

物联网与人工智能应用开发丛书



# 微控制器USB的信号 和协议实现

USB in MCU-Signal and Protocol

“工业和信息化领域专业技术人才知识更新工程”培训用书

工业和信息化部人才交流中心 ○编著  
恩智浦（中国）管理有限公司



中国工信出版集团



电子工业出版社  
<http://www.ptpress.com.cn>

# 版权信息

书名：微控制器USB的信号和协议实现

编者：工业和信息化部人才交流中心，恩智浦(中国)管理有限公司

出版社：电子工业出版社

出版时间：2018-03

ISBN：978-7-121-33801-4

# **物联网与人工智能应用开发丛书指导委员会**

刁石京 工业和信息化部电子信息司司长

丁文武 国家集成电路产业投资基金股份有限公司总裁

王希征 工业和信息化部人才交流中心主任

魏少军 清华大学微电子所所长，中国半导体行业协会副理事长、设计分会理事长

王传臣 电子工业出版社党委书记、社长

郑力 恩智浦半导体全球资深副总裁兼大中华区总裁

# **物联网与人工智能应用开发丛书专家委员会**

主任：李宁

委员（按姓氏笔画排序）：

王永文 王自强 王志华 王朋朋 王维锋  
邢建平 吕浩 任爱光 任霞 杜高明  
李金城 何小庆 张之戈 张兴 张悦  
林国辉 林恩雪 易生海 金宇杰 单光存  
赵毅强 耿卫东 董亚峰 董雷宏 栗涛  
傅雄军 曾卫明 廉小亲

# 《微控制器USB的信号和协议实现》撰文 部

刘君杰 石宏宽 蒋宇达  
朱乐乐 张春源 胡唯唯

# 序一

中国经济已经由高速增长阶段转向高质量发展阶段，正处在转变发展方式、优化经济结构、转换增长动力的攻关期。习近平总书记在十九大报告中明确指出，要坚持新发展理念，主动参与和推动经济全球化进程，发展更高层次的开放型经济，不断壮大我国的经济实力和综合国力。

对于我国的集成电路产业来说，当前正是一个实现产业跨越式发展的重要战略机遇期，前景十分光明，挑战也十分严峻。在政策层面，2014年《国家集成电路产业发展推进纲要》发布，提出到2030年产业链主要环节达到国际先进水平，实现跨越发展的目标；2015年，国务院提出“中国制造2025”，将集成电路产业列为重点领域突破发展首位；2016年，国务院颁布《“十三五”国家信息化规划》，提出构建现代信息技术和产业生态体系，推进核心技术超越工程，其中集成电路被放在了首位。在技术层面，目前全球集成电路产业已进入重大调整变革期，中国集成电路技术创新能力和中高端芯片供给水平正在提升，中国企业设计、封测水平正在加快迈向第一阵营。在应用层面，5G移动通信、物联网、人工智能等技术逐步成熟，各类智能终端、物联网、汽车电子及工业控制领域的需求将推动集成电路的稳步增长，因此集成电路产业将成为这些产品创新发展的战略制高点。

展望“十三五”，中国集成电路产业必将迎来重大发展，特别是十九大提出要加快建设制造强国，加快发展先进制造业，推动互联网、大数据、人工智能和实体经济深度融合等新的要求，给集成电路发展开拓了新的发展空间，使得集成电路产业由技术驱动模式转化为需求和效率优先模式。在这样的大背景下，通过高层次的全球合作来促进我国国内集成电路产业的崛起，将成为我们发展集成电路的一项重要抓手。

在推进集成电路产业发展的过程中，建立创新体系、构建产业竞争力，最终都要落实在人才上。人才培养是集成电路产业发展的一个核心组成部分，我们的政府、企业、科研和出版单位对此都承担着重要的责任和义务。所以我们非常支持工业和信息化部人才交流中心、恩智浦（中国）管理有限公司、电子工业出版社共同组织出版这套“物联网与人工智能应用开发丛书”。这套丛书集中了众多一线工程师和技术人员的集体智慧和经验，并且经过了行业专家学者的反复论证。我希望广大读者可以将这套丛书作为日常工作中的一套工具书，指导应用开发工作，还能够以这套丛书为基础，从应用角度对我们未来产业的发展进行探索，并与中国的发展特色紧密结合，服务中国集成电路产业的转型升级。

