目　录

第三部分 USB 配置

第四部分 USB 主机软件

第五部分 USB 设备类

第六部分 主控制器和集线器:实现实例

第一部分：USB的出现

第一章对USB的需求

第二章解决方案

对 USB 的需求

✎ 本　章

今天的个人电脑上所使用的大多数外围设备仍然是基于接口实现的，这些接口最早由 IBM 公司在 20 世纪 80 年代早期设计。这些接口的设计有很多缺陷，它们不仅给设计者带来了许多麻烦，而且也给用户带来诸多不便。本章将对这些缺点进行总结和回顾。在这个过程中我们将会注意到：这些缺点中的很多直接导致了人们需要更好的解决方案来实现外围设备与计算机之间的连接。

✎ 下一章

下一章将讨论外围设备与计算之间连接的可能的解决方案，在连接过程中会遇到很多问题，这些问题中的大多数将在下一章中加以阐述。此外，下一章还将具体介绍 USB 的解决方案。

🕮 目前 PC 的 I/O 模式的缺点

在个人电脑的领域中，外围设备存在着很多问题，这些问题大致可以归结到成本、配置以及和个人电脑的连接等几个方面。而 USB 正是作为克服这些困难的一种解决方案出现的。简而言之，USB 建立了一套连接和访问外围设备的方法，这些方法可以有效地减少总体成本。而且从终端用户的角度来看，它可以简化设备的连接和配置，不仅如此，它还可以解决老式的外围设备所存在的某些技术问题。下面几节将详细介绍现在的 PC 外围设备所存在的几种不同问题，并且将详细阐述 USB 标准所面对的挑战。

技术问题

图 1-1 表示了传统的 I/O 模式，在那里外围设备通常被映射为 CPU 的 I/O 地址空间，并且被分配一个指定的 IRQ（中断请求），在某种情况下它们也可以是一个 DMA 通道。这些系统资源被分配给指定的外围设备，这些地址的分配通常是由 IBM 公司和其他设备制造商指定。而且在事实上，这种地址分配方法已经成为一种标准。软件开发者要利用这些关于中断请求和 DMA 通道的信息对指定的设备进行访问。

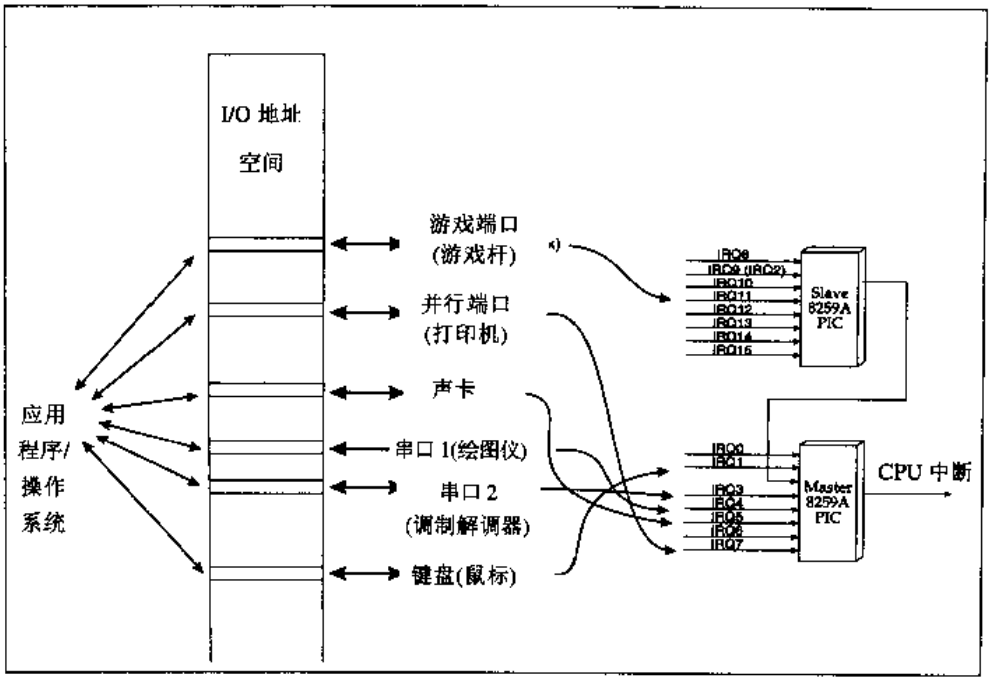


图 1-1 传统外围设备所用的系统资源

就如同在图中所示的那样，外围设备消耗了个人电脑的许多系统资源，他们使许多系统资源不可使用，并且很容易产生冲突，这就造成了许多问题。

中断

在大多数情况下，许多关键的系统资源问题都涉及到中断请求分配，因为 PC 上所用到的很多设备都要求使用中断。这一点对于那些通过 ISA 总线和 PC 相连的外围设备来说尤其是这样，因为 ISA 总线不能可靠地支持共享式中断。表 1-1 列出了每个 IRQ 以及使用每个 IRQ 的典型设备。从这个表中我们可以看到，很多 IRQ 指定给特定的设备，这种做法完全是出于传统的考虑。但是还有一些 IRQ 可以被很多种外围设备使用。基于 PCI 总线的系统也包括 ISA 或 EISA 两种总线。在这种情况下，中断的缺陷就可

能变成一个主要问题。因为理想的情况是：IRQ 中的某几个应该给留下了，以供那些可能需要使用它们的总线扩展卡使用。

表 1-1　　典型的基本设备占用中断的情况

IRQ	设　　备
IRQ0	系统时钟（专用于系统板）。
IRQ1	键盘（专用于系统板）。
IRQ2	用于从属中断控制器串行通道（外围设备不可使用）。
IRQ3	串行鼠标、调制解调器、绘图仪、串口打印机、游戏端口、输入笔、红外线端口。
IRQ4	串行鼠标、调制解调器、绘图仪、串口打印机。
IRQ5	总线型鼠标、并口打印机、声卡、局域网适配器、磁带驱动器、游戏端口。
IRQ6	软盘驱动器。
IRQ7	并口打印机。
IRQ8	RTC 报警器（专用于系统板）。
IRQ9	局域网适配器、视频适配器、磁带驱动器、游戏端口。
IRQ10	局域网适配器、声卡。
IRQ11	局域网适配器、SCSI 控制器、PCMCIA 控制器。
IRQ12	PS/2 鼠标、PCMCIA 控制器。
IRQ13	数字协处理器出错（专用于于系统板）。
IRQ14	硬盘驱动器。
IRQ15	SCSI 控制器、PCMCIA 控制器。

I/O 地址

在 PC 的环境中，I/O 地址冲突是十分常见的。注意外围设备通常需要大块的 I/O 地址单元，这样他们才能够报告设备的状态信息并且给设备发送命令。尽管 x86 处理器有能力访问 64KB 的 I/O 地址单元（对外围设备来说这已经足够了），但是传统的扩展卡通常只能对 16 条地址线中的 10 个进行解码。这就导致了最多只能有 1KB 的地址空间块可以被 ISA 的扩展卡所使用。而且，有限的解码还造成了一个著名的混淆效应，那就是造成 1KB 的 I/O 地址空间的上 768 字节不可使用。若需要了解进一步的信息，可以参见由 MindShare 著的《ISA 系统体系结构》一书。

非共享式接口

标准 PC 的外围设备接口（例如与串行口和并行口连接）支持单个设备的连接。由于在一个给定的时刻只有一个外围设备可以连接，这样，连接的灵活性就被最小化。由于这个限制，所以通常是做一块专用的扩展卡插到扩展总线上（例如 ISA、EISA 或 PCI），通过它来为新的外围设备建立一个连接点，但是这种做法相对来说成本比较昂贵。

终端用户所关心的问题

当最终用户把外围设备连接到计算机上去时，会遇到各种各样的问题，我们下面将讨论这些问题。

种类繁多的数据线

鼠标、键盘、打印机、外接调制解调器、Zip 驱动器、绘图仪等设备都需要专用的数据线。他们中大多数都是完全不同的。图 1-2 描绘了在 USB 出现之前普通 PC 的主板上的外设连接口。当用户把这些外围设备连接到计算机时，他们不得不面对很多种不同的接口和数据线，这会使用户觉得十分麻烦。

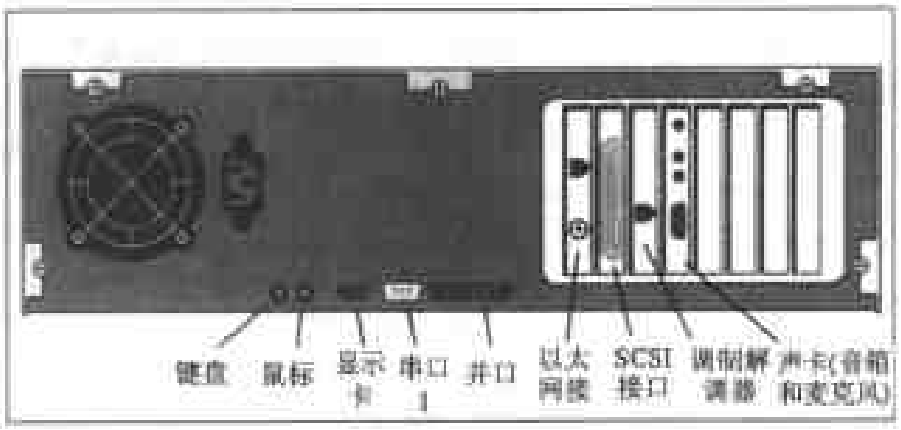


图 1-2　主板上的外设连接口

扩展卡的安装和配置

当外围设备购买以后，在安装时，他们中大多数都需要安装扩展卡。第一步当然

就要打开计算机的机箱盖（而且这还不算，你往往还需要一把螺丝刀），然后设置开关和跳线来配置这块卡（你可能会对数字 03F8h 不理解），把这个卡插入，然后合上箱盖。但是麻烦还刚刚开始。一旦系统上电以后，你还必须从软盘安装这个设备所用的软件，特别是对非即插即用设备来说更是如此。这种过程无论对一个新手还是一个有经验的用户来说都是十分麻烦的。最后，当做完这些工作后，这个设备可能会正常工作，但也可能不正常工作，这往往是因为硬件和软件的冲突引起的，所以你还要确保它所使用的系统资源没有被系统中的另外一个设备所使用。

外围设备不能热插拔

你有没有在计算机上电之前忘记把你的鼠标插入鼠标接口的经历？在启动过程完成以后，你的鼠标当然不能工作。所以你就要重启！由于软件检查硬件是否存在，而且系统仅仅为那些检测到的硬件安装驱动程序，大多数新连接上的外围设备在重新启动系统之前是不能用的，因为只有通过重启软件才能够检测到刚才连接上去的设备，并且为这个设备装入必要的驱动程序。

当设备连接上以后，用户还必须安装新的软件并且重启系统。在这个过程中，必须选择系统资源并且把它分配给新的设备（例如 I/O 地址空间，IRQ，和 DMA 通道），如果想使他们正确的工作，还要确保你所选择的资源没有被系统中的其他设备所使用。

成本

基于传统 PC 的设计，实现系统以及外围设备的协调工作的成本是非常昂贵的，这是由于标准外围设备连接器和相关数据线的高昂成本。由于很多种外围设备都使用标准的连接器，所以一个必要的做法是制作一块扩展卡，通过它来连接你的外围设备和系统。但也使得这种解决方案的成本变得很昂贵。

下一章我们将讨论这些问题的解决方案。

解 决 方 案

✎ 上一章

上一章叙述了在 USB 出现之前，把外围设备连接到计算机上去时将会面临的问题。同时它还规定了连接计算机和外围设备的新方法所应该包含的特性。

✎ 本 章

本章讨论了几种可能的解决方案，这些方案都可以满足外围设备的新连接机制的需求。当然，这里是指在 PC 的环境下。

✎ 下一章

下一章对 USB 环境做了一个概述。它定义了 USB 的通信模型，描述了 USB 实现中的所有硬件和软件元素，并且还对元素之间的相互作用作了说明。

🕮 设计目标

制造商和用户在旧的连接机制上已经发现许多缺点，所以作为一种新的解决方案就应该克服这些现存的缺点，不仅如此，它还要求能够提供进一步的发展和扩充空间。上面提到的缺点概括如下。

🕮 新解决方案的挑战

一种新的解决方案应该能够克服已经被发现的缺点，并且还要能够提供新的能力。

因此设计目标应包括：

- 一个连接任何 PC 外围设备的简单连接器类型；
- 可以把很多外围设备连接到同一个连接器上的能力；
- 一种消除系统资源冲突的方法；
- 自动检测和配置外围设备；
- 以较低成本实现系统和外围设备的连接；
- 提高性能；
- 支持对新设计的外围设备的连接；
- 支持老式的的硬件和软件；
- 低功率；
- 这些目标将在下面的部分分别进行讨论。

提高系统性能

用来替代传统外围设备的新的标准应该提供更为优秀的系统性能。在这一点上 PCI 总线做得不是很好。这一点体现在当它访问基于 ISA 总线老式外围设备的时候，系统的性能就会出现下降。但是 USB 外围设备能够充分发挥自己的性能，而且它不会对系统的总体性能的表现产生任何副作用。

即插即用支持

自动配置功能对于满足最终用户的需求是非常关键的。为了做到即插即用，这种新的解决方案至少应该实现以下两点：第一，不需要通过开关的设置和手工跳线来配置设备；第二，当有新的外围设备连接到计算机上去时，不需要安装新的软件。总之，对用户来说，把设备连接到计算机上的工作应该是非常简单的，并且连接完成后该设备立刻就可以使用。

热插拔

当大多数老式的 I/O 设备连接到系统上去时，如果不重新启动整个系统，那么他们是无法使用的。重新启动系统是为了让软件能够检测到新连接上的外围设备。新的解决方案应该提供一种检测方法，当新的外围设备被连接到系统后，系统能够检测到它，并且为这个设备安装相应的软件，使该设备可以被访问。

升级/扩展的空间

新的解决方案应该为设备将来的升级留下扩展空间，以支持外围设备的更新换代。应该提供一些支持，使电话功能集成进个人电脑变得更加容易。新的解决方案还应为新的智能外围设备提供充分的灵活性，如：交互式的外围设备（例如电脑游戏上用的）、家庭自动化、数字音频、电话、压缩视盘等。

对老式硬件/软件的支持

新的解决方案还应该加入对老式的硬件和软件的支持。能够和老式的外围设备进行通信的应用程序在新的外围设备上也应该能正常工作，老的和新的外围设备能够协同工作。从而使新老设备之间实现平滑过渡。

老的操作系统不认识 USB 端口。同样，为老的系统设计的应用程序也不能正常工作。解决这个问题有两个选择：一是你不再使用老的操作系统；另外一个选择是：通过安装设备驱动程序加入对 USB 的支持，从而这个操作系统的新版本也就认识了 USB 端口。

低廉的成本

连接外围设备的新方案应该提供以上所述的特征。但是要注意的是：它不能大幅度地增加外围设备的成本；当然，整个系统的成本也不能提高。例如，如果一个传统鼠标的成本明显低于 USB 鼠标的成本，那么用户当然会拒绝使用鼠标的 USB 解决方案。

对可能的解决方案的分析

对于连接和访问外围设备的新方案来说，存在很多种选择都可以满足需求。下面的部分将简要地讨论几种不同方案的选择。这些方案都可能被选择使用。假定所有的解决方案在其他方面都是基本相同的，那么在这些方案的选择过程中，一个重要的因素就是对这些解决方案所产生的性价比的衡量。每一个新的解决方案所支持的应用范围都被列在表 2-1 中。从这个表中我们可以看到，USB 应用于某些外围设备时需要很少的带宽，但是有些设备，例如在磁盘驱动器和压缩视盘方面的应用就需要非常大的带宽。

表 2-1　　应用场合，相关的性能要求和所希望的特性

性　能	应　用	特　性
低速： 交互设备 10Kb/s~100Kb/s	键盘、鼠标 输入笔 游戏外设 虚拟现实外设 显示器配置	低成本 热插拔 易于使用 适用于多种外设
中速： 电话，音频 500Kb/s~10Kb/s，000Kb/s	ISDN PBX POTS 数字音频 扫描仪/打印机	低成本 易于使用 能保证的等待时间 能保证的带宽 动态连接/取下 适用于多种外设
高速： 视频，磁盘，局域网 25Mb/s－500Mb/s	台式电脑硬盘驱动器 视频会议 即插即用局域网	高带宽 能保证的等待时间 易于使用

表 2-2　　不同解决方案的性能和复杂性的对比

总线名称	数 据 率	主机复杂性	外设复杂性
存取型总线	100Kb/s	简单的硬件或者软件异步收发报机	简单的硬件或者软件异步收发报机
GeoPort	2.048Mb/s	计算机控制的通用异步收发报机	计算机控制的通用异步收发报机
IEEE 1394（Firewire）	400Mb/s	12,000~20,000 个门电路	5,000~7,000 个门电路
USB	12Mb/s	10,000 个门电路	1,500~2,000 个门电路

存取型总线

此方案的主机和外围设备的复杂性较低，这就使成本变得较为合理。然而，100Kb/s 的低带宽使得这种解决方案不能满足所有外围设备的需求。

Geo 端口

它主要是作为苹果机上的一种专门解决方案。它几乎完全应用于远程通信的功能，不支持 PC 下外围设备所希望的功能。

IEEE 1394

这种解决方案通常被称为 FireWire，为所有的外围设备应用提供了充足的带宽。但是与之相关的复杂性使得它的成本变得非常昂贵，特别是当实现低性能/低成本的外围设备时，它就更加不划算了。

USB—最好的平衡点

通用串行总线（USB）在连接 PC 外围设备方面是一个最好的解决方案，至少从性价比上来说是如此。连接到 USB 端口的设备可以加入附加的连接，用于连接其他的 USB 设备，如图 2-1 所示。这些附加的连接是通过 USB 集线器提供连接的，USB 集线器可以是一个独立设备，也可以集成到其他的 USB 外围设备中，例如打印机或键盘。USB 的特性见表 2-3。

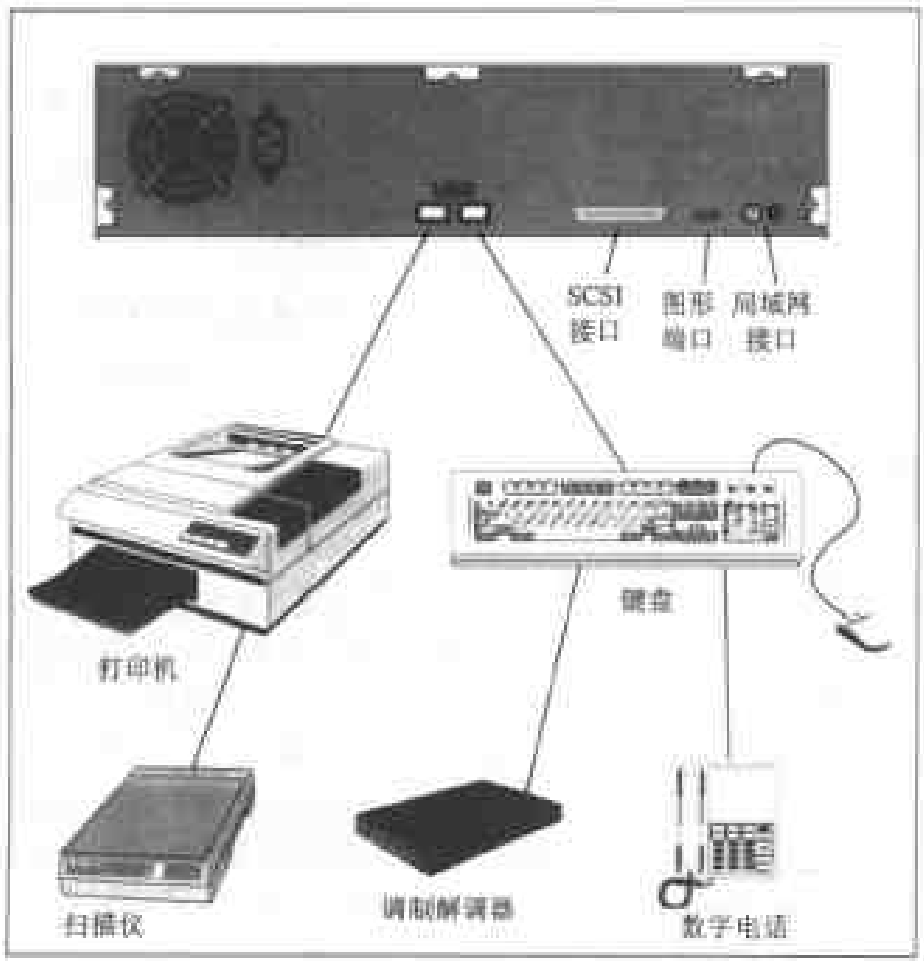


图 2-1　USB 设备的连接方式

表 2-3　　USB 特性

特　性	描　　述
低成本	为了把外围设备连接到 PC 上去，USB 提供了一种低成本的解决方案。
热插拔	设备连接后由 USB 自动检测，并且由软件自动配置，完成后立刻就能使用，不需要用户进行干涉。
单一的连接器类型	USB 定义了一种简单的连接器，它可以用来连接任何一个 USB 设备。多个连接器可以通过 USB 集线器连接。
127 个设备	每个 USB 总线支持 127 个设备的连接。
低速或全速设备	USB 支持两种设备传输速率：1.5Mb/s 和 12Mb/s。较低的那个传输速率能够实现低速/低成本的 USB 设备。这可以归结于降低了所使用的数据线的成本，这种数据线不需要屏蔽。
数据线供电	外围设备能够直接通过数据线进行供电。5 伏的直流电压可以直接加在数据线上。至于电流的大小则取决于集线器的端口，它的范围可以从 100mA 到 500mA。
不需要系统资源	不像 ISA、EISA、PCI 设备，USB 设备不需要内存和 I/O 地址空间，而且也不需要中断请求线路。
错误检测和恢复	USB 事务处理包括错误检测机制，它们用以确保数据无错误发送。在发生错误时，事务处理可以重来。
电源保护	如果连续 3ms 没有总线活动的话，USB 就会自动进入挂起状态。处于挂起状态的设备消耗的电流不超过 500μs。
支持四种类型的传输方式	USB 定义了四种不同的传输类型来满足不同设备的需求，这些传输类型包括：块传输、同步传输、中断传输和控制传输。

USB 模式

USB 从以前是和老式 PC 的 I/O 实现有关的系统资源的问题中解放出来，这些资源限制与 I/O 地址空间、IRQ 以及 DMA 通道相关，但是在 USB 的实现模式下就不存在这些问题了。每一个位于 USB 上的设备都被分配了一个地址，整个地址只有 USB 子系统知道，并且它不占用任何系统资源。USB 支持最多 127 个地址，它限制了在一个 USB 端口上所支持的 USB 设备的数目。USB 通常包含一定数量的专用寄存器和端口，它们可以被 USB 设备驱动程序间接存取。这些寄存器被称作 USB 设备的端点。

当一个事务处理在 USB 总线上发送时，所有的设备（除了低速设备）都将看到事务处理。每个事务处理从一个包传输开始，这个包不仅定义了事务处理类型，还定义了 USB 设备和端点地址。这些地址由 USB 软件进行管理，而其他非 USB 设备和在系

统中相关的软件不会和这些地址发生冲突。

每个 USB 设备都必须有一个端点地址 0，它是为配置保留的。通过端点 0，USB 系统软件从设备那里访问 USB 设备的描述符。这些设备描述符提供了关于设备标识、端点数目和每一个目标地址的信息。在这种方式下，系统软件可以检测设备的类型或类别，并且确定应该采用怎样的方式对设备进行访问。

怎样得到 USB 的规范说明文档

USB 的规范说明文档可以从这个 USB 的网站上获得：www.usb.org

这个网址有 USB 规范说明文档的 1.0 版本和设备类型说明书，还有一些别的信息，它们都和 USB 有关。

第二部分　USB 解决方案

USB的总体情况

✎ 上一章

上一章讨论了基本的设计目标，这些目标都是为了克服那些传统的实现所带来的问题而提出的。此外，还讨论几种可能的解决方案。在这几种解决方案中，包括 USB 总线的解决方案，并且着重介绍了它的关键特征。

✎ 本　章

本章概括性地描述了 USB 总线的传输基本概念，介绍了 USB 总线系统软件、系统硬件和 USB 设备之间的交互。描述了 USB 总线的通信过程、设备框架的概念。在 USB 总线系统中的每一个硬件和软件元素都加以介绍，对这些元素的基本功能也进行了说明。

✎ 下一章

USB 总线定义的一种连接器类型，用于把 USB 总线外围设备连接到主机系统上。下一章介绍了 USB 连接器和所用数据线的物理特征。

🕮 概述

图 3-1 提供了一个基于 PCI 系统内 USB 实现的系统视图。USB 主控制器位于 PCI 总线上，他获得了一个描述 USB 事务处理的列表，该列表上的项是由系统软件安排好的，他们将在 USB 总线上发送。控制器将会按照事务处理列表依次执行列表上的每一项。本章将对通信过程作一个概述，后续章节将详细描述 USB 所采用的机制和处理方法。

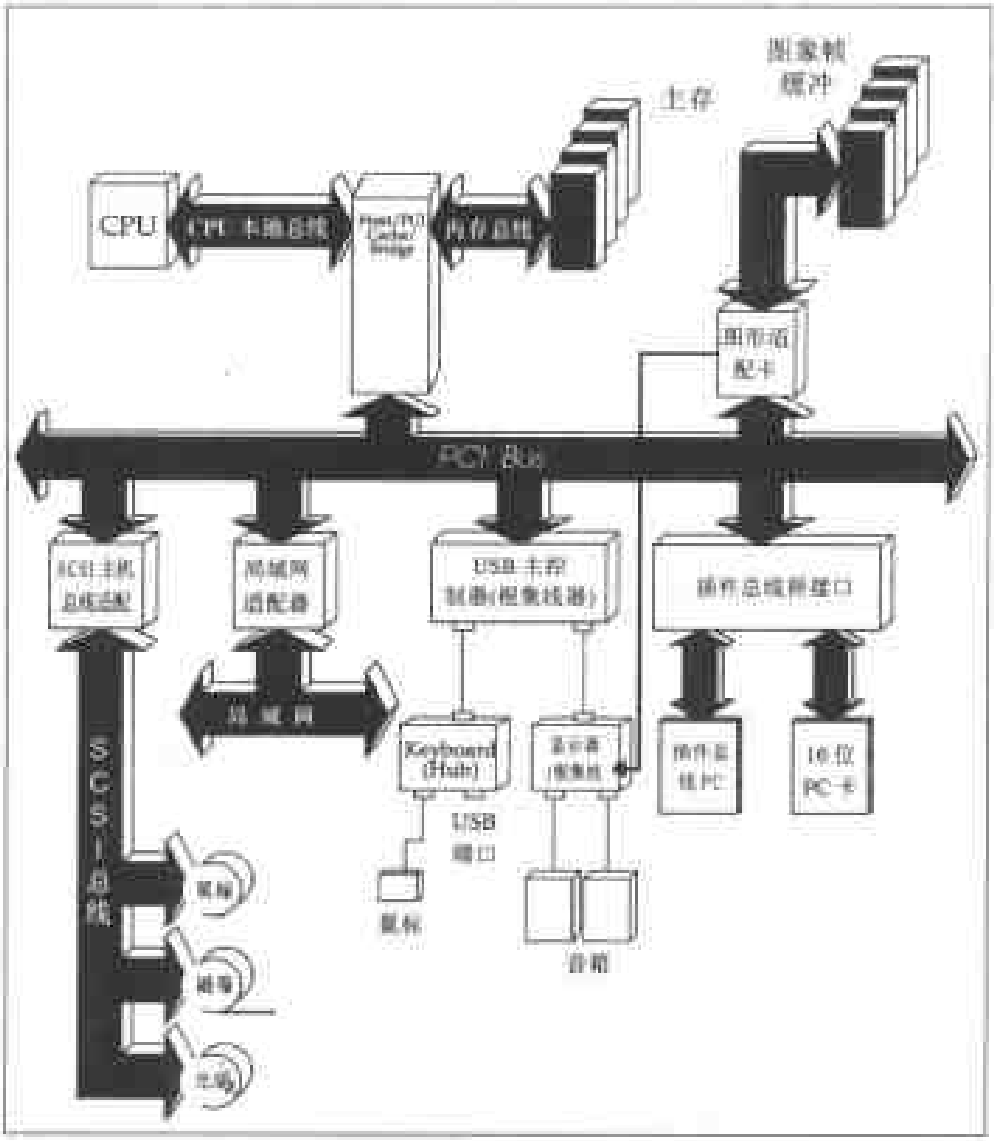


图 3-1　USB 系统在 PCI 平台上的实现

硬件和软件元素

图 3-2 所示的就是包含在 USB 系统中的硬件和软件元素。所有的 USB 事务处理都由 USB 软件进行初始化。这些访问一般都由一个 USB 设备驱动程序产生，它们负责和 USB 设备进行通信。这些 USB 驱动程序提供了 USB 设备驱动程序和 USB 主控制器之间的接口。这些软件负责把客户请求转换为一个或多个事务处理，他们被直接送往一个目标 USB 设备或者从一个目标 USB 设备发出。

和 USB 解决方案相联系的基本硬件和软件元素包括：

- USB 硬件

- USB 主控制器/根集线器；
- USB 集线器；
- USB 设备。

● 通用性软件

- USB 设备驱动程序；
- USB 驱动程序；
- USB 控制器驱动程序。

下面的部分将介绍在 USB 传输中每一个组件所起的作用。在阅读下面内容时请参见图 3-2。这些硬件和软件元素的详细内容将在后续章节中陆续加以解释。

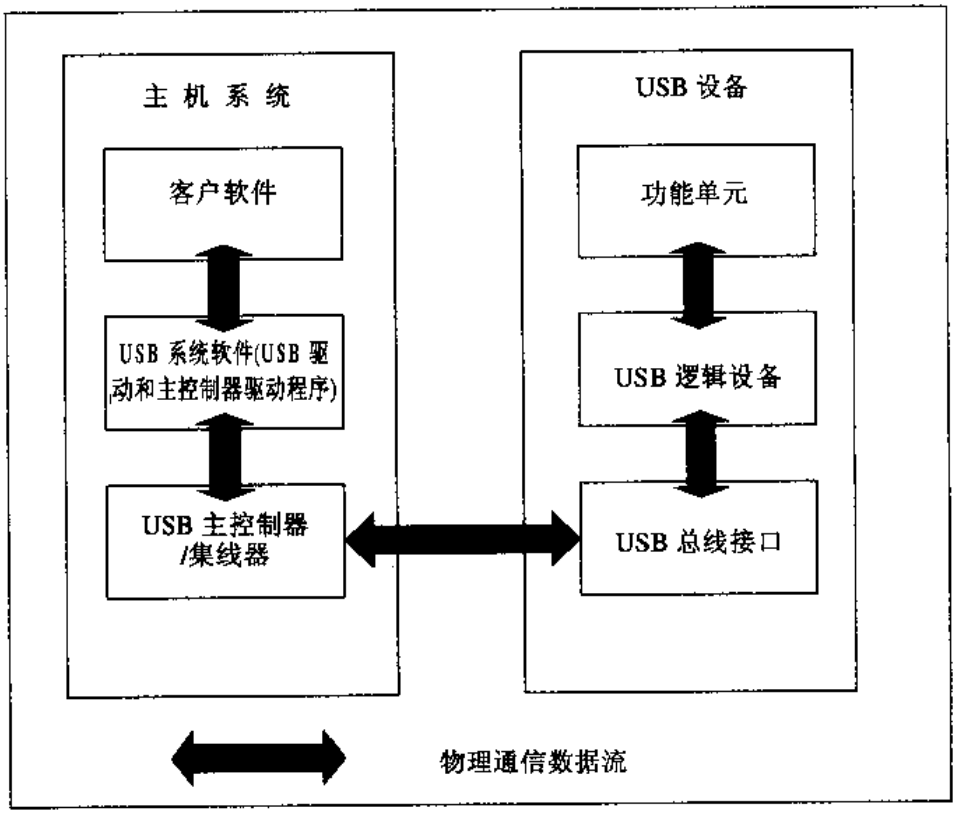


图 3-2　USB 系统中通信数据流

USB 设备驱动程序

USB 设备驱动程序（或客户驱动程序）通过 I/O 请求包（IRP）将请求发送给 USB 设备驱动程序。这些请求包将一个给定的传输初始化，这种传输可以来自于一个 USB 目标设备，也可以发送到 USB 设备。例如，一个 USB 键盘驱动程序必须初始化一个中断传输，这个中断传输是通过 IRP 建立的，此外，它还建立一个内存缓冲区，从 USB 键盘上返回的数据将被放到缓冲区内。注意，USB 的客户驱动程序并不知道 USB 的串行传输机制。

USB 驱动程序

USB 驱动程序知道 USB 目标设备的特性，也知道如何通过 USB 和设备进行通信。USB 特性由 USB 驱动程序检测，这是在设备配置过程中通过分析设备描述符得到的。例如，某些设备要求在每一个时间片（Frome）中都有一个确定信息吞吐量，而另外一些设备则可能只需要每隔一定的时间片才进行一次周期性的存取工作。

当从 USB 客户驱动程序收到一个 IRP 时，USB 驱动程序把该请求组织为若干个专门的事务处理（transaction），这些事务处理将在一系列一毫秒的时间片中执行。USB 驱动程序 Setup 事务处理，Setup 事务处理的基础是对 USB 设备的请求的了解，对客户驱动程序需要的了解和对 USB 的限制能力的了解。

根据操作环境的不同，USB 驱动程序可以是捆绑在操作系统中，也可以是以可装载的设备驱动程序形式作为一个扩展加入到操作系统中。

USB 主控制器驱动程序

主控制器驱动程序（HCD）安排事务处理在 USB 上广播。事务处理由主控制器驱动程序安排，方法是建立一系列的事务处理。每个列表由几个将要进行的事务处理组成，他们由一个或几个和总线相连的 USB 设备产生。一个事务处理列表或时间片列表定义了一系列的将要被执行的事务处理。他们将在一系列长度为 1 毫秒的时间片中被执行。USB 主控制器以毫秒的时间间隔执行这些事务处理。要注意是：USB 客户请求的一次快速传输可能被作为一系列的事务处理执行，他们被安排和执行于连续的长度为 1 毫秒时间片。事实上事务处理的安排要取决于多种因素，包括：事务处理的类型、设备指定的传输要求以及其他 USB 设备的事务处理状况。

USB 主控制器通过它的根集线器或者集线器初始化事务处理。每个时间片都从一个时间片起始（SOF）事务处理开始，随后是包含在当前列表中的所有事务处理的广播。例如，如果一个请求事务处理是关于把数据传递到 USB 打印机的请求，主控制器将从一个内存缓冲区中获得将要被发送的数据，该内存缓冲区由客户软件系统，并通过 USB 发送数据。控制器的集线器部分将请求的事务处理转换为低层协议，该协议是 USB 需要的。

USB 主控制器

在 USB 上的所有通信都在主机端产生，它是由软件控制。主机硬件组成了 USB 的主控制器，它初始化 USB 系统上的事务处理，根集线器为 USB 设备提供了连接点

（或端口）。目前已经开发出了两个USB主机主控器方案：

- 开放主控制器（OHC）；
- 通用主控制器（UHC）。

这两种主控制器执行同样的基本工作，只是在执行方式上有轻微的差异。第十八和第十九章讨论了这两种主控制器的操作。

☑ 主控制器

主控制器负责产生事务处理，这些事务处理已经由主机软件安排好。主控制器驱动程序（HCD）软件在内存中建立一个数据结构的连接列表。这些数据结构定义那些安排好的将要被执行的基本事务处理。这些数据结构被称为传输描述符，包含了主控制器产生事务处理所需要的全部信息，这些信息包括：

- USB的设备地址；
- 传输类型；
- 传输方向；
- 设备驱动程序的内存缓冲区地址。

主控制器对一个目标设备执行写操作，它从一个内存缓冲区（由USB设备驱动程序提供）读取数据，并把数据发送到目标及设备。主控制器对数据执行一个并行到串行的转换，建立USB的事务处理，而它传输到根集线器以后在总线上发送。

如果要求一个读传输，主控制器就建立一个读事务处理，并把它发到根集线器。根集线器在USB上发送读事务处理。目标设备认出地址，并且确定是所要求的数据，设备就把数据发回根集线器，根集线器再把数据传递到主控制器。主控制器对数据进行串行到并行的转换，并把数据放到放设备驱动程序的内存缓冲区中。

注意：USB根集线器和目标设备在一个事务处理过程中会进行错误检查，检测到的错误由根集线器识别出来，并送到主控制器，主控制器对错误进行记录，并向主机软件报告。

☑ 根集线器

事务处理由主控制器产生，他先被送到根集线器，然后发送到USB上，每一个USB事务处理在根集线器处产生。根集线器为USB设备提供连接点，并执行下面的关键操作：

- 控制它的USB端口的电源；
- 激活和禁止端口；
- 认出和每一个端口相连的设备；
- 设置和报告与每一个端口相连的状态事件（当主机软件进行查询的时候）。

根集线器由一个集线器控制器和中继器组成，如图3-3所示。集线器控制器对集线器自身的存取作出反应。例如，主机软件提出的请求，加上或断开某个端口上的电源，中继

器把事务处理传输到 USB 和主控制器或者从主控制器和 USB 传到中继器。

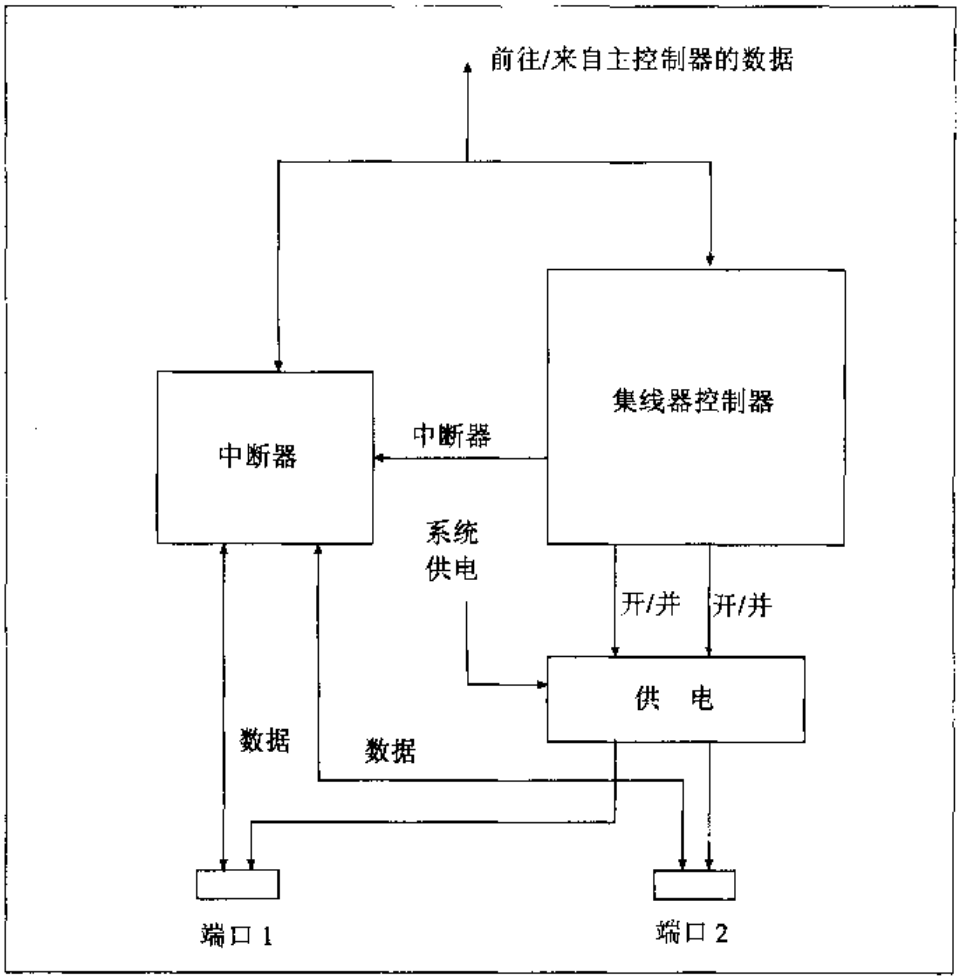


图 3-3 根集线器功能单元的分块图

USB 集线器

除了根集线器之外，USB 系统还支持附加的集线器，它允许对 USB 系统进行扩展，附加继电器提供了一个或多个 USB 端口用于连接其他的总线设备。USB 集线器可以被集成到一个设备内部，如键盘和显示器（称为复合设备），或者作为一个单独的设备实现，如图 3-4 所示。此外，集线器可能是由总线供电，就是说：从他自己处获得电源，并为和 USB 相连的所有设备供电。总线供电的集线器由于受到总线提供功率的限制，所以最多只能支持四个 USB 端口，第九章讨论了关于 USB 设备电源的内容。

集线器包含两个主要的功能单元：

- 集线控制器；
- 中继器。

图 3-5 表示了这些功能。

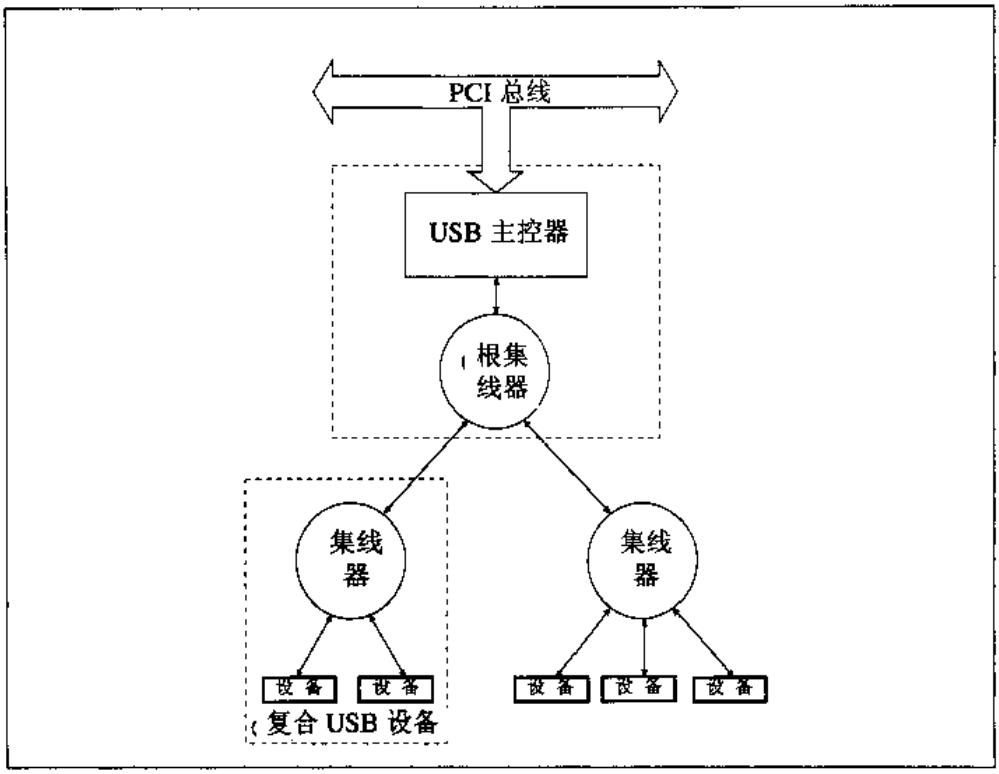


图 3-4　USB 集线器的类型

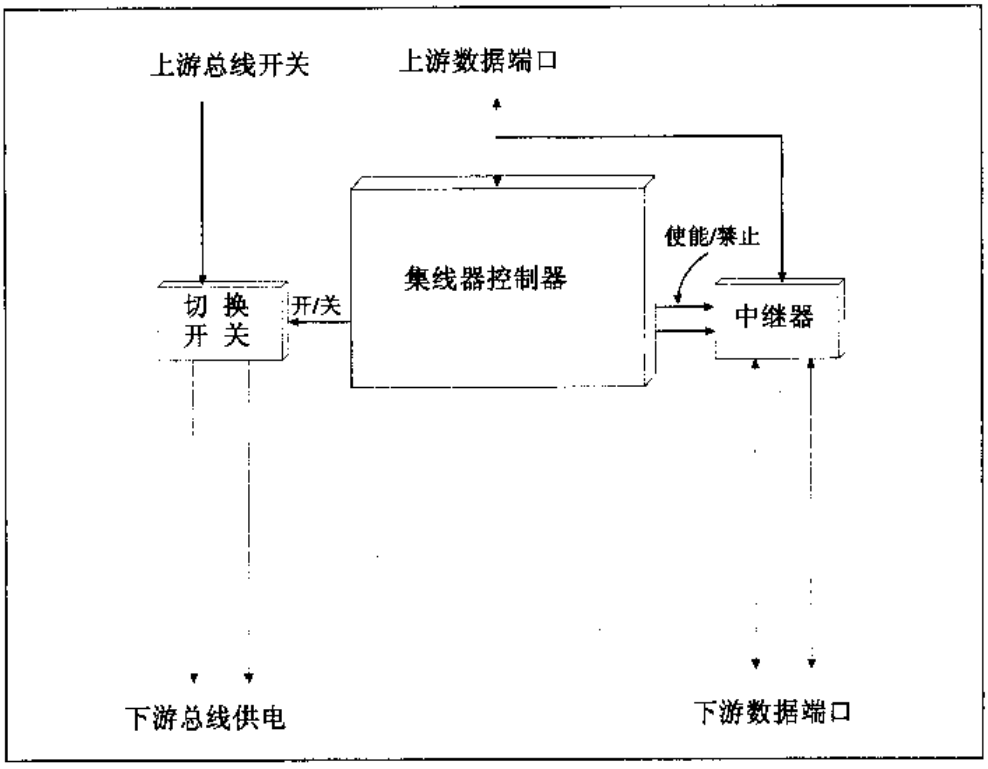


图 3-5　主集线器功能单元

☑ 集线控制器

集线控制器包含一个 USB 接口，和一系列的接口引擎（SIE）。它还包含了一些设备的描述符，软件通过读取设备描述符来识别集线器设备。集线控制器收集集线器和端口的状态信息，这些状态信息也由 USB 主机软件读取，用来检测设备的连接、断开并确定其他的状态信息。控制器还接受来自主机的软件的命令，控制主机操作的不同方面（例如：上电和激活端口）。

☑ 集线中继器

参考图 3-6。到达集线器的总线数据流必须向上游传递（就是朝主机方向）或者向下游传递（远离主机方向）。在主机上产生的传递将到达集线器的根端口，并且必须被传到所有活动的端口，当一个目标设备对一个主机初始化的事务处理做出响应后，它必须发送一个对上游的响应，就是说集线器必须把数据从下游端口传到上游的根端口。

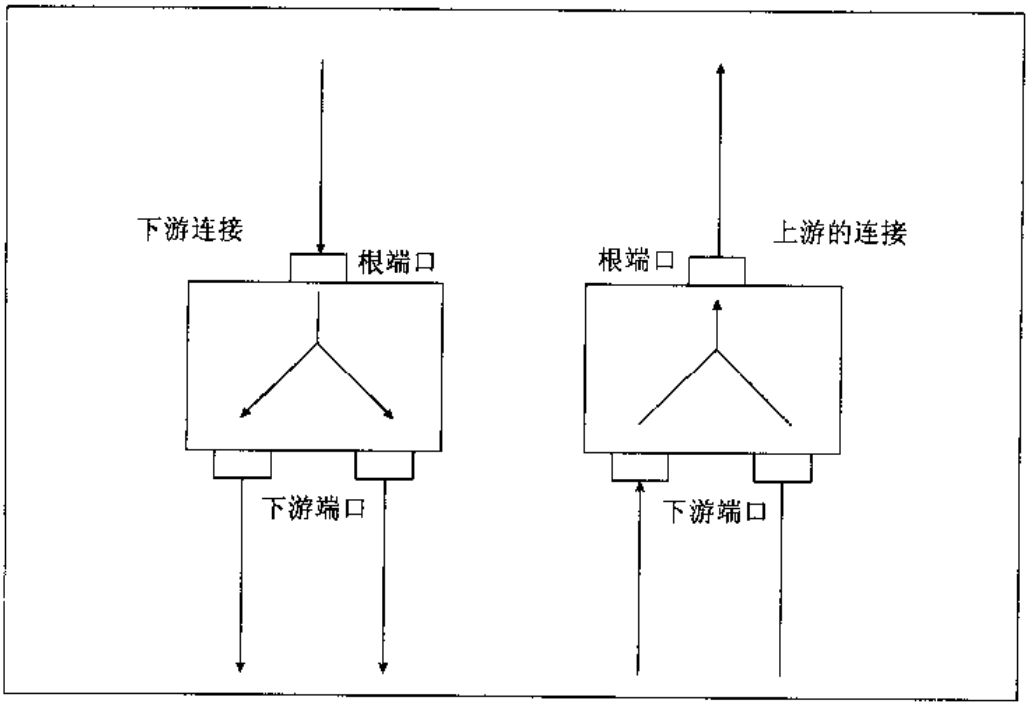


图 3-6　集线中继器和上游端口与下游端口的连接

☑ 配置过程中集线器的作用

集线器还在 USB 设备的热插拔（在运行时自动检测和配置）过程中起着一个很重要作用。在一个设备和集线器接上或者和集线器断开时，集线器必须能够把这些动作识别出来，并且当主机软件对集线器进行查询时集线器能够向主机报告这些事件。

USB 设备

USB 设备包含一些设备描述符，它们指出了一个给定设备的属性和特征。这些设备描述符向主机软件提供了一系列 USB 设备的特征和能力，用于配置设备和定位 USB 客户软件的驱动程序。USB 设备驱动程序也可以用设备描述符来确定需要的附加信息，这些信息用于保证以正确的形式对设备进行访问。这项机制被称为设备构架，软件必须理解这个机制，因为软件用它来正确地配置和访问设备。在标题为“设备构架”的部分有关于这部分内容更加完整的讨论。

正如前面所提到的，USB 设备既可以作为全速设备实现，也可以作为低速设备实现。

☑ 高速设备

高速设备可以看到 USB 上广播的所有事务处理，并可以作为全特性设备实现。这些设备接受并发送串行数据，最高传输速率为 12Mbps。

☑ 低速设备

低速设备不仅在吞吐量上有限制（1.5Mbps），而且在功能支持上也有相应的限制。进一步说：低速设备仅能看到后接一个前导包的 USB 事务处理。在全速事务处理的过程中，低速集线器端口保持非活动状态，它可以防止全速总线的通信通过低速数据线传送。前导包指出接下来的事务处理将以低速广播。集线器在检测到一个前导包后，将激活低速端口，允许低速设备看到低速总线活动。

USB 的通信模型

USB 设备不直接消耗系统资源，这一点它和位于其他一般总线结构上的设备不同，事实上就是 USB 设备不映射到内存或者输入输出地址空间，也不使用 IRQ 或者 DMA 通道。此外，所有的事务处理都是由主机系统产生的。USB 系统需要的系统资源仅仅是内存位置，内存位置由 USB 系统软件使用，内存和（或）I/O 地址空间、IRQ 则是由 USB 主控制器使用。这就消除了以前很多标准外围设备实现过程中所遇到的困难，它们都要求大量的 I/O 地址空间和 IRQ。

通信流

图 3-7 所示的就是 USB 系统使用的基本通信流和系统资源。当 USB 客户应用程序

调用一个 USB 系统软件并且请求一次传输的时候，它就对传输进行初始化。USB 客户驱动程序提供一个内存缓冲区，用于传输数据时进行数据的存储，数据传输可以是传向 USB 设备，也可以来自 USB 设备。传输一般是在一个 USB 设备的给定寄存器（或端点）和客户设备驱动程序之间进行的，每次传输都通过一个通信管道进行，通信管道是在设备配置的过程中建立的。USB 系统软件把客户请求划分为单独的事务处理，它们和设备的总线带宽需求以及 USB 协议机制保持一致。

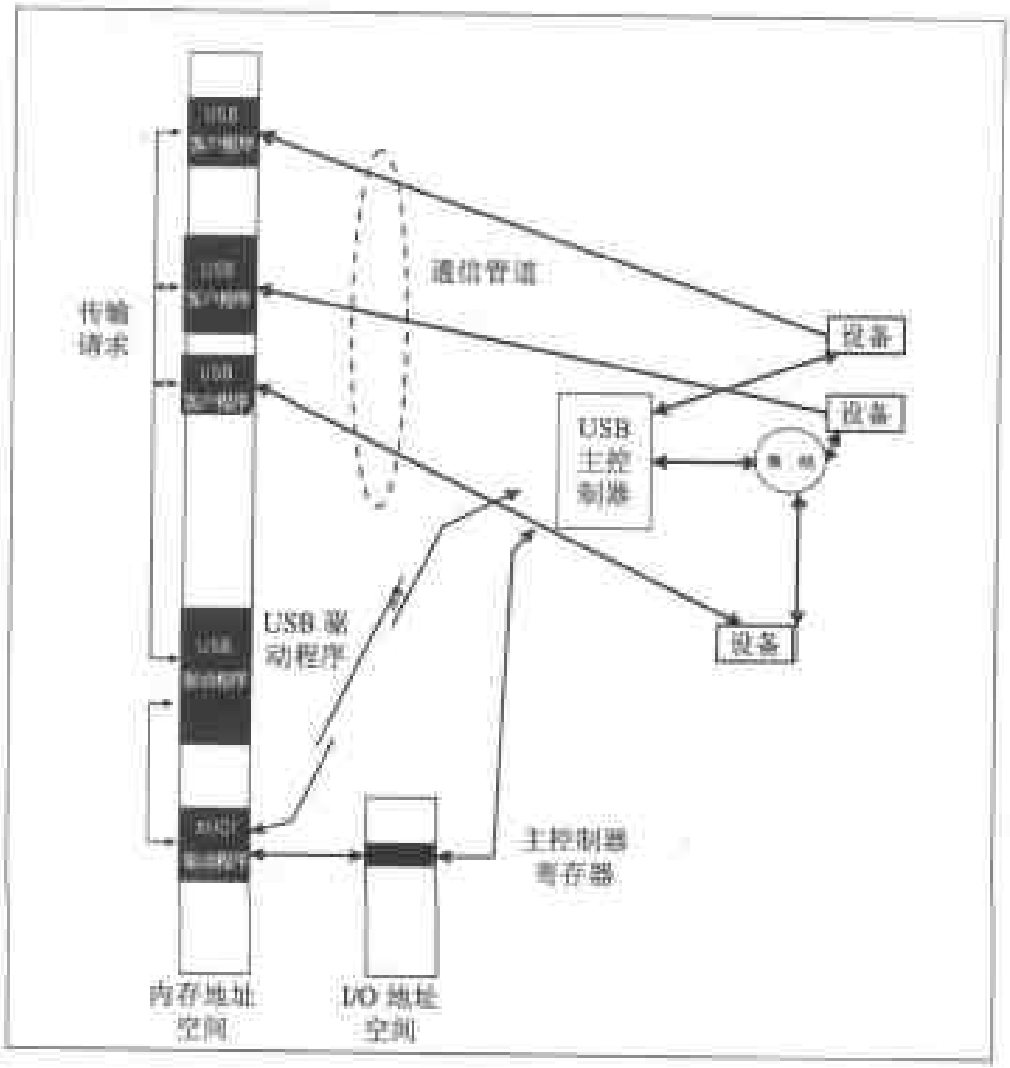


图 3-7　通信模型

这些请求被传递到 USB 主控制器驱动程序，驱动程序将会安排这些事务处理在 USB 上的执行顺序，主控制器执行的事务处理基于传输描述符的内容，它由 HCD 建立。HCD 知道所有必要的信息，这些必要的信息用于通过 USB 执行请求的事务处理。包含在一个传输描述符中的关键信息包括：

- 目标 USB 设备的地址；
- 执行传输的类型；
- 信息包的大小；

- 客户内存缓冲区的位置。

主控制器可以有一些寄存器，这些寄存器被映射到处理器的 I/O 和内存地址空间。这些寄存器控制主控制器的操作，并且它们必须由 HCD 载入数值，以保证能执行所希望的操作。例如，一个装入地址指针的寄存器指出了内存位置，在这个内存位置处存在传输描述符。

主控制器获取主控制器驱动程序产生的传输描述符。主控制器产生传输设备描述符指出的每一个描述符都定义了一个给定的事务处理，这项事务处理必须执行，以满足客户程序的传输请求。每一个事务处理都导致数据从客户缓冲区传输到 USB 设备，或者从 USB 设备到数据缓冲区，这取决于传输的方向。当传输完成以后，USB 系统软件就会通知客户驱动程序传输已经完成了。

传输，IRP，时间片和包

本章中的图 3-9 就是在 USB 的通信过程中所使用的机制，以及在 USB 系统中每两层之间的关系。传输由客户驱动程序初始化，这时，它向 USB 驱动程序发送一个传输请求。此外，事务处理通过底层打包的事务处理在 USB 上执行。下面的部分将讨论完成一次 USB 传输所需要涉及到的每一个层。

☑ 传输

每个 USB 功能的设计都和一些寄存器相联系，当客户驱动程序使用它的功能时，就要用到这些寄存器。每个端点都有自己支持的特殊的传输特性。例如，当把信息传输到一个扬声器时，数据传输必须以一个恒定速率进行，以避免音频失真。其他端点也可以有不同的特性，因而也就需要不同的传输类型。USB 所支持的传输类型包括：

- 等时传输；
- 块(bulk)传输；
- 中断传输；
- 控制传输。

客户驱动程序能够理解和每一个传输相关的特性，这些端点和它的功能相关。这些信息是通过读取设备的描述符来获得的。第 6 章分别描述了每一个传输类型的特性。

☑ USB 驱动程序，IRP 和时间片

当一个客户驱动程序希望执行一些传输时，不管这种传输是来自或者是传向一个给定的端点，它都会调用 USB 驱动程序来初始化这次传输。这次请求的传输被称为一个 I/O 请求包（IRP）。一些传输是由大块数据组成的。由于 USB 是一个共享的总线（就是说，很多设备可以同时使用同一条总线），所以一般说来某个设备不能一次在 USB

上执行整个的一次传输。事实上，一次传输需要分为几个部分，这些部分被分散在很长的一段时间上执行，采用的是分段执行的办法（这些分段就被称为事务处理）。这就保证了可以把带宽分配给总线上的其他的设备。

USB 通信是基于这样一种方式的，它们以 1 毫秒的时间片为单位来进行数据传输。每个 USB 设备在这些长度为 1 毫秒的时间片内要求得到一部分 USB 带宽。带宽的分配依赖于设备请求的吞吐量（这是由设备描述符指出的）和还没有被其他 USB 设备使用的带宽。一个 USB 设备被连接上以后，系统软件就要对它进行配置。首先系统软件分析它的设备描述符，以确定它所要求的带宽。然后软件就会检查当前剩下的未分配带宽。如果请求的带宽可以得到满足，那么这个设备就被正确地配置。如果设备要求的带宽不能都得到满足，那么设备就不能被正确地加以配置，而且用户将会得到设备没有正确配置的通知。

图 3-8 列出了一些和 USB 相连的设备，以及一些可能的事务处理，这些事务处理可能在一个 1 毫秒时间片内执行。这是一个人为的例子，目的是演示 USB 时间片的共享特性。并不是每一个 USB 设备都必须要在每一个时间片中传输数据。例如，主机软件将每隔若干个时间片对键盘进行击键查询。在每一个时间片中，系统给设备分配它所要求的那部分总线带宽。对于数据传输量很大的块传输，例如打印工作，它就会被分配到大量的时间片上。实际需要的时间片的数量需求依赖于几个方面，包括：打印机的 USB 接口的传输能力，对块传输所加的特定的限制，以及其他设备使用的总线带宽的数量，当然这些设备是指当前安装在 USB 上的设备。

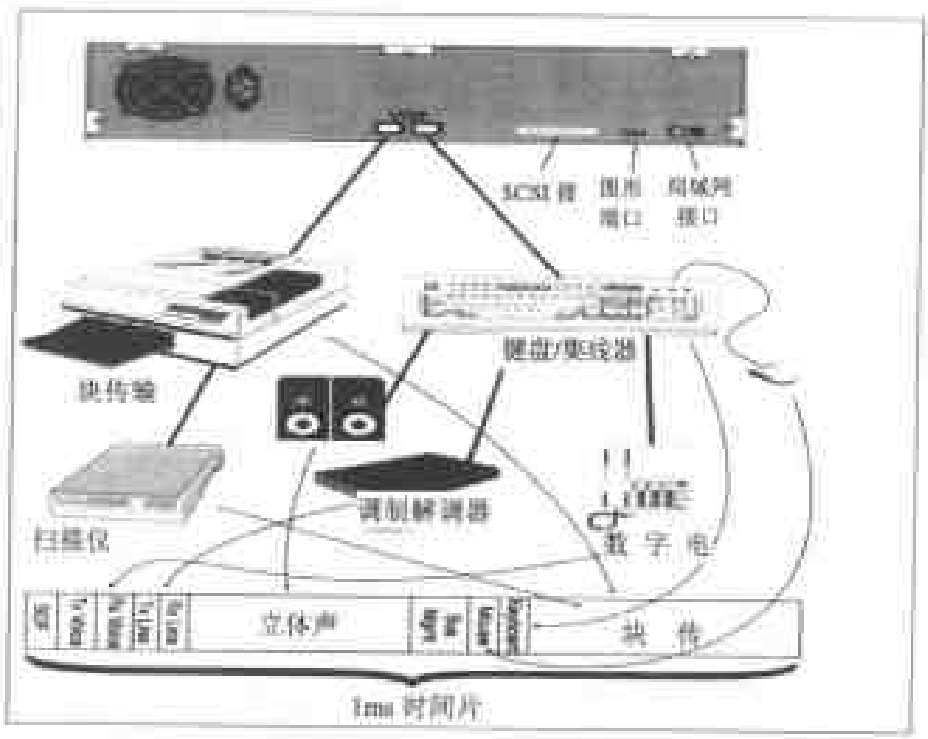


图 3-8 在一个时间片内 USB 设备执行的传输

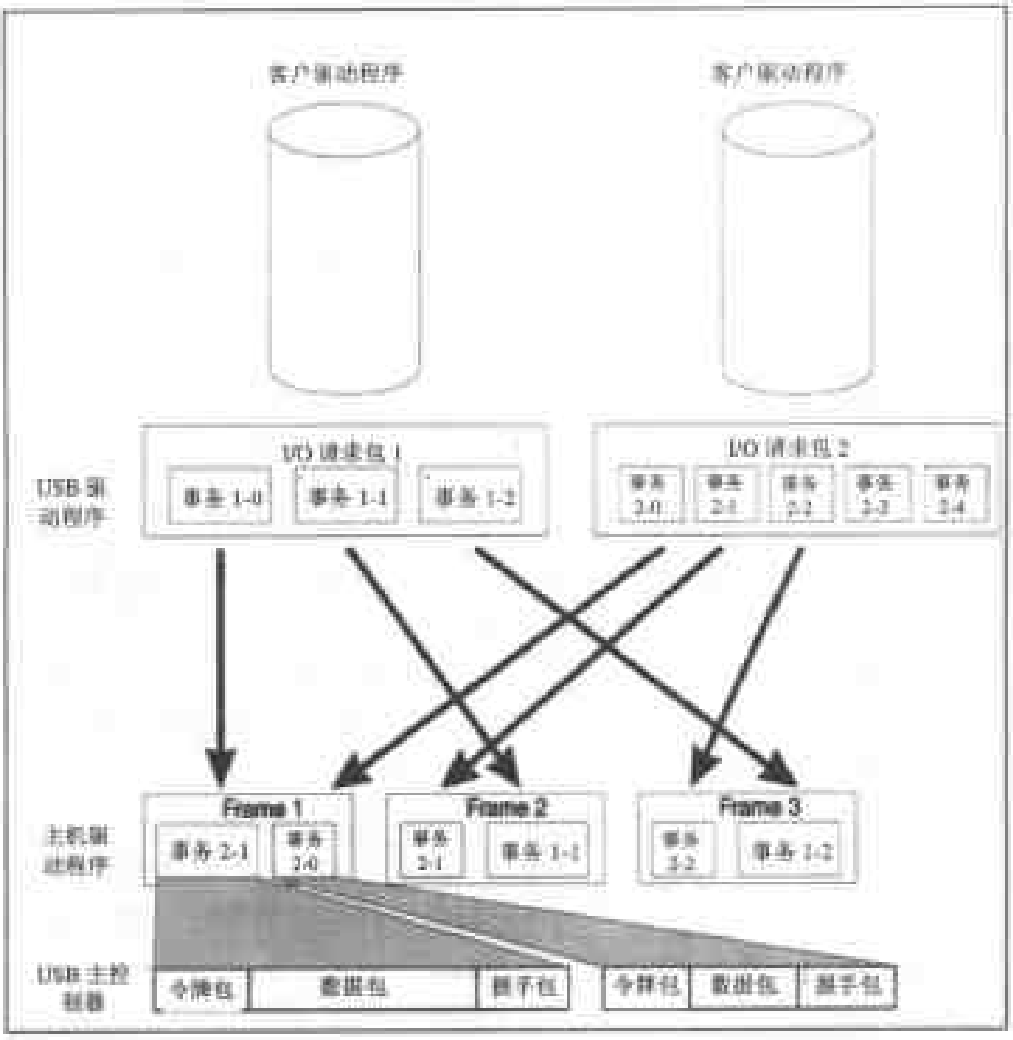


图 3-9　ZRPs、传输、时间片和数据包之间的关系

☑ 主控制器驱动程序和事务处理

主控制器驱动程序从 USB 驱动程序处得到包请求，并安排它们在一系列的时间片上执行。安排的顺序取决于主控制器驱动程序定义的法则。这个法则基于 USB 传输能力和限制。（它们将在后续章节中加以讨论）。

具体的安排工作是通过建立一系列的数据结构来进行的（称为传输描述符），它定义了在 USB 上将要执行的每个后续的事务处理。主控制器读取和解释这些传输描述符，并执行这些传输描述符所描述的 USB 事务处理。

☑ 主控制器和信息包

主控制器和根集线器在 USB 上产生事务处理。事务处理由一系列的信息包组成，一般说来包括令牌包、数据包、握手包。关于事务处理和信息包的进一步信息请参阅

第七章。

设备构架（设备如何让软件识别自己）

USB 的设计实现了类设备驱动程序。一套有相似属性和提供相似服务的设备都被定义到一个给定的设备类别中。这些常用的设备类拥有一个通用的类驱动程序，该驱动程序可以为所有这个类别的设备提供驱动。

设备描述符

一个设备通过许多设备描述符向主机软件描述它自己，如图 3-10 所示，这些描述符包括：

- 设备描述符——每个设备有一个设备描述符，包含了关于缺省通信管道的信息。它用来配置设备，还包括设备的一般信息。这些设备描述符还标出了一个设备所支持的可能的配置数量（一个或多个）。
- 配置描述符——一个设备对它所支持的每一种配置都有一个配置描述符。例如，一个高功率设备也可以支持一个低功率的模式，从而导致了每一个供电模式都有一个配置描述符。这种配置描述符包括关于配置的一般信息，并且定义了当使用这些配置的时候接口的数量。
- 接口描述符——一个给定的配置可以支持一个或多个接口。多接口设备的一个例子是 CD-ROM，在这种情况下，有三个设备驱动程序可以用来访问设备的不同的功能单元。一种设备驱动程序用于设备的大规模存储接口（用于文件存储），另一种用于音频设备驱动程序（用来播放 CD），还有一个用作视频图像驱动程序（用于播放图像）。

接口描述符提供了关于接口的一般信息。它也指出了特定的接口所支持的设备类。此外，它指出了该接口进行通信时所使用的端点描述符的数量。

- 端点描述符——一个设备接口包含一个或多个描述符，每一个描述符都定义了一个通信点（例如，一个数据寄存器）。端点描述符包含了一些信息：例如端点支持的传输类型，（就是指同步、块、中断和控制传输）以及支持的最高传输速率。
- 字符串描述符——字符串描述符可以为所有的设备定义，也可以为一个给定的定义，还可以为每一个接口定义。这些字符串描述符描述了配置和接口，并且字符串是以 UNICODE（统一字符编码标准）的形式给出的，它们可以被显示出来，而且用户可以读取它们。

- 类特定描述符——某些设备类除了标准描述符外，还需要其他的描述符，这些描述符由相关设备类说明书加以定义。

图 3-10 所示的就是一套描述符。在本例中，定义了两个单独的配置文件，每一个都包括两个接口描述符。这里没有列出可选的字符串的描述符，也没有列出任何类特定描述符，但是某些设备类可能会需要它们。

