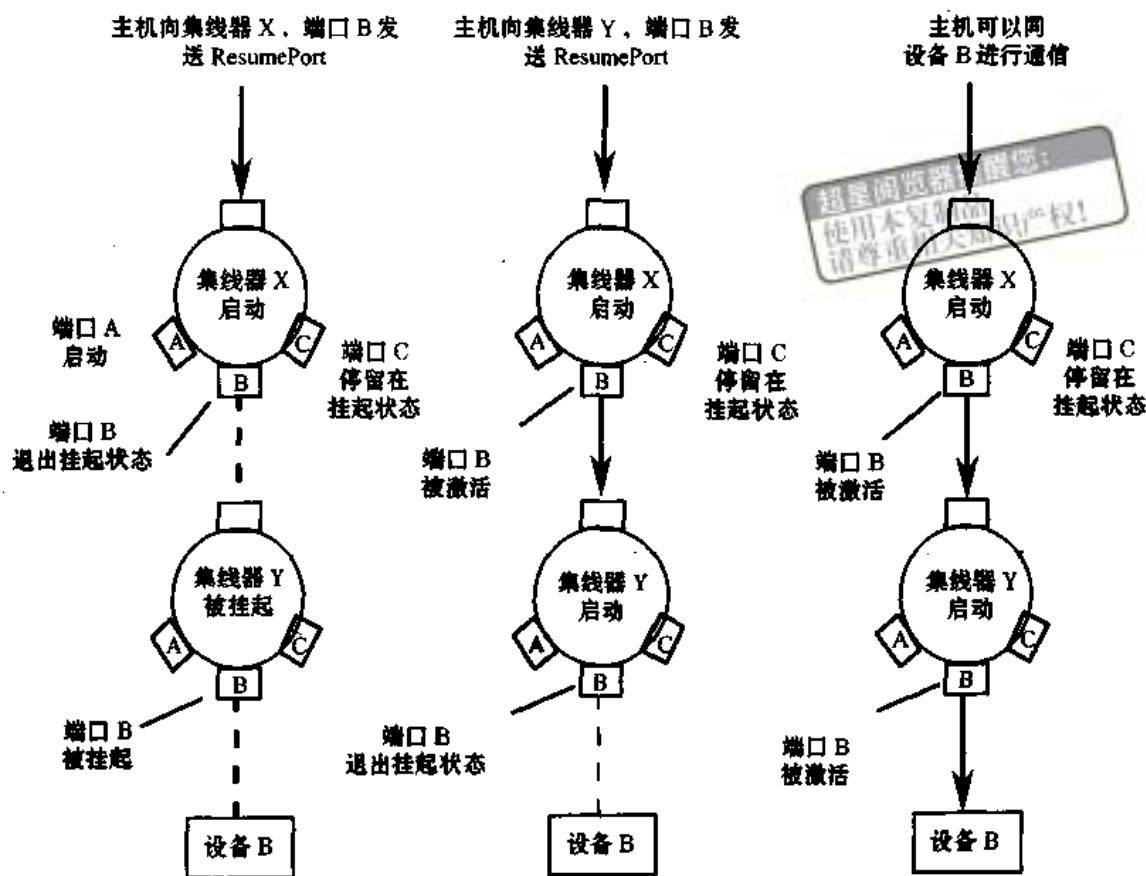


图 11-15 由主机发起的重新开始信号通过被挂起的集线器

## 11.6 USB 集线器复位操作

一个 USB 集线器必须可以利用一个主机请求来产生每个端口的复位信号，并可以在它的根端口上接受复位信号。下面几节我们将讨论集线器复位操作和它同重新开始、插入检测和上电事件的交互。

### 11.6.1 集线器在根端口上接收复位信号

送往一个集线器的复位信号仅在该集线器的根端口的上行方向有定义。一个启动的集线器如果在其检测到了一个 SE0 信号持续了  $2.5\mu s$  或更长时间，那么它就可以开始它的复位时序了，并且必须要在不迟于  $5.5\mu s$  的连续 SE0 时完成它的复位时序。 $2.5\mu s$  的下限是为了防止将低速率 EOP 选通信号（它的最大长度为  $1.3\mu s$ ）错误地解释为复位信号的需要。严格被挂起的集线器必须将一个复位信号的开始边沿解释为一个重新开始信号事件并开始它的唤醒时序。该集线器必须启动并在不迟于复位信号结束的  $10ms$  内完成复位。

在完成了一个复位操作之后，一个集线器控制器会处于下列状态：

- (1) 集线器控制器缺省地址为 0。
- (2) 集线器控制比特被设置为缺省数值。

- (3) 集线器中继器处于 WFSOP 状态。
- (4) 所有的下行端口处于掉电状态(实现了电源开关的集线器)。
- (5) 所有的下行端口处于未连接的状态(没有实现电源开关的集线器)。

如果一个总线上包含了具有电源开关的端口,就不能保证主机的复位信号会一直向下行方向传送。主机必须保证在它通过总线枚举过程之后,其拓扑结构中的每一层都被复位,而且枚举复位必须一层接一层地进行。但是,处于掉电状态的设备如果其断电时间足够长的话,它就会有效地被复位,而且在它之下的那些自供电的设备/集线器会将自己和下行端口复位。

### 11.6.2 端口复位

一个集线器可以利用 SetPortFeature(PORT\_RESET)请求来实现对每个端口的复位。该请求会指出一个下行端口号。作为对一个 SetPortFeature(PORT\_RESET)请求的响应,该集线器会把一个 SE0 信号驱动到它的下行端口上至少 10ms,使总线返回到空闲(J)状态并把该端口置为激活状态。SetPortFeature(PORT\_RESET)请求是一个基本操作;而复位开始和结束位置间所存在的 10ms 则由集线器来控制。该集线器必须能够将复位请求的状态返回到主机;即无论复位操作是否完成,主机都不必在复位期间跟踪消耗的时间。这一端口复位请求在任何情况下都可以发送到一个端口上去;但是,如果该复位请求发送到了一个处于掉电或未连接状态的端口,就不会产生任何下行信号。

设备接入检测要求所讨论的端口的电源开关处于闭合状态(如果电源开关是可选的话)。当一个设备接入时,一个集线器可以通过一个 SE0 至 DIFF1 或 DIFF0 的总线变化检测出一个接入操作。这就要求在连接检测进行时,被禁用的端口不会被集线器所驱动。由于在检测到上一个拆除事件时,该端口会被禁用,而它的输出驱动器也会被浮动,所这不会产生什么问题。通过检测 D+ 或 D- 信号线中的哪一条被拉至高电平,主机就可以确定设备的速率。

在一个设备所连接的端口被激活之前,我们必须保证该端口已经被复位。由于不可能依赖于一个拆除事件所造成的  $V_{bus}$  断电来重启一个设备,因此在被激活之前,一个端口必须被复位。这可以通过一个基本的 SetPortFeature(PORT\_RESET)请求来实现,它可以发出一个 SE0 复位信号,然后激活该端口。USB 设备,包括集线器必须在复位信号被重新声明之后不迟于 10ms 就可以对一个主机访问作出响应。

### 11.6.3 电源供给和复位时延

由于在 USB 系统当中,USB 元件可以进行热插拔,而且集线器可以实现电源开关操作,因此有必要了解在电源开关和/或设备接入之间所存在的时延和设备内部的电源何时才能稳定。

图 11-16 给出了一个设备接入了一个电源开关闭合的集线器上的端口的情况。这里需要考虑两个时延。 $\Delta t_1$  是集线器端口转变为工作状态所需的时间。 $\Delta t_2$  是设备内部的电源稳定下来所需的时间。如果一个设备接入了一个没有电源开关的集线器端口或是接入了一个电源开关已经闭合的集线器端口,我们仅需要考虑  $\Delta t_2$ 。 $\Delta t_1$  是集线器端口开关类型的函数,该参数可以通过一个集线器控制器命令来读取。 $\Delta t_2$  对于所有的集线器

和设备实现而言,必须小于 100ms。有必要对  $\Delta t_2$  加上一个最坏情况下的上限,这是因为它是依设备而定的,并且只有在上电和复位时序结束时才能报告。

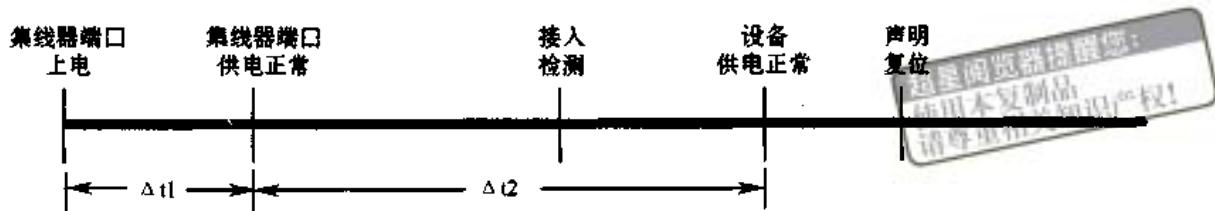


图 11-16 上电定时

如图 11-16 所示,在一个设备的内部电源稳定下来之前,就可以检测一个设备的接入。这必须保证有最少 10ms 的时间,这期间一个设备的内部电源会进入稳定状态并声明一个复位信号。因此,在一个设备接入之后,不能立即声明一个复位信号,而必须等待  $\Delta t_2$  的时间。如果该集线器实现了电源开关,在进行复位之前必须经过  $\Delta t_1 + \Delta t_2$  的时间。

USB设备必须以这样一种方式上电，它在复位过程中不会驱动D+和D-数据线（除了具有上拉电阻的一端）。这样做是使上行集线器可以向下行方向驱动复位信号并保证下行方向的设备可以检测到复位信号的要求。

## 11.7 集线器电源分配要求

集线器可以向下行方向的元件提供一定数量的电源，并负责在枚举阶段向主机报告它的电源分配能力。USB 的要求规定应该支持通用的合法的总线拓扑结构，同时还必须阻止为不合法的拓扑结构上电。一个不合法的电源拓扑结构是指违反了枚举过程中所建立的对电源分配的约定的结构。

集线器即可以是由本地供电，也可以是由总线供电，还可以是两种供电方式的混合。举例来说，一个集线器的 SIE 和上拉电阻可以从总线上获得电源供应，而由一个本地的电源为它的下行端口供电。一个集线器只能向下行方向提供电源而不会把电源驱动到上行方向。对于集线器电源分配的完整的讨论，是在 7.2 节进行的。

总线供电的集线器必须能够为它的下行端口提供端口电源开关，并且在该集线器退出上电状态或在它的根端口上接收到了一个复位信号时，断掉所有下行端口上的电源。端口也可以在主机软件的控制下，闭合或关闭电源开关。一个具体的实现可以为每一个端口都提供一个电源开关或者为所有的端口提供一个单一的开关。端口复位请求不会影响用于一个端口的电源开关的状态。一个集线器端口必须通电，从而在下行方向上进行设备连接检测操作。

为了安全,所有的本地供电的集线器在其下行端口上都必须实现电流限制。在任何一种情况下,任一个 USB 集线器端口也不能拉下 25VA(实际的过电流跳闸点可能低于这一数字)。如果出现了一个过电流条件,即使这只是暂时的现象,也必须将其报告给主机。这可以通过一个由集线器请求反映的一个过电流状态来实现。一旦检测到了一个过

电流状态,就会进入过电流检测状态,该状态可以由一个主机请求来清除,或者在集线器刚复位时予以清除。过电流检测操作必须禁用所有的受影响的端口。如果该过电流引起了永久性的断电现象(例如一个保险丝熔断现象),该集线器必须在其一退出复位状态或上电时就报告该现象。

过电流保护即可以对所有的下行端口整个地予以实现,也可以以每一个端口为基础来实现。这些端口可以选择使其分成两个或多个子组,每一个子组都会有其自己的过电流保护。

## 11.8 集线器端点组织

除了端点 0 之外,集线器类型还定义了一个额外的端点,该端点是所有设备都需要的:状态变化端点。主机系统可以通过状态变化端点来接收端点和集线器的状态变化指示。状态变化端点支持中断传输。如果该集线器在它的任何一个端口上都没有检测到变化,那么该集线器就会向状态变化端点上的请求返回一个 NAK 握手分组。当集线器检测到了任何变化时,集线器会用说明了发生变化的实体的数据来响应。驱动集线器的主机软件负责检查传送来的数据以决定哪一个实体出现了变化。集线器在逻辑上的组织如图 11-17 所示。

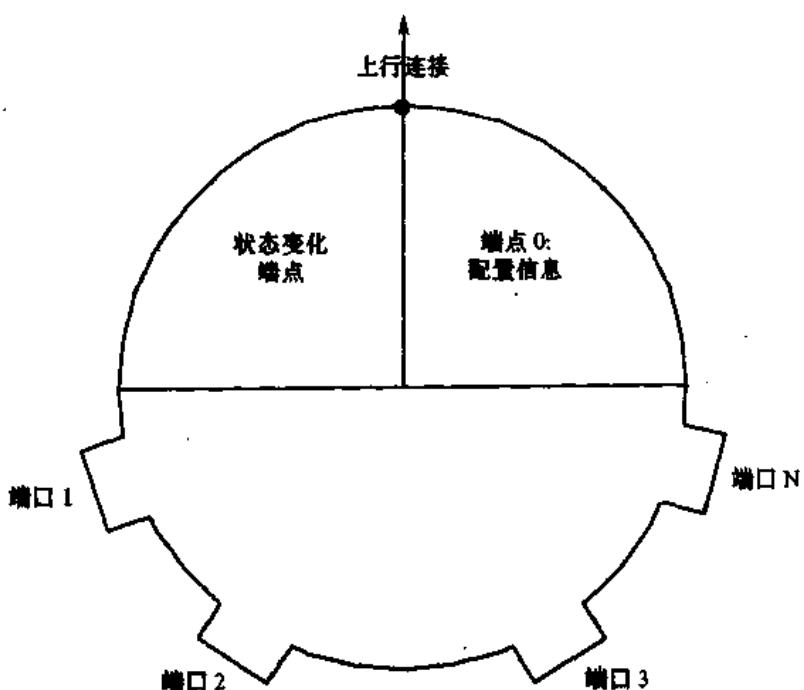


图 11-17 集线器组织示例

### 11.8.1 集线器信息体系结构和操作

集线器描述符以及集线器/端口状态和控制可以通过缺省管道来访问。当一个集线器在一个端口上检测到了一个变化或该集线器改变了自己的状态时,状态变化端点会以 11.8.3 节所描述的形式向主机传输数据。

USB集线器会检测端口状态的变化。设备接入集线器上的一个端口会引起不同的硬件操作事件。而且，主机系统软件会通过向集线器发送命令来引起集线器状态的变化，如图11-18所示。由于有两种原因可以使集线器的状态发生变化，因此USB集线器会为每一个硬件引起的时间报告一个状态变化。该集线器会在主机查询时持续报告一个状态的变化，直到主机对一个特殊的事件成功地作出了应答。利用该报告机制，系统软件可以确定自从集线器报告了最后一个事件之后都发生了哪些变化。这一方法可以将系统软件所必须携带的设备状态信息减少至最低限度。

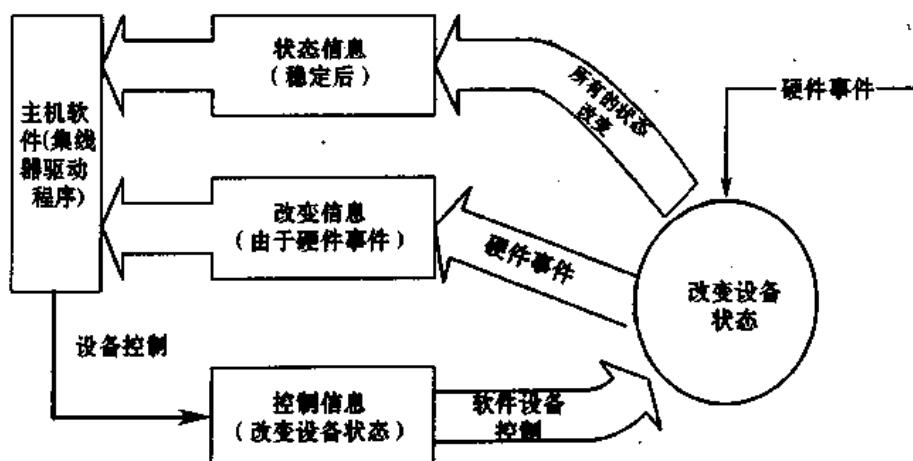


图11-18 状态、状态变化和控制信息对于设备状态的关系

主机软件会利用中断管道连同状态变化端点一起来检测集线器和端点的状态所发生的变化。

### 11.8.2 端口变化信息处理

集线器会基于每一个端口来通过端口命令报告一个端口的状态。主机软件可以通过清除同集线器所报告的状态变化相对应的变化状态来应答一个端口变化。该应答会为那个端口清除变化状态，从而使将来传送至该状态变化端点的数据不需要报告前一个事件。这允许为以后的变化重复该过程（参见图11-19）。

### 11.8.3 集线器和端口状态变化位图

集线器和端口状态变化位图如图11-20所示，指出了集线器或一个端口是否经历了一个状态变化。这一位图也指出了哪一个端口出现了状态变化。该集线器可以在状态变化端点上返回这一数值。集线器会按字节来报告这一数值。这就是说，如果一个集线器有四个端口，它会返回一个字节的数值，并在无效的端口标号域的位置报告一个0。系统软件清楚一个集线器有多少端口（这由集线器描述符来报告）并会相应地对集线器和端口状态变化位图来进行解码。该集线器会在该集线器和端口状态变化位图的比特0处，报告集线器状态上出现的任何变化。