王东明

工业和信息化部电子信息司司长

2018年1月

## 序二

随着摩尔定律逐步逼近极限，以及云计算、大数据、物联网、人工智能、5G等新兴应用领域的兴起，细分领域竞争格局加快重塑，围绕资金、技术、产品、人才等全方位的竞争加剧，当前全球集成电路产业进入了发展的重大转型期和变革期。

自2014年《国家集成电路产业发展推进纲要》发布以来，随着“中国制造2025”“互联网+”、大数据等国家战略的深入推进，国内集成电路市场需求规模进一步扩大，产业发展空间进一步增大，发展环境进一步优化。在市场需求拉动和国家相关政策的支持下，我国集成电路产业继续保持平稳快速、稳中有进的发展态势，产业规模稳步增长，技术水平持续提升，资本运作渐趋活跃，国际合作层次不断提升。

集成电路产业是一个高度全球化的产业，发展集成电路需要强调自主创新，也要强调开放与国际合作，中国不可能关起门来发展集成电路。

集成电路产业的发展需要知识的不断更新。这一点随着云计算、大数据、物联网、人工智能、5G等新业务、新平台的不断出现，已经显得越来越重要、越来越迫切。由工业和信息化部人才交流中心、恩智浦（中国）管理有限公司与电子工业出版社共同组织编写的“物联网与人工智能应用开发丛书”，是我们产业开展国际知识交流与合作的一次有益尝试。我们希望看到更多国内外企业持续为我国集成电路产业的人才培养和知识更新提供有效的支撑，通过各方的共同努力，真正实现中国集成电路产业的跨越式发展。



2018年1月

## 序三

尽管有些人认为全球集成电路产业已经迈入成熟期，但随着新兴产业的崛起，集成电路技术还将继续演进，并长期扮演核心关键角色。事实上，到现在为止还没有出现集成电路的替代技术。

中国已经成为全球最大的集成电路市场，产业布局基本合理，各领域进步明显。2016年，中国集成电路产业出现了三个里程碑事件：第一，中国集成电路产业第一次出现制造、设计、封测三个领域销售规模均超过1000亿元人民币，改变了多年来始终封测领头，设计和制造跟随的局面；第二，设计业超过封测业成为集成电路产业最大的组成部分，这是中国集成电路产业向好发展的重要信号；第三，中国集成电路制造业增速首次超过设计业和封测业，达到最快。随着中国经济的增长，中国集成电路产业的发展也将继续保持良好态势。未来中国将保持世界电子产品生产大国的地位，对集成电路的需求还会维持在高位。与此同时，我们也必须认识到，国内集成电路的自给率不高，在很长一段时间内对外依存度会停留在较高水平。

我们要充分利用当前物联网、人工智能、大数据、云计算加速发展的契机，实现我国集成电路产业的跨越式发展，一是要对自己的发展有清醒的认识；二是要保持足够的定力，不忘初心、下定决心；三是要紧紧围绕产品，以产品为中心，高端通用芯片必须面向主战场。

产业要发展，人才是决定性因素。目前我国集成电路产业的人才情况不容乐观，人才缺口很大，人才数量和质量均需大幅度提升。与市场、资本相比，人才的缺失是中国集成电路产业面临的最大变量。人才的成长来自知识的更新和经验的积累。我国一直强调产学研结

合、全价值链推动产业发展，加强企业、研究机构、学校之间的交流合作，对于集成电路产业的人才培养和知识更新有非常正面的促进作用。由工业和信息化部人才交流中心、恩智浦（中国）管理有限公司与电子工业出版社共同组织编写的这套“物联网与人工智能应用开发丛书”，内容涉及安全应用与微控制器固件开发、电机控制与USB技术应用、车联网与电动汽车电池管理、汽车控制技术应用等物联网与人工智能应用开发的多个方面，对于专业技术人员的实际工作具有很强的指导价值。我对参与丛书编写的专家、学者和工程师们表示感谢，并衷心希望能够有越来越多的国际优秀企业参与到我国集成电路产业发展的大潮中来，实现全球技术与经验与中国市场需求的融合，支持我国产业的长期可持续发展。