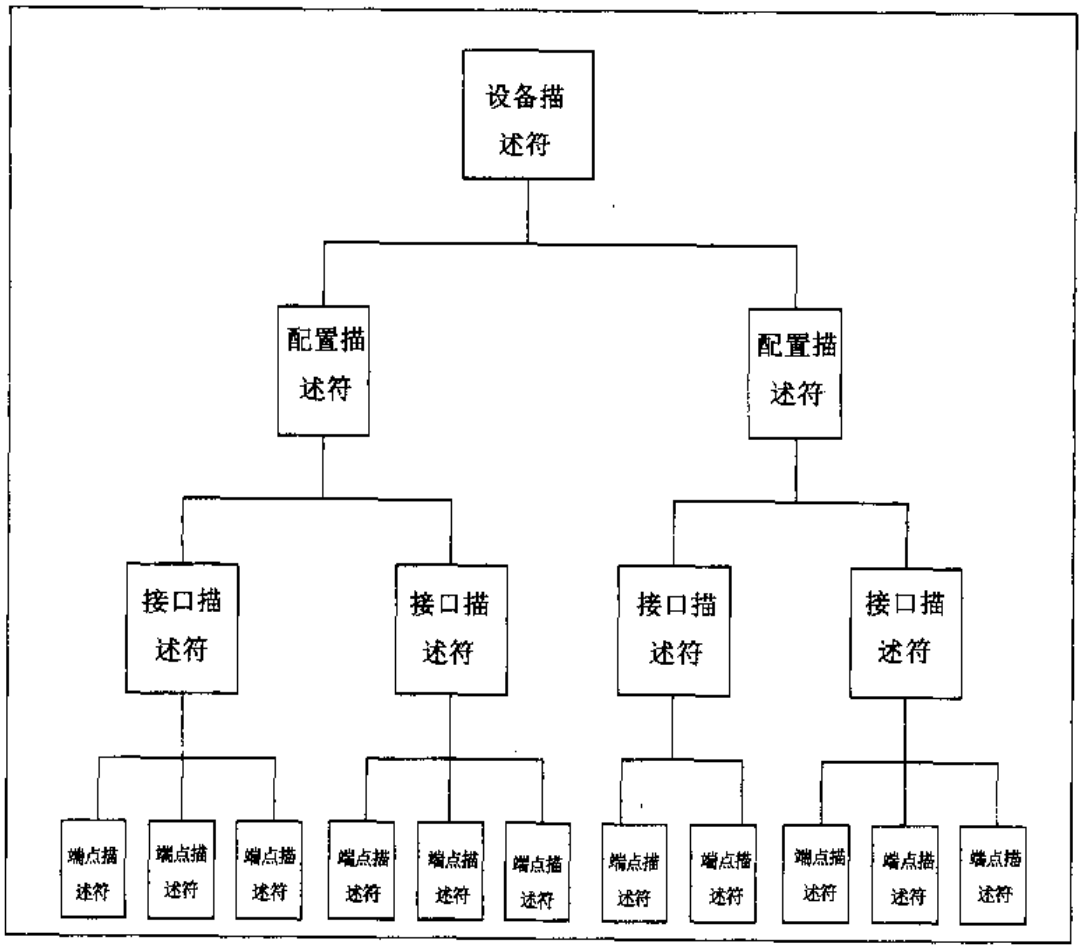


图 3-10　存储设备描述符

设备构架

设备框架提供了三个逻辑层，它描述了主机硬件和软件之间的关系以及每个 USB 设备相应的视图。图 3-11 所示的就是这些层以及主机和给定的 USB 设备之间的关系。这种分层的方法有助于解释每一层主机软件之间的关系以及它们在 USB 系统的职责。提供这些独立层的目的是简化读者对 USB 通信机制的理解，下面的部分将讨论这些层。

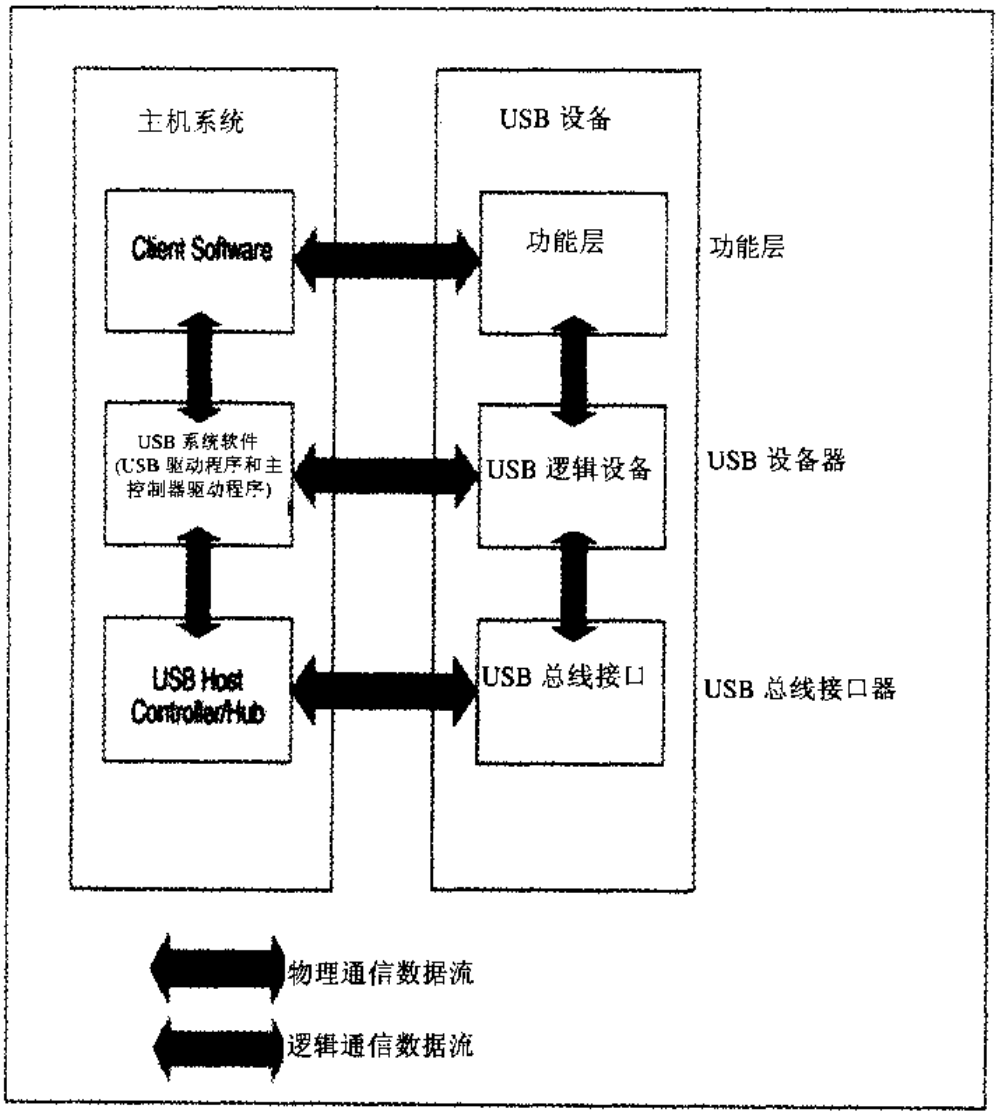


图 3-11　设备框架——硬件的软件视图

☑ USB 总线接口层

USB 总线接口层提供了在 USB 数据线上的数据的底层传输。这一层由下面几部分组成：

- 物理连接；
- 电器信号环境；
- 信息包传输机制。

这一层代表了通过 USB 数据线进行的实际数据传输，这种传输在主机系统和 USB 设备之间进行。主机一方由 USB 主控制器和根集线器组成，而 USB 方则由设备中的 USB 接口组成。关于在 USB 数据线上进行数据传输的细节可以参见后续章节。

☑ USB 设备层

USB 设备层代表了 USB 的一部分，它理解实际的 USB 通信机制和 USB 功能设备所要求的传输特性。这一层由主机方的 USB 系统软件和设备方的 USB 设备逻辑视图组成。USB 系统软件把一个逻辑设备看作一个端点的集合，它们组成一个给定的功能接口。

USB 系统软件提供一些服务，这些服务用于实现客户软件和它的 USB 功能之间的接口。USB 系统软件知道 USB 传输机制的具体细节，而且它必须为 USB 设备进行通信分配总线带宽。逻辑 USB 设备代表端点的集合，通过他们，客户程序可以和它的功能单元进行通信。USB 系统软件通过设备描述符来了解这些端点，由 USB 系统软件加以分析，获得该给定设备的传输特性。这些特性和系统软件对于 USB 传输机制的了解程度有关，当进行配置时，允许为每一个功能设备保留带宽。

USB 系统软件执行很多关键的功能，包括：

- 设备的连接/断开检测；
- 设备配置；
- 带宽分配；
- 管理客户程序和设备之间的控制流；
- 管理客户程序和设备之间的数据流；
- 收集状态和事务处理的统计信息；
- 事务处理的安排；
- 控制电气接口（例如，管理数据线功率的限制）。

注意一套 USB 系统软件存在于系统中，用于管理所有和 USB 总线相连的所有 USB 设备的访问。USB 系统软件是由以下部分组成的：

- USB 驱动程序（USBD）——为客户软件驱动程序提供接口和服务，分配总线带宽，并管理配置过程；
- USB 主控制器驱动程序——控制对主控制器的操作，安排事务处理，并监视事务处理的完成状态。

此外本书还提供了以上这些基本工作的主要描述。在第十六章有对软件层的更加易于理解的描述，标题为“USB 主机软件”。

☑ 功能层

这层代表客户软件和一个给定的设备功能接口之间的关系。每个接口都由一类特定的设备组成，每一类设备都有相应的设备驱动程序来操纵它。USB 客户软件不能像在别的环境下（例如 ISA、PCI、和 PCMCIA）一样直接访问它们的功能单元，由于它们没有被直接映射到内存和 I/O 地址空间，此外 USB 图形设备驱动程序必须使用 USBD 编程接口来访问它们的设备。

USB 客户驱动程序把他们的 USB 设备看作是由一个给定的接口组成，它知道如何操纵它们。USB 系统软件必须向 USB 客户程序报告接口的类型和其他设备描述符。

USB 外围连接

如前所述，USB 为系统的附属设备提供了一个简单的连接器类型，USB 支持两种不同的设备速率：

- 低速设备——1.5Mbps；
- 高速设备——12Mbps。

所有的 USB 设备通过一个 USB 集线器和系统相连，所有这些集线器提供了一个或多个端口。图 3-12 所示的就是几种和 USB 端口相连的设备，这些端口由系统提供。注意：一个集线器端口可以和一个全速设备相连，也可以和一个低速设备相连。一些设备，例如键盘和鼠标一般在低速环境下操作，而别的设备，例如数字电话则必须工作于全速状态。每个 USB 端口必须既能够既支持低速，又能够支持全速设备。除非该端口用于和某个设备永久相连。一个设备的速度在它连接到集线器一端的同时被检测到（参见第五章，标题为“信号环境”的部分，那里有进一步信息）。

当一个事务处理被主机系统初始化以后，所有的传输设备和所有的集线器将看到这一事务处理。每个事务处理包含一个地址字段，它标识目标设备或集线器，低速设备只能看见低速事务处理，它总是跟在一个高速的前导事务处理之后，它的作用是使所有的集线器激活它们的低速端口。

USB 使用差分信号来执行信息的串行传输，这种传输在根集线器和 USB 设备之间进行。由于 EMI 的差别，低速设备和高速设备所用的数据线在电气特性上是不同的，关于高速和低速数据线的电器特性可以参见第四章。

USB 设备的电源通过 USB 数据线提供，或者由和设备相连的本地电源提供。第九章详细描述了 USB 的电源事务。

拓扑

USB 采用了一个层次化的新结构，具体而言就是集线器为 USB 设备提供连接点。主控制器包含集线器，它是系统中所有的 USB 端口的起点。如图 3-12 所示，根集线器提供了一定数量的 USB 端口（本例中是 3 个），USB 设备和附加的集线器可以连接到那里。

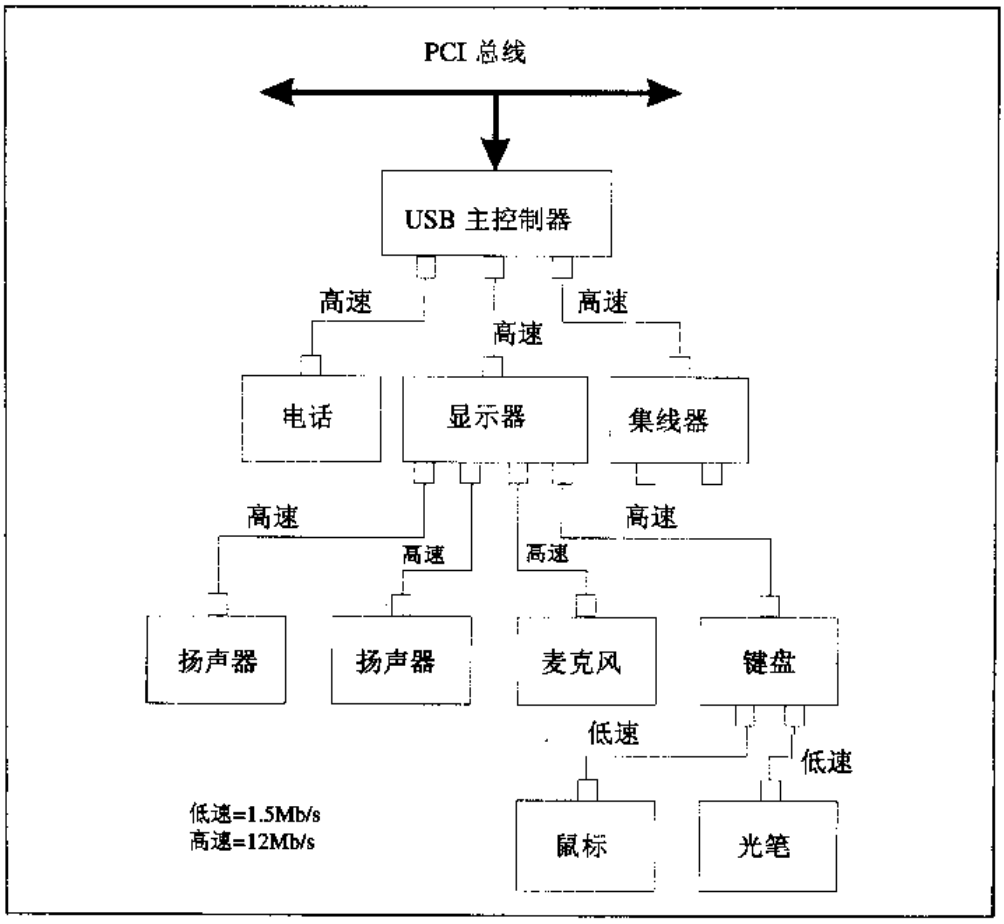


图 3-12　USB 的层次拓扑图

物 理 环 境

✎ 上一章

上一章对 USB 传输的基本概念作了一个概述，并且描述了 USB 的系统软件、系统硬件和 USB 设备之间的交互。对 USB 通信过程也加以了描述，包括设备框架概念。对 USB 系统中的每一个硬件和软件元素都加以介绍，并且对它们的基本功能做了一些说明。

✎ 本 章

USB 定义了一种连接器类型，用于把所有的 USB 外围设备和主机系统连接，本章描述了 USB 连接器和数据线的物理方面的特性。

✎ 下一章

USB 采用反向非归零（NRZI）编码形式，并采用了差分信号通过 USB 数据线传输信息。下一章讨论了 USB 所采用的差分信号和 NRZI 编码技术。

🕮 连接器

USB 连接器被设计为允许 USB 外设连接到集线器端口。集线器端口位于计算机的背面，也可以和其他外设相连接，例如显示器和打印机，还可以位于独立的集线器设备上。

很多 USB 外设有 USB 数据线永久连接，其他的则使用可分离的 USB 数据线，如果同样的连接器用于一个 USB 数据线的两端，它将还有可能在两个 USB 端口直接连

接。为了防止一个可分离的数据线被同时插入两个 USB 端口，专门设计了一个独立的连接器，用于外设电源连接。这两种连接器的类型描述如下：

- A 系列连接器——提供 USB 端口和 USB 外设数据线之间的连接。A 系列的插座是作为集线器端口连接器使用的，而 A 系列的插头和外设数据线相连，插头用来和 USB 外设相连；
- B 系列连接器——提供了和 USB 外设的数据线连接，当采用可分离的数据线时，就采用这种连接器。B 系列的插座在外设上提供，B 系列的插头和数据线相连接。

每个连接器有四个管脚，两个用于传输差分数据，两个用于给 USB 设备供电。注意电源管脚比数据管脚长，这样就能确保 USB 设备的外围设备能够首先收到电源信号，然后收到数据信号（电源管脚 7.41 毫米，而数据管脚为 6.41 毫米）。

连接器管脚都有编号，数据线导体使用颜色进行编码，这样就便于使用者识别，见表 4-1。

表 4-1　　连接器的管脚设计

管脚编号	信号名字	数据线颜色
1	Vcc	Red
2	－Data	White
3	＋Data	Green
4	Ground	Black

A 系列的连接器

A 系列连接器用于把一个外设数据线和一个 USB 集线器端口相连。该插座有四种变体，它们可以通过通孔或采用表面加载技术（SMT）来获得。这四种变体是：

- 垂直加载；
- 右角加载；
- 堆栈右角加载；
- 平面加载。

B 系列连接器

B 系列连接器用在有可分离数据线的外部设备上。规范说明书并没有为 B 系列插座定义加载变体；但是作者猜想 B 系列和 A 系列的插座变体的定义是一样的。

数据线

USB 规范说明书为 12Mb/s（全速）的信号定义了全速 USB 通道，为 1.5Mb/s（低速）的信号定义了一个子通道。全速通道要求一个特定的数据线，它是专门为与 EMI 兼容而设计的。低速数据线允许使用一种相对便宜的数据线，用于低速/低成本的外围设备，例如鼠标和键盘。下列部分详细描述了每一种数据线类型的特征。

低速数据线

图 4-1 表示一个低速数据线的横截面，也被称为子通道数据线。这些数据线仅仅用于 1.5Mb/s 的信号并且用于一个子通道的应用程序，那种情况下不需要大带宽。对于子通道应用程序来说，最大的数据线长度不应超过 3 米。

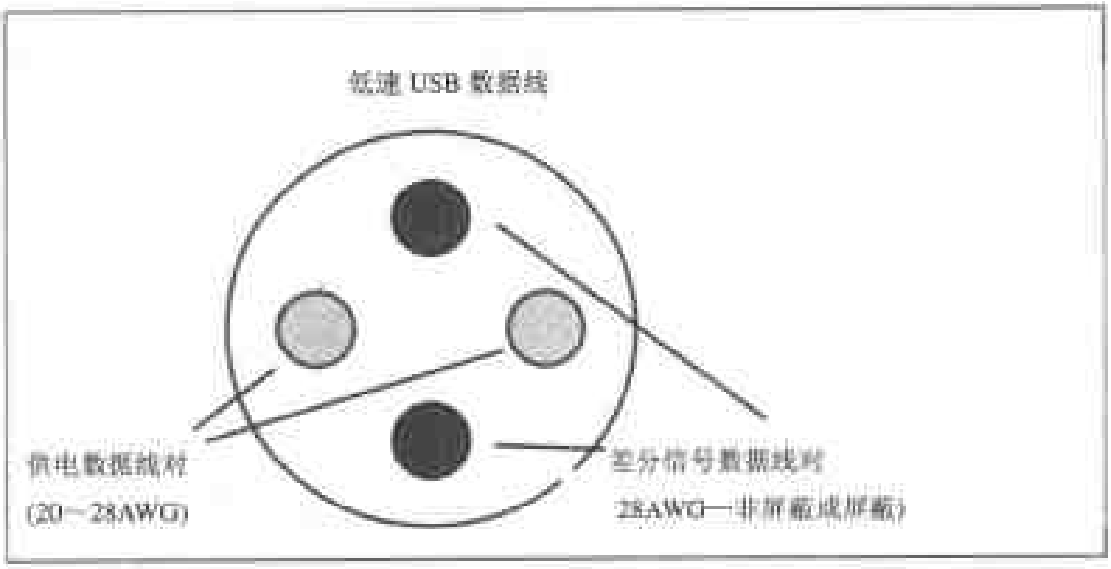


图 4-1 低速电缆段的横截面

差分数据信号对可以是非双绞线的 28AWG 标准导线。此外，低速数据线不要求屏蔽，除了上面提到的区别之外，规范说明书定义的全速和低速数据线在机械特性上是相同的。

全速数据线

全速 USB 设备不仅要求屏蔽数据线，而且这两条差分数据线必须是双绞线形式，如图 4-2 所示。高速数据线支持的最大数据线长度为 5 米。当操作的频率范围在 1～

16MHz 的时候，在这样的数据线长度上，最大的传输延迟必须小于或等于 30 纳秒。如果数据线不能满足传输延迟 30 纳秒的限制，那么该数据线就必须被截短，直到满足要求，可以参见表 4-2。

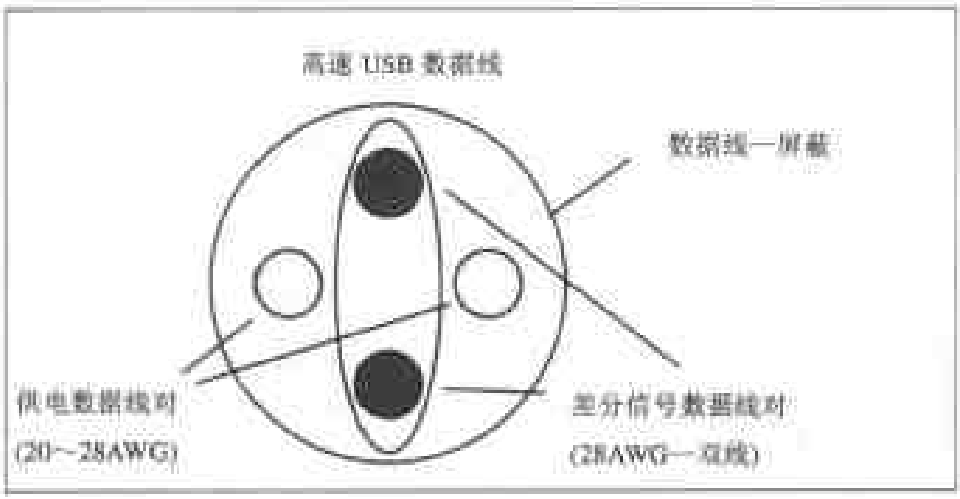


图 4-2　高速电缆段的横截面

表 4-2　　线路传播延迟

数据线传输延迟	最大数据线长度
9.0ns/m	5.3m
8.0ns/m	3.7m
7.0ns/m	4.3m
6.5ns/m	4.6m

数据线供电

数据线供电电压是 5V 直流电，可以用来为外设供电。数据线供电方式提供最多 500 毫安，最小 100 毫安的电流。某些外设可以用它自带的本地电源供电，这时它就可以不使用数据线供电。

电气和机械规范说明书

关于连接器和数据线的电气机械特性如果在本文中未能提及，那么读者可以参见 USB 1.0 规范说明书。关于如何获得规范说明书的信息可以参见本书前面的内容。

信 号 环 境

✎ 上一章

USB 定义了一种连接器类型，用于把所有的 USB 外围设备连接到主机系统上。前面的章节描述了 USB 连接器和数据线的物理方面的特性。

✎ 本 章

USB 采用了 NRZI（反相非归零）编码和差分信号，用于在 USB 数据线上传输信息。本章讨论 USB 所采用的差分信号和 NRZI 编码技术。

✎ 下一章

USB 支持四种传输类型，中断传输、块（bulk）传输、同步(isochronous)传输、和控制传输，这些传输类型，启动的过程和执行传输的过程将在下一章加以介绍。

🕮 概述

USB 串行数据是用 NRZI 进行编码的，编码过程是在通过 USB 数据线进行传输之前进行的。图 5-1 就是在通过 USB 数据线段进行的信息传输时包含的步骤。NRZI 编码首先由 USB 代理执行，它负责发送信息。接下来，编码后的数据被放入 USB 数据线，这是由差分驱动程序完成的。接收器放大传来的差分数据，并把 NRZI 数据发送到解码器，对数据进行编码和采用差分信号进行传输有助于确保数据的完整性和消除噪声干扰。

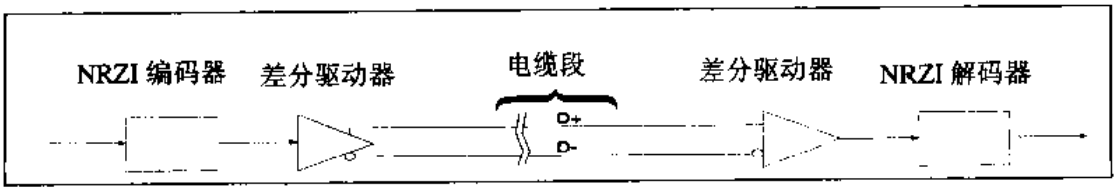


图 5-1　在 USB 上采用 NRZI 编码和差分信号的传输方式

检测设备连接和速度

在进行信息传输之前，无论数据是发送给 USB 设备还是来自给定的 USB 设备，主机软件首先都必须检测到 USB 设备是否存在。所有这些设备的设计思想是，当 USB 设备连接到集线器的端口上的时候，该设备就会自动被检测到。反之，当设备和系统断开连接时，系统也能够自动的检测到设备已经不存在了。用于检测设备连接所采用的机制同时也提供一种途径，可以用于检测该设备是一个全速还是低速设备。

一个 USB 的集线器通过监视差分数据线来检测设备是否已连接到自己的一个端口上。当然，前提条件是数据线的电源已经加在这个端口上了。参见图 5-2，当没有设备连接到 USB 端口时，和 D+和 D− 线相连的下拉电阻就能够保证了两条数据线都是近地的。USB 设备必须至少在 D+和 D− 线的任意一条上有一个上拉电阻（这取决于该设备的速度），如图 5-2 所示。当一个设备连接好以后，电流就会流过分压器，分压器是由集线器的下拉电阻器和 D+或 D−线上的上拉电阻组成。由于下拉电阻为 15 千欧，而设备的上拉电阻为 1.5 千欧，所以数据线上将会有升高将近 90%的 Vcc 电压。当集线器检测到一条数据线电压接近 Vcc 的时候，而其他保持近地电压，那么他就知道该设备已经连接好了。

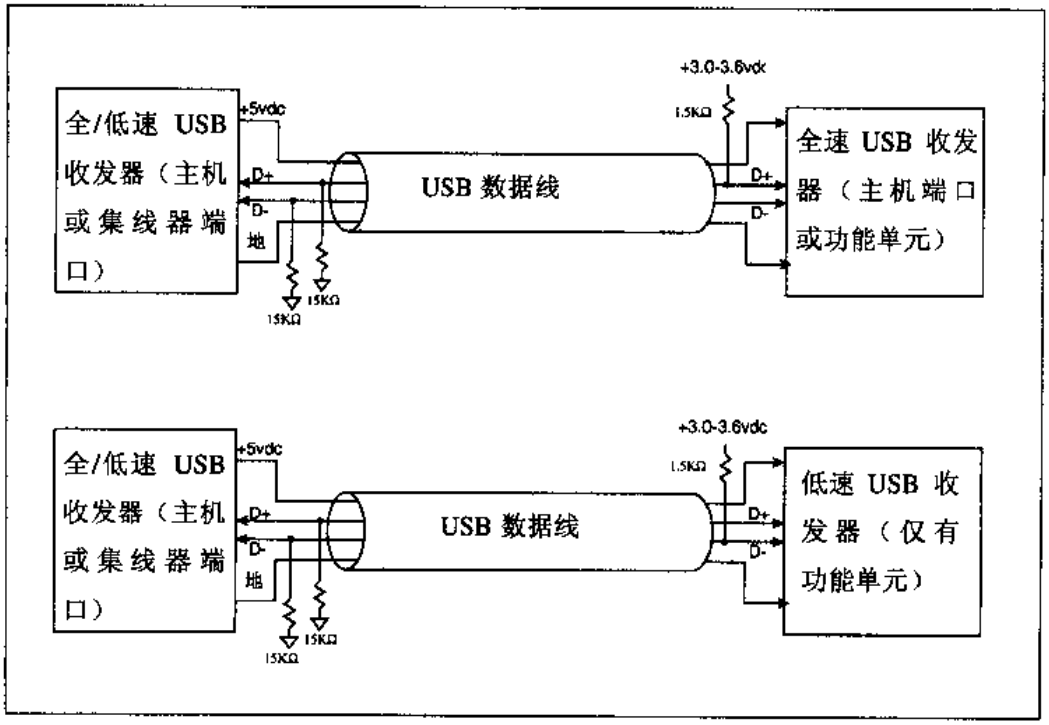


图 5-2　集成器和设备的电阻连接

从图中可以看到，全速设备在 D+线上有一个上拉电阻，而低速设备则在 D− 线上有一个上拉电阻，这样就可以识别设备的速度。图 5-3 所示的就是 USB 设备连接和

断开连接时的电压的分级。当 D+和 D—的电压都下降到 Vse（min）以下，或者说只有直流 0.8V 以下，而且这种情况持续了 2.5 微秒以上的话，就认为该设备器断开连接了。当 D+或 D—升高到 Vse（max）以上，或者说直流 2.5 伏以上的状态，并且这种情况超过了 2.5 微秒时候，集线器就认为该设备已经连接上了。当它检测到该设备已经连接上以后，集线器就在它的端口状态寄存器中设置适当的状态位，当它发现设备断开连接以后，它就把相应的状态位复位。主机软件周期性地检查每个集线器，以检查是否有设备和集线器进行连接或者断开连接。

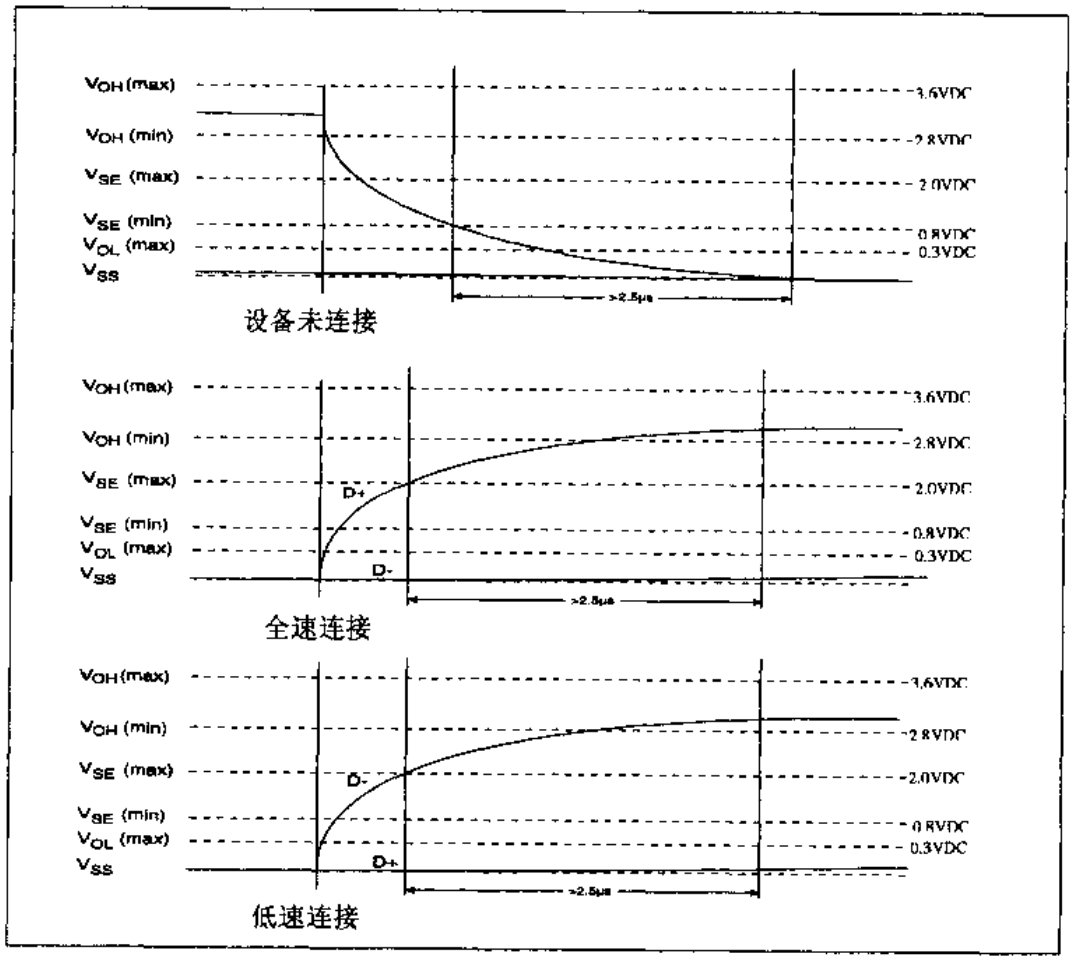


图 5-3　设备连接与分离

USB 设备有两种信号状态，对于低速和高速设备，这两种状态是相反的：

- J 状态；
- K 状态。

当一个设备一开始和 USB 相连的时候，它的系统数据线接近 Vcc，而另一个则近地，该状态被称为“J”信号状态，也就是该设备的闲置状态。当信号跳变发生时，两条数据线发生状态切换，导致上面的“J”状态转化为“K”状态。

NRZI 编码

数据通过 USB 传输采用的编码方式是 NRZI（反向非归零）编码方式。这种编码方式既能够确保数据发送的完整性，又不需要独立的时钟信号和数据一起发送。NRZI 编码绝非一种新的编码方式，人们使用它已经有几十年了，所以它的应用领域十分广阔。

图 5-4 所示的就是一个串行数据流和相应的经过 NRZI 编码的数据，在 NRZI 编码的数据流中的电平跳变代表 0，而没有电平跳变则代表 1。NRZI 编码器必须和传来的数据流保持同步，这样才能正确地对数据进行采样。NRZI 编码数据流必须在一个数据窗口内被采样，以检测是否从前一位时间到现在发生了电平的跳变。解码器在每一位时间对数据流进行采样，以检测是否发生了跳变。

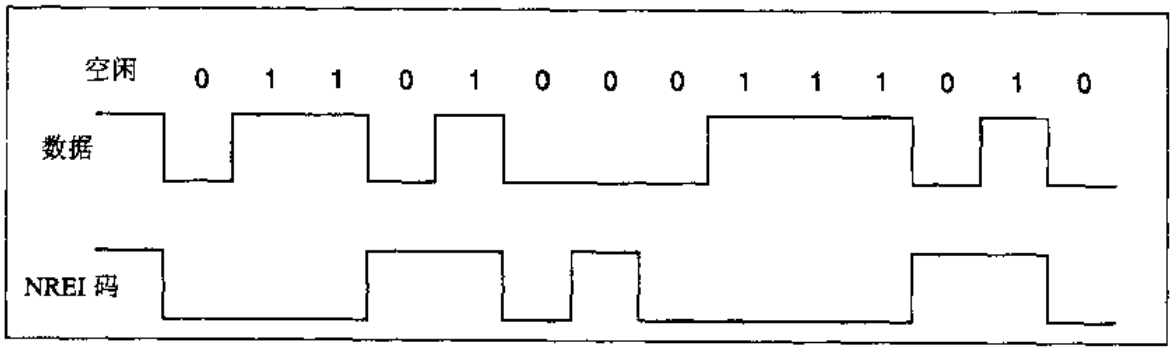


图 5-4　NRZI 编码的数据

数据流的跳变使解码器可以和收到的数据保持同步，因而可以不使用一个独立的时钟信号。但是要注意，一长串的连续 1 将会导致无电平跳变，从而引起接收器最终丢失同步信号，解决办法是采用位填充的办法。

位填充

位填充的做法是这样的，在连续传输 6 个 1 的情况下，强制在 NRZI 编码的数据流中加入跳变。这就确保接收器至少可以在每 7 个位的时间间隔内从数据流中会检测到一次跳变。这就使接收器和传送的数据保持同步。NRZI 数据的发送器负责在 NRZI 数据流中插入一个 0（填充位）。接收器必须设计为能够在 6 个连续 1 之后自动知道将有一个跳变发生，并且把跟在这 6 个连续 1 之后的那个 0 抛弃掉。

图 5-5 中，上面的线表示发送到接收器那里的原始数据。注意数据流包含了一个连续 8 个 1 的长串。第二条线代表加入了填充位之后的原始数据。可以看到在数据流中，这个填充位被插到第 6 个和第 7 个 1 之间。第 7 个 1 的发送延迟了一个数据位的

时间，这就是为了使填充位可以被插入。接收器知道第 6 个连续的 1 之后将会是一个填充位 0，所以就把它忽略了。需要注意的是，如果原始数据第 7 位是 0，填充位还是会被加入的，而且还是加在同样的位置，这就导致了在填充后的数据流中会有两个连续的 0。

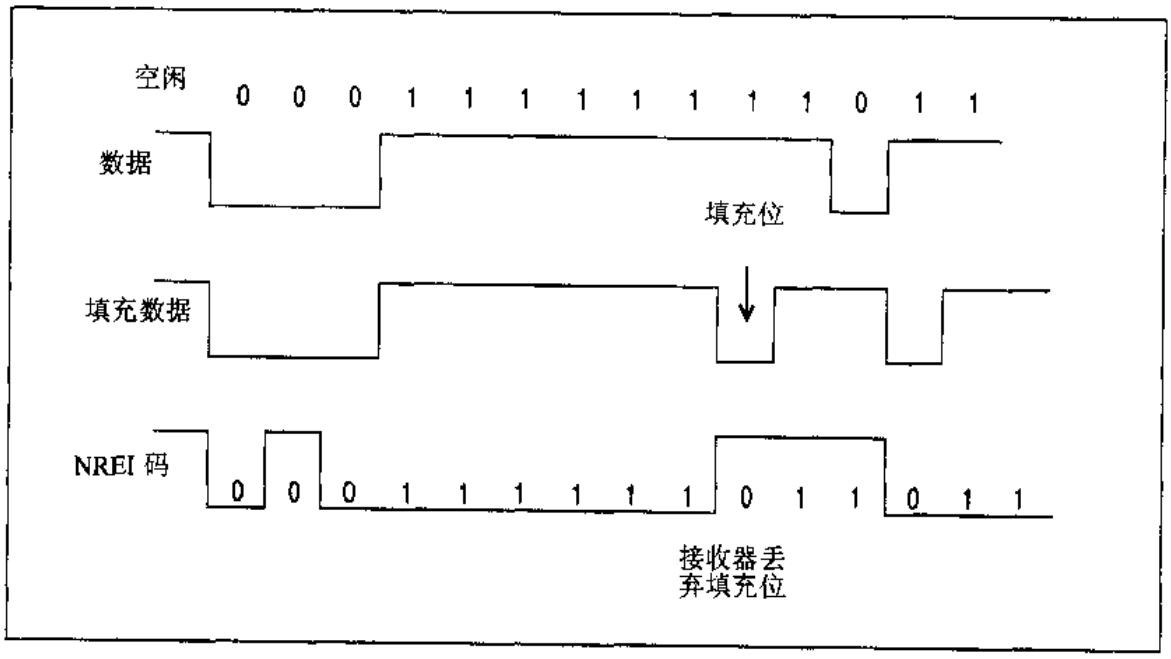


图 5-5 填充位

差分信号

USB 采用了差分信号来减少噪声。差分驱动器和接收器用来减弱信号噪声的来源，这些噪声来源包括：

- 放大器噪声——当信号被驱动器和信号接收器放大后，就会引入噪声；
- 数据线噪声——因为数据线上的电磁场干扰，也会引入噪声。

图 5-6 所示的就是在一个 USB 集线器和设备之间的数据信号。一个差分驱动器把差分数据放在了 D+和 D−线之间，他们之间的相位差为 180 度。在数据线的另一边的差分接收器对数据线之间的信号差别进行放大。因为在电脑上引入的噪声是同相位的，所以不会被放大，原始信号也就不会受到影响。

注意，差分信号是以半双工的形式实现的，就是说，数据线的任何一部分都可以传送和接收数据，但是在某一时刻只能进行发送或者只能进行接收，两者不能同时进行。半双工实现要求驱动器在不传送数据时进入高阻抗状态。

USB 设备也可以采用两个单端接收器，以识别特定的总线状态。例如当两条差分数据线的电压降低持续 2.5 微秒以上时，就会产生 USB 设备的复位信号。

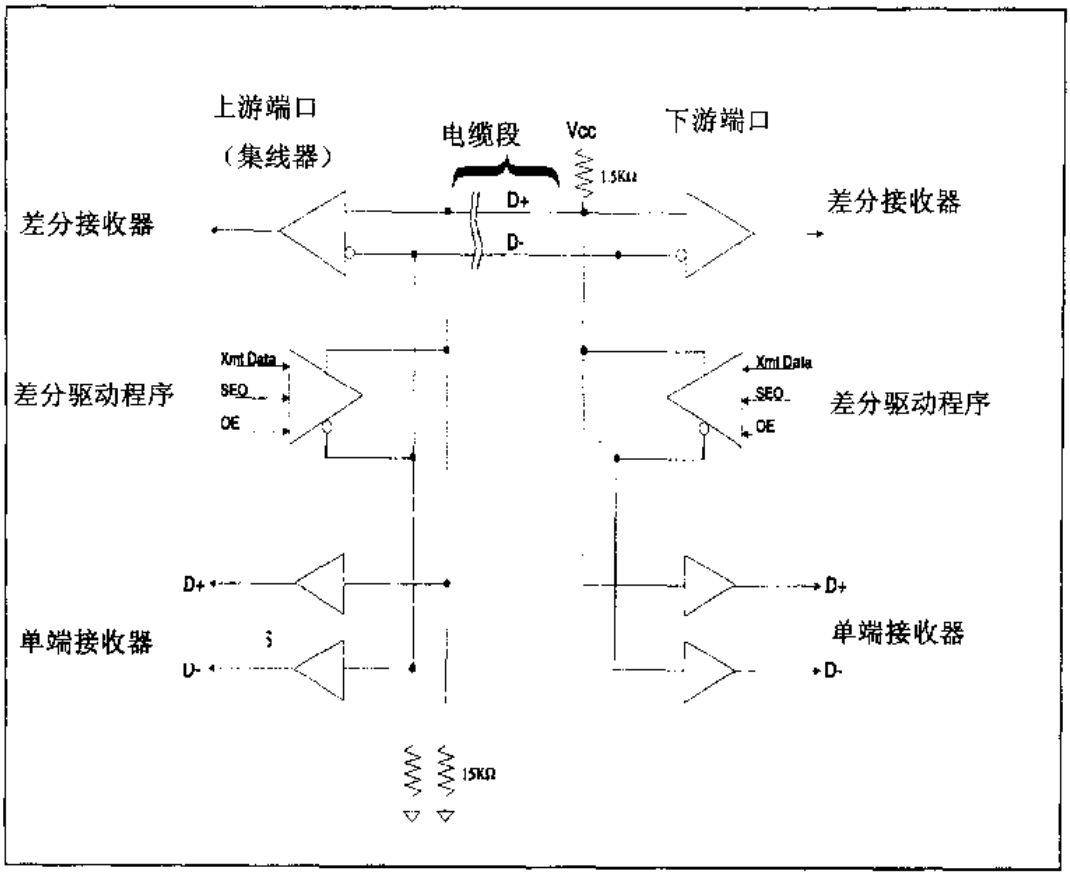


图 5-6　集线器和设备之间的信号接口

差分驱动器

USB 差分驱动器可以设计为低速、全速、或者既可用于低速又可用于高速的情况。一个差分驱动器采用反转和非反转缓冲区，输入信号可以应用于这两个缓冲区，产生两个输出（D＋和 D－）。驱动器的静态输出摆动在负载为 1.5 千欧时必须低于 V_{OL}（3.6 伏），而在负载为 15 千欧时必须高于 V_{OH}（2.8 伏）。在不同的高低状态之间的输出摆动必须被很好地加以平衡，以最小化信号失真。要求使用失真率控制使辐射噪声和串扰最小化。

☑ 低速驱动器

低速缓冲区把它们的信号驱动到一条非屏蔽的，非双绞的数据线上，并且只能用于在低速设备和低速集线器端口之间发送信号。信号的上升和下降时间必须大于 75 纳秒，以保证 RFI 在 FCC 的 B 类限制内，同时还要少于 300 纳秒，以限制时间延迟和信号失真。使用指定的转换速率，驱动器必须到达指定的静态信号级要求，而且是平滑

地上升和下降，还必须有最小的反射和回环。

低速驱动器使用的信号转换是基于在设备端的 D－线的，D－线上有一个 1.5 千欧的电阻。低速设备的结果空闲状态是一个差分信号“0”，或者说（D+）－（D－）< －200mV 而且 D+<Vse（min）。对于低速设备而言，这也被称为是“J”状态，而“K”状态是用一个差分信号“1”来代表的。数据线长度限制为 3 米。

低速缓冲区用于所有的低速设备上的所有上游端口。在它们的下游端口那边，所有的集线器设备也必须支持低速驱动器的特性（必须使用低速信号的协议和转换速率），因为一个低速设备也可以通过一个低速数据线连接。注意这些集线器端口必须能够支持全速设备的功能。

☑ 全速设备驱动器

全速驱动器信号是通过全速数据线传送的（屏蔽/双绞线）。一条全速数据线的特征阻抗是 90 欧姆，最大长度是 5 米。图 5-7 是一个全速差分驱动器的实例。注意驱动器的相等的输出阻抗必须在 29 和 44 欧姆之间。如图 5-7 所示，可以用电阻串联的办法达到所要求的阻抗。

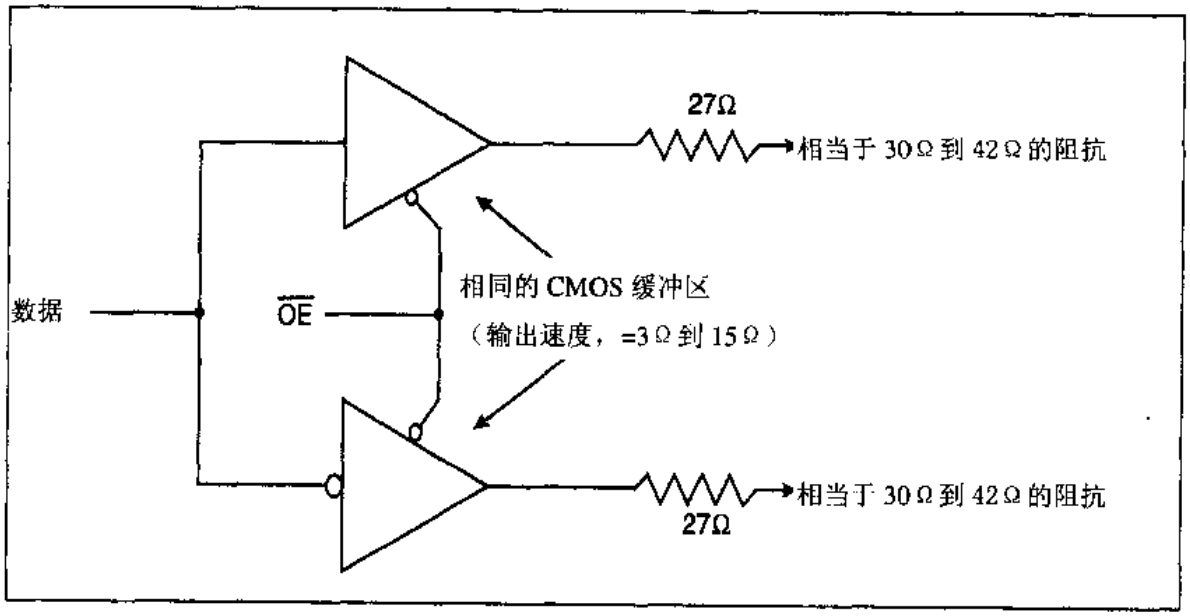


图 5-7　全速 CMOS 差分驱动器的例子

全速驱动器传输必须是严格单调的，必须和控制信号失真以及 RFI 互相匹配，而且上升时间必须在 4ns 到 20ns 之间，可以用一个 50pF 的电容来调节。但是注意，驱动器和数据线阻抗是不匹配的，所以要有来自数据线的接收器端的反射。图 5-8 所示的就是全速信号的波形。注意接收器一端的反射引起了接收器在驱动器完全结束输出摆动之前检测信号的跳变。

全速驱动器所用的信号转换是基于 D+线的，D+线的一端是设备端，那里有一个 1.5 千欧的电阻。所以一个全速设备的闲置状态是差分信号“1”，或者（D+）－（D

－）>200mV 且 D－<Vse（min）。对全速设备而言，这也被称为“J”状态，而 K 状态则代表差分信号“0”，它是和低速信号的转换状态相反的。

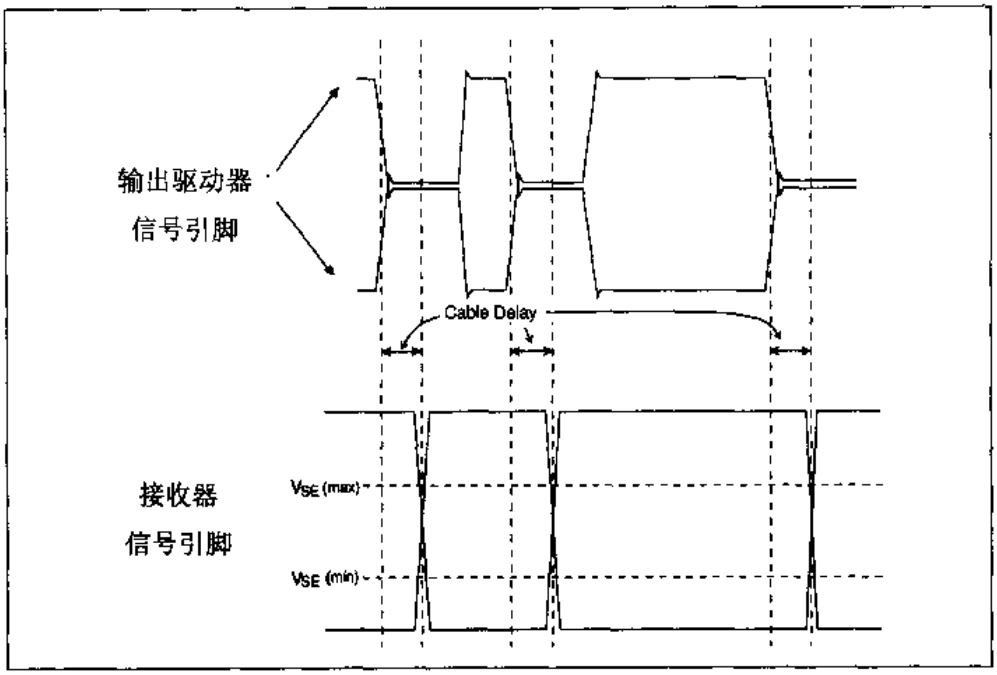


图 5-8　全速驱动器的波形

一个全速驱动器被设计为在全速和低速数据传输率下均可发送数据。要注意的是，全速和低速信号都使用全速信号协议和转换速率。因为它驱动数据经过全速数据线到一个全速接收器。

差分接收器

USB 差分放大器的输入灵敏度是 200mV，要求是两个差分数据输入都在 0.8 伏～2.5 伏的范围内，当然这里的电压都是对地而言的。输入灵敏度减小了由于缓冲区本身带来的噪声。

单端接收器

单端接收器用于检测总线上不同的状态条件。两个单端接收器每一个监视其中一条数据线。正常情况下，D＋和 D－线处于相反的状态，这是因为在其中一条数据线上的上拉电阻和差分信号造成的。在这些状态下，差分放大器将放大两条数据线上的信号差。然而，在某些情况下，两条数据线都被驱动到低电平，以代表一个特殊的状态。

例如：在一个包传输完成之后，两条数据线都被驱动到低电平，持续两个位的时间，表示包传输结束，或者说是 EOP。在这种情况下，因为对接收器的输入来说没有什么不同，所以差分放大器将没有输出。当 D+和 D−线都是低电平时，信号端接收器能够检测到 EOP 状态。

🕮 USB 信号状态的小结

表 5-1 概括了所有的 USB 信号状态。第一栏表示状态，第二栏定义了驱动器端的相关信号状态，第三栏定义了接收器的状态。

表 5-1　　USB 总线状态

总线状态	信号极别	
	发送方驱动器	接收方
差分信号“1”	(D+)−(D-)>200mV 且 D+或 D-<Vse（min）	
差分信号“0”	(D+)−(D-)<-200mV 且 D+或 D-<Vse（min）	
数据 J 状态： 低速 全速	 差分信号“0” 差分信号“1”	
数据 K 状态： 低速 全速	 差分信号“1” 差分信号“0”	
闲置状态： 低速 全速	 差分信号“0”且 D->Vse（max）且 D+<Vse（min） 差分信号“1”且 D->Vse（max）且 D+<Vse（min）	
恢复状态： 低速 全速	 差分信号“1”且 D+>Vse（max）且 D-<Vse（min） 差分信号“0”且 D->Vse（max）且 D+<Vse（min）	
信息包开始状态（SOP）	数据线从闲置状态切换为 K 状态	
信息包结束状态（EOP）	D+和 D-<Vse（min），并且持续两个位的时间，后接一个位的闲置时间。	D+和 D-<Vse（min），并且持续不小于一个位时间，后接一个 J 状态
断开连接（仅适用于上游设备）	NA	D+和 D-<Vse（min），并且持续不小于 2.5 微秒
连接（仅适用于上游设备）	NA	D+或 D-<Vse（max），并且持续不小于 2.5 微秒
复位（仅适用于下游设备）	D+和 D-<Vse，并且持续 10 毫秒	D+和 D-<Vse（min），并且持续不小于 2.5 微秒（必须在 5.5 微秒内作出判断）

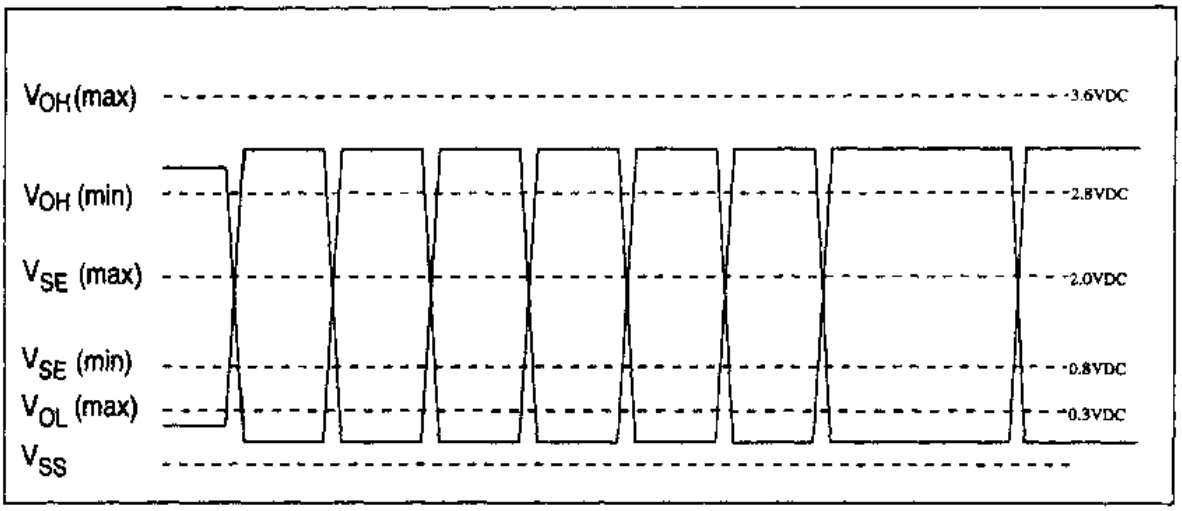

VOH(max)
3.6VDC
VOH (min)
2.8VDC
VSE (max)
2.0VDC
VSE (min)
0.8VDC
VOL (max)
0.3VDC
VSS

图 5-9　USB 信号级别

USB 传输

✎ 上一章

USB 采用 NRZI 编码和差分信号在 USB 数据线上传输信息。前面的章节讨论了 USB 所使用的差分信号和 NRZI 编码技术。

✎ 本　章

USB 支持 4 种传输类型：中断、块（bulk）、同步、和控制。本章将介绍这些传输类型，此外还将介绍用来启动和执行这些传输的过程。

✎ 下一章

在 USB 上的每一个传输广播都是由一些信息包组成的。这些信息包被组合起来定义一个专门的事务处理，这个事务处理属于一个大的传输过程的一部分。在下一章，定义了事务处理的每一个类型，其中包括组成它们的每个专门的信息包。

🕮 概述

一个给定的 USB 设备的每一个端点都有特殊的性质，这些特性规定了如何访问这些端点。传输特性和应用程序的要求有关。USB 的规范说明书已经定义了下面 4 种传输类型，它们每一个都反映了 USB 设备的端点可能要求的传输性质。

中断传输——中断传输典型地被用于在传统 PC 产品中被称为中断驱动设备的器件。由于 USB 不支持硬件中断，所以中断驱动的 USB 设备必须要被周期性地查询，以确定设备是否有数据要传输。举个例子，在传统的 PC 系统中，每当键盘上的一个键被按下时，就会产生一个硬件中断，该中断通知处理器必须执行一个软件中断例行程序来为这个键盘动作服务。但是，USB 键盘是通过周期性的查询来确定是否有键盘数据（例如，由于某个键被按下而产生的数据）需要被传输。该过程的查询速率是很关键的；它

的频率必须足够快，以确保没有数据丢失，但也不能太快以至于造成总线带宽被不必要地减少。

- 块（bluk）传输——块传输用于传输大块的没有周期和传输速率要求的数据。传输到 USB 打印机上的打印作业就是一个使用块传输方式的例子。虽然传输速率对性能表现很重要，但是一个传送速度缓慢的打印作业并不会导致数据丢失或被破坏。
- 同步传输——同步传输要求有一个恒定的发送速率。使用同步传输方式的应用程序必须保证在发送方和接收方之间能够实现传输率的匹配。例如，一个 USB 的麦克风和扬声器应该使用同步传输方式来保证不会因为通过 USB 传输数据而产生频率失真。
- 控制传输——控制传输用来把特定的请求传送给 USB 设备，它经常在设备配置中被使用。它通过一个特定的传输顺序把请求（或命令）传递给设备，有时在这之后就是数据传输，并且以一个结束状态来结束传输。

一个给定的设备可能有很多的端点，每一个端点可以支持一种不同的传输类型。例如，当一个文件管理程序存取一个基于 USB 的 CD-ROM 的时候，数据端点被定义为块传输端点，但是 CD 音频程序执行的存取要求从数据端点执行同步传输。

客户启动传输

在配置过程中，USB 驱动器读出设备描述符以确定一个给定的设备为了要实现它的功能而要求的端点类型。USB 驱动程序确定是否有足够的总线带宽可以满足为所有的端点提供传输的需求。如果带宽够用，USB 驱动程序就建立一个通信管道，和这个通信管道相联系的总线带宽就被保留下来了。驱动程序通知适当的 USB 设备驱动程序供它使用的通信管道已经存在了。图 6-1 表示出了通信管道的概念，这个已经建立起来的通信管道是为客户驱动程序和它的功能单元进行通信而建立的。这个通信管道保持不活动直到客户要求通过其中的一个管道进行传输。还应该指出的是，USB 设备驱动程序知道的仅仅是通信管道和它必须与之通信的端点，它不知道低层 USB 的传输机制的特性。

参考图 6-2。USB 设备驱动程序调用 USB 驱动程序开始一个传输过程。客户传输请求被称为 I/O 请求信息包，或者称为 IRP。IRP 导致在 USB 上执行数据传输过程。USB 驱动程序分配在每个时间片中用于本次传输所需的总线带宽。传输请求可能要用很多时间片才能完成。

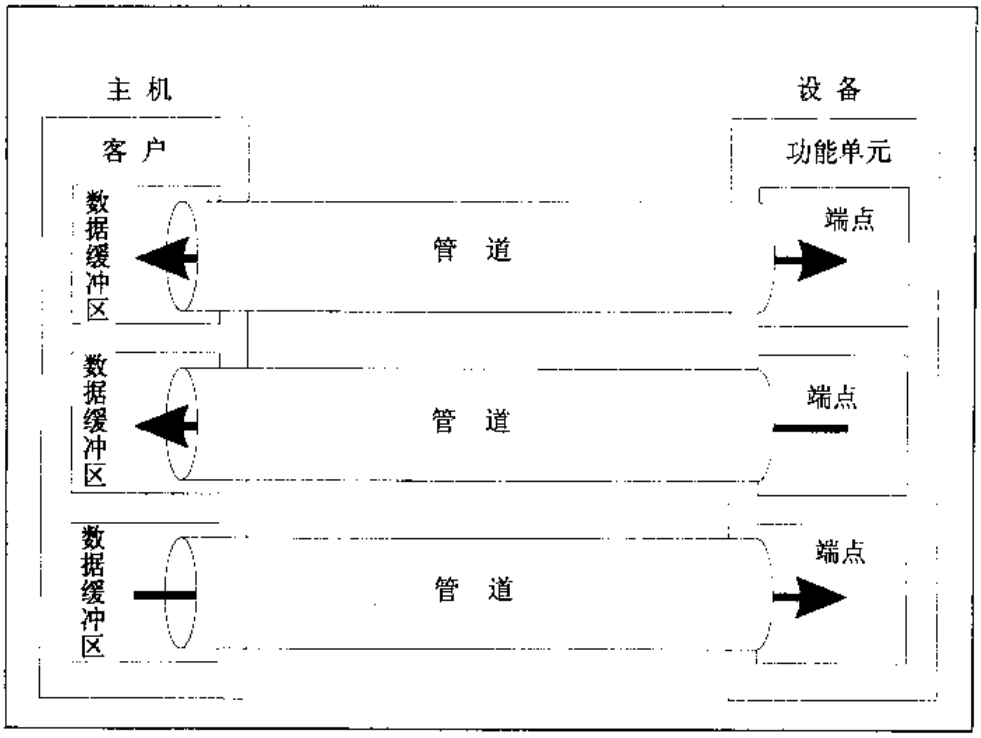


图 6-1　在客户软件的内存缓冲区和设备端点之间的通信管道

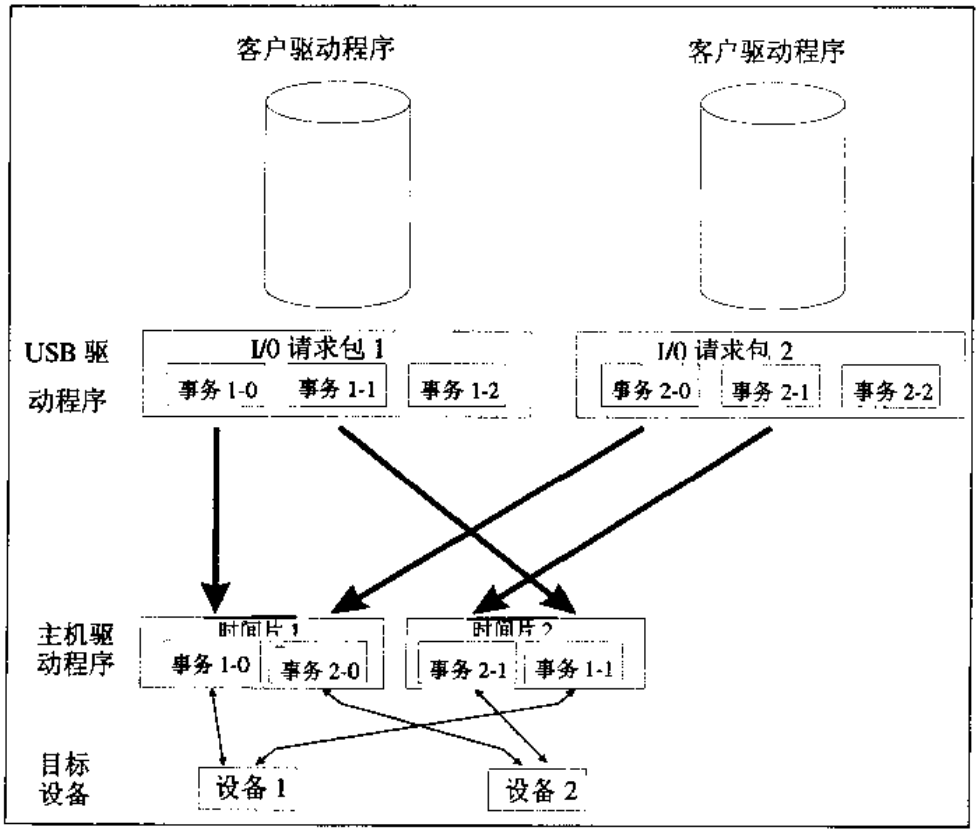


图 6-2　客户请求转换为 USB 事务处理

主控制器的驱动程序对所有的 USB 驱动程序提供给它的 IRP 进行调度。然后主控制器执行已经排定的某个传输过程，就是说在每个时间片中执行多事务处理（从目标 USB 设备读取或者写到目标 USB 设备），直到传输结束。

基于时间片的传输

由于 USB 被很多种设备所共享，所以在 1 毫秒的时间片内可能会混合执行多种类型的 USB 传输。由于中断和同步传输必须按照一定的时间间隔发生，在每一个时间片的执行过程中，它们都有明确的优先权。规范说明书规定了可以用于周期性（中断方式和同步方式）传输的最大 USB 带宽为总的带宽的 90%，而控制传输在每个时间片有 10%的保留带宽，块传输使用剩下的带宽。

传输类型

下面的部分描述了 4 种传输类型的能力和限制。每个设备的端点都有一个相关的描述符，该描述符定义了端点所需要的传输类型。表 6-1 列出了一个端点描述符的一部分，它包含了传输类型字段。阴影部分定义了和给定端点相关的传输类型。

表 6-1　　端点描述符的传输类型定义

字段编号	字　段	字段宽度	取　值	描　述
0	长度	1	数字	描述符的大小，以字节为单位。
1	描述符的类型	1	常数	端点描述符的类型。
2	端点的地址	1	端点	这个设备描述符所描述的 USB 设备上的端点地址。这个地址按如下方式编码： 位 0..3 端点号； 位 4..6 保留，设置为 0； 位 7 传输方向（不考虑控制端点）； 0=OUT 端点； 1=IN 端点。
3	属性	1	位图	一旦设备配置完成以后，这个字段描述了端点的传输类型。 位 0..1 传输类型； 00 控制传输； 01 同步传输； 10 块传输； 11 中断传输； 其他位保留。

同步

要求恒定传输速率的实时应用程序一般都采用同步传输方式。它包括用于音频传输的应用程序（例如：CD 音频、扬声器、数字电话）。同步发送的特色是要求及时地提供数据，这比验证数据是否正确发送要重要得多。正是因为这样，所以在同步传输中，数据的有效性是不能保证的。

图 6-3 是一个在 CD-ROM 和扬声器之间进行同步数据传输的例子。一个同步传输的应用程序必须在应用程序软件和充当接收器或者数据源的 USB 设备之间维持同步。在 CD-ROM 和内存缓冲区之间的数据传输速率以及内存缓冲区和扬声器之间的数据传输速率的结合，加上数据传输率的匹配保证了在 CD-ROM 和扬声器之间的同步数据传输，从而消除了可能的音频失真。

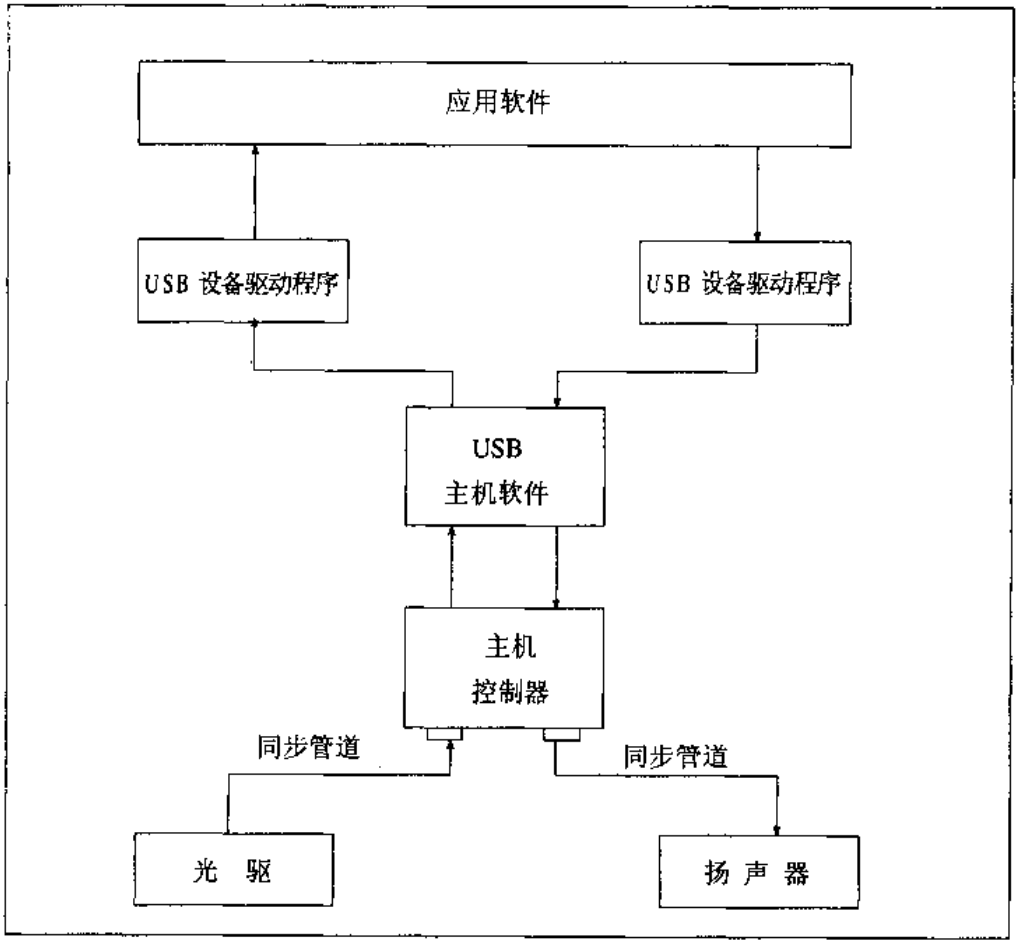


图 6-3 USB CD-ROM 和扬声器之间的同步应用

规范说明书定义了三种方法可以让 USB 同步设备保证它的数据传输率和主机系统之间的同步。它们是：

- 异步——使用异步方式的同步端点以某一个速率发送或接收音频数据。该速率

和一个外部时钟或者独立运行的内部时钟保持一致。该设备不能使传输率和 USB 时钟（基于一毫秒的时间片的开始）保持同步。

- 同步——同步端点有自己的定时方法的设备。该设备也许能够和 USB 提供的 1 毫秒的 SOF 定时保持同步。如果设备不能使它自己的时钟与 USB 的时钟保持同步，就可能要求设备的驱动程序调整 USB SOF 定时，从而允许 USB 和它的时钟同步。USB 允许一个在 USB 上的设备成为一个主设备，主设备能够调整 SOF 直到它和它的采样时钟同步。（参见 253 页的“总线管理”一部分的内容，以获得 SOF 调整的进一步细节信息）。
- 自适应——这些设备的发送和接收音频数据的速率是有明确的范围的，在这个范围内，允许它们和 SOF 定时加在接口上的速率保持同步。