集线器和端口状态变化位图的大小会有所差别，但最少应为一个字节。集线器会根

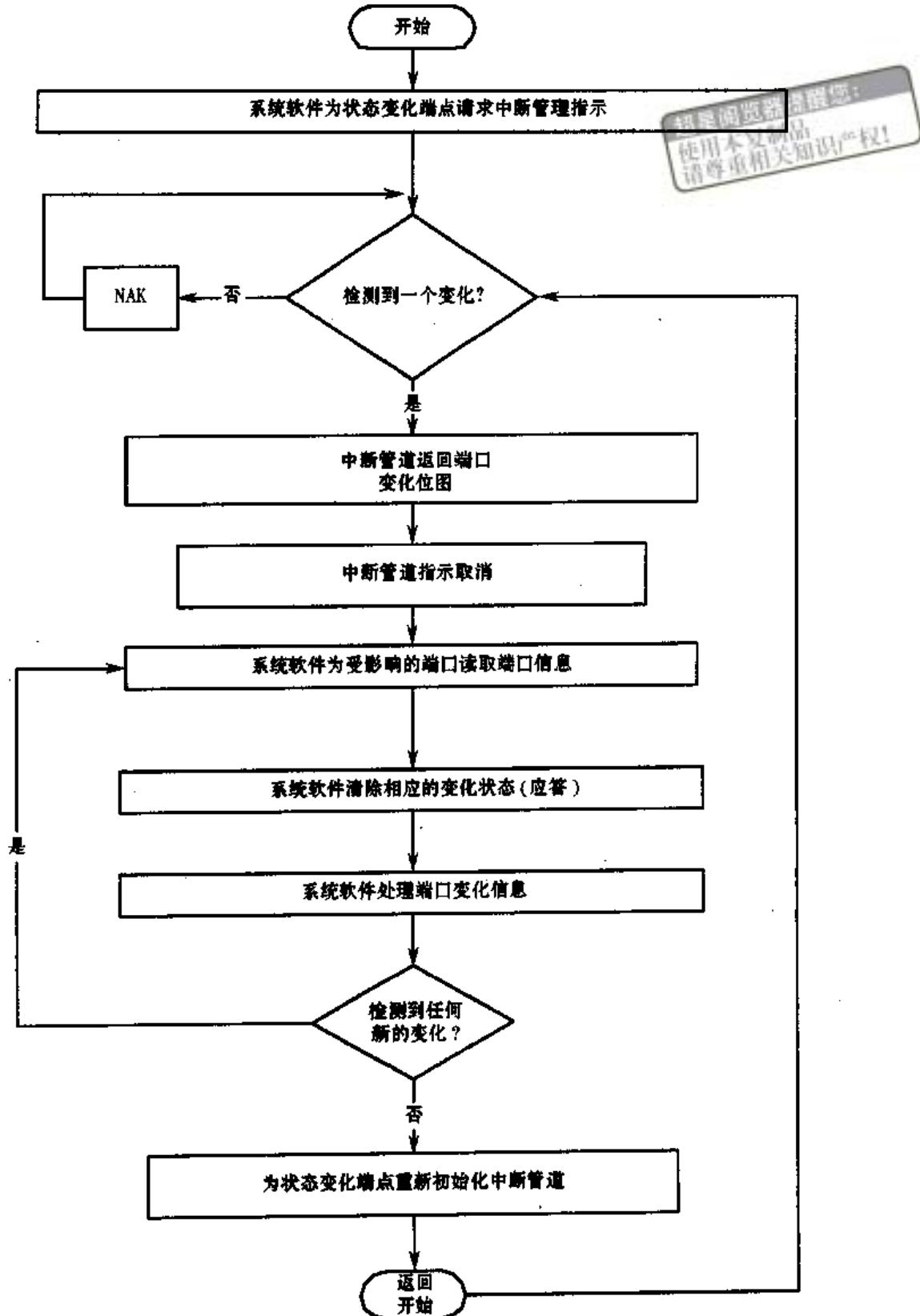


图 11-19 端口状态控制模式

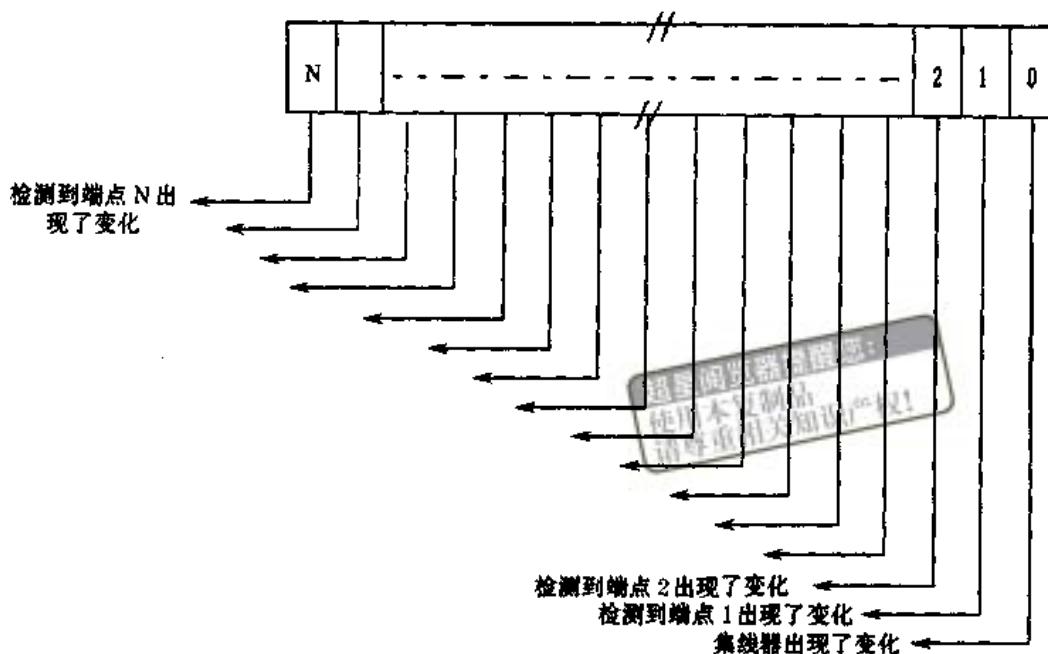


图 11-20 集线器和端口状态变化位图

据字节粒度的要求(即将位图进位到最接近的字节数),报告和该集线器的端口一样多的比特数。

主机控制器会在任何时刻查询状态变化端点,并且集线器和端口状态变化位图所返回的任何一个状态发生变化的比特都是非 0 值。集线器会在一帧的末尾(EOF2)对变化进行抽样,从而为下一个 USB 帧周期内的一个可能的数据传输作好准备。如果检测到了一个变化,那么在下一个 USB 帧中,数据就会通过状态变化端点来传输。图 11-21 给出了由于集线器和端口状态变化比特的抽样机制。

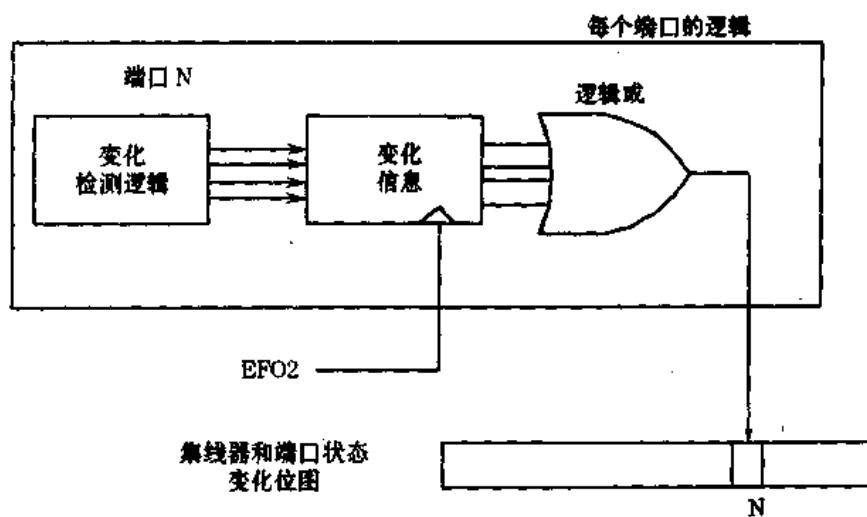


图 11-21 集线器和端口变化比特抽样举例

## 11.9 集线器配置

集线器可以通过标准的 USB 设备配置命令来进行配置。关于电源要求和寻址能力，一个未经配置的集线器会表现出和其它所有未经配置的设备一样的操作。未经配置的集线器不会向下行端口供电。配置一个集线器会激活状态变化端点。在适当的时刻，系统软件可以向集线器发出命令来把端口电源开关变换到闭合和断开状态。

在由配置操作确定该集线器的特性之前，系统软件会检查集线器的描述符。通过检查一个集线器的特性，系统软件会保证，如果在对一个集线器端口供电之后会违反 USB 电源拓扑结构的要求，它可以通过不对该集线器端口供电的方法来禁止出现这样的不合法的电源拓扑结构。

## 11.10 集线器端口电源控制

集线器允许端口的电源可以由主机系统来控制。正如前面提到的一样，在下行端口上集线器支持每个端口或组模式的电源开关。开关端口电源可以利用下列定义的集线器命令来实现。每个端口都具有电源开关的集线器通过在端口电源控制请求的一个请求域内设置一定的值，也可以允许组模式的电源开关（参见 11.12.2 第 9 小节中 SetPortFeature(PORT\_POWER) 请求的定义）。

集线器可能会需要为一些端口屏蔽掉组模式的电源控制。这样可以允许一个集线器为某些端口进行独立的电源开关控制操作，而不考虑通用的端口开关特性。举例来说，考虑一个具有组模式的端口电源控制的集线器，在它的一个端口（一个嵌入的端口）上永远都连接了一个设备的情况。如果用于这个嵌入的端口的端口电源控制屏蔽域指出组模式的电源开关已经屏蔽掉了，任何用来控制处于组模式的端口的集线器命令都不会影响这个嵌入的端口。

## 11.11 描述符

集线器描述符来自于通常的 USB 设备结构。集线器描述符定义了在集线器上的一个集线器设备和端口。主机可以通过集线器缺省管道来访问集线器描述符。

USB 规范（参见第 9 章）定义了下列描述符：

- (1) 设备；
- (2) 配置；
- (3) 接口；
- (4) 端点；
- (5) 字符串（可选）。

集线器类型定义了额外的描述符（参见 11.11.2 节）。另外，在 USB 设备的框架结构中还允许供应商专用的描述符的存在。集线器使用和第 9 章所定义的命令一样的标准的

USB设备命令。

### 11.11.1 标准描述符

在标准的 USB 描述符中,为集线器类型预定了特定的域。其它域要么是依应用而定的,要么就是不是应用于该类型的。

 **注意** 对于下列所给出的描述符和域,一个域内的比特是用 Little – endian 的方式来组织的;这就是说,在比特 0 位置是最有效位,而在比特 8 的位置是一个字节中的最高有效位。

#### 设备描述符(Device Descriptor)

bDeviceClass	= HubClass
bDeviceSubClass	= HubSubClass
wMaxPacketSize0	= 8 bytes

#### 接口描述符(Interface Descriptor)

bNumEndpoints	= 1
bInterface	= 1

#### 配置描述符(Configuration Descriptor)

MaxPower	= 在该配置中,这一集线器所消耗的总线电源供应的最大值
----------	-----------------------------

#### 端点描述符(Endpoint Descriptor) (用于状态变化端点)

bEndpointAddress	= Implementation dependent
wMaxPacketSize	= Implementation dependent
bmAttributes	= 方向 = 输入, 传输类型 = 中断方式 (0b00000111)
bInterval	= 0xFF (所允许的最大时间间隔)

集线器类型的驱动程序可以利用 GetDescriptor 设备请求来从主机的系统软件中检索一个设备的配置情况。根据规范的定义,GetDescriptor 设备请求所返回的第一个端点描述符是状态变化端点描述符。集线器还可以在这一类型定义所要求的最少的端点之外,定义额外的端点。但是,集线器对这一类型标准的配置总是将状态变化端点描述符作为标准接口内的第一个端点描述符而返回。

### 11.11.2 集线器描述符

除了在上一小节中提到的标准描述符之外,在 USB 规范中还有专门为集线器这一 USB 设备类型而定义的描述符。这些描述符由 bDescLength、bDescriptorType、bNbrPorts、wHubCharacteristics、bPwrOn2PwrGood、bHubContrCurrent、DeviceRemovable 和 PortPwrCtrlMask 等几个域组成。其中 PortPwrCtrlMask 域在集线器描述符中的偏移量是一个变量,这是因为 DeviceRemovable 域所占据的空间是随集线器所支持的端口数的不同而变化的。为了在主机和集线器之间实现通信,我们必须利用这些域来获得有关集线器的信息,如电源开关模式,所要求的电流和该集线器所支持的下行端口数等信息。因

此,我们有必要对其进行详细地介绍。由于集线器描述符比较复杂,所包含的域也比较多,所以我们将它的各个域都列在了表 11-9 中,以方便读者的查询。

表 11-9 集线器描述符

偏移量	域	尺寸	说 明
0	bDescLength	1	这一描述符中的字节数,包括该字节
1	bDescriptorType	1	描述符类型
2	bNbrPorts	1	该集线器支持的下行端口数
3	wHubCharacteristics	2	<p>D1..D0: 电源开关模式            00 —— 成组的电源开关(所有的端口同时通电)            01 —— 个别的电源开关            1X —— 无电源开关(端口总是在集线器通电时上电,并在集线器断电时断电)</p> <p>D2: 标示一个复合设备            0 —— 集线器不是一个复合设备的一部分            1 —— 集线器是一个复合设备的一部分</p> <p>D4..D3: 过电流保护模式            00 —— 全局过电流保护集线器将所有端点获得的电流总数作为过电流而予以报告,而不会分解出单个端口的过电流状态            01 —— 个别端口的过电流保护,集线器以每一个端口为基础报告过电流状态。每个端口都有一个过电流指示符            1X —— 无过电流保护。这一选项仅允许由没有实现过电流保护的总线供电集线器来使用</p> <p>D15..D5: 保留</p>
5	bPwrOn2PwrGood	1	从顺序对一个端口开始供电起,到该端口上的电源供应稳定下来所需要的时间(在 2ms 时间间隔之内)。系统软件利用这一数值来决定在访问一个上电后的端口之前,需要等待多长时间
6	bHubContrCurrent	1	集线器控制器电子设备所需要的的最大电流数,单位是 mA
7	DeviceRemovable	根据集线器上的端口数而变化	<p>指出一个端口上是否有一个可移动的设备接入了。如果一个不能移动的设备接入了一个端口,那么该端口就永远也不会收到一个插入变化指示。该域以字节为粒度而予以报告。在一个字节之内,如果某个位置没有用于一个端口,那么代表该端口特性的这一域就会复位为“0”</p> <p>比特定义:</p> <ul style="list-style-type: none"> <li>0 —— 设备是可移动的</li> <li>1 —— 设备是不可移动的(永远连接)</li> </ul> <p>这是一个对于集线器上的单个端口的位图:</p> <p>Bit 0: 保留供将来使用</p> <p>Bit 1: 端口 1</p> <p>Bit 2: 端口 2</p> <p>等等。</p> <p>Bit n: 端口 n (根据具体的应用,最多可以支持 255 个端口)</p>
变量	PortPwrCtrlMask	根据集线器上的端口数而变化	指出一个端口是否会受一个组模式的电源控制请求的影响。该域被置位的端口需要一个手动的 SetPortFeature(PORT_POWER) 请求来控制该端口的电源状态

(续)

偏移量	域	尺寸	说 明
			<p>比特定义：</p> <p>0 —— 端口未屏蔽掉组模式电源控制能力</p> <p>1 —— 端口不会受一个组模式的电源控制命令的影响。必须向该端口发送一个手动的命令来关闭和打开电源</p> <p>这是一个对应于集线器上的单个端口的位图：</p> <p>Bit 0: 保留供将来使用</p> <p>Bit 1: 端口 1</p> <p>Bit 2: 端口 2</p> <p>等等。</p> <p>Bit n: 端口 n (根据具体的应用, 最多可以支持 255 个端口)</p> 

## 11.12 请 求

### 11.12.1 标准请求

与前面我们提到的标准描述符一样, 对于集线器的访问, USB 规范也定义了一系列的标准请求, 表 11-10 列出了集线器对标准设备请求的响应。这些请求不仅可以用来访问一般的 USB 设备, 也可以用它来访问 USB 集线器。

当然我们在使用这些标准的请求来访问 USB 集线器时, 同样要遵循和访问一般的 USB 设备时一样的原则。

表 11-10 集线器对标准设备请求的响应

bRequest	集线器的响应
CLEAR_FEATURE	标准
GET_CONFIGURATION	标准
GET_DESCRIPTOR	标准
GET_INTERFACE	可选。集线器仅需要支持一个接口
GET_STATUS	标准
SET_ADDRESS	标准
SET_CONFIGURATION	标准
SET_DESCRIPTOR	可选
SET_FEATURE	标准
SET_INTERFACE	可选。集线器仅需要支持一个接口
SYNCH_FRAME	可选。集线器不需要具有同步端点