魏少军 教授

清华大学微电子所所长

2018年1月

# 序四

## 千里之行 始于足下

人工智能与物联网、大数据的完美结合，正在成为未来十年新一轮科技与产业革命的主旋律。随之而来的各个行业对计算、控制、连接、存储及安全功能的强劲需求，也再次把半导体集成电路产业推向了中国乃至全球经济的风口浪尖。

历次产业革命所带来的冲击往往是颠覆性的改变。当我们正为目不暇接的电子信息技术创新的风起云涌而喝彩，为庞大的产业资金在政府和金融机构的热推下，正以前所未有的规模和速度投入集成电路行业而惊叹的同时，不少业界有识之士已经敏锐地意识到，构成并驱动即将到来的智能化社会的每一个电子系统、功能模块、底层软件乃至检测技术都面临着巨大的量变与质变。毫无疑问，一个以集成电路和相应软件为核心的电子信息系统的深度而全面的更新换代浪潮正在向我们走来。

如此的产业巨变不仅引发了人工智能在不远的将来是否会取代人类

工作的思考，更加现实而且紧迫的问题在于，我们每一个人的知识结构和理解能力能否跟得上这一轮技术革新的发展步伐？内容及架构更新相对缓慢的传统教材以及漫无边际的网络资料，是否足以为我们及时勾勒出物联网与人工智能应用的重点要素？在如今仅凭独到的商业模式和靠免费获取的流量，就可以瞬间增加企业市值的IT盛宴里，我们的工程师们需要静下心来思考在哪些方面练好基本功，才能在未来翻天覆地般的技术变革时代立于不败之地？

带着这些问题，我们在政府和国内众多知名院校的热心支持与合作下，精心选题，推敲琢磨，策划了这一套以物联网与人工智能的开发实践为主线，以集成电路核心器件及相应软件开发的最新应用为基础的科技系列丛书，以期对在人工智能新时代所面对的一些重要技术课题提出抛砖引玉式的线索和思路。

本套丛书的准备工作始终得到了工业和信息化部电子信息司刁石京司长，国家集成电路产业投资基金股份有限公司丁文武总裁，清华大学微电子所所长魏少军教授，工业和信息化部人才交流中心王希征主任、李宁副主任，电子工业出版社党委书记、社长王传臣的肯定与支持，恩智浦半导体的任霞女士、张伊雯女士、陈劼女士，以及恩智浦半导体各个产品技术部门的技术专家们为丛书的编写组织工作付出了大量的心血，电子工业出版社的董亚峰先生、徐蔷薇女士为丛书的编辑出版做了精心的规划。著书育人，功在后世，借此机会表示衷心的感谢。

未来已来，新一代产业革命的大趋势把我们推上了又一程充满精彩和想象空间的科技之旅。在憧憬人工智能和物联网即将给整个人类社会带来的无限机遇和美好前景的同时，打好基础，不忘初心，用知识充实脚下的每一步，又何尝不是一个主动迎接未来的良好途径？

鄭力

写于2018年拉斯维加斯CES科技展会现场

# 前言

USB作为一种通用的标准通信接口，由于其良好的数据传输可靠性和低廉的成本，在各个领域都被广泛使用；而在最前沿的物联网和人工智能技术应用中，USB凭借其良好的数据传输性能和可扩展性，能够为物联网提供快速的通信和扩展接口，为人工智能大量的数据交换和传输提供重要通道。在节点、网关、路由及边缘计算的各种设备中，USB都将发挥重要作用。因而，越来越多的开发人员开始关注和学习USB相关的知识。然而，与其他的总线协议IIC或者SPI相比，USB在物理信号和协议上无疑要复杂许多。不论是硬件板级设计，还是软件驱动、应用程序的编写，难度都大大提高，这给广大开发人员的学习和使用带来很多困难。