只有全速设备支持同步传输。

☑ 传输方向

同步传输是单向的。一个端点不能双向传输信息。要求既能发送数据又能接收数据的同步设备至少要有两个端点，这两个端点一个用于接收数据，一个用于发送数据。

☑ 服务周期

在 USB 上，同步传输作业是经过安排后按照一定的规则执行于连续的时间片上的，每个时间片为 1 毫秒。这就保证了传输可以维持一个恒定的速率。

☑ 带宽分配

在每个同步传输期间，数据大小是有限制的，在每一个时间片期间为 1023 字节。端点描述符定义了它所支持的最大数据大小。如果一个同步端点要求比可以得到的更多的总线带宽，那么该设备将不会被成功配置，因为同步管道要求有保证的数据发送。

☑ 错误恢复

由于一个恒定的数据传输率对于一个同步应用来说是至关重要的，所以同步传输不支持错误检测和错误恢复。错误恢复包括在错误被检测到时，重新传输数据。进行重新传输可能会导致来自 USB 设备的数据率和来自应用程序或目标设备的数据率之间失去同步。因而，它不允许重新传输。

中断传输方式

中断传输方式总是用于对设备的查询，以确定设备是否有数据需要传输（例如，设备有一个中断请求有待执行）。因此中断传输的方向总是从 USB 设备到主机。如果设备

当时没有数据要发送（例如，没有中断有待执行），那么，设备不返回确认信息，表示现在没有数据要求发送。

☑ 服务周期

采用中断传输是因为设备要求周期性地安排执行任务，所以不会发生超时运行的情况。全速中断传输可以频繁到在每一个时间片都发生传输，也可以慢到每 255 个时间片才发生一次传输。低速传输的最快频率是每 10 毫秒发生一次传输，也可以慢到每 255 毫秒发生一次传输。给定设备所要求的查询间隔在中断端点描述符中说明。表 6-2 是端点描述符的一部分，它定义了类型和中断查询间隔。注意 6 号字段定义了查询间隔，它以一毫秒为单位递增。

表 6-2　　端点描述符的中断查询间隔定义

字段编号	字　段	字段大小	取　值	描　述
3	属性	1	位图	一旦设备用配置值配置完成以后，这个字段描述了端点的属性。 位 0..1 传输类型 00 控制传输 01 同步传输 10 批量传输 11 中断传输 其他位保留。
4	最大信息包的大小	2	数字	当选择这个配置的时候，端点能够发送或接受的最大信息包的大小。 对于同步断点来说，这个值用来保留每一个时间片内数据负载要求的在调度表里面的总线时间。这个管道可以是正在使用中的，实际上它可以使用比保留的总线带宽少的总线带宽。如果有必要的话，设备将会报告它实际上通过正常途径，非 USB 定义的机制所使用的带宽。
6	查询间隔	1	数字	数据传输端点的查询间隔。以微秒为单位。 批量传输和控制传输端点将会忽略这个字段。对于同步传输端点来说，这个字段必须是 1。对于中断端点来说，这个字段的取值范围是 1 到 255。

☑ 总线带宽分配

在每一个时间片期间，对于全速设备来说，中断传输所支持的最大数据大小是 64 字节。每次传输的信息包的大小都必须是端点描述符所规定的最大信息包的大小，最后一个信息包除外。一个比最大信息包小的信息包被认为是最后一个信息包。

像同步端点一样，中断管道要求保证数据在明确的查询周期里被发送。由于缓冲区

溢出而造成在查询周期内对端点的存取失败可能会导致数据丢失。如果可以获得的带宽不能支持中断管道，那么设备将不被配置。

☑ 错误恢复

中断传输支持错误恢复。在执行传输的时候如果检测到错误，那么在下一个服务周期内就将试图进行重新传输。

控制传输

控制传输提供了一种方法来配置 USB 设备，并对它的操作的某些方面进行控制。每个设备都必须实现一个缺省的控制端点（它总是 0 号端点）。控制端点用来配置设备、控制设备状态、以及该设备操作的其他方面。控制端点响应一些 USB 特殊请求，这些特殊请求通过控制传输发送。例如，当在设备上检测到设备时，系统软件必须读取设备描述符，以确定该设备的类型和操作特性。USB 规范说明书还为集线器和其他的一些类型的 USB 设备定义了一些设备请求。这些请求可以是为所有的 USB 设备定义的标准请求，也可以是为特殊设备类定义的设备请求，还可以是厂商自定义的请求，厂商自定义的请求只有对应的设备驱动程序和设备知道。第 15 章“设备请求”中有一个 USB 设备支持的标准命令的详细列表。

控制传输至少由两个阶段组成，也有可能是三个阶段：

- 建立阶段——控制传输总是从建立阶段开始，本阶段把信息传送给目标设备，定义对 USB 设备的请求类型（例如，读设备描述符）；
- 数据阶段——这个阶段仅仅是为要求数据传输的请求定义的。例如，在数据阶段，读描述符的请求把描述符的内容发送给系统。一些请求在建立阶段之外不要求数据传输；
- 状态阶段——这个阶段总是用来报告被请求的操作的结果。

☑ 总线带宽分配

控制传输从建立阶段开始，它包含 8 字节的信息包。这个 8 字节的信息包定义了在控制传输的数据阶段所传输的数据的数量。在数据阶段，信息包限制的最大数据载荷为 64 字节。控制传输能够保证得到 10%的总线分配。如果有额外的可利用的带宽，那么控制传输可以分配到多于 10%的带宽。

☑ 错误恢复

控制传输参与错误检测和恢复机制，努力提供一种最大限度的恢复。恢复机制失败的后果是十分严重的。

块（bluk）传输

块（bluk）传输用于对数据传输率没有特殊要求的设备。块（bluk）传输设备的一个典型的设备是打印机。因为对打印机来说，用较慢的数据传输率不会产生什么问题。除非某个着急的用户急于等待打印结果出来。

☑ 总线带宽分配

由于块（bluk）传输对于数据发送的速率没有特殊的要求，所以它们在每一个时间片期间都被分配为最低的优先权。分配给同步传输和中断传输的带宽总共为 90%，分配给控制传输的带宽为 10%，在一个完全分配的时间片内没有给块传输留下任何带宽。块（bluk）传输的带宽是基于所有其他传输的带宽已经被安排好以后剩下的带宽来安排的。如果没有带宽可以分配给块（bluk）传输，那么块（bluk）传输就被推迟，直到总线的负载减少。然而，在没有其他传输类型的时候，总线带宽的很大一部分将分配给块（bluk）传输，从而达到一个很高的执行效率。

块（bluk）传输的最大信息包的大小限制为 8、16、32 或 64 字节。不允许出现别的大小的信息包。当块（bluk）传输发生时，所有的信息包的大小必须是最大信息包容量字段中指出的最大容量，传输的最后一个信息包除外。如果传输所希望大小的信息包失败，那么将会导致错误和传输终止。

块（bluk）传输端点总能用软件加以配置，因为它对数据发送的速率没有特殊的要求。如果没有总线带宽可以分配，那么块（bluk）传输将会被推迟，直到有可分配的总线带宽为止。

☑ 错误恢复

块（bluk）传输支持错误检测和恢复，在块（bluk）传输中，数据的完整性远比数据的传输率重要。块（bluk）传输使用所有可利用的错误检测和恢复的形式。

USB 事务处理

✎ 上一章

上一章介绍和描述了四种 USB 支持的传输类型：中断传输，块传输，控制传输和同步传输。以及这四种传输类型的适用场合、能力和限制。

✎ 本　章

在 USB 上的每次传输广播都由一些信息包组成。这些信息包组合起来定义某个事务处理，而某个事务处理又是作为一个大的传输的一部分执行的。每一种事务处理类型在本章都加以了定义。此外，本章还定义了组成它们的每一个信息包。

✎ 下一章

USB 支持主机系统和目标 USB 设备之间的数据无损传输。下一章详细讲述了错误检测和处理机制，USB 采用这些机制来保证数据发送的正确性。

🕮 概述

前面的章节讨论了用于和设备内的端点进行通信的不同的传输类型。传输在 USB 上执行，期间要进行一个或多个事务处理，每个事务处理由一系列的信息包组成。图 7-1 就是在执行一次传输的过程中，不同的层之间的关系。传输过程从 USB 设备驱动程序要求执行一次传输开始，到通过 USB 数据线传送结果信息包结束。这个传输可以来自设备，也可以到达设备。本章讨论专门的事务处理，它由主机进行初始化，用于把数据发送到 USB 设备和从 USB 设备把数据发送出去。每个事务处理由一个或多个信息包组成，它们通过 USB 进行发送。

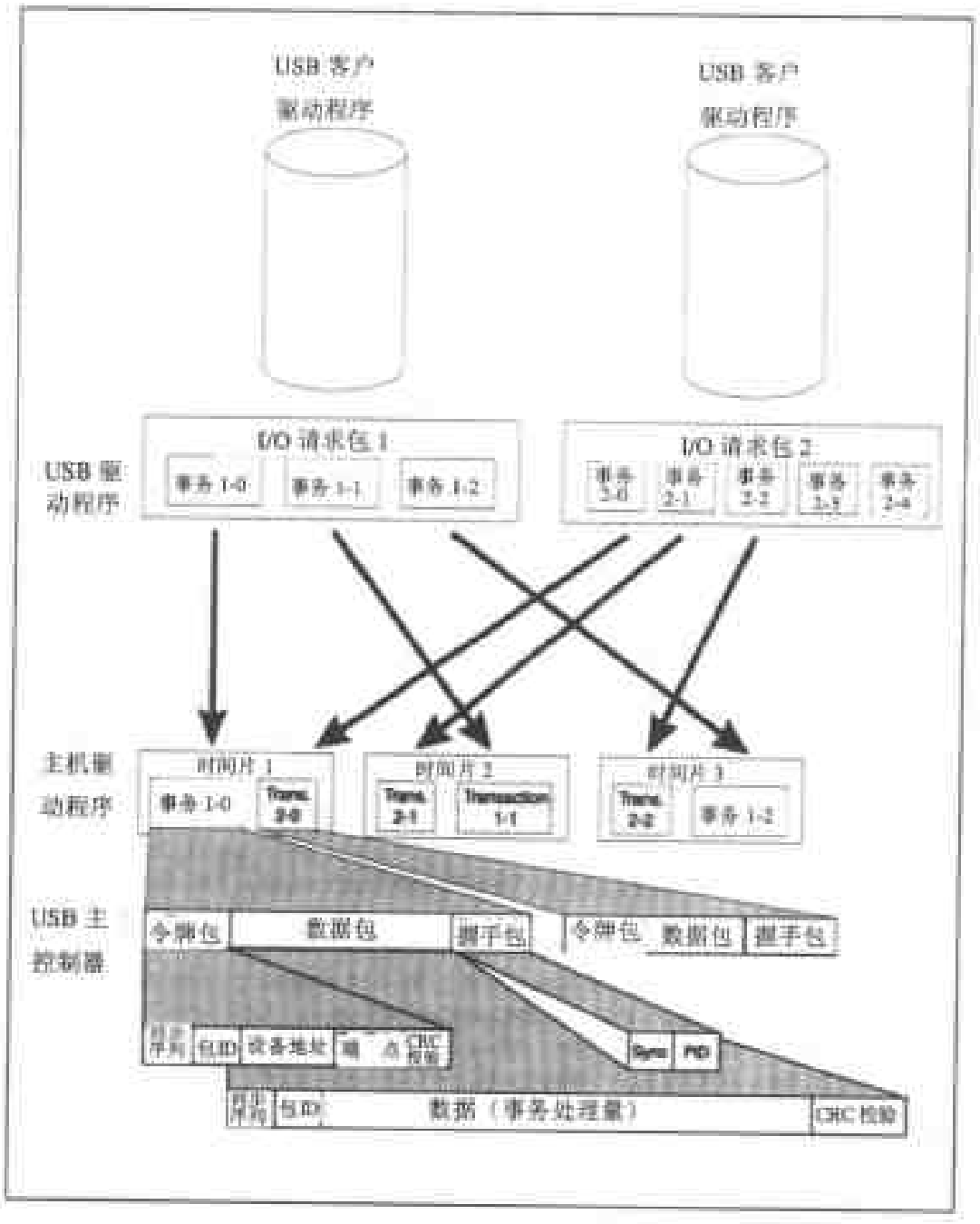


图 7-1　USB 传输所涉及到的层次

信息包——USB 事务处理的基本构成单位

事务处理一般由三个阶段组成，或者说由信息包组成，如图 7-2 所示。但是一个事务处理可以由一个、两个、或三个阶段组成，这依赖于传输的类型。

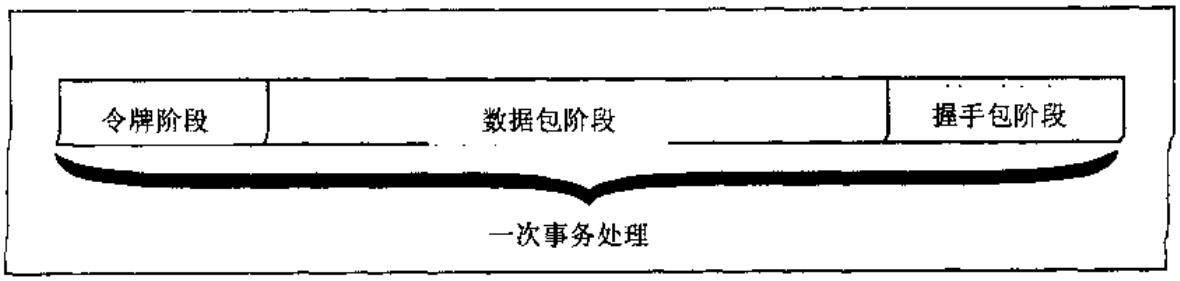


图 7-2　很多 USB 传输由 3 个阶段组成

- 令牌包阶段——每个事务处理都从一个令牌阶段开始，它定义了事务处理的类型。当事务处理的目标是一个指定的 USB 设备时，这个阶段还包括设备地址。某些令牌包是独立的，因而没有其他附加的信息包跟随，而另一些令牌包总是跟随有一个到两个附加的信息包；
- 数据包阶段——很多事务处理类型都包括一个数据阶段，该阶段负责运送和传输相关的数据。数据阶段在一个事务处理中可以传输的最大数据包大小是 1023 字节。但是，允许的最大的数据包取决于所执行传输的类型；
- 握手阶段——所有的 USB 传输的实现都要保证能够进行数据发送，除了同步传输外，还有握手阶段。握手阶段对数据发送方提供了一个反馈信号，通知发送方数据是否已经被正确地接收到了。如果在一次事务处理中遇到了错误，发送方就会进行重新传输（参见第八章，该章对错误处理作了详细的解释）。

信息包是用来执行所有的 USB 事务处理的机制。图 7-3 所示的就是 USB 信息包的基本格式。每个前行数据包是一个同步序列，还允许 USB 设备和传来的数据位的传输速率同步，这些数据位就是信息包中的数据。信息包的类型由一个位组合加以定义，称为信息包的 ID。在 ID 之后是关于这个信息包的具体内容（例如：地址或者数据）。它根据包的类型的不同而不同，每个包以一个循环冗余校验码（CRC）序列结束，它用来验证包中的具体信息是否被正确地发送。每个包的结束都由一个包结尾状态来标识。关于每一个包的类型的细节在下面部分加以描述。

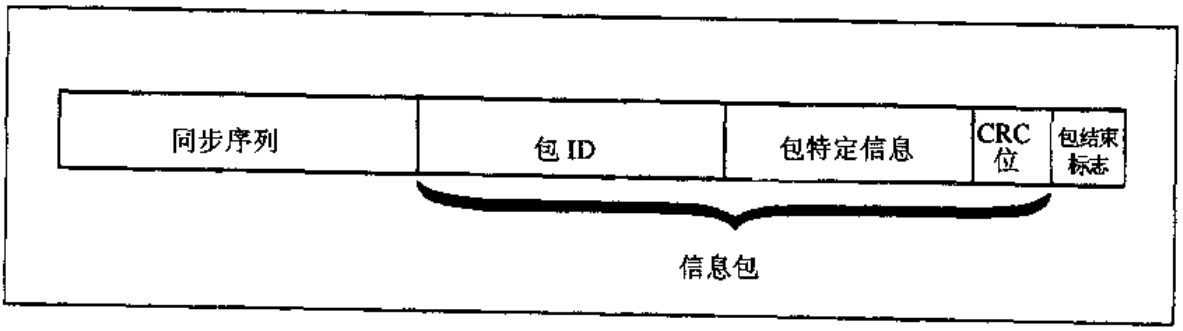


图 7-3　数据包的格式

同步序列

图 7-4 所示的就是同步序列，同步序列由 8 个位组成，由 7 个连续的逻辑值 0 开始，结束为逻辑值 1。由于 0 用差分数据线上的电平跳变来编码，所以这 7 个 0 在每一位开始的时候都建立一个电平跳变。这就提供了一个时钟，用来同步。同步性序列还通知 USB 接收器马上要有一个信息包被发送，这个信息包紧跟在这 8 位同步序列之后。

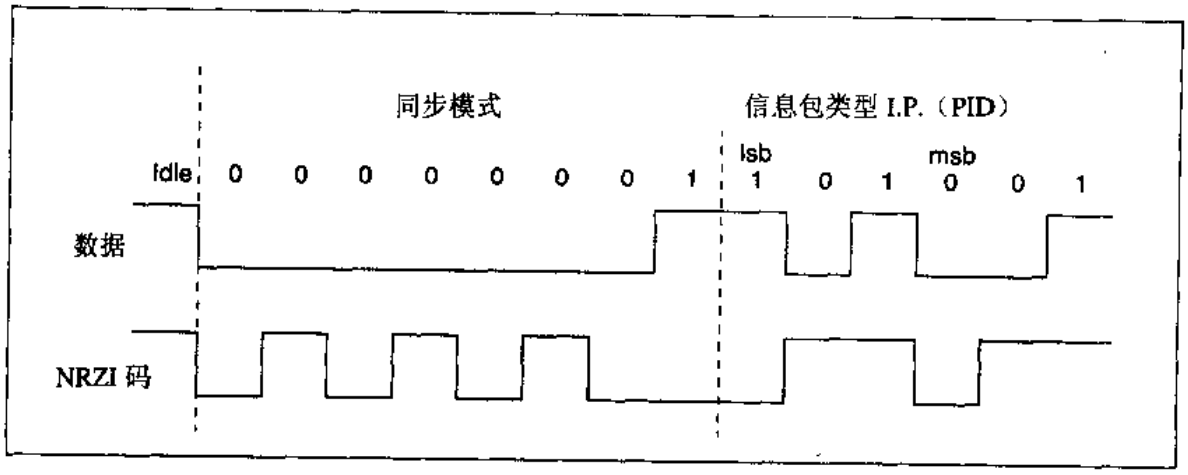


图 7-4 同步序列

信息包可以在 USB 上广播，或者是全速（12Mbps）或者是低速（1.5Mbps），并且该速度决定了包中数据将以什么速度进行传输。USB 接收器必须检测到包中的每一位的逻辑状态值，这是通过对数据线进行采样获得的，每个采样都要保证采样点的正确。同步序列以正在使用的传输率进行发送，允许接收器和任何一个传送过来的数据率同步。

但是要注意，低速 USB 设备不能以全速方式通信。所以，低速数据线仅仅能够传输低速的事务处理。USB 集线器在低速事务处理被执行之前，禁止使用低速端口。在开始低速传输之前，必须给集线器一段时间用于激活它们的低速端口。这是通过一个特殊的信息包来完成的。这个包被称为前导包，它专门用来通知集线器激活它们的低速端口。前导包必须直接位于每一个低速事务处理之前。一旦低速包开始传输，总线操作就返回全速操作状态。

信息包的标识符

信息包的标识符定义了信息包的目标和内容，它分为以下 4 类：

- 令牌包——令牌包在 USB 事务处理的开始被发送，用于定义传输的类型（例如：传输到或者传输自 USB 设备）。

- 数据包——该包在事务处理中跟随在令牌包之后，该事务处理需要把数据传输到 USB 设备或者需要从 USB 设备把数据传输到别处。
- 握手包——握手包一般从接收方返回到发送方，它向发送方提供了一个信息反馈，告诉发送方事务处理成功或失败。在某些情况下，要求把数据发送到系统的 USB 设备可以发送一个握手包，以此指出当前没有数据可发送。
- 专用包——目前，仅有一个专用包的定义，就是前导包，它用于激活低速端口。

信息包的格式和长度取决于它的类型，令牌包全长 4 字节，但是它包含了不同的信息，这些信息用于描述它所定义的事务处理的某些方面。数据包则随着传输类型的不同在长度上也不同，传输类型则和事务处理相关。例如，块传输的数据包限制为每次事务处理 64 字节，而同步传输的数据包则是每次事务处理 1024 字节。

参考图 7-5。信息包的 ID 由四位标识符字段组成，后接四位检测字段，检测字段包含了包 ID 的反码，在这 8 位包 ID 之后是包的具体信息。

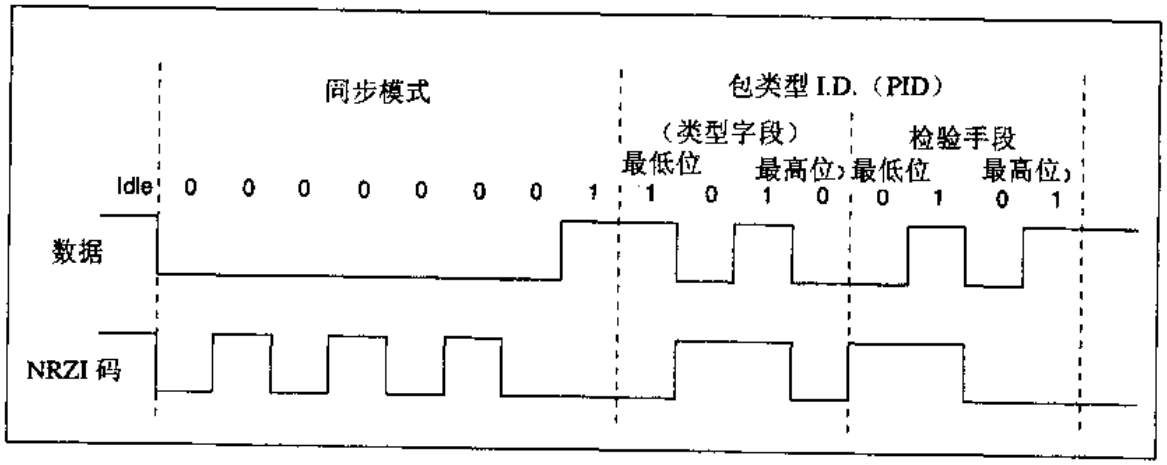


图 7-5　数据包标识符的格式

信息包的特定信息

每个信息包包含的信息和它所执行的工作有关，这些信息可以组成一个 USB 设备地址，一个时间片的序列，从 USB 设备发出或者发送到 USB 设备的数据，等等。这些信息对于一个给定事务处理能否成功完成是很关键的，并且在每一个包的结束处有 CRC 校验。

循环冗余校验（CRC）

发送一个给定包的 USB 代理要计算一个 5 位或 16 位的 CRC，这取决于它的类型。

数据包使用 16 位的 CRC，而别的信息包则使用 5 位的 CRC。CRC 的产生和检验仅仅用于信息包的具体数据，包的 ID 有自己的校验位，这部分内容读者可以参见“CRC 错误”部分。

信息包的结束（EOP）

每个信息包的结束都由发送方发出一个信号来表示，具体方法是把两条差分数据线上的电压降低，并且这个低电压将持续两个位的传输时间，而且后接一个位的空闲时间。当接收方检测到两条差分数据线都是低电平，并且持续了一个位的时间时，接收方就认为所接收信息包已经结束了。图 7-6 所示的就是一个在 USB 上的 EOP。

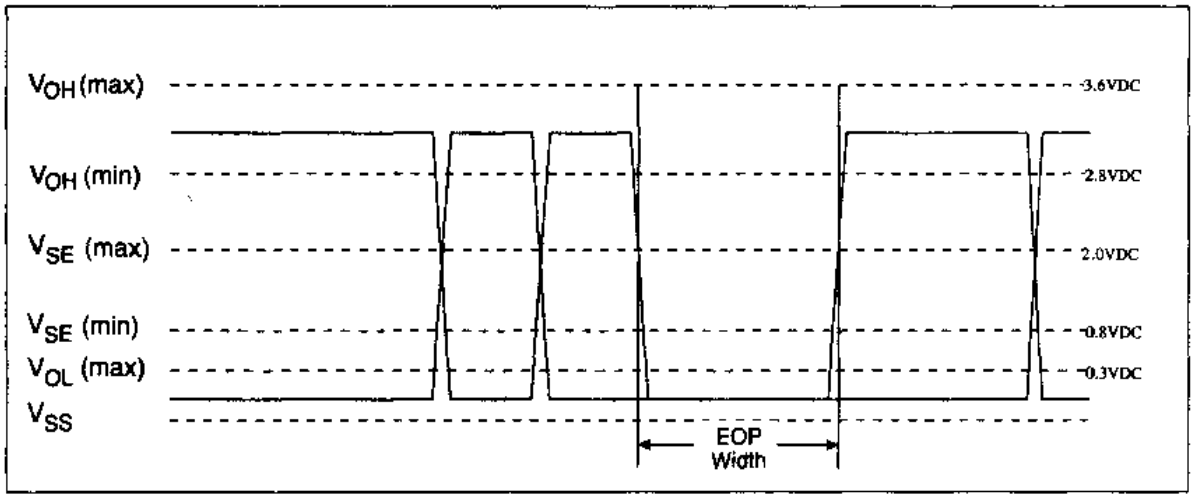


图 7-6 数据包信号的结束

令牌包

令牌包定义了在 USB 上进行广播事务处理的类型。所有的事务处理都从一个令牌包开始。令牌包的四种类型由 USB 的规范说明书定义：

- SOF（时间片开始）——指出下一个 1 毫秒时间片的开始；
- IN——表示一个 USB 事务处理，这个事务处理把数据从目标 USB 设备发送到系统；
- OUT——表示一个 USB 事务处理，这个事务处理把数据从系统发送到目标 USB 设备；
- SETUP——表示一次控制传输的开始。SETUP 是控制器传输的第一阶段，并用来把一个请求从系统发往目标 USB 设备。

表 7-1 列出了所支持的令牌类型，并且描述了嵌在信息包中的具体信息和指定的

行为。每一个令牌包由它们的 ID 进行标识，如第 3 栏所示。

表 7-1　　USB 标识

PID 类型	PID 名称	PID[3:0]	令牌包的描述
令牌	SOF	0101b	包含时间片的开始（SOF）标志和时间片编号。SOF 令牌由同步端点使用，用来同步它的传输。
令牌	SETUP	1101b	包含 USB 的设备地址和端点号。传输从主机到设备，用于建立一个控制端点（例如配置）
令牌	OUT	0001b	包含 USB 设备地址和端点号，传输方向是从主机到设备。
令牌	IN	1001b	包含 USB 设备地址和端点号。传输方向是从设备到主机。

SOF 包

SOF 包提供了一种方法用于目标设备识别一个时间片的开始。举个例子，同步应用程序可以用 SOF 在一个指定的 1 毫秒时间片的开头触发和同步传输的开始。SOF 包对于所有的全速传输设备广播（包括集线器）。广播发生在每个时间片的开始。SOF 包从来不发送给低速设备，因为它们不能支持同步传输。

SOF 包内含有一个 11 位的时间片编号，如图 7-7 所示，时间片的编号由接收器加以验证，验证时要用到 5 位 CRC。注意，SOF 包定义了一个事务处理，它仅仅由令牌包组成。没有数据包或者握手包和 SOF 包相联系。因而，发送的有效性不能得到保证。所以 USB 目标设备必须执行检查，并且采取下面的适当的行动。

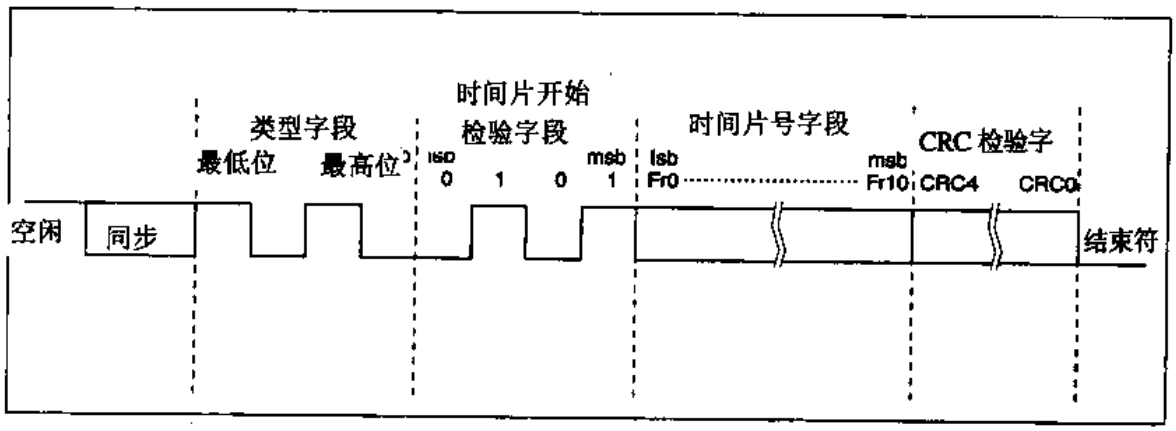


图 7-7　SOF 包的格式

- PID 检查错误——忽略这个包；
- 时间片 CRC 错误——忽略时间片编号。

IN 包

当软件希望从一个指定的设备读取信息时，就用到一个 IN 令牌。IN 包通知目标 USB 设备数据正在被系统请求。IN 事务处理可能会被用于各种 USB 传输类型中，包括：

- 中断传输；
- 块传输；
- 控制传输的数据阶段；
- 同步传输。

如图 7-8 所示，一个 IN 令牌包由 ID 类型字段、ID 校验字段、USB 设备和端点地址以及 5 位的 CRC 组成。一个 IN 事务处理从一个 IN 包广播开始，它由根集线器发出，后接一个从目标 USB 设备返回的数据包。在某些情况下，还包括一个从根集线器发回到目标设备的握手包，用来确认数据接收。注意在同步传输中使用的 IN 事务处理不包括握手包。

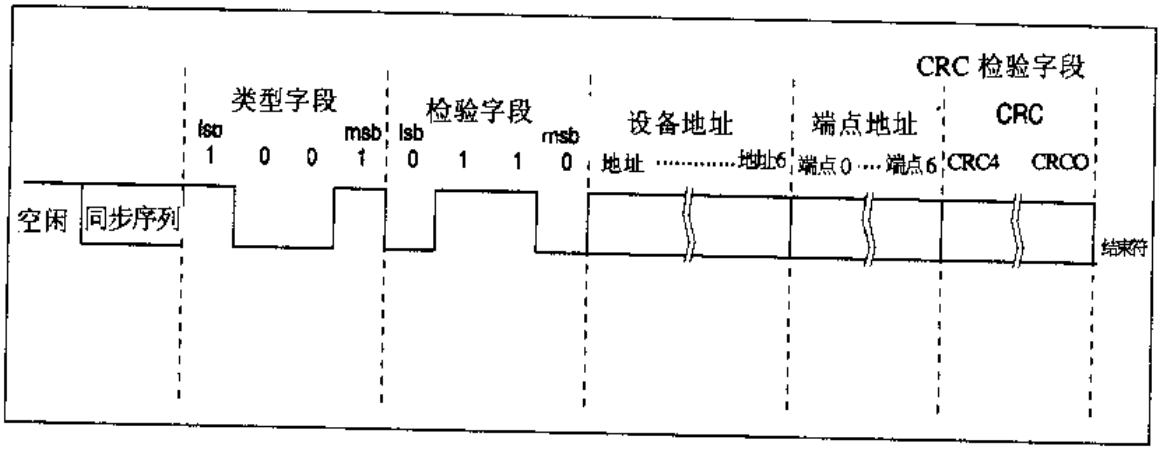


图 7-8 IN 令牌包的格式

数据的数量可以在一个 IN 传输的过程中被传输，这依赖于传输的类型，就像我们在第 6 章中讨论过的一样。

OUT 包

系统软件指定一个 OUT 事务处理，当数据被发送到一个目标 USB 设备时，有三种类型的传输采用 OUT 事务处理。

- 块传输；
- 控制传输的数据阶段；
- 同步传输。

图 7-9 所示的就是 OUT 令牌包的内容。一个 OUT 包由包 ID、类型检查字段、USB 目标设备和端点 ID 以及 5 位 CRC 组成。OUT 令牌包后跟随一个数据包和一个握手包（仅用于块传输）。数据的大小由采用 OUT 事务处理的传输来管理。

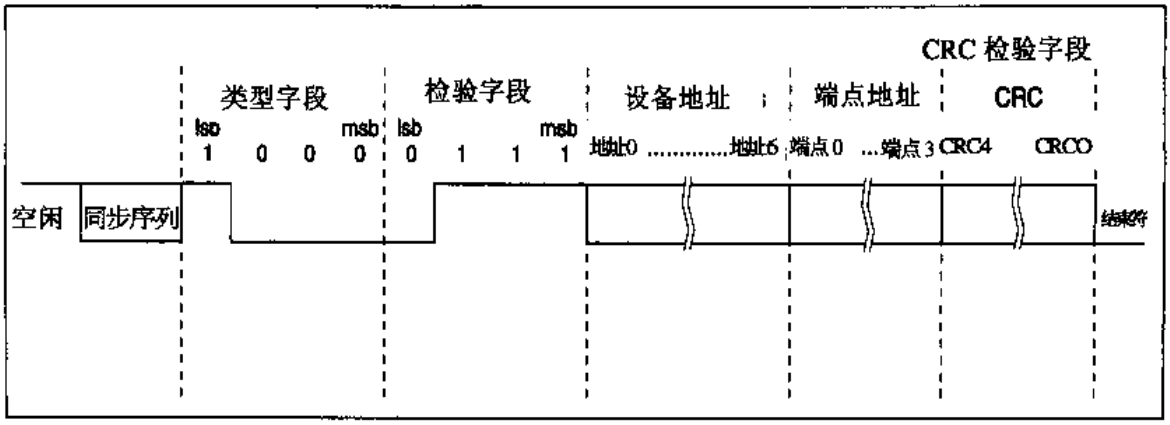


图 7-9　OUT 令牌包的格式

SETUP 包

SETUP 包仅仅在控制传输的建立阶段使用。SETUP 事务处理启动一个控制传输，这个阶段被定义为建立阶段。一个 SETUP 事务处理在格式上和一个 OUT 事务处理是类似的：SETUP 包后跟随着一个数据包，和一个确认包。SETUP 包把一个将要由目标设备执行的请求传送。大量的请求被集线器和其他 USB 设备所支持，它们在第 13 章和第 15 章被定义。根据请求，SETUP 事务处理可以跟随一个或多个 IN 或 OUT 事务处理（数据阶段）。或者可以仅仅伴随一个状态阶段，状态阶段由一个最后的数据包组成，它从端点传向主机系统。

数据包——Data0 和 Data1

数据包是和给定的事务处理相关的。数据包的传输方向由事务处理类型确定，数据包既可以传向 USB 也可以从 USB 设备传出，它们的方向列于表 7-2。

表 7-2　　数据包的传输方向

事务处理类型	数据包的方向
IN 事务处理	来自 USB 设备。
OUT 事务处理	传向 USB 设备。
SETUP 事务处理	传向 USB 设备。

目前已经定义的两种类型的数据包（Data0 和 Data1）用于支持在发送方和接收方之间的长传输同步。例如，如果一个耗时很长的传输从主机方被发送到一个打印机，传输将会被分割成更小的数据块加以执行，但是这将使用大量的时间片。为了验证在一次传输过程中，数据事务处理并没有丢失，可以采用一种称为数据触发的技术。若想了解更为详细的信息，可以参见“数据触发错误”部分。

Data0 数据包的格式如图 7-11 所示，Data1 数据包的格式见图 7-12。

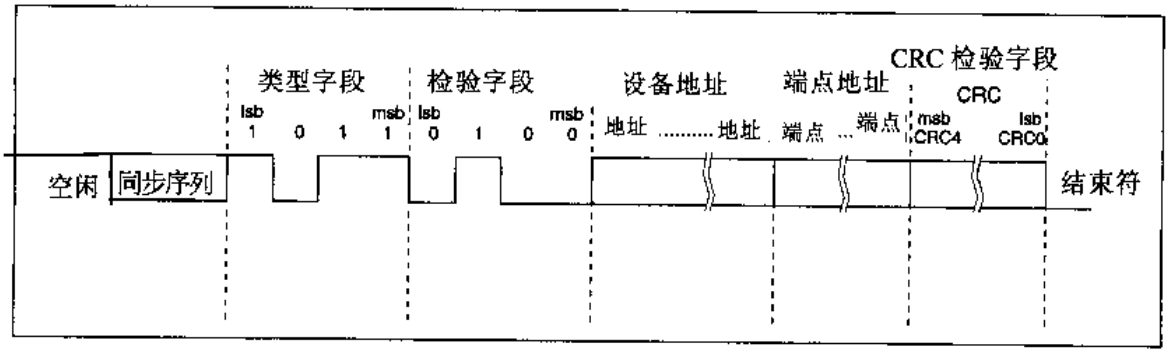


图 7-10　SETUP 令牌包的格式

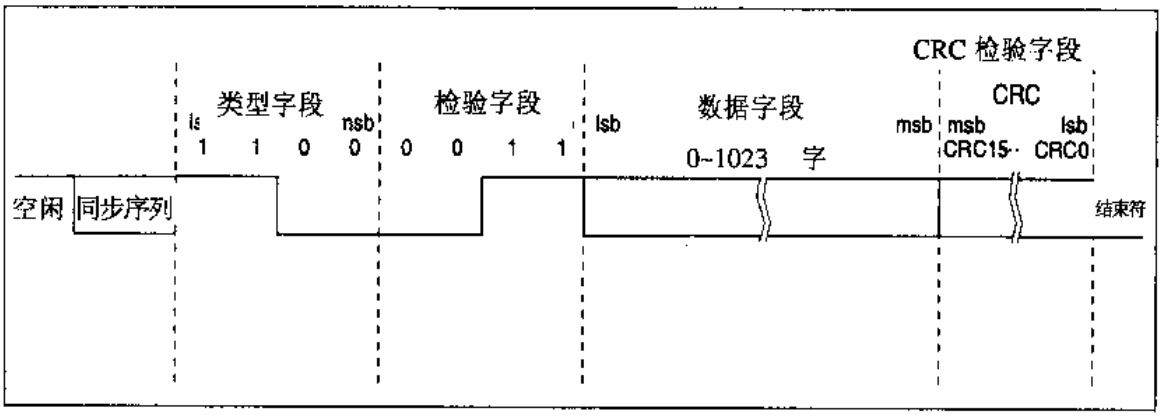


图 7-11　Data 包的格式

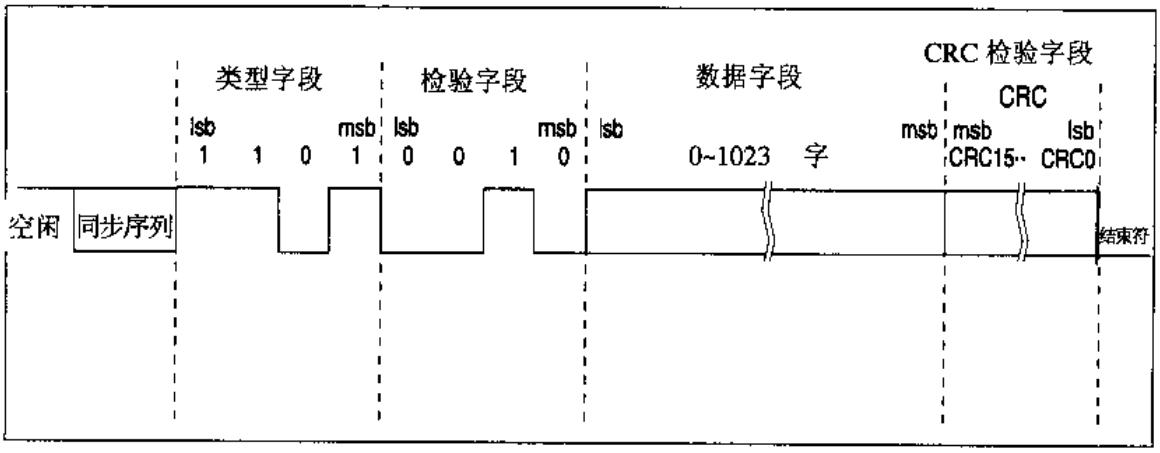


图 7-12　Data 包的格式

握手包

USB 设备使用握手包来报告一个给定的事务处理的状态。数据的接收器（目标设备和根集线器）负责给发送方发回一个数据包。通过不同的握手包，可以报告 3 种可能的结果：

- 确认包（ACK）——确认数据包被无错误地接收了；
- 非确认包（NACK）——报告主机目标设备暂时不能接收返回的数据。在中断事务处理中，NAK 表示当前没有数据可以返回到主机（例如，当前没有有待执行的中断请求）；
- 停止包（STALL）——目标设备用来报告它不能完成传输，并且要求软件进行干预，使设备从停止状态恢复。

握手包的格式如图 7-13 所示。

前导包

集线器通过禁止低速端口的方法阻止高速事务处理在低速数据线上进行处理。在广播一个低速信息包之前，必须广播一个前导包，目的是通知所有的集线器在这个前导包后面跟随一个低速的事务处理。集线器必须对这个前导包作出响应，响应的办法是激活集线器的低速端口。但是其他的设备将会忽略这一前导包。主机保证在前导包之后立即进行信息包的传输，这些信息包以低速发送。所有的低速信息包在所有的数据线上被广播，所以这就要求高速集线器端口往下游方向既能发送低速信息包，又能发送全速信息包。上游的连接不受包是否由全速或低速发送的影响。就是说，所设计的集线器接收器和上游发送器既可以处理低速信息包，又可以处理全速信息包。

图 7-14 所示的就是一个前导包的格式，前导包由一个同步序列和一个包的 ID 组成的，该包以全速传输。在 PID 之后，主机必须延迟一会儿再开始低速信息包的传输，这种延迟是通过插入四个全速位的传输时间实现的。这个延迟给了集线器一段时间用来激活它的低速端口。在这段延迟过程中，集线器就可以使它的端口（低速或全速）到闲置状态。注意前导包和其他的信息包不同，因为它不是以 EOP 结束的。

一旦前导包和 4 位时间延迟完成之后，主机就在 USB 上发送一个低速的令牌包，用它来指定所执行事务处理的类型。当它检测到低速 EOP 时，集线器重新禁止它的低速端口。

在 IN 事务处理中，目标设备以低速向主控制器返回数据。向上返回到集线器接收器端口的数据可以是低速、也可以是全速的事务处理。在 OUT 事务处理中，传输到 USB 设备的数据包也必须以低速传输。在这个数据包之前，还要求发送一个前导包。

低速设备可以支持控制和中断传输，但是这只适用于在一次事务处理中数据的大

小限制为 8 字节长的情况。

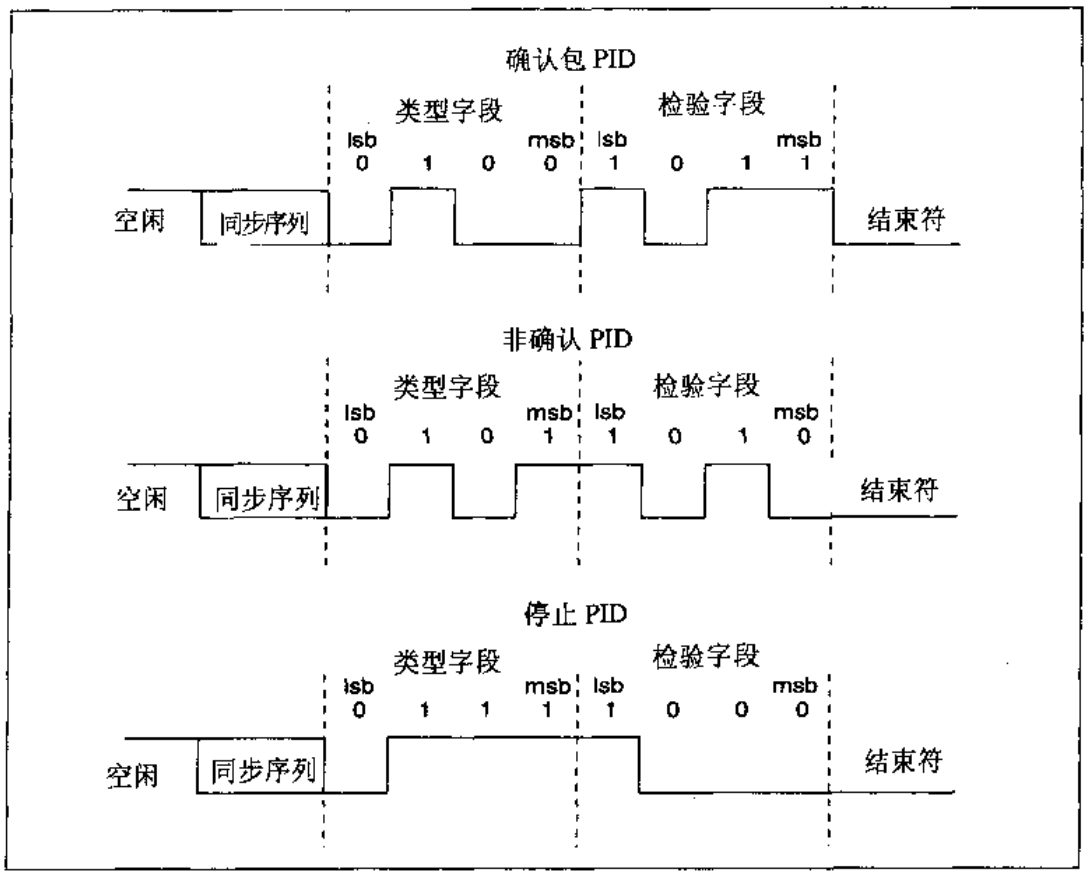


图 7-13　握手包的格式

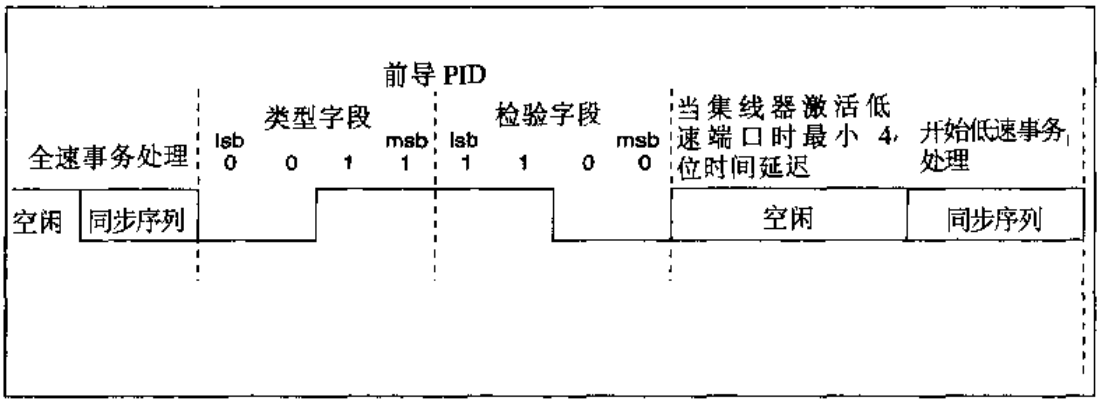


图 7-14　前导包的格式

事务处理

下面的部分描述了每一种事务处理的类型，并指出了每种事务处理可能采取的形式。

IN 事务处理

一个典型的 IN 事务处理由令牌阶段、数据阶段和握手阶段组成。然而在某些情况下，一个 IN 事务处理可以不由所有这 3 个阶段组成。IN 事务处理在所有的四种类型传输的执行过程中都会采用，当执行 IN 事务处理时可能有以下几种情况发生：

- 数据被无错误地接收；
- 数据接收的时候有错误；
- 目标设备暂时不能返回数据；
- 错误状态被清除之前，目标设备不能返回数据；
- 发生了一次同步传输，这样，目标设备会返回数据，但是数据阶段之后没有握手动作。

☑ IN 事务处理无错误的情况

在图 7-15 中，在执行一次中断传输、块传输、同步传输或控制传输的时候，都可能发生 IN 事务处理。在本例中，目标设备返回到根集线器的数据被无错误地接收，而且把一个 ACK 握手包返回到目标设备。请注意，数据包被定义为 data0 包，data0 数据包用于单个数据阶段的传输和中断传输，并且在一个要求多个 IN 事务处理的第一次事务处理期间总是使用 data0 数据包（例如：来自 USB 设备的块传输和控制传输），在多事务处理传输中，连续的事务处理将在 data0 和 data1 之间进行交替。

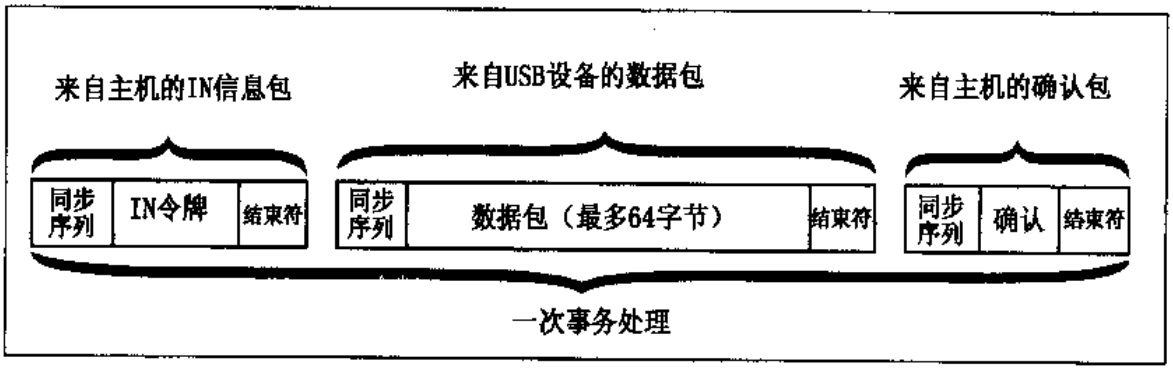


图 7-15　无错误的 IN 事务处理

☑ IN 事务处理出错的情况

在图 7-16 中，IN 事务处理由两个阶段组成：IN 令牌阶段和数据阶段。因为当主机收到数据包时会检查到错误，所以不会有握手包从主控制器返回到目标设备。而且由于主机没有返回确认信息，所以目标设备将会检测到一个超时状态。主机负责在以后的某个时间重新进行 IN 事务处理。

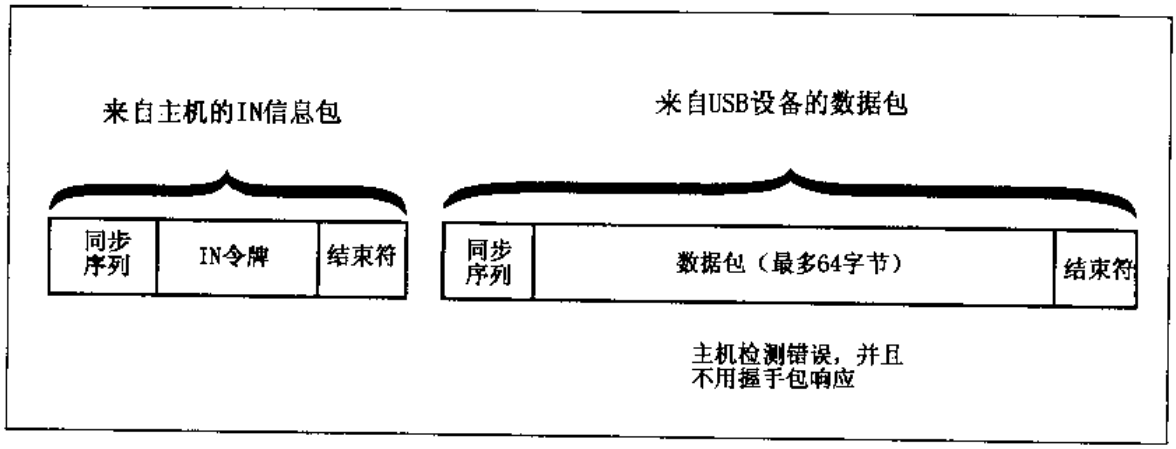


图 7-16　数据阶段出现错误的 IN 事务处理

☑ 没有有待执行的中断或目标设备忙的 IN 事务处理

在图 7-17 所示的状态下，目标设备暂时不能返回数据，因为设备在事务处理的数据阶段不能返回数据，所以它在数据阶段返回一个 NAK。以此通知集线器和主控制器设备暂时不能返回数据。如果在设备中当前没有中断有待执行，那么这种形式的 IN 事务处理就可能在中断传输过程中发生。此外如果在设备中存在一个忙状态，那么这种 IN 事务处理也可以发生在块传输和控制传输过程中，从而阻止它返回数据。

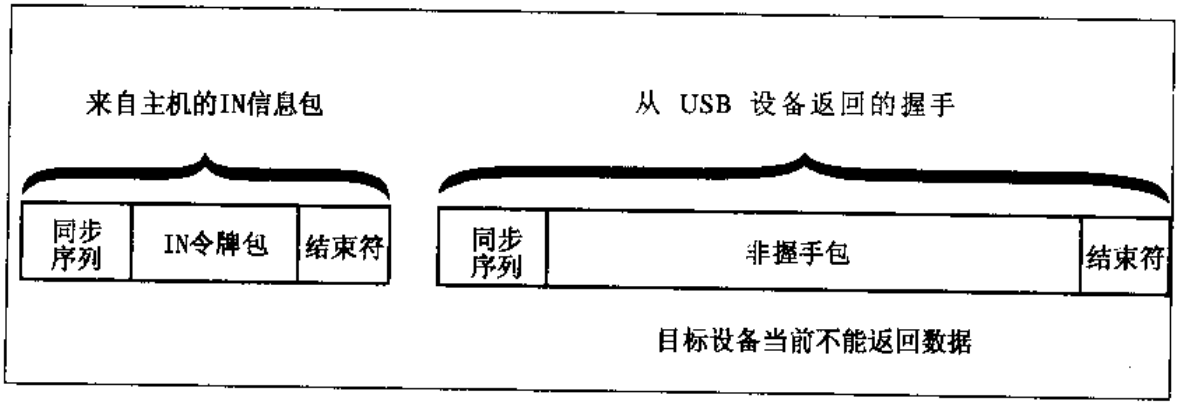


图 7-17　目标暂时不能返回数据的 IN 事务处理

☑ 目标设备被停止的 IN 事务处理

图 7-18 所示的是一个状态，在该状态下，目标设备已经进入一个错误状态，该状态阻止它返回数据。目标设备返回一个 STALL 握手包给根集线器，通知系统软件必须清除这个错误状态才能返回数据。注意这个错误可能是正常操作的一部分，而不是一个严重的 USB 设备问题。例如，一个扫描仪在扫描过程中意外断电，那么扫描仪将不能通过 USB 接口返回数据。但是一旦电源恢复供电，传输就能继续进行了。

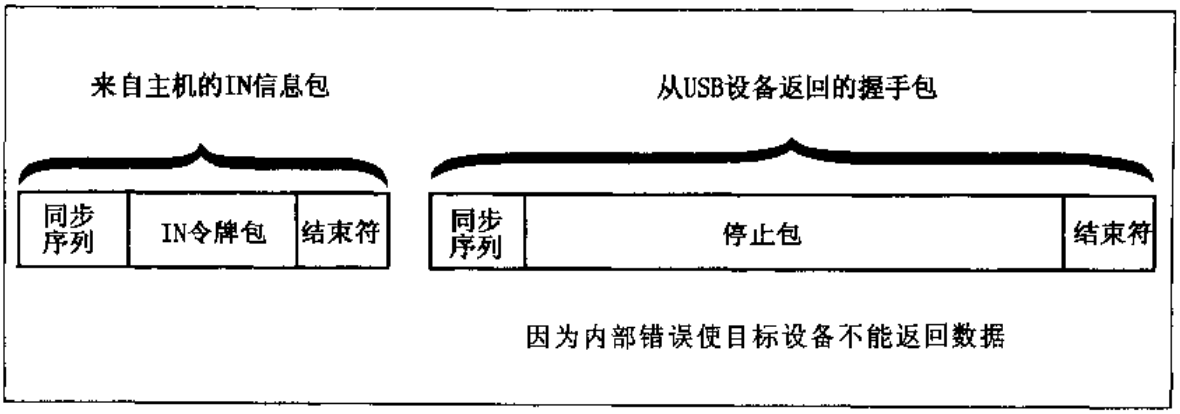


图 7-18　目标器被停止的 IN 事务处理

☑ 在同步传输过程中的 IN 事务处理

在图 7-19 中，IN 事务处理来自于同步传输。主控制器和目标设备知道事务处理是一次同步传输的一部分。目标设备知道这次事务处理是同步传输的一部分，因为端点地址选择为它的同步端点中的一个。主控制器知道事务处理是同步的，因为系统软件在传输描述符中指出了传输类型。而传输描述符则是由主控制器执行读取工作的。IN 事务处理仅有令牌阶段和数据阶段，并不包括握手阶段，因为在同步传输中，数据发送的完整性等方面并不能得到保证。

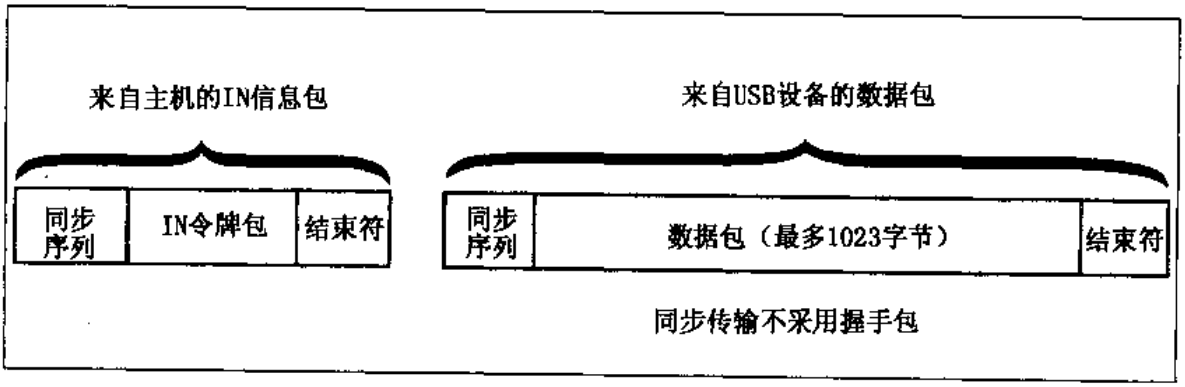


图 7-19　在同步传输中的 IN 事务处理

OUT 事务处理

一个典型的 OUT 事务处理由令牌阶段、数据阶段和握手阶段组成。然而在某些情况下，一个 OUT 事务处理可以不由 3 个阶段组成，OUT 事务处理在执行块传输、控制传输和同步传输时可能被采用。当执行一个 OUT 事务处理时可能发生以下几种状态：

- 数据被无错误发送；
- 数据发送出错；
- 目标设备暂时不能返回数据；
- 错误状态被清除之前，目标设备不能接收数据；
- 发生了一次同步传输数据被发送到目标设备，但是在数据阶段之后没有握手。

这些状态中每一个都将在下面的部分加以详细讨论。

☑ 无数据包错误的 OUT 事务处理

图 7-20 所示的就是一个正常的 OUT 事务处理，在这种情况下，数据被成功地发送到目标设备而不会出错，目标设备无错误地接收到数据后经根集线器返回一个 ACK 握手包。这样的 OUT 事务处理在块传输和控制传输期间都可能会发生。

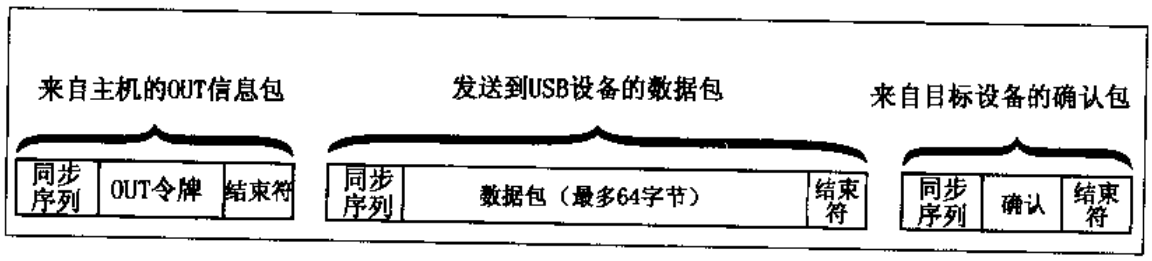


图 7-20　无错误的 OUT 事务处理

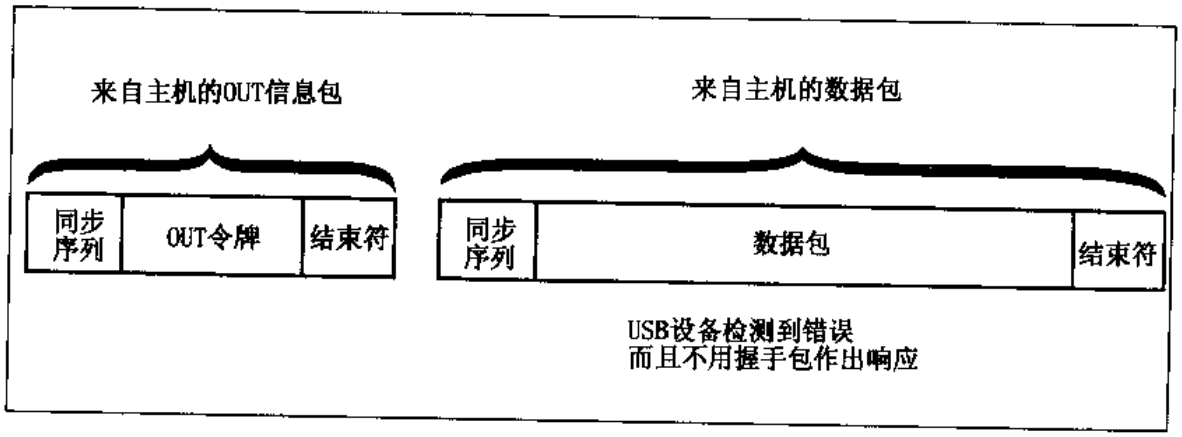


图 7-21　数据包出错的 OUT 事务处理

☑ 出错或目标设备忙的 OUT 事务处理

图 7-22 假定所执行的事务处理和前面一段所描述的事务处理相同，只是目标设备在接收到数据的时候检测到了错误。此时目标设备丢弃了数据，并且没有发出握手包作出应答。由于所期望的握手包并没有出现，所以主机将会检测到总线超时状态，主机将会在以后的一段时间内重试这次 OUT 事务处理。

☑ OUT 事务处理——目标设备不能接收数据

如果目标设备暂时不能接收数据，那么事务处理将会如图 7-22 所示。目标设备会无错误地接收到数据，但是不能接收该数据（例如：因为忙或者缓冲区满的状态）。相应地，在该事务处理握手阶段，目标设备将返回 NAK。这就通知了主机，目标设备并未接收到数据，而且事务处理必须在以后重新尝试。

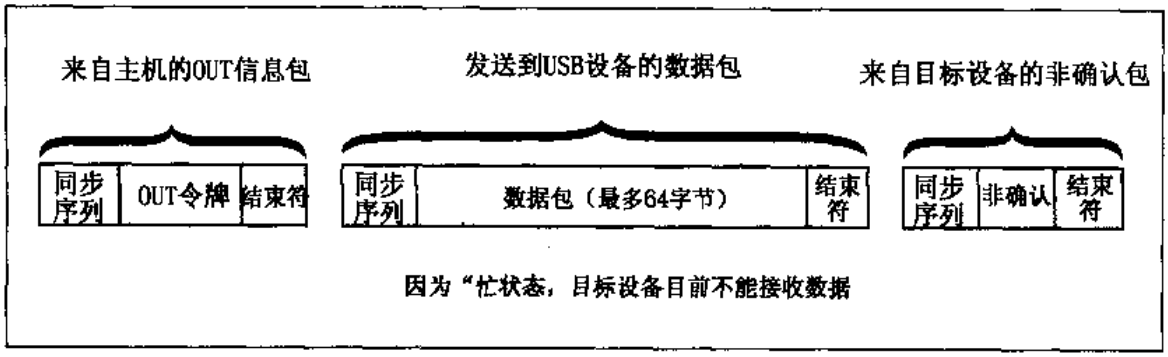


图 7-22　不能接收数据的目标的 OUT 事务处理

☑ 目标设备停止的 OUT 事务处理

图 7-23 所示的事务处理是因为目标设备由于一个内部错误状态接收到数据。目标设备接收到 OUT 令牌，认为它已经进行编址了，然而当主机发送数据包之后，目标设备不能接收到数据，所以在该事务处理的握手阶段给主机返回一个 STALL 状态。这个状态就被告诉主机软件，主机软件就会清除这个目标设备内的错误状态。

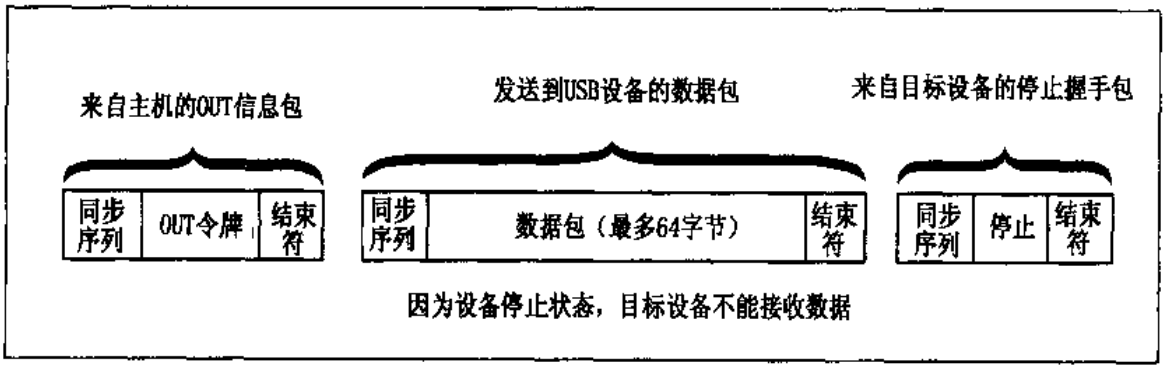


图 7-23　停止端点的 OUT 事务处理

☑ 同步传输期间的 OUT 事务处理

图 7-24 所示的是在一个同步传输过程中执行的一个 OUT 事务处理。不管目标设备接收到的数据是否带有错误，都不会产生握手阶段。

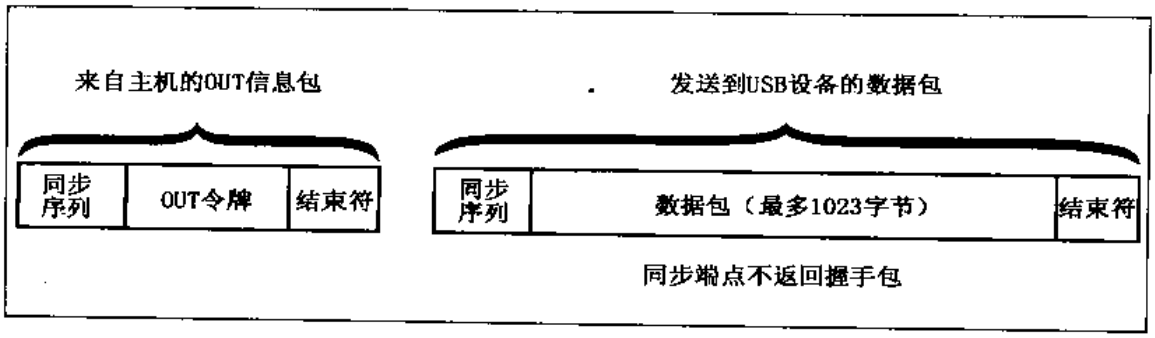


图 7-24　在同步传输期间的 OUT 事务处理

SETUP 事务处理/控制传输

控制传输提供了一种可以配置的机制，经配置后可以向设备发送命令或请求状态，也可以由设备发送命令或请求状态、发送命令、向或从目标设备请求状态。控制传输总是从一个 Setup 事务处理开始，称为建立阶段。建立阶段定义了控制传输的性质。某些控制传输包括一个数据阶段，该数据阶段由一个或几个 IN 或 OUT 事务处理组成，常常由它们来发送控制传输的数据。数据是发送到目标设备还是从目标设备处接收都是在建立阶段定义的，控制器传输的最终阶段是状态阶段。该阶段确认所要求的操作已经成功完成了。控制器传输以两种基本形式存在：