### 11.12.2 专用类型请求

集线器类型定义了所有集线器都必须响应的请求, 如表 11-11 所列。

表 11-11 集线器类型请求

请求	bmRequestType	bRequest	wValue	wIndex	wLength	数据
ClearHubFeature	00100000B	CLEAR_FEATURE	特性选择符	0	0	无
ClearPortFeature	00100011B	CLEAR_FEATURE	特性选择符	端口	0	无
GetBusState	10100011B	GET_STATE	0	端口	1	每个端口的总线状态
GetHubDescriptor	10100000B	GET_DESCRIPTOR	描述符类型和描述符索引	0 或语言 ID	描述符长度	描述符
GetHubStatus	10100000B	GET_STATUS	0	0	4	集线器状态和变化指示符
GetPortStatus	10100011B	GET_STATUS	0	端口	4	端口状态和变化指示符
SetHubDescriptor	00100000B	SET_DESCRIPTOR	描述符类型和描述符索引	0 或语言 ID	描述符长度	描述符
SetHubFeature	00100000B	SET_FEATURE	特性选择符	0	0	无
SetPortFeature	00100011B	SET_FEATURE	特性选择符	端口	0	无

表 11-12 列出了集线器类型请求代码。

表 11-12 集线器类型请求代码

bRequest	数 值
GET_STATUS	0
CLEAR_FEATURE	1
GET_STATE	2
SET_FEATURE	3
reserved for future use	4 - 5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7

下面是用于集线器类型的特性选择符,如表 11-13 所列。请参考 GetHubStatus 和 GetPortStatus 对特性的描述。

表 11-13 集线器类型特性选择符

	接收者	数 值
C_HUB_LOCAL_POWER	集线器	0
C_HUB_OVER_CURRENT	集线器	1
PORT_CONNECTION	端口	0
PORT_ENABLE	端口	1

(续)

	接收者	数 值
POR T _ SUSPEND	端口	2
POR T _ OVER _ CURRENT	端口	3
POR T _ RESET	端口	4
POR T _ POWER	端口	8
POR T _ LOW _ SPEED	端口	9
C _ POR T _ CONNECTION	端口	16
C _ POR T _ ENABLE	端口	17
C _ POR T _ SUSPEND	端口	18
C _ POR T _ OVER _ CURRENT	端口	19
C _ POR T _ RESET	端口	20

### 1. 清除集线器特性

该请求可以清除集线器状态内所报告的数值,如表 11-14 所列。

表 11-14 集线器状态内所报告的数值

bmRequestType	bRequest	wValue	wIndex	wLength	数据
0010000B	CLEAR_FEATURE	特性选择符	0	0	无

清除了一个特性之后,会禁用该特性,请参考表 11-13 中对特性选择符的定义。如果一个特性选择符对应于一个变化指示符,那么清除这个指示符会对这个变化作出应答。C\_HUB\_LOCAL\_POWER 和 C\_HUB\_OVER\_CURRENT 都可以利用该请求来对其作出应答。

### 2. 清除端口特性

该请求会复位在端口状态内所报告的数值,如表 11-15 所列。

表 11-15 端口状态内所报告的数值

bmRequestType	bRequest	wValue	wIndex	wLength	数 据
0010001B	CLEAR_FEATURE	特性选择符	端口	0	无

端口号必须是一个大于 0,并对该集线器来说是一个有效的端口号。

在清除了一个特性之后,会禁用该特性,请参考表 11-13 中对特性选择符的定义。如果一个特性选择符对应于一个变化指示符,那么清除这个指示符的操作就会对这个变化作出应答。利用该请求,可以对连接、激活、挂起、复位和过电流状态内出现的变化作出应答。

清除了 PORT\_SUSPEND 特性之后会在所说明的端口上产生一个由主机发起的重新开始操作。清除了 PORT\_ENABLE 特性会使该端口被禁用。清除了 PORT\_POWER 特性会根据由于主机开关电源所采用的方法而产生的限制,而使该端口断电。如果一个集线器使用了组模式的电源开关,在任一个端口实际断电之前,要求所有的端口都必须处于断电状态。

### 3. 获得总线状态

这是一个可以选择的用于对每个端口的诊断请求, 它会读取在最后一个 EOF2 抽样的总线状态值, 如表 11-16 所列。

表 11-16 获得总线状态

bmRequestType	bRequest	wValue	wIndex	wLength	数据
10100011B	GET_STATE	0	端口	1	每个端口的总线状态

端口号必须是一个大于 0, 并对该集线器来说是一个有效的端口号。

集线器必须实现一个可供选择的辅助诊断手段来简化系统的调试。集线器通过这一可选的请求来实现这个辅助诊断手段。这一诊断特性可以提供对在最后一个 EOF2 点抽样的总线状态的显示。

实现了该诊断特性的集线器应该将这一总线状态存放在每一个 EOF2 状态内, 从而为接下来的一个 USB 帧内可能的请求作好准备。

所返回的数据是按照下列方式进行比特映射的。D- 信号的值在比特 0 域内返回, 而 D+ 信号的值则在比特 1 域内返回。比特 2~7 保留以供将来使用, 并应该将这些比特设置为 0。

不支持该请求的集线器会用一个停止握手来响应。

### 4. 获得集线器描述符

该请求可以返回集线器描述符, 如表 11-17 所列。

表 11-17 获得集线器描述符

bmRequestType	bRequest	wValue	wIndex	wLength	数据
10100000B	GET_DESCRIPTOR	描述符类型和 描述符索引	0	描述符长度	描述符

用于集线器类型的描述符的 GetDescriptor 请求沿用了和标准的 GetDescriptor 请求一样的使用模式(参见第九章)。标准的集线器描述符由描述符类型 0 来指出。所有的集线器都要求实现一个具有描述符索引 0 这样的一个集线器描述符。

### 5. 获得集线器状态

该请求可以返回当前的集线器状态和从上一个应答开始后变化的状态, 如表 11-18 所列。

表 11-18 获得集线器状态

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_STATUS	0	0	4	集线器状态和 变化指示符

数据中的第一个字节包含了 wHubStatus(参见表 11-19)。而数据中的第二个字节则包含了 wHubChange(参见表 11-20)。

所返回的域是按照可以允许系统软件确定哪一个状态发生了变化的方式来组织的。在合适的地方, 位于 wHubStatus 和 wHubChange 域内的比特是按照一个接一个的方式来

进行安排的。

利用 ClearHubFeature 请求可以对本地电源和过电流变化作出应答,如表 11-19 和 11-20 所列。

表 11-19 集线器状态域, wHubStatus

比特	说 明
0	<p>本地电源状态:这是本地电源所处的状态。 该域仅适用于其 USB 接口引擎(SIE)是总线供电的集线器,或那些既可以支持自供电配置又可以支持总线供电配置的集线器。该域会作为集线器电源变化的结果而返回。它报告本地电源是否正常,并总是允许系统软件来确定从这一集线器所连接的设备上移走电源的原因,或对本地电源供应状态的变化作出反应。 如果集线器不支持这一特性,就要保留该域并遵循以下的保留比特的定义。 该域为 SIE 和集线器的其余部分指出电源状态 0 = 本地电源供应正常 1 = 本地电源供应丢失(无效)</p>
1	<p>过电流指示符:该域仅适用于以整个集线器为基础的,对过电流条件予以报告的集线器(在集线器描述符中报告)。 如果一个集线器不是就整个集线器来报告过电流情况,那么就要对该域进行保留并遵循以下的保留比特的定义。 该域指出经过所有的端口的电流总和已经超过了所说明的最大值,所有端口的电源都会被关闭。有关过电流保护的详细信息,请参考 7.2.1 第 2 小节 该域指出了由于所有端口的电流消耗而引起的一个过电流条件 0 = 所有电源都正常工作 1 = 在整个集线器上存在一个过电流条件</p>
2~15	保留 读取这些比特将返回 0 值

表 11-20 集线器变化域, wHubChange

比特	说 明
0	<p>本地电源状态变化: (C_HUB_LOCAL_POWER) 该比特与前面本地电源状态,即 wHubStatus 中的比特 0 相对应。该域仅适用于其 USB 接口引擎(SIE)是由本地电源供电(即自供电)的集线器,或那些既可以支持自供电配置又可以支持总线供电配置的集线器。该域会作为集线器电源变化的结果而返回。 如果集线器不支持这一特性,就要保留该域并遵循以下的保留比特的定义。 该域指出本地电源状态是否出现了变化 0 = 本地电源状态没有发生变化 1 = 本地电源状态出现了变化</p>
1	<p>过电流指示符变化: (C_HUB_OVER_CURRENT) 该比特与前面过电流指示符,即 wHubStatus 中的比特 1 相对应。该域仅适用于以整个集线器为基础的,对过电流条件予以报告的集线器(在集线器描述符中报告)。 如果一个集线器不是就整个集线器来报告过电流情况,那么就要对该域进行保留并遵循以下的保留比特的定义。 该域可以指出过电流指示符是否出现了一个变化。只有当出现了一个过电流条件时,该域才会被置位。(即系统软件对这一变化的应答不会使其它的变化被报告) 0 = 在过电流指示符上没有出现变化 1 = 在过电流指示符上出现了变化(即产生了过电流条件)</p>
2~15	保留 读取这些比特将返回 0 值

### 6. 获得端口状态

该请求可以返回当前的端口状态和从上一个应答开始后变化的状态,如表 11-21 所列。

表 11-21 获得端口状态

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	GET_STATUS	Zero	Port	Four	Port Status and Change Indicators

端口号必须是一个大于 0,并对该集线器来说是一个有效的端口号。

数据中的第一个字节包含了 wPortStatus(参见表 11-22)。而数据中的第二个字节则包含了 wPortChange(参见表 11-23)。

所返回的域是按照可以允许系统软件确定哪一个状态发生了变化的方式来组织的。在合适的地方,位于 wPortStatus 和 wPortChange 域内的比特是按照一个接一个的方式来安排的。

表 11-22 端口状态域, wPortStatus

比特	说 明
0	当前的连接状态: (PORT_CONNECTION) 该域反映了当前情况下,一个设备是否连接接入了这一端口。这一数值反映了该端口的当前状态,并且不会直接同使得插入状态变化(比特 0)置位的那个事件相对应 0 = 在这一端口上没有出现任何设备 1 = 一个设备出现在了这一端口上  注意: 对于连接了一个不可移动的设备的端口,该域始终为 1
1	端口被激活/禁用: (PORT_ENABLE) 端口仅能由主机软件来予以激活。而与之不同的是,端口却既可以由一个缺省条件(拆除事件或包括一个过电流指示在内的其他缺省条件)来禁用,也可以由主机软件来禁用 0 = 端口被禁用 1 = 端口被激活
2	挂起: (PORT_SUSPEND) 该域指出在这一端口上的设备是否被挂起了。将该域置位后,会使该设备挂起,而不能传播下行的总线通信流。复位该域会使这一设备重新开始工作。在一个总线处理期间,不能重新开始总线通信。如果是这一设备自己发出了一个重新开始信号的情况,该域就会被集线器所清除 0 = 未被挂起 1 = 挂起
3	过电流指示符: (PORT_OVER_CURRENT) 该域仅适用于以每个端口为基础的,对过电流条件予以报告的集线器(在集线器描述符中报告)。 如果一个集线器不是就每个端口来报告过电流情况,那么就要对该域进行保留并遵循以下的保留比特的定义。 该域指出端口上的设备所消耗的电流已经超过了所说明的最大值,该端口的电源都会被关闭。有关过电流保护的详细信息,请参考 7.2.1 节中的第 2 小部分。 该域指出了由于这一端口上的设备而引起的一个过电流条件 0 = 用于该端口的所有电源都正常工作 1 = 在该端口上存在一个过电流条件,会对该端口断电

(续)

比特	说 明
4	复位：(PORT_RESET) 当主机希望对接入的设备进行复位时，该域会被置位。它会一直保持置位状态，直到主机关闭了复位信号并且复位状态变化域被置位 0 = 没有声明复位信号 1 = 声明了复位信号
5~7	保留 读取这些比特将返回 0 值
8	端口电源：(PORT_POWER) 该域反映了一个端口的状态。由于集线器可以实现不同模式的电源开关，该域的含义会根据所使用的电源开关模式而有所不同。设备描述符会报告集线器所使用的电源开关模式。 主机不会向它的端口提供任何电源，直到这些端口进入了配置完成状态 0 = 该端口处于掉电状态 1 = 该端口处于通电状态 注意：不支持电源开关的集线器在该域内总是返回一个 1
9	接入低速率设备：(PORT_LOW_SPEED) 该域仅与一个设备是否被接入有关 0 = 全速率设备接入了这一端口 1 = 低速率设备接入了这一端口
10~15	保留 读取这些比特将返回 0 值

表 11-23 端口变化域, wPortChange

比特	说 明
0	连接状态变化：(C_PORT_CONNECTION) 指出在这一端口的当前连接状态出现了一个变化。即使系统软件已经清除了一个连接状态变化，集线器设备也会为该集线器设备连接状态的任一变化而将该域置位 <sup>①</sup> 0 = 当前连接状态没有出现变化 1 = 当前连接状态出现了一个变化 注意：对于接入了不可移动的设备的端口，在出现了一个复位条件之后，该域会被置位以向系统软件指出在该端口上出现了一个设备
1	端口激活/禁用变化：(C_PORT_ENABLE) 该域仅能在检测到了一个由于硬件变化而在端口激活/禁用状态中出现的变化时被激活 0 = 端口激活/禁用状态没有出现变化 1 = 端口激活/禁用状态出现了变化
2	挂起变化：(C_PORT_SUSPEND) 该域指出了在主机可见的接入设备电源状态中出现了变化。它说明了该设备已经脱离了挂起状态。而进入到挂起状态不会使该域置位。只有当整个重新开始过程结束之后，挂起变化域才会被置位。也就是说，该集线器在这一端口上中断了重新开始信号，并且允许该设备可以在 3ms 之后实现同 SOF 的再次同步 0 = 无变化 1 = 重新开始结束
3	过电流指示符变化：(C_PORT_OVER_CURRENT) 该域仅适用于以每个端口为基础的，对过电流条件予以报告的集线器（在集线器描述符中报告） 如果一个集线器不是就每个端口来报告过电流情况，那么就要对该域进行保留并遵循以下的保留比特的定义 该域可以指出每个端口的过电流指示符是否出现了一个变化 0 = 过电流指示符没有出现变化 1 = 过电流指示符出现变化

(续)

比特	说    明
4	复位变化: (C_PORT_RESET) 当一个端口上的复位操作结束时, 该域会置位。当复位处理结束后, 该域被复位时, 该端口的激活状态也会被置位, 而且挂起变化域会复位 0 = 无变化 1 = 复位结束
5~15	保留 读取这些比特将返回 0 值

① 举例来说, 如果在系统软件清除变化的条件之前所插入的状态发生了两次变化, 集线器硬件会设置一个已经设置了的比特(即该比特保持置位状态)。但是, 该集线器只有当集线器控制器向状态变化端点请求了一个数据传输时, 它才可以传送这个变化比特。在这种情况下, 系统软件将负责确定状态变化的历史。

## 7. 设置描述符

该请求将重写集线器的描述符, 如表 11-24 所列。

表 11-24 设置描述符

bmRequestType	bRequest	wValue	wIndex	wLength	数据
00100000B	SET_DESCRIPTOR	描述符类型和 描述符索引	0	描述符长度	描述符

用于集线器类型的描述符的 SetDescriptor 请求沿用了和标准的 SetDescriptor 请求一样的使用模式(参见第 9 章)。标准的集线器描述符由描述符类型 0 来指出。所有的集线器都要求实现一个具有描述符索引 0 这样的一个集线器描述符。

该请求是可选的。利用这一请求可以向一个专用的描述符内写入数据。在控制处理的数据传输阶段内, 主机会提供将要传送至集线器的数据。该请求会一次写完整个的集线器描述符。

集线器由这一请求所接收的所有字节, 以保证整个描述符已经成功地从主机传送过来。一旦成功地完成了该总线传输, 该集线器就会更新所指明的描述符所包含的内容。

不支持该请求的集线器可以用一个停止握手来响应。

## 8. 设置集线器特性

该请求可以设置在集线器状态内报告的数值, 如表 11-25 所列。

表 11-25 设置集线器状态

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	SET_FEATURE	特性选择符	0	0	无

设置一个特性后, 就会将该特性激活, 参见表 11-13 中对特性选择符的定义。利用该请求不会对指示符变化作出应答。

## 9. 设置端口特性

该请求可以设置在端口状态内报告的数值, 如表 11-26 所列。

表 11-26 设置端口特性

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	SET_FEATURE	特性选择符	端口	0	无

端口号必须是一个大于 0，并对该集线器来说是一个有效的端口号。

设置一个特性后，就会将该特性激活，参见表 11-13 中对特性选择符的定义。利用该请求不会对指示符变化作出应答。

设置端口 PORT\_SUSPEND 特性会使总线上的通信流量终止于该端口，并因此而挂起该设备。设置复位特性(PORT\_RESET)会使集线器在该端口上发出复位信号。当复位信号结束时，集线器会设置 C\_PORT\_RESET 变化指示符并立即激活该端口。有关对主机所发起的复位行为的完整讨论请参考 11.6.2 节。



# 第12章 USB产品开发和驱动程序设计

超星阅览器提醒您：  
使用本复制品  
请尊重相关知识产权！

本章将介绍一些与实际的开发工作联系。在这一章中，我们首先讨论在 Windows 我们会给出一些 USB 设备的设计实例。最后 (Windows WDM) 开发的有关问题。

## 12.1

在前面我们已经提到，由于发展，新的 USB 设备不断涌现。或 Windows 98 之后会出现那个主机系统的 USB 设备呢？

在下面几节中我们将讨

本章具体包括以下内容：



Windows 95 中的 USB 设备；



Windows 98 中的 USB 设备；



Windows 98 中的 IEEE 1394 设



USB 硬件产品开发；

### 12.1.1 Windows 95 中的 USB 设备

在最初推出 Windows 95 操作系统时,USB 技术还没有像今天这样成熟。因而当时也就没有为 Windows 95 所支持。但是,随着 USB 技术的日益成熟和完善,如今它已经是一种十分流行的外设接口方式了。而 Microsoft 也为 Windows 95 提供了一些补丁,从而使那些依然留恋于 Windows 95 操作系统的用户同样可以跟上潮流,在该操作系统环境下使用 USB 设备。