本书是“物联网与人工智能应用开发丛书”中的一本，本系列丛书中还有另外一本《微控制器USB的技术与应用入门》讲解如何开发USB设备应用实例。而本书更关注于USB底层的信号和协议，从实践出发，结合实际的示波器波形图和USB协议分析仪提供的记录信息对USB的信号和协议进行系统的讲解。本书对在USB实际开发过程中研发人员经常碰到的一些问题进行了进一步讨论，不仅说明了USB的信号与协议“是什么”的问题，而且进一步解释了“为什么”的问题，可以为读者深入理解USB的信号和协议提供指导。

本书基于常用的两款USB控制器，详细地阐述了如何对USB控制器进行实际的编程，使之产生正确的USB信号来完成USB协议，使读者在实际的学习开发过程中能有所借鉴。

同时，本书结合USB音频类的相关知识，使用MCUXpresso SDK软件包中USB扬声器演示程序的源代码，对于本书中所涉及的技术从软件实现的角度进行讲解，使读者能够有更加直观的认识和理解。

USB相关的知识非常庞杂，本书难以涵盖USB所有的方面，由于篇幅所限，只选取了USB认证和最新的USB TYPE-C与供电技术（Power Delivery）进行了梳理和讲解，务求能让读者对于USB认证的内容、流程及相关的技术难点有清楚的认识并对最新的USB技术发展能有所涉及。

本书在归纳总结USB规范的同时，更多地讲解USB规范为什么要这么定义，帮助读者从更深的层次理解USB协议，适合希望对USB技术有深入了解的读者阅读和学习。同时，本书所讨论的话题都是在实际的工程开发过程中会碰到的具体问题，也可以为广大USB开发的工程人员提供参考和借鉴。

全书共6章，第1、第6章由刘君杰执笔，第2章由胡唯唯和张春源执笔，第3章由石宏宽和朱乐乐执笔，第4章由蒋宇达执笔，第5章由石宏宽执笔。全书由刘君杰负责统稿。胡唯唯承担了大量的绘图和后期的文字校对、录入等工作，在此表示衷心的感谢。同时，感谢恩智浦半导体高级软件开发经理曾荣给予了很多富有建设性的意见和思路，感谢在本书的编写过程中给予指导和建议的老师和工程师同事们。

特别感谢本系列丛书指导委员会及专家委员会的各位专家对本书大纲结构给予了宝贵的建议。

USB相关知识不仅复杂，所涵盖的方向也非常多，技术发展也日新月异，很多理论和应用技术问题还有待进一步深入探索和发展。由于作者水平有限，加之时间仓促，疏漏和不足之处在所难免，希望得到广大读者的批评指正。

《微控制器USB的信号和协议实现》编者团队

2017年9月

# 缩略词

- ADP : Attach Detection Protocol , 连接检测协议
- BD : Buffer Descriptor , 缓冲区描述符
- BDT : Buffer Descriptor Table , 缓冲区描述符表
- DCD : Device Controller Driver , 设备控制器驱动
- DFP : Downstreaming Face Port , 下行端口 , 如USB主机的端口
- dQH : Device endpoint Queue Head , 设备端点队列头
- dTD : Device Transfer Descriptor , 设备传输描述符
- EHCI : Enhanced Host Controller Interface , 增强主机控制器接口
- EOP : End of Packet , 包结束
- HCD : Host Controller Driver , 主机控制器驱动
- HNP : Host Negotiation Protocol , 主机角色协商协议
- KHCI : Kinetis Host Control Interface , Kinetis主机控制器接口
- iTD : Isochronous Transfer Descriptor , 同步传输描述符
- PD : Power Delivery , 供电协商
- PID : Product ID , 产品识别码
- QH : Queue Head , 队列头
- qTD : Queue Element Transfer Descriptor , 队列元素传输描述符
- SRP : Session Request Protocol , 会话请求协议
- SOF : Start of Frame , 帧开始