- 传输由建立阶段和状态阶段组成；
- 传输由建立阶段、数据阶段和状态阶段组成。

SETUP 事务处理的数据阶段包括 8 字节的信息，如表 7-3 所示，该信息指出了大量用于定义将被执行的设备请求信息。

表 7-3　　SETUP 事务处理数据阶段的格式

字段编号	字段名	字段大小	字段取值	说　明
0	请求类型	1	位图	请求的特征。 D7 数据传输的方向： 0=主机到设备； 1=设备到主机。 D6..5 类型： 0=标准； 1=类别； 2=供应商； 3=保留。 D4..0=接收方： 0=设备； 1=接口； 2=端点； 3=其他； 4..31=保留。
1	请求	1	数值	特定的请求。

续表

字段编号	字段名	字段大小	字段取值	说　明
2	数值	2	数值	以字长为单位，随请求而变化的字段。
4	索引	2	字段编号索引	以字长为单位，随请求而变化的字段。一般用于传递一个索引或者字段编号。
6	长度	2	计数器	如果本次传输要求有一个数据阶段，它就表示传输的字节数。

☑ 两阶段控制传输

两阶段的状态控制传输仅由建立阶段和状态阶段组成，如图 7-25 所示。在这种情况下，在 SETUP 事务处理阶段发送的 8 字节包含了所有执行指定请求所需要的信息(例如：远程唤醒请求)。状态阶段由一个 IN 事务处理组成，该阶段验证请求是否被成功地处理。

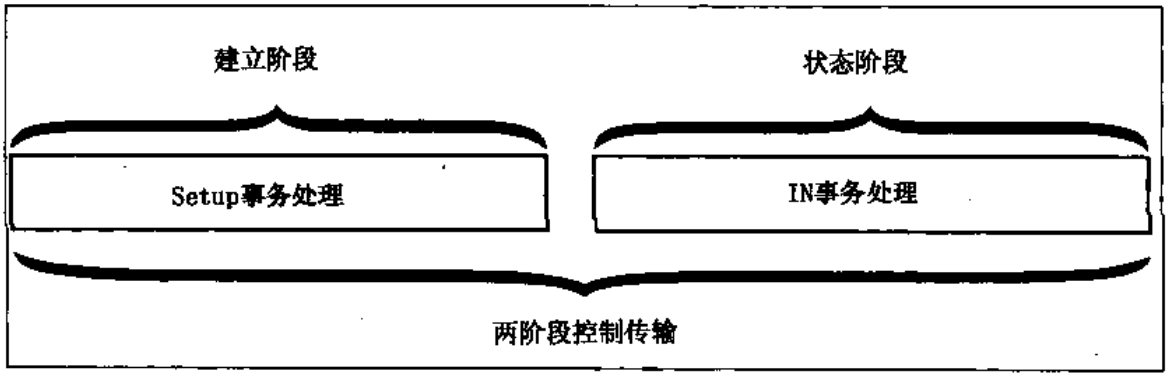


图 7-25　两阶段控制传输的格式

☑ 有 IN 数据阶段的 3 阶段控制传输

图 7-26 所示的就是一个控制传输，要求数据从目标设备返回到主机。例如，当主机系统发送一个读取设备描述符的请求后，就可以执行一次控制传输。这次传输的第一阶段由 SETUP 事务处理组成，它定义了控制传输的性质。SETUP 事务处理由建立令牌、数据包和握手组成。在 SETUP 事务处理的过程中，数据总是被发送到目标设备，指出请求的类型。在建立阶段之后，主机初始化一个或多个 IN 事务处理，提示目标设备返回主机请求的数据。主机用一个 OUT 事务处理来结束控制传输，该 OUT 事务处理请求验证设备端点是否已经成功返回了描述符的内容。注意如果目标设备在 OUT 事务处理过程中发送了一个 ACK 握手，那就表示请求已经成功地执行完了。主机发送的 OUT 数据包的长度为 0。

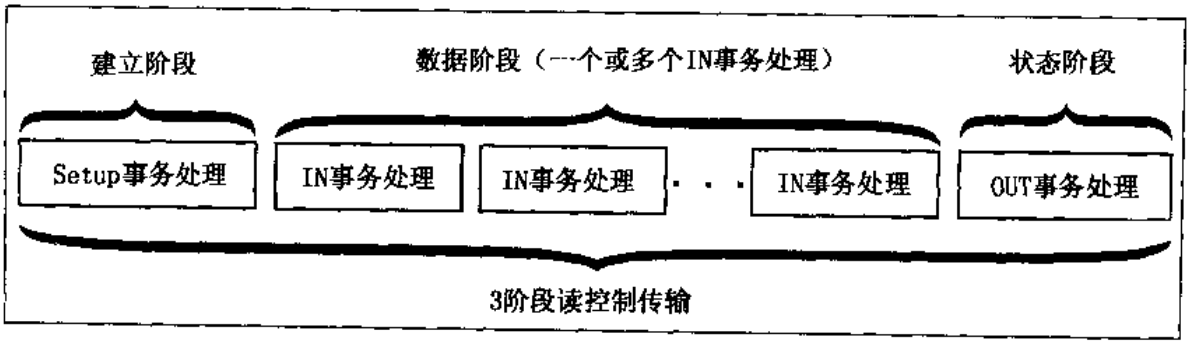


图 7-26　来自目标器的控制传输请求数据

☑ 有 OUT 数据阶段的 3 阶段控制传输

图 7-27 所示的是一种控制传输的格式，在这种格式中，命令被发送到一个控制器。SETUP 事务处理定义了被发送的请求，而且后接数据（一个或多个 OUT 事务处理）。然后主机向控制端点发送一个 IN 令牌，从而获得完成状态。目标设备返回一个长度为 0 的数据包，指出它已经成功地处理了启动控制请求。

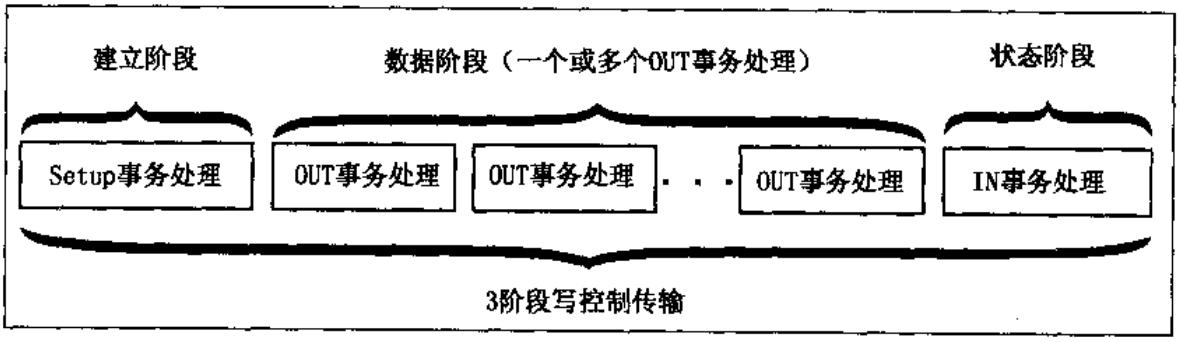
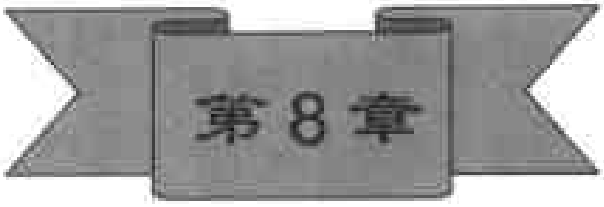


图 7-27　控制传输向目标器的控制端点发送一条命令

☑ 出错的控制传输

主机系统尝试重试传输，以确保控制传输成功完成。目标设备和主机采取的动作依赖于错误状态的本质以及传输期间错误发生的时刻，第 8 章将详细介绍相关内容。



错 误 恢 复

✎ 上一章

上一章介绍了事务处理和信息包的类型，并且解释了在每种传输类型中它们是如何被使用的。对每一种传输类型的不同事务处理类型我们都加以了说明。

✎ 本 章

除了同步传输之外，所有的传输都是在 USB 上完成的，即使检测到错误状态的时候也是这样。本章讨论了错误的类型，错误检测的机制，并且解释了错误恢复机制。

✎ 下一章

下一章讨论了 USB 上的电源分配，并且讨论了和总线供电设备相关的事情。此外还讨论了主机软件在检测和报告与电源相关的问题时所起的作用。

🕮 概述

在通过了 USB 进行数据传输时，很多错误状态都是可以由硬件检测到的。前一章已经介绍了握手包，它被设计到 USB 事务处理协议内部，用它来验证一个包是否已经被成功地接收到。本章详细介绍了 USB 错误检测机制，并且描述了相关的错误恢复过程。USB 所支持错误监测机制包括：

- 信息包错误检查；
- 错误 EOP；
- 总线超时（没有响应）；
- 数据触发器错误检查；

- 串扰——在时间片结束之后发生事务处理；
- LOA——总线活动丢失。

信息包错误

USB 设备检测三种类型的包错误：

- 包 ID（PID）检查；
- 循环冗余校验（CRC）；
- 位填充错误。

如果发生了以上三种错误状态中的任何一个，那么这个包的接收方就会忽略这个信息包，并且不会以任何方式对它作出响应。如果接收方接收到的信息包含有任何错误的话，那么接收方绝不会向发送方发回一个信息包。注意，对于 USB 设备或者主机来说，信息包的错误类型检测并不是很重要，因为它和错误恢复有关。然而主机系统可以获取关于信息包错误性质的统计数字。下面的部分讨论了和信息包相关的各种形式的错误。

PID 检查

在 USB 上广播的每个包都是以一个包 ID（PID）开始的，它由四个位组成，并且后接一个 PID 的校验字段，如图 8-1 所示。这个校验字段是 PID 的反码（1 的补码）。所有可能的 USB 设备必须进行 PID 的校验，如果发现错误这个信息包就会被忽略，因为如果发生错误的话，这个信息包的类型就是不可知的。

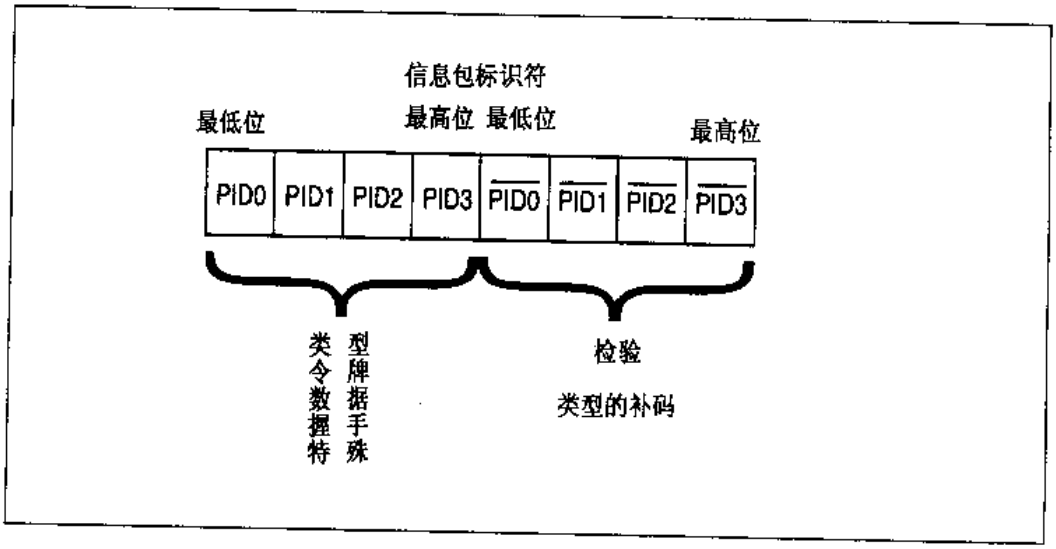


图 8-1 PID 检查

CRC 错误

每个信息包都包含 CRC 位，这些位用来验证那些跟在 PID 后面的信息。这些信息特征随着信息包的类型不同而不同，每个信息包都包含 5 位或者 16 位 CRC，这取决于信息包的大小。此外，CRC 中关于信息包的类型部分的字段也是由信息包的类型决定的，参见表 8-1。

表 8-1　　信息包的类型和 CRC

信息包的类型	字 段 名	字段的最大长度	CRC 的位的数目
时间片的开始	时间片编号	11 位	5
IN	设备和端点地址	11 位	5
OUT	设备和端点地址	11 位	5
SETUP	设备和端点地址	11 位	5
DATA0	数据量	1023 字节	16
DATA1	数据量	1023 字节	16
ACK	NA－仅用于信息包的 ID	NA	NA
NAK	NA－仅用于信息包的 ID	NA	NA
STALL	NA－仅用于信息包的 ID	NA	NA
PREAMBLE	NA－仅用于信息包的 ID	NA	NA

令牌包的 5 位 CRC 字段基于如下的生产多项式：

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

代表这个生成多项式的位模型是：00101b。如果数据被接收方无错误地接收到，那么余项应该是 01100b。

数据包的 16 位 CRC 字段基于如下的生成多项式：

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

代表这个生产多项式的位模型是：1000000000000101b。如果数据被接收方无错误地接收到，那么余项应该是 1000000000001101b。

注意如果 CRC 包含 6 个连续的 1，那么 CRC 就必须含有填充位。

位填充错误

位填充保证了 NRZI 数据的发送方和接收方维持同步，它的实现机制是：如果数据流中检测到 6 个连续的“1”，那么就强制插入一个跳变。可以参见标题为“位填充”的部分，那里有关于这个机制的更详细的说明。

USB 接收方在数据流中收到 6 个连续的“1”之后总是希望看到一个确定的跳变（填充位）。如果这个应该出现的填充位没有出现，那么就说明这个包已经被破坏了，或者是发送方没有正确地产生填充位，但也可能是接收方没有对 NRZI 数据进行正确的解码。

信息包相关错误处理

任何一个信息包相关的错误处理都会引起同样的出错报告。在任何一种情况下，被破坏的包的接收方必须忽略这个包并且不作出任何响应。然而，具体的出错处理的方法则与事务处理的类型以及给定事务处理的被破坏的包的类型有关。

☑ 令牌包错误

✓ IN 包错误

考虑一个 IN 包发生了错误的情况。由于在令牌包中检测到一个错误，目标设备将会忽略这个信息包，并且不对主机作出响应。主机则希望目标设备返回一个数据包或者握手包，作为对 IN 包的响应。于是主机将检测到总线超时和事务处理失败，然后主机将会重新进行那次失败的事务处理。

✓ OUT 或者 SETUP 包错误

如果主机广播 OUT 或者 SETUP 包时，有一个包错误被检测到，那么主机将会在令牌包后跟随一个数据包。目标设备也许可以对数据包进行正确的解码，但是并不能确定自己就是主机所希望的接收方（因为地址 CRC 错误），或者不能检测到数据包的含义（因为 PID 检验错误）。由于目标设备没有对 OUT 令牌作出响应，也没有对后续的数据包作出响应，主机将会检测到一个总线超时，并且知道事务处理已经失败。然后主机将会重新安排事务处理。

☑ 数据包错误

✓ 在 OUT 或 SETUP 事务处理期间

在数据包中发生错误将会引起接收设备丢弃数据并且不对发送方作出响应。在 OUT 事务处理期间，目标设备返回有效数据，但是主机接收到的是一个破坏的数据包。由于主机没有以 ACK 握手包作为响应，目标设备被告知主机没有接收到数据。然后主机必须重新进行这次事务处理。

✓ 在 IN 事务处理期间

在 IN 事务处理期间，目标设备将会返回有效的数据，但是主机接收到的是一个被破坏的数据包，因为主机不会对 ACK 握手包作出响应，目标设备被通知主机没有接收

到数据。主机必须重新进行这次事务处理。

☑ 握手包错误

握手包错误使目标设备和主机在事务处理是否成功完成的问题上出现意见分歧。下面的内容详细解释了这个问题。

✓ 在 OUT 事务处理期间

在 OUT 事务处理期间，令牌包和数据包可能是被正确接收到的，所以目标设备将向主机返回一个 ACK 握手包，表示数据已经被正确的接收到了。当 ACK 包被接收到时，如果检测到错误，那么主机将会忽略这个握手包，而这次握手也会以失败而告终。但是目标设备事实上接收到了数据，只是被破坏的握手包向主机误报了完成状态。那么现在，主机和目标设备在事务处理是否成功完成的问题上就出现意见分歧。

主机确信错误已经发生了，所以它将会尝试再次发送数据包。而目标设备则认为数据包已经被成功地接收了，并且希望接收下一个数据项。这就会引起一个有效的数据包被接收两次，这完全是因为主机和目标设备之间的同步问题造成的。

✓ 在 IN 事务处理期间

当在目标设备和主机之间成功进行数据数据传输时，相似的情况也可能在 IN 事务处理中发生。主机向目标设备返回 ACK，但是目标设备检测到一个 ACK 包的错误。目标设备没有得到传输完成的确认信息，但是事实上主机已经成功地收到了数据。

主机认为传输已经成功地完成了，它要求传输下一个数据项。然而，目标设备认为传输已经失败了，然后会再次发送同样的数据。

在目标设备和主机之间的这种不能达成一致的情况必须通过使用数据触发机制来处理。参见标题为“数据触发机制”的部分。

🕮 总线超时

在一个事务处理中的发送方和接收方必须知道它们要等待多久来获得响应。例如，发送方必须在发送完令牌包和数据包之后从希望的目标设备处获得响应。然而，如果在指定的总线周转期内没有得到响应，就认为是出错了。

规范说明书中把没有响应定义为一种方法，用这种方法通知信息包的发送方该信息包没有被正确的接收到。就是说，如果在收到的信息包里检测到错误，那么接收方就不会作出响应。由于没有作出响应，所以包的发送方将会检测到一个总线超时，从而认为在包的传输过程中发生了某些错误。

总线超时定义了从主机出发的下游支持的最大数量的数据线段，从主机到下游端口

传输数据的相关延迟时间包括：

- 数据线延迟＝30ns　最大；
- 集线器延迟＝40ns　最大。

从根集线器的下游端口到第一个集线器的下游端口的总时间延迟的最大值是 70ns。最坏的情况是在一个下游的路径上有 6 个数据线段，就是图 8-2 中的情况。在主机端口到设备数据线的上游端点之间的总的来回时间延迟是 700ns。此外，当信号在目标设备数据线上传输时，也会产生额外的延迟。然后目标设备对令牌包解码，访问选定的端点，对将要返回到数据线的上游端点的返回包作初始化。从设备数据线的上游端点那里信息包结束（EOP 到空闲的转换）算起，直到来自设备的 SOP 转换返回到目标设备数据线的上游端点为止的那一段时间被定义为 7.5 个位时间延迟。

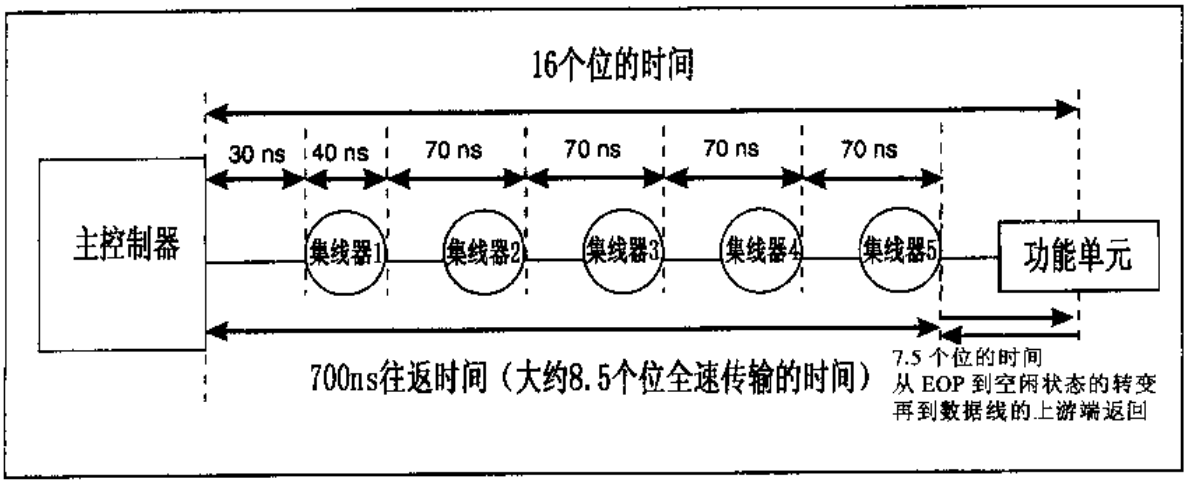


图 8-2　全程延迟

事务处理的发送方在 16 个位时间内一定不会超时，但是在 18 个位时间后则一定超时。对于全速和低速事务处理而言都是如此。

错误的 EOP

如果接收方在发送方实际完成传输之前检测到包的结束（EOP），那么就会产生问题，这种情况被称为错误的 EOP。如果接收方对这个过早的 EOP 作出响应，比如返回一个数据包或者握手包，那么在总线上就会产生碰撞。碰撞的结果会对两个连续的包造成破坏。幸运的是错误的 EOP 能够被检测到，所以这种碰撞也能被避免。造成信息包被截断的错误 EOP 往往会导致一个 CRC 错误，因为一个截断的包的 CRC 校验基本上不可能是对的。下面的部分说明了检测和响应机制。

在主机传送期间的错误 EOP

检测到任何形式的信息包错误的 USB 目标设备简单地忽略这个信息包，不对主机作出响应。所以，如果发生了 EOP 错误，那么产生的 CRC 错误就会迫使目标设备等待主机发送下一个信息包。这种被动的行为避免了碰撞的发生。另外，当主机没有从目标设备出得到响应时，它就认为信息包的传输已经失败了，然后会重新进行传输。

由于上面描述的错误状态来自于一个错误的 EOP，主机可以在目标设备检测到错误的 EOP 后继续传输数据。目标设备认为包的剩余部分是无效的，而它最终会在包结束处检测到真正的 EOP。然后目标设备等待主机发送下一个信息包。

目标设备发送中的错误 EOP

如果目标设备正在发送数据，而主机检测到一个错误的 EOP，那么就会出现 CRC 错误，而主机就会忽略这个包，不对目标设备作出响应。由于没有响应，所以也就不会产生碰撞。因为目标设备没有收到握手信号，它就知道主机没有收到包，于是它就等待下一次事务处理初始化。

在这种情况下，主机需要知道什么时候可以安全地发送下一个数据包。由于过早的检测到了 EOP，目标设备可以继续发送数据。如果主机在 16 个位时间的总线超时周期内没有检测到别的总线事务，它就可以认为目标设备没有发送别的数据。然后主机就可以安全的发送下一个信息包，并且确信不会在总线上产生碰撞。然而，如果总线转换继续，则主机将等待 EOP，然后在发送下一个信息包之前等待 16 个位时间。注意，167 个位时间是必须的，这是为了保证目标设备能够检测到主机没有响应（就是说目标设备的超时计数器溢出）。这就确保了目标设备可以认为信息包的传输已经失败了。

数据触发出错

数据触发是一种机制，用来确保数据传输的发送方和接收方之间保持同步，特别是在需要大量的单独的事务处理的一次长传输过程中。数据触发机制解决了和握手包出错相关的问题，该问题在标题为“握手包错误”的部分有叙述。

☑ 无错误的数据触发处理

数据触发仅仅支持中断传输、块传输和控制传输。支持数据触发机制的一次数据传输的发送方和接收方必须都有触发位。当双方都认为数据传输已经正确完成以后，发送方和接收方都会把它们的触发位转换为和原来相反的状态。两种数据包的类型（DATA 和 DATA1）被交替传送，并且包的接收方将会把它和触发位作比较，以确定接收的包

是否正确。发送方使用的数据包的类型和它的当前的触发位保持一致。（例如，如果触发位是 0，那么数据包就用 DATA0）。为了解释数据触发机制的概念，可以考虑下面的事务处理，参见下面的部分：

☑ 在 OUT 事务处理期间的数据触发

图 8-3 所示的就是从主机到目标设备的块数据传输过程中信息包序列和触发位的转变情况。假定发送方和接收方的触发位开始都是 0。传输的处理过程如下：

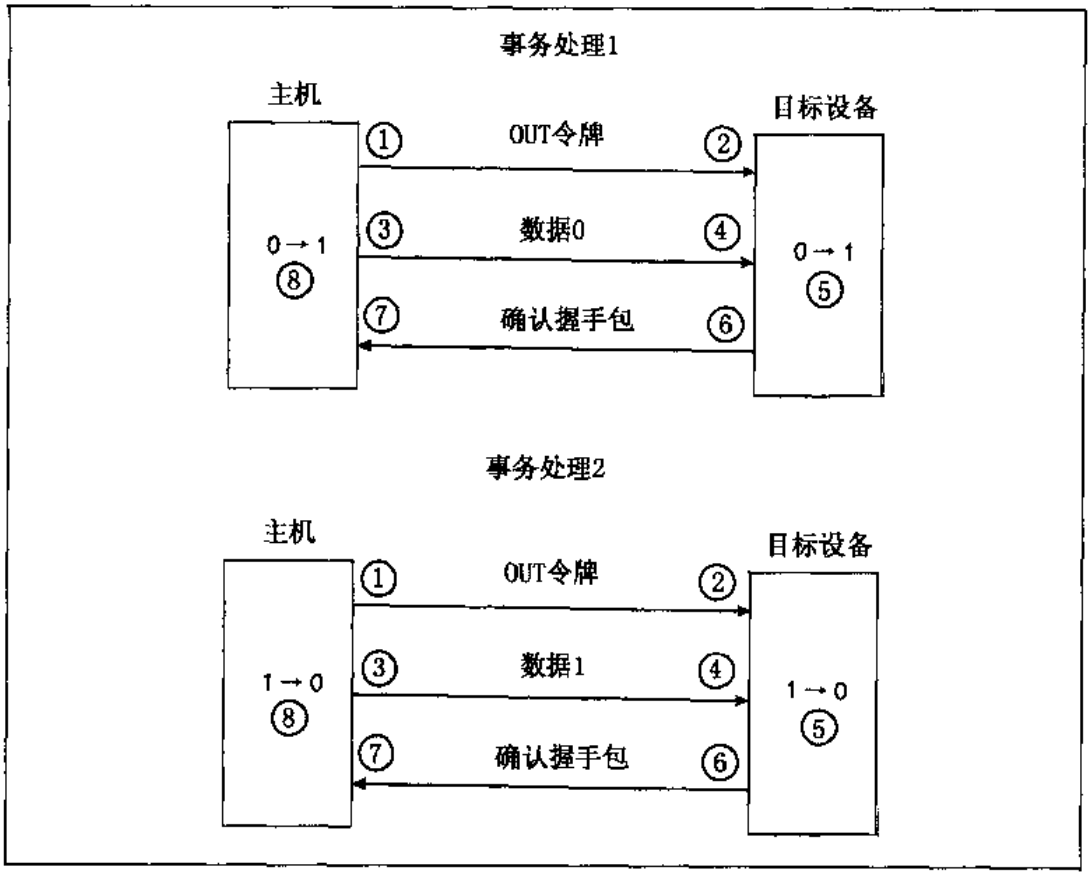


图 8-3　数据触发序列和无错误的 OUT 事务处理

✓ 事务处理 1

（1）主机向目标设备发送一个 OUT 令牌。
（2）目标设备无错误地收到该令牌包。
（3）主机向目标设备发送一个 DATA0 数据包（和它的触发位保持一致）。
（4）目标设备收到数据包 DATA0，它和触发位一致。
（5）成功收到数据包 DATA0，触发位转变为 1。
（6）目标设备向主机发送一个 ACK 握手包，通知主机数据已经被无错误地接收到了。

（7）主机无错误地收到 ACK 握手包。

（8）成功地收到 ACK 握手包，主机把触发位转换为 1。

✓ 事务处理 2

（1）对目标设备的下一个事务处理从发往目标设备的下一个 OUT 包开始。

（2）目标设备无错误地收到 OUT 令牌包。

（3）主机向目标设备发送一个 DATA1 数据包（和它的触发位保持一致）。

（4）目标设备收到数据包 DATA0，它和触发位一致。

（5）目标设备成功收到数据包 DATA1，触发位转变为 1。

（6）目标设备向主机发送一个 ACK 握手包，通知主机数据已经被无错误地接收到了。

（7）主机无错误地收到 ACK 握手包。

（8）成功地收到 ACK 握手包，主机把触发位变为 0。

这个处理过程对每一次事务处理都是这样，直到整个传输完成。只要收到的数据包和触发位相一致，并且发送方无错误得接收到 ACK 握手包，那么发送方和接收方就保持同步。

☑ 在 IN 事务处理中的数据触发机制

图 8-4 所示的就是两个 IN 事务处理的顺序，在一次无错误的数据传输过程中将会发生触发位的转变。该过程发生的顺序如下：

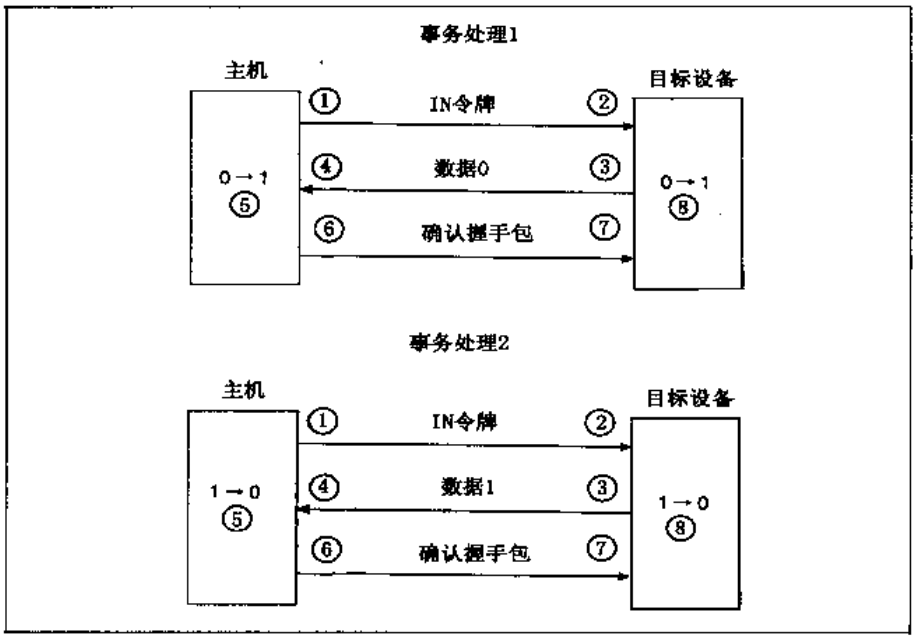


图 8-4　数据触发序列和无错误的 IN 事务处理

✓ 事务处理 1

（1）主机向目标设备发送一个 IN 令牌。
（2）目标设备无错误得收到该令牌包。
（3）然后接收方向主机返回一个 DATA0 数据包（和它的触发位保持一致）。
（4）主机收到数据包 DATA0，它和触发位一致。
（5）成功收到数据包 DATA0，触发位转变为 1。
（6）主机向目标设备发送一个 ACK 握手包，通知目标设备数据已经被无错误地接收到了。
（7）目标设备无错误地收到 ACK 握手包。
（8）成功地收到 ACK 握手包，目标设备把触发位转换为 1。

✓ 事务处理 2

（1）对目标设备的下一个事务处理从发往目标设备的下一个 IN 包开始。
（2）目标设备无错误地收到 IN 令牌包。
（3）目标设备返回一个 DATA1 数据包（和它的触发位状态保持一致）。
（4）主机无错误地收到数据包 DATA1，它和主机的触发位一致。
（5）主机成功收到数据包 DATA1，主机的触发位转变为 1。
（6）主机向目标设备发送一个 ACK 握手包，通知目标设备数据已经被无错误地接收到了。
（7）主机无错误地收到 ACK 握手包。
（8）成功地收到 ACK 握手包，主机把触发位转换为 0。

这个处理过程对每一次事务处理都是这样，直到整个传输完成。只要收到的数据包和触发位保持一致，并且发送方无错误得接收到 ACK 握手包，那么发送方和接收方就保持同步。

数据包出现错误时的数据触发处理

如果数据包的错误仅仅在传输期间发生，那么主机和目标设备的触发位都不会改变。下面的部分描述了当数据包的错误发生时，数据触发的操作情况。

☑ 数据触发和数据包错误——OUT 事务处理

图 8-5 所示的就是连续的 OUT 事务处理的处理顺序，在第一个事务处理期间，发生了错误。

主机和目标设备所采取的行动如下：

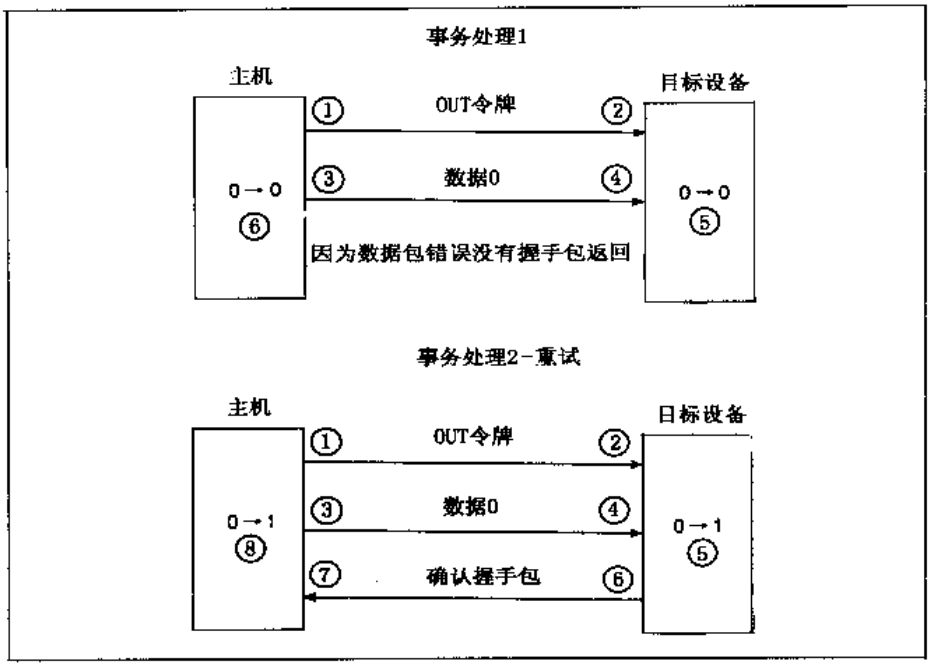


图 8-5　数据触发和数据出错的 OUT 事务处理

✓ 事务处理 1

（1）主机向目标设备发送一个 OUT 令牌。
（2）目标设备无错误地收到该令牌包。
（3）然后主机通过一个 DATA0 数据包向目标设备发送数据（和它的触发位保持一致）。
（4）当目标设备收到数据包 DATA0 时，发生了错误。
（5）因为检测到了数据包的错误，目标设备将会忽略这个包。数据被丢弃，也不会有握手包发往主机。因为数据包没有被正确的接收到，触发位保持不变。
（6）主机等待握手包的返回，但是没有响应。在总线的超时周期（16 个位时间）到达以后，主机检测到没有响应，于是它认为数据包的传输不成功。主机的触发位保持不变，主机必须在以后重新进行此次事务处理。

✓ 事务处理 2--重新进行

（1）主机再次向目标设备发送 OUT 令牌包。
（2）目标设备无错误地收到 OUT 令牌包。
（3）然后主机向目标设备发送一个 DATA0 数据包（和它的触发位保持一致），这个数据包在前一次传输中没有成功。
（4）目标设备成功地收到数据包 DATA0（它和触发位保持一致）。
（5）因为正确的收到了数据包 DATA0，所以目标设备的触发位变为 1。
（6）目标设备发送一个 ACK 握手包来通知主机数据已经被无错误地接收到了。

（7）主机无错误地接收到了 ACK 握手包。

（8）成功地收到了 ACK 包，主机把它的触发位变为 1。

由此可以看出，尽管传输发生了错误，但是主机和目标设备还是能够保持同步。

☑ 数据触发和数据包错误——IN 事务处理

图 8-6 所示的就是在一个连续的 IN 事务处理期间的一系列包的传输处理过程。在本例中，一个 IN 事务处理过程中发生了一个数据包错误。

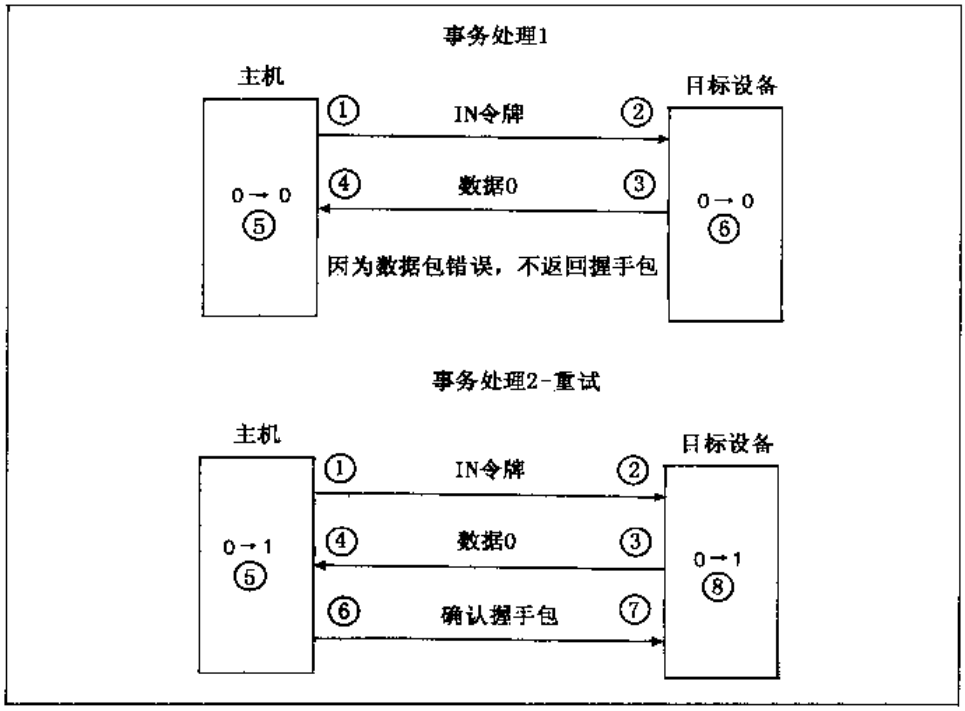


图 8-6　数据触发和数据包错误的 IN 事务处理

主机和目标设备采取的行动如下：

✓ 事务处理 1

（1）主机向目标设备发送一个 IN 令牌包。

（2）目标设备无错误地收到该令牌包。

（3）目标设备通过一个 DATA0 数据包向主机返回数据（和它的触发位保持一致）。

（4）主机收到数据包 DATA0 时遇到了包错误。

（5）因为检测到了数据包的错误，主机将会忽略这个包。数据被丢弃，也不会有握手包发往目标设备。因为数据包没有被正确的接收到，触发位保持不变。

（6）目标设备等待握手包的返回，但是没有响应。在总线的超时周期（16 个位时间）到达以后，目标设备检测到没有响应，于是它认为数据包的传输不成功。目标设备的触发位保持不变，主机必须在以后重新进行此次事务处理。

✓ 事务处理 2--重新进行

（1）主机再次向目标设备发送 IN 令牌包。
（2）目标设备无错误地收到 IN 令牌包。
（3）然后目标设备向主机发送一个 DATA0 数据包（和它的触发位保持一致），在前一次传输中没有成功。
（4）目标设备成功地收到数据包 DATA0（它和触发位一致）。
（5）因为正确的收到了数据包 DATA0，所以主机的触发位变为 1。
（6）主机发送一个 ACK 握手包来通知目标设备数据已经被无错误地接收到了。
（7）目标设备无错误地接收到了 ACK 握手包。
（8）成功地收到了 ACK 包，目标设备把它的触发位变为 1。

由此可以看出，尽管传输发生了错误，但是发送方和接收方还是能够保持同步。

握手包错误时的数据触发

上面已经讨论了一些主机和目标设备采用的数据触发机制，主要是在数据包不出错或出错时的情况。尽管目标设备和主机的触发位保持同步是重要的，但是错误恢复机制并不需要使用数据触发位。然而，如果在握手期间发生了错误，主机和目标设备就变得不同步，如果不用数据触发机制，那么数据就会丢失。

☑ 握手错误的数据触发机制——OUT 事务处理

图 8-7 所示的是一个 OUT 事务处理，这次事务处理由于 ACK 包错误而失败。对错误的检测和恢复的每一步列举如下：

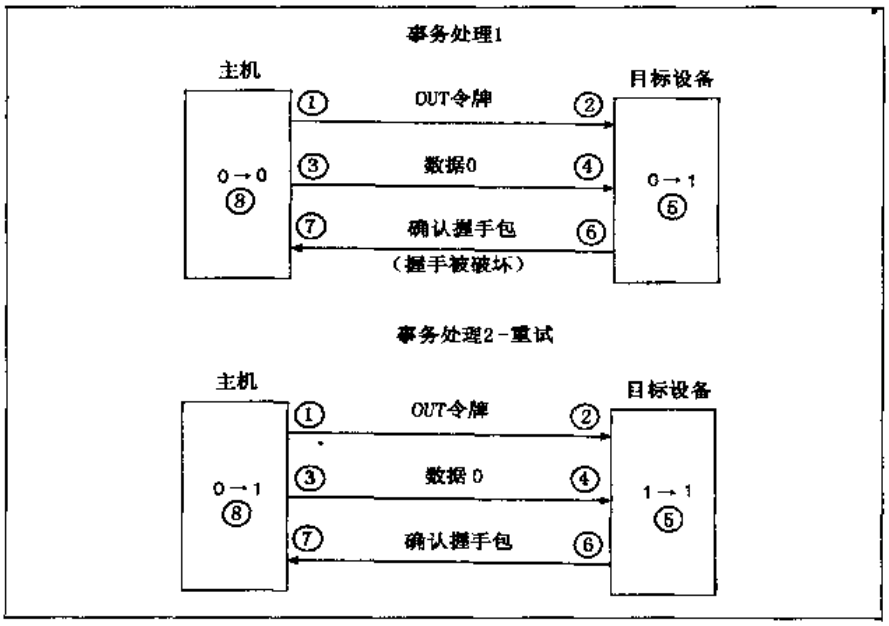


图 8-7　数据触发和握手错误的 OUT 事务处理

✓ 事务处理 1

（1）主机向目标设备发送一个 OUT 令牌。
（2）目标设备无错误地收到该令牌包。
（3）然后主机向目标设备发送一个 DATA0 数据包（和它的触发位保持一致）。
（4）目标设备收到数据包 DATA0，它和触发位一致。
（5）成功收到数据包 DATA0，触发位转变为 1。
（6）目标设备向主机发送一个 ACK 握手包，通知主机数据已经被无错误地接收到了。
（7）主机收到的 ACK 握手包出现了错误。
（8）由于主机检测到了错误，所以它不能验证目标设备已经成功地接收到数据。因而，主机的触发位没有改变（还是 0）。主机假定目标设备没有收到数据，因而会重新启动一次传输。

✓ 事务处理 2

（1）主机向目标设备发送 OUT 包。
（2）目标设备无错误地收到 OUT 令牌包。
（3）主机重新向目标设备发送一个 DATA0 数据包（和它的触发位状态保持一致）。
（4）目标设备无错误地收到数据包 DATA0，但是它和触发位不一致。
（5）目标设备认为它自己和主机之间不同步，因而丢弃数据，并且保持触发位不变（1）。
（6）目标设备向主机发送一个 ACK 握手包，通知主机数据已经被无错误地接收到了，这是因为主机明显没有收到前一次的 ACK 握手包。
（7）主机无错误地收到 ACK 握手包。
（8）成功地收到 ACK 握手包，主机把触发位转换为 0。现在主机和目标设备就为下一次事务处理作好了准备。

主机和目标设备暂时在数据传输是否完成的问题上没有达成一致。然而，数据触发机制保证了不同步的状态能够被检测到，并且能够重新同步。

☑ 握手错误的数据触发机制——IN 事务处理

图 8-8 所示的是一个 IN 事务处理，这次事务处理由于 ACK 包错误而失败。对错误的检测和恢复的每一步列举如下：

✓ 事务处理 1

（1）主机向目标设备发送一个 IN 令牌。
（2）目标设备无错误地收到该令牌包。
（3）目标设备向主机发送一个 DATA0 数据包。

（4）主机无错误收到数据包 DATA0，它和触发位一致。

（5）成功收到数据包 DATA0，主机触发位转变为 1。

（6）主机向目标设备发送一个 ACK 握手包，通知目标设备数据已经被无错误地接收到了。

（7）目标设备收到的 ACK 握手包出现错误。

（8）由于目标设备检测到了错误，所以它不能验证主机已经成功地接收到数据。因而，主机的触发位没有改变（还是 0）。目标设备假定主机没有收到数据，因而会尝试重新进行一次传输。目标设备将会在下一次事务处理中重新返回相同的数据。

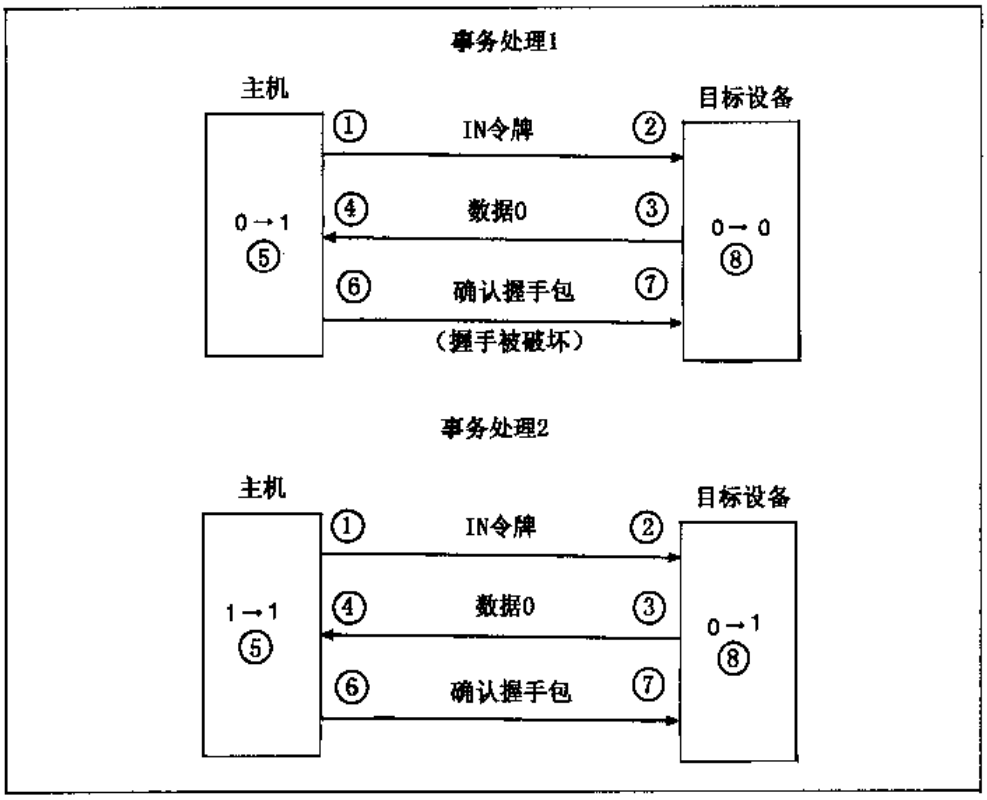


图 8-8　数据触发和握手错误的 IN 事务处理

✓ 事务处理 2

（1）主机向目标设备发送 IN 令牌以获得后续的数据。

（2）目标设备无错误地收到这个令牌包。

（3）目标设备重新向主机发送一个 DATA0 数据包（和它的触发位状态保持一致）因为它相信主机在上一次上一次事务处理中没有收到数据。主机的触发位保持不变。

（4）主机无错误地收到数据包 DATA0，但是它和触发位不一致。

（5）主机认为它自己和目标设备之间不同步，于是丢弃数据，因为它知道在上一

次传输过程中主机已经正确的收到了同样的数据。并且保持触发位不变（1）。

（6）主机向目标设备发送一个 ACK 握手包，通知目标设备数据已经被无错误地接收到了，这是因为目标设备明显是没有收到前一次的 ACK 握手包。

（7）目标设备无错误地收到 ACK 握手包。

（8）成功地收到 ACK 握手包，目标设备把触发位转换为 1。现在主机和目标设备就为下一次事务处理作好了准备。

主机和目标设备暂时在数据传输是否完成的问题上没有达成一致。然而，数据触发机制保证了不同步的状态能够被检测到，并且能够重新同步。

特殊情况：在控制传输中的数据触发机制

在控制传输过程中，数据触发机制用来保证主机和目标设备之间保持同步。SETUP 事务处理从一个 DATA0 数据阶段开始，并且每一个后续的数据阶段都由 DATA0 和 DATA1 交替进行，如图 8-9 所示。

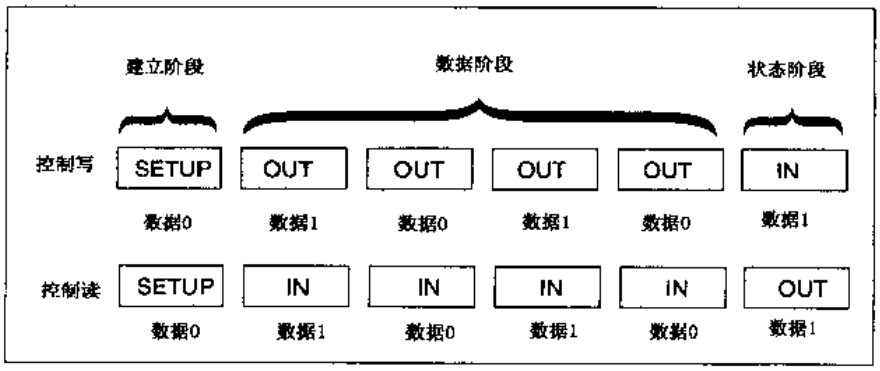


图 8-9　在控制传输中的数据触发

在一个控制读期间，当一个数据触发错误发生在数据阶段的最后一个数据事务处理上时，就会产生一个问题。如果数据阶段的最后一个 IN 事务处理产生一个错误的握手，目标设备就认为根集线器没有成功地收到最后一个 IN 事务处理，并且希望重试一次。目标设备不改变它的触发位，并且准备重发 IN 数据。根集线器收到了数据阶段的最后一个 IN 事务处理，而且没有错误地改变它的触发位，然后主机发送一个 OUT 事务处理，进入 SETUP 阶段。在这种情况下，数据触发处理失败，因为目标设备希望重新传输，而且根集线器已经转移到了 OUT 状态阶段。

这个问题可以采用一个特殊的协议来避免，该协议要求主机在成功地收到数据阶段的最后一个 IN 事务后发送一个 OUT 建立令牌。目标设备认为令牌指示了一个 OUT 数据阶段，把主机的行为解释为验证最后一个 IN 事务处理成功完成。目标设备改变它的触发位，并和根集线器一起进入状态阶段。

串扰

如果在总线上的一个设备没有结束它的事务处理（就是说：它不停地串扰），总线失败就可能发生。这种不断的串扰有可能对整个总线造成死锁，所以这种情况必须避免。串扰在时间片结束时被检测到，因而它的特征是 SOP 和总线行为相继通过时间片的结束处。如果设备由于串扰造成总线在时间片结束时不处于空闲状态，那么这个设备必须被隔离，方法是把和这个设备相连的集线器端口禁止掉。

活动丢失（LOA）

另外一种可能的总线故障是活动丢失，或者说 LOA。LOA 的特征是一个设备从一个包的传输开始，后接总线上的一个恒定的 J 或 K 状态，而且没有 EOP。像串扰一样，LOA 有可能对总线造成死锁，因而必须避免。而且 LOA 错误由集线器在时间片的结束时检测到，这一点也是和串扰错误类似的。

串扰/LOA 检测和恢复

集线器负责检测串扰和 LOA 错误，并从中恢复过来。这些错误的特征是都有确定的特征出现在时间片的末尾。主机在包结束之前中断事务处理，以确保在时间片结束的时候不会有总线活动发生。在正常状态下，最后一个事务处理以一个 EOP 结束，后接一个空闲状态。在时间片结束时，如果总线不处于空闲状态，那么就会产生错误。

时间片计时器

集线器负责跟踪每一个时间片，并在每一个时间片结束的地方对总线行为进行采样。在每一个集线器内的计时器由 SOF 包进行同步，并且在每一个时间片的开始时进行广播。这些计时器必须在连续缺乏两个 SOF 包的时候还能保持同步。

图 8-10 演示了两个采样窗口，它们都靠近时间片的结束处（EOF）。如果集线器检测到在 EOF1 后总线行为（串扰）或者总线处于非空闲状态，那么集线器就必须终止上游的通信，使总线空闲。所有在上游方向的集线器将会检测到总线的空闲状态。然而，如果集线器在 EOF2 后检测到一个非空闲状态，它就必须禁止那个和问题设备相连的端口。以这种方式，集线器能够确保当下一个时间片到来时，总线能够处于空闲状态。

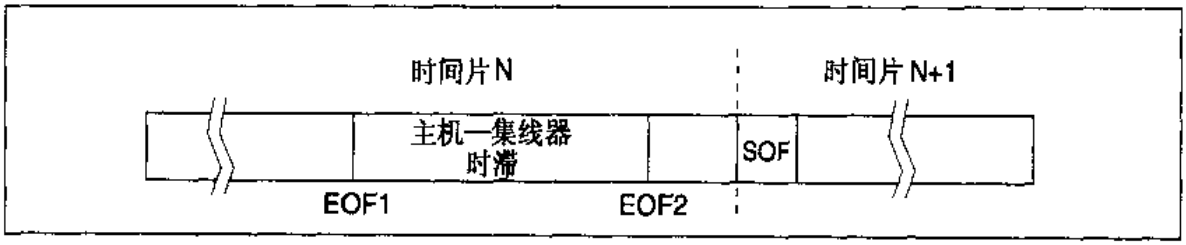


图 8-10　集线器 EOF 点

主机到集线器的时滞

在产生时间片的主机和集线器的计时器之间存在时滞。时滞的原因包括：

- 集线器可能错过最多两个来自主机的连续的 SOF，它的最大时间漂移是±3 个时钟周期；
- 主机的时钟在每一个时间片最多可能调节一个位时间，从而导致 1＋2＋3＝6 个时钟周期漂移。

以上原因就会导致一个最大为 9 个时钟周期的主机－集线器漂移。

为了支持对串扰和 LOA 的检测和恢复，第二个 EOF 点必须和下一个 SOF 点分离。所有的集线器 EOF2 点必须至少比主机发送下一个 SOF 提前一个位时间。如果是因为时滞的问题，一个来自于下游的 EOP 在集线器检测到 EOF2 点之前没有到达集线器，那么集线器将会检测到一个疏忽错误。

图 8-11 所示的就是和最近的一个 EOP 有关的 EOF1 和 EOF2 的范围，该 EOP 由集线器发送。规范说明书定义了 EOF1 点在下一个 SOF 之前的 32 个位时间时发生。集线器可以开始发送 EOP 的最早时刻是在第一个 EOF 点之前的 9 个位时间，或者说是第 41 个位时间。最晚是在第 23 个位时间。EOP 必须至少在下一个 SOF 之前的 19 个位时间完成，或者说是在中间的 EOF2 点（在 SOF 之前的 10 个位时间）之前的 9 个位时间。EOF2 的必须至少提前 SOF1 个位时间发生。

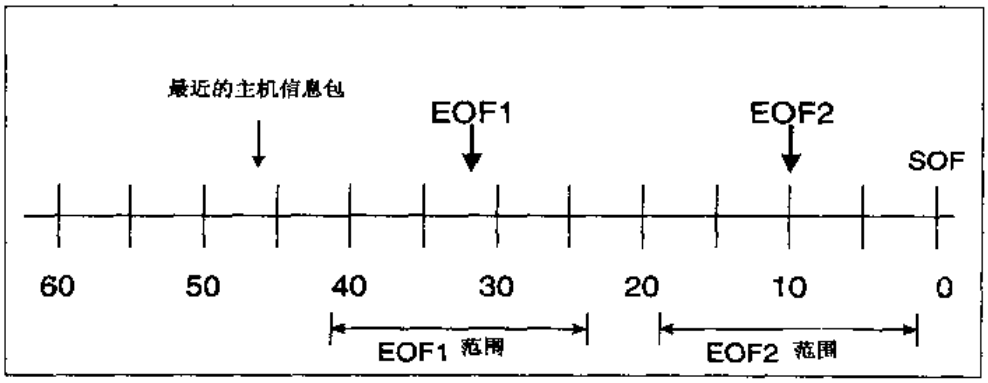


图 8-11　EOF 定时范围

集线器中继器状态机制

图 8-12 所示的就是集线器中继器的状态机制。注意该图所描述的状态机制是基于 1.1 版本的 USB 规范说明书的。在 1.0 版本中的状态图和它是不一样的。

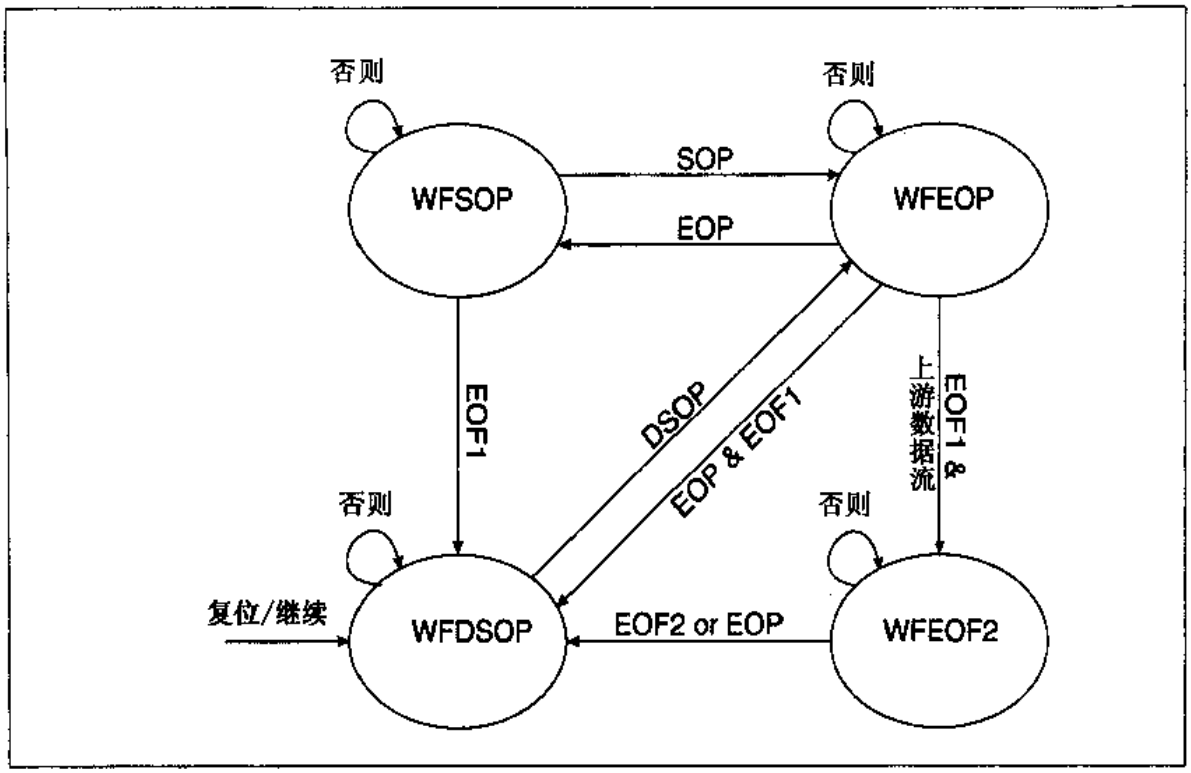


图 8-12　集线中继器的状态图

在信息包快要结束的时候集线器的行为对于 LOA 和串扰检测是关键的。在总线空闲状态时，集线器中继器处于等待包开始（WFSOP）状态。当检测到 SOP 时，中继器转变为等待包结束（WFEOP）状态。当 EOP 发生后，中继器返回到 WFSOP。

通常，在 EOF1 采样点之前包就结束了。在这种情况下，集线器中继器将处于 WFSOP 状态，当 EOF 发生时，它就转变为等待下游的包的开始状态（WFDSOP 状态）。然而，如果在 EOF1 采样点之前 EOP 没有发生，那么中继器将会处于 WFEOP 状态，在 EOF1 发生时，就会产生错误。注意 EOP 和 EOF1 可能同时发生，从而导致到 WFDSOP 的状态转变，但是不会发生错误。

当一个上游的连接在靠近时间片结束的时候发生，集线器中继器将会处于 WFEOP 状态，如果 EOF1 采样点在信息包结束之前发生（比如，EOP），集线器的状态就转变为等待第二个时间片结束（WFEOF2）的状态。集线器在以下两种情况下将会产生 WFEOF 状态：

- EOP 在 EOF1 和 EOF2 之间发生，然后集线器转变为 WFSOF 状态并且等待下

一个 SOP（它被假定为 SOF）；

- EOF2 采样点在 EOP 被检测到之前发生。集线器转变为 WFSOF 状态，并且禁止了初始化上游的端口，从而隔离了遭遇 LOA 或者串扰的端口。

一旦中继器处于 WFSOF 状态，它就保持那个状态，直到集线器检测到下一个时间片的开始，比如说，下一个下游的包的开始（DSOP）转变为 WFEOP 状态。

同步传输（无保证的发送）

同步传输的本质要求数据能够以持续的速度发送，所以它不支持重新传输，而且当执行和同步传输相关的事务处理时，不会发出握手包。

中断传输的错误恢复

中断传输使用握手和数据触发机制来验证数据的正确发送。如果一个给定的中断传输失败，那么在下一次服务期间，这次传输将会重新进行。

块传输错误恢复

块传输支持握手和数据触发检查来验证正确的数据发送。如果一个事务处理失败，那么将安排它在以后重试。

控制传输错误恢复

因为控制传输由三个阶段组成，它们代表一些和错误恢复相关的特殊问题。所有的三个阶段都支持握手，数据触发机制也用来验证数据的发送。数据触发从建立阶段开始，而且在整个状态阶段一直采用。

USB 供电分配

✎ 上一章

除了同步传输之外，所有的传输类型被设计为在 USB 上完成，甚至当错误状态被检测到时也是这样。上一章讨论了错误的类型，用于错误检测的机制，以及错误恢复机制。

✎ 本　章

本章讨论了 USB 电源分配，和总线供电设备相关的事件以及自供电设备的操作。本章还讨论了在检测和报告和电源相关的问题时主机软件所起的作用。

✎ 下一章

USB 设备支持电源保护，它可以进入挂起状态。下一章讨论电源管理的 USB 实现。

🕮 USB 的供电

所有的 USB 端口都可以为和他相连的设备供电。外围设备和集线器可以连接到一个给定的端口，并且使用可利用的数据线供电，或者实现自供电。本章对数据线供电和自供电的集线器和设备都作了讨论。

🕮 集线器

集线器的一个主要功能是分配和控制数据线功率。集线器可以从上游数据线处得到所需要的供电，也可以为它们的下游端口供电。集线器必须能够提供对电源的管理，而且出于安全考虑，电流必须限制在下游端口。

像所有的 USB 设备一样，集线器使用标识符来表示它的功能。一个集线器描述符的一个主要部分是和供电有关的事务。

电流预算

一个全速端口必须能够向和它连接的设备提供至少 5 个单位的电流（500 毫安）。自供电的集线器（包括根集线器）由本地电源供电，所以能够向每个端口提供最大的额定电流。然而，总线供电的集线器的用电只能来自上游的数据线，然后再把它分配到各个 USB 端口。所以对总线供电的 USB 集线器来说，这一限制严重地制约了和它相连的 USB 设备的用电量。一个端口的可用电流的最小数量是 100 毫安

集线器在它的配置描述符中指出它自己是总线供电的还是自供电的，参见表 9-1。阴影部分显示了一个位映射字段，它表示集线器是使用总线供电，自供电还是两者都可。

表 9-1　　在配置描述符里定义的集线器电源

字段编号	字段名	字段长度	字段取值	说　明
0	长度	1	数字	描述符的大小，以字节为单位。
1	描述符类型	1	常数	配置。
2	总体长度	2	数字	该配置返回的数据长度。包括所有描述符（配置、接口、端点、类别以及厂商自定义的描述符）加在一起的长度。
4	接口数	1	数字	这个配置所支持的接口数量。
5	配置值	1	数字	作为参数使用的一个数值，用它来进行配置的选择。
6	配置	1	索引	描述本配置的字符串描述符的索引。
7	属性	1	位图	配置特性 D7 总线供电 D6 自供电 D5 远程唤醒 D4..0 保留（置为 0） 某个设备在运行时使用的是总线供电还是自供电可以通过 Get Status 设备请求来获得。 如果一个设备配置支持远程唤醒功能，D5 就会被置为 1。
8	最大耗电量	1	毫安	在本配置中集线器消耗的最大数量的总线电流（数值以 2 毫安为单位）。

过流保护

USB 端口必须是对电流作限制的，这完全是出于安全考虑。从人身安全的角度出发，对一个端口的供电不能超过 5 安培。这个电流保护的规定可以用于一组端口，也可以单独对每一个端口作规定，只要电流满足 5 安培的限制就可以了。注意，一个总线供电的集线器只能从数据线那里获得电压，再把它分配到集线器各个的端口上。所以在这种情况下，不需要规定电流限制。

压降预算

可以通过数据线给 USB 外围设备供电。供电的集线器端口的电压不能超过直流 4.75V。然而在总线供电的集线器上的电压不能低于 4.40V。如图 9-1 所示。所以当 USB 设备处于它们的数据线的上游端点时，它必须保证工作在 4.40V 电压以上。

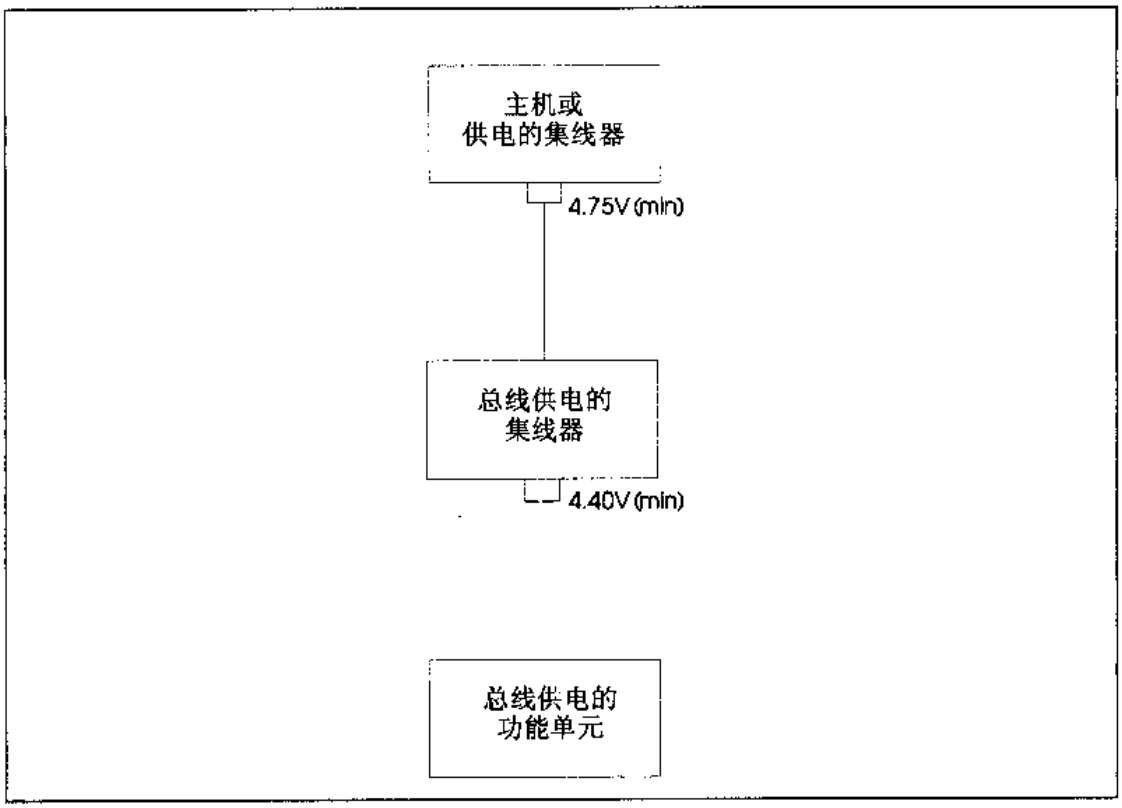


图 9-1　最小电缆电压和电压下降预算

电源开关

集线器采用以下几种方式中的一种给它的端口供电：

- 直接供电——没有开关；
- 所有端口共用开关；
- 每个端口有专用开关。

一个给定的集线器使用哪一种电源开关在它的 0 号端点的描述符中有说明。表 9-2 是集线器端点描述符的一部分。注意在第 3 个字段中的数据位 D0 和 D1 集线器支持的电源开关模式。

表 9-2　　集线器所支持的电源开关模式，它是由集线器的类别描述符定义的

字段编号	字段	字段大小	说　　明
0	DescLength	1	描述符的字节数，包括这个字节在内。
1	DescriptorType	1	描述符的类型。
2	NbrPorts	1	集线器支持的下游端口的数目。
3	HubCharacteristics	2	D1...D0 电源的开关模式。 00 组电源开关（所有的端口同时供电）。 01 独立的端口供电开关。 1X 没有电源开关（当集线器打开的时候，端口总是有电的，当集线器关闭的时候，端口就没有电了。 D15...D2 定义其他的集线器特性。

总线供电的集线器

总线供电的集线器的用电全部来自总线。因而它们不能把从上游端口来的所有的电流都分配给它的所有的功能单元，这些功能单元包括：集线器控制器、嵌入式的功能单元（如果有的话）以及所有的下游端口。这就造成了这样一个事实：为了保证集线器能够实现它的所有功能，总线供电的集线器必须连接到一个能够提供 500 毫安电流的上游端口。