但是在安装通用串行总线驱动程序补丁之前,必须明确当前所运行的 Windows 95 版本为 950B。如果所使用的版本不是 950B,那么你的计算机就不能接受这些 USB 文件,安装工作会失败。

要检测所使用的 Windows 95 版本是否为 950B 也很容易,所采取的步骤是:

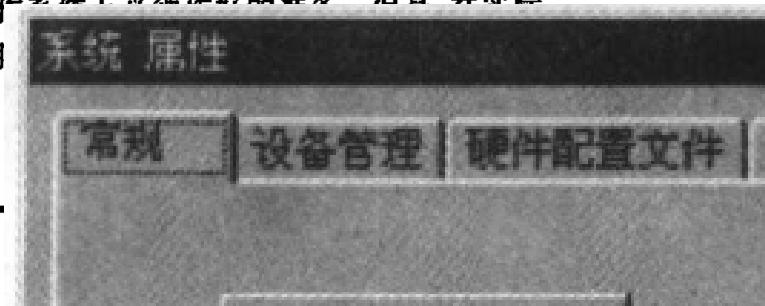
- (1) 打开计算机,进入 Windows 95 环境;
- (2) 在桌面上的“我的电脑”图标上点击鼠标左键;
- (3) 在弹出的菜单中选择“属性”选项;
- (4) 在系统窗口的菜单栏内选择“常规”选项。

这时我们就可以看到关于我们所使用的计算机的一些常规信息,如图 12-1 所示:

图 12-1 Windows 95 版本

如果显示的系统是 4.00.950B,则说明所使用的 Windows 95 版本是 950B。否则,就不是 950B 版。这时,用户就需要将他的操作系统升级到 Windows 95 B 版或 Windows 98 后,才能使用 USB 设备。

上面我们讲的是为了使用 USB 设备,在操作之前,首先必须要明确你所使用



USB 应用的硬件基础。只有在软件和硬件都可以支持 USB 设备的情况下,才可以使用 USB 设备。用户可以参考有关的主板说明书来确定其是否支持 USB 功能。

在 1998 年以前出现的一些主板可以支持 USB 总线协议,但是在主板上并没有提供 USB 接口(即 USB 根集线器的两个 USB 端口),因此还需要用户来参照主板说明来扩展 USB 接口。通常我们需要一根带有两个 USB 接口的 USB 电缆,然后找到主板上的 USB 线并与之连接,再将其引到机箱外即可。但是这时我们还不能使用 USB 设备。在使用 USB 设备之前,我们还必须激活 CMOS 中的 USB 功能(有关内容请参考主板说明)。当我们打开计算机电源后,在 Windows 95 环境中(95OB 版,并安装了 USB 驱动程序)采用如下步骤,就可以了解是否打开了 CMOS 中的 USB 功能。

- (1) 打开“开始菜单”;
- (2) 用鼠标点击“设置”选项;
- (3) 然后打开“控制面板”;
- (4) 在“控制面板”中找到“系统”并打开它;
- (5) 在“系统”窗口中选择设备管理器选项。

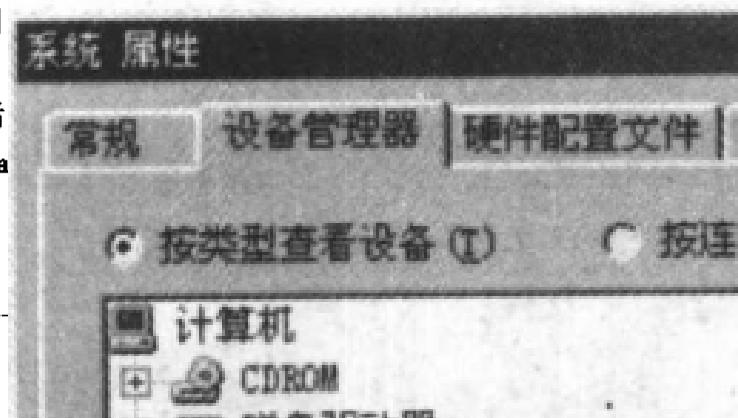
这时,我们就可以看到计算机系统的设备清单了,如图 12-2 所示。



图 12-2 Windows 95 设备管理器(无 USB 设备)

如果系统中的设备管理器如图 12-2 所示,没有任何 USB 设备,则说明还没有激活主板上的 USB 功能。为此,我们需要退出 Windows,然后在 CMOS 中选择适当的设置来激活 USB 功能。这时,重复上述步骤,我们会看到现了一些变化,如图 12-3 所示。

图中出现了“其它设备”的图标,打开该图标后 Universal Serial Bus”两个设备。其中“PCI Universa



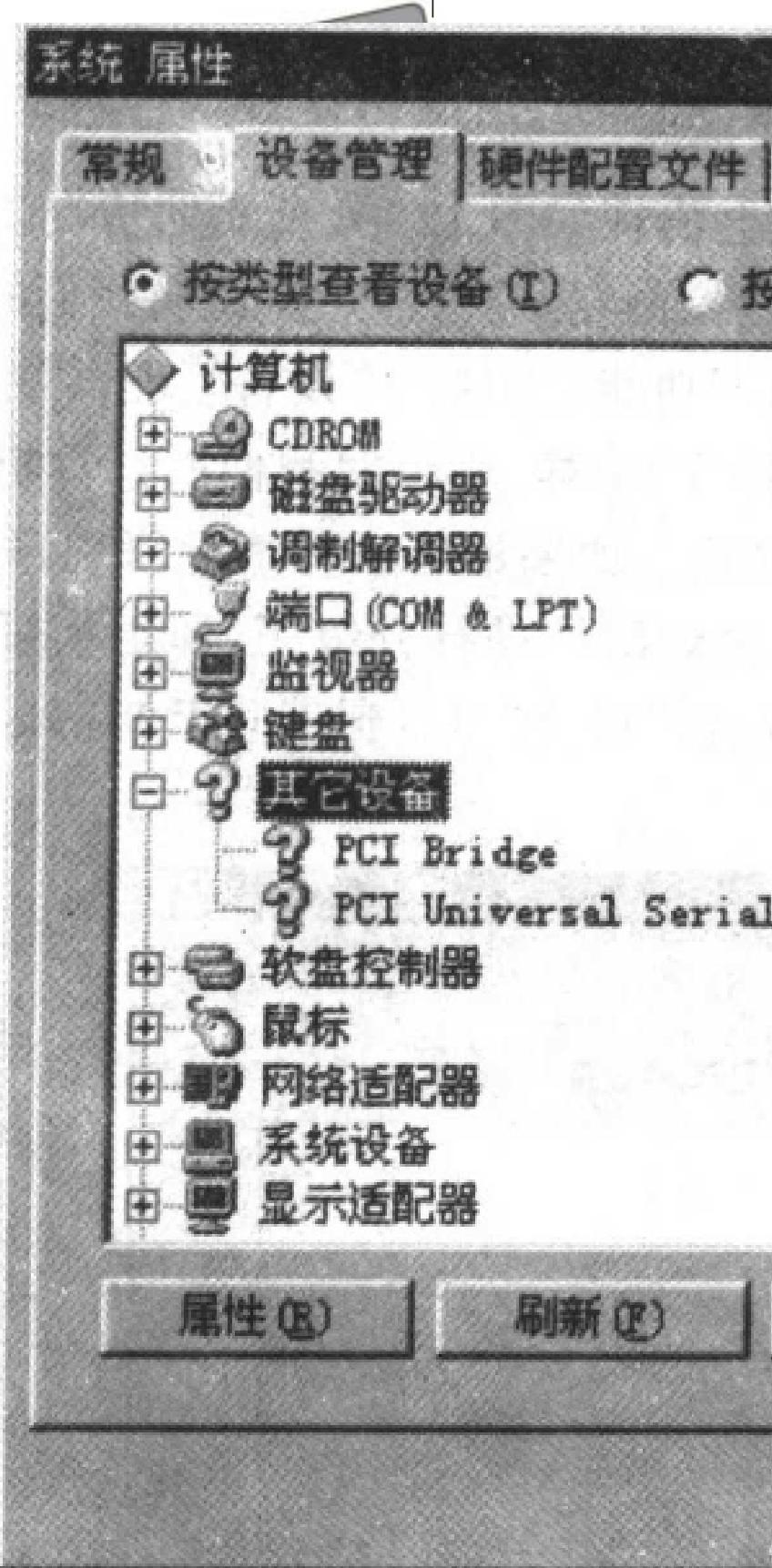


图 12-3 Windows

线的连接设备。这就说明我们已经成功地安装了一个 USB 设备的相应驱动程序，就可以使用了。

### 12.1.2 Windows 98 中的 USB

我们已经谈到了，Windows 98 操作系统在 Windows 98 环境下，使用 USB 的过程要简单的多。你可以在“设备管理器”中看到通用串行总线。以下有两种方法，具体的操作步骤如下：

#### 1. 方法一

- (1) 启动 Windows 98；
- (2) 打开“开始菜单”；
- (3) 用鼠标点击“设置”选项；
- (4) 然后打开“控制面板”；
- (5) 在“控制面板”中找到“系统”并打开；
- (6) 在“系统”窗口中选择设备管理器。

#### 2. 方法二

- (1) 打开计算机，进入 Windows 98 环境；
- (2) 在桌面上的“我的电脑”图标上点

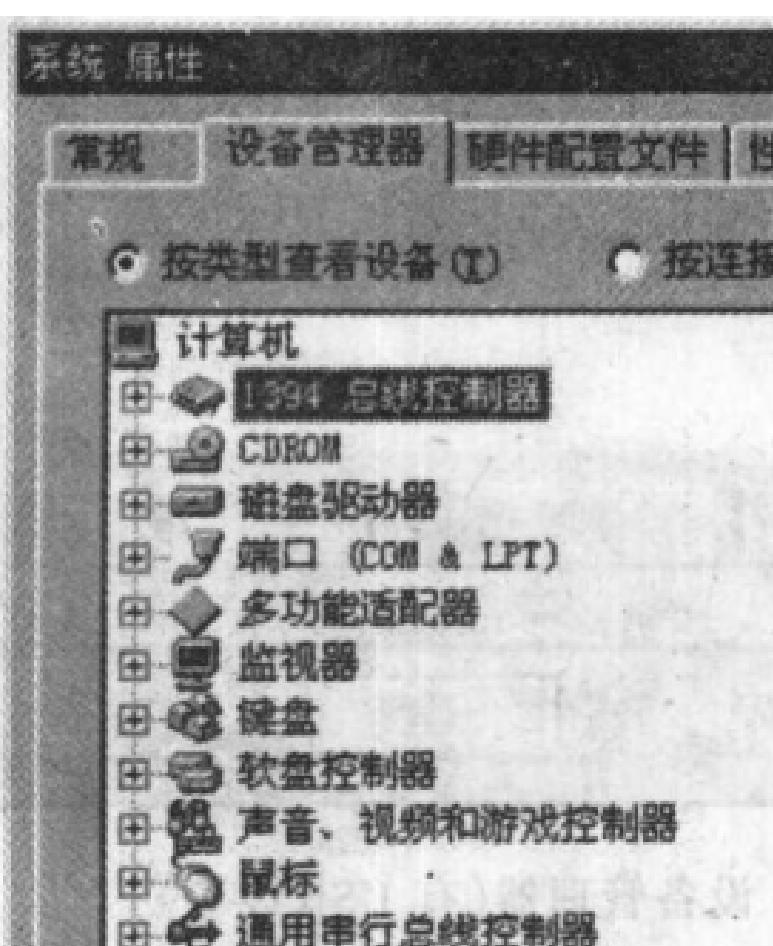
- (3) 在弹出的菜单中选择“属性”选项；
- (4) 在系统窗口的菜单栏内选择“设备管理器”选项。

如果已经启动了主板上的 USB 功能，就会出现如图 12-4 所示的情况。在图中出现了通用串行总线的专用图标和“通用串行总线控制器”的字样。这说明我们的计算机可以支持 USB 总线协议，而且我们已经打开了这一功能。如果在“设备管理器”中未出现通用串行总线的专用图标和“通用串行总线控制器”的字样，则说明我们的计算机不支持 USB 功能或者是还没有在 CMOS 中选择合适的设置。如果是后一种情况，我们可以采用上一节的方法来解决。如果是前一种情况，就必须对硬件进行升级了，即更换可以支持 USB 功能的主板。当我们打开“通用串行总线控制器”后，可以看到不同于 Windows 95 所显示的情况，如图 12-5 所示。

图 12-4 Windows 95 的“设备管理器”

在图 12-5 中，我们可以看到在我们的主机例说明 Windows 98 对 USB 设备的显示情况，又会有所差别）。“Intel 82371AB/EB PCI to USB 总线主控制器”，至于它的作用我们在前面已经注意根据具体的硬件环境，即主板所采用的芯片会不同。另外一个 USB 设备是“USB Root Hub”，它的两个 USB 端口。对于目前所使用的计算机，在 Windows 98 环境下，看到这一 USB 设备。

接下来，我们就可以在主机中接入 USB 设备了，因此在运行 Windows 98 的过程中，就可以接适配器那样只有在关闭了系统之后才可以运行。



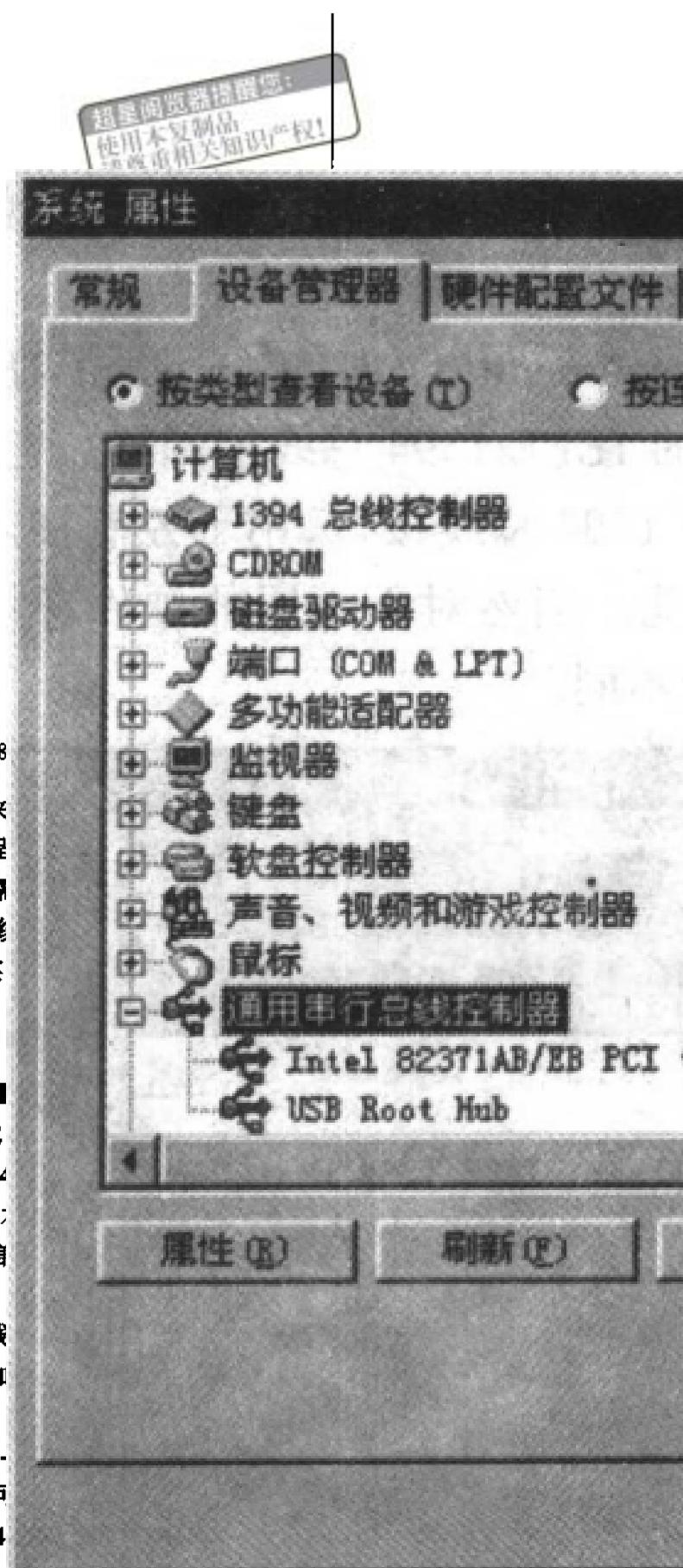


图 12-5 Windows 98

会自动检测到该硬件设备，并弹出一个窗口来  
备，并会提示用户安装或选择该设备的驱动程  
用户可以十分方便地完成这些操作。如果不  
运行的过程中，直接拔下该设备，而不会对系  
影响。由此可见，USB 总线技术的出现，确实  
单了，这也是 USB 技术提出的最初目的。

### 12.1.3 Windows 98 环境下的 IEEE 1394

在第 1 章中我们已经提到，USB 和 IEEE 1394 是两种串行总线技术。相对于 USB 技术而言，IEEE 1394 的传输速率高达 400Mb/s。因此，在实际应用中，USB 总线技术主要应用于像打印机、扫描仪和调制解调器等；而对于像实时的图像信号处理来说，就要用到 IEEE 1394 技术了。

鉴于 USB 和 IEEE 1394 这两种串行总线技术在 Windows 98 环境下的 IEEE 1394 设备进行简单的讨论，以加深读者对这两种总线技术的比较的认识。