SOP : Start of Packet , 包开始

SiTD : Split Transaction Isochronous Transfer Descriptor , 分片事务同步传输描述符

TPL : Target Peripheral List , 目标外设列表

UFP : Upstreaming Face Port , 上行端口 , 如USB设备的端口

USB : Universal Serial Bus , 通用串行总线

USB-PET : USB Protocol and Electrical Tester , USB协议和电气的测试仪

VID : Vendor ID , 供应商识别码

# 第1章

# 深入理解USB信号

从1996年USB规范1.0正式发布至今，USB规范经历了几次较大的升级，从1.0的低速（Low Speed）模式，到1.1的全速（Full Speed）模式，再发展到2.0的高速（High Speed）模式，直到最新的3.0、3.1及3.2的超高速（Super Speed）模式。然而在微控制器的应用领域，由于应用场景和成本的限制，现阶段USB 3.0暂时未被广泛应用；而USB 1.0由于其受限的应用场景也基本不会被使用，这导致目前在微控制器领域一般只会使用USB 2.0或者USB 1.1。由于USB规范的向下兼容性，USB 2.0规范包含了USB 1.1规范的所有内容，故本章将只讨论USB 2.0信号相关的内容。本章首先介绍最基本的USB电气特性、物理信号，再结合实际的示波器波形图深入讨论USB 2.0下的高速、全速和低速的电气和信号特性，最后将详细介绍USB设备连接和断开检测、设备速度检测等一些在实际开发学习过程中会碰到的重点和难点问题。

## 1.1 USB的电气特性和信号

本节将深入讨论USB的物理层，了解USB 2.0下的高速、全速和低速的电气和信号特性，包括驱动特性、信号状态、USB的数据传输字节序、数据编解码、位填充和同步方式。

## 1.1.1 电气特性

### 1.USB电缆

USB 通过一根四线的电缆传送信号和电源，其中的 D+、D-两根线用于传输差分信号，V<sub>BUS</sub>为电源线，GND为地线（见图1-1）。

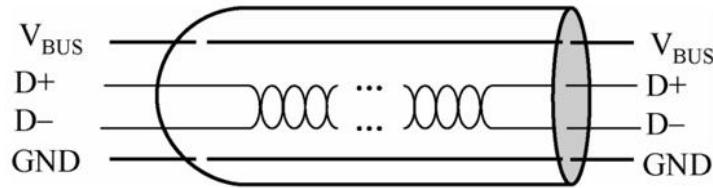


图1-1 USB电缆

为了保证信号传输的质量和抑制干扰，USB采用差分信号进行传输。差分信号可以有效地抑制在低电平时的干扰。当信号以较低电平进行传输时，比较容易受到其他信号的干扰，而差分信号则采用大小相等、极性相反的信号，所以能使信号的电平加倍，减少干扰信号对USB信号的影响。更重要的是，如果两根信号线都存在噪声干扰，差分信号的相减可以抵消噪声，因此差分信号对信号干扰有着天然的免疫力，这也是USB传输可靠性的一个保证。

在 USB 接口中，电源线和地线的触点比D+和 D-这两根数据线更加靠近 USB 接头边缘，这样的设计使得USB设备连接到USB主机时电源线和地线会先于数据线进行接触，从而避免在动态插入时电流对数据线的影响。USB连接器的内部结构如图1-2所示。