在集线器配置过程中的电源

在给连接设备供电之前，必须首先配置集线器。主机软件必须能够访问集线器的描述符，因为描述符定义了它的功能（例如，端口功能、总线供电还是自供电）。在对它进行配置之前，USB 设备（包括集线器）使用的电流不能超过 100 毫安。这是因为连接到总线供电集线器的设备最多只能获得 100 毫安的电流。在配置集线器之前它不能取得 100 毫安的电流为它自己的嵌入式设备或端口供电。

连接到 500 毫安端口的总线供电的集线器

由于对一个总线供电的集线器来说，最大可获得的电流是 500 毫安。所以它能够

支持的最大端口数是 4。图 9-2 表示的是一个总线供电的集线器的模块图，表示对一个嵌入式功能单元和四个端口的供电分配情况。由于每个端口支持的电流至少为 100 毫安（一个单位电流），所以集线器控制器和嵌入式功能单元加起来所用的电流不能超过 100 毫安。

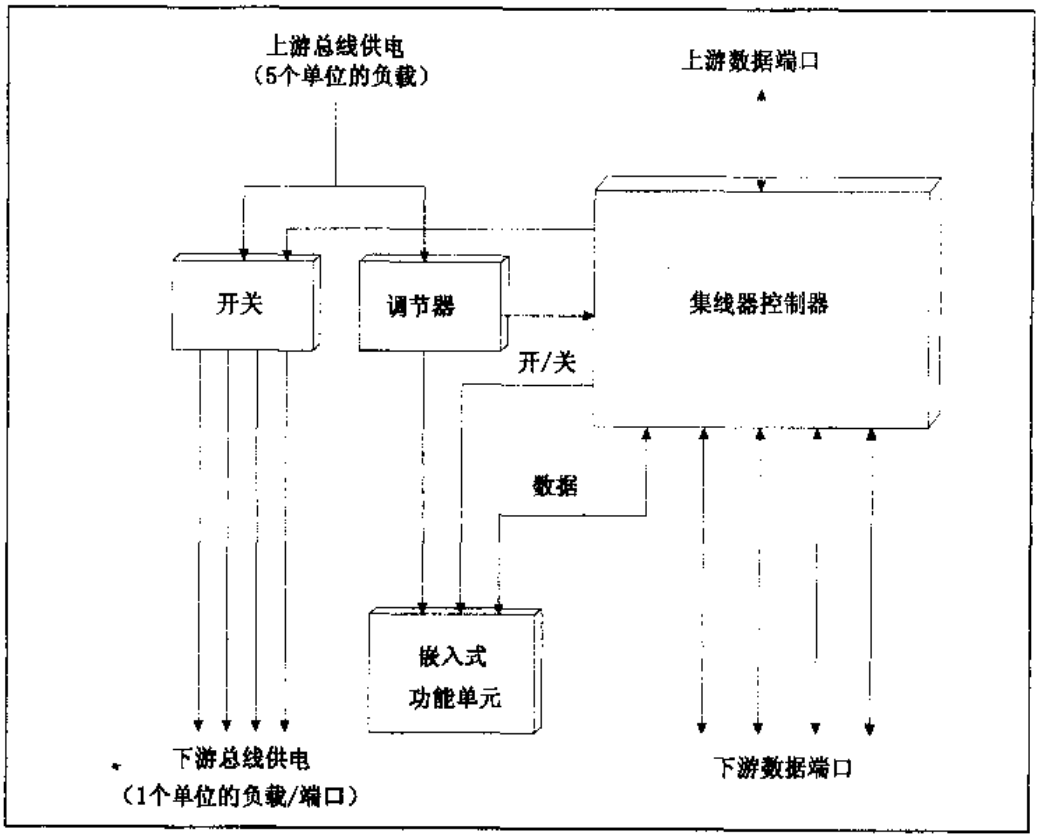


图 9-2　有嵌入式功能单元和 4 个端口的总线供电的集线器

和 100 毫安端口相连的总线供电的集线器

如果一个总线供电的集线器和另一个总线供电的集线器相连，而连接的端口最多仅仅能够提供 100 毫安的电流，那么这个可利用的电流必须用来给集线器控制器供电。这就允许配置软件对描述符进行访问以确定该设备对电源的需求。如果总线供电的集线器包含一个嵌入式的功能单元（比如一个复合设备），那么该单元可能和集线器控制器一起被供电。允许所有的电流被集线器控制器使用，而嵌入式功能单元的消耗电流不超过 100 毫安。

如果拉电流超过 100 毫安，嵌入式功能单元就必须是能够被单独关闭的。在这种方式下，配置软件可以通过读取设备描述符来确定所有设备的电源需求，并且仅仅激活那些当前电流能够支持的功能单元。在这种情况下，配置软件不能配置集线器，因为可用电流不足以给每一个功能单元供电。

和大于 100 毫安而小于 500 毫安端口相连的总线供电的集线器

如果嵌入式功能单元消耗电流＋集线器控制器消耗电流>100 毫安，总线供电的集线器必须为它的下游端口提供电源开关，这种情况就像上面要对嵌入式功能单元提供电源开关一样。由于两者相加使用的电源超过 100 毫安，所以在配置过程中，它们不能同时被激活。所以嵌入式功能单元要有自己单独的电源开关。这一需求也提供了别的灵活性：当和集线器相连的端口只能提供足够的电流激活集线器控制器和嵌入式功能单元的时候，那么和设备相连的那些端口就必须被关闭，因为没有足够的电流提供给它。

对于总线供电的集线器来说，规范说明书的陈述是："对下游端口的供电必须要有开关"，但是没有明确表示端口必须被单独关闭。端口供电可以单独关闭的优点在于可以根据当前的电流大小来选择开启的端口。

电流限制

由于总线供电的集线器仅有的供电来源是来自数据线，所以它最多只能有 500 毫安的电流，并且每个端口不能超过 5.0 安培的电流限制。如果从上游 USB 数据线处获得的总电流超过 5.0 安培，上游集线器的电流限制将会阻止这种过流状态的发生。

总线供电的集线器设备

总线供电的设备可以连接到一个全额端口上，或者连接到一个总线供电的集线器端口上，而后者最多只能提供 100 毫安的电流。配置可能因为供电不足而失败，这完全取决于 USB 设备是一个低功率设备还是高功率设备，或者它是连接到一个总线供电还是自供电的集线器上。

低功率设备

低功率设备消耗的电功率不超过一个单位。USB 的规范说明书指出：当电源首次加到 USB 设备上时，它消耗的电流必须低于 100 毫安。因为低功率设备消耗的电流绝对不会超过 100 毫安，所以不需要特别的设计考虑。而且，由于集线器端口可以提供的最小电流大于或等于低功率设备所要求的电流，所以并不存在限制。

高功率设备

高功率设备消耗的电流大于 100 毫安，但是不能从总线处拉出大于 500 毫安的电流。高功率设备可以设计为采用自己的本地电源供电，以消除非法的配置。

☑ 在配置过程中的供电

图 9-4 所示的是一个总线供电的设备连接到一个全额端口。当总线电源初次被应用到一个 USB 设备上时，它一定不能消耗超过 1 个单位的电功率，或者说 100 毫安，直到设备被配置为止。一旦设备被配置完成，设备就能够获得最大电流。

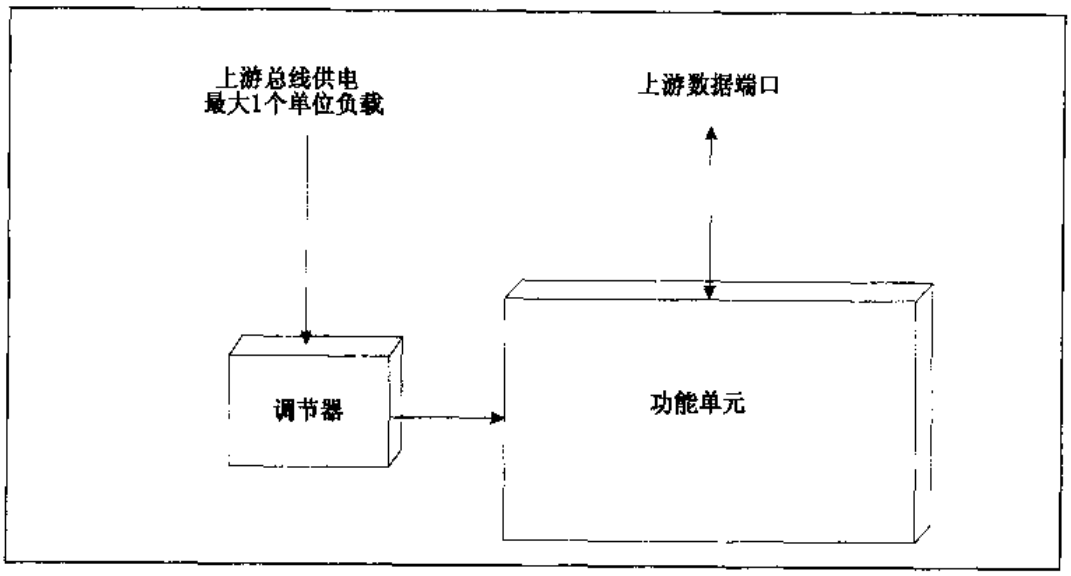


图 9-3　低功率 USB 功能单元

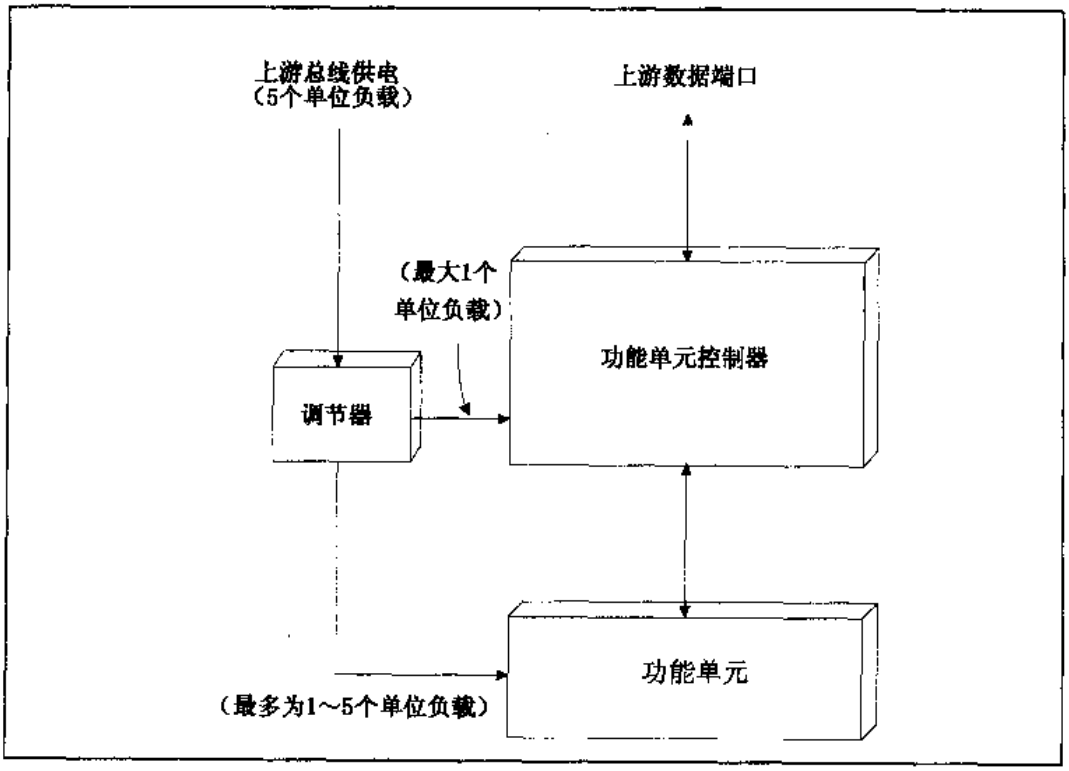


图 9-4　高功率/总线供电的功能单元

☑ 端口供电不足

如果一个高功率的设备连接到一个总线供电的集线器上，而且完全由总线供电，那么该设备很可能因为没有足够的电流使用而不能正常工作。主机软件必须检测到这个高功率设备的功能单元所要求的用电情况，这是通过读取它的配置描述符来获得的。如果设备的电流需求超过了端口可提供的电流，那么设备就不应该被配置，而且这一情况应该向用户报告。表 9-3 就是配置描述符的一部分，它定义了设备完成配置以后的最大的总线功率消耗（阴影部分）。

注意，一个高功率设备能够提供一个可选的电功率配置，该配置把全部的功率消耗减为 100 毫安，从而确保它能够适合所有的 USB 端口。然而，这种低功率的配置可能会导致设备性能的降低，当然这种降低是相对于它的全功率配置而言的。

表 9-3　　定义在设备配置描述符里面的最大供电量

段编号	字段名	字段大小	字段取值	说　明
7	属性	1	位图	配置特性 D7 总线供电 D6 自供电 D5 远程唤醒 D4..0 保留（置为 0） 某个设备在运行时使用的是总线供电还是自供电可以通过 Get Status 设备请求来获得。 如果一个设备配置支持远程唤醒功能，D5 就会被置为 1。
8	最大耗电量	1	毫安	在这种特殊的配置下，而且 USB 设备是完全可操作的情况，该设备的最大耗电量，以两毫安为单位（例如，50=100 毫安）。 注意：设备配置报告这个配置是总线供电的还是自供电的。 设备状态返回报告设备是否是自供电的。如果一个设备和它的外部电源断开连接，它就会更新它的设备状态，指出这个设备不再是自供电的了。 当一个设备失去自己独立的供电电源的时候，它从总线处能够获得的用电量不能超过它的配置值所指出的耗电量。

🕮 自供电的集线器

集线器可能被集成到其他的 USB 设备内，这些设备由自己供电，例如显示器或者独立的集线器都能够通过本地电源自己供电。这种集线器的明显的优点是它能够为它的每一个下游 USB 端口提供全部 500 毫安的电流。但是它也有一些显著的缺点，相对于总线供电的集线器来说，它的复杂性和成本都比较高。自供电的设备可能还有电流限制（每个端口的电流不能超过 5.0 安培），这是规范说明书中的规定，它完全是出于安全的考虑。

自供电的集线器也可以集成可选的特性，例如给每个端口加入 LED，指出哪个端口被加电了或哪个端口被激活了。

在配置过程中的供电

图 9-5 是一个基本的自供电集线器的模块图。集线器必须在它的 D+数据线上包含一个 1.5 千欧的上拉电阻，以次来标识它自己是一个全速设备。系统已经检测到了和它连接的设备，所以它必须访问设备的描述符来决定设备的类型。这就要求它对集线器控制器的控制端点进行访问，以读取集线器的描述符。

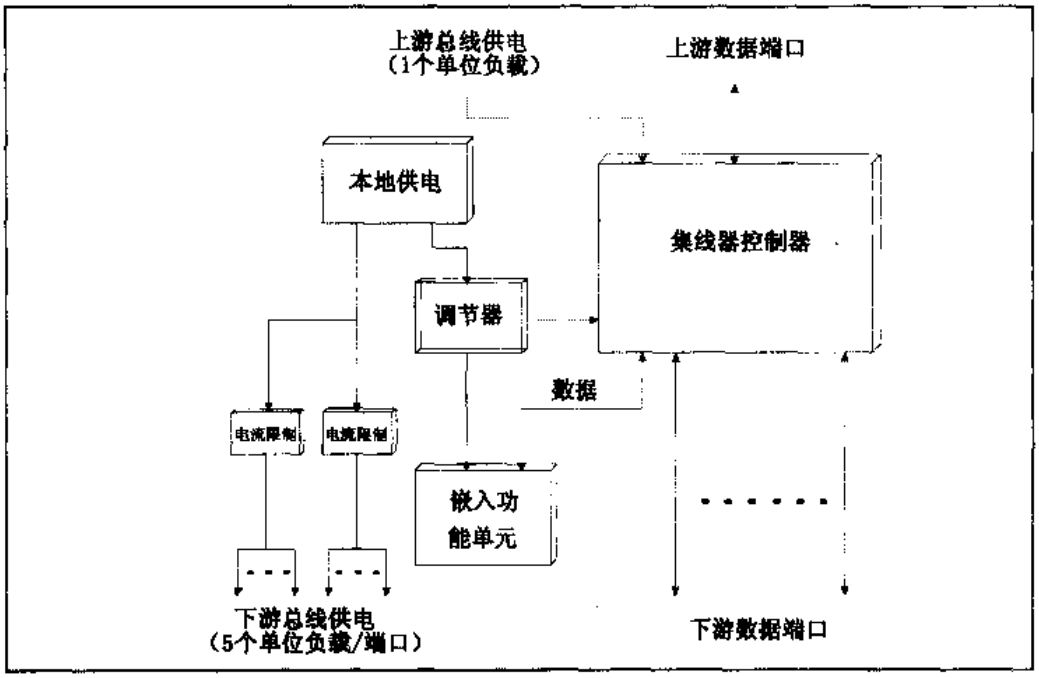


图 9-5　有嵌入式功能单元的自供电的集线器

☑ 本地供电总线接口

自供电集线器可以使用本地电源单独为它的主控制器供电。结果是：如果集线器不使用交流电源，那么 D+线上的 1.5 千欧的电阻可能没有 Vcc 可用，而当设备连接时，系统可能检测不到它。这一点可能使最终用户感到很迷惑，他们希望所有的 USB 一连接上以后就能自动地开始工作。

上拉电阻也有可能是由总线供电，但是集线器控制器是由本地电源供电的。在这种情况下，设备连接将会被检测到，但是主机试图读取设备描述符则不会成功。特别是主机产生的对缺省端点的访问不会从集线器处得到响应。主机将会检测到一个错误状态，但是不知道是什么引起了这个错误。

在图 9-5 中的点线表示用本地电源供电的总线接口的供电选项。它介于本地供电

和集线器控制器电子设备之间。

☑ 混合供电设备

自供电的集线器可以通过 USB 数据线为整个总线接口和集线器控制器的功能单元供电，使它在连接后能够被检测到并且读取它的描述符。然而，嵌入式功能单元和每个端口的供电则来自本地电源，所以它被称为是一个混合供电的集线器，和别的 USB 设备一样，在配置过程中，它不能从数据线处拉取超过 100 毫安的电流。

混合实现的优点在于：能够区分出一个未被供电但是已经连接了的集线器和一个断开连接的集线器，或者一个虽然连接但是明显已经损坏了的集线器。

在图 9-5 中，标识为上游供电的点划线表示集线器控制器和它的总线供电的供电操作。

电流限制

图 9-5 还表示了一个组形式下电流限制的实现方式。由于电流限制是每个端口 5.0 安培，所以当拉电流限制超过 5.0 安培时，限流器就必须跳开。例如，如果一个自供电的集线器支持用一个限流器对 10 个集线器端口进行管理，那么在每一个端口上的合法电流（500 毫安）相加将会超过最大电流限制。所以，一个 10 个端口的集线器要用两个限流器来实现，每一个限流器为 5 个端口的端口组提供保护。

自供电设备

自供电设备必须在它的 D+线上提供一个 1.5 千欧的电阻，表示它是一个全速设备。注意，一个自供电设备可能是一个低速设备，然而，作者很怀疑这样的设备是否会出现。

在配置期间的供电

一个设备的总线接口可能由总线电源供电，或者单独由本地电源供电。这影响到当设备连接好以后，它是否能够被检测到，以及它的描述符能否被读取。下面的部分描述了每一种情况。

☑ 本地供电的总线接口

一个自供电的集线器可能单独从本地电源给它的主控制器供电。这样以后的结果是：如果集线器不使用交流电源，那么 D+线上的 1.5 千欧的电阻可能没有 Vcc 可用，而当设备连接时，系统可能检测不到它。

上拉电阻也有可能是由总线供电，但是集线器控制器是由本地电源供电的。在这种情况下，可以检测到设备连接，但是当主机试图读取设备描述符时则不会成功。特别是，主机对缺省端点采取的访问不会从集线器处得到响应。主机将会检测到一个错误状态，但是不知道是什么引起了这个错误。

在图 9-6 中的点线表示用本地电源供电的总线接口的供电操作。它介于本地供电和集线器控制器电子设备之间。

☑ 混合供电设备

一个自供电的设备可能为它的功能单元控制器使用总线供电，但是拉电流限制还是 100 毫安。在图 9-6 中的点线表示是否使用总线供电的功能是可选的。从总线处获得的电源使它有可能检测到集线器的连接并且读取它的描述符。然而，功能单元的供电是从本地获得的。所以它被称为混合供电设备。

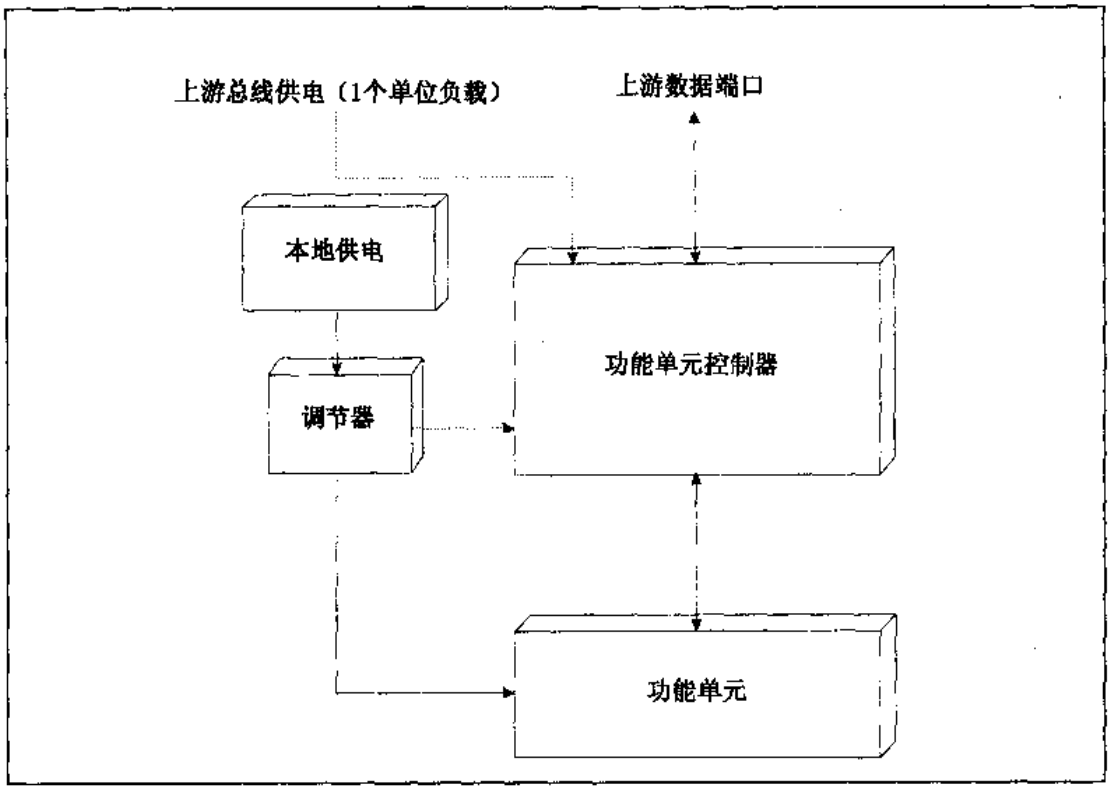


图 9-6　自供电设备

混合实现的优点在于：能够区分出一个没有被供电的设备和一个断开连接的设备，或者一个虽然连接但是明显已经损坏了的设备。

USB 电源管理

✎ 上一章

上一章讨论了 USB 电源分配，还有和总线供电设备相关的事务，以及自供电设备的操作。上一章还讨论了主机软件在检测和报告与供电相关的问题时候的作用。

✎ 本　章

USB 设备通过进入挂起状态支持省电模式，本章讨论了电源管理的 USB 实现。

✎ 下一章

下一章对配置过程进行了一个概述，定义和讨论了每一个主要的步骤。

🕮 供电保持——挂起

供电保持是通过挂起的办法来实现的，这是一种软件控制的办法。USB 支持两种类型的挂起方式

- 全部挂起——所有的 USB 设备进入挂起状态；
- 选择挂起——被选择的设备进入挂起状态。

当设备进入挂起状态，它消耗的电流不能超过 500 微安，在 3 毫秒没有总线行为之后，设备将会进入挂起状态。通常，当设备在每一个 1 毫秒的时间片的开始处收到一个 SOF 令牌后，它就不会进入挂起状态，即使除此之外没有其它的 USB 通信发生也是如此。

然而，低速设备仅仅能够看到低速通信，意味着它们不会看见 SOF 包，也不能看见全速的事务处理。所以集线器必须在所有的低速端口上发出一个空闲信号给 K 状态事务处理，该信号的时间间隔必须少于 3 毫秒，以阻止低速设备进入挂起状态。

设备对挂起的响应

当设备进入挂起状态后，它必须保持它的状态，而且消耗的电流不能超过 500 微安。这种限制包括与在 D－和 D＋线上的上拉和下拉电阻相关的拉电流。

某些设备可能需要唤醒系统，以响应一个外部事件。例如，一个调制解调器的功能单元可能被设计为能够把一个系统从挂起状态唤醒。如果调制解调器从内部线接到一个响铃指示，那么就有必要通知系统软件让调制解调器注意。

集线器对挂起的响应

当一个集线器在它的上游端口上连续超过 3 毫秒没有活动，那么它就会进入挂起状态。由于集线器在 3 毫秒内没有检测到总线活动，所以根据定义，连续 3 毫秒没有活动被广播到集线器的任何一个下游端口。所有的下游端口，包括集线器本身，将会几乎同时检测到挂起状态。

检测到挂起状态后，集线器采取以下的行动：

- 使它们的中继器进入等待包开始的状态（WFSOP）；
- 闲置所有的输出设备；
- 保持所有的控制位和状态位的静态值；
- 为所有的下游端口保持当前状态。

所有的内部时钟都被停下来，而且集线器功能单元的用电消耗量也被减到最小值。

由于连接到下游端口的设备也进入了挂起状态，它们的拉电流每个最多也能够到 500 微安，这意味着在挂起过程中，集线器本身可以从上游端口拉取超过 500 微安的电流。此外，在挂起状态下，集线器必须能够为下游端口提供定义的最大电流，以支持远程唤醒功能。就是说，一个设备可以在挂起期间驱动总线，从而唤醒系统，这将在本章的后面讨论。

全局挂起

全局挂起状态就是自上向下使整个 USB 设备网进入挂起状态。这就给 USB 设备提供了最小的电源消耗。当 USB 上有一个过长的动作的时候，主机一般启动全局的挂起状态。

启动全局挂起

当所有的来自根集线器的下游通信都完成之后，全局挂起状态就被启动了。这是在软件控制之下完成的，通过向根集线器的控制端点发送一个全局挂起请求。所有的 USB 设备（集线器和功能单元）在 3.0 毫秒没有活动后将会自动进入挂起状态。

从全局挂起状态恢复

当设备从总线上检测到恢复信号之后，它就会从挂起状态恢复过来。恢复信号是由非空闲状态发出的（K 状态）。

恢复可以以下面的方式启动：

- 由根集线器初始化，在所有的下游数据线段引起恢复信号；
- 由任何一个下游数据线段上的设备初始化，它被连接到一个激活的端口上，引起集线器把恢复信号回送给那个端口，其它的被激活的下游端口，还有向上回到根集线器端口；
- 通过设备连接激活；
- 通过设备断开连接激活；
- 通过复位信号激活，它引起所有设备被重新配置。

下面的部分详细描述了每一种类型的恢复中集线器和设备的操作。

☑ 主机启动的恢复

主机软件可以启动 USB 的唤醒功能，它是通过向根设备发送一个恢复请求来完成的。根集线器通过发送一个恢复信号来恢复所有被激活的集线器端口，如图 10-1 所示。

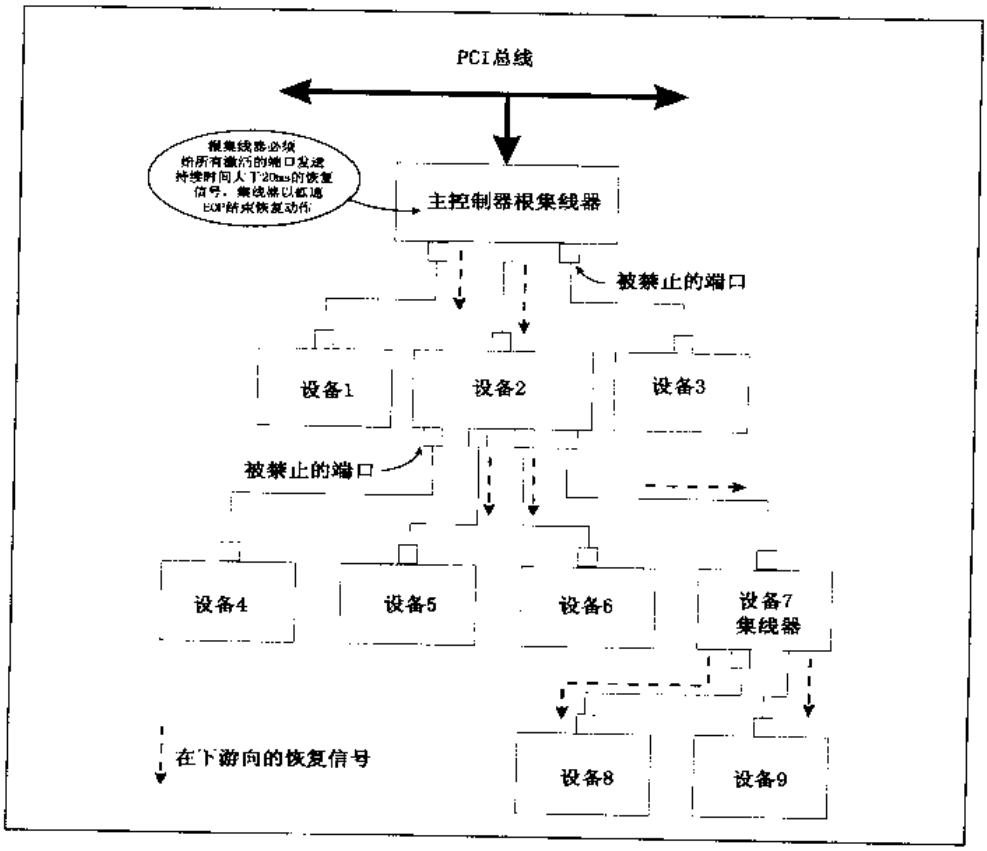


图 10-1　主机初始化恢复

恢复信号必须至少维持 20 毫秒，以给每一个连接的设备充足的时间从挂起状态恢复，并为接收事务处理做好准备。根集线器结束恢复信号是通过持续两个低速位时间驱动一个 EOP 来完成的。

收到 20 毫秒恢复信号的下游集线器必须也能够为所有激活的下游端口发出恢复信号，如图 10-1 所示。

☑ 从设备处的远程唤醒

一个处于挂起状态的设备可能对一个外部事件作出响应，只要通过 USB 向系统发出一个唤醒信号即可。恢复信号由设备初始化，方法是在总线上驱动一个 K 状态（非空闲状态）。集线器采取的行动是：

- 向上游端口发出恢复信号；
- 向所有激活的下游端口发出激活信号；
- 向始发设备发回恢复信号。

图 10-2 表示了事件的序列，并描述了和设备启动恢复相关事件的顺序。下面就是具体采取的步骤：

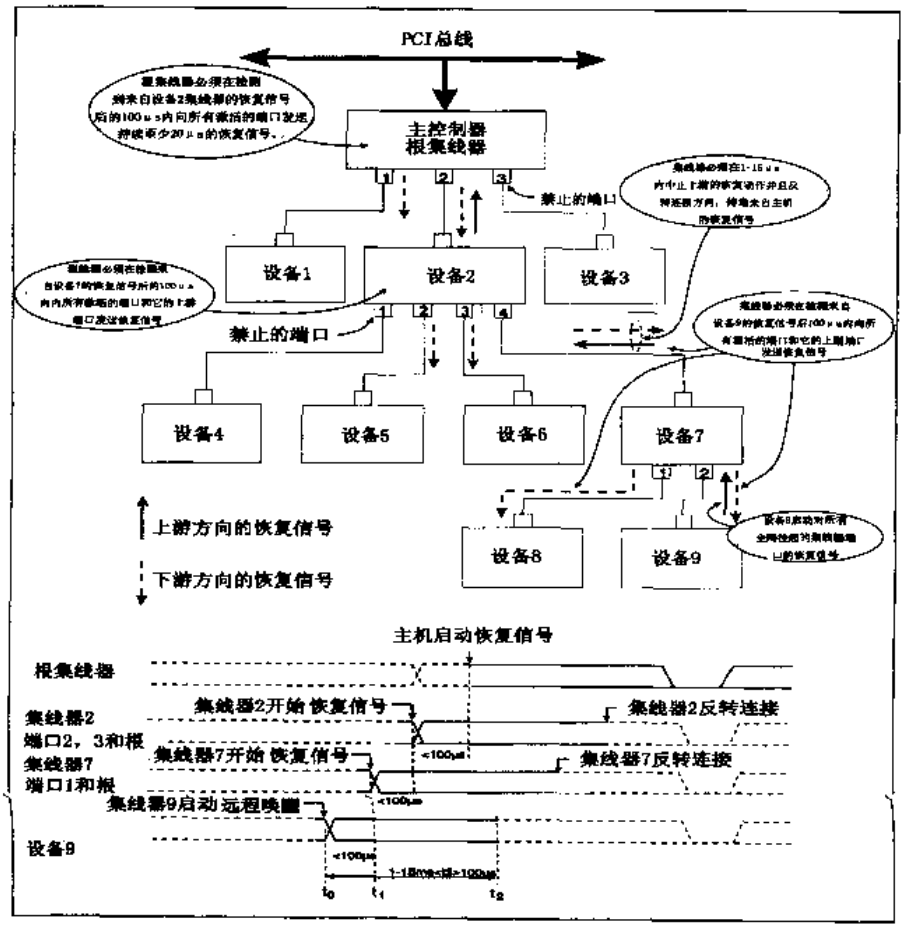


图 10-2　图为从目标设备唤醒的全局性恢复

（1）设备向集线器端口发出一个 K 状态信号。

（2）端口检测到恢复信号（t_0）。

（3）恢复信号在收到恢复信号之后 100 微秒的时间内（t_1）被集线器 7 广播到他的上游端口以及所有已经激活的下游端口。

（4）集线器 2 在从集线器 7 处收到恢复信号后的 100 微秒内，对所有它的激活的端口发出恢复信号。

（5）在 t_0 之后的 1-15 毫秒之内（但是不早于 100 微秒），集线器 7 停止在上游方向驱动恢复信号，并且使连接反向，这被称为 t_0。

（6）在从集线器 7 处收到恢复信号后的 10 毫秒之内（但是不早于 50 微秒），集线器 2 停止在上游方向驱动恢复信号，并且使连接反向。

（7）当根集线器检测到恢复信号后，它在下游方向上的所有端口上初始化 20 毫秒的恢复信号，包括原来的端口。

（8）根集线器通过驱动两个低速 EOP 结束恢复信号。

注意在 t_0 和 t_2 期间，主机可能已经开始从上游方向驱动恢复信号，此时下面的集线器还在驱动恢复信号。由于把一个总线段的两个端点驱动到同样的状态是可以的，所以不会有什么问题。

☑ 由于设备连接或断开连接产生的远程唤醒

当集线器检测到一个设备已经被连接或者断开连接，它就给系统发信号通知，这种方式类似于远程唤醒功能。

选择性挂起

允许使用软件的办法使某个设备或总线上的某一段设备挂起。当集线器的某个端口进入挂起状态时，选择性挂起由软件进行启动。在挂起状态下的端口阻塞所有的下游数据流，允许设备向下检测 3.0 毫米的不活动区域，然后进入不活动状态。

启动选择挂起

主机软件可以挂起一个单独的设备或者位于一个特定的总线段上的一组设备。选择挂起通过一个控制传输实现，它发送一个 SetPortFeature 的请求（PORT_SUSPEND）参见设置/清除端口特征部分。收到请求的集线器将把指定的端口置于挂起状态。注意再一次信息包的传输过程中，集线器是不会把某一个端口挂起的。在信息包传输结束之后，端口就被挂起。

当设备处于挂起状态时，除了端口复位请求之外端口不能向相连的设备传输数据流。类似的，集线器不会在下游端口向上游方向传递活动。由于总线上的数据流被阻止

传递，所有位于挂起端口下方的总线数据线上的设备将不会检测到总线上的活动，在 3 毫秒以后就进入挂起状态。

从选择性挂起状态恢复

选择性恢复可以由主机系统启动，或者由被挂起的远程设备启动。这两种形式的选择性挂起恢复如下文所描述。

☑ 主机初始化选择性挂起恢复

主机初始化选择性恢复是通过一个 ClearPortFeature(PORT_SUSPEND)请求传递给被挂起的集线器端口。这引起端口向连接到端口的设备发送恢复通知信号。挂起通知信号必须持续至少 20 毫秒。然后是一个低速 EOP 信号。在集线器内设置一个状态位指出恢复已经完成，设备已经做好了被访问的准备。状态位只有在低速 EOP 信号被发送之后 3 毫秒才能够被设置。这个延迟确保了在被作为事务处理的目标设备被访问之前有时间同步它们的时间片计数器。然而恢复的设备将在 3 毫秒的时间内看到总线活动。

☑ 从设备那里选择性唤醒

选择性挂起支持远程唤醒功能，就像全局挂起一样。然而在选择性挂起的环境下，很可能别的设备被唤醒，并且接收到总线活动。全局挂起和选择性挂起的本质区别在于集线器。设备知道的仅仅是它在总线上检测到 3 毫秒的不活动状态，然后它就转入挂起状态，然而集线器本身和集线器的其他端口并没有挂起，还在接收总线的活动。所以当选择性挂起设备发出恢复通知的时候，集线器一定不能向上传递恢复信号，也不能传递给其他的端口。实质上，远程唤醒是属于选择性挂起设备和被挂起的集线器端口之间的事件，和其他设备与端口无关。

当选择性挂起的集线器端口从设备处监测到远程恢复信号时，它将作出如下的响应：

（1）从设备处接收到恢复信号的 100 微秒内把恢复信号返回给设备。

（2）继续发送恢复信号，这一过程将至少持续 20 毫秒。

（3）以一个低速 EOP 信号中止恢复信号。

（4）在 EOP 信号之后的 3 毫秒设置恢复完成状态位。

总之，选择恢复单独包括被挂起的集线器端口，以及和目标设备相连的集线器端口。别的设备都没有被通知唤醒，因为它们当前可能正在进行事务处理，该事务处理在 USB 上被广播。恢复信号也不应该被发送到选择性挂起的设备。

由于在这种状态下的选择性挂起还没有被主机启动，主机必须基于某个规则检查状态（对集线器进行查询）以确定是否发生了恢复。（相关章节标题为“获取端口状态请求”的部分定义了用于检查端口状态的机制）

如果在端口处的唤醒设备连接或者断开连接，那么集线器将会采取相同的行为，只是恢复信号并没有发送到初始化端口。取而代之的是：端口的输出缓冲区进入高阻状态，端口状态位被改变，反映设备连接或者断开连接。

当集线器被挂起时的选择性挂起

在一个单独的集线器端口被挂起之后，主机软件顺序挂起整个集线器（通过全局或者选择性挂起）。要求集线器采取行动时可能发生 3 种状态：

- 当前连接到被挂起端口的设备发出恢复通知信号；
- 设备被连接到选择性挂起的端口；
- 设备从选择性挂起的端口断开连接。

☑ 设备发出恢复信号

如果连接到被挂起端口的设备发出恢复通知信号，挂起的集线器将采取下面的行动，如图 10-4 所示。

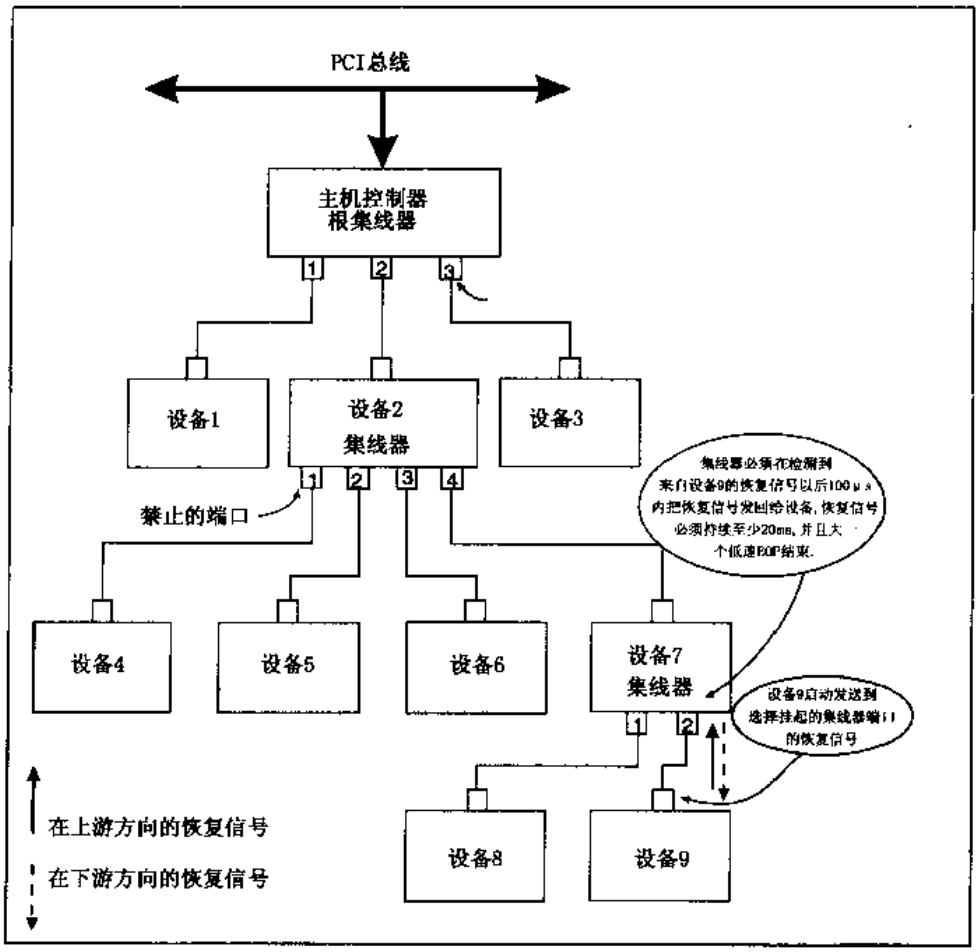


图 10-3　目标设备的选择性恢复

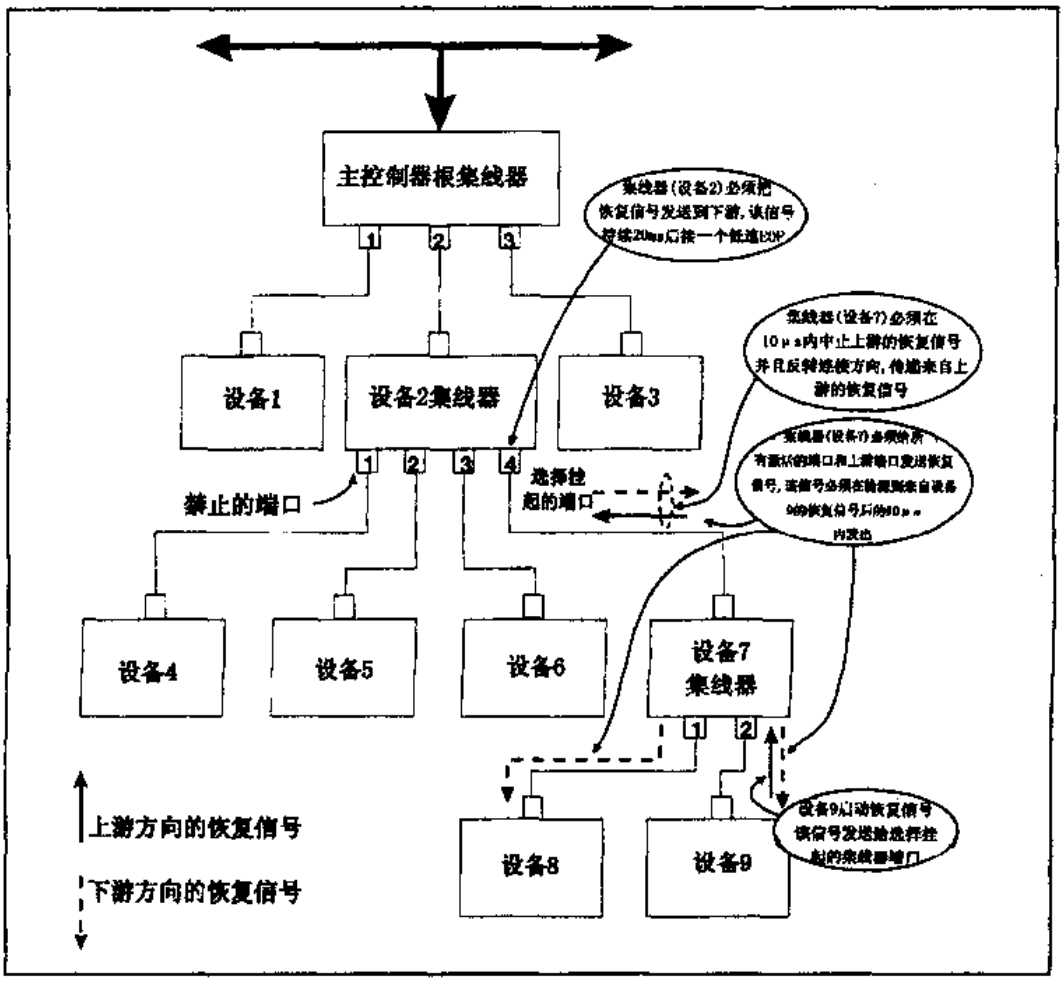


图 10-4　对挂起集线器的设备初始化选择性恢复

（1）恢复信号被发送到所有的下游端口，那些端口被激活，然后回到被挂起的端口（必须在从设备处接收到恢复信号后的 100 微秒的时间内得到恢复通知，并且必须继续发出恢复信号，这一过程至少持续 20 毫秒。注意选择性挂起端口继续保持他们的选择性挂起状态。

（2）恢复被信号也被发送到根集线器，这一过程也必须在 100 微秒内完成。然而，恢复必须被释放，并且在 10 毫秒内完成连接的反向。

（3）上游端口将向集线器发回恢复信号，这一过程至少持续 20 毫秒。这个恢复信号被返回给所有激活的下游端口，包括一开始就被唤醒的设备。

（4）根端口会在恢复信号结束的时候驱动一个低速的 EOP 信号，集线器进入等待状态，等待时间片状态(WFSOF)的开始.

（5）集线器设置状态指出选择性的挂起端口目前已经被唤醒，这个位必须在 3 毫秒之后被置 1，这样才能给集线器的时间片定时器足够的时间和主机保持同步。

要注意的是上游端口可以因为全局挂起被挂起，也可以被选择性的挂起，在全局挂起的情况下，恢复信号将被向上反映到根集线器上去。如果上游端口被选择性挂起，如

图 10-4 所示，那么恢复信号将被返回到下游的集线器，但是不会再传递到其他的集线器端口。标为设备 2 的集线器还没有被挂起，但是它的 4 号端口已经被选择性挂起。当恢复信号在 4 号端口被检测到时，集线器将会向设备 7 返回恢复信号，而不会采取进一步的动作。

☑ 端口接受连接或者断开连接

如果选择性的挂起端口接受到一个单端 0（SE0）状态到空闲状态的转变（设备连接）或者一个 SE0 到空闲状态的转变（设备断开连接），那么恢复信号就不会被发回到选择性挂起端口，而且端口的输出缓冲区也会被设置为高阻状态，集线器将会设置状态位，指出连接或断开连接的事件，这时端口也就不再是挂起状态了。

🕮 在全局挂起之后的选择性挂起

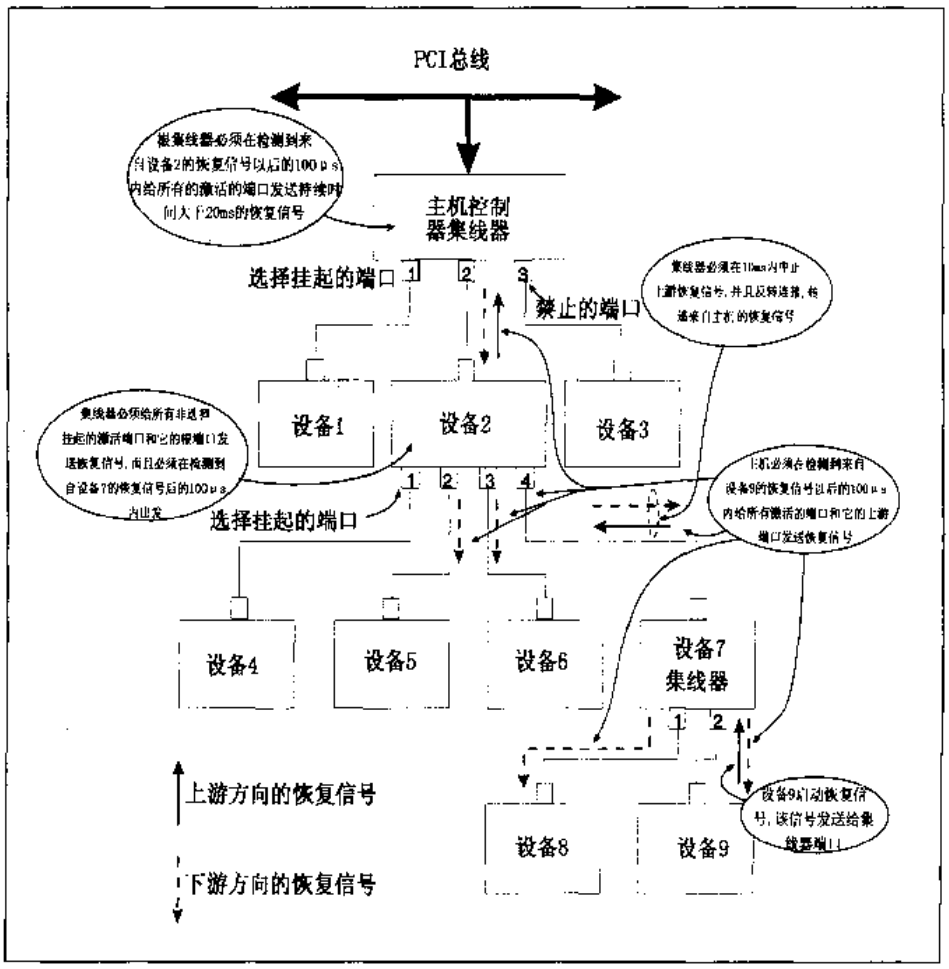


图 10-5 选择性恢复和全局挂起

考虑这种情况：在再一个全局挂起之后，有几个设备被选择性挂起，图 10-5 表示

了这样的一个例子。端口 1 位于根集线器上，端口 1 还连接设备 2，连接设备 7 的端口被选择性挂起。因为全局挂起，所以所有别的设备都处于挂起状态下。在本例中，当设备 9 发出远程唤醒后，集线器（设备 7）就向前传递恢复信号给它的其他端口，以及它的根端口。设备 2 认出恢复信号，并向除端口 1 之外的所有端口发送恢复信号。根集线器在端口 2 上检测到恢复信号，并把恢复信号返回给设备 2，而不是端口 1，因为端口 1 被选择性挂起，也不会传给端口 3，因为它被禁止了。

一旦系统完成了全局恢复处理，主机就可能在全局挂起之前选择性的唤醒那些被选择性挂起的设备。

通过复位恢复

当一个被挂起的集线器在它的上游端口上检测到一个复位信号的时候（大于 2.5 微秒的 SE0），它就必须启动它的唤醒序列。这时候就必须唤醒集线器，并且从发出复位信号开始的 10 毫秒内要完成复位动作。在复位动作完成之后，集线器的控制器必须处于以下状态。

- 缺省地址时 0；
- 控制位设置为缺省值；
- 集线器的中继器处于 WFSOP 状态；
- 所有的下游端口处于断电状态（有供电开关的集线器）；
- 所有的下游端口处于断开状态（无供电开关开关集线器）。

包含供电开关端口的总线不能保证主机启动的复位能够传递到所有的下游端口。这是因为支持电源开关的上游集线器在复位后进入了断电状态。接着在复位过程完成之前，下游的电源可能从总线供电的集线器和设备处那里断开连接。然而，如果给设备供电的电源断开足够长时间的话，这个断电的设备将会被复位，。注意，一旦执行了复位操作之后，所有的设备都需要重新进行配置。

唤醒后的集线器时间片定时

当处于挂起状态下的集线器关闭了它们的时间片定时机制以后，就可以减少电源的消耗。当恢复动作发生之后，时间片定时机制必须被重新启动，并且留出时间和主机达到同步，同步是依靠接受 SOF 包来实现的。在集线器回到唤醒状态和第一个 SOF 包离开集线器之间的时间间隔是不能加以检测的。通常，集线器在时间片结束的时候检查总线的状态，以确定一个设备是否处于非空闲状态。然而，在离开挂起状态之后，集线器没有时间片定时的概念，而且也不能检测到这些形式的总线错误。

为了阻止设备串扰或者总线挂起引起的 LOA，在集线器收到一个 SOF 包之前，它绝对不可以把总线动作向上游方向传递。上游的子数据流被一个集线器阻塞，这时它就

进入等待时间片开始的状态（WFSOF）。集线器离开挂起状态，并进入了 WFSOF 状态，从而也就阻塞了上游总线的数据流通。当它们在被唤醒后收到第一个 SOF 包的时候，集线器从 WFSOF 转向 WFSOP 状态。参见“集线器中继器状态机制”一部分。

注意，即使当总线处于 WFSOF 状态时，下游的数据流通也是由下游设备进行处理的。如果下游设备有自己的地址的话，它们就很可能通过把信息包发回主机进行响应。由于包的传输被集线器阻塞，主机将会检测到超时，因为没有来自设备处的响应。所以规范说明书建议主机不要对位于一个总线段上的任何设备进行编址，如果那个总线段刚被恢复的话，这就可以避免不必要的总线超时。

第三部分　USB 配置

配 置 处 理

✎ 上一章

USB 设备支持省电模式，这是通过它们进入挂起状态实现的。上一章讨论了电源管理的 USB 实现。

✎ 本 章

本章对将配置处理作一个概述。包含在 USB 设备配置中的每一个主要步骤都加以定义，并且对它们进行了讨论。

✎ 下一章

下一章详细描述了配置 USB 集线器的处理细节。

🕮 概述

主机软件负责检测和配置所有和根集线器相连的设备，标识和配置一个 USB 设备的过程通常被称为 USB 设备枚举。设备的枚举从根集线器开始，每个集线器端口在被枚举到的时候都必须上电。当上电的时候，集线器确定是否有一个设备连接到端口上，或者已经连接上的设备是全速设备还是低速设备。如果设备存在的话，在根集线器上的状态位将被设置，以反映设备的连接情况。软件识别出一个设备已经被连接到集线器的端口上，激活这个端口，把 USB 设备复位，分配一个唯一的地址，并且完成这个配置。这个操作在每一个端口上都被执行，直到连接到根集线器上的所有设备都被配置完成。

本章的讨论假设软件已经初始化了 USB 的主控制器，而且它能够产生 USB 事务处理。

不同的操作系统会定义不同的软件组件，这些软件组件包含在 USB 的设备配置中，

而且将会按照它们自己的顺序执行配置。下面的讨论包含了配置过程中主要的步骤。下面的列表列出了当配置一个和根集线器端口相连的设备的时候，主机软件和根集线器的行为。

- 主机配置根集线器；
- 主机请求电源应用到端口上；
- 集线器检测设备连接；
- 集线器设置状态改变；
- 主机查询集线器，并且标记出某个设备已经连接上了；
- 主机给集线器发送端口激活信号；
- 主机给端口/USB 设备发送复位信号（最少 10 毫秒）；
- 端口现在已经激活了，可以获得 100 毫安的电流；
- USB 设备回答以缺省的地址(0)；
- 主机要求设备的控制端点在地址 0，以决定缺省的管道所支持的最大数据量；
- 主机为 USB 设备安排唯一的地址；
- 主机从设备描述符哪里读取配置信息，并依照它进行配置；
- 主机验证设备所需要的 USB 资源需求是否可以得到满足。

主机向 USB 设备发送一个配置值，指出它将会怎样被使用。当设备收到配置值的时候，设备就使用它所描述的特征。该设备现在已经为客户端软件的访问做好了准备，而且能够获取配置中所指出的总线供电量。

上面所执行的操作主要是通过控制事务处理来实现的。软件使用集线器内的控制端点（端点 0）和每一个设备存取它们的描述符，并且控制其他设备所支持的特征。（集线器和设备的请求在第 13 章和第 15 章分别有详细的描述）

下面的部分提供了一个对配置过程的详细描述。关于集线器和设备配置的更多的细节可以参见后续章节。

配置模型

图 11-1 所示的模型就是基于上面的配置过程的，它标出了那些概念上的软件元素。

在系统的引导和初始化的过程中，所有的 USB 设备和集线器将会被检测和加以配置。接下来被初始化的设备可以被断开连接，新的设备也可以连接上去。集线器客户应用程序周期性地查询所有的 USB 集线器，来检测设备连接或者断开连接。如果因为一个新的设备连接而导致状态改变，那么就会引起新的配置过程。

集线器的客户应用程序必须通知客户软件，新的设备已经被连接上了。作为响应，配置软件必须标识出它识别出来的新的设备，这是通过软件读取设备的标准描述符，用缺省的控制管道实现的。从描述符那里读取的信息指出了设备所需要的 USB 资源。（例如：总线带宽和供电需求）。这些描述符必须由配置软件做出解释。

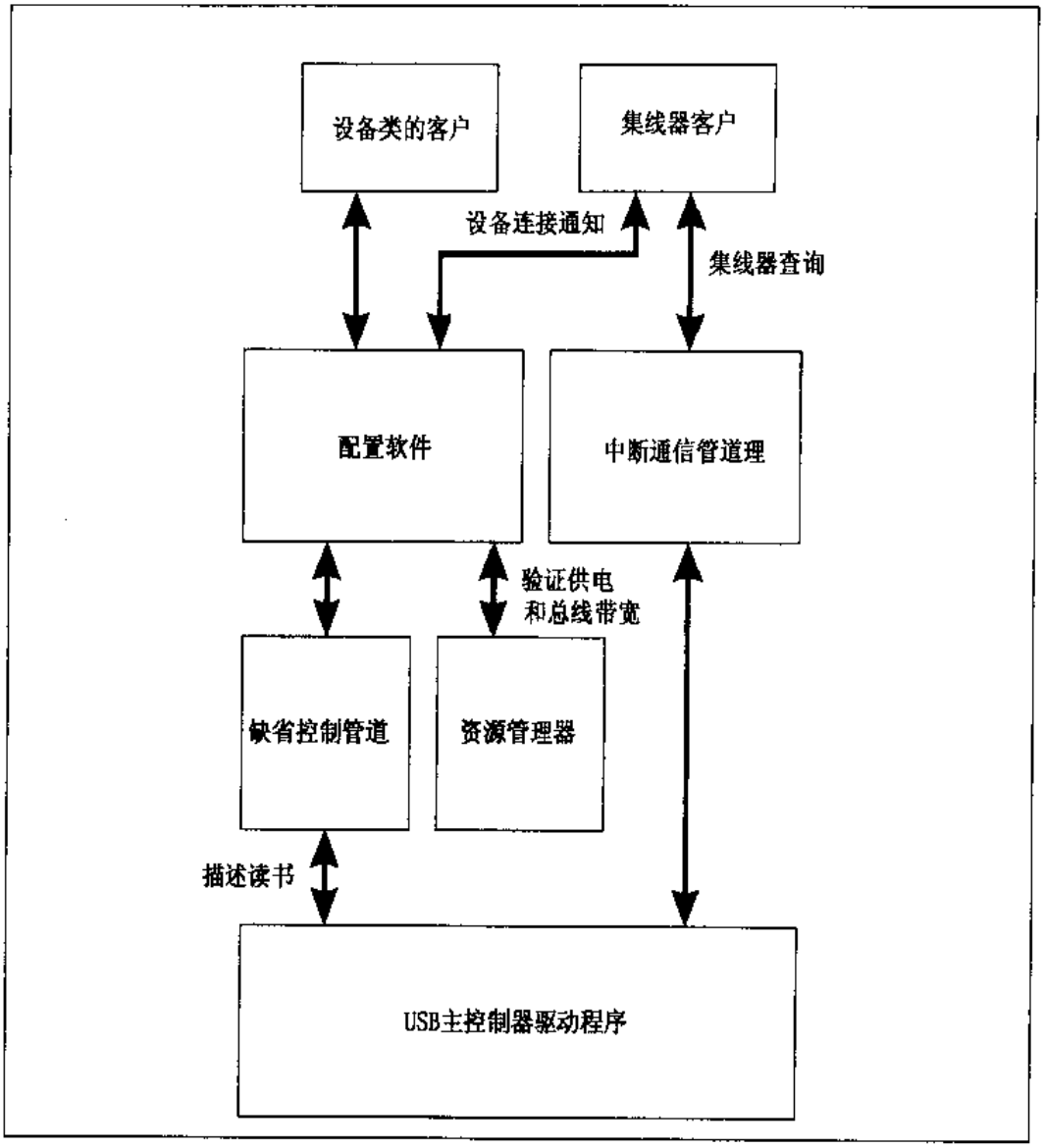


图 11-1　配置软件模型

一个设备可以支持多于一个的配置，这需要软件进行配置选择。设备的每一种配置都提供了可选的替代方案。例如，一个设备既可以支持高电压配置，也可以支持低电压配置。就是说，如果集线器端口不能支持一个高电压的功能，那么配置软件可以选择一个低电压的配置方案。注意，客户驱动程序在选择配置方案的时候也会涉及到。

配置软件必须保证可用的 USB 资源能够支持所选择的设备配置。当对设备进行配置的时候，资源管理软件必须跟踪所有的 USB 设备所消耗的总线带宽。如果 USB 能够提供所需要的总线电压，而且能够支持在设备配置文件中指出的所有的端点所要求的带宽，那么每一个端点都会建立起一个通信管道，设备配置也就完成了。如果 USB 不能支持请求的配置，那么设备就不会被配置，用户将会接受到这一通知。

根集线器的配置

主机软件通过配置根集线器开始 USB 设备的枚举过程。目前已经定义了两种集线器：通用主控制器（UHC）和开放主控制器（OHC）。每一个都提供同样的基本功能，完成同样的工作，但是采用不同的方式完成这些工作。这两个主控制器在第 18 章和 19 章被分别加以讨论。

根集线器必须要有一个状态变化端点（见图 11-2）它能被集线器的客户应用程序使用，用来检测关于每一个端口的状态改变。一旦集线器配置完成以后，软件就可以查询状态变化的端点，确定当前哪个端口有设备连接上去。

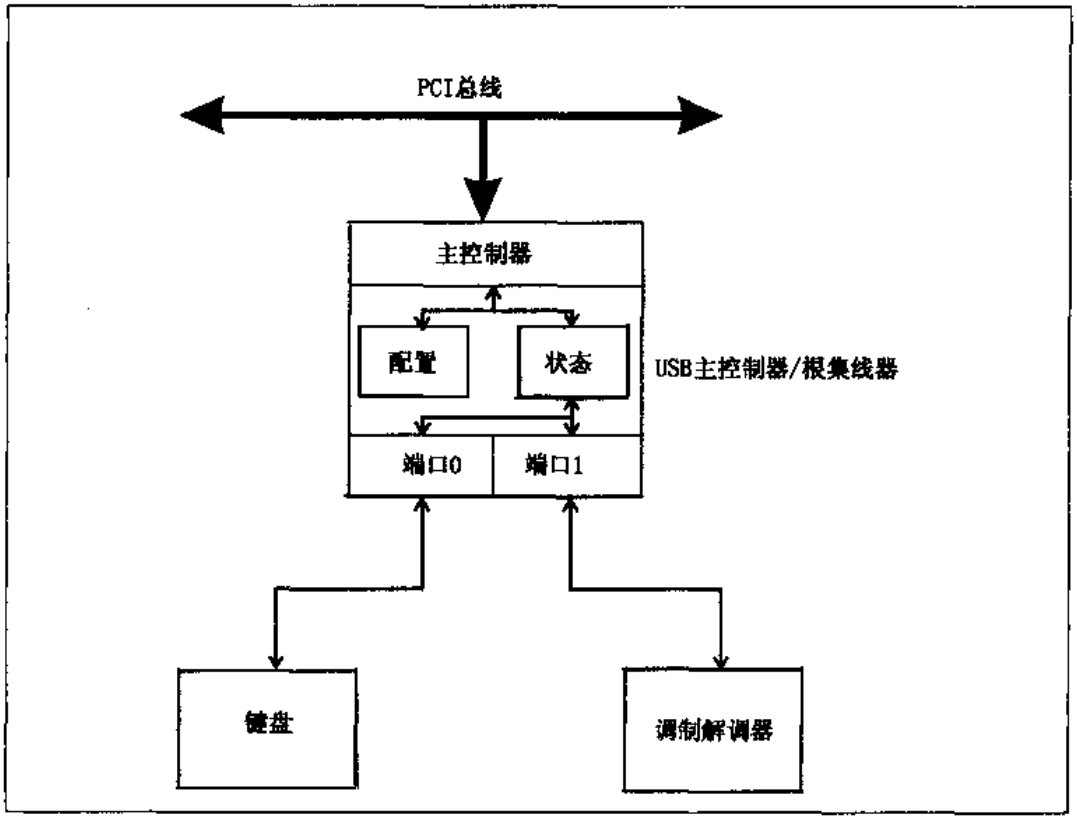


图 11-2　根集线器的控制和状态转换端点

设备为了配置而隔离

每个设备一开始都和 USB 隔离，因为所有的集线器端口一开始都是被禁止的。当电源被加到端口上的时候，主机软件检测到设备的存在，这种检测是通过读取状态变化端点来获取的。主机软件必须激活每一个和设备相连的端口。软件通过使用设置端口激活特征请求来激活一个端口。每一个设备都被顺序复位，并且被加以配置。

复位操作强迫设备被分配为缺省的地址（0）

在访问一个设备之前，该设备必须被复位。迫使它对它的缺省设备地址（0）做出响应。主机软件使用设置端口复位特征请求引起集线器把选择的端口复位。每一个 USB 设备在复位后响应一个 0 地址。在这种方式下，配置软件可以在缺省的地址读取每一个设备的描述符。

主机分配唯一的设备地址

在配置过程中，每一个设备被分配一个唯一的地址，这个地址用来做响应。因为在激活每一个下一个端口之前，都被安排一个唯一的地址，所以不会有冲突发生。标准的 Set Address 请求由软件使用，用于安排设备的地址。

主机软件验证配置

主机软件必须对它所检测到的每一个设备进行探测，以确定与设备相连的端点可以基于剩余的带宽实现分配。它必须保证设备所要求的总线供电量可以由和它相连的集线器端口提供。

设备可以有一个或多个配置可以选择。每一个配置描述符都代表一套不同的资源，设备可以选择它们。主机软件确保 USB 资源（总线电源和带宽）的需求能够得到满足。如果一个给定的配置不能得到满足，那么下一个配置就被采用。如果在尝试了所有的配置以后，USB 还是不能提供所需要的必要资源，设备就不会被配置。

☑ 供电需求

主机软件必须验证设备需要的总线供电可以由集线器端口来提供。在配置过程中，设备被指定不得消耗多于 100 毫安的电流，只有在设备配置完成以后，它才能消耗配置时所选择的最大功率。需要的最大总线功率在配置描述符中加以定义。类似的，集线器也会通过它的描述符报告它能够提供的功率数量。如果每一个配置都比集线器所能提供总线供电量多，那么这个设备就不会被配置。

☑ 总线带宽

主机软件必须验证所能满足的 USB 设备所要求的总线带宽。每一个配置都有一套端点，它必须是由客户应用程序能够访问的。每个端点描述符指定了它所需要的最大数量的 USB 带宽。如果对所有的端点来说，有足够的带宽可用，软件就能够建立一个通信管道，为每一个设备端点保留指定的总线带宽。在给所有的端点分配了指定的带宽之后，设备的配置就完成了。

如果总线带宽的需求得不到满足，那么就会检查其他的配置方案。如果每一种配置方案都超出了可用的总线带宽，那么设备就不能被配置。

分配的配置值

一旦选择了一个配置，主机软件就会配置这个设备，它是通过分配一个配置值来实现的，这个配置值对应于所选择的配置方式。该配置值从所选择的配置描述符中获得，然后通过设置配置请求写入设备中。该设备现在可以由客户软件存取，并且可以使用在它在配置中所指出的电流最大值。

客户软件接收通知

当一个设备被成功地配置以后，USB 系统软件必须找到合适的类驱动程序或者驱动程序，用来访问这个设备。系统软件必须通知这类 USB 设备它已经被安装好了，并且必须提供关于这些设备的特征和能力的信息。USB 系统软件识别设备驱动程序以及和驱动程序进行通信的过程是独立于操作系统的。

集线器配置

✎ 上一章

上一章对配置的过程作了一个概述。在 USB 设备枚举过程中的每一个主要的步骤都被加以了定义和讨论。

✎ 本　章

集线器是 USB 设备枚举过程中的关键。本章集中讨论集线器的特征以及包含在设备配置中的特征。

✎ 下一章

下一章讨论了 USB 集线器所定义的控制传输请求。集线器像所有的 USB 设备一样，支持标准请求，同时他也支持集线器特有的请求。

🕮 集线器的配置

集线器必须像其他的所有设备一样能够进行配置，但是它还要能够识别其他和它端口相连的设备。配置软件所采用的这一步包括：

- 读取标准设备描述符，获取配置设备所需要的信息；
- 给集线器分配唯一的地址；
- 给端口上电；
- 检查集线器的状态变化端点，检测端口事件；
- 读取状态信息，确定事件的性质；
- 激活端口，提供对相连设备的存取。