细心的读者可能已经注意到了，在图 12-5 所示的“设备管理器”对话框中显示出了“IEEE 1394 总线控制器”的字样。这说明在我们的计算机系统中可能已经安装了 IEEE 1394 主控制器（目前主要是 1394

1394 主控制器集成在了主板上,但是应用非常有限)。当我们点击“1394 总线控制器”图标后,可以见到如图 12-6 所示的情况。

这是在系统管理器中出现了“Texas Instruments OCHI Compliant IEEE 1394 Host Controller”的 IEEE 1394 设备。它是系统中的 IEEE 1394 总线控制器。它的作用同 USB 总线主控制器的作用类似,也是主机的 IEEE 1394 总线接口,用于控制接入主机的 IEEE 1394 设备和主机之间进行交互,完成通信功能。当然对于不同的硬件环境,系统所显示的 IEEE 1394 总线主控制器的类型也会有所不同。



图 12-6 Windows 系统属性对话框

由前几章的内容,我们可以知道运行 Windows 98 的同时,插入一个 IEEE 1394 接口卡,系统会自动检测并安装驱动程序。这里需要注意的是,必须有相应的 BIOS 支持 IEEE 1394 总线协议的支持。而对于 Windows 98 第三方驱动程序的安装,建议参考 Microsoft 有关文档。

## 12.2

在设计一个 USB 设备之前,我们

计方案实现起来比较简单,还是使最后的产品具有低廉的价格,或者是提供良好的操作性能。这些都是我们在开始具体的设计之前所必须予以考虑的。在明确了设计的大前提之后,我们就要考虑实际的设计细节了。

我们要首先明确所设计的产品的 USB 设备类型。表 12-1 给出了主要的 USB 设备类型。正像我们在第 11 章中所介绍的那样,不同类型的 USB 设备,USB 规范定义了不同的专用请求和专用的类型描述符。集线器设备类在标准 USB 规范中给定。关于这些类型的完整说明,读者可以参考 USB 网站 [www.usb.org/developers](http://www.usb.org/developers) 上给出的完整说明。这里就不再一一介绍了。

一类很重要的 USB 设备是人工输入设备(HID),这些设备包括鼠标、键盘等。虽然这些设备不必非要用 USB 实现来实现不可,但这些设备可以和 USB 设备模型实现很好的匹配。Windows 中包括了系统驱动程序用于与 USB 设备打交道。有关驱动程序的设计,我们将在下一节进行详细的讨论。

如果 USB 设备有一个接口描述符,指出它属于 HID 类,Windows 会自动装入系统 HID 设备驱动程序和 USB HID 小驱动程序,并读取 HID 设备描述符。这样用户只需写一些客户驱动程序就可以实现同 HID 设备的交互。

表 12-1 主要的 USB 设备类

设备类型	设备举例	USB 设备类常量
显示	监视器	USB_DEVICE_CLASS_AUDIO
通信	调制解调器	USB_DEVICE_CLASS_COMMUNICATION
人工输入设备(HID)	键盘,鼠标	USB_DEVICE_CLASS_HUMAN_INTERFACE
海量存储器	硬盘驱动器	USB_DEVICE_CLASS_STORAGE
物理反馈设备	强制反馈游戏杆	USB_DEVICE_CLASS_PHYSICAL_INTERFACE
打印机		USB_DEVICE_CLASS_PRINTER
电源	不可中断电源	USB_DEVICE_CLASS_POWER
集线器		USB_DEVICE_CLASS_HUB
音频	USB 音箱,声卡	USB_DEVICE_CLASS_AUDIO

在设计一个 USB 设备时,我们还要考虑以下一些因素:

- (1) 需要同步通信方式还是批量通信方式。
- (2) 是否需要每隔几 ms 就利用中断方式进行查询。
- (3) 设备应使用总线供电方式,还是自供电方式。
- (4) 是否允许为了电源控制而挂起该设备。
- (5) 能否用低速率 USB 来支持该设备。
- (6) 该设备是否符合某一 USB 设备类型的特征,并使用 Windows 操作系统提供的类驱动程序(关于 USB 设备驱动程序将在下一节中介绍)。
- (7) 为了满足这些要求应该选用那些 USB 微控制器。

目前在市场已经出现了许多 USB 的评估板。这样,在我们设计一个 USB 功能设备时,就不需要从最初的测试设备作起。只要根据需要,购买一块合适的评估板就可以开始开发工作了。从而可以节省大量的时间,大大提高开发的速度和可靠性,进而缩短开发周期,赢得宝贵的商机。这里我们给出了目前比较常用的几种不同厂商的评估板,供广大读者参考:

- (1) Intel 公司的 930 和 931 型评估板。
  - (2) Anchor Chips 公司开发的基于 8051 单片机的评估板。在该评估板的包装之内, 提供了用于开发一个 USB 设备所需的工作代码, 其中包括了一个 USB 主驱动程序。
  - (3) SMC 和西门子公司也提供了基于 8051 单片机的解决方案。
  - (4) Philips 公司提供了用于 USB 设备开发的评估板和系列芯片。
- 除了评估板之外, 其它一些设备也会有助于我们的设备开发工作, 如:
- (1) USB 总线分析器, 用于总线信号、时序分析。
  - (2) USB 集线器, 用于连接多个 USB 设备。
  - (3) USB 仿真器, 用于在开发过程中, 仿真实际的 USB 微控制器所实现的功能。
  - (4) USB 鼠标、键盘、数字相机等 USB 设备(这些设备可以帮助我们直观地了解 USB 总线技术)。

以上是我们在开发一个 USB 设备硬件过程中所必需的硬件设备。在 Windows 操作系统中, 为了支持 USB 设备, 即使我们所设计的 USB 设备可以在 Windows 操作系统中得以应用, 还必须设计该设备的驱动程序。在此我们也给出设计 Windows 环境下的 USB 设备驱动程序所需要的软件环境, 至于对 Windows 设备驱动程序的实际问题, 我们将在下一节中介绍。

- (1) Windows 98 或 Windows 2000。
- (2) Windows 98 或 Windows 2000 设备驱动程序开发包(Driver Development Kit, 简称为 DDK)。
- (3) USB DDK。
- (4) Win32 DDK。
- (5) Visual C++。

以上我们就开发 USB 设备所需考虑的问题和 USB 设备类型, 以及开发所需的硬软件进行了简单的介绍, 下面我们将就一些实际的问题进行详细的介绍。

### 12.2.1 设计选择

要设计一个客户 USB 设备, 首先必须对 USB 规范有一个全面而且深入的了解。只有熟悉了 USB 规范的方方面面之后, 才能对不同的传输方式、端点类型、管道模型和 USB 所定义的状态有深刻的认识, 然后才能根据实际情况的需要, 选择不同的实现方法, 进行方案论证, 最终确定一个设计方案。

在分析和设计一个 USB 客户设备时, 我们通常要做如下考虑。

#### 1. USB 接口选择

通常, 我们可以采用四种办法来使单个 USB 设备对主机表现为多个设备: 复合、混合、综合以及公共类型。

(1) 复合设备: 对于一个 USB 系统来说, 一个复合设备看起来更像一个集线器。由一个复合设备所支持的单个设备, 在一个 USB 系统中就好像是接在一个集线器端口上的设备。唯一的不同之处在于, 一个复合设备上的设备是从电气特性上实现设备的连接或拆除的, 而集线器则是从物理上接入或拆除一根 USB 电缆而实现一个设备和主机的连接的。这就是两者的根本区别。

这种实现方式的好处在于 USB 规范已经提供对这种类型的定义(如前面我们提到的一样),并且已经有了全面软件支持。而缺点是需要一个额外的芯片电路来提供各个设备的内部连接,该芯片电路就相当于一个内部集线器。

(2) 混合设备:对于一个 USB 系统而言,一个混合设备是通过一个 USB 地址来使用代表不同设备的多个接口的。一个 HID 类型的设备需要两个端点:控制端点和中断输入端点。当我们实现了一个混合设备后,控制端点 0 要为所有的 USB 设备所共享,同时还要分配分离的接口来包含那些中断端点。

这种实现方式的好处在于 USB 规范已经提供对这种类型的定义,并且已经有了全面软件支持。而不足之处在于,要连接或拆除一个混合设备中的某个独立的设备,就必须得拆除该混合设备中的所有设备,并用定义了一个新的设备集的配置、接口和端点描述符来重新连接。

(3) 综合设备:综合设备仅由 HID 类型定义。在一个 HID 报告描述符(Report Descriptor)中,可以通过声明多个最高层应用集合来创建一个综合设备。这种实现方式与混合实现方式相类似,不同的是在一个综合设备内所定义的所有设备都要共享同一个中断端点。

这种实现方式的好处在于 USB 规范已经提供对这种类型的定义,并且已经有了全面软件支持。而不足之处在于,要连接或拆除一个综合设备中的某个独立的设备,就必须得拆除该综合设备中的所有设备,并用定义了一个新的设备集的新报告描述符来重新连接这些设备。另外,综合设备还有一个缺点,那就是利用这种实现方式我们只能支持 HID 类型的设备。

对于一个复合设备而言,供应商需要提供一个集线器,并为每一个 USB 功能模块提供一个 SIE,而且每个 SIE 至少要有两个 USB 端点(接收和发送串行器和数据 FIFO 缓冲区),当然还要有一个地址识别逻辑。而混合设备则需要提供一个为所有设备所共享的控制端点 0 和一个单一地址的识别逻辑。而一个综合设备则仅需实现两个端点和单个地址识别逻辑。前两种实现方式的成本可能会比较高。而且混合设备和综合设备都不能提供动态接入功能。

(4) 公共类型设备:公共类型设备定义了一个特性集合,包括动态接口和共享的端点。这些特性可以为我们设计一个 USB 设备提供灵活的通用解决方案。

使用普通类型来实现一个 USB 设备,需要为每个它所支持的设备提供一个动态接口。另外还要提供一个静态接口来向 USB 系统指出动态接口中的状态变化。公共类型的设备定义了接口如何来共享端点。而且共享端点可以允许使用打包后的分组来传送数据(即来自于多个接口的数据负载被放置在同一分组内)。通过这种将来自于多个接口的数据混合在同一 USB 分组之内的方法,可以提高数据传输时 USB 总线的利用率。这里我们有必要解释一下共享端点。

一些设备的设计者会发现他们需要许多端点和管道,这意味着要占用更多的系统资源。基本 USB 标准指出,只有缺省管道(对应于端点 0)才可以在不同的设备接口之间实现共享。

共享端点 USB 扩展允许几个逻辑管道在一个常规“物理”管道上传送。这样的共享管道必须有相同的传输类型(控制传输、中断传输、批量传输或同步传输),并朝着相同的

方向传送。每个逻辑管道为发送的数据定义了一个或多个逻辑数据包,从而可以为多个逻辑管道传送数据。

处理共享管道的主机和设备在物理上复用(multiplex)来自于每个逻辑信源发出的数据。对于同步管道而言,合并的最大分组尺寸必须可以放入一帧之内。软件则必须保证一个逻辑管道上的挂起操作不会停止传输另外一个管道上的数据流。

流量控制可以使用与数据传输方向相反的管道来实现,在目标提供一个允许发送的信号之前,信源不能在逻辑管道上发送数据,但是它可以接收逻辑数据包的数量。

这种实现方式的优点是可以利用一个物理设备来提供对多个设备的支持(注意与集线器是两个完全不同的概念),支持共享端点使得 USB 总线利用率获得提高。

## 2. 端点类型选择

不同类型的端点有不同的特征。控制端点由消息或者主机发送的请求组成。中断传输、批量传输和同步传输为流管道。

通常只有一个控制管道用于端点 0。除了传送 USB 配置和配置标准传输之外,还可以将这个管道用于类定义的或用户自己的厂商定义的请求。如果需要的话,还可以定义更多的控制管道。

中断管道通常用于用户输入数据,但也可以从设备接收产生速率比较低的少量数据。中断管道可以与同步管道一起使用,用来提供反馈信息,使得数据速率可以是变化的。

定期产生或消耗数据抽样值的设备将要使用同步传输。同步传输的长度可以是变化的,并优先于批量传输,因为这样可以节省 USB 总线带宽。同步传输不会进行差错检测,但接收端可以通过检查帧标号来确定是否出现了传输错误。应用程序或设备必须自行解决在传输过程中可能出现的数据传输错误。

批量传输工程要检查数据错误,但是它的优先级最低,只有在总线有可用带宽,并且没有其它传输类型等待总线传输时使用。因此,批量传输适用于突发性比较强的数据传输场合。

低速率设备除了端点 0 之外,只能再支持两个端点,而全速率设备最多可以有 16 个输入端点和 16 个输出端点。一个端点会检查最大的分组大小,从而确定与之对应的管道类型是否可以接受这些数据。

## 3. 热插拔和设备枚举

USB 设备应该能在任意时刻接入到一个 USB 系统当中。而 USB 协议则控制了对一个 USB 系统中的所有设备进行枚举,并向系统指出一个新的 USB 设备已经接入到了系统之中,等待主机对其进行配置。对一个设备的接入检测,是通过定期查询集线器端口的状态变化来实现的。这在我们设计一个 USB 设备时必须予以考虑。

## 4. 地址

USB 提供了两种方法来识别 USB 设备:序列号和端口地址。尽管序列号是可选的,但是引入它的原因主要是因为它总是同设备一起提供的。如果没有为设备提供序列号,那么我们就必须使用端口地址。应用端口地址,只要一个设备同主机连接的总线拓扑结构没有发生变化,那么在系统每次进入工作状态时它总是会收到同样一个地址。

## 5. 电源管理和唤醒

我们在设计一个 USB 设备时,首先要明确该设备是由总线供电,还是采用自供电方

式。这要根据设备的用电量,以及总线供电所能提供的最大单位电流(参见第 7 章)来作出决定。另外,USB 规范还支持三种电源工作模式:上电、挂起和掉电。USB 规范可以允许一个 USB 被置为挂起状态,但它仍然可以保持唤醒一个 USB 系统的能力。对于一个设备,还应该确定是否使其实现远程唤醒功能。



## 6. 自举设备支持

在前面我们提到的公共类型的 USB 设备时,我们还要决定是否应支持自举设备。通常情况下,这以为着要使用一个固定的,低开销的方法来使一个 BIOS 可以识别和使用一个自举设备。

在设计 USB 设备的过程中,选择同步传输类型是一个十分富有挑战性的决定。因为信源会产生规则数目的采样值;而信宿则必须接收这些样值,通过适当的处理过程来产生相关的输出。在同步通信方式中,实现一系列的设备和进程的同步是非常复杂的问题。因为每个设备的工作频率,甚至是主机的内部工作频率都可能出现偏差或抖动。即使在名义上应该完全相等的两个时钟信号,由于信道时延和噪声的影响,也会不可避免地会出现抖动,从而使两者产生差别。

处理同步工作方式中的这些问题的方法很多,如采用速率自适应和变换技术。一种常见的方法是把采样组合成服务间隔,并通过适当的缓冲操作来缓冲这些样值。如果没有充足的样值出现,可以通过产生新的样值来进行填充。另外一方面,如果到达的样值太多,所采用的缓冲区不能装载这些数据,这时就只能丢弃一些数据了。当然,这会对数据的完整性产生影响。因此,我们要设计一个合理长度的数据缓冲区,从而将这种情况出现的概率降至最低限度。该技术要求为一个服务时间间隔缓存一组样值,每个速率同一个进程相适应。

另外,我们也可以采用将所有的设备都同步到 1kHz 的 USB 帧时钟的方法来解决不同设备的同步问题。正如前面我们已经讲到的一样,可以通过每一帧中的帧开始(SOF)开始信号来提供所有设备所需的同步信息。采用这种方法之后,即使所接收到的样值数不同,信源仍然会在每一帧中产生正确数目的样值数据。如果没有检测到 SOF 信号,或 SOF 信号遭到了破坏,设备也必须采用一定的操作来保证其同步方式不会被破坏。详细的内容请参考本书第 5 章。

一个自适应设备可以同主机进行交互,告诉主机它要发送或接受多少样值。它还可以利用一个中断管道来告诉主机,如何来调整它的抽样频率。但是,我们要注意,在一个 USB 系统中,一次只能允许一个 USB 设备获得对帧时钟频率的控制权。即使在一个 USB 系统中可能同时存在着多个 USB 自适应设备,但是在其中一个自适应设备获得了对帧时钟频率的控制权之后,其他的自适应设备都必须工作在这一帧时钟频率之下,直到它自己获得了对帧时钟频率的控制权。

(1) 帧长度调整:可以采用使设备获得 USB 帧频率的控制权来实现同步。SOF 的控制者可以通过对 USB 时钟进行微量的调整来实现和它的内部时钟相同步。每一帧中,一个控制者只能改变一个全速率比特周期。

(2) 模式:同步传输的最后一个可用的方法是让每个传输的大小以一定模式进行变化,以匹配信源数据速率。主机和端点之间必须就传输模式达成一致。在第 5 章中我们已经举出了一个应用实例(参见 5.10 节),这里我们再回顾一下。

音频信号可以使用 44.1kHz 的抽样频率,这意味着每秒要产生 44100 个 16 位的样值。对于每个 USB 帧来说,它要包含 44.1 个样值,可见这并不是一个整数。因此必须采用适当的传输模式来传送这些数据。例如每一帧中可以传送 45 个样值,同时要有一个特殊的标志来指示出哪一帧中有 45 个样值,而不是有 44 个样值。在这种情况下,可以使用每 10ms 就重复一次的传送模式。该模式由 9 个具有 44 个样值的帧和一个具有 45 个样值的帧来组成。这样平均起来,每一帧中都有 44.1 个样值。从总体上看,正好满足我们的要求,即每秒 44100 个样值。