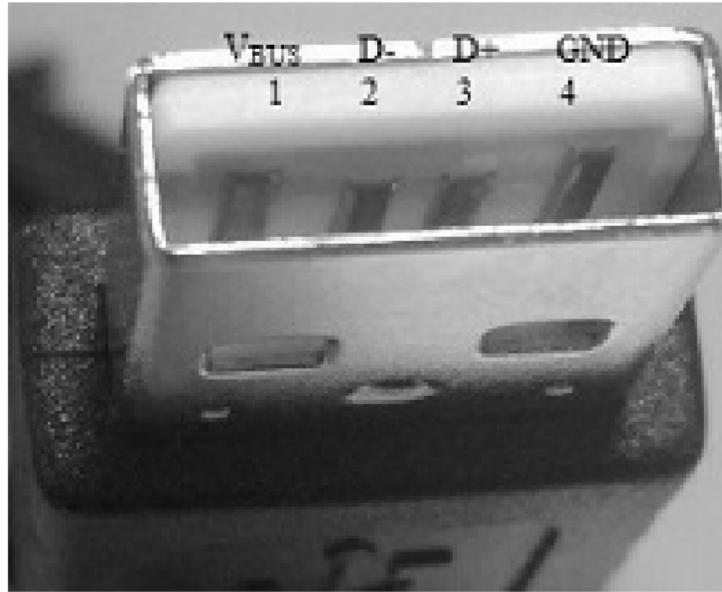


图1-2 USB连接器的内部结构

另外，驱动器的输出支持三态，而信号线在连接时默认为高阻态，这也是降低电流所带来的负面影响的一种方式。

## 2.USB的电压和电流

USB有一个很好的特性，就是USB设备可以从USB总线上取得电源，这也是USB充电的基础。USB 2.0规范中定义的供电电压范围为4.4V到5.25V，最大供电电流为500mA。从电源角度可以将USB设备分为自供电和总线供电两种设备类型，自供电USB设备不需要使用USB主机（Host）或集线器（Hub）的电源，自身有电源供应；总线供电指USB设备的电源来自USB电缆中V<sub>BUS</sub>线。如果是总线供电设备，USB规范按照设备工作时汲取的电流大小将USB设备分为低功耗设备和高功耗设备。在USB 2.0规范中规定，低功耗设备任何情况下不得汲取超过100mA的电流。高功耗设备在正确配置之前不得汲取超过100mA的电流；如果已经配置，任何情况下不得汲取超过500mA的电流。如果设备进入挂起状态，在任何情况下低功耗设备不得使用超过500μA的电流，高功耗设备不得使用超过2.5mA的电流。

每个USB设备需要在自己的配置描述符中声明其对V<sub>BUS</sub>上电流的要求，由USB主机来进行统一管理。实际上，每个USB主机都有一个

额定的电流，当主机上连接的设备过多，导致无法为新接入的设备提供足够的电流时，设备将无法被正常识别且不能工作，同时在USB主机端需要有明确的提示信息，表明该错误发生。在针对USB主机的认证过程中有一项专门的测试用例来验证USB主机是否给出了明确的信息，我们会在后续的章节中对USB认证进行更详细的讨论。

下面介绍自供电USB设备或集线器，图1-3是自供电USB设备的内部电源结构。USB控制器可以由 $V_{BUS}$ 或者本地电源供电，且最大从上行 $V_{BUS}$ 端口汲取不超过一个单位负载（100mA）的电流。需要注意的是，对于自供电设备，除了USB控制器之外，其他模块只能由本地电源供电。如果自供电USB控制器由 $V_{BUS}$ 供电，当本地电源无法提供电源时，设备依旧能够工作。

如果由本地电源向USB设备控制器供电，此时USB线缆中的 $V_{BUS}$ 线可以不连接到USB设备控制器上。但在这种情况下，利用 $V_{BUS}$ 进行USB插拔检测的机制将会失效，因此这样的设备多应用在不需要支持插拔功能的系统中。例如，平板电脑内部主板通过USB接一个自供电的WiFi模块，因为平板电脑的USB主机无须为自供电的WiFi模块提供 $V_{BUS}$ ，所以可以简化主机板级设计的电路。由于这种不带USB接口的内接USB设备情况比较特殊，本书不再讨论。