正如上面所指出的，集线器除了缺省端口之外还必须实现一个状态转换端口。图 12-1 所示的就是所需的集线器端口。缺省的控制端口提供了对描述符的存取，描述符定义了

设备所要求的配置类型。集线器也可以作为一个复合设备的一部分来实现。因而描述符还可以描述除了集线器最小需求之外的附加功能。

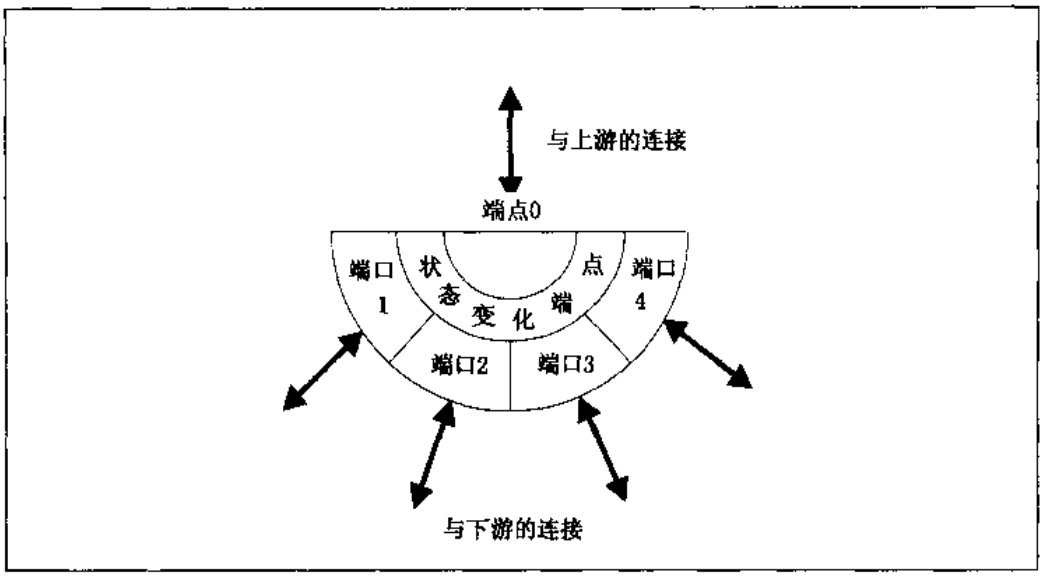


图 12-1　要求的集线器端点

缺省管道

所有包含集线器的设备在端点 0 都有缺省的控制管道。主机软件有缺省控制管道，用来配置集线器和 USB 设备。这些通信管道在初始化的过程中被建立，所以 USB 设备可以基于建立时的缺省值被存取。配置过程需要大量缺省的管道存取操作，用于配置和控制集线器的特征，包括：读取设备的描述符、给集线器端口上电、端口复位、读取端口的状态以及激活端口。

状态变化管道

集线器必须实现一个状态变化端点，该端点可以被查询，以检测集线器端口所发生的状态变化，（例如设备连接和断开连接）。要注意的是，状态端点为所有的集线器端口提供了状态信息。配置软件通过读取端点描述符确定了状态改变寄存器的特征和端点号。

读取集线器的端点描述符

集线器有一类特殊的描述符，称为集线器描述符。该描述符包含关于集线器的具体信息。集线器类的描述符是通过一些特殊的获取描述符请求来读取的。

集线器象所有的其他设备一样，也包含标准描述符，必须读取那些描述符，这样才能确定如何配置集线器。标准描述符通过标准请求 Get Descriptor 读取。集线器包含以下

的标准描述符，如图 12-2 所示：

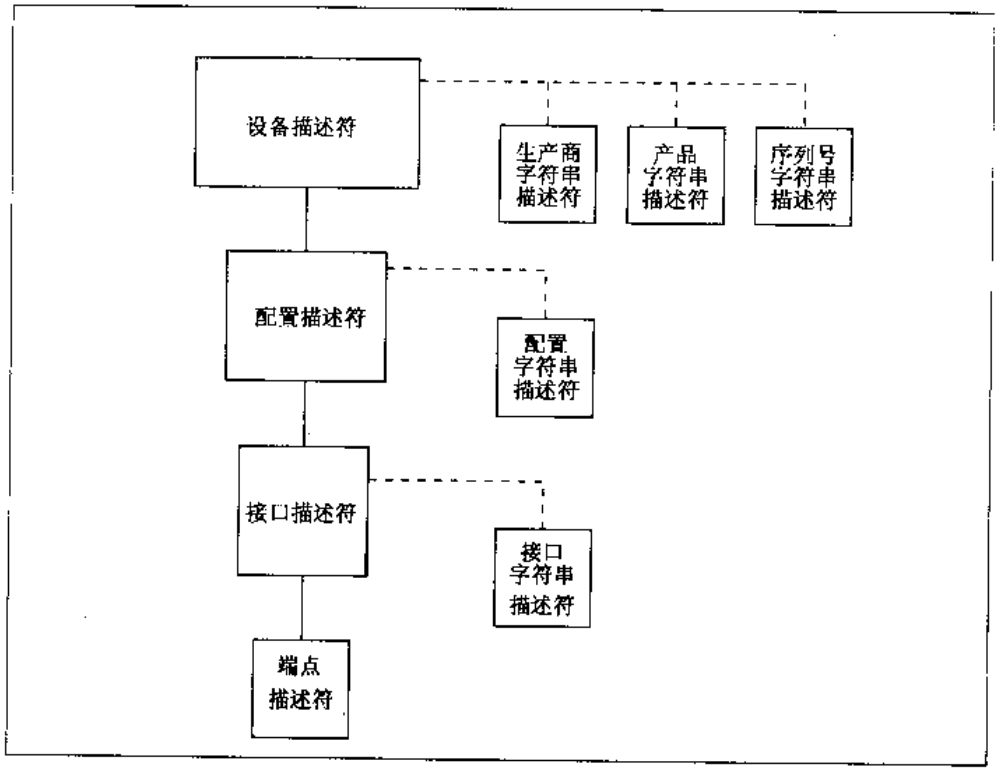


图 12-2　标准的集线器描述符

- 每个 USB 设备包含一个单独的设备描述符，它描述了设备所支持的配置数量；
- 每个设备包含一个或多个配置描述符，它描述一个或多个接口；
- 每一个接口描述符定义了和接口相关的端点数量。

端点描述符指出了和给定端点相关的属性，还有主机软件所需要的信息，以确定端点应该如何被存取。

字符串描述符是可选的，它是由 UNICODE 字符组成的，提供了可以显示出来供人们阅读的信息。

集线器设备描述符

集线器实现了标准的设备描述符，就像其他的 USB 设备一样。然而，集线器包含了一些预定义的描述符字段，如下所示：

- DeviceClass 字段＝HubClass；
- DeviceSubClass 字段＝HubSubClass；
- MaxPacketSize0 字段＝8 字节。

参见表 12-1。配置软件的第一次存取是对设备描述符进行存取，在那里确定缺省管道所支持的最大数据量（见表中 7 号字段）。在这种情况下，信息包的大小预定义为 8 字

节长。软件也可以通过读取设备描述符来检测设备的类型。然而，如果集线器是一个复合设备，那么类的字段将会是 0，而且接口描述符将定义设备的类别（一个接口用于集线器的功能模块，另一个接口用于每一个嵌入式的功能单元）。

表 12-1　　集线器的设备描述符

字段编号	字段名	字段长度	字段取值	说　明
0	长度	1	数字	描述符的长度（以字节为单位）。
1	描述符类型	1	01h	设备描述符的类型。
2	USB	2	BCD	本字段标识出这个 USB 设备和它的描述符所遵循 USB 规范标准的版本号，它是以 2 进制编码的形式出现的（例如，1.00 表示为 0x100）。
4	设备类别	1	集线器类别代码	集线器的类别代码（由 USB 分配）。 如果这个字段被设置为 0，那么在一个配置中的每一个接口都会指出自己的类别信息，而且不同接口的操作是彼此独立的。
5	设备子类	1	0	集线器的子类编码（由 USB 分配） 这个字段的编码由设备类字段的值加以限制。 如果设备类字段被置为 0，那么这个字段必须也被置为 0。
6	设备协议	1	0	协议代码（由 USB 指定） 这个字段的编码由设备类字段和设备子类字段的值加以限制。 如果一个设备支持基于设备而不是基于接口的类特定协议，那么这个代码就标识出设备使用的协议，该协议由设备类的规范说明书定义。 如果一个字段被置为 0，那么该设备就不使用基于设备的特定的类协议。但是它可以使用基于接口的特定的类协议。 如果这个字段被设置为 0xFF，那么这个设备就使用一个基于设备的厂商自定的协议。
7	最大包大小	1	08h	端点 0 的最大数据包大小。
8	生产厂商	2	ID	生产厂商的 ID（由 USB 指定）。
10	产品	2	ID	产品 ID（由制造商指定）。
12	设备	2	BCD	二进制编码形式的设备批号。
14	设备制造商	1	索引	描述制造商的字符串描述符的索引。
15	产品	1	索引	描述产品的字符串描述符的索引。
16	系列号	1	索引	描述设备系列号的字符串描述符的索引。
17	配置数目	1	数字	可能的配置数目。

集线器配置描述符

一个典型的集线器有一个配置描述符。这个描述符和标准 USB 设备的描述符一样定义。配置软件可以存取这个描述符，获取下面的信息：

- 配置所支持接口的数量；
- 用来配置集线器的配置值；
- 集线器是自供电还是总线供电；
- 在配置中集线器的最大耗电量。

表 12-2 定义了集线器设备所实现的标准配置描述符字段，在下面的讨论中可以参考表 12-2。

☑ 接口的数量

接口的数量定义在表中 4 号字段，它取决于集线器是否是一个复合设备。如果集线器是一个单独的功能模块（非嵌入式设备），那么集线器就只包含一个接口。然而，包含集线器的复合设备含有多个嵌入式设备，每个设备都包含一个或多个接口。

☑ 配置值

在第 5 行的配置值是 USB 在进行枚举时使用的，用来为集线器指定所选择的配置。配置值用 Set Configuration 请求来确定。同时也使对集线器状态变化端口的存取操作变为可行，计数器读取状态变化端口后就可以确定连接到集线器端口的设备数量。

如果集线器没有自己的电源开关的话，设置配置也允许电源加到集线器的端口上。如果有供电开关的话，必须要进行额外的存取动作才能给端口加电。注意端口在被加电之前是不会报告设备被连接上的。

☑ 总线供电或自供电集线器

表中的 7 号字段被定义为设备属性，这些属性指出集线器采用的是总线供电方式还是自供电方式，或者两种均可。如果两个位字段都被设置的话，那么这个集线器就是一个混合型的供电设备。如果集线器是一个总线供电型的设备，那么配置软件所用的最大的总线功率字段可以用来确定在每一个端口可以获得的最大供电量。关于总线供电和自供电的集线器的有关内容的进一步信息可以参见第 9 章。

☑ 最大总线耗电量

这个字段仅仅对总线供电或混和供电的集线器有效。它指出集线器将从 USB 总线上获取的最大电流。配置软件可以用这个值来减少每个端口所能得到的最大数量的电流。

这个值包括上拉和下拉电阻，集线器控制器，嵌入式设备和其他端口的耗电量。

表 12-2　　集线器配置描述符

字段编号	字段名	字段大小	字段取值	说明
0	长度	1	数字	描述符的长度，以字节为单位。
1	描述符的类型	1	02h	配置。
2	总长度	2	数字	为配置而返回的数据的总长度。包括所有描述符（配置、接口、端点、类特定和厂商自定的描述符）的总长度。
4	接口数量	1	数字	本配置所支持的接口的数量。
5	配置值	1	数字	作为一个 Set Configuration 的参数，在选择配置的时候会用到。
6	配置	1	索引	描述配置的字符串描述符的索引。
7	属性	1	位图	配置特性 D7 总线供电 D6 自供电 D5 远程唤醒（集线器不采用） D4...0 保留（置为 0） 一个设备在运行时的配置究竟是总线供电还是本地电源自供电可以用 Get Status 设备请求加以确定。
8	最大耗电量	1	毫安	本配置中集线器在总线上的最大耗电量。这个值包括所有端口的集线器控制器，所有的嵌入式设备。（数值以 2 毫安为单位）

集线器接口描述符

接口描述符是每一个功能单元都含有的。第一个接口描述符定义集线器的功能单元，而且嵌入式设备的后面的接口将会被包括（例如：复合集线器）。规范说明文档在一个集线器的接口描述符里预定义了两个字段的值：

- 端点数量（表 12-3 中的 4 号字段）——集线器的功能单元必须包含至少一个端点描述符，用来定义状态变化端点，所以预定义的端点数量是 01h。然而，附加的端点可以被定义为一个集线器类的设备。注意，缺省的端点是由规范说明书预定义的，所以不需要设备描述符。计数器知道每个设备都实现缺省的控制端点，从而允许对这个设备进行存取。因而，缺省端点并不需要描述符。注意当读取描述符的时候，如果这个集线器是一个复合设备的话，USB 计数器不知道被描述的接口是一个集线器功能块。
- 接口（见表 12-3 的第 8 行）——值 01h 标识出描述这个接口的字符串描述符的

偏移量。

表 12-3　　集线器接口描述符

字段编号	字段名	字段长度	字段取值	说　明
0	长度	1	数字	描述符的长度，以字节为单位。
1	描述符的类型	1	常数	接口描述符的类型。
2	接口数	1	数字	接口的编号。该值标识出本配置所支持的并发接口的队列的索引，它的最小值是 0。
3	可选设置	1	数字	用于为在前面的的字段标识出来的接口选择一个可选设置的数值。
4	端点号	1	数字	本接口所使用的端点编号（0 端点出外）。集线器必需要有一个状态变化端点。（集线器必需要有一个状态变化端点，但是还可以有其他的端点）。
5	接口类	1	类	类代码（由 USB 指定）。 如果这个字段被置为 0，那么这个设备就不属于任何一个 USB 指定的设备。
6	接口子类	1	子类	子类代码（由 USB 指定）。 这个代码受到接口字段中取值的限制。
7	接口协议	1	协议	协议代码（由 USB 指定） 如果这个字段被置为 0，那么这个设备就不在这个接口上使用任何特定的类协议。如果这个字段被设置为 0xFF，那么该设备就在这个接口上使用一个厂商自定的协议。
8	接口	1	01h	字符串描述符的索引。

状态端点描述符

集线器的功能单元仅仅定义了状态变化端点。表 12-4 列出了状态端点描述符的定义。计数器软件从端点描述符处读取下面的信息：

- 状态变化端点的地址；
- 传输方向；
- 端点支持的传输类型；
- 所支持的最大数据；
- 对端点进行查询的时间间隔。

注意，规范说明书中的要求是，当提出标准的 Get Descriptor 请求之后，状态变化端点必须是从集线器那里读取的第一个端点描述符。

☑ 状态变化端点的地址/传输方向

状态变化端点号在端点描述符的 2 号字段中被指定。该字段指出端点号（位 1-3），这个编号是在设计的时候确定的，而且是被硬件实现到设备的解码器中去的，它并不是

独立的。这个字段中的 7 号位指出了和端点相连的数据传输的方向。这个位必须加以设置，这样才能指出传输的方向是从状态变化端点到主机（例如：IN 事务处理）。

☑ 传输类型

端点所要求得传输类型定义在属性字段（3 号字段）。这个状态变化端点是基于中断传输类型进行存取的，而且它已经由规范说明书加以了预定义。主机软件将会按照一定的间隔查询状态寄存器，以确定是否发生了状态的改变。“检测集线器状态改变”部分描述了在一次中断传输过程中从状态改变端口读取的信息。

☑ 所支持的最大数据包

所支持的最大数据包字段（4 号字段）指出了在一次数据传输过程中，端点所能够支持的最大的数据量。一次中断传输的可选项是 8，16，32，或 64 字节，这与端点的数据缓冲区的大小有关，而且是根据它来选择的。

☑ 查询间隔

由于状态变化端点是为中断传输定义的，查询间隔（6 号字段）定义了每隔多久这个端点将会被查询。规范说明书预定义了状态变化端点的查询间隔，它的最大可允许的值是 FFh。所以配置软件会每隔 255 毫秒查询这个端点，以检测端口事件（例如，设备连接或者断开连接）

表 12-4　　集线器的状态端点描述符

字段编号	字段名	字段长度	字段取值	说　　明
0	长度	1	数字	描述符的长度，以字节为单位。
1	描述符的类型	1	常数	端点描述符的类型。
2	端点地址	1	端点	描述符所描述的 USB 设备上的端点地址。这个地址按以下的方式编码： 位 0...3 端点号； 位 4...6 保留，置为 0； 位 7 必须是“1”——IN 端点。
3	属性	1	位图	这个字段描述了端点的属性，当它进行配置的时候，采用下面的配置值： 位 0...1 传输类型； 11 仅中断传输使用； 所有其他位均保留。
4	最大包大　小	2	数字	当配置完成以后，本端点可以接收或者发送的最大信 0 息包的大小。
6	间隔	1	FF	数据传输的时候，查询的时间间隔。以微秒为单位。对于中断端点来说，它的范围是 1-255。

集线器类的描述符

有一个类特定的描述符是为集线器设备所定义的，参见表 12-5。这个描述符是通过类专有的 Get Descriptor 请求读取的。注意，这个描述符是由规范说明文档预定义的，索引从 0 开始。USB 计数器读取集线器的类描述符，以此确定以下信息：

- 是否支持带开关的供电模式；
- 集线器是不是复合设备的一部分；
- 设备是否为全局或单独的端口，有没有过流保护；
- 从软件提出给一个端口供电的要求到供电成功的时间间隔；
- 集线器控制器所要求的最大总线电流（对比表 12-2 的第 8 行）；
- 连接到端口的设备是否和端口断开了连接；
- 端口加电模式是组模式还是单独模式；
- 在阅读下面的部分时参见表 12-5。

☑ 电源开关模式的实现

一个集线器可以以三种方式控制电源。见 3 号字段，位 1..0，在集线器里面，类描述符定义了集线器将会采用哪一种模式：

- 组供电模式——所有端口的电源都被同时打开；
- 单独端口供电开关——电源被分别加到每一个端口上。Set Port Power Feature 请求用来把电源分配到每一个被选择的端口；
- 无电源开关——当配置集线器的时候，电源被加到所有的端口上。（例如，当作出 Set Configuration 请求的时候）。

注意，如果组供电开关被定义了，某些端口可能不受影响。进一步的内容可以参见“端口电源掩码”部分。

☑ 复合设备或者仅有集线器

集线器是否作为复合设备的一部分实现由集线器类描述符的 3 号字段的 2 号位确定。

☑ 过流保护模式

一个集线器可以选择不同的过流保护模式，只要它能够满足安全需求就可以了，但是不允许某个端口使用超过 5 安培的电流。3 号字段的 3、4 号位指出集线器当前采用了哪一种过流保护模式。如表 12-5 所示。

表 12-5　　集线器的类描述符

字段编号	字段名	字段大小	说　　明
0	DescLength	1	描述符的字节数，包括这个字节在内。
1	DescriptorType	1	描述符的类型＝19h（集线器类描述符）。
2	NbrPorts	1	集线器支持的下游端口的数目。
3	HubCharacteristics	2	D1...D0 电源的开关模式。 00 组电源开关（所有的端口同时供电）。 01 独立的端口供电开关。 1X 没有电源开关（当集线器打开的时候，端口总是有电的，当集线器关闭的时候，端口就没有电了。 D2 标识是否是一个复合设备。 0 集线器不是复合设备的一部分。 1 集线器是复合设备的一部分。 D4...D3 过流保护模式。 00 全局过流保护。当所有端口的电流总和超过规定的时候，集线器就会报告。但是某个单独的端口过流的时候，集线器不会作出报告。 01 单独端口的过流保护。某一个端口过流的时候，集线器就会作出报告，每一个端口都有自己的过流指示符。 1X 无过流保护。这个选项只适用于那些总线供电的集线器，它们不必实现过流保护。 D15...D5 保留。
5	PwrOn2PwrGood	1	从往一个端口上加电开始到那个端口可以使用为止的时间（以 2 毫秒为单位）系统软件根据这个值确定在访问这个上电的端口之前要等待多久。
6	HubContrCurrent	1	集线器控制器的最大电流需求，以 2 毫安为单位。
7	DeviceRemoveable	取决于端口的数量	指出和该端口相连的设备是否是可取下的设备。如果连接在这个端口上的是一个不可取下的设备，那么这个设备绝对不会收到一个状态变化的通告。这个字段的报告是以字节为单位的。在一个字节内，如果给定位置没有端口存在，那么这个字段代表的端口就回到 0 值。 位定义： 0 设备是可以取下的； 1 设备是不可取下的； 这是一个位图，对应于集线器上的某个端口： 位 1 保留； 位 2 端口 1； 位 3 端口 2； . . 位 n 端口 n（取决于实现）。

续 表

字段编号	字段名	字段大小	说　　明
可变	PortPwrCtrlMask	取决于端口的数量	指出一个端口是否被一个组模式的供电控制请求所影响。有这个字段设置的端口总是会要求一个人为的SetPortFeature（PORT_POWER）请求来控制端口的供电状态。 位定义： 0 端口不屏蔽组模式供电控制功能； 1 端口不受组供电模式命令的影响。必须人为得把命令发送到那个端口对是否供电进行控制。 这是一个位图，对应于集线器上的某个端口： 位 1 保留； 位 2 端口 1； 位 3 端口 2； . . 位 n 端口 n（取决于实现）。

☑ 电源开启到电源正常使用的延迟

配置软件必须知道要用多久设备才能够把电源加到设备上去。一旦系统软件要求给集线器上电，那么就会有一个延迟，直到电源在这个端口可以使用。软件必须等待直到它能够确定电源已经加在了端口上了，然后他才能够对端口进行存取。5 号字段定义了在端口上电请求和电源可以正常使用之间的延迟，这个值被定义为 2 毫秒。

☑ 集线器控制器的最大总线电流

在集线器类描述符的 6 号字段指出了集线器控制器所消耗的最大数量的控制电流。注意这个值和配置描述符中的最大电流字段中的值是不一样的，后者是指设备所消耗的总电流（参见标题为“最大总线功率消耗”部分）。

☑ 可断开/不可断开的设备

在 7 号字段的设备断开字段提供了一个集线器所支持的所有端口的位映射。每一个位置都对应一个给定的集线器（例如，位 1 表示端口 1）。如果这个位字段对应的端口被清 0，那么这个设备就是可断开连接的，如果这个为被置 1，那么就表示这个设备是永久连接的（例如一个嵌入式设备）。这个字段的大小是一个字节，所以支持端口 1 到端口 7，如果要支持 8-15 号端口，那么就要再加上一个字节，依此类推。

☑ 端口电源掩码

在集线器类描述符中的偏移量依赖于设备的可断开字段的大小。端口电源的控制掩码由一个位映射图组成，就像是设备可移动字段的组成一样。每个位都对应于一个给定的端口，而且定义了端口是通过组电源模式供电还是必须采用单独供电模式。如果组供电被掩码了，位字段就被设置，供电请求就必须通过“Set Port Power Feature”请求进行。

🕮 给集线器供电

一旦一个集线器配置完成以后，这个集线器上的端口就需要上电。当读取集线器的类描述符的时候，配置软件就会检测到端口的供电模式。如果必须手动上电，那么软件就会发送一个“Set Port Power Feature”请求。一旦某个端口被上电，集线器端口就能够检测到设备被连接到端口上，并设置状态位指出这个连接事件。集线器端口也就从原来的断电状态转变为上电状态。软件会检测到这个状态变化。

🕮 检查集线器的状态

配置软件查询状态变化端点，检测当前哪一个端口有状态改变事件。状态变化端点仅仅报告某个集线器或者集线器端口是否发生了状态改变，但是不会没有指出变化的特点。要确定变化的具体情况必须向集线器或者端口提出特别的请求：

- 已经发生的特殊事件；
- 引起该事件的对象的当前状态。

🕭 检测集线器状态变化

图 12-3 所示的就是集线器和端口状态改变位图，当状态变化端点被查询的时候返回该位图。所有的状态变化在时间片结束的时候被采样（EOP2 采样点），而且在下一个时间片过程中是可以读取的。配置软件知道端口的数量，而且当集线器端口状态变化端点被查询的时候，位图的大小也被返回。在字节大小字段报告状态，如果这个字段返回 0 就表示还没有实现的端口。对于大多数的端口来说将会返回一个字节，因为集线器实现的端口很少有多于 7 个的时候。

当软件查询时候，假定发生了状态变化，那么就会返回图 12-3 所示的位图。如果所有的位都没有置 1，那么就没有发生变化，在这种情况下，状态变化端点将会在 IN 事务处理的过程中返回 NAK，表示没有发生变化。

如果一个指定的端口改变被检测到，那么软件就会执行一个 Get Port Status 请求，集

线器的请求在第 13 章加以了定义。

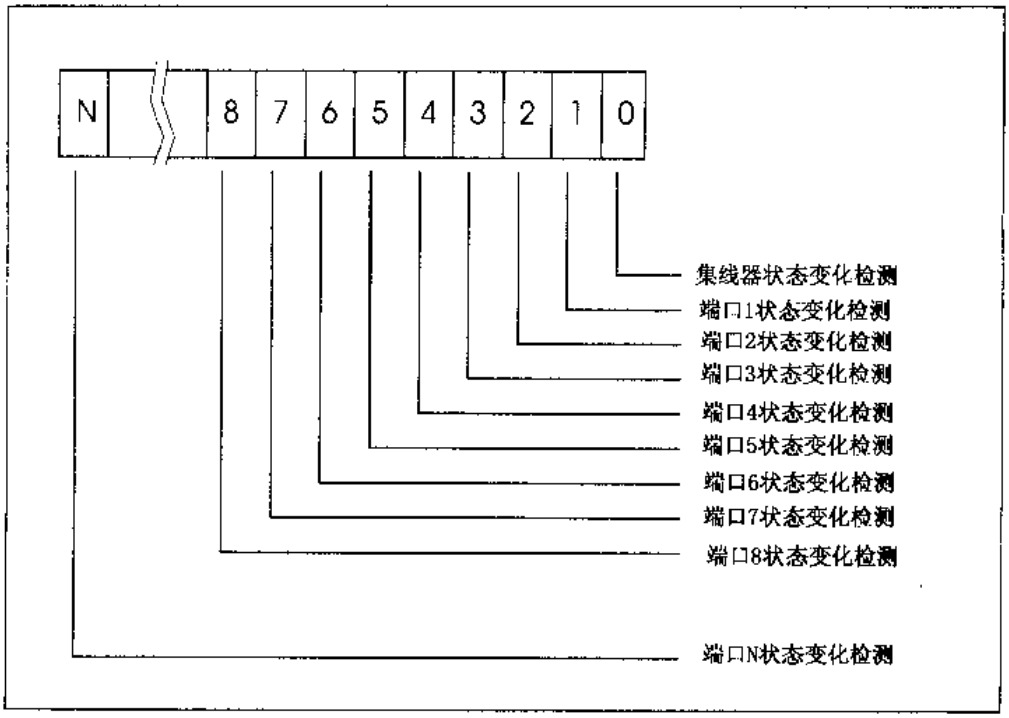


图 12-3　集线器的端口状态转换位图

读取集线器状态字段

如果检测到集线器的状态改变，软件就能够执行 Get Hub Status 请求，以获得变化的源。集线器的状态变化包含以下部分：

- 本地电源变化；
- 过流变化。

第 13 章详细描述了集线器的状态字段。详细内容可以参见标题为“集线器状态”的部分。一旦指定的集线器状态项的改变被检测到时候，软件就能采取合适的动作（例如，通过通知用户事件的办法）。通过使用 Clear Hub Feature 请求，软件能够读取和清除集线器的状态改变字段。例如，如果一个本地供电变化发生的时候，软件就使用“Clear Hub Local Power Feature”请求。

读取端口状态字段

当端口状态改变发生时，软件使用“Get Port Status”请求，确定哪一个端口特征发生了改变。集线器返回了端口状态字段，它在下面的部分被讨论，“读取端口状态字段”。可能的端口状态改变如下：

- 连接状态改变——设备或者和端口连接或者从端口断开连接；

- 端口激活/禁止变化——由于硬件事件引起的变化；
- 挂起改变——当恢复完成的时候，指出变化；
- 过流指示改变——仅仅由集线器使用，用来报告某一个端口过流；
- 复位变化——当复位处理完成的时候，就设置这个变化。

考虑下面的例子。当电源加到当前有设备相连的端口上时，就会指出一个连接状态改变。那么端口状态改变字段就将会改变，端口状态字段将会设置，电流状态字段将会指出某个设备当前被连接。配置软件认为设备已经被连接，并且通过执行一个“Clear Port Feature”请求来执行一次改变。

激活设备

当检测到设备已经连接到端口上以后，配置软件将会激活这个端口，并且尝试配置这个设备，一个端口是通过“Set Port Enable Feature”请求进行软件激活的。一旦端口被激活，它就转变到激活状态下，它就可以接收总线上的数据了。

集线器端口状态概述

下面的表列出了集线器的端口状态以及因为给定的信号事件或控制请求所发生的状态改变。

表 12-6　　集线器的端口状态

信号/状态	电源关闭	断开连接	禁　止	激　活	挂　起
根集线器端口复位（带电源开关的集线器）	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
根集线器端口复位（没有电源开关的集线器）	N.A.	进入断开连接状态	进入断开连接状态	进入断开连接状态	进入断开连接状态
ClearPortFeature PORT_POWER（带电源开关）	停留在断电状态	进入断电状态	进入断电状态	进入断开连接状态	进入断开连接状态
SetPortFeature PORT_POWER（带电源开关）	进入断开连接状态	N.A.	N.A.	N.A.	N.A.
SetPortFeature PORT_RESET	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	忽略	忽略	进入激活状态	停留在激活状态	忽略

续 表

信号/状态	电源关闭	断开连接	禁 止	激 活	挂 起
ClearPortFeature ORT_ENABLE	忽略	忽略	停留在禁止状态	进入禁止状态	忽略
下游信息包流通（集线器唤醒）	不向前传递	不向前传递	不向前传递	进入禁止状态	不向前传递
上游信息包流通（集线器唤醒）	不向前传递	不向前传递	不向前传递	向前传递数据	设置状态字段，不向前传递
SetPortFeature PORT_SUSPEND	忽略	忽略	忽略	进入挂起状态	忽略
ClearPortFeature ORT_SUSPEND	忽略	忽略	忽略	忽略	继续
断开连接检测	忽略	忽略	进入断开连接状态	进入断开连接状态	进入断开连接状态
连接检测	忽略	进入禁止状态	N.A.	N.A.	N.A.

集线器请求

✎ 上一章

集线器是 USB 设备枚举的关键。前一章的讨论集中在集线器的特征和设备配置中的特征。

✎ 本　章

本章讨论了为 USB 总线定义的控制传输请求。集线器和所有的 USB 设备一样，支持标准请求，此外它还必须支持集线器类专有的请求。

✎ 下一章

下一章讨论了和 USB 设备相连的非集线器设备的配置。讨论了设备描述符和其他与设备的配置相关的特性和特征。

🕮 概述

集线器必须对大量的 USB 设备请求或者命令作出响应。用标准请求配置集线器，控制 USB 接口的状态以及其他的一些特性。集线器必须支持和类有关的请求，用于控制特定的集线器和端口特征。所有的请求都是由主机发送的，采用的是控制传输机制。在进行配置之前，设备对应于它的缺省地址 0，这就允许了配置软件在配置过程中用设备地址 0 请求获取设备描述符的内容（从端点 0 读取）。

控制传输是用来传输设备请求的，最少要由建立阶段和状态阶段组成，但是也有可能包含数据阶段，这完全取决于所执行的请求类型。建立阶段是由一些 SETUP 事务处理组成的，如图 13-1 所示。SETUP 事务处理的 8 字节的数据定义了主机发送的请求类型。

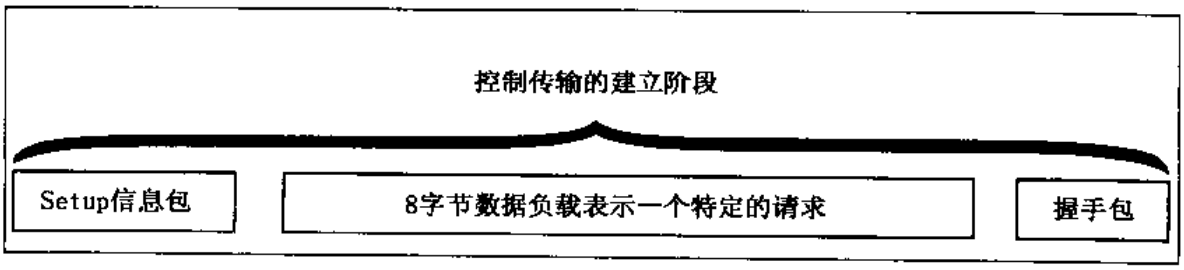


图 13-1　指定被执行的设备请求的建立事务处理的格式

集线器请求类型

8 字节的数据包定义了请求的类型，见表 13-1。数据包的 0 字节包含一个位映射，它定义了：

- 数据传输的方向；
- 请求的类型；
- 接受请求。

数据包的 0 字节是由一些位映射值组成的，这些值指出了一个标准请求的数据包的类型。就是说，如果第 5 和第 6 位都是 0 的话，那么请求就是一个标准请求，标准请求参见表 13-2。值 01b 指出请求是集线器专有的，集线器专有的请求列于表 13-4。

表 13-1　　SETUP 事务处理数据阶段的格式

字段编号	字段名	字段大小	字段取值	说　明
0	请求类型	1	位图	请求的特征： D7 数据传输方向： 0＝主机到设备； 1＝设备到主机； D6...5 类型； 0＝标准； 1＝类别； 2＝生产厂商； 3＝保留； D4...0 接收方； 0＝设备； 1＝接口； 2＝端点； 3＝其他； 4...31＝保留。
1	请求	1	数值	特定请求。
2	数值	2	数值	字长字段，根据请求的，类型而变化。
4	索引	2	索引或者偏移量	字长字段，根据请求的，类型而变化。一般用于传递一个索引或者偏移量。
6	长度	2	计数器	如果本次传输有一个数据阶段的话，那么它就表示要传输的字节数。

标准请求和集线器响应

集线器必须支持标准设备请求，如同别的任何 USB 设备一样。表 13-2 列出了标准请求和集线器对这些请求的响应。进一步的细节内容可以参见第 15 章关于标准请求的部分。

表 13-2　　集线器对标准设备请求的响应

请求	请求字段的取值	集线器的响应
CLEAR_FEATURE	1	清除设备中被选择的特征。
GET_CONFIGURATION	8	返回用于配置设备的配置值。
GET_DESCRIPTOR	6	返回被选择的描述符。
GET_INTERFACE	10	可选的（仅要求支持一个接口的集线器）。
GET_STATUS	0	返回关于设备状态的信息。
SET_ADDRESS	5	用于给设备安排一个唯一的地址。
SET_CONFIGURATION	9	通过指定一个所选配置描述符的配置值对设备进行配置。
SET_DESCRIPTOR	7	可选的（用于更新或者修改一个选择的描述符）。
SET_FEATURE	3	设置选择的特征，该特征和设备相关。
SET_INTERFACE	11	可选的（仅要求支持一个接口的集线器）。
SYNCH_FRAME	12	可选的（集线器不需要有同步端点）。

集线器类的请求

集线器必须支持特殊的类请求。当表 13-1 中的请求类型字段（位 5..4）被设置为 01b 的时候，这个请求就被解释为特殊类请求。集线器类的请求列于表 13-3 中。

表 13-3　　集线器的类请求代码

请　求	取　值
GET_STATUS	0
CLEAR_FEATURE	1
GET_STATE	2
SET_FEATURE	3
保留	4-5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7

每一个集线器类的请求格式和定义参见表 13-4。只有集线器设备支持这些特定的请求。每一个请求将在下面的部分详细加以讨论。

表 13-4　　集线器的类特定请求

请求类型	请求	取值	索引	长度	数据
00100000B	CLEAR_FEATURE（01h）	特性选择器	0	0	无
00100011B	CLEAR_FEATURE（01h）	特性选择器	端口	0	无
10100000B	CLEAR_BUS_STATE（02h）	0	端口	1	各个端口的总线状态。
10100000B	GET_DESCRIPTOR（06h）	描述符类型和描述符索引	0 或语言 ID	描述符的长度	描述符。
10100000B	GET_STATUS（00h）	0	端口	4	集线器状态和变化指示符
10100011B	GET_STATUS（00h）	0	端口	4	端口状态和变化指示符。
00100000B	GET_DESCRIPTOR（07h）	描述符类型和描述符索引	0 或语言 ID	描述符长度	描述符。
00100000B	SET_FEATURE（03h）	特性选择器	0	0	无
00100011B	SET_FEATURE（03h）	特性选择器	端口	0	无

获得/设置位描述符

类特有的 Get Hub Descriptor 请求给主机软件提供了一个读取集线器类描述符的方法。在 SETUP 事务处理中的“数值”字段必须申明，它是集线器描述符的索引号。集线器类描述符的格式和定义可以在表 12-5 中找到。

Set Descriptor 请求是可选的。这个请求用来更新集线器的描述符，它仅仅对那些提供更新描述符机制的集线器有效。

获取集线器状态的请求

Get Hub Status 请求返回当前的集线器状态，用的是状态变化指示符。返回的字节两个表示状态，两个表示请求的数据阶段的状态改变，注意集线器状态信息在本质上是全局的，而且可以适用于所有的和集线器相连的端口。

集线器的状态字段

集线器的状态信息定义了两个字段：

- 本地供电状态；
- 过流指示。

☑ 本地供电状态

本地供电状态字段仅仅适用于那些既支持自供电，又支持总线供电的集线器，或者说支持混和供电的集线器设备（例如，总线接口由总线供电，端口则是由本地电源供电）。这个字段指出集线器的本地供电机制是好的还是坏的。由于 USB 的接口逻辑电路是由 USB 总线供电的，所以即使集线器其余部分已经没有供电了，也会返回状态信息。位定义是：

- 0＝本地供电正常；
- 1＝失去本地供电。

如果这个位返回“0”，那么给接口和集线器的供电就是正常的。注意，如果总线接口的集线器失去供电，那么当执行“Get Hub Status”请求的时候就会发生超时的情况，因为集线器不能作出响应。如果接口的电源是好的，那么事务处理就能够正常地完成，而且会报告本地供电的电流状态。

☑ 过流指示

过流指示仅仅适用于那些能够对全局过流保护（所有端口电流的总和）作出报告的集线器上。具体报告全局过流状态还是单独某个端口的过流状态是由集线器描述符的“Hub Characteristic”字段决定的（见表 13-5）。这个字段的定义是：

- 0＝所有电源的操作正常；
- 1＝一个过流条件存在于集线器方。

如果出现过流状态，那么所有端口的都会失去供电。

表 13-5　　在 Get Hub Status 请求期间返回的集线器状态字段的格式

7	6	5	4	3	2	1	0
保留（返回 0）						过流指示符	本地供电状态

15	14	13	12	11	10	9	8
（返回 0）							

集线器状态变化字段

有两个位字段定义在集线器状态变化字段内，见表 13-6：

- 本地供电状态变化；
- 过流指示改变。

☑ 本地供电状态变化

这个字段直接对应于本地供电状态位，它报告本地电源的状态。如果本地供电状态位被实现，那么就必须要有这个位，反之也然。相应的变化位指出自从最后一次确认以后，是否在本地供电状态上有变化。位的定义如下：

- 0＝本地供电状态没有发生变化；
- 1＝本地供电状态发生了变化。

如果这个位被设置，那么就会检查本地供电状态，以确定当前的本地供电状态。

☑ 过流指示变化

这个位字段对应于过流指示位，而且如果有过流指示位，那么这个字段必须也要有，反之也然。位的定义如下：

- 0＝过流状态指示符没有发生变化；
- 1＝过流指示符发生了变化。

变化位告诉软件它们过流状态发生了变化，软件应该检查过流指示符位，以此确定当前电流限制逻辑电路的电流状态。（例如：因为发生了过流状态，所以所有的端口都断电了）。

表 13-6　　在集线器的状态请求期间返回的集线器状态变化字段的格式

7	6	5	4	3	2	1	0
保留（返回 0）						过流指示符变化	本地供电状态变化

15	14	13	12	11	10	9	8
（返回 0）							

设置/清除集线器的特征请求

集线器类的“Set Feature”和“Clear Ferture”请求的定义取决于特征选择字节的定义。选择字节定义了集线器的特性，它是 0 或者 1，这个特性被加到特征位上去，因而

这个 SETUP 事务处理的“索引”字段就是集线器特性的 00h。表 13-7 列出了为集线器类请求定义的集线器特性。

表 13-7　　特性选择器和索引值

	接收方	取　值	索　引
C_HUB_LOCAL_POWER	集线器	00h	00h
C_HUB_OVER_CURRENT	集线器	01h	00h

集线器本地供电变化请求

如果软件通过“Get Hub Status”请求检测到一个集线器本地供电状态改变。软件就会通过“Clear Hub Local Power Change Feature”请求确认这个请求。这个请求清除了变化字段，这样集线器本地供电的后续请求就可以被检测到。

“Set Hub Local Power Change”请求可以用来设置集线器本地供电的状态位，引起相应的变化位被设置。虽然规范说明文档没有定义这些设置特征请求的使用方法，但是当用于调试目的的时候还是有用的，用于模拟一个集线器的本地电源变化。注意这个请求不能用来确认一个变化的状态。

集线器过流变化请求

“Clear Hub Over-Current Change”请求用来确认一个集线器的过流状态，它是通过“Get Hub Status”请求来检测的。这个请求清除了指示符的改变位，从而使它有可能在集线器的过流指示符上辨认出后续的改变。

“Set Hub Over-Current Change”请求设置过流指示符，引起变化反应在相应的指示符变化位字段。因为有了“Set Hub Local Power Change”请求，所以该请求看上去已经实现对调试功能的支持了。

获取端口状态请求

“Get Port Status”请求被定义在 SETUP 事务处理的接收字段。获取集线器状态请求把接收者定义为设备（例如集线器）。然而，“Get Port Status”状态的接收方被定义为“其他”（在这里，“其他”是指端口）。“索引”值定义了选择哪一个端口。请求包含一个数据阶段，在这个阶段将返回端口状态和端口变化指示符。

就象集线器状态信息一样，端口状态信息返回四个字节的信息，两个字节用于定义端口状态字段，另外两个字节用于端口变化字段。

端口状态字段

端口状态字段见表 13-8，低 7 位字段用来表示被选择的端口的当前状态。每一个位都将在下面的部分加以讨论。

表 13-8　　在 Get Port Status 请求期间返回的端口状态字段的格式

7	6	5	4	3	2	1	0
保留（返回 0）		复位状态	过流指示符	挂起状态	端口激活/禁止	过流指示符变化	本地供电状态变化

15	14	13	12	11	10	9	8
（返回 0）						连接的低速设备	端口供电

☑ 当前连接状态字段

这个字段反映了当前是否有一个设备被连接到了端口上。字段中的值反映了端口当前的状态，而且可能不直接对应于引起插入状态变化位（位 0）被设置的事件。

- 0＝当前没有设备和这个端口相连；
- 1＝当前有一个设备和这个端口相连。

对于那些和不可移动设备连接的端口而言，这个字段总是 1。

☑ 端口激活/禁止

端口可以只通过主机软件激活。但是也可以通过一个缺省条件（断开连接的事件或者其他的缺省状态，包括一个过流指示）把端口禁止掉，或者由主机软件禁止这个端口。

- 0＝端口被禁止；
- 1＝端口被激活。

☑ 挂起

这个字段指出挂起行为。设置这个字段将会引起设备挂起，从而不再向下传递总线的数据。重新设置这个字段引起设备恢复。总线数据传输不能在总线事务处理的过程中被恢复。如果设备本身接到一个恢复信号，那么这个字段就会被集线器清 0。

- 0＝没有挂起；
- 1＝挂起。

☑ 过流指示

这个字段仅仅适用于集线器，它为每一个端口报告过流状态，如果集线器没有报告哪一个端口有过流情况，那么这个字段就会被保留，并且返回所有的 0 值。

当设置过流指示标志后，就表示连接到这个端口的设备所用的电流已经超过了指定的最大值，而且那个端口已经被停止供电了。对端口不再供电的动作还反映在端口供电激活/禁止字段上。

这个字段指出因为设备连接到这个端口而造成的过流状态。

- 0＝这个端口上所有的供电操作都是正常的；
- 1＝这个端口存在一个过流状态。而且这个端口已经停止供电。

☑ 复位

当主机希望把所有和它相连的设备复位的时候，这个字段就会被置 1。此后在复位信号被集线器和复位状态改变该字段设置之前，这个位一直保持 1。

- 0＝没有发出复位信号；
- 1＝发出了复位信号。

☑ 端口供电

这个字段反映了端口的供电状态。因为集线器可以实现不同方式的端口电源开关，所以这个字段中剩下的位就用来表示不同的供电方案。集线器的类描述符用来报告集线器实现的供电开关的类型。在端口处于配置状态之前，集线器不给它们供电。

- 0＝端口没电；
- 1＝端口上电。

不支持供电开关的集线器在这个字段总是返回 1。

☑ 低速设备的连接

这个字段仅当设备连接的时候才有关。

- 0＝和端口相连的是全速设备；
- 1＝和端口相连的是低速设备。

端口变化字段

这个字段见表 13-9，有 5 个位字段被定义，用来报告指定端口的状态和标志改变。每一个位字段都将在下面的部分加以讨论。

表 13-9　　在“Get Port Status”请求期间返回的端口变化字段的格式

7	6	5	4	3	2	1	0
保留（返回 0）			复位完成变化	过流指示符变化	挂起变化（恢复完成）	端口激活/禁止变化	连接状态变化

15	14	13	12	11	10	9	8
（返回 0）							

☑ 电流状态改变

表示在当前端口的连接状态下发生了一个变化。当集线器检测到一个状态改变已经发生以后，它就会设置这个位。

- 0＝当前连接状态没有发生变化；
- 1＝当前的连接状态已经改变了。

如果和这个端口相连的是永久连接的设备，那么在复位以后这个位就会被置 1。

☑ 端口激活/禁止改变

当硬件事件启动一个端口禁止变化事件以后，这个位就被置 1。（例如，一个断开连接的事件或者其他缺省的条件，包括一个过流指示）这个位不受主机软件启动激活/禁止变化事件的影响。

- 0＝没有发生端口激活/禁止的变化；
- 1＝因为硬件事件发生了端口激活/禁止状态的变化。

☑ 挂起变化（恢复完成）

这个字段指出连接到这个端口上的设备已经完成了恢复过程（例如，集线器终止了恢复信号，然后是 3 毫秒的不活动期，使设备可以和 SOF 重新同步），当设备进入挂起状态以后，这个位不会被置 1。

- 0＝没有变化；
- 1＝恢复完成。

☑ 复位完成

当这个端口的复位操作完成以后，这个字段就会被置 1。复位完成也会引起端口的

激活状态位被置为 1 和挂起变化字段复位。

- 0＝没有改变；
- 1＝复位完成。

设置/清除端口特征

复位和清除特征请求可以指定一个给定的集线器特征，或者可以和单独的端口相联系。集线器特征请求和端口的特征请求是不同的，它们的区别在于 SETUP 事务处理阶段的“索引”字段的值。这个索引字段标志出请求所适用的端口号。注意，端口号 0 是不允许的，因为集线器将会把这个请求解释为集线器特定请求，而不是端口特定请求。表 13-10 列出了可以被置 1 或者清 0 的专门的端口特征。

表 13-10　　端口特定请求的特征选择和索引值

	接收方	取　值	索　引
PORT_CONNECTION	端口	00h	端口号
PORT_ENABLE	端口	01h	端口号
PORT_SUSPEND	端口	02h	端口号
PORT_OVER_CURRENT	端口	03h	端口号
PORT_RESET	端口	04h	端口号
PORT_POWER	端口	08h	端口号
PORT_LOW_SPEED	端口	09h	端口号
C_PORT_CONNECTION	端口	16h	端口号
C_PORT_ENABLE	端口	17h	端口号
C_PORT_SUSPEND	端口	18h	端口号
C_PORT_OVER_CURRENT	端口	19h	端口号
C_PORT_RESET	端口	20h	端口号

在标题为“获取端口状态请求”的部分的前一部分定义了状态和指示符位、状态和指示符变化位，它们是由集线器为每个端口维护的，当发出一个 Get Port Status 请求时，它们就会被返回。这些位返回集线器支持不同的端口特征。设置和清除端口特征请求提供了一种方法用于激活和禁止特定的特征，而且允许软件确认变化，当执行获取端口状态请求的时候，就可以检测到这些变化。

获取总线状态

定义这个可选的请求是为了使问题的诊断变得方便，这是通过提供总线状态信息来实现的。返回的信息在最后的 EOP2 点被采样，这个点在最后的选择端口被检测到。总线状态信息在数据阶段以单字节的方式被返回。位定义可以参见表 13-11。

表 13-11　　在“Get Bus Status”请求期间返回的总线状态字段的格式

7	6	5	4	3	2	1	0
保留（返回 0）						在最后一个 EOF2 点采样的 D+上的状态。	在最后一个 EOF2 点采样的 D-上的状态。

USB 设备配置

✎ 上一章

上一章讨论了为 USB 集线器定义的控制传输请求。就像是所有的 USB 设备一样，集线器支持标准请求，而且也必须支持集线器类的特定的请求。

✎ 本　章

本章讨论了非总线设备的配置，这些设备和 USB 相连。设备描述符和其他的特征相联系，这些请求在设备类的相关章节加以了讨论。

✎ 下一章

下一章讨论了 USB 设备所要求的标准请求。每一个设备类都可以定义和类相关的特定请求，他们将在相关设备类的章节中加以讨论。

🕮 概述

在配置一个设备以前，和设备连接的集线器必须已经被配置完成，而且端口已经上电了。接着，集线器和配置软件必须检测到和端口连接的设备。

集线器必须知道设备已经被连接上了，它是通过监视 D+和 D-端口的信号来做出判断的。

- 集线器为端口设置状态信息，指出设备已经连接上了以及设备的速度；
- 配置软件读取端口状态，而且认为所有的全速端口设备已经连接上了；
- 然后软件就激活这个端口，所以集线器将会把总线上的数据传递给设备；
- 然后配置软件发送一个 Reset Port 请求，迫使这个端口进入缺省状态。在缺省状态下，设备不会被配置，而且仅仅响应对被访问的目标设备 0 和端点 0 作出响应。

配置软件现在可以开始设备的配置过程。这个过程和配置一个集线器的过程是相似的。

- 主机请求设备的控制端点（0），以确定缺省管道所支持的最大数据量；
- 主机给 USB 设备分配一个唯一的地址；
- 主机从描述符那里读取配置信息并加以执行；
- 主机验证那些设备需要的 USB 资源是否可以获得；
- 主机给 USB 设备发送一个配置值，指出如何使用该设备。当设备收到配置值的时候，设备采用它所描述的特征。设备现在已经为客户软件的存取做好了准备，而且现在它可以获取描述符中所指定的电流。

读取和解释 USB 的描述符

设备的实现必须建立描述符，用以反映设备的特征和行为。本章提供了标准描述符的定义和格式，它和每一个 USB 设备有关。标准的描述符包括：

- 设备描述符——描述了设备所支持的配置的数量；
- 配置描述符——指出了一个或多个接口而且定义了某些和这个配置相关的属性；
- 接口描述符——定义了端点的数量，它们和接口相关，而且定义了某些和接口相关的属性；
- 端点描述符——指出了和给定的端点相关的属性，以及那些主机软件需要的信息，这些信息可以确定这个端点应该怎样被访问；
- 字符串描述符——可选的描述符，由 UNICODE 字符串组成，它提供了那些可显示出来供人们读取的信息；
- 类特定描述符——当定义一个特殊的设备类的时候，这个设备类就必须要一个附加的描述符。

每一个描述符都包含一个类型字段，用它来识别上面列出的这些描述符的类型。注意，1.0 版的规范说明书中定义的仅有的特定类描述符是集线器设备类的描述符。其他类描述符，如果要求的话，都被定义在相关类的描述符内。

表 14-1　　描述符的类型值

描述符类型	取　值
设备	1
配置	2
字符串	3
接口	4
端点	5

设备类

设备配置的一个重要的方面是确定一个特定的设备属于哪一个设备类。一个设备的类定义提供了主机和客户软件使用的信息，以确定一个设备如何被控制和访问。主机软件使用设备类定义标识相应的 USB 类的设备驱动程序。知道和类特定描述符相关的定义的类驱动程序就可以给设备赋值，以确定设备的特性。

设备类被定义在单独的设备说明文档内。在写下面这些的类的时候，说明文档已经有了设计计划了。

- HID 设备类——人机接口设备；
- 通信设备类；
- 监视设备类；
- 海量存储设备；
- 音频设备类。

第 17 章提供了一个对所有设备类的概述

设备描述符

表 14-3 定义了设备描述符的格式。下面的部分讨论了在配置过程中，主机软件如何给设备描述符赋值。某些字段在下面的部分没有加以讨论，因为对这些字段的定义包含在表 14-3 中，不需要分类（这是作者的观点）。

类代码字段

类代码在设备描述符里可以定义，也可以不定义，因为某些设备可以有大量的接口，需要不同的类驱动程序。如果一个设备可以被某个驱动程序所访问的话，那么类代码将会在设备描述符中指出，如果某个设备需要多于一个的类驱动程序控制和访问，那么类代码将会被定义在接口描述符内。类似有很多例子，其中某些例子列于下面。

只有一个类定义的设备的特征是只有一个编程接口，例如：

- 集线器设备；
- 麦克风；
- 扬声器；
- 鼠标；
- 键盘。

具有多个类定义的设备的特征是具有多个编程接口，这类设备包括：

- 数字 USB 电话——这个设备的特征是属于两个不同的类，音频接口（发送器和

接收器）和人机接口（拨号装置）；

- CD-ROM——这个设备的特征是有几个不同的编程接口，有自己的设备类的定义，包括音频，视频和大容量存储。具体使用那一种设备类的驱动程序，取决于当前使用那一种软件应用程序；
- 复合设备——一个复合设备是指把一个简单的接口用于两个不同的功能单元。例如，一个键盘也可以有一个集成的扫描仪，和/或一个 USB 的耳机插座。每一个设备都有自己的设备描述符，定义了它的设备类的类型；
- 混合设备——在规范说明书里，一个复合设备被定义为集成了其他的功能单元的集线器类设备。例如一个 USB 打印机也可以包含一个集线器。

如果设备描述符没有定义一种设备的类型（类代码字段＝00h），那么接口描述符将定义每一个被描述的接口的类的类型。注意，如果类代码是 0 的话，子类字段也必须是 0。

如果一个类代码字段包含 FFh，那么就意味着描述符的定义是厂商自定的，而且只有厂商自定的 USB 设备驱动程序能够正确地解释描述符和访问这个字段。

最大数据包大小字段

这个字段被配置软件用来确定端点 0 所支持的最大数据包的大小。当开始访问设备时，它的最小数据包的大小是 8 字节。如果这个字段定义的数据包的大小指定了一个最大值，那么设备就包含了一个大的数据缓冲区，它可以支持所定义的数据包的大小。对端点 0 的后续访问就可以被更加有效地执行。

设备制造商、产品、序列号

可选的字符描述符也可以被包含进来，提供人类可读的信息，这些信息和制造商，产品以及设备的序列号相关。每个这些字段都定义在设备描述符的 16...14 号字段，标识出字符串描述符的位置，它们标示出设备的制造商，并且描述这些产品，列出设备的序列号。在这些描述符里的数值被定义为索引。索引被软件用来存取相应的字符串，存取动作是通过执行“Get Descriptor”请求来完成的。为了存取一个字符串描述符，当执行“Get Descriptor”请求的时候，软件必须指出描述符的类型，（01h）和索引号（在描述符里被指出）。

表 14-2 列出了“Get Descriptor”请求的建立数据的定义。注意，数值字段指出了描述符的类型和索引，而且索引字段包含了语言 ID。

表 14-2　　用于读取字符串描述符的“Get Descriptor”请求的定义

请求的类型	请　求	数　值	索　引	长　度	数　据
10000000B	GET_DESCRIPTOR (06h)	描述符的类型和索引	语言 ID	描述符的长度	描述符

配置的数目

一个设备可以定义附加的配置，这样可以提高配置过程的灵活性，而且增加了设备可以被成功配置的可能性。举个例子，一个设备可以消耗 2 个单位的电流（200 毫安）；然而，如果设备和一个总线供电的 USB 设备相连的话，那么它就只能用到 100 毫安的电流。在这种情况下，这种设备就不能被支持，主机软件也不会激活这个设备。为了避免这个问题，设备的设计者可以提供一个可选的配置，把设备对总线电流数量的要求减少到 100 毫安。

配置字段的数量指出了所支持的配置的数目。主机软件读取配置信息，而且决定选择那一种配置。这份规范说明指出可以通过询问设备驱动程序来确定选择哪一种配置，但是对这种来自设备驱动程序的输入机制并没有加以定义。

表 14-3 务　　驱动程序描述符的定义

字段编号	字段名	字段大小（以字节为单位）	字段数值	说　　明
0	长度	1	数字	描述符的长度（以字节为单位）。
1	描述的符类型	1	常数	设备描述符，类型＝01h。
2	USB	2	BCD	本字段标识出这个 USB 设备和它的描述符所遵循 USB 规范标准的版本号，它是以 2 进制编码的形式出现的（例如，1.00 表示为 0x100）。
4	设备类别	1	类	类别代码（由 USB 分配）。 如果这个字段被设置为 0，那么在一个配置中的每一个接口都会指出自己的类别信息，而且不同接口的操作是彼此独立的。 如果这个字段被赋予一个数值，这个数值必须在 1 和 0xFEh(254)之间，那么这个设备就支持在不同的接口上的不同的类规范，而且这些接口可能不能独立操作。这些值为一组接口区分出它们的类定义。（例如，一个带有音频和数字接口的 CD-ROM 设备，它要求把弹出 CD 或者播放 CD 的控制传输过去）。 如果这个字段被置为 0xFF，那么这个设备类就是设备供应商自定的。
5	设备子类	1	子类	子类编码（由 USB 分配）； 这个字段的编码由设备类字段的值加以限制。 如果设备类字段被置为 0，那么这个字段必须也被置为 0。

续表

字段编号	字段名	字段大小（以字节为单位）	字段取值	说　　明
6	设备协议	1	协议	协议代码（由 USB 指定）。 这个字段的编码由设备类字段和设备子类字段的值加以限制。如果一个设备支持基于设备而不是基于接口的类特定协议，那么这个代码就标识出设备使用的协议，该协议由设备类的规范说明书定义。 如果一个字段被置为 0，那么该设备就不使用基于设备的特定的类协议。但是它可以使用基于接口的特定的类协议。 如果这个字段被设置为 0xFF，那么这个设备就使用一个基于设备的设备供应商自定的协议。
7	最大数据包大小	1	数字	端点 0 能处理的最大数据包的大小（只有 8、16、32、或 64 是有效的）。
8	设备供应商	2	ID	设备供应商的 ID（由 USB 安排）。
10	产　品	2	ID	产品 ID（由设备制造商指定）。
12	设　备	2	BCD	设备的出厂编号，它以二进制形式出现。
14	设备制造商	1	索引	描述产品制造商的字符串描述符的索引。
15	产　品	1	索引	描述产品的字符串描述符的索引。
16	系列号	1	索引	描述设备的系列号的字符串描述符的索引。
17	配置数目	1	数字	可能的配置数目。

配置描述符

软件读取配置描述符，获得关于一个给定的选项的全局信息。下面的部分讨论了配置软件确定配置特性的时候需要检测的字段。在下面的讨论中参见表 14-4

接口的数量

如前面所讨论的，一个给定的设备可以有两个或多个接口，它们会要求不同的设备类驱动程序。一个接口是由一套端点集合组成的，通过它们，一个给定的设备驱动程序就可以对它的设备进行控制和通信。Number Interfaces 字段指出了接口的数量，它们在配置过程中被赋值。

配置值

一旦配置软件已经选择了某一个已经被设备定义的配置，那么就必须配置这个设备。每一个配置描述符都有一个唯一的配置值，用来配置这个设备。在配置值被写到设备上之前，设备不会消耗超过 100 毫安的电流，而且这时设备不是完全可操作的。

配置软件通过使用“Set Configuration”请求配置一个设备。这个配置值是在“Set Configuration”期间的 SETUP 事务处理的数值字段被指定的，参见表 15-2。一旦被配置后，设备就会使用选择的配置所定义的特征。

属性和最大功率

配置属性定义了设备被如何供电，以及它是否支持远程唤醒功能。一个设备配置报告这个配置是总线供电的还是自供电的。设备状态报告了该设备是否是自供电的。如果设备和它的外部电源断开连接，那么它就会更新设备的状态，指出这个设备不再是自供电的了。

当一个设备不能得到外部供电的时候，除了在配置描述符中指出的功率之外，它可以不增加从总线处获得的功率。

如果一个设备和它的外部电源断开连接以后可以继续操作，那么它就会继续操作。如果这个设备不能继续操作的话，那么这个设备就不能再支持某些操作了。主机软件可以通过检查状态和设备的断电情况而确定失败的原因。这些信息可以通过 Get Status 请求获取信息（见“设备状态”部分）。

表 14-4　　配置描述符定义

字段编号	字段名	字段大小（以字节为单位）	字段取值	说　明
0	长　度	1	数字	描述符的长度（以字节为单位）。
1	描述符的类型	1	常数	配置值＝02h。
2	总长度	2	数字	用于配置的返回数据的总长度。包括所有被返回的用于配置的描述符（配置描述符、接口描述符、端点描述符、类或设备供应商自定的描述符）加在一起的长度。
4	接口的数　目	1	数字	配置所支持的接口数目。
5	配置值	1	数字	作为 Set Configuration 的一个参数选择配置值。

续 表

字段编号	字段名	字段大小（以字节为单位）	字段取值	说　明
6	配　置	1	索引	描述配置的字符串描述符的索引。
7	属性	1	位图	配置特性： D7 总线供电； D6 自供电； D5 远程唤醒； D4...0 保留（复位为 0）。 设备在运行的时候采用的是总线供电还是本地电源供电可以通过使用 Get Status 设备请求来确定。 如果设备配置支持远程唤醒的话，D5 位就被置为 1。
8	最大供电量	1	毫安	当设备完全可操作的时候，在这种特定配置下总线供电的 USB 设备的最大耗电量。以 2 毫安为单位（例如，50＝100mA）。

接口描述符

接口定义了一套端点，一个给定的类设备通常是基于永久设备类的特征进行操作的。一些设备可以定义一个简单的接口，但是某些设备可以要求几个接口。在表 14-5 列出了接口描述符的格式和定义。下面的部分进一步描述了接口描述符的字段，从作者的观点来看它们需要被申明。

接口数目和可选的设置

在接口描述符里的接口数目和可选设置的字段用于支持 USB 说明书中所支持的可选设置特征。设备可以在相同的配置下定义可选的特征。它的意图是在正常地操作中允许对一个配置进行调整，在初始化的配置完成以后。一个支持可选设置的设备将包含一个或多个附加的接口和端点描述符，同样的接口，但是包含可选的设置。

举个例子，考虑图 14-1 中的描述符树，注意到所有的 3 个描述符都有一个“接口号”0，指出每一个都为接口 0 定义了设置值。然而，每一个接口描述符的可选设置字段都是不同的。在配置过程中，可选设置 0 是缺省使用的。其他的设置（1 或 2）可以在配置以后被选择，用于调整配置。主机软件通过“Set Interface”请求（见表 15-2）用可选的设置值选择可选的接口。

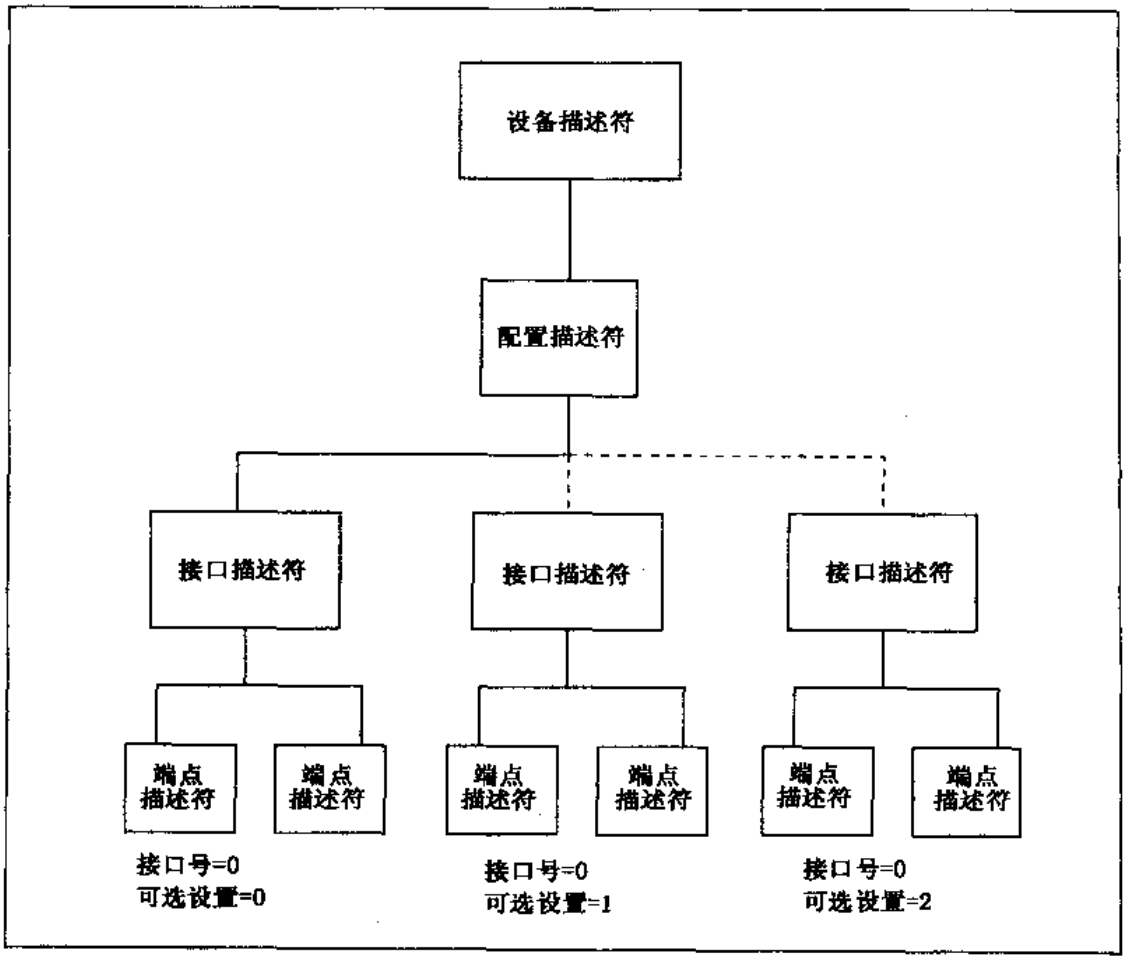


图 14-1　包含可选接口设置的描述符树

当发出“Get Configuration”请求后，设备就会返回每一个主要的接口（可选设置=0）接着是和它相连的端点名以及每一个可选的端点。在这种情况下的配置过程中，主机软件可以检测到可选的设置是存在的，但是它们在配置的过程中被忽略了。

端点的数目

支持接口所要求的端点数目在接口描述符的 4 号字段被指定。这个值列出了除了端点 0（缺省端点）之外的端点的数目。端点的数目也不包括和可选设置相关的端点。

接口类和子类

这些字段包含数值，指出一个给定的设备类和子类，允许软件定位相关的类驱动程序，它们控制和接口相关的端点。这些值是由专门的设备类说明书定义的。举个例子，音频类代码是 01h，它的子类代码的范围是 01-06h，而且定义了这样的信息，音频信息是否是脉冲编码调制的、杜比环绕的、MPEG1 的，等等。更进一步的细节可以参见音频设备类的说明书。

协议

这个字段和设备类、子类字段一起使用。一些设备类的说明书可以定义和给定的接口相联系的协议编码。例如音频设备类的说明书定义了协议编码，它定义了音频接口是单声道、立体声等。可以查询这些设备类的说明书，以确定协议编码和定义。

表 14-5　　接口描述符的定义

字段编号	字段名	字段长度	字段取值	说　　明
0	长度	1	数字	描述符的长度，以字节为单位。
1	描述符的类型	1	常数	接口描述符的类型。
2	接口数	1	数字	接口的编号。该值标识出本配置所支持的并发接口的队列的索引，它的最小值是 0。
3	可选设置	1	数字	用于为在前面的的字段标识出来的接口选择一个可选设置的数值。
4	端点数目	1	数字	本接口所使用的端点数目（0 端点出外）。如果这个字段中的数值是 0，那么这个接口仅仅使用端点 0。