### 12.2.2 USB 设备实现举例

图 12-7 给出了一个 4 端口 USB 集线器的模块。其中一个主芯片是集线器控制器,用来提供整个 USB 集线器的 5 个端口,其中端口 1 是用于和主机相连的根端口(上行端口),而端口 2~5 则对应于该集线器的 4 个下行端口,即这 4 个端口可以用来连接 USB 设备。

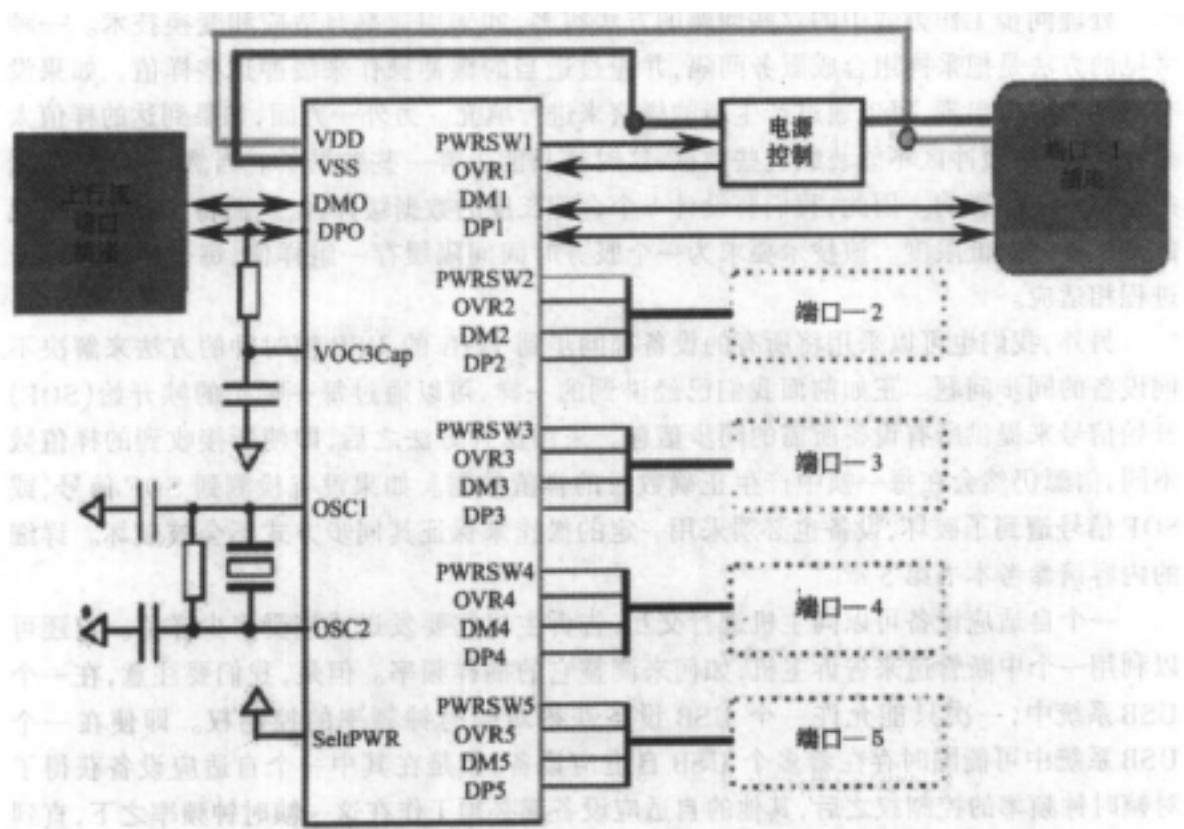


图 12-7 4 端口 USB 集线器

图 12-8 中所示为一个通过 USB 接口来同主机通信的一个 SCSI 扫描仪的模块图。图中“D12”代表 Philips 公司生产的一块型号为 PDIUSBD12 的接口芯片。该芯片是一块具有集成的 SIE、FIFO 存储器、发送器和电压调整器的高性能 USB 接口芯片。它通常应用于一个基于微控制器的系统中,并且可以通过高速的并行接口和系统中的微控制器进

行通信,其中并行接口速率可以达到 2MB/s。对于这个微控制器而言,PDIUSBD12 仅仅是一个具有 8 位数据总线和一个地址比特的存储设备。

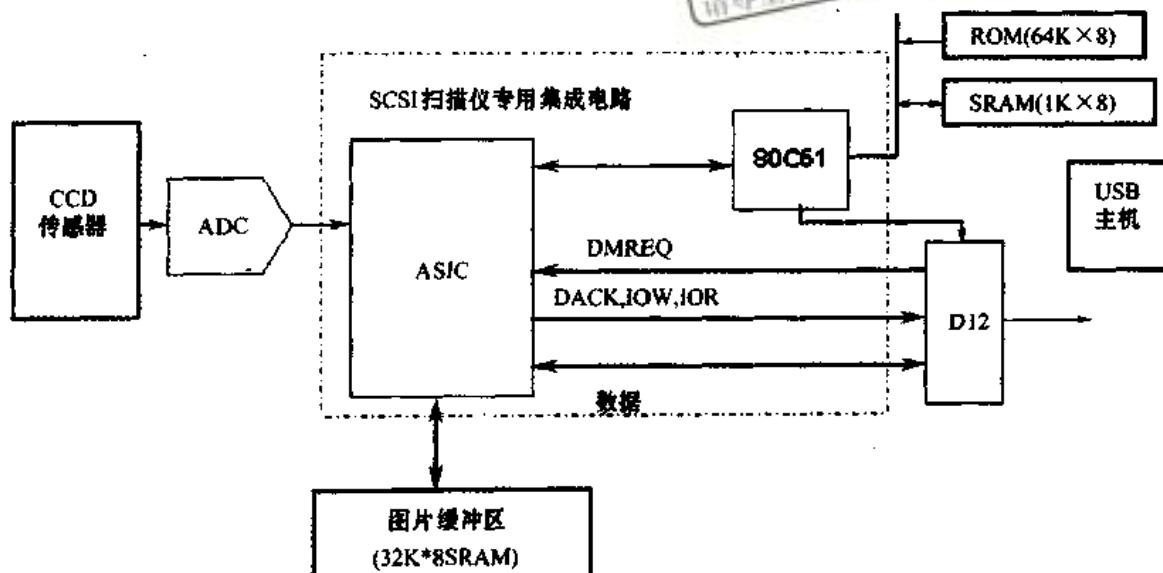


图 12-8 USB 扫描仪实现举例

芯片中的集成的 SIE 用来实现 USB 协议层的完整功能。这些功能包括:同步方式识别,并/串转换、比特填充/解填充、CRC 校验/生成、PID 确认/生成、地址识别和握手信号的鉴定/生成等。也就是说,利用该芯片可以完成微控制器所送出的信号到符合 USB 规范的信号的转换。

利用这种模块化的解决方案来实现一个 USB 接口,可以允许设计者从市场上的多种微控制器中选择一种合适的产品来使用。这种灵活性可以通过使用现有的体系结构来降低开发费用并减小固件投资。另外还可以缩短开发周期。

如图 12-8 中采用了一个“80C51”的微控制器。由于 80C51 是应用非常广泛的一种单片机控制器,在其基础上开发了大量的软件包。读者对其的应用也比较熟悉。因此,用户可以利用其熟悉的 80C51 进行核心程序的设计,即用 80C51 来实现其所需的功能。至于如何将数据信号转变为符合 USB 规范的信号,则完全由 PDIUSBD12 来完成。这样一来,既能利用 80C51 中丰富的软件资源,又可以利用 USB 总线技术的即插即用、热插拔等特性。所以开发设备的成本可以降低,周期也可以缩短。

图中所示的就是利用该芯片和一块 80C51 微控制器,将一个符合 SCSI 标准的扫描仪转变成一个符合 USB 总线规范的设备。这种解决方案在对现有硬件进行最小改动的情况下,实现了到 USB 技术的过度。

当然,在其它一些应用条件下,用户不可能为了实现一个简单的功能,再去使用一个微控制器。对于这种情况,许多厂家也提供了支持 USB 规范的芯片。用户可以使用这些厂家所提供的指令集来对其进行编程,从而实现自己所需要的功能。目前,这些芯片的价格有的已经降得很低了,完全可以提供很高的性能价格比。所以,在设计一个 USB 设备之前,应该多参考一些厂家的解决方案,根据实际需要,权衡利弊后再做决定。这里仅给出一些设计实例,以供读者参考。

### 12.3 USB 设备驱动程序设计

在本节中,我们讨论 USB 结构是如何同 Windows 操作系统相适应的。有关 Windows 对 USB 支持的更详细的说明,请参考 Microsoft 提供的设备驱动程序开发包(Device Driver Developer's Kit,即 DDK)中的文档。设计 Windows 98 环境下的设备驱动程序是一件十分复杂的事情,有关的内容完全可以写成几本书。因此在这里我们不可能对其进行十分细致的介绍。所以,在此我们仅讨论同 USB 设备的驱动程序有关的一些内容,当然我们也会介绍一些有关 Windows 设备驱动程序的基本内容。

USB 设备驱动程序必须遵循由 Microsoft 为 Windows 98 及其以后版本所定义的 Win32 驱动程序模型。这些驱动程序就是 WDM(Windows 设备驱动程序模型),它的扩展名为 .sys(其它文件类型也可以使用 .sys 扩展名)。

与其它的底层驱动程序一样,一个 WDM 驱动程序没有应用程序可以使用的功能,这是因为它负责在一个特权层实现和操作系统的通信。一个 WDM 驱动程序可以允许或否定一个应用程序对一个设备提出的访问。例如,一个游戏杆的驱动程序可以允许任一个应用程序来使用一个游戏杆,或者它可以为某个应用程序而保留以供其独占使用。Windows 为其它底层设备驱动程序和 WDM 驱动程序所保留的能力还包括对硬件中断的响应和 DMA 传输。

Win32 设备驱动程序模型是设计用来为运行在 Windows 98 和其以后版本(包括 Windows 2000 系统)下的任一设备,提供一个通用的驱动程序模型。

WDM 定义了一个基本模型,处理所有类型的数据。例如,USB 类驱动程序为所有 USB 设备提供了一个抽象的模型,并具有由所有客户驱动程序使用的定义好的接口。有了对所有设备类型共同的核心驱动程序模型,使驱动程序开发人员更容易从一种类型的设备转移到另外一种类型的设备上去。而且它也意味着驱动程序模型的内核实现尽可能地是固定的。如果每一种类型的设备都具有不同类型的驱动程序模型,那么我们的开发工作无疑会变得很繁琐。

早期的 Windows 版本使用了不同的驱动程序模型。Windows 95 使用的是 VxD(虚拟设备驱动程序)。Windows NT 4.0 则使用的是称之为内核模式驱动程序的设备驱动程序。如果用户要支持 Windows 95 和 Windows NT 4.0,那么它必须要为每一个操作系统开发一个设备驱动程序。但是,WDM 设备驱动程序却可以在 Windows 98 和 Windows 2000 下运行。

在 Windows 98 系统环境下的 USB 设备驱动程序是一个 WDM 驱动程序。尽管 Windows 98 可以继续支持 VxD(还包括 DLL 文件中所包含的驱动程序),但是 USB 设备必须使用 WDM 类型的设备驱动程序,这是因为这些设备必须同系统总线驱动程序进行通信,即我们在下面所要讲到的 USBD 驱动程序。

Win32 设备驱动程序模型并不是一个全新的概念。它实际上是一个 Windows 95 和 NT 特性的混合。一个 WDM 驱动程序是一个 NT 内核态驱动程序再加上 Windows 95 的即插即用和电源管理特性。Windows 95 的最后版本(OSR 2.1 和更高版)也可以支持 WDM 驱动程序。但是这些版本的操作系统并没有为零售用户提供,而仅用于那些 PC 厂

商在销售新机器时一同发送。在 Windows 98 系统中,对 WDM 的支持得到了进一步的扩充和改进,并呈现出完全不同的特性。

Win 32 驱动程序有两种工作模式,即内核模式和用户模式:

(1) 内核模式:当 CPU 运行于内核模式时,一切都可以运行,即所运行的程序没有任何操作系统的限制。任务可以运行特权级指令,对任何 I/O 设备有全部的访问权,还能够访问任何虚抵制和控制虚拟内存硬件。这种模式对应于 Intel 80x86 上 0 级环。

(2) 用户模式:在这个模式中,硬件可以防止特权指令的执行,并进行内存和 I/O 空间的引用的检查。这就操作系统限制任务对各种 I/O 操作的访问,并捕捉违反系统完整的任何行为。在用户模式中,运行的代码如果不通过操作系统中的某种门机制,就不能进入内核模式。在 Intel 80x86 处理器上,该模式对应于 3 级环。

Windows 系统提供了 PCI, PnPISA, USB 和 IEEE 1394 总线驱动程序。PCI 总线驱动程序枚举和配置 PCI 总线上的设备。PnPISA 总线驱动程序对可以使用即插即用配置的 ISA 设备进行枚举和配置。USB 总线驱动程序则用于枚举和控制 USB 总线。主控制器驱动程序是访问两种类型的 USB 设备的标准。USB 客户驱动程序可以通过使用 IOCTL,由 USB 类驱动程序来访问 USB 设备。IEEE 1394 总线驱动程序枚举和控制 IEEE 1394 高速总线。

USB 是使用标准 Windows 系统 USB 类驱动程序访问 USBDI(Windows USB 驱动程序接口)的 USB 设备驱动程序。USBD.sys 就是 Windows 系统中的 USB 类驱动程序,它使用 UHCD.sys 来访问通用的主控制器接口设备,或者使用 OpenHCI..sys 访问开放式主控制器接口设备。USBHUB.sys 是根集线器和外部集线器的 USB 驱动程序。在 PCI 枚举器发现了 USB 主控制器之后,它会自动装入相关的驱动程序。这也就是本章第一节中所说明的,只要我们启动了主板 CMOS 中的 USB 功能,就会在设备管理器中见到“通用串行总线”的标记和字样。下面我们就来介绍 USBDI。

### 12.3.1 Windows USB 驱动程序接口

系统 USB 驱动程序处理连接 USB 设备的大多数繁杂的工作。事实上,一些 HID USB 设备(如 USB 键盘、鼠标和游戏杆等设备)是可以被 Windows 自动识别的,不需要用户再编写额外的设备驱动程序。

但是,大多数的 USB 设备需要由用户来编写设备驱动程序与设备打交道,并响应内核态或用户应用程序的请求。在内核级,命令由客户驱动程序使用内部 IOCTL 发送给 USB 系统,例如 IOCTL\_INTERNAL\_USB\_SUBMIT\_URB 允许发出 USB 请求块(URB)给系统 USB 驱动程序。URB 允许发出几个功能调用给 USB 系统。

用户态 USB 实用程序也可以发出几个普通 IOCTL 给 USB 设备,目的仅仅是得到连接设备的信息。

在讨论如何使用这个 USB 驱动程序接口之前,我们先来介绍一下 USB 软件体系结构:

#### 1. USB 软件体系结构

如图 12-9 所示,Windows 对构成一个 USB 主机的不同软件部分进行了十分清楚的划分。其中 USB 客户软件仅仅包含了用来控制不同的 USB 外设的设备驱动程序。USB

客户软件会通过一个 Windows 所定义的一个软件接口来同根集线器驱动程序进行通信。而 USB 根集线器驱动程序则要通过 USBDI(通用串行总线驱动程序接口)来实现同通用串行总线驱动程序(USBD)的通信。然后,USBD 会选择两种主控制器驱动程序之一来同其下方的主控制器进行通信。最后,主控制器驱动程序会直接实现对 USB 物理总线的访问(通过 PCI 枚举器软件)。

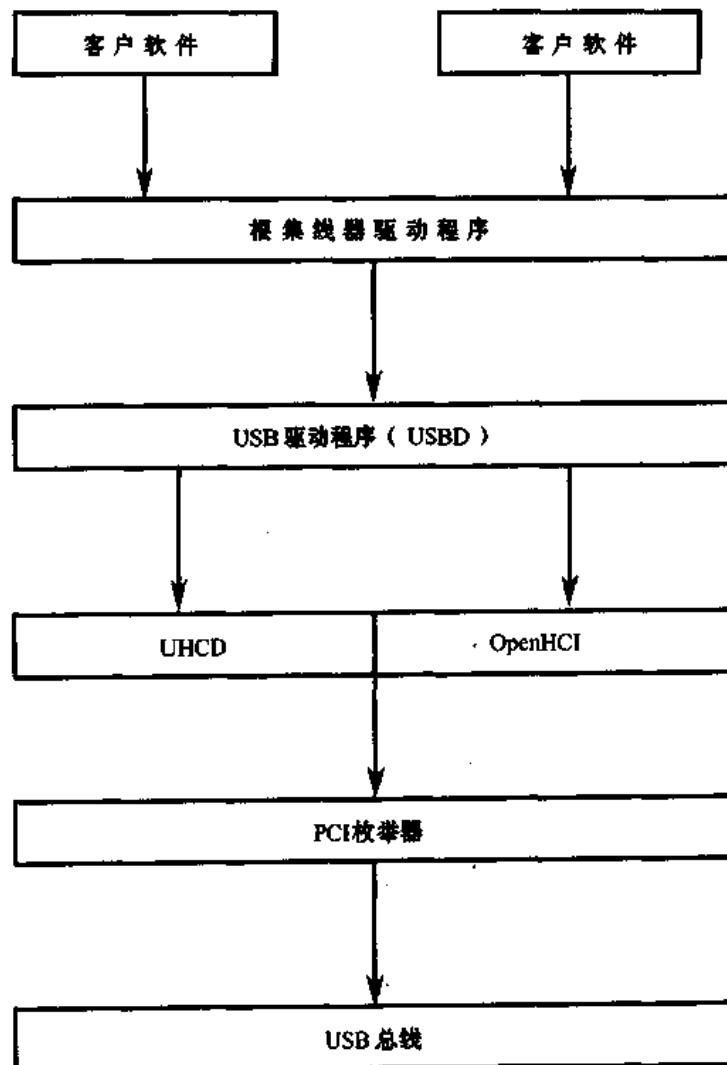


图 12-9 USB 驱动程序体系统结构

像 Windows NT 环境下的设备驱动程序一样,USB 驱动程序软件堆栈内的层间通信也使用了一个称之为 IRP(L/O 请求分组)的机制。一个设备驱动程序要和另外一个设备驱动程序进行通信时,它会用 IRP 结构来组装一个请求,并把该请求传递到下一层上去。下一层正确接收了该请求之后,会向发送方发出一个应答(ACK)握手信号。重复该过程,就可以在不同分层结构之间实现通信了。

虽然不同的软件分层通信时要使用 IRP,而实际的 USB 请求却是存放在一个被 Microsoft 称之为一个通用串行总线请求(URP)的结构内。在 URP 内包含了不同的域,在不同的软件分层结构中,每一个分层会使用不同的域来进行处理操作。

## 2. USBD

通用串行总线驱动程序(USBD)是 USB 系统中负责管理通用串行总线的工作,位于主机上的一个软件。USBD 负责控制所有的 USB 协议操作和高层的中断处理控制。在 Windows 98 中,Microsoft 定义了一个新的设备驱动程序模型,称之为 Windows 设备驱动程序模型(Windows Driver Model 或 WDM)。

WDM 最初是由 Microsoft 在 1997 年提出的,目的是在不同版本的 Windows 操作系统中,为一定的设备类型提供一个持续的、通用的设备驱动程序模型。实际上,为 Win9x 平台开发的 WDM 设备驱动程序无须改动,就可以在 Windows NT 4.0 环境下使用。这也是 WDM 的特点之一。

### 3. UHCD 和 OpenHCI

这是不同的 USB 供应商提出的用于管理一个系统中的 USB 主控制器的两个软件接口。Microsoft 提供了对两种接口的支持。就是在图 12-9 中所指出的通用主控制器驱动程序(Universal Host Controller Driver,或简称为 UHCD)和开放主控制器接口(Open Host Controller Interface,或简称为 OpenHCI)。

主控制器驱动程序(用来控制总线)通过主控制器来实现和物理总线的通信。Windows 在 USB 总线和主控制器之间又放置了一个 PCI 总线枚举器。在系统中,由 PCI 总线枚举器负责在其检测到系统中存在一个通用串行总线时,装载适当的 USB 系统软件。这就是我们在第一节中所介绍的“PCI to USB Host Controller”的作用。

### 4. 根集线器驱动程序

从图 12-9 中,我们可以看到主控制器驱动程序处于 USBD 的下方,而根集线器却位于 USBD 之上。在一个支持 USB 系统的主机中,都会有一个根集线器来提供两个 USB 端口,即位于主机箱上的两个 USB 端口。一个 USB 系统中的所有 USB 设备,包括 USB 功能设备和 USB 集线器都必须以一定的方式(直接接入或通过 USB 集线器以菊花链式的星型接入)接入一个 USB 根集线器。在 Windows 环境中,所有的客户驱动程序都可以通过指向这一根集线器驱动程序的接口来和所有接入的 USB 设备进行通信。另外还有一种方法就是通过 USBDI 而直接同 USBD 进行交互。

### 5. USB 客户

USB 客户软件也是一种设备驱动程序,它位于根集线器驱动程序之上。它通过定义的一个称之为 USB 接口的层间接口来访问其下方的 USB 软件。

在表 12-2 中我们给出了 Windows USB 接口所提供的 USB 接口函数的总结。对于这些函数的具体使用情况,请参考相应版本的 Windows 操作系统的设备驱动程序开发包(Device Driver Developer's Kit,即 DDK)中的文档。

表 12-2 USB 编程函数

Windows 提供的函数	目的
UsbBuildFeatureRequest()	该函数允许客户软件打开或关闭一个 USB 设备上的个别特性
UsbBuildGetDescriptorRequest()	该函数向调用者返回来自于主控制器的,有关某个描述符的信息
UsbBuildGetStatusRequest()	该函数允许客户软件查询一个给定的 USB 设备、端点、管道或接口的状态

(续)

Windows 提供的函数	目的
UsbBuildInterruptorBulkTransferRequest( )	该函数会在主机和一个设备之间建立一个用于批量传输或中断传输的请求
UsbBuildSelectConfigurationRequest( )	该函数提供了设置一个 USB 设备的配置的机制
UsbBuildSelectInterfaceRequest( )	该函数用于主机软件选择一个 USB 设备所使用的“可供替换”的端点
UsbBuildVendorRequest( )	该函数允许用户向一个 USB 设备上的某个管道、端点或接口发出专用的设备请求
Usb_CreateConfigurationRequest( )	该函数分配和构成一个 URB 来配置(或重新配置)一个接人的 USB 设备
Usb_GetStatusLength( )	该函数向调用者返回一个给定的接口描述符的长度(包括所有端点)
Usb_GetUSBDIVersion( )	该函数返回在本机上所运行的 USBD 接口的版本
Usb_ParseConfigurationDescriptor( )	该函数用来对所有可用的配置描述符进行扫描,从而为客户软件寻找要匹配的那个描述符,并且返回第一个匹配值
Usb_ParseDescriptors( )	该函数对所有可用的描述符进行搜索来寻找一个满足用户所说明的条件的描述符,并会向用户返回第一个匹配值
Usb_RegisterHcFilter( )	该函数允许程序员在主控制器路径中加入自己的代码

## 6. USB 驱动程序装载

Windows 使用 Device 或 Interface 描述符中的值来选择装入哪个驱动程序。Windows 一开始使用 Device 描述符的厂商和产品域(idVendor、idProduct 和 bcdDevice)形成硬件 ID。如果找不到与硬件 ID 匹配的型号的安装 INF 文件,Windows 将从 Interface 的类型域 bInterfaceClass、bInterfaceSubClass 和 bInterfaceProtocol 形成兼容 ID。然后,Windows 会搜索处理这些兼容 ID 中某一个的安装文件。如果没有找到安装文件,它会提示用户安装新的设备驱动程序。选择的安装文件会指定要装入的设备驱动程序。

Interface 描述符的类域 bInterfaceClass 对于基本 USB 设备是 0。如果该接口满足某个标准设备类规范,bInterfaceClass 是表 12-1 中列出的一个值。

注意 Device 描述符的类域在设备驱动程序的选择过程并未使用,而 bDeviceProtocol 域可以用于指示支持哪些新的特性。

USB 类规范定义了在某一种类型的设备中的接口和端点。一个类型可以为类设备、接口和端点定义一个或多个类特定的额外的描述符。这些设备仍需提供标准的设备、配置、接口和端点描述符。

### 12.3.2 USBDI 的 IOCTL

为了编写 USB 设备驱动程序,通常还要在源代码中包含 DDK 所提供的几个头文件。这些头文件存放在 \98DDK\inc\win98 这样一个目录之下。这些头文件的用途可以总结如下:

- usb100.h 定义了在 USB 设备驱动程序设计中所要用到的各种常量和数据结构。
- usbdi.h USBDI 例程,其中包括对 USBD 和 USB 设备驱动程序通用的数据结构,适用于内核和用户模式;
- usbdlib.h URB 构造和各种历程,定义了 USBD 所输出的服务,适用于内核和用

户模式。

**usbioctl.h** 给出了对 IOCTL 的定义,其中包括对 USBD 和 USB 设备驱动程序通用的数据结构,适用于内核和用户模式。

USB 类驱动程序主要通过表 12-3 所示的 USB 驱动程序接口(USBDI)的内部 IOCTL 使用。因为它们都是内部 IOCTL,所以只能用于内核的调用(如设备驱动程序)但却不能用于用户态的应用程序。DDK usbioctl.h 头文件源代码说明了如何定义和使用这些 IOCTL,这些内部 IOCTL 实际上是利用 Windows 系统提供的 ICO\_CODE 宏而由驱动程序开发人员新定义的 I/O 控制代码。

表 12-3 USBDI 的内部 IOCTL

内部 IOCTL	说 明
IOCTL_INTERNAL_USB_SUBMIT_URB	该 IOCTL 是客户驱动程序用来发送 URB 的(USB 请求块),停止等待结果
IOCTL_INTERNAL_USB_RESET_PORT	复位并重新启动一个端口
IOCTL_INTERNAL_USB_GET_ROOTHUB_PDO	该 IOCTL 由集线器驱动程序在内部使用
IOCTL_INTERNAL_USB_GET_PORT_STATUS	该 IOCTL 返回当前的端口状态
IOCTL_INTERNAL_USB_ENABLE_PORT	该 IOCTL 将请求该集线器重新启用一个被禁用的端口
IOCTL_INTERNAL_USB_GET_HUB_COUNT	该 IOCTL 由集线器驱动程序在内部使用
IOCTL_INTERNAL_USB_CYCLE_PORT	该 IOCTL 将模拟一个集线器端口上出现的接入/拆除操作

### 12.3.3 USBDI 结构定义

前面,我们已经在第 9 章中详细地介绍了 USB 规范中的各种描述符。这里我们将介绍 Windows 操作系统中对这些描述符所定义的数据结构。通过两者的对比,我们可以对 Windows 对 USB 规范的支持有一个深入的了解。

#### 1. 设备描述符数据结构

```
typedef struct _USB_DEVICE_DESCRIPTOR {
    UCHAR bLength;
    UCHAR bDescriptorType;
    USHORT bcdUSB;
    UCHAR bDeviceClass;
    UCHAR bDeviceSubClass;
    UCHAR bDeviceProtocol;
    UCHAR bMaxPacketSize0;
    USHORT idVendor;
    USHORT idProduct;
    USHORT bcdDevice;
    UCHAR iManufacturer;
    UCHAR iProduct;
```



```

    UCHAR iSerialNumber ;
    UCHAR bNumConfigurations ;
} USB_DEVICE_DESCRIPTOR, *PUSB_DEVICE_DESCRIPTOR

```

在该数据结构中的每个参数都同第 9 章中的设备描述符中所定义的各个域相对应(参见 9.6.1 节对设备描述符的定义)。

## 2. 配置描述符数据结构

```
typedef struct _USB_CONFIGURATION_DESCRIPTOR {
```

```

    UCHAR bLength ;
    UCHAR bDescriptorType ;
    USHORT wTotalLength ;
    UCHAR bNumInterfaces ;
    //...
    UCHAR iConfiguration ;
    UCHAR bmAttributes ;
    UCHAR MaxPower ;
} USB_CONFIGURATION_DESCRIPTOR, *PUSB_CONFIGURATION_DESCRIPTOR

```

在该数据结构中的每个参数都同第 9 章中的配置描述符中所定义的各个域相对应(参见 9.6.2 节对设备描述符的定义)。

## 3. 接口描述符数据结构

```
typedef struct _USB_INTERFACE_DESCRIPTOR {
```

```

    UCHAR bLength ;
    UCHAR bDescriptorType ;
    UCHAR bInterfaceNumber ;
    UCHAR bAlternateSetting ;
    UCHAR bNumEndpoints ;
    UCHAR bInterfaceClass ;
    UCHAR bInterfaceSubClass ;
    UCHAR bInterfaceProtocol
} USB_INTERFACE_DESCRIPTOR, *PUSB_INTERFACE_DESCRIPTOR

```

有关接口描述符中各个参数的定义请参考 9.6.3 节。

## 4. 接口选择数据结构

在进行选择配置时,我们还需要利用各种数据结构。Windows 98 DDK(Driver Development Kit)中为我们提供了这些数据结构,供驱动程序开发人员使用。这些数据结构包括 USBD\_INTERFACE\_LIST\_ENTRY, USBD\_INTERFACE\_INFORMATION 和 USBD\_PIPE\_INFORMATION。下面我们将对这三种数据结构进行简要的介绍。

USB 客户驱动程序使用 USBD\_INTERFACE\_LIST\_ENTRY 结构来建立一个插入到一个配置请求中的接口阵列。该数据结构定义如下:

```
typedef struct _USBD_INTERFACE_LIST_ENTRY {
    PUSB_INTERFACE_DESCRIPTOR InterfaceDescriptor;
```

```
PUSBD_INTERFACE_INFORMATION Interface;
| USBD_INTERFACE_LIST_ENTRY, *PUSBD_INTERFACE_LIST_ENTRY;
```

其中 InterfaceDescriptor 是指向一个 USB\_INTERFACE\_DESCRIPTOR 结构的指针,而该结构则描述了一个要添加至配置请求的接口。另外,Interface 则是指向用来说明 InterfaceDescriptor 所指向的那个接口的属性和设置的 USBD\_INTERFACE\_INFORMATION 结构的指针。

USB 客户驱动程序会使用 USBD\_INTERFACE\_INFORMATION 来为一个 USB 设备上的配置保存有关一个接口的信息。该结构定义如下:

```
typedef struct _USBD_INTERFACE_INFORMATION {
    USHORT Length;
    UCHAR InterfaceNumber;
    UCHAR AlternateSetting;
    UCHAR Class;
    UCHAR SubClass;
    UCHAR Protocol;
    //...
    USBD_INTERFACE_HANDLE InterfaceHandle;
    ULONG NumberOfPipes;
    USBD_PIPE_INFORMATION Pipes[0];
} USBD_INTERFACE_INFORMATION, *PUSBD_INTERFACE_INFORMATION;
```

USB 客户驱动程序使用 USBD\_PIPE\_INFORMATION 来保存来自于某个接口的有关一个管道的信息。该结构定义如下:

```
typedef struct _USBD_PIPE_INFORMATION {
    USHORT MaximumPacketSize;
    UCHAR EndpointAddress;
    UCHAR Interval;
    USBD_PIPE_TYPE PipeType;
    USBD_PIPE_HANDLE PipeHandle;
    ULONG MaximumTransferSize;
    //...
} USBD_PIPE_INFORMATION, *PUSBD_PIPE_INFORMATION;
```

其中 MaximumPacketSize 代表最大的分组尺寸;EndpointAddress 表示端点地址;Interval 代表主机对该管道的查询周期,单位是 ms;PipeType 代表该管道所使用的传输类型(控制传输、同步传输、中断传输和批量传输);PipeHandle 是一个主控制器定义的用来访问该管道的句柄;MaximumTransferSize 则说明了用于该管道上的一个传输请求的最大字节数。

## 5. 字符串描述符数据结构

```
twoedf struct USB_STRING_DESCRIPTOR {
```



```
UrbControlFeatureRequest;
struct _URB_CONTROL_SYNC_FRAME_REQUEST
    UrbControlSyncFrameRequest;
struct _URB_CONTROL_VENDOR_OR_CLASS_REQUEST
    UrbControlVendorClassRequest;
struct _URB_CONTROL_GET_INTERFACE_REQUEST
    UrbControlGetInterfaceRequest;
struct _URB_CONTROL_GET_CONFIGURATION_REQUEST
    UrbControlGetConfigurationRequest;
}

| URB, *PURB;
```

其中的参数都是为一个请求或命令而定义的格式。从上面我们可以看到明显的 \_URB\_\* 的结构。

最后我们还有必要讨论以下 \_URB\_HEADER 数据结构。该数据结构的定义如下：

```
struct _URB_HEADER {
    USHORT Length;
    USHORT Function;
    //...
    USBD_STATUS Status;
    //...
};
```

Function 说明了一个数字代码，该代码指出了该 URB 所要求的操作。这些功能代码包括 URB\_FUNCTION\_SELECT\_CONFIGURATION, URB\_FUNCTION\_ABORT\_PIPE, URB\_FUNCTION\_GET\_CONFIGURATION 和 URB\_FUNCTION\_RESET\_PIPE 等等，共 39 种之多。在这里不可能一一予以介绍，有关内容请参考 Windows DDK 说明文档。不过有一点要明确，\_URB\_HEADER 结构是所有的 USB 请求中的一个成员而这些 USB 请求则是 URB 结构的一部分。该结构提供了有关发往主控制器的每一个请求的通用信息。

## 附录 A USB 字汇表

在此我们给出了 USB 规范中的术语和缩写的列表,供广大读者参考。

ACK	应答。在 USB 总线上的 USB 设备,或一个主机和 USB 设备之间,指示一个肯定应答的握手分组。
Active Device	有效设备。一个通电并且不处于挂起状态的设备。
Asynchronous Data	异步数据。通过 USB 总线传输的、没有定时要求的数据。与之相对的是同步数据。
Asynchronous RA	异步 RA。RA 过程中的输入数据速率 $F_{si}$ 和输出数据速率 $F_{so}$ 是独立的(即它们没有共享主时钟)。
Asynchronous SRC	异步 SRC。SRC 过程中的输入数据速率 $F_{si}$ 和输出数据速率 $F_{so}$ 是独立的(即它们没有共享主时钟)。
Audio Device	音频设备。一个发送或接收经抽样后的模拟信号的设备。
AWG #	由美国电线规格标准所定义的一个电线交叉部分的测量。
Babble	串扰。总线上持续出现的、不是任何设备或主机所希望出现的信号。
Bandwidth	带宽。在单位时间内经过总线所传输的数据量,典型的单位是 b/s 或 B/s。
Big Endian	一种将多个字节中的最高有效位放置在低端存储地址上的数据存储模式。例如,在以 Big Endian 格式存放的一个字中,高有效字节放置在低端存储地址上而低有效字节则存放在高位地址上。参见 Little Endian。
Bit Stuffing	比特填充。在数据流中插入“0”以在数据线上产生一个电气变化,从而允许一个 PLL(锁相环)保持锁定状态。
Buffer	缓冲区。计算机(或一个 USB 设备)由于存放不能立即使用的数据的暂时存储空间。例如,如果到达的数据比应用软件所能使用的要快,那么这些数据就会被主机软件之下的元件所缓存。
Bulk Transfer	批量传输。主要用于那些可以利用任何可用的带宽进行传送,或可以延迟到有可以利用的带宽时再进行传送的数据,它具有非周期、突发性强的特点。