续表

字段编号	字段名	字段长度	字段取值	说　　明
5	接口类	1	类	类代码（由 USB 指定）。 如果这个字段被置为 0，那么这个设备就不属于任何一个以定义的 USB 设备。 如果这个字段的值是 0xFF，那么这个接口类就是由设备供应商自定的。除了这两个值之外所有的数值均是由 USB 分配的。
6	接口子类	1	子类	子类代码（由 USB 指定）。 这个代码受到接口字段中取值的限制。 如果接口类字段被置为 0，那么这个字段也必须置为 0。如果这个接口类字段不是 0xFF，那么所有的数值都由 USB 分配。
7	接口协议	1	协议	协议代码（由 USB 指定） 这些代码由接口类和接口子类字段的取值限制。如果一个接口支持类特定的请求，那么这个代码就标识出设备所使用的协议，该协议由设备类说明书加以定义。 如果这个字段被置为 0，那么这个设备就不在这个接口上使用任何特定的类协议。如果这个字段被设置为 0xFF，那么该设备就在这个接口上使用一个厂商自定的协议。
8	接口	1	索引	字符串描述符的索引。

端点描述符

端点描述符定义了在一个给定的设备里实现的实际寄存器。表 14-6 列出了一个端点描述符的格式和定义。这些描述符定义了每一个寄存器的功能和特定的信息，例如：

表 14-6　　端点描述符的定义

字段编号	字段名	字段长度	字段取值	说　明
0	长度	1	数字	描述符的长度，以字节为单位。
1	描述符的类型	1	常数	端点描述符的类型＝05h。
2	端点地址	1	端点	描述符所描述的 USB 设备上的端点地址。这个地址按以下的方式编码： 位 0...3 端点号； 位 4...6 保留，置为 0； 位 7 传输方向，对于控制端点可忽略； 0 OUT 端点； 1 IN 端点。
3	属性	1	位图	这个字段描述了端点的属性，当它进行配置的时候，采用下面的配置值： 位 0...1 传输类型： 00　控制传输； 01　同步传输； 10　块传输； 11　中断传输。 所有其他位均保留。
4	最大包大小	2	数字	当配置完成以后，本端点可以接收或者发送的最大信息包的大小。 对于同步端点而言，该值用于保留进度表中的总线时间，这是每一个时间片传输数据都需要的。在传递数据的时候，这个管道实际上可以使用比保留带宽更少的带宽。如果有必要的话，设备通过正常的，非 USB 定义的途径报告实际使用的带宽。
6	间隔	1	数字	数据传输的时候，查询的时间间隔，以微秒为单位。 对于块传输端点和控制传输端点而言，这个字段是无效的。对于同步端点而言，这个字段必须被置为 1，对于中断端点来说，它的范围是 1-255。

- 端点要求的传输类型；
- 传输的方向（IN 或 OUT）；
- 带宽需求；
- 查询间隔。

配置软件必须确定 USB 是否能够支持端点描述符所指定的传输，传输所要求的带宽大小在 MaxPacketSize 字段被指定。如果端点的带宽需求超过了 USB 的所能提供的带宽，那么设备就不会被配置，而且用户会得到设备没有配置的通知。

需要注意的是在某些情况下，可以用两个或更多的描述符描述一个寄存器。例如，如果要在设备里实现一个输入/输出寄存器，就必须建立端点描述符用于双向传输数据，除非端点被定义为一个控制端点。控制传输是 USB 传输中仅有的支持双向数据流的传输方式。所有其他的传输方式都是单向的。

例如，如果一个输入/输出寄存器用于和海量存储设备之间实现双向传输，那么寄存器就必须有两个相应的端点：一个用于把数据传输到寄存器，另外一个用于从寄存器中读取数据。基于这两个端点的描述符，配置软件将会建立一个 IN 数据块通信管道和一个另外的 OUT 数据块通信管道，用于海量存储类的设备。

设备状态

表 14-7 列出了设备在配置过程中的状态。设备状态列表中的第一行从左到右是状态变化的典型的顺序。但是要注意的是一个设备可以从任何一个其他状态进入挂起状态。

连接的状态

表 14-7 的第一列指出设备没有和 USB 相连。在这种情况下设备的状态是未知的。设备如果有自己的本地电源的话就可以上电，否则就需要从总线那里获得电源。

供电的状态

一旦设备和 USB 相连以后，那么可以由集线器供电也可以不由集线器供电，就是说，某些集线器有供电开关，而有的则没有。当设备连接上以后，如果端口没有上电，那么主机软件就必须给端口加电，检测设备并激活端口。在这种情况下，设备可以被供

电，但是不能超过 100 毫安。

缺省状态

在收到一个复位信号以后（D+和 D-驱动程序都被集线器驱动到低电平，并且持续 10 毫秒）。驱动程序现在对它的缺省地址 0 作出响应。配置软件可以用地址 0 存取设备的描述符，通过缺省的控制管道（总是在端点 0 处）这个设备还能接收控制写，允许一个地址由配置软件进行指定。

表 14-7　　设　备　状　态

连接	供电	缺省	地址	配置	挂起	状　态
否	—	—	—	—	—	设备没有连接到 USB，所以其他的属性都是不重要的。
是	否	—	—	—	—	设备连接到了 USB，但是没有被上电，所以其他的属性都是不重要的。
是	是	否	—	—	—	设备连接到了 USB，而且也已经上电了，但是还没又被复位。
是	是	是	否	—	—	设备连接到了 USB，已经上电，复位动作也已经完成，但是还没有被分配一个唯一的地址。设备在缺省地址处作出响应。
是	是	是	是	否	—	设备连接到了 USB、已经上电、复位动作和地址分配动作也已经完成，但是设备还没有被配置。

续 表

连接	供电	缺省	地址	配置	挂起	状　态
是	是	是	是	是	否	设备连接到了 USB、已经上电、复位、地址分配和配置工作也已经完成，也不处于挂起状态。主机现在可以使用设备所提供的功能。
是	是	是	是	是	是	设备至少已经连接到了 USB 上、已经上电、复位、处于挂起状态、有一个唯一的地址，并且已经配置完成。但是因为设备处于挂起状态，所以主机不能使用设备提供的功能。

地址状态

在从控制软件处收到一个“Set Address”请求之后，设备就会进入它的地址状态。现在它仅仅对应于它的新地址，非 0 地址。

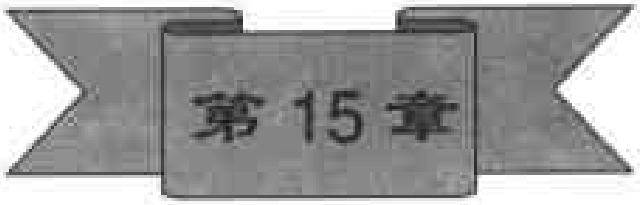
配置状态

一旦一个配置软件已经确定了设备可以获得足够的功率和总线带宽，那么它就可以对设备进行配置。配置值是在所选择的配置描述符里被定义的，它已经被写到设备内，并通过“Set Configuration”请求完成配置。现在这个设备可以获取最大所需的电流，并且已经为 USB 设备驱动程序访问做好了准备。

挂起状态

当总线保持空闲状态超过 3 毫秒以后，设备驱动程序就会进入挂起状态。在挂起状

态下，设备不能消耗超过 500 微安的电流。关于设备挂起的进一步信息可以参见第 10 章。



设 备 请 求

✎ 上一章

上一章讨论了和 USB 相连的非集线器设备。还讨论了设备描述符以及其他和设备配置相关的设备描述符以及其他特性和特征。

✎ 本 章

本章讨论了 USB 设备所要求的标准请求。每一个设备类都可以定义类的特定请求，这些请求在相关设备类的章节加以了讨论。

✎ 下一章

主机软件由三层组成：USB 设备驱动程序、USB 驱动程序和主控制器驱动程序。下一章将讨论每一个层的作用，并且将描述编程接口的需求。

🕮 概述

所有的 USB 设备都必须响应不同的请求，称为标准请求。这些请求用于配置一个设备，控制 USB 接口的状态，还有其他大量的特征。设备请求通过控制传输机制由主机发送。在配置之前，设备对它的缺省地址 0 作出响应。这就允许了配置软件在配置过程中用设备地址 0 要求获取任何设备配置描述符的内容（从端点 0 处）。

此外，设备还可以支持类的特定请求，除了集线器设备之外，这些类的请求由设备类的规范说明文档定义。它们并不包含在规范说明书的主体部分。类似的，一个设备可以支持设备供应商自定义的请求，这些请求和设备供应商对设备的具体实现相关。

控制传输用于传输设备请求，最少要由建立阶段和状态阶段组成，但是也有可能包含数据阶段，这完全取决于被执行的请求的类型。建立阶段是由 SETUP 事务处理组成

的，见图 15-1。一个 SETUP 事务处理的 8 字节的数据定义了主机发送的请求的类型。本章仅仅讨论标准设备的请求。可以参见关于特定的设备类内容的章节以获得和类特定请求相关的进一步信息。

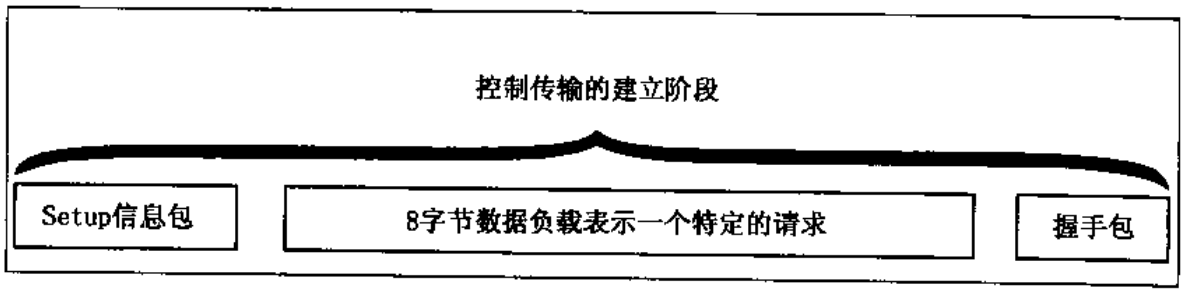


图 15-1　指定被执行的设备请求的建立事务处理的格式

标准设备请求

当控制传输启动以后，事务处理的建立阶段就指出了设备需要执行的特定请求。建立阶段的格式见表 15-1，而图 15-2 则为每一个标准请求类型定义了建立数据的内容。

表 15-1　　在 SETUP 事务处理阶段数据负载的格式

字段编号	字段名	字段长度	字段取值	说　明
0	请求类型	1	位映射	请求的特征： D7 数据传输方向： 0=主机到设备； 1=设备到主机； D6..5　类型（h）： 0=标准； 1=类别； 2=厂商； 3=保留。 D4..0 接收方： 0=设备； 1=接口； 2 端点； 3=其它。 4..31=保留。
1	请求	1	值	特定请求。
2	值	1	值	根据请求变化的字段，以字为单位。
4	索引	2	字段编号索引	根据请求变化的字段，一般用于传送字段编号的索引。
6	长度	2	计数	如果某个传输要求数据阶段，那委它就表示传输的字节数。

表 15-2　　标准设备请求

请求的类型	请　求	取　值（2字节）	索　引（2字节）	长　度	数　据
00000000B 00000001B 00000010B	CLEAR_FEATURE(01h)	特征选择符	0 接口端点	0	无
10000000B	GET_CONFIGURATION (08h)	0	0	1	配置值
10000000B	GET_DESCRIPTOR(06h)	描述符的类型和索引	0 或者语言 ID	描述符的长度	描述符
10000001B	GET_INTERFACE (10h)	0	接　口	1	可选的接口
10000000B 10000001B 10000010B	GET_STATUS (00h)	0	0 接口端点	2	设备接口或者端点状态
00000000B	SET_ADDRESS (05h)	设备地址	0	0	无
00000000B	SET_CONFIGURATION (09h)	配置值	0	0	无
00000000B	SET_DESCRIPTOR (07h)	描述符的类型和索引	0 或者语言 ID	描述符长度	描述符
10000000B 10000001B 10000010B	SET_FEATURE (03h)	特征选择符	0 接口端点	0	无
00000001B	SET_INTERFACE (11h)	可选设置	接　口	0	无
10000010B	SYNC_FRAME (12h)	0	端　点	2	时间片数目

注意在表 15-1 中，0 字节的第 5 和第 6 位是 00b，表示这个特定的请求是一个标准请求字节。第二个字节（请求字段）定义了将会执行哪一个标准请求，但是对其他字段的定义则是取决于特定的请求。例如，“CLEAR_FEATURE”请求把“数值”字段定义为特征选择符。表 15-3 列出了请求所适用的特征，下面的部分描述了每一个标准请求。

注意到表 15-2 的最后一列的标题是“数据”，它指出该请求在控制传输过程中是否需要一个数据阶段。例如，“Get Configuration”请求使用数据阶段传输配置值。但是很多请求可以不经过数据阶段就执行。

如果请求包含的字段含有非法的值或者不支持的数值，那么请求到达的端点将会自动进入停止阶段。主机软件必须用 Clear Stall 请求清除停止阶段。尽管控制端点处于停止阶段，但是它必须继续接受建立阶段，。如果缺省的控制端点没有对 SETUP 事务处理作出响应，那么这个设备就必须复位以清除这个状态。

设置/清除特征

“Set Feature”和“Clear Ferture”请求提供了一个激活或禁止一套特征的办法，它们由特征选择符中的值来加以定义，对于标准设备请求定义了两个特征，参见表 15-3。

表 15-3　　特征选择符

特征选择符	接 收 方	数　值
DEVICE_REMOTE_WAKEUP	设　备	1
ENDPOINT_STALL	端　点	0

设备的远程唤醒

某些设备可以被设计用于在全局挂起事件的情况下唤醒系统，或者唤醒一个集线器的端口，那个端口被选择性地挂起（关于挂起的详细内容参见第 10 章）“Set Feature”请求和在设备远程唤醒被选择的情况下，使设备发信号唤醒集线器。如果清除设备远程唤醒请求，那么就不允许设备给集线器发送远程唤醒信号。一个设备是否能够发送远程唤醒信号是通过 Get Status 请求报告给软件的。

端点停止

软件可以停止一个给定的端点或者清除停止状态。“Set Endpoint Stall”和“Clear Endpoint Stall”请求定义了在设备内的哪一个端点是通过 SETUP 事务处理的“索引”字段的目标。每一个端点都定义了一个停止位，它指出这个端点当前是否是停止的。停止位和“Get Status”请求一起被读取。

设置/获取配置

主机软件通过选择一个或多个在设备描述符内定义的配置对设备进行配置，给设备分配相应的 8 位配置值。“Set Configuration”请求是软件用来安排配置值的。相反的，软件可以用“Get Configuration”请求查询哪一个配置分配给了设备。注意配置值在控制传输的数据阶段被配置。

设置/获取描述符

设备描述符使用“Get Descriptor”请求被读取。“Set Descriptor”请求是可选的，它被那些提供增加或更新存在的设备描述符途径的设备所支持。在任何一种情况下，一个特定的描述符和在描述符内的索引可以作为目标。标准的“Set/Get Descriptor”请求仅仅支持设备、配置、和字符串描述符。每一个这样的描述符都有一个相关的值定义描述符的类型，见表 15-5。接口和端点描述符只能通过获取配置描述符请求来读取，而且和

选择的配置描述符一起返回。

主机软件一开始就访问设备的描述符，确定一个设备的缺省控制管道（端点 0）所支持的最大数据包。主机软件可以通过发送一个“Get Descriptor”请求确定端点 0 的最大数据包，它的值列于表 15-4。“数值”字段的高字节包含了描述符的类型（01＝设备描述符），低字节包含了描述符的索引，设备将会从指定的索引开始返回设备描述符的内容。描述符的内容在控制传输的数据阶段被返回。

表 15-4　　在 Get Descriptor 请求阶段 SETUP 事务处理的内容

请求的类型	请　求	数值（2 个字节）	索引（2 个字节）	长度（2 个字节）	数　据
10000000B	GET_DESCRIPTOR(06h)	01h=设备描述符 07h＝索引	0 或者语言 ID	描述符的长度	描述符的内容

注意，如果一个字符串描述符被指定，索引值就将包含语言 ID。对所有的描述符类型来说，这个索引值被清零。

表 15-5　　可以通过标准 Get/Set Descriptor 请求指定的描述符类型

描述符类型	取　值
设　备	1
配　置	2
字符串	3

设置/获取接口

Set 和 Get Interface 请求用来指出在一些选择的配置中使用的可选设置。就是说，一些设备可能会有一些接口包含相互排斥的设置。在这种情况下，提供一些用于选择可选的设置的值，这个值必须用“Set Interface”请求选择。主机可以通过使用“Get Interface”请求确定当前选择了哪一个可选的设置。

获取状态

主机可以通过 Get Status 请求从设备处获得状态信息。状态信息在控制传输的数据阶段被返回。主机软件可以从三个和设备相关的独立的层获得状态信息。

- 设备状态——提供了适用于整个设备的全局状态；
- 接口状态——返回所有的 0。接口状态被定义为保留；
- 端点状态——提供了关于被选择的端点的信息。

请求的接收方在 0 字节的请求类型字段被指出，参见表 15-1。如下面所说明的，对接收方来说状态信息以小末尾的格式被返回。由于接口的状态信息是保留的，所以只含有设备和端点的状态信息。

设备的状态

表 15-6 表示了当一个“Get Status”请求被作出的时候，主机返回的信息，在这里设备是作为接收方的。

表 15-6　　在 Get Status 请求期间返回的设备状态信息

7	6	5	4	3	2	1	0
保留（置为 0）						远程唤醒	自供电

15	14	13	12	11	10	9	8
保留（置为 0）							

自供电位

自供电位反映了设备当前是自供电的还是总线供电的。这个字段不可以由“Set Feature”和“Clear Feature”请求改变。

远程唤醒位

这个位反应了当前这个设备处于活动状态还是禁止状态，用于产生恢复信号唤醒集线器的端口。缺省的设置是这个位在复位以后被清 0，从而禁止远程唤醒。主机软件可以用“Set and Clear Device Remote Wake Feature”请求设置和清除这个位。

端点状态

表 15-7 列出了当一个端点被作为请求的接收方返回信息的时候。端点数在 SETUP 事务处理的索引字段被指出。低字节包含它所支持的端点号和传输的方向（IN 或 OUT）。端点状态信息仅仅由停止位组成，所有其他位的位置被定义为保留的。

停止位指出端点当前是否是停止的。这个位可以用 Set 和“Clear Endpoint Stall Feature”进行改变。停止位在一个“Set Configuration”和“Set Interface”请求后总是返回到0。

表 15-7 在 Get Status 请求期间返回的端点状态信息

7	6	5	4	3	2	1	0
保留（置为0）							停止

15	14	13	12	11	10	9	8
保留（置为0）							

同步时间片

这个请求是由同步端点使用的，它使用隐式同步。为了保持同步，同步端点可能需要跟踪时间片编号。根据重复的特定形式，同步端点的事务处理在大小上有所变化。主机和端点必须与重复模式开始的那个时间片保持一致。主机用 Sync Frame 请求指出重复模式从哪一个时间片开始。Sync Frame 请求的数据阶段包含了模式开始的时间片号。收到时间片的编号以后，设备就可以开始监视每一个在 SOF 期间发送的时间片编号。

第四部分　USB 主机软件

第十六章 USB 主机软件

USB 主机软件

✎ 上一章

上一章讨论了 USB 设备所要求的标准请求。每一个设备类都可以定义特定的类请求。这取决于设备类的定义。

✎ 本　章

主机软件由三种类型的软件组成：USB 设备驱动程序、USB 驱动程序和主机软件控制器的驱动程序。本章讨论每一层的功能，然后描述它们的编程接口的需求。

✎ 下一章

下一章将介绍设备类的概念，而且将对每一个设备类进行讨论，它们在作者写这本书的时候已经定义好了。

🕮 USB 软件

主机软件在 USB 设备驱动程序（或者说客户驱动程序）和必须与之通信的设备之间提供了一个接口。USB 驱动程序不知道 USB 的具体实现机制。也就是说它们不知道 USB 的特征、能力或限制，也不知道该设备是 USB 设备。所以，USB 的主机软件必须接收 USB 设备驱动程序的传输请求，然后基于 USB 的需求执行请求的传输。下面是 USB 系统一般都支持的功能。

- USB 接口控制；
- 配置服务；
- 总线和设备管理；

- 电源控制；
- 设备数据存取；
- 事件通知；
- 收集状态和活动的统计数据；
- 错误发现与处理。

USB 软件是基于设备构架的，它由 USB 的规范说明书定义。这个构架描述了每一个软件元素的逻辑视图。下面的部分将会讨论 USB 软件层和它们的 USB 设备的视图之间的关系，参见图 16-1。

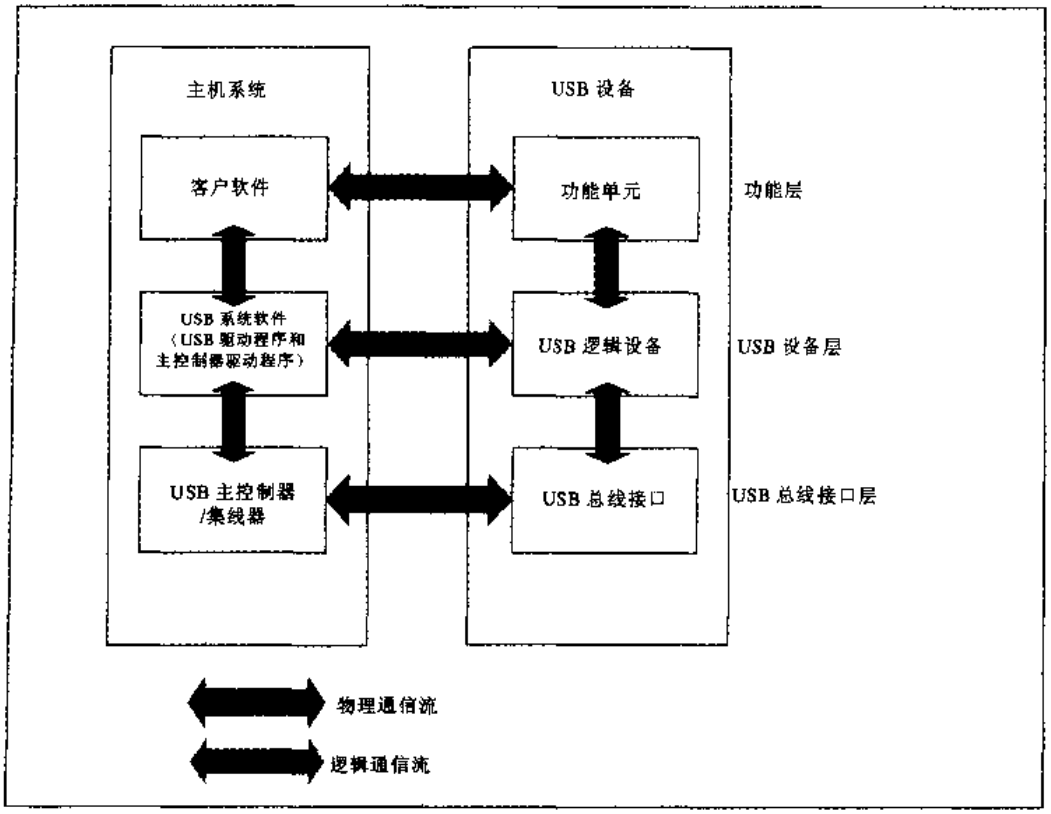


图 16-1　设备框架——硬件的软件视图

功能层

在 USB 上面执行的事务处理是由 USB 设备驱动程序（客户驱动程序）启动的。在配置过程中，集线器的客户端存取总线，当访问其他 USB 设备的时候，客户驱动程序可能是专用于某一个类别的，也可能是厂商自定的。不管是哪一个 USB 的设备驱动程序，如果它希望访问一个给定的 USB 设备，它都必须使用主机软件请求它的 IO 传输在 USB 上执行（通过一个 IO 请求包或者 IRP）。这些客户应用程序仅仅知道它们希望操纵的设备的接口（由一个端点的集合组成）。所以一个 USB 客户驱动程序对 USB 的可见

度仅仅限于：

- 在它们的设备中的接口；
- 类特定的描述符，它们已经被预定义，用来帮助它们确定它们的接口的特定的特征；
- 主机软件所提供用于访问和控制它们的功能单元的机制。

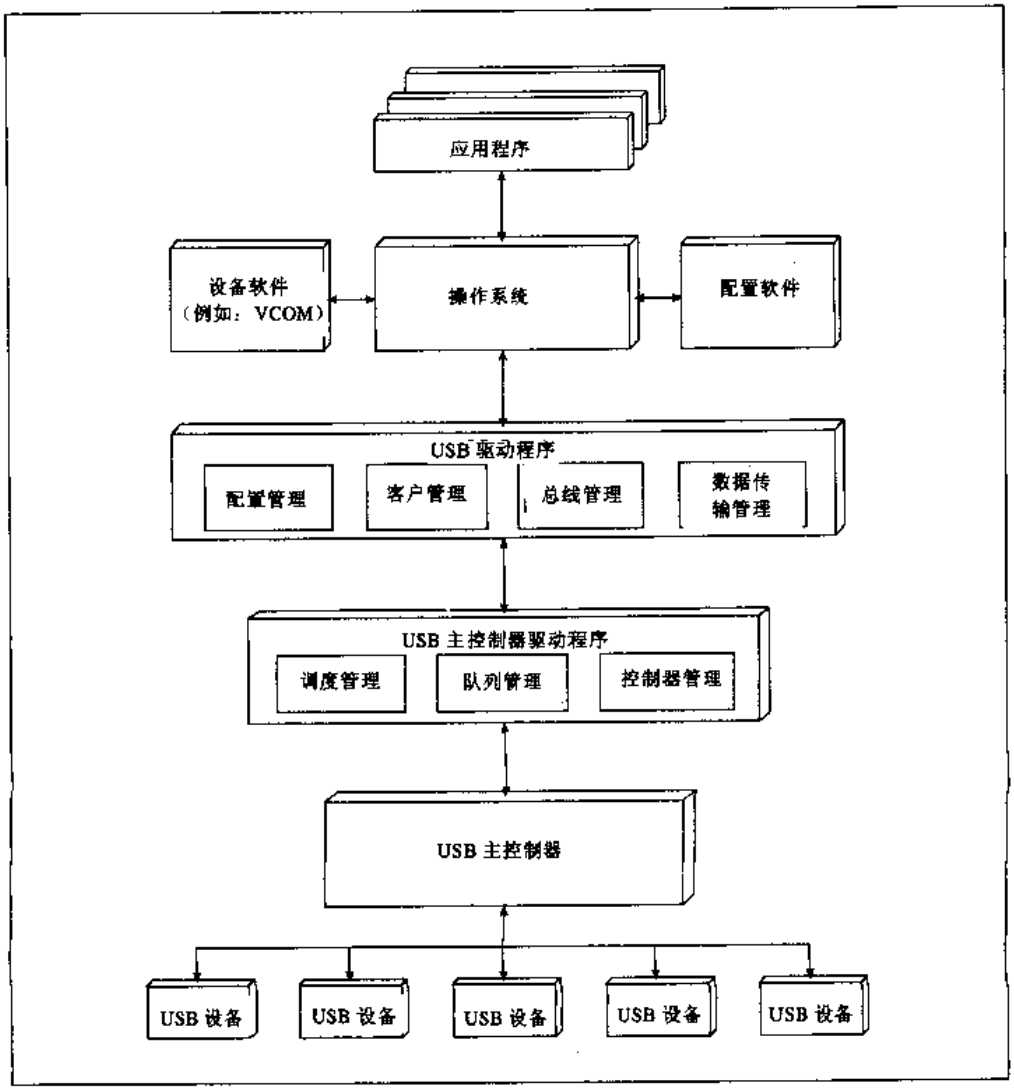


图 16-2　软件层次

设备层

客户启动的传输必须根据 USB 的特征和目标 USB 设备的能力来执行。USB 的主机软件通过提供它们可以启动传输和控制它们的设备的服务对 USB 的客户应用程序提供支持。

主机软件也保证总线可以支持所有的和 USB 相连的设备。这些描述符提供了必要的信息确定哪一个驱动程序可以使用这个设备，需要多少总线带宽。在这种情况下，主机软件可以建立起通信管道，以后当客户访问这些接口的时候就可以使用这些管道。

主机软件必须把 IRP 传递给主控制器的驱动程序（HCD），它知道主控制器的设计。HCD 把 IRP 以它明白的格式传递给主控制器。

接口层

本层由主控制器（包括根集线器）、USB 数据线和设备的 USB 接口组成。这些组件实际上把数据和控制信息传递给 USB 的设备，或者把它们从 USB 设备那里传递到主机。

软件组件

三个软件组件组成了主机的 USB 解决方案，和每一个层相关的主要功能包括以下部分（但是请注意，规范说明书中并没有对功能做详细的划分）。

- USB 客户驱动程序——客户驱动程序是软件，它控制一个给定的 USB 功能设备。连接到 USB 设备的每一种类型的功能单元都必须要有客户驱动程序。这些功能单元不知道和 USB 传输机制相关的细节，而且必须依赖于 USB 的主机软件才能管理它们基于 USB 的能力和限制的传输请求。客户驱动程序的理想实现是基于设备类而定义的。客户驱动程序把 USB 设备看作是一个可以被访问的端点的集合，它可以被控制并与它的功能单元进行通信。
- USB 驱动程序（USBD）——USB 驱动程序知道设备的需求（通过设备描述符获知），也知道 USB 的能力。在这种情况下，USBD 必须把 IRP 划分成 USB 和设备需要大小的块。USBD 确保每一个设备能分配到它所要求的 USB 资源，这样它就可以支持 USB 设备配置。它也能在配置的过程中为每一个检测到的端点建立的通信管道，但是只有必要的 USB 带宽才能被分配到。USBD 提供了一个编程接口，称为 USBDI（USB 驱动程序接口），给客户驱动程序一种方式，用于传输请求，传输的方向可以是来自或前往 USB 的功能单元。大量的客户服务是由 USB 的驱动程序提供的，它帮助 USB 的客户控制和访问它们的功能单元。
- USB 的主控制器驱动程序——已经定义了 USB 主控制器的两种实现方式：开放主控制器和通用主控制器。如果主机软件对两种控制器都支持的话，那么就必须实现两个主控制器的驱动程序。主控制器的驱动程序提供了对 USB 的低级支持。通过把 IRP 转换成为单独的事务处理后在 USB 上执行。在 USB 驱动程序和主控制器的驱动程序之间的编程接口的实现是很特殊的。并且它的编址方式不由 USB 规范说明书定义。

注意，USB 的规范说明书没有定义 USBD 和 HCD 的确切的职责划分。后续的章节定义了它们的编程接口的基本需求，但是这些接口的精确的实现是依赖操作系统的。

USB 驱动程序（USBD）

USB 驱动程序包含下面的功能：

- 配置管理；
- 总线管理（跟踪和分配总线的带宽）；
- 数据传输管理；
- 提供客户服务（USBDI）。

每一个 USBD 功能都在下面加以说明。

配置管理

主机软件必须支持 USB 设备的自动配置。包含在软件中的实际的配置元素和接下来所进行的配置步骤是不同的。配置从根集线器开始，而且一次进行一个端口，直到所有的端口被检测到和配置完成。

配置行为依赖于对每一个设备内部的缺省控制端点的访问。在配置开始之前，USBD 初始化缺省的控制管道。在配置过程中，这个管道被反复使用。实际的软件组件包含在配置中，随实现的不同而不同，但是从概念上来说，包含的组件可以被认为是：

- 集线器客户驱动程序；
- 配置软件；
- USBD。

USB 配置要求的元素

USB 配置包括配置 3 个不同的元素：

- 设备配置——设备配置包含访问设备的描述符，确定设备所需要的 USB 资源（例如，电源和总线带宽），通过为每一个端点建立通信管道进行资源的分配，为每一个选择的的配置安排配置值，设备配置在本书的第 3 部分讨论过了。
- USB 配置——在设备配置过程中建立的通信管道在被初始化之前是不能被设备驱动程序使用的。在初始化的过程中，设备驱动程序必须为管道设置规则。例如，必须设置服务间隔、客户程序能使用的最大数据传输量。一旦管道的规则

建立以后，它就可以被使用了。

- 功能单元的配置——一个给定 USB 设备的驱动程序可能要完成附加的配置，这些配置可能是 USB 软件完全不知道的。这种配置一般和一个特定的设备类型相关，或者可能是一个厂商自定的设备或驱动程序的一部分。

一旦建立起了一个配置，USBD 就允许对配置进行修改，修改是通过调整和一个给定的接口相关的设置，或者通过选择一个完全不同的配置而实现的。

分配 USB 的资源

主机软件必须能够确定某个和 USB 相连的设备是否可以基于当前可以利用的 USB 资源进行操作。主机软件跟踪两个 USB 资源

- 电源；
- 总线带宽。

在配置操作完成之前，由软件负责确保设备有必要的资源可以使用。

☑ 供电验证

某些设备可以要求比在一个给定的端口上所能提供的更多的 USB 总线功率。当一个高功率的设备连接到一个总线供电的集线器端口的时候，就有可能发生这种情况。通过读取一个集线器标准描述符，配置软件可以确定一个给定的 USB 端口所能够提供的功率的数量（允许的最小电流是 100 毫安）。软件也能确定和这个端口相连的设备要求的电流数量，这是通过读取它的描述符做到的。注意，描述符可能被读取，因为一个设备不能使用多于 100 毫安的电流（确保可以使用的电流大小）。

如果设备要求比端口提供的更多的电流，那么软件就不能配置这个设备，并且要向用户报告供电不足。

☑ 跟踪和分配总线带宽

在一个用于和设备端点进行通信的管道建立起来之前，USBD 必须确定 USB 是否可以支持它。每一个端点描述符都包含了支持端点所需要的总线带宽。端点描述符指出的总线带宽仅仅包含数据的大小，而不包括额外开销的时间。所以 USBD 必须计算出在总线上传输数据所需要的执行时间。一旦计算出来所需要的总执行时间以后，就必须检查时间片的安排，以确定是不是要支持新的管道。

为了计算一个给定的管道所需要的总线带宽，必须知道以下几个参数：

- 数据的字节数——这个信息是从 MaxPacketSize 字段读取的，它在端点描述符内；
- 传输类型——同步和中断传输要求带宽有保证，而控制传输有 10%的保留带

宽。

块传输在一个时间片内没有可保证的带宽。传输类型也确定了和事务处理相关的额外的开销。例如，同步传输不伴随握手包。额外开销包括：

- 令牌包——同步位（8）＋PID（8）＋Address（11）＋CRC（5）＋EOP（2）＝34 位；
- 数据包额外开销——同步位（8）＋PID（8）＋CRC（16）＋EOP（2）＝34 位；
- 握手包（如果允许的话）——同步位（8）＋PID（8）＋EOP（2）＝18 位。

传输类型被编码进端点描述符的属性字段。

- 主机恢复时间——主控制器从上一次传输中恢复和为下一次传输做好准备所需要的时间。这个参数是主控制器特有的；
- 集线器低速建立——如果事务处理的目标是低速设备，那么发送前导包所需的时间和激活低速端口所需要集线器延迟就必须包括进来；
- 位填充时间——位填充时间必须也是可计算的。由于位填充是一个数据流的功能。它对主机软件来说是不可知的，最坏的情况是位填充包含了理论上最大数目的附加位（1.1167*8*字节数）；
- 拓扑深度——发送包到达一个设备并且接收到响应所需要的往返时间取决于传输必须经历的数据线段的数目。最大的往返时间延迟是 16 位。当计算事务处理所需要时间的时候，软件可以使用最坏情况下的值，也可以确定目标设备位于拓扑中的什么位置，并基于传输必须经过的数据线段的数目确定延迟时间。

一旦支持管道所需要的整个总线时间被计算出来后，如果管道可以被支持，那么 USBD 就建立这个管道，并且把事务处理加到时间片的进度表中去。

☑ 总线带宽回收

由于在很多的实例中，带宽分配是基于最坏的情况的（例如位填充），在一个时间片期间，在所有安排好的事务处理完成以后，就是总线带宽的回收了。主控制器可以回收剩下的带宽，执行附加的事务处理，以达到充分利用总线的目的。总线可以为控制传输和块传输重新分配的带宽。因为同步和中断传输在前面已经被安排好了，而且它们就已经完成了。一个主控制器如何实现总线带宽的重新分配是和具体的实现相关的。

🕮 总线管理

USBD 必须允许客户成为 USB 上的主客户。一个主客户可以调整在一个时间片内的位的次数。主客户可以给当前的时间片加上或减去一个位的时间以调整 SOF 间隔。某些端点要求它们的内部时钟和基于 1 毫秒的 SOF 的 USB 的数据传输率保持同步。然而，

一个设备可能不能使它的内部时钟和 1 毫秒的 SOF 保持同步。一个客户驱动程序访问这样一个设备可以调整 SOF 的间隔，这样它就可以使 SOF 和它的内部时钟保持同步。

注意，USBD 仅仅支持一个主客户。如果某个客户不能变成主客户，那么它必须能够通过调整它的数据流和主机保持同步。

数据的传输管理

在配置过程中建立的每一个管道都是为一个特定的端点定义的，这个端点在一个设备的功能接口内。一个给定的接口和在这个接口内的每一个端点都是由一个客户软件驱动程序加以管理的。在管道可以被使用之前，客户程序必须对他进行初始化。在管道初始化期间，客户要指出管道的规则。规则定义了每一个 IRP 传输的数据的数量和最大的服务间隔（如果需要的话）。客户程序也可以从 USBD 那里要求得到状态信息的通知（例如，完成状态）。客户还必须指出它的内存缓冲区的位置，从端点处读取的数据就被存储到缓冲区那里，或者从数据缓冲区那里读取数据后写到端点处。

USBD 可能需要把 IRP 分成几块，它们可以被端点（缓冲区大小）和与 USB 协议相关的约束支持。

提供客户程序服务

USB 驱动程序可以提供编程接口（USBDI）它被客户软件用来控制和访问 USB 设备。通过这些机制，USBD 提供了大量的客户程序的服务。一个给定的 USB 驱动程序的实现必须提供以下的软件机制。

- 命令机制——命令机制允许客户程序配置和控制 USBD 操作，然后配置和控制一个 USB 设备。一些命令可以利用管道机制；
- 管道机制——管道机制允许 USB 客户程序管理特定的数据和控制传输。这个管道机制不允许直接访问设备的缺省管道，因为它是 USBD 所有的。

管道机制

USB 支持两个基本类型的管道：

- 缺省管道（USBD 所有）——缺省管道在原则上是用于存取设备的配置信息的。USB 驱动程序有责任通过缺省管道建立和管理访问，除非 USBD 使用缺省的管道填充请求的一部分；

- 客户程序管道（消息和流式管道）——一个客户管道是不由 USBD 所有和管理的管道。客户使用管道把信息传递到 USB 功能单元端点或者从 USB 功能单元的端点把数据传递出来。客户程序有责任提供缓冲区，用以支持指定的数据传输率。所有 4 种类型的管道（基于四种传输类型）都是被支持的。

☑ 客户管道的需求

USBD 客户程序管道机制要求支持以下的功能。这些功能给客户提供一个高速/低额外消耗的方法用于数据传输：

- 取消 IRP；
- 对 IRP 排队；
- 管理管道的规则；
- 请求主客户的状态（调整 S.O.F）。

命令机制

命令机制允许一个客户访问和控制 USB 设备。例如，可以对一个设备的数据和控制端点进行读写访问。这些命令机制提供了以下类型的访问/服务。USBDI 必须为下面的行为提供一种机制：

- 接口状态控制；
- 管道可以被复位；
- 管道可以被取消；
- 获取描述符；
- 获取当前配置设置；
- 接上设备（集线器的客户）；
- 断开设备（集线器的客户）；
- 管理状态；
- 发送类的命令；
- 发送设备供应商定义的命令；
- 建立可选的设置；
- 建立一个配置；
- 建立最大包大小；
- 设置描述符。

第五部分　USB 设备类

第十七章　设备类

设 备 类

✎ 上一章

主机软件由 3 层组成：USB 设备驱动程序、USB 驱动程序和主控制器驱动程序。前一章讨论了每一层的角色，还说明了它们对编程接口的需求。

✎ 本 章

本章介绍设备类的概念，讨论了它们在 USB 中的角色。讨论了已经定义的前 5 种类型。对这些类的讨论给读者提供了一些有用的信息，这些信息是关于每个类的定义和它们所使用的 USB 机制。一个对设备类的细节性的讨论需要有对相关领域有彻底的了解，例如电话和音频设备。

✎ 下一章

下一章介绍了 Intel 定义的通用主控制器。讨论了传输机制，详细介绍了通用主控制器驱动程序建立的传输描述符。

🕮 概述

定义设备类的最初想法是为一套有相似的属性和提供相似服务的设备建立一个设备驱动程序。一个给定的类定义可以描述在这个设备类中的特定的设备类型的专门的特征，从而为 USB 设备驱动程序提供了它操纵这个设备所需要的信息。

设备类的定义和一个功能的接口相关，这个接口用来访问和控制专门的一类设备。图 17-1 就是一个 USB 的 CD-ROM 设备，它有两个接口。支持两个或多个功能单元的设

备被称为复合设备，每一个功能单元都要求一个专门的接口。用 CD-ROM 从一个程序磁盘上读取文件的时候，就需要一个海量存储类设备的驱动程序，但是当播放 CD 的时候，就需要一个音频类设备驱动程序。每一种驱动程序都访问端点的集合，这些端点为设备类组成编程和数据接口。

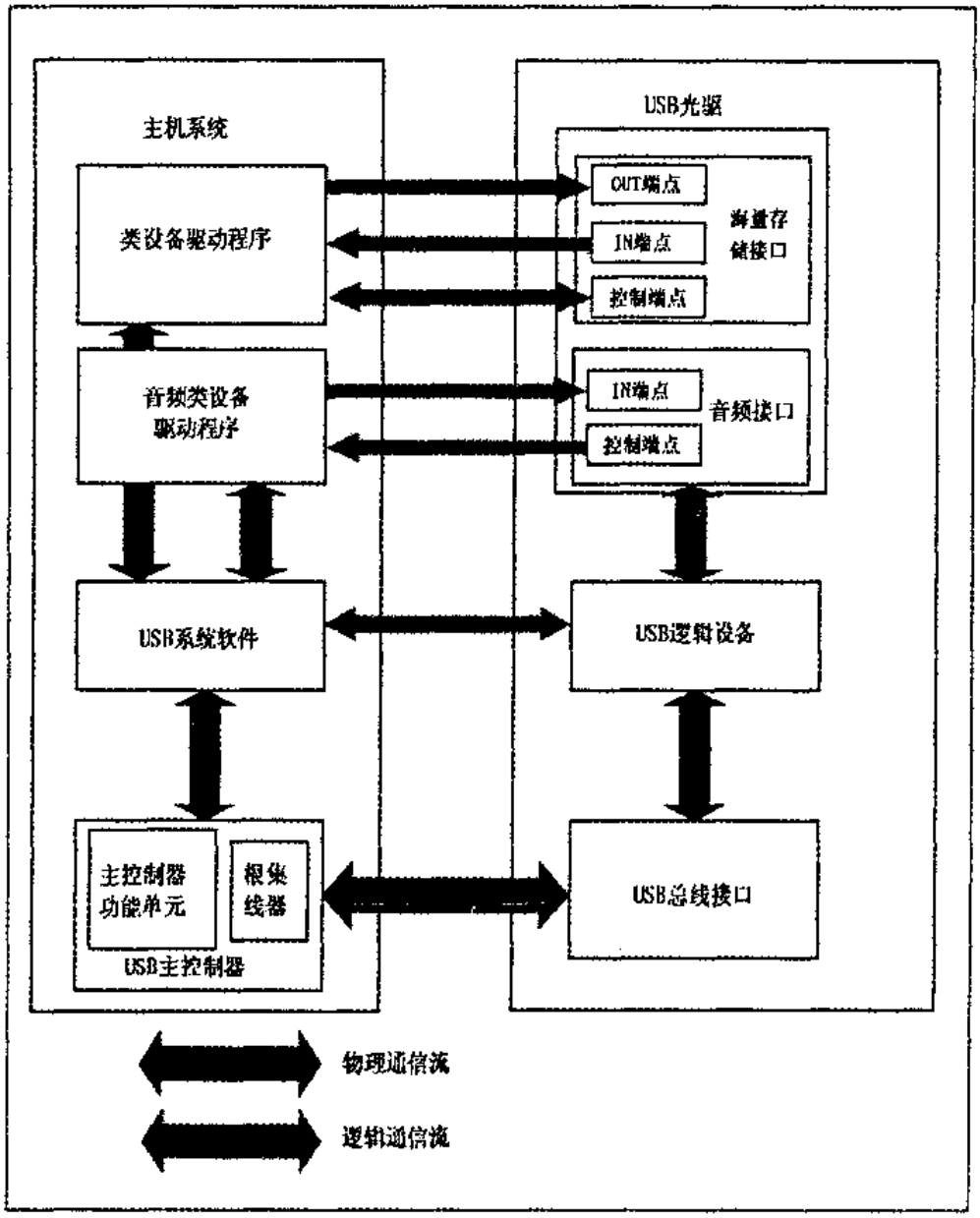


图 17-1　CD-ROM 支持海量存储和音频接口

注意，设备类特定的 USB 组件是功能接口和 USB 设备驱动程序。和设备相关的描述符指出了几项和设备类定义相关的项，包括：

- 设备类的代码字段——来自于设备描述符（如果设备包括一个功能接口），和/或来自每一个功能接口的描述符；

- 子类代码字段——这个字段的定义是设备类特定的；
- 协议字段——这个可选的字段可以为一个给定的设备类或子类定义，用于定义一些设备所支持的编程接口的元素。

这些设备类的代码可以由主机软件用来定位合适的设备驱动程序，这些设备驱动程序用来存取一个设备的功能接口。所有其他标准描述符的信息都和 USB 的特定信息相关，主机软件可以在不知道设备类的定义的情况下解释这些信息。

注意，某些设备类的规范说明书定义了附加的描述符，主机软件并不知道它们。这些描述符是用于特定的类设备驱动程序的，用于检测设备的属性和特性，以便于访问这个设备。

总之，类的定义有助于把一组常用的设备归类，这样就可以使用一个通用的驱动程序，并且允许设备向主机和 USB 设备类驱动程序描述它的能力。类代码为 USB 主机软件提供了一种机制，用来标识相应的设备驱动程序，它被设计为操纵一个给定的 USB 设备的功能单元接口。这些专门的类规范说明书描述了在这个类中的设备所支持的特定的特征和属性，并且定义了 USB 类设备驱动程序使用的控制机制，用于存取和操纵它的功能单元。

设备类

开始的时候定义了 5 种主要的设备类型，它们可以支持 USB 上的大多数常用的功能设备，需要的时候还可以加上其他需要的设备类。开始定义的那几个类如下：

- 音频设备类——实时音频信息接收器的源设备；
- 通信设备类——和电话线相连的设备（非本地网）；
- 显示设备类——用于在软件控制下配置监视器；
- 人机接口设备类——被最终用户操纵的设备；
- 海量存储设备类——用于大量信息的设备（例如，软盘驱动器、硬盘驱动器、磁带驱动器）。

在作者编写本书的时候，设备类的说明书还没有全部完成。因为这些定义还没有完成，所以本书并没有提供关于它们的细节性的信息。

其他的设备类尚处于定义阶段，或者正处于考虑阶段。它们包括：

- 图象设备类——处理图象的设备（静态或动态图象）。这是一个新的组，但是也有可能分为“静止”类和“视频”类；
- 物理接口设备类（PID）——对操作者提供触觉反馈的设备。例如：用不同电阻模拟不同力量的游戏杆，它是从 HID 设备中分出来的；
- 电源设备类——为系统或外围设备提供电源的设备。例如：不间歇电源和智能电池。可以是独立的设备，也可以集成到设备中去；

- 打印机设备类——可能会被合并入存储设备类。

下面的部分将会描述前面所讨论的最初定义的 5 种设备的类的主要特征。

音频设备类

音频类的说明文档定义了标准化的音频传输机制，用于传播和控制数字音频。音频类的一个主要的焦点是音频数据流的同步，保证没有声音的失真。

每一个音频功能单元都有自己的设备接口，用于访问和控制功能单元。音频设备是那些和 USB 兼容音频数据流进行交互的设备。音频设备被分为下面的子类：

- 8 位脉冲编码调制（PCM）音频数据；
- 16 位 PCM 音频数据；
- 16 位杜比环绕数据；
- IEC958 音频编码数据；
- MPEG1 音频编码数据；
- AC3 音频编码数据。

每一个这样的子类都有协议代码，用于进一步定义音频设备。子类代码和协议代码在表 17-1 加以了定义。注意协议代码仅仅适用于某些子类，一些协议适用于多个子类。至于 AC3 编码数据在本书的书写时候还没有完全定义完成。

表 17-1　　音频子类和协议

子类代码	子类名称	协议代码	协议名称
01h	8 位脉冲编码调制（PCM）音频数据	01h 02h	单声道 立体声
02h	16 位 PCM 音频数据	01h 02h 03h 04h	单声道 立体声 四声道 双声道立体声
03h	16 位杜比环绕数据	02h	立体声
04h	IEC958 音频编码数据	05h 06h	IEC958 一般用户版 IEC958 一般专业版
05h	MPEG1 音频编码数据	07h 08h	第一层 第二层
06h	AC3 音频编码数据	TBD	TBD

标准音频接口需求

一个音频设备需要的标准端点如下：

- 控制端点 0——用于操纵设置和获得音频功能单元的状态；
- 中断端点——用于获得状态信息；

- 同步端点——一个或多个用于音频数据传输的同步端点。注意一个异步端点可能伴随一个同步端点。

同步端点的数目需求是为每一个音频子类/协议的组合指定的，见表 17-1 的定义。例如，所有的 PCM 音频数据子类/协议的组合需要一个单独的同步端点，当然除了 16 位的 PCM 立体声和立体声的接口之外，后者需要两个同步端点。

同步的类型

音频设备类的说明文档定义了三种方法，同步端点可以用这三种方法保证自己和主机之间的同步：

- 异步——同步端点用异步方式发送或接收音频数据，数据速率和外部时钟或者一个空闲的内部时钟一致。设备传输率不能和 USB 时钟保持一致。（基于 1 毫秒的时间片起始）；
- 同步——设备的同步端点有自己的定时机制，而且需要把它们的音频数据率和 USB 的时间片的 SOF 保持同步。为此定义了两种方法：
  （1）使采样时钟和 1 毫秒的 SOF 同步。
  （2）调整 SOF 直到它和采样时钟同步。
- 适配器——这种设备有一个数据传输率的范围，在给定范围内，它们可以发送和接收音频数据，允许它们和所采用的数据率保持同步。

音频设备的特定描述符

音频设备的说明书定义了一个类特定的描述符。这些特定的类描述符是附加到标准接口上和端点描述符上的。设备是基于标准的端点描述符信息配置的，就是说，总线带宽是基于描述符里指定的带宽需求分配的。一旦装载了音频类设备，它就必须从设备那里获得附加的信息，这样才能完整地理解设备所支持的属性，以及数据同步化的方法。可以参见音频设备说明书获得关于这些类专用描述符的格式和定义的详细信息。

音频类的特定请求

这些音频类的规范说明书定义了类特定的请求，它可以控制不同的音频属性，这些属性可以被划分以下两个部分：

- 音频控制属性——这类请求控制音频功能，例如音量和音质。这些属性是通过音频类说明书定义的音频控制块实现的。音频控制块包含可以由软件通过类特定请求操作的参数；
- 端点属性——这些属性控制音频数据传输的不同的方面，例如采样频率。但是音频类说明书允许其他的厂商自己定义一些属性。有一个“Get/Set System Exclusive”属性的类特定请求可以控制厂商自定的属性。参考音频类规范说明书

可以得到关于音频类特定请求的定义和使用方面的细节。

通信设备类

任何和一个电话线相连的 USB 设备都属于通信设备类的范畴。在本书书写的时候，已经定义了两个子类：

- 电话接口；
- 厂商自定。

但是，大量的协议被定义在电话子类中，见表 17—2。这些协议指出了用于控制通信功能单元的控制协议。

表 17-2　　电话设备使用的电话协议类型和代码

协议代码	描　述　符	相关的参考文档
00h	未定义。	NA
01h	普通的 AT 命令（与贺氏产品兼容）。	V.225ter
02h	可选的 PSTN modem 命令集。	V.25 位
04h	串行 ISDN 终端适配器控制。	V.120
08h	带内 DEC 控制。	V.ib
10h	ISDN TA 控制。	Q.931
20h	保留。	NA
40h	音频类说明书没有定义其他标准 DEC 控制协议。该设备所使用的控制协议被定义在字符串描述符内。	NA
80h	使用设备制造商所有的 DEC 控制协议。该协议在用一个字符串描述符表示。	NA

通信设备接口

USB 通信类设备有很多接口，随着它们的特性的不同也不同。这些可能被使用的端点的定义如下：

- 控制端点 0——用于发送对时间要求不高而且对带宽要求较少的请求。这就保证了控制管道不会因为一些过长的命令序列而变得饱和，这些很长的命令是某些控制协议所必须的；
- 中断端点——用于报告设备产生的事件（例如，开/关异常指令和用户按键）和通信网络事件，例如输入调用通知。也可以用来确定可以使用的接口和相关的数据格式；
- 同步端点——同步端点用于发送和接受数据，并保证数据传输有一个固定的位速率，用于那些要求传输延迟时间很小的实时通信；
- 块端点——当数据是由对时间要求不是很高的字符组组成的时候，就会使用这个

端点，例如数据输入输出一个普通的调制解调器。

通信类特定的描述符

通信设备定义了两种特定的类描述符。这两个描述符包括：

- 类特定配置描述符；
- 类特定的字符串描述符（因为它们基于它的协议代码定义了设备的某些方面，所以也被称为协议描述符）。

关于这些描述符的格式和定义的细节请参考规范说明书。

通信类的特定请求

下面的类特定请求是由通信设备类的规范说明书定义的。他们是：

- 发送封装命令；
- 获取封装响应；
- 报告格式（被封装的协议消息）；
- 可使用接口的通知；
- 选择接口协议命令；
- 获得接口命令。

请参考规范说明书以获得关于这些请求的格式和定义的进一步细节

显示设备类

这个设备类定义了用于控制显示设置的机制，例如亮度，对比度和颜色。从传统上来说，这些控制可以在一个硬件的控制面板上用手动的方式来实现。USB 的显示功能允许这些调整可以在软件控制下实现。

这些显示类和子类在设备的描述符内被加以定义，见表 17-3。

表 17-3　　显示类标准设备描述符的定义

字段编号	字段名	大小（字节）	取值	说明
4	设备类	1	类别	对于显示设备类，类代码=04h
5	设备子类	1	子类	子类代码： 01h=CRT 02h=平面显示器 03h=3D 显示器。

标准显示设备类接口

USB 显示设备只要求缺省的控制端点，用它们把控制信息传输到显示器那里。这个类只定义了一个配置和一个接口。注意，由于除了缺省的端点之外，其他的端点都没有被使用，这些接口描述符没有指出端点。

显示设备的特定描述符

一个 USB 显示设备用三种特定的设备描述符：

- 显示描述符——这个显示描述符定义了哪一个控制是设备所支持的，并且指出了显示特性。这些信息是基于 VESA 扩展显示标识符（EDID）规范说明书的。这个描述符通过 Get Display ID 请求获得；
- 显示状态描述符——这个描述符为大量的显示设置提供了状态信息，还显示所使用的水平和垂直刷新频率，该描述符是通过 Get Display Status 请求获得的；
- 显示控制描述符——这个描述符用于确定可以设置的可能的值。独立的显示控制描述符对每一个控制都是存在的，由控制代码在规范说明书中定义。被引用的控制代码在“Get Max”和“Get Current”请求中被说明，它们返回描述符的内容。

显示设备类特定请求

对显示设备来说，定义了 7 种特定的设备请求：

- Get Display ID；
- Get Max；
- Get Current；
- Set Current；
- Get Display Status；
- Degauss；
- Set Display Power Mode。

这些请求提供了获取关于显示设备的当前设置和设置改变的状态的机制。可以参见显示设备类的规范说明书获得进一步的信息。

海量存储设备类

海量存储设备类和大多数的设备类不同，因为它们在系统启动的时候就会被使用。这就需要系统的 BIOS 必须能够初始化和存取 USB 的存储设备。海量存储的定义必须同

时得到设备驱动程序和系统的 ROM 的支持。USB 的海量存储设备类规范说明书定义了几种类型的海量存储设备。目前已经定义了 5 个子类：

- 一般的海量存储子类——这个子类定义了那些一般以随机形式存取存储介质的设备；
- CD-ROM 子类——CD-ROM 当然是只读的，这些驱动程序可能只包含出了海量存储接口之外的接口，这是为了支持音频和视频方面的应用（和海量存储类无关）；
- 磁带——磁带驱动器是唯一一种以数据流的方式从驱动器那里读写数据的设备。因为磁带驱动器以线形的形式存取数据，所以数据发送的时间就很重要。如果数据不能按时送到，就会发生数据中断，就需要时间重试；
- 固态存储设备——这些设备没有移动的部分需要控制，但是需要特殊的命令执行很费时间的写，因为写伴随着一个擦除操作。

一个设备类的代码和子类代码是在接口描述符内定义的。表 17－4 列出了海量显示设备类的编码，以及在接口描述符内的子类代码字段。

表 17-4　　海量存储类设备编码和子类设备编码

字段编号	字段名	大小（字节）	取　值	说　明
5	接口类	1	类　别	对于海量存储设备类，类代码=01h
6	接口子类	1	子　类	海量存储设备的子类代码，定义如下： 01h=一般海量存储设备 02h=CD-ROM 03h=磁带 04h=固态存储设备 05—FFh 保留

标准海量存储接口

海量存储设备使用的标准接口（除了 CD－ROM）是由 3 个端点组成的：

- 控制端点（缺省端点 0）没有被接口定义；
- 块传输端点支持 IN 事务处理；
- 块传输端点支持 OUT 事务处理；
- 中断传输端点。

控制端点

海量存储设备驱动程序和存储设备使用设备的缺省控制端点把命令发送给海量存储单元。这些命令是用 SCSI 命令的结构发送的。当主机访问指定的端点的时候，设备一般会对命令作出响应。例如，根据所用的命令，当中断端点被存取的时候，设备可能返回状态，或者当主机存取一个块传输端点的时候，把数据传输到介质或把数据从介质传输到主机处。

☑ 块(bulk)传输端点

块传输端点用于传输那些从海量存储设备处读取的数据，或者传输那些要写到介质上去的数据。由于块传输是单向的，所以读和写都要求独立的块端点。

☑ 中断端点

当中断端点被海量存储类设备请求的时候，它就会返回状态信息，确定命令的完整状态。那些命令在前面已经被发送给海量存储设备。

一般海量存储子类

一般的海量存储子类被设计为支持可移动设备的子类，而且被看作是有它们自己的永久锁定的介质。这些子类所支持的设备包括：

- 软驱；
- 磁光设备；
- 高密软驱；
- 硬盘驱动器。

对所有这些设备的存取是很相似的，因为它们以一个面向块/扇区的形式支持随机读取和写操作。这些设备支持 SCSI 命令协议，而且它们通常通过缺省控制端点发送命令。为了向后兼容，某些设备（例如软驱）必须在更老的操作系统下和那些介质协同工作，例如使用柱面、磁头和扇区（CHS）来做定位的 DOS 这些设备必须能够确定和报告设备的几何结构，这就和介质有关了。在这种方式下，CHS 信息可以由 BIOS 或操作系统转换为逻辑块。

CD-ROM 子类

和 CD-ROM 相关的子类需要一个和其他的子类所使用的不同的端点集合，这是因为 CD-ROM 有能力存储 CPU 的数据（在文件里的代码和数据），音频和视频数据、CD-ROM 必须支持海量存储和音频接口，而且可能还要包括一个音频/视频接口。这些附加的端口可能包含同步端点，而且必须能够被其他的类驱动程序访问和存取（例如音频类驱动程序）。这些规范说明书定义了在接口描述符的“接口号”字段必须定义的接口号。因为要求有海量存储接口和音频接口，所以 CD-ROM 将至少定义两个接口，它们的接口号是：

- 00h=海量存储接口；
- 01h=音频接口；
- 02h=音频和视频接口。

除了块“OUT” 端点之外，CD-ROM 海量存储接口使用同样的端点。因为数据只能从介质那里读取，所以不需要块“OUT”端点。CD-ROM 的中断端点仅仅用于报告介质

变化。

CD-ROM 是面向扇区/块的设备，但是不像一般的子类设备，扇区的大小可以很大。CD-ROM 的扇区大小范围是 2000 字节到 3000 字节，但是一般的子类设备为 2048 个字节。

磁带子类

这个设备中的子类是那些有可移动介质的设备，要求流式数据。对流式磁带存取的特点是可能需要很多额外的时间寻找所要求的信息，所以用术语伪随机存取来描述对磁带设备的访问。

USB 的磁带设备支持 QIC－157 标准所支持的高级技术连接包接口(ATAPI)。

固态存储设备子类

固态海量存储设备一般要求一个内存块在数据写之前先擦除掉。固态设备仅仅被指定传输数据，而且不需要任何同步的端点。这些接口使用 SCSI－2 协议是“直接存取存储介质和光存储器设备”定义的协议的一部分。参见关于 SCSI 命令细节的规范说明书。这些命令是固态存储设备所支持的。

类和设备特定 USB 请求

海量存储设备使用的编程接口包括对缺省控制端点的使用，用来给设备发送命令。这些命令是通过 USB 控制传输进行的。这些设备支持 USB 规范说明书所定义的标准请求，同时也支持类特定请求和子类特定请求。

一个类特定请求是为海量存储设备定义的，用于发送子类特定（或设备特定）命令。这些请求被定义为“Accept Device－Specific Command（ADSC）”请求。这些请求被定义为请求数 0。当发送的时候，设备的特定命令在控制传输的数据阶段被发送。命令的格式取决于海量存储设备的子类，如下所述：

- 一般海量存储设备——ANSIX3.131，小型计算机系统接口；
- CD－ROM——SFF8020i，用于 CD－ROM 的 ATA 包接口；
- 磁带——QIC－157，用于磁带的 ATA 包接口；
- 固体状态存储设备——QIC－157，用于磁带和 SCSI 命令集的 ATA 包接口（修改版）。

人机接口设备类

人机接口(HID)类属于人类操纵的设备，用于控制计算机操作的一些方面。在这个类

里的设备可能由下面的类型组成：

- 键盘和指示设备；
- 面板控制；
- 和设备相关的控制，例如电话（拨号装置），VCR 远程控制，以及游戏模拟设备；
- 不可以控制设备操作的设备，但是提供一个和其他的 HID 设备相似的数据格式。

和大容量的存储设备一样，某些 HID 设备可能需要用于启动过程。因而有必要在系统的 BIOS 内支持这些设备，这样通过一个基于固件的 HID 客户应用程序就可以访问根输入设备了。

其他的类定义定义了设备子类和协议的定义，它们特征化了这些设备，而且指出了设备使用的协议，它给软件提供访问这个设备所需要的信息。HID 的规范说明书不使用和其他规范说明书相同的方法。因为这样的方法太严格了，当引入新的设备的时候，就不得不建立起新的子类和协议定义，使它们越来越多。所以 HID 设备使用整个描述符来描述设备的特征和定义协议。

实体（entities）已经规定了一套低级参数，它们可以结合起来描述设备的行为特征。在这种情况下，某个设备可以通过选择描述它的输入输出特性的实体描述它自己。已经定义了几组实体标识一个 HID 设备的不同方面的特征。读者可以参考 HID 类的定义，以获得关于实体描述符定义的细节性内容。

第六部分 主控制器和集线器：实现实例

第十八章 通用主控制器

第十九章 开放主控制器

第二十章 TUSB2040 集线器

通用主控制器

✎ 上一章

前一章介绍了设备类的概念，讨论了它们在 USB 中的作用。介绍了最初的五种类型的设备类。讨论这些设备类是为了给读者一个关于每一个类的感性认识，并让读者知道它们使用的 USB 机制。关于设备类的详细讨论需要对相关字段有详细的了解，例如电话和音频类。

✎ 本 章

本章介绍了 Intel 定义的通用主控制器。详细讨论了传输机制和通用主控制器驱动程序所建立的传输描述符。

✎ 下一章

下一章介绍了开放主控制器的设计，它是由康柏、微软以及美国国家半导体公司联合制定的。

🕮 概述

通用主控制器（UHC）和通用主控制器驱动程序（UHCD）负责安排和执行从 USB 驱动器向前传输的 IRP。UHC 还集成了根集线器的功能，它和 USB 集线器的定义兼容。集成到 UHC 中的根集线器有两个 USB 端口。下面的部分描述了 UHC 通过 USB 用于安排和产生事务处理的机制。

UHC 被集成到 Intel 的 PIIX3 PCI ISA 扩展总线桥接器和以后的芯片中。它是作为

一个 PCI 主控器实现的，可以执行事务处理从内存中获取和更新数据结构，它们都是由 UHCD 建立的。

通用主控制器的事务处理安排

每一个在 1 毫秒的时间片内执行的一系列的事务处理过程可以参见图 18-1。注意首先安排的是周期性的传输（同步和中断），然后才是控制和块传输。周期性的传输可以占用最多达 90%的总线带宽，控制传输则要保证至少 10%的带宽。

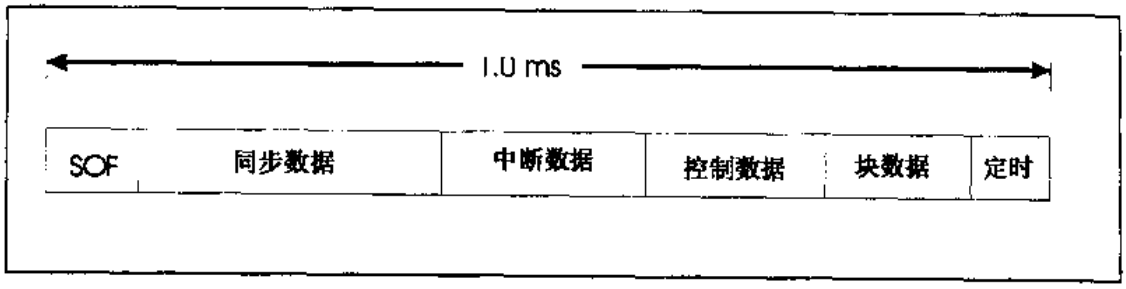


图 18-1　通用主控制器的传输安排

UHCD 通过建立一系列的传输描述符安排事务处理，把它们连接起来就组成了在一个给定的时间片中事务处理的集合。这就被称为时间片列表，并且被定位在系统内存中。

通用主控制器时间片列表访问

图 18-2 演示了 UHC 用于访问内存中的时间片列表的机制。这些包含的组件是：

- 时间片的开始（SOF）计数器——这个计数器每隔 12MHz 的时钟周期就增加 1。当计数器溢出的时候，下一个时间片就开始了。这个时钟也是 USB 位定时器的源，用于根集线器初始化的传输；
- SOF 修改寄存器——这个计数器用于调整位的时间编号，它包含在每一个时间片内。这样就可以改变时间片开始的间隔。修改寄存器支持主客户特征，它允许单个客户驱动程序调整 SOF 定时，以允许 USB 的时间片传输率和它的同步传输率保持一致。缺省值导致产生 1 毫秒时间片；
- 时间片计数器——时间片计数器在每一个时间片开始的时刻加 1，以选择时间片列表中的下一个入口。每个入口包含一个指向第一个传输描述符的指针；
- 帧数目寄存器——UHCD 把固定的开始时间片号编程到寄存器内，在时间片的列表内定义了开始的入口点，这个值一旦被装入，时间片计数器就会和 SOF 计数器一起增加；
- 基于地址寄存器的时间片列表——UHCD 标识出 4KB 时间片列表的基地址。

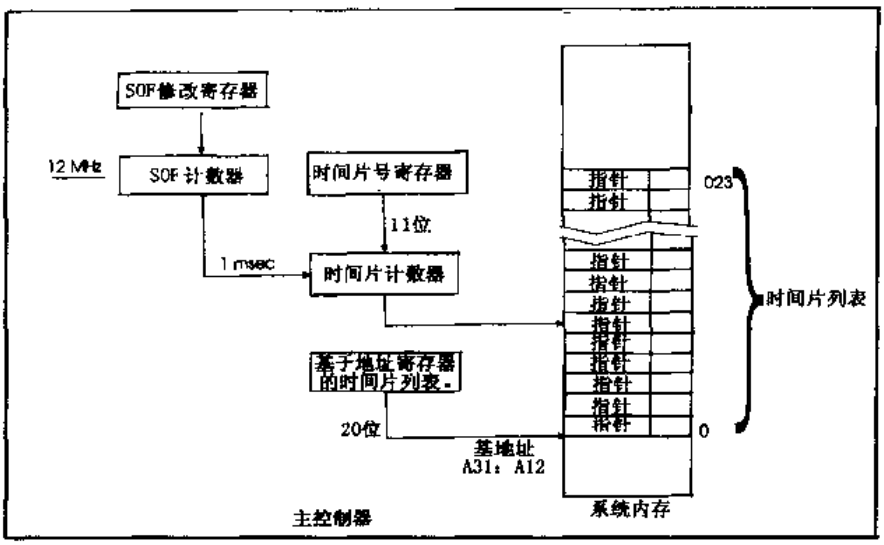


图 18-2　对时间片列表的访问

时间片列表是一个最多 1024 个入口的数组，每个入口对应一个时间片。每个入口包含一个指针，指向数据结构的连接列表，它包含主控制器建立起一个事务处理所需要的信息，该事务处理通过 USB 把数据传递到根集线器那里。USB 读取和解释每一个传输描述符，并且为每一个描述符产生相应的事务处理。

UHC 传输安排机制

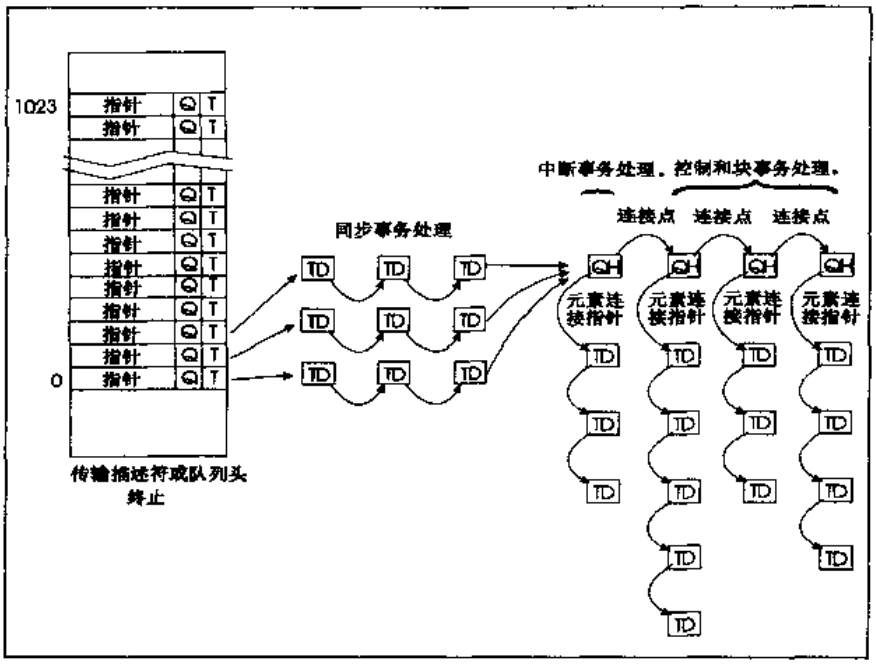


图 18-3　传输机制和执行顺序

图 18-3 就是时间片列表和传输描述符连接的列表，它定义了 UHC 执行的事务处理。这个图表假定很多不同的设备被连接到 USB 上，所有的传输类型都被这些设备使用。传输描述符连接的顺序取决于每一个事务处理在 USB 上发送的顺序。注意时间片列表的入口指向那些为同步传输端点定义的传输描述符。非同步传输可以排成队列，这样就允许在事务处理失败以后重试，但是同步传输不能被重试。

每一个队列的头（QH）和相关的传输描述符（TD）列表是和一个给定的传输类型相关的。第一个队列分配给中断传输，然后是控制传输队列，最后是块传输队列。当所有安排好的传输完成以后，主控制器就可以分配剩下的总线带宽，并执行未完成的控制和块传输。

总线带宽重新分配

总线带宽可以进行重新分配，所以当所有的需要安排的事务处理安排完成以后，UHC 就可以使用剩余的时间片时间执行附加的事务处理，这就要求最后一个 QH 指回控制和块传输队列的开始处。每个 QH 也有一个中止位，当该位被置 1 的时候传输就被中止了，不会发生总线带宽的重新分配。

UHC 跟踪时间片的定时，以监视时间片剩下的部分。一个采样点（称为 PreSOF 点）允许 UHC 确定是否有足够的时间在数据包结束前开始下一个事务处理。在软件控制下，PreSOF 点可以被选择为 36 字节或 64 字节的数据包。如果数据包不能在剩余的时间内完成传输，那么它就不会被传输。

传输描述符

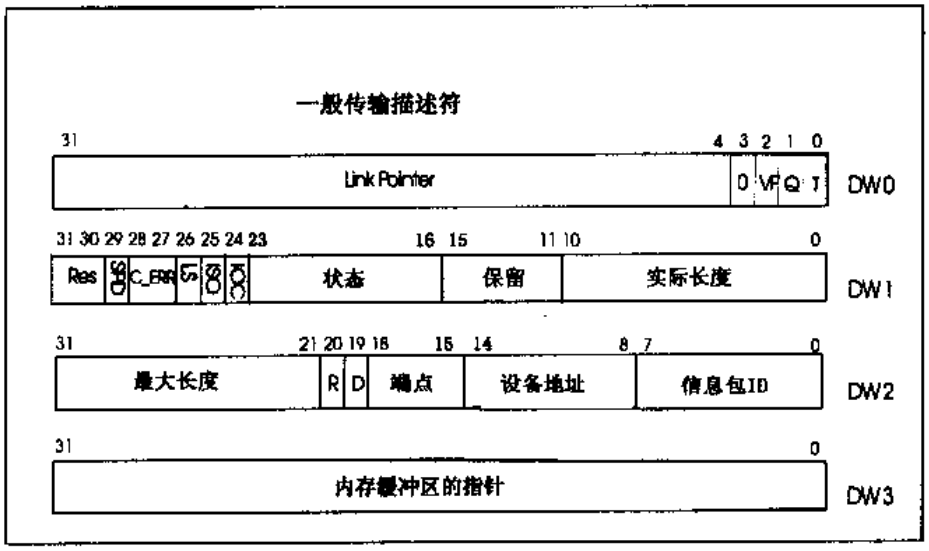


图 18-4　传输描述符的格式

这一部分定义了传输描述符的类型和队列的头标。图 18-4 就是一个传输描述符的内容。一般说来，传输描述符包含 UHC 需要的信息，产生一个事务处理，并且报告状态，包括：

传输类型（同步和其他）

- 令牌包的类型（IN、OUT、SETUP）；
- 传输方向；
- 数据包的大小；
- 数据触发位；
- 内存缓冲区定位；
- 完成状态。

传输描述符由 4 个双字节组成（DW0—DW3）。在传输描述符里的每一个 DW 的定义可以参见表 18-1～18-4。

表 18-1　　字段的定义（DW0）

位	字段名	说　明
0	T	结束——连接点是有效的（0）或无效的（1）。
1	Q	QH（队列头标）或 TD（传输描述符）选择——通知 UHC，它把端点连接到 QH（1）或 TD（0）。
2	VF	纵向优先——指出处理 TD 的方法是深度优先（1）还是广度优先（0）。
31：4	连接点	指向另外一个 TD 或者 QH（地址位 31：4）。

表 18-2　　DW1 的定义

位	字段名	说　明
10：0	实际长度	在传输完成以后 UHC 写的传输的实际长度。
23：16	状　态	UHC 发送的状态信息，指出完成的状态，包括： 活动——由软件设置，激活由 UHC 执行的事务处理。当再次在调度表中遇到这个事务处理的时候，UHC 就会清除它，并指出该描述符不可执行。 停止——由 UHC 设置，表示遇到了一个端点停止。 数据缓冲区错误——由 UHC 设置，表示遇到了一个数据缓冲区上溢或者下溢。 串扰检测——当执行 TD 的时候，UHC 检测到一个串扰状态。因为这表示一个严重的错误，所以 UHC 也会设置一个停止位。 接收到 NAK——当目标设备返回 NAK 的时候，UHC 就会设置该位。 CRC/超时错误——如果一个总线超时或者 CRC 错误被检测到，那么 UHC 就会设置该位。
24	IOC	因为完成产生一个中断——在 TD 完成的那个时间片的结束的时候 UHC 应该产生一个中断。
25	ISO	同步选择——该 TD 是同步的（1）。
26	LS	低速设备——必须使用前导包。
28：27	C_ERR	出错计数器——用两个位的字段表示在执行该描述符的时候发生的错误的数目。当检测到一个错误的时候，UHC 就会减少计数器的值。（串扰和停止不会引起计数器值的减少）
29	SPD	短数据包的检测——该位激活短数据包的检测，它来自访问目标设备的排好队的事务处理（IN 事务处理中的）。

表 18-3　　DW2 的定义

位	字段名	描　述　符
7：0	包 ID	指出使用哪一种类型的令牌包：IN、OUT 或者 SETUP
14：8	设备地址	由这个 TD 存取的设备地址
18：15	端　点	被这个 TD 存取的设备地址
19	数据触发	指出该 UHC 应该发送或者等待哪一个数据包。（对同步传输而言总是 0）
31：21	最大长度	指出本次传输所允许的最大字节数。最大值是 1280（4FFh），它是能保证在一个时间片内发送出去的最大数据包大小（不包括 PID 或 CRC）。

表 18-4　　DW3 的定义

位	字段名	描　述　符
31：0	内存缓冲区的指针	指向事务处理中所使用的那个内存缓冲区的开始。这个缓冲区必须至少和 DW2 中的最大长度字段中的值一样长。

队列头

队列头标识出了一个已经被排序的传输描述符的连接列表。队列头包括一个指针，它指向第一个要被执行的 TD（称为 QH 元素连接指针）而且指向下一个 QH（称为连接指针）。QH 也有一个中止位，允许软件结束时间片事务处理，而不需要进行总线带宽的重新分配。图 18-5 就是 QH 的格式，至于每个字段的描述符的具体内容可以参见表 18-5 和 18-6。

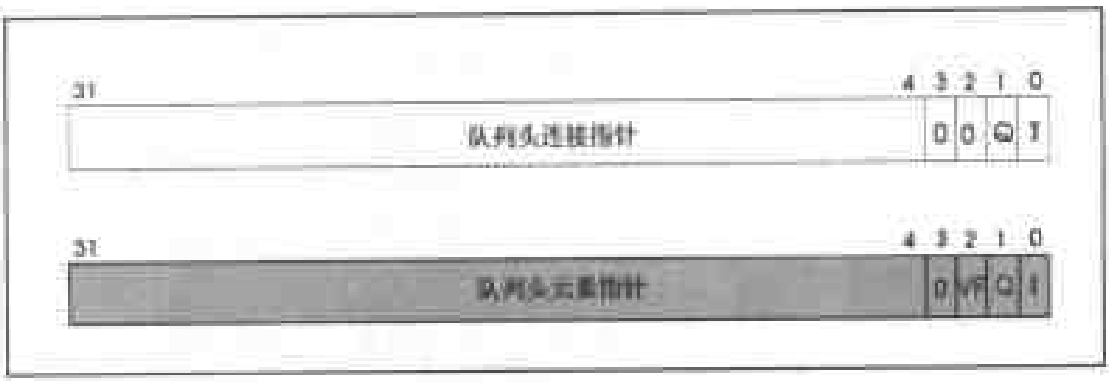


图 18-5　队列头链表和元素链表指针

表 18-5　　队列头标连接指针的定义

位	名 称	说　明
0	T	结束——1＝最后一个 QH（指针无效，在执行完队列里所有的 TD 以后，不再继续进行处理）。0＝指针有效（处理下一个描述符）
1	Q	QH/TD 选择——1＝QH，0＝TD。定义了连接指针引用的下一个描述符的准确位置，所以可以正确地读取描述符。
3：2	保留	保留——必须置为 0。
31：4	QHLP	队列头标连接指针——指向在该列表中下一个要处理的描述符的地址。

表 18－6　　队列头标元素连接指针的定义

位	名 称	说　　明
0	T	结束——1＝结束（没有有效的队列入口）。0＝指针有效（处理下一个 TD）。
1	Q	QH/TD 选择——1＝QH，0＝TD。定义了连接指针引用的下一个描述符的准确位置，所以可以正确地读取描述符。
2	VF	UHC 忽略该位。
3	保留	保留——必须置为 0。
31：4	QHLP	队列头标元素连接指针——指出在该列表中下一个要处理的 TD 或者 QH 的地址。

🕮 UHC 控制寄存器

UHC 把它的寄存器映射到 PCI 的 I/O 地址空间，这些寄存器由 UHCD 存取，用于控制 UHC 操作的不同的方面，参见表 18-7。注意基地址是由 PCI 配置软件在 PCI 的枚举过程中确定的。

表 18-7　　UHCI I/O 寄存器

I/O 地址	寄存器的访问方式	寄存器说明
基地址 +00-01h	读/写	USB 命令寄存器——写这个寄存器会引起指定的控制动作。位字段定义如下： 最大信息包：选择 32 或者 64 字节的数据包用于总线带宽的重新分配。 配置标志：表示配置没有完成。而且对硬件没有造成影响。 软件调试：用于使能或禁止调试功能，和运行/停止字段相关。 强迫全局恢复：引起主机控制器广播恢复信号。 进入全局挂起状态：引起所有下游的 USB 事务处理停止，导致全局挂起。 全局复位：迫使 UHC 在 USB 上发送 10ms 的复位信号。 UHC 复位：引起内部定时器、计数器、状态机等回到它们的缺省状态。 运行/停止状态：允许主控制器进行单步的 USB 事务处理。用于和软件调试位进行与运算。
基地址 +02-03h	读/写控制	USB 状态寄存器——该寄存器提供了不同的状态，这些状态位如下定义： HC 停止：表示主控制器已经停止执行。它由运行/停止位引发，也可以由内部错误产生。 UHC 处理错误：表示当处理一个 TD 的时候发生了一个十分严重的错误。UHC 自动设置停止位，以阻止 TD 的进一步执行。 PCI 总线错误：表示在 PCI 事务处理的过程中发生了一个严重的错误，引起执行中止。 恢复检测：在挂起期间，当 UHC 从 USB 上检测到一个远程唤醒信号的时候，它就会设置该位。 USB 出错中断：表示 USB 事务处理产生了一个错误的状态。 USB 中断：当 TD 完成以后，而且 TD 的 IOC 位被设置以后，该位就会被设置。

续 表

I/O 地址	寄存器的访问方式	寄存器说明
基地址+04-05h	读/写	USB 中断使能——使能或禁止以下的中断源： 短数据包中断使能； 完成时中断使能； 恢复使能； 超时/CRC 使能。
基地址+06-07h	读/写	时间片编号——包含当前的时间片号和时间片列表的索引值。
基地址+08-0Bh	读/写	时间片列表的基地址——包含在内存中的时间片列表的基地址。可编程的范围仅仅限制在 4KB 内。
基地址+0Ch	读/写	起始时间片修正——包含一个加到 SOF 计数器的起始值上的修正值，用于调整一个时间片内的位计时的编号。缺省值是 64，它被加到 SOF 计数器的缺省值 11936 上，所以就产生 12000 或者说 1ms 的 SOF 间隔。
基地址+10-11h	读/写控制	端口 1 状态/控制——这个寄存器控制根集线器的端口 1，而且反映了端口状态的变化。它的位字段的定义如下： 挂起：表示端口当前是否处于挂起状态。 端口复位：表示当前端口是否已经复位了。 低速设备连接：表示连接设备的速度。 恢复检测：表示已经给 USB 设备发出了一个远程唤醒信号。 线状态：这两个位反映了 D+和 D-线上的逻辑电平的状态，它可以用于调试。 端口使能/禁止变化：表示端口已经因为设备断开连接或者在端口上检测到串扰和 LOA 而被禁止。 端口使能/禁止：表示端口当前是使能的还是禁止的。 连接状态变化：表示设备和端口连接或者断开连接。 当前连接状态：表示当前是否有设备连接到端口上。
基地址+12-13h	读/写控制	端口 2 状态/控制——和端口 1 的定义一样。

开放主控制器

✎ 上一章

上一章介绍了 Intel 定义的通用主控制器。对传输机制进行了讨论，而且对通用主控制器的驱动程序进行了详细的说明。

✎ 本　章

本章讨论了开放主控制器的设计，它是由康柏、微软和国家半导体公司联合制定的。

✎ 下一章

下一章给出了一个基于德州仪器 TUSB2040 集线器的 USB 集线器实例。

🕮 概述

开放主控制器（OHC）和开放主控制器驱动程序（OHCD）负责安排和执行 IRP 从 USB 驱动程序向前传递。OHC 还集成了根集线器的功能，它和 USB 集线器的定义兼容。集成到 OHC 内的集线器有两个 USB 端口。以下的部分描述了 OHC 和 OHCD 通过 USB 安排和产生事务处理的机制。

🕮 开放主控制器传输安排

图 19-1 画出了开放主控制器执行的传输顺序。注意在时间片的开始可以有一个保留部分，（它是用于非周期性的传输的），使控制传输保证有 10%的带宽，而且确保某

些非周期性的传输（控制和块传输）在每一个时间片内都能够被执行。接下来，周期性传输（中断传输和同步传输）可以用去最多 90%的带宽，如果时间有剩余，那么就可以执行剩下的非周期性的传输。

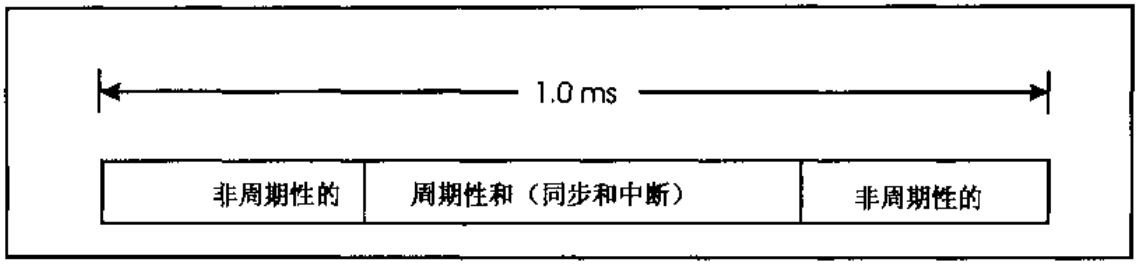


图 19-1　USB 传输时间安排

OHCD 通过建立一系列的传输描述符进行事务处理的安排，它们连接起来组成事务处理的集合，在给定的时间片内执行。这被称为时间片列表，定位在系统的内存中。

开放主控制器的传输机制

图 19-2 演示了一种机制，用于在每一个连续的时间片内产生事务处理。OHCD 建立了一个描述符，并且把它们放进一个内存区域，称为主控制器的通信区域（HCCA）。这些描述符包括端点描述符，（EDs）和传输描述符（TDs）。OHCD 给每一个端点在系统内安排了一个 ED。ED 包含地址号和端点号，这就为控制器提供了必要的信息用于和端点进行通信。每个 ED 在图 19-2 中都代表一个环。一个 TD 序列和每一个 ED 相连，那些 ED 代表那个端点有待完成的事务处理。

一个给定的传输类型的 ED 都连接在一起，而且控制器内的寄存器指向它们。注意有一个寄存器指向每一个非周期性的传输类型（控制 ED 和块 ED），还有一个独立的寄存器（HCCA 寄存器）指向同步传输（中断和同步）的连接列表，该列表指向处理中断和同步传输的 32 个入口之一。头指针所指向的寄存器指向 ED 列表。

OHC 转换 ED 列表的顺序见图 19-1。控制器通过存取控制和块描述符开始转换，在前一次传输的结束处停止。在一个预定的间隔以后，它就被编程进这个控制器，它不再进行非周期性的传输，而是通过 HCCA 寄存器进行周期性的传输。注意中断的安排和其他的传输类型不一样。HCCA 寄存器在每一个时间片的结束增加，这样它就指向下一个中断 TD 的列表，它们被安排在当前时间片内完成。中断列表的结束总是和同步 ED 的开始相一起。一旦同步传输完成以后，如果在当前时间片有足够的时间，那么控制器就可以继续处理控制和块传输。

在 OHC 执行完成之后，或者在执行 TD 的时候发生了一个错误时，OHC 就把传输描述符和“Done”序列连接起来。然后 OHCD 就可以检查已经完成的序列，用以完成状态信息。

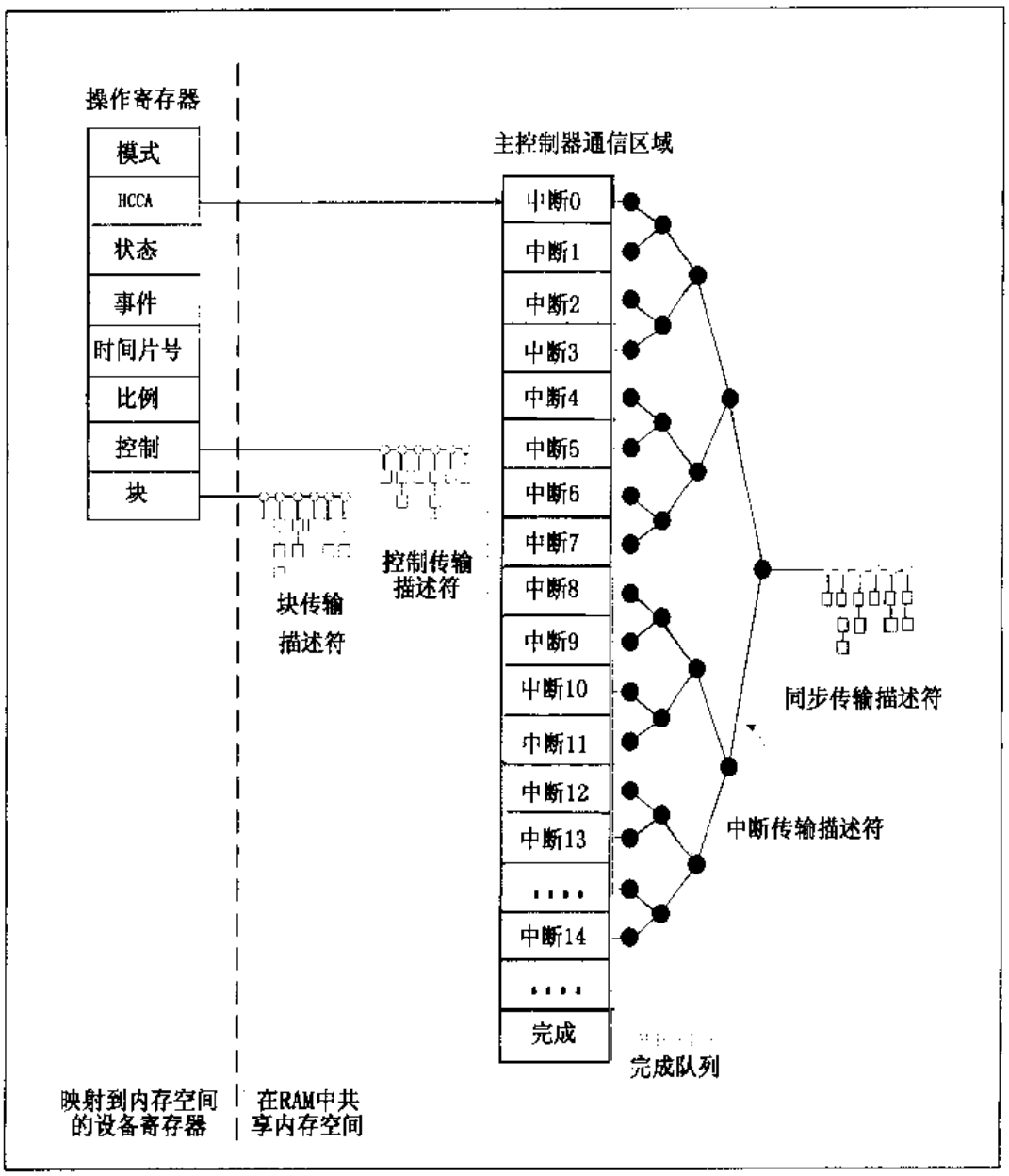


图 19-2　传输安排机制

ED 和 TD 的列表结构

所有的传输都是用标准 TD 列表结构进行安排的，如图 19-3 所示。头指针是 OHC 寄存器，用于控制和块传输。每一个时间片选择不同的中断头指针，当 HCCA 寄存器的值增加的时候，中断头指针定义在 HCCA 的内存区域。同步 ED 只是和最后一次当前中断列表中的 ED 连接。当 IRP 被 USB 设备驱动程序请求的时候，TD 被排队到每一个 ED 中。一个当前处于空闲状态的端点不会对任何 TD 进行排队。

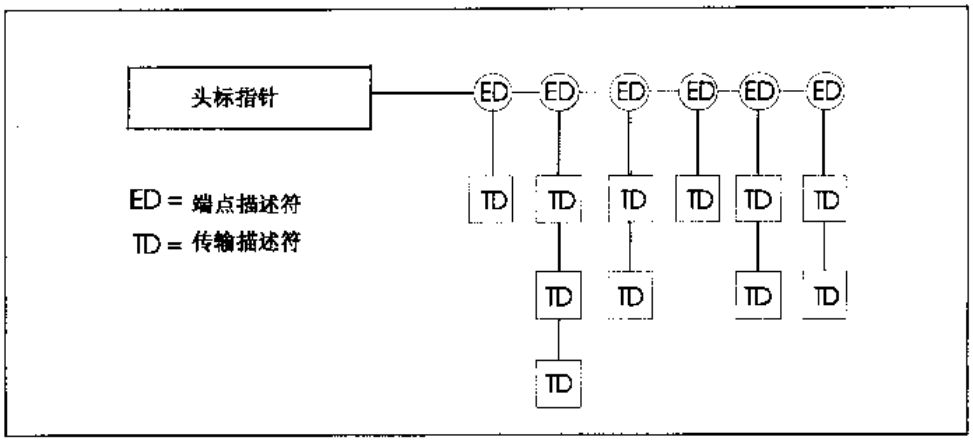


图 19-3　传输队列

☑ 中断和同步传输的处理

在时间片开始的时候，控制和块传输就被排队，这个过程一直到 HcFmRemaining 寄存器的剩余字段小于或等于 HcPeriodicStart 寄存器的剩余字段。比块传输执行更多的是控制传输，这个比例是由 HcControl 寄存器的“控制块服务比例”字段指定的。控制器执行这些传输，基于一个和比例相关的 robin 序列。（例如，三个控制传输伴随着一个块传输）。如果在完成周期性的传输以后，时间片还剩下足够的时间，那么控制器就返回控制和块（bulk）的列表，然后继续处理剩下的事务。

☑ 完成序列

当 OHC 完成了一次传输以后，TD 就被连接到了完成序列，并被写回 HCCA。在这种方式下，OHCD 可以搜索 Done 队列，以确定哪一个事务处理已经出队了，以及它的完成状态是什么。

中断传输安排

图 19-4 从概念上演示了中断 ED 是如何被连接的，从而确保了中断传输在每一个确定的查询间隔内发生。HCCA 的指针寄存器指出了 HCCA 在内存中的基地址，那里是一个有 32 个中断头指针的列表。时间片号的低 5 位被当作一个进入中断头指针列表的索引。在每一个时间片内，执行一个不同的中断事务处理队列，它的执行基于和选择的中断传输头指针连接的 ED。

中断 ED 被连接，所以它们每隔 n 个时间片就会出现在列表中，它代表了查询间隔。例如，32 毫秒的查询间隔将仅仅连接一个中断头指针。因为每隔 32 个时间片一个给定的头指针被存取一次，就是每 32 毫秒发生一次。在另一种情况下，一个查询间隔为 1 毫秒的中断端点将会连接到每一个中断头指针的列表中。在这种方式下，所安排的中断事务处理的时间间隔可以是 1 毫秒、2 毫秒、4 毫秒、8 毫秒、16 毫秒或 32 毫秒。

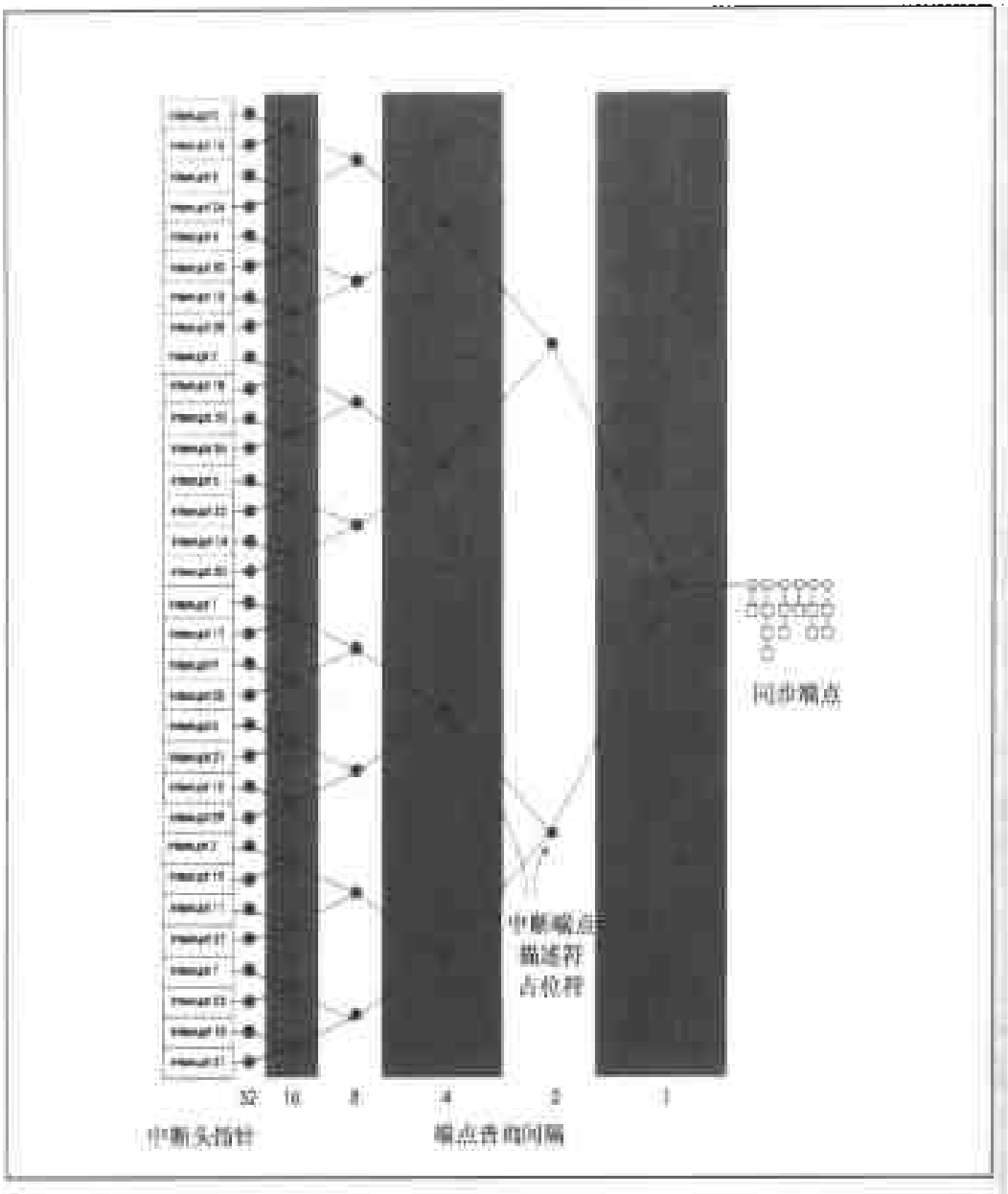


图 19-4　中断安排

端点描述符

每一个 USB 上的端点都有端点描述，它们提供了和位置相关的信息（设备地址和端点号）以及端点的特性。此外，当 OHCD 已经对传输排队以后，ED 就包含指向 TD 列表的指针。图 19-5 显示了 ED 的格式，表 19-1 到表 19-4 定义了在这个描述符里的每一个字段。

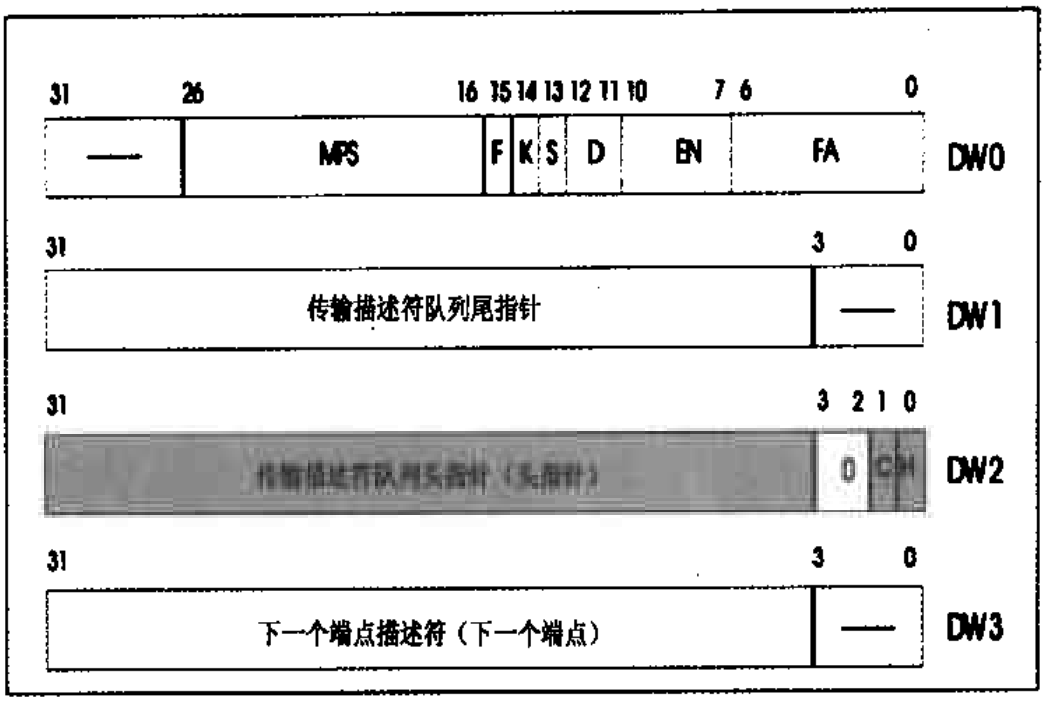


图 19-5　端点描述符的格式

表 19-1　　**端点描述符字段的定义（DW0）**

位	字段名	说　明
6：0	FA	设备地址——USB 设备包含目标设备端点。
10：7	EN	端点号——在设备中的目标端点号。
12：11	D	方向——传输方向，如下编码： 00b＝从 TD 处获得方向； 01b＝OUT； 10b＝IN； 11b＝从 TC 处获得方向。
13	S	Speed——指出端点的数据线传输速度。 0＝全速（12Mb/s）； 1＝低速(1.5Mb/s)。
14	K	空指令——当该位被设置的时候，主控制器继续下一个 ED，而不出力任何和 ED 有关的 TD。
15	F	格式——定义了和 ED 有关的 TD 的格式。 0＝中断、块、或控制 TD 格式。 1＝同步 TD 格式。
26：16	MPS	最大数据包的大小——指出端点所支持的最大的数据包的大小。

表 19-2　　端点描述符字段的定义（DW1）

位	字段名	说　　明
3：0	"——"	未定义——主控制器驱动程序可以利用这个字段进行任何它希望的处理，但是主控制器将忽略这个字段。
31：4	TailP	传输描述符队列尾指针——指向和 ED 有关的队列中的最后一个 TD。如果 TailP 和 HeadP 的值相等，那么 ED 就不包含主控制器可以处理的 TD。如果它们的值相等，那么 TD 就指出一个有效的内存缓冲区，用于双向信息的传输。

表 19-3　　端点描述符字段的定义（DW2）

位	字段名	说　　明
0	H	停止——当该位被设置的时候，该位对 TD 队列的处理就已经停止了。这通常是因为处理 TD 队列的过程中发生了错误。
1	C	触发进位——当必须从某个 TD 传输到下一个 TD 的时候，该位就包含触发位的值。
2	0	该字段必须由主控制器驱动程序写为 0。
31：4	HeadP	传输描述符队列头指针——指向下一个 TD。它是端点将要处理的列表。

表 19-4　　端点描述符字段的定义（DW3）

位	字段名	说　　明
3：0	"——"	未定义——这些字段可以由主控制器驱动程序用于任何目的，并且被主控制器忽略。
31：4	NextED	下一个端点描述符的指针——指向在连接列表中的下一个 ED 描述符，如果是 0 就表示连接列表的结束。

传输描述符

当一个 IRP 被传输到 OHCD 后，传输描述符就会被排序。TD 包含了指定的信息，用于执行传输，例如内存缓冲区位置、OUT 数据或者 IN 数据就被放在缓冲区内。一个给定的传输很可能花大量的时间片才能完成。完成状态将在 TD 中报告。定义了两种形式的传输描述符：

- 一般 TD；
- 同步 TD。

一般 TD 用于所有的传输，除了同步传输之外。在第一次传输完成之后，支持同步传输的设备驱动程序一般要求附加的传输，而且提供缓冲区，使数据率可以匹配。同步 TD 支持以下的机制。

一般传输描述符

一般的 TD 格式在图 19-6 中列出。每一个字段的内容定义见表 19-5～19-8。

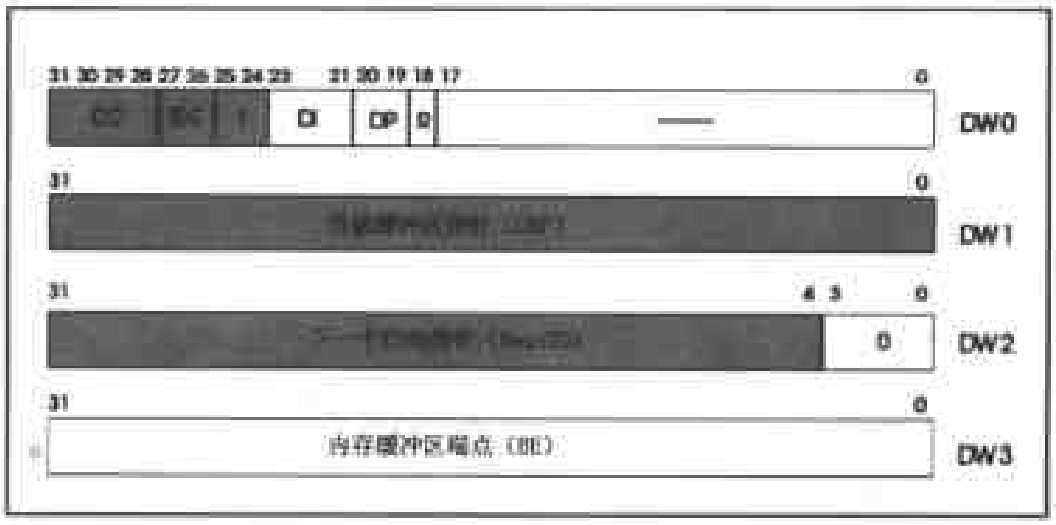


图 19-6　传输描述符的格式

表 19-5　　传描述符字段的定义（DW0）

位	字段名	说　　明
17：0	"------"	未定义——这些字段可以由主控制器驱动程序用于任何目的，并且被主控制器忽略。
18	R	缓冲区舍入——如果是 0 的话，该位就表示来自端点的随后一个数据包必须恰好填满定义的数据缓冲区。如果是 1 的话，那么就表示最后一个数据包可以比定义的缓冲区小，但是不会发生错误。
20：19	DP	方向/PID——这个包 ID 用于字段中指定的包，它还定义了传输的方向。字段的编码如下： 00b＝建立（到端点）； 01b＝OUT（到端点）； 10b＝IN（来自断电）； 11b＝保留。
23：21	DI	延迟中断——包含一个延迟数，它表示在产生一个中断之前主控制器必须等待的时间片的数目，表示 TD 已经成功完成了。如果该字段包含"11b"，那么就没有中断和完成 TD 相关。
25：24	T	数据触发——该字段的最低位表示数据触发状态。该值只有当本字段的最高位是"1"的时候才有效。如果本字段的最高有效位是"0"，那么触发位必须从 ED 中的触发进位字段那里获得。
27：26	EC	错误数——这个字段包含和事务处理相关的错误，它们发生在传输描述符的处理过程中。每发生一次传输错误计数器就加 1，当计数器的值增加到 3 以后，错误的类型就会记录在错误状态码字段（位 31：28），而且端点已经停止了。如果 TD 在发生 3 个错误之前完成，那么这个字段就被置为 0。
31：28	CC	错误状态代码——指出错误的类型，错误在执行 TD 的时候发生。

表 19-6　　传输描述符字段的定义（DW1）

位	字段名	说　　明
31：0	CBP	当前缓冲区的指针——该字段指向下一个内存位置的物理地址。该地址在从端点发送或接收数据的时候布用。0 表示这个 TD 的所有字节已经传输完成了。

表 19-7　　传输描述符字段的定义（DW2）

位	字段名	说　　明
3：0	0	主控制器必须把这个字段置为 0。
31：4	NextTD	下一个传输描述符——指向本端点列表的下一个传输描述符。

表 19-8　　传输描述符字段的定义（DW3）

位	字段名	说　　明
31：0	BE	内存缓冲区的结束地址——这个字段包含本传输描述符的内存缓冲区中最后一个字节的地址。

☑ 同步传输描述符

图 19-7 所示的是 32 字节的同步传输描述符的格式和内容。因为同步端点有定时元素保证数据率的匹配，同步传输描述符可以执行、跟踪和缓冲最多 8 个连续的数据帧。

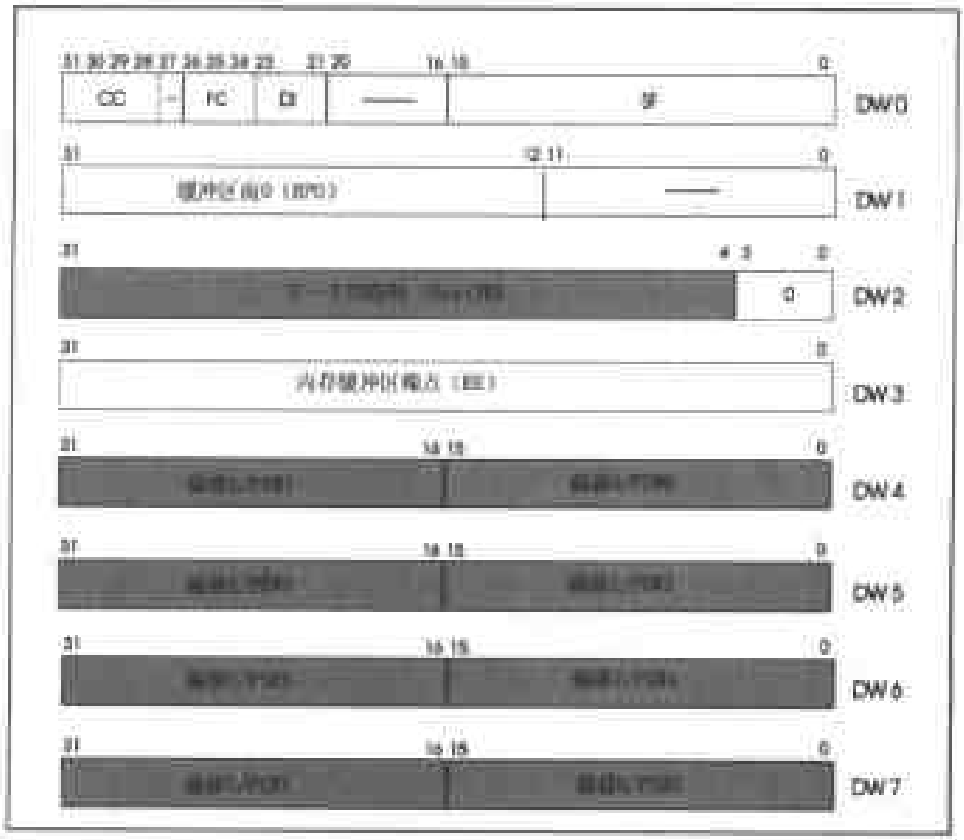


图 19-7　同步传输描述符

描述符包含一个开始的时间片号（在开始时间片字段），它由 16 位组成，它确定应该开始哪一个同步传输。OHC 从实际时间片计数器的低 16 位减去 16 位的开始时间片值，确定该从哪一个时间片开始。如果结果是负的，控制器就会认为还没有到达开始点，并且前进到下一个 ED。但是当开始时间片和当前时间片号匹配的时候，OHC 就用一个时间片参量 0 开始传输。数据被放在一个缓冲区内，它是由 BufferPage0 和 BufferEnd 字段指定的。数据放在缓冲区内的地址是由 Offset0 字段定义的。从下一个时间片开始的数据被放在 Offset1 指定的缓冲区内，依此类推。

接着下一个数据包的传输，在 TD 内的偏移量被完成状态信息替换，称为包状态字（PSW）。前四位定义了状态码，剩下的位指出传输数据量的大小。

同步 TD 中的每一个字段在表 19-9～19-13 中加以了描述。

表 19-9　　同步传输描述符字段的定义（DW0）

位	字段名	说　　明
15：0	SF	起始时间片——包含时间片号的低 16 位，同步传输按照它被安排执行。
20：16	"——"	未定义——该字段由主控制器的驱动程序用于任何目的，主控制器忽略该字段。
23：21	DI	延迟中断——包含一个延迟数，它表示在产生一个中断之前主控制器必须等待的时间片的数目，表示 TD 已经成功完成了。如果该字段包含"111b"，那么就没有中断和完成 TD 相关。
25：24	FC	时间片计数器——由 TD 所描述数据的数据包（时间片）的数目。0=1 个时间片，7=8 个时间片。
27	"——"	未定义——这个字段可以由主控制器驱动程序用于任何目的，主控制器忽略该字段。
31：28	CC	错误状态码——指出当执行 TD 的时候所发生错误的类型。

表 19-10　　同步传输描述符字段的定义（DW1）

位	字段名	说　　明
11：0	"——"	未定义——这个字段可以由主控制器驱动程序用于任何目的，主控制器忽略该字段。
31：12	BP0	缓冲区页 0——指向物理页的编号，因为这个缓冲描述符开始的数据从这里开始。

表 19-11　　同步传输描述符字段的定义（DW2）

位	字段名	说　　明
4：0	0	主控制器必须把这个字段置为 0。
31：5	NextTD	下一个传输描述符——指向连接列表中的下一个同步 TD。

表 19-12　　同步传输描述符字段的定义（DW3）

位	字段名	说　　明
31：0	BE	内存缓冲区的结束地址——这个字段包含本传输描述符的内存缓冲区中最后一个字节的地址。

表 19-13　　同步传输描述符字段的定义（DW4-7）

位	字段名	说　　明
15：0 31：16	OffsetN	偏移量 0～7——指出数据缓冲区内的偏移量，那个时间片的数据就被存储在那里。
15：0 31：16	PSWN	包状态字 0～7——用于包放到内存缓冲区以后报告完成状态，同时指出传输数据包的实际大小。

开放主控制器寄存器

控制寄存器映射到 PCI 的内存地址空间，通过 PCI 配置基地址寄存器。这些寄存器可以按照功能分组。如下所述，还可以参见图 19-8。

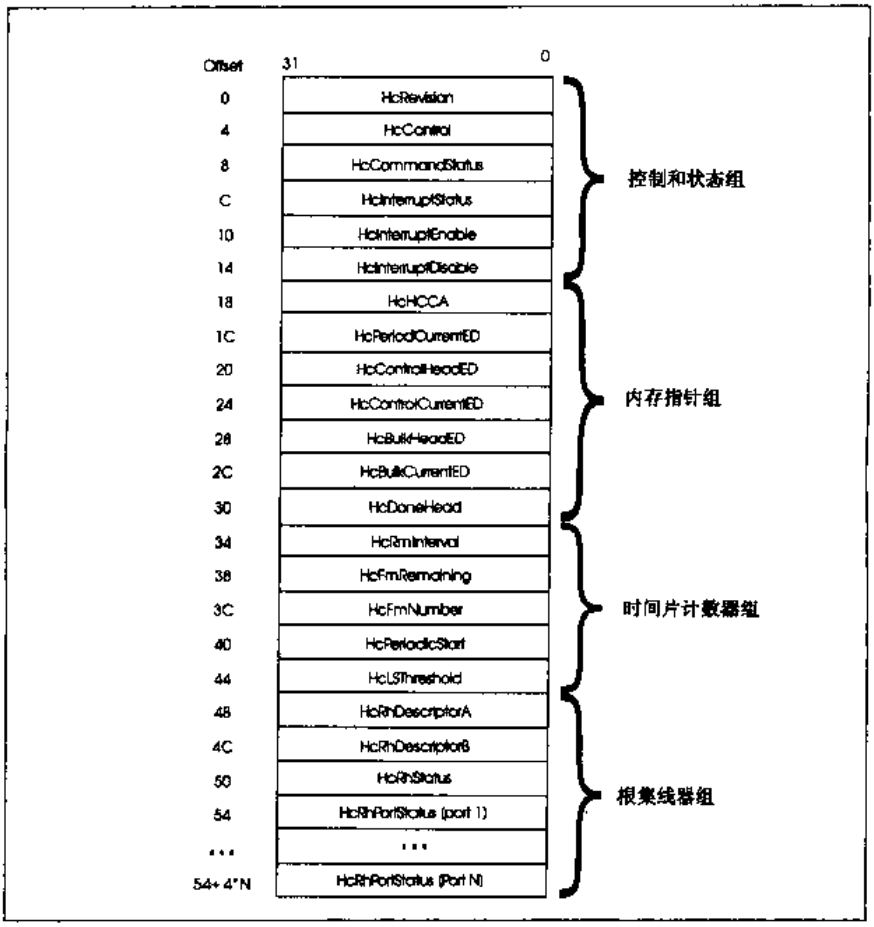


图 19-8　开放主机控制寄存器

- 主控制器控制和状态——这些寄存器定义了主控制器的操作模式，反映了主控制器的当前状态，提供了中断控制和状态，并且反映了错误的状态；
- 内存指针——这个寄存器提供了指向数据结构的指针，和主控制器驱动程序进行通信，并执行内存中的基于传输描述符的事务处理；
- 时间片计数器和控制——时间片的定时状态和控制是由一套寄存器提供的，并且管理 SOF 定时和控制事件，它们和时间片的定时间隔有关；
- 根集线器的状态和控制——这些寄存器用于控制根集线器。有两套寄存器控制这两个端口。关于这些寄存器的详细信息请参考开放主控制器规范说明书。

TUSB2040 集线器

✎ 上一章

上一章讨论了开放主控制器的设计，它是由康柏、微软和国家半导体公司联合设计的。

✎ 本　章

本章对德州仪器制造的 TUSB2040 USB 集线器作了一个介绍。

🕮 概述

TUSB2040 是一个有四个端口的 CMOS 设备，它是基于 USB1.0 规范说明书的。还有一种 7 个端口的集线器（TUSB2070）。集线器的关键特征包括：

- 使用 Intel 串行接口引擎（SIE）；
- 支持总线供电模式和自供电模式；
- 包括集成的 USB 收发器；
- 在所有的端口上支持全速和低速操作；
- 支持挂起和恢复操作；
- 3.3V 直流选项；
- 48MHz 石英晶振荡器；
- 在 28 针 DIP 包上可用。

图 20-1 所示的就是组成集线器的功能块。某些集线器的设计是基于微控制器的，但是 TUSB2040 采用了一种状态机的设计办法。

芯片的时钟是通过 48MHz 的石英晶振荡器获得的。下面的部分讨论了组成 TUSB2070 的主要元素。

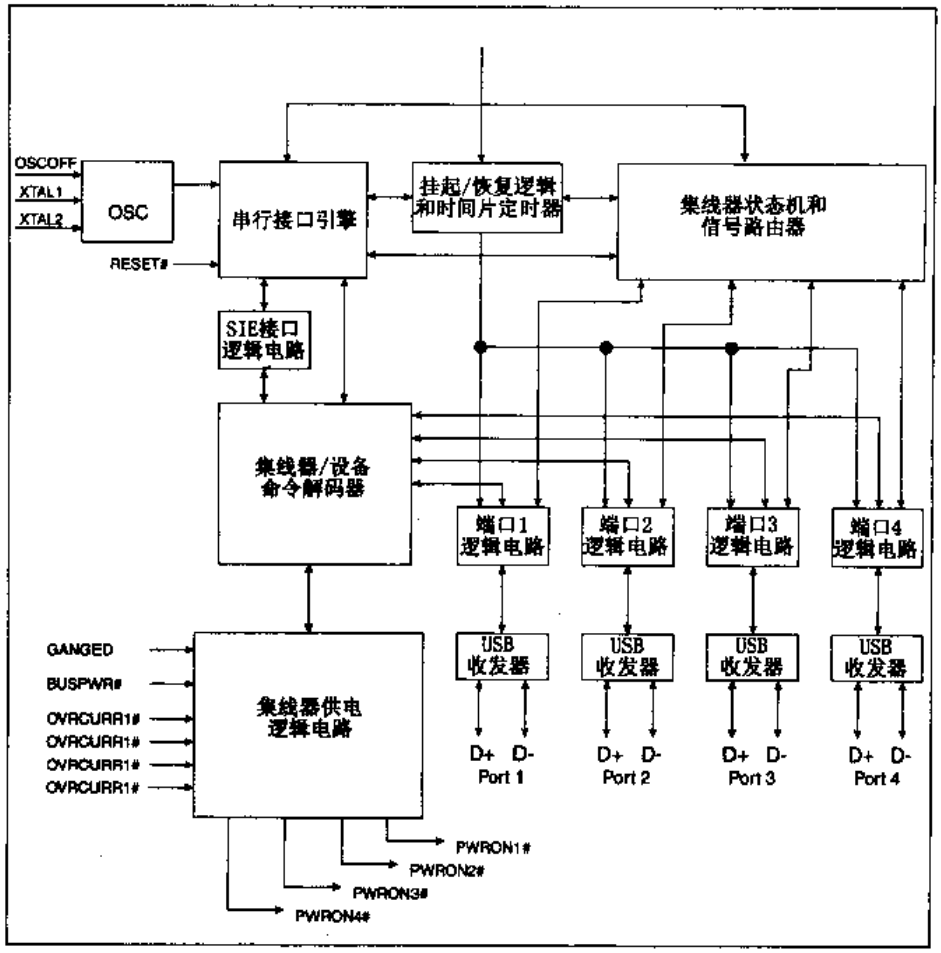


图 20-1　TUSB2040 集线器的分块图

供电控制

USB2040 可以采用总线供电的方式，也可以采用自供电的方式。如果采用总线供电的方式，就要把 BUSPOWER＃管脚接地，如果采用自供电的方式，就要设法把该管脚定为 3.3V 直流电压。电源开关和过流保护可以用专用端口保护或组保护模式。可以把 GANGED 管脚定为地（专用电源开关和过流保护），也可以把该管脚接上 3.3V 直流电（组开关和电流保护）。这些内容在下面的部分加以讨论。

单独端口控制的自供电

图 20-2 所示的是一个自供电的方式，它有一个单独电源端口的开关，而且有过流

保护。专门的端口电源开关和过流保护是是由 4 个 TPS2014 提供的。当 PWRON＃管脚加上规定电压的时候，端口就被上电，从而引起相应的 TPS2041 把它的输出管脚电压切换到 5V 直流电。如果一个 TPS2041 检测到一个过流状态，它就给它的 OC＃（过流）管脚加上电压，这就通知 YUSB2040 集线器发生了过流状态。

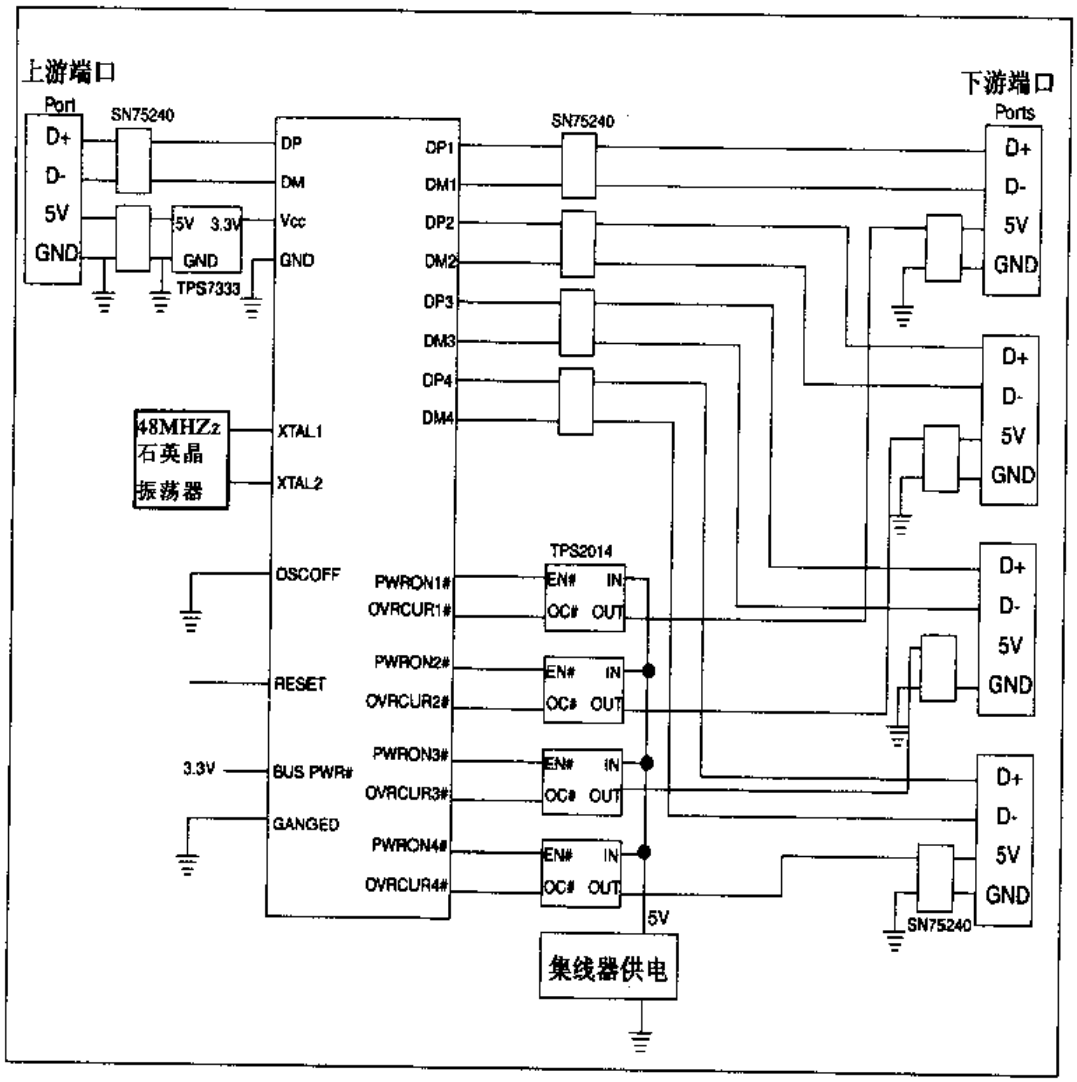


图 20-2　TUSB 2040 作为自供电集线器的实现，它有单独的端口供电开关和过流保护

注意在上面这个例子中，尽管 TUSB2040 采用的是自供电方式，但是它实际上是由 USB 供电的。这就允许了 USB 软件在本地供电被关掉的情况下存取集线器，也就给了软件一个检测非本地供电状态的机会，并把检测结果报告用户。SN75240 的瞬间抑制器可以减弱涌入电流和电压尖峰。

组端口控制的自供电集线器

图 20-3 所示的是 TUSB2040 作为一个自供电的集线器的例子，但是用一个 TPS2015

执行群电压开关和过流保护会降低成本。当 GANGED 管脚为 3.3V 的时候，TUSB2040 是为组选项配置的。

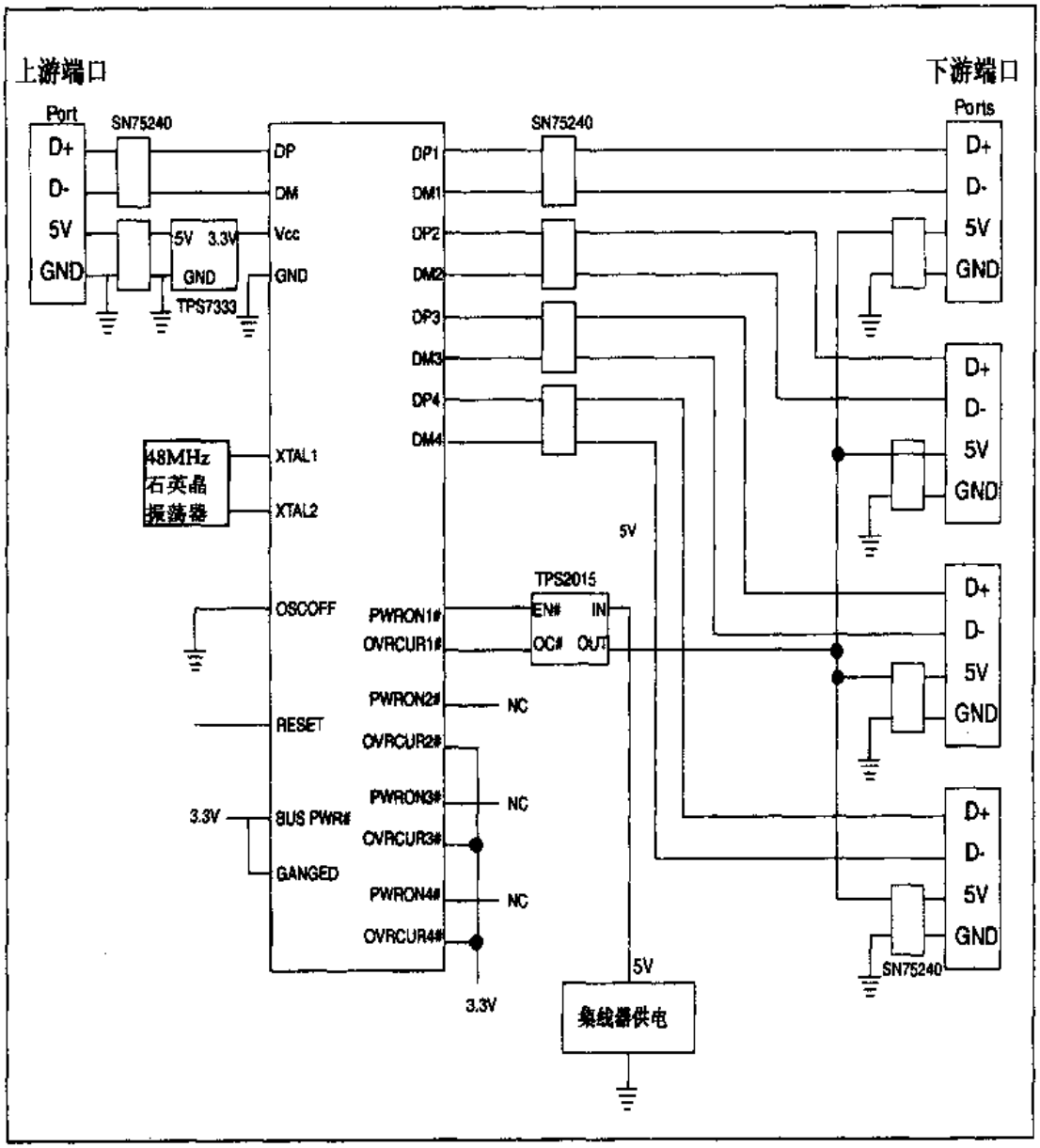


图 20-3　带组供电开关和过流保护的自供电集线器

组端口控制的总线供电的集线器

当 BUSPOWER＃管脚为地的时候，TUSB2040 是为总线供电选项配置的。TBS2015 的输入是由 USB 数据线供电的，同时它也给端口供电。图 20-4 就是一个总线供电的配置，它具有组供电开关和过流保护。

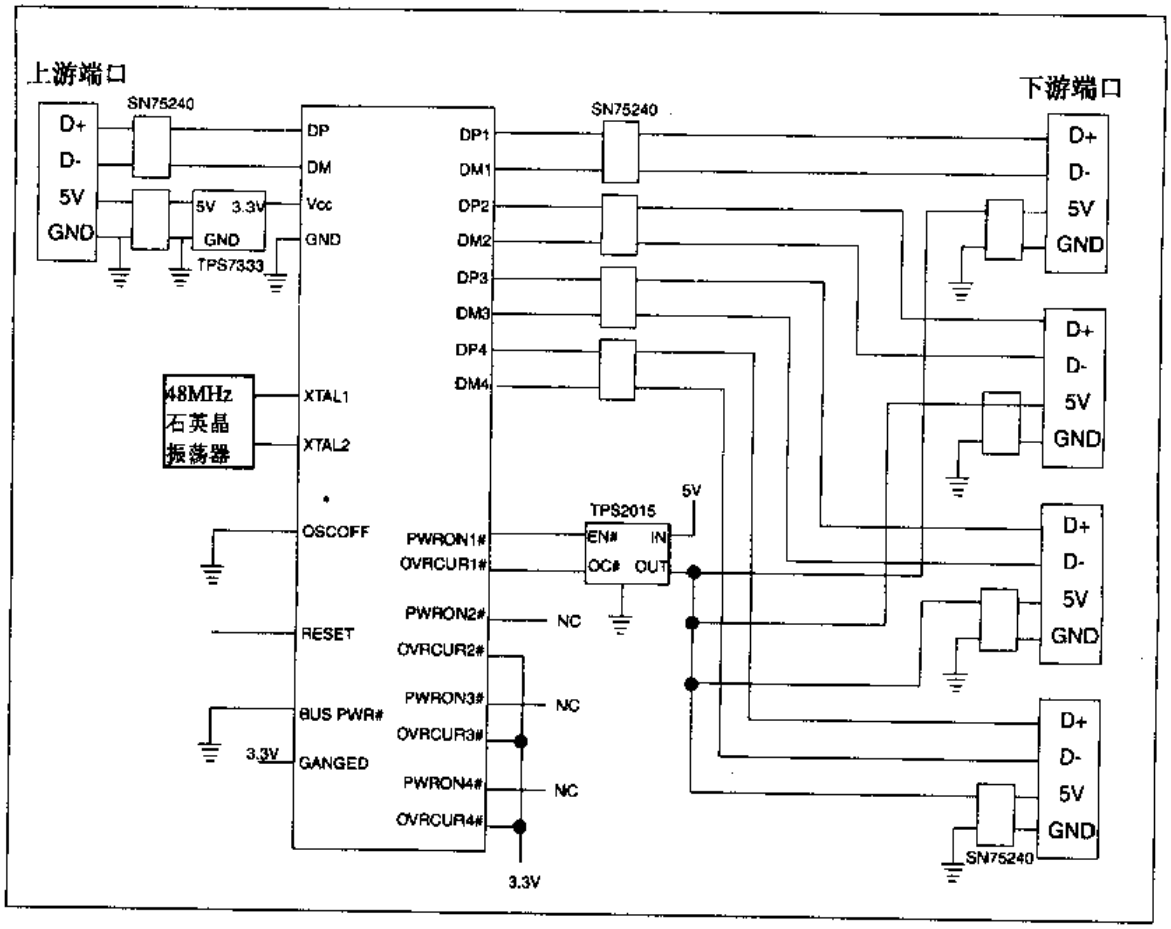


图 20-4　用于总线供电和组供电开关的 TUSB 2040 的配置

USB2040 集线器

概述

对通用产品来说，设计者可以用它们的 HP 逻辑分析器作为无源的 USB 分析器。FuturePlus 协议逻辑电路、USB 事务处理反汇编语言可以和高级触发器、HP 的逻辑分析器的系统性能软件协同工作，它们给了用户一个强劲的工具用于调试、检测和验证 USB 外围设备以及基于 USB 的系统的应变性能。

能力

- 无源 USB 总线分析；
- 完整的 USB 串行到并行解码；
- 以高速（12Mb/s）或低速（1.2Mb/s）自动检测和操作，包括动态速度改变；
- 自动 USB 复位检测；
- 在令牌包中指出的地址和端点一直保持到传输结束；
- 允许简单触发，存储指定端点的限制和性能表现；
- 支持完整的 USB 规范说明书；
- 支持所有类型的数据传输，包括同步传输；
- 支持动态热交换；
- 使用现有的 HP 逻辑分析器；
- 只要求两次加电时诊断；
- 为 HP 逻辑分析器提供的完整的配置文件和 USB 事务处理反汇编工具；
- 使用 HP 的增强触发能力、叉域分析、存储限定器、系统性能软件，用于完成

USB 的性能监视。

实现

USB 预处理器提供了两个功能：1）在 USB 和 HP 逻辑分析器之间提供了一个电器和机械的接口，用于无源总线检视。2）提供了检测点用于测量 USB 的功率和信号的逼真度。

状态分析模式

包含 FSUSB 的软件包含完整的为 HP 逻辑电路提供的配置文件和 FuturePlus USB 事务处理反编译器。在状态分析模式下，分析器的主时钟是从 USB 的协议中派生出来的。不管是在低速还是高速情况，在预处理过程中，USB 的串行数据被转换成并行数据。任何 USB 的复位是自动检测的。因为在令牌包中指定的地址和端点在传输完成之前是一直保持不变的，对特定端点的触发，条件存储和性能监视是容易的！你的 HP 逻辑分析器的增强触发功能允许你触发：

（1）任何地址和端点。

（2）任何的数据形式。

（3）任何数据 CRC。

（4）任何 USB 错误（CRC 错误，串行位填充错误，和时间片的丢失）。

（5）坏的或非法的 PID，或者任何这些的组合。

所有的 USB 循环和事务处理标识符（SOF、OUT、IN、SETUP、DATA0、DATA1、ACK、NAK、STALL、和 PRE）被协议敏感的时钟逻辑电路解码，并且被作为一些独立的位给逻辑分析器。这些包的标识符允许用户：

（1）存储所有的 USB 数据传输。

（2）仅仅存储某些包的类型。

（3）仅仅把包传输给或者传输到某个功能单元。

FuturePlus 反汇编语言使分析被存储的 USB 数据传输结果变得简单和精确。另一个优点是：FSUSB 电路的用电从逻辑分析器那里获得，而不是你的 USB 总线！

定时分析模式

FSUSB 提供了定时器或实际位流的模拟分析。使用这些模式，用户可以使在比特流中的数据和数据的实际解码联系起来。就是定时测量可以存取的信号：clk12、mdata、lbc_idle_state、pid_state、sof_or_adr_state、sof_or_ep_state、crc_state、pre_data_state、data_state、sof_tick。

交叉域分析

你需要在多个域里面分析数据吗？只要简单地使用预处理器监视 USB，然后用另一个 FuturePlus 系统预处理器监视你的另一个总线。FuturePlus 有用于 PCI，ISA，VME，VXI、PMC 和 SIMM 总线的预处理器。你可以建立自己的自定义测量系统，在总线之间的交叉域触发，这样就可以在一个显示器上看同时来自多个总线上的数据。类似的，你可以为你的主机处理器把一个预处理器和另一个逻辑分析卡相连。然后你就可以使用 HP 的软件分析器（B4620A）查看源代码、可执行代码，相应的 USB 包同时进行传输。

Powered by xiaoguo’s publishing studio
QQ:8204136