Bus Enumeration	总线枚举。是对总线上接人的 USB 设备进行检测和识别。
Capabilities	性能。是指一个设备可以由主机软件进行控制的功能参数。
Characteristics	特性。通用串行总线设备的那些不变的属性。例如，设备类型是一个设备的特性。
Client	客户。驻留在主机上的，同主机进行交互来安排一个功能设备和主机间数据传输的软件。对于所传输的数据，客户是开放的数据提供者和使用者。
COM Port	通信端口。在个人计算机上，最典型的是一个八位的异步串行端口。
Configuring Software	配置软件。负责配置一个 USB 设备的主机软件。它可以是专用于该设备的一个配置程序或软件。
Control Pipe	控制管道。与一个消息管道相同。
Control Transfer	控制传输。USB 四种传输类型之一。主要用于命令/状态操作，由主机软件发起的请求/响应通信过程，具有突发性、非周期的特点。
CRC	参见循环冗余校验。
CTI	计算机电话集成。
Cyclic Redundancy Check	参见循环冗余校验。在数据上进行的校验，用于检测在数据传输和读写过程中是否出错。通常情况下，CRC 结果和数据一同存储或传送。通过对存储或传送的 CRC 和计算出的 CRC 进行比较，可以确定是否出现了一个错误。
Default Address	缺省地址。由 USB 规范定义的，在一个设备首次上电或复位时使用的一个地址，其值为 00h。
Default Pipe	缺省管道。由 USB 系统软件定义的一个消息管道，用于在主机和一个 USB 设备端点 0 之间传送控制和状态信息。
Device	实现一个功能的逻辑或物理实体。对实际的实体进行描述依赖于引用的环境。在最低层，设备指的是单个的硬件元件，如在一个存储器设备中。而在更高一层，它指的是一个可以实现一个特殊功能的硬件元件的集合，例如一个 USB 接口设备。而在更高的层上，设备指的是由一个接入 USB 总线的实体所实现的功能；例如，一个数据/FAX Modem 设备。 当作为一个非专用基准来使用时，一个 USB 设备可以是一个集线器，也可以是一个功能设备。

Device Address	设备地址。通用串行总线上的一个设备的地址。在一个设备首次上电或复位时使用的地址是一个缺省地址。而集线器和功能设备则由 USB 软件来分配一个唯一的 USB 地址。
Device Endpoint	设备端点。在主机和设备之间存在的通信流中,可以唯一确认一个 USB 设备是信息的发起者或接收者的部分。
Device Resources	设备资源。由通用串行总线设备所提供的资源,如缓冲区空间和端点。参见主机资源和通用串行总线资源。
Device Software	设备软件。负责使用一个通用串行总线设备的软件。该软件可以用于,也可以不用于为使用一个设备,而对设备进行的配置。
DMI	桌面管理接口 (Desktop Management Interface) 的缩写。它是由桌面管理任务组织所开发的管理主机系统元件的方法。
Downstream	下行。数据流向为离开主机。一个下行端口指的是在一个集线器上,从电气特性上讲距离产生来自于集线器的数据的主机最远的端口。下行端口用于接收上行数据通信流。
Driver	驱动器/驱动程序。当我们指的是硬件时,它代表用来驱动一个外部负载的 I/O 平台。当所说的是软件时,它代表负责同一个硬件进行接口的程序;即一个设备驱动程序。
End User	中断用户。一个主机的使用者。
Endpoint	端点。参见设备端点。
Endpoint Address	端点地址。在一个通用串行总线设备上,一个设备地址和一个端点号的联合。
Endpoint Number	端点号。在一个通用串行总线设备上,一个唯一的管道端点。
EOF1	帧定时点 #1 结束位置。集线器可以用它来监视或拆除在接近或跨越一个帧的末尾时持续进行的通用串行总线活动。
EOF2	帧定时点 #2 结束位置。集线器可以用它来监视在接近一帧的末尾时的通用串行总线活动。
EOP	分组结束。
Fs	参见抽样速率。
False EOP	假 EOP。一个会被分组接收者解释为一个分组结束的虚假的、通常是由噪声而引入的事件。
FireWire	IEEE P1394 总线标准。

Frame	由一个 SOF 令牌的开始到下一个 SOF 令牌开始时的这段时间;包括一系列的处理。
Frame Pattern	帧模式。在每一帧所传输的许多样值中,表现出一种中继模式的一个帧序列。对于一个 44.1kHz 的音频传输而言,帧模式可以是有 9 帧包含 44 个样值,再跟上一个包含了 50 个样值的帧。
Full - duplex	全双工。可以在两个方向同时进行的计算机数据通信方式。
Function	功能设备。一个向主机提供了一种功能的通用串行总线设备。例如,ISDN 连接设备,数字麦克风或扬声器等等。
Handshake Packet	握手分组。用于应答或丢弃一个特定条件的分组。例如,一个 ACK 和 NAK。
Host	主机。安装了通用串行总线控制器的主计算机系统。它包括主机硬件平台(CPU,总线等)和所使用的操作系统。
Host Controller	主控制器。它是主机的通用串行总线接口。
Host Controller Driver	主控制器驱动程序。对通用串行总线硬件进行抽象的软件层。主控制器驱动程序为同一个主控制器的交互提供了一个 SPI。主控制器驱动程序隐藏了主控制器硬件实现的具体细节。
Host Resources	主机资源。由主机所提供的资源,例如一个缓冲区空间和中断。参见设备资源和通用串行总线资源。
Hub	集线器。向通用串行总线提供了额外连接的一个通用串行总线设备。
Hub Tier	集线器层。在一个 USB 网络拓扑结构中的连接层,作为数据必须通过的许多集线器而给出。
IEEE P1394	一种高性能的串行总线。P1394 的目标是硬盘和音频外设,这些应用的带宽要求将超过 100Mb/s。该总线协议在同样的四根信号线上,既可以支持同步传输,也可以支持异步传输。
Industry Standard Architecture	工业标准体系结构。用于 IBM AT 或 XT 兼容机的 8 位或 16 位扩展总线。
Interrupt Request	中断请求。一个允许设备请求主机的关注的硬件信号。主机通常会调用一个中断服务函数来控制产生这一请求的条件。
Interrupt Transfer	中断传输。它是四种通用串行总线传输类型之一。中断传输的特征是小数据量、非周期、低频率、时延固定的由设备发起的、用来向主机指出该设备的服务需要的通信。参见中断请求。
IRQ	

<b>ISA</b>	参见工业标准体系结构 (Industry Standard Architecture)。
<b>ISDN</b>	综合业务数据网。
<b>Isochronous Data</b>	同步数据。一种其定时信息由其传输速率来指示的数据流。
<b>Isochronous Device</b>	同步设备。由 USB 规范定义的，具有同步端点的一个实体，它可以产生或接收抽样后的模拟数据流或同步数据流。
<b>Isochronous Sink Endpoint</b>	同步接收端点。该端点能够利用一个同步数据流。
<b>Isochronous Source Endpoint</b>	同步发送端点。该端点可以产生一个同步数据流。
<b>Isochronous Transfer</b>	同步传输。四种通用串行总线传输类型之一。在处理同步数据时，要使用同步传输。同步传输在主机和设备之间提供了周期性的、连续的通信。
<b>Jitter</b>	抖动。由于机械或电气变化而引起的同步性能下降而产生的误差。更具体地说，是经过一个传输媒质后数字脉冲产生的相位抖动。
<b>Line Printer Port</b>	打印机端口线。用于访问一个打印机的端口。在大多数个人计算机上，最常用的是一个 8 位的并行接口。
<b>Little Endian</b>	一种将多个字节中的最低有效位放置在低端存储地址上的数据存储模式。例如，在以 Little Endian 格式存放的一个字中，低有效字节放置在低端存储地址上而高有效字节则存放在高位地址上。参见 Big Endian。
<b>LOA</b>	活性损失。其特征是一个分组开始后，没有相应的分组结束标志(EOP)。
<b>LPT Port</b>	参见打印机端口线。
<b>LSB</b>	最低有效位。
<b>Message Pipe</b>	消息管道。利用一个请求/数据/状态变化表来传输数据的管道。其数据具有一定的结构，这样可以允许请求被可靠地标识和通信。
<b>Micro Channel Architecture</b>	微通道体系结构。应用于 IBM PS/2 兼容机的一种 32 位扩展总线。
<b>Modem</b>	调制器和解调器的缩写。它是将信号在模拟和数字方式之间进行转化的元件。最典型的应用是通过一个一般是模拟工作方式的电话往来传输来自于计算机的数字信息。
<b>MSB</b>	最高有效位。
<b>NACK</b>	否定应答。用来指示一个否定应答的握手分组。
<b>Non Return to Zero Invert</b>	非归 0 反转编码。一种串行编码方式，它使用相反的交替变化的高低电压来表示“1”和“0”，在编码的比特之间不返回 0 电平(参考)。它可以减少对定时脉冲的要求。

NRZI	参见非归 0 反转编码(Non Return to Zero Invert)。
Object	对象。代表一个通用串行总线实体的主机软件或数据结构。
Packet	分组。为传输而组织的数据集合。典型的分组包括三个部分:控制信息(即信源,目的地和长度)、所要传输的数据、以及差错检测和纠正比特。
Packet Buffer	分组缓冲区。一个通用串行总线设备用来发送和接收单个分组的逻辑缓冲区。它可以决定该设备可以接收或发送的最大分组尺寸。 赵国华 使用本文件品 权属本公司
Packet ID	在一个通用串行总线分组中,用来指出分组类型,以及推断出的分组格式和应用于该分组的差错检测类型的一个域。
PCI	参见外围元件互连。
PCMCIA	参见个人计算机存储器工业协会。
Peripheral Component Interconnect	外围元件互连。在个人计算机上使用的一种 32 位或 64 位与处理器独立的扩展总线。
Personal Computer Memory Card International Association	个人计算机存储器工业协会。该组织标准化并提出 PC 卡的技术。
Phase	阶段。一个令牌、数据或握手分组;一个处理具有三个阶段。
Physical Device	物理设备。一个具有物理应用的设备;如扬声器、麦克风和 CD 播放器。
PID	参见 Packet ID。
Pipe	代表一个设备上的一个端点和主机上的软件之间联系的一个逻辑抽象。一个管道具有若干属性:例如,一个管道可以用流方式(流管道)或消息方式(消息管道)来传送数据。
Plug and Play	即插即用。该技术用于配置 I/O 设备来使用一个主机中的没有冲突的资源。由即插即用技术来管理的主机资源包括 I/O 地址范围,存储器地址范围,IRQ 和 DMA 通道。
PnP	参见即插即用。
Polling	轮询(查询)。对于多个有数据要传输设备,每次对一个设备进行询问。
POR	参见上电复位。
Port	端口。对一个系统或电路进行访问的接入点。对于通用串行总线而言,它指的是一个通用串行总线设备的连接点。

Power On Reset	上电复位。当通电后,恢复一个存储设备、积存器或存储器到预先决定的状态。
PLL	锁相环。作为一个相位检测器来保持输入的频率和振荡器的相位的一个电路。
Programmable Data Rate	可编程的数据速率。它可以是一个固定的数据速率(单一频率的端点),限定数目的数据速率(32 kHz, 44.1 kHz, 48 kHz, ...),或一个可以连续编程来改变的数据速率。一个端点的确切的可编程能力必须在适当的专用类型的端点描述符中予以说明。
Protocol	协议。一个有关设备间数据传输的格式和定时的专用原则,进程或惯例的集合。
RA	参见速率适配。
Rate Adaptation	速率适配。该过程将一个以 $F_s$ 频率抽样的输入数据流以由所采用的速率适配算法所决定的质量损失转变为一个以 $F_o$ 频率抽样的输出数据流。这一过程需要差错控制机制。 $F_s$ 和 $F_o$ 可以是不同的和异步的。 $F_s$ 是速率适配(RA)的输入数据速率; $F_o$ 是速率适配(RA)的输出数据速率。
Request	向一个通用串行总线设备发出的、包含在 SETUP 分组中的数据部分的一个请求。
Retire	撤消。结束一个用于传输的服务并向适当的软件客户通知该结束信号的一个动作。
Root Hub	根集线器。直接接入主控制器的一个通用串行总线集线器。该集线器接入主机,并位于 0 层。
Root Port	一个集线器上的上行端口。
Sample	抽样。一个端点所处理的最小数据单位;它是一个端点的特性。
Sample Rate ( $F_s$ )	抽样速率。1s 的抽样数。
Sample Rate Conversion	抽样速率的转变。RA 过程的一个应用,用于抽样后的数据流。差错控制机制由填充技术而取代。
SCSI	参见小型机系统接口。
Service	服务。一个由 SPI 而提供的进程。
Service Interval	服务时间间隔。在向一个通用串行总线端点发出的连续的数据传输或数据接收请求之间所经过的周期。
Service Jitter	服务抖动。相对于其预计的传输时间而出现的服务传输偏差。
Service Rate	服务速率。在单位时间内,向一个给定的端点发出的服务数。

Small Computer Systems Interface	小型机系统接口。利用通用的系统硬件和软件,而允许外设接入一个主机的一种局部 I/O 总线。
SOF	帧开始的缩写。SOF 是一帧中的第一个处理操作。SOF 允许端点来识别一帧的开始并将其内部端点时钟同主机同步。
SPI	参见系统编程接口。
SRC	参见抽样速率转换。
Stage	阶段。组成一个控制传输的序列的一部分;建立阶段、数据阶段和状态阶段。
Stream Pipe	流管道。把数据作为一个没有通用串行总线所定义的结构的数据流进行传输的管道。
Synchronization Type	同步类型。一种代表一个同步端点同其他的同步端点连接的能力的分类。
Synchronous RA	同步 RA。RA 过程的输入数据速率 $F_{in}$ 和输出数据速率 $F_{out}$ 都是从同一个主时钟所获得的。在 $F_{in}$ 和 $F_{out}$ 之间具有规定的关系。
Synchronous SRC	同步 SRC。SRC 过程的输入数据速率 $F_{in}$ 和输出数据速率 $F_{out}$ 都是从同一个主时钟所获得的。在 $F_{in}$ 和 $F_{out}$ 之间具有规定的关系。
System Programming Interface	系统编程接口。由系统软件提供的,用于服务的一个定义的接口。
Termination	终端。接在电缆一端的被动元件,以阻止信号反射或回送。
Time-out	超时。对于某个事先确定的时间间隔,检测到了在此期间总线缺乏活动。
Token Generator	令牌发生器。参见启动器。
Token Packet	令牌分组。一种类型的分组,用以指出在总线上所进行的是什么样的处理。
Transaction	处理。指向一个端点的服务传输;包括一个令牌分组,可选的数据分组和可选的握手分组。根据处理的类型,会允许/需要专门的分组。
Transfer	传输。在一个软件客户和它的功能模块之间,进行数据传输的一个或多个总线处理。
Transfer Type	传输类型。确定在一个软件客户和它的功能模块之间数据流所具有的特性。定义了四种传输类型:控制、中断、批量和同步。
Turnaround Time	转向时间。它指的是一个设备在接收完一个分组之后,开始传输一个分组之前所需等待的时间,它可以阻止在

通用串行总线总线上的冲突。该时间是基于电缆的长度和传播时延,以及相对于通用串行总线上的其它设备而言传输设备的位置而确定的。

**Universal Serial Bus**

通用串行总线。它指的是一个通用串行总线设备、软件和硬件的集合,可以允许它们同功能设备向主机提供的功能进行连接。

**Universal Serial Bus Device**

通用串行总线设备。包括集线器和功能设备。参见设备。

**Universal Serial Bus Interface**

通用串行总线接口。在通用串行总线电缆和一个通用串行总线设备之间所存在的一个硬件接口。它包括使得一个通用串行总线设备可以接收和发送分组所需的协议引擎。

**Universal Serial Bus Resources**

通用串行总线资源。由通用串行总线提供的资源,例如带宽和电源。参见设备资源和主机资源。

**Universal Serial Bus Software**

通用串行总线软件。基于主机的,负责管理主机和接入的通用串行总线设备之间所进行的交互的软件。

**USB**

参见通用串行总线。

**USBD**

参见通用串行总线设备驱动程序。

**Universal Serial Bus Driver**

通用串行总线设备驱动程序。负责向客户提供通用的,用来指挥在一个或多个主控制器上的一个或多个功能的,驻留在主机上的软件实体。

**Upstream**

上行。数据流通向主机的方向。一个上行端口指的是在电气特性上最接近于主机的端口,它可以产生来自于集线器的上行通信量。另外,上行端口还负责接收下行数据通信量。

**Virtual Device**

虚拟设备。由一个软件接口层所代表的设备;即可以再生出音频.WAV文件的一个硬盘,以及相应的设备驱动程序和客户软件。

**WEOF2**

等待 EOF2 点(Wait for EOF2 point)。四种集线器中继器状态之一。

**WFEOF**

等待分组结束(Wait for end of packet)。四种集线器中继器状态之一。

**WFSOF**

等待帧开始(Wait for start of frame)。四种集线器中继器状态之一。

**WFSOP**

等待分组开始(Wait for start of packet)。四种集线器中继器状态之一。



Powered by xiaoguo's publishing studio  
QQ:8204136