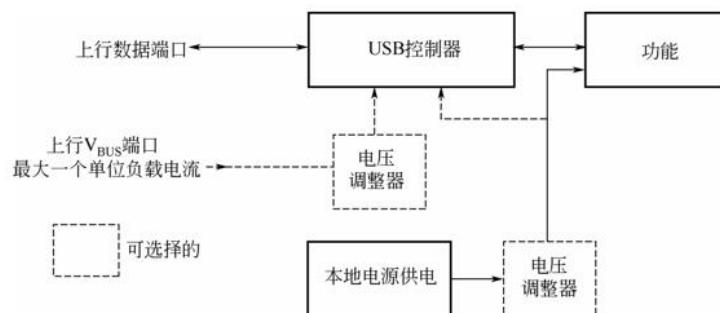


图1-3 自供电USB设备的内部电源结构

### 3.电压下降预算

因为USB的设备供电类型和总线拓扑结构较复杂，为了保证USB设备能够正常工作，协议需要对总线电压下降有一些要求。当下游端

口的设备要求提供最大电流时，集线器和各种USB传输线共同协调满足USB定义的最小电压要求。

- USB定义的连接线的阻抗为 $30\text{m}\Omega$ ，经过它们的压降为 $100\text{mV}$ 。
- 由主机或者自供电集线器端口供电的电压范围为 $4.75\text{V}$ 到 $5.25\text{V}$ ，可以支持如USB打印机等高功耗设备供电。
- 由总线供电集线器端口供电的电压范围为 $4.4\text{V}$ 到 $5.25\text{V}$ ，仅支持如USB鼠标等低功耗设备供电。
- 总线供电集线器最大允许 $350\text{mV}$ 的电压降，是指从总线供电集线器端口到它的下游端口之间的压降。
- 在 $V_{\text{BUS}}$ 上的A型插头和B型插头之间的最大电压降为 $125\text{mV}$ 。
- 所有电缆GND上的上行和下行之间的最大电压降为 $125\text{mV}$ 。
- 在上行电缆末端的连接器上的电压低至 $4.40\text{V}$ 时，所有集线器和设备必须能提供配置信息，而低功耗设备在这个最低电压时必须能正常工作。
- USB设备如果需汲取超过一个单位负载( $100\text{mA}$ )的电流，则它的最低工作电压必须为 $4.75\text{V}$ (在上行电缆末端连接器上测量)。

图1-4是最坏情况下的压降拓扑图，其中显示了由总线供电集线器驱动一个总线供电设备(低功耗USB设备)时的最坏情况下总线拓扑的最小允许电压。

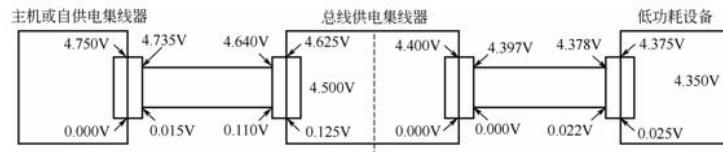


图1-4 最坏情况下的压降拓扑图

在USB认证过程中，上述提到的要求会在相应的兼容性测试中被一一测试，如果板级设计没有考虑上述要求，很有可能导致兼容性测试失败。

#### 4.浪涌电流限制

USB总线的一个基本原则就是：一个集线器或设备的插入和拔出行为，不能影响到总线上其他设备的正常工作。

当一个设备或集线器连接到总线网络中时，将会在 $V_{BUS}$ 和GND上引入一个确定数目的板上电容。此外，当采用自供电时，USB设备上的电源调整器可能会在最短时间内，供给电流到它的输出旁路电容和设备上。因此，如果没有措施去阻止它，将会产生一个很大的浪涌电流到设备上，并可能会把集线器上的 $V_{BUS}$ 拉低到它的最低工作电平之下。浪涌电流也可能发生在一个高功耗设备切换到它的高功耗工作模式时。这些问题必须通过限制浪涌电流和提供足够的电容在每个集线器上来解决，以阻止向其他端口提供超额的电流。另一个限制浪涌电流的动机是最小化接触电弧，以延长连接器接触寿命。集线器 $V_{BUS}$ 的最大压降不能超过330mV，或者设备信号的10%额定波动幅度。为了达到此要求，以下条件需要被满足：

- 能在下行电缆末端放置的最大负载为  $10\mu F$ ，且并联上  $44\Omega$  的电阻。这个 $10\mu F$ 电容代表所有直接连在设备的 $V_{BUS}$ 线上的旁路电容，再加上设备上通过电源调整器的所有可见的电容效应。这个 $44\Omega$ 电阻代表连接时由设备汲取的一个单位负载电流。
- 假如设备上需要更多的旁路电容，设备必须综合考虑 $V_{BUS}$ 浪涌电流限制，需匹配上述负载的特性。
- 集线器下行端口  $V_{BUS}$ 电源线必须在每个集线器上旁路至少  $120\mu F$  低等效串联电阻（ESR）电容。在旁路电容和连接器之间的标准旁路方式应该使电感和阻抗最小，以减小集线器下行端口  $V_{BUS}$ 上的

压降。旁路电容本身应具有较低的损耗因子，以允许在较高频率下去耦。

当一个高功耗总线供电设备从低功耗配置转换成高功耗配置时，其上行集线器 $V_{BUS}$ 上的压降不能大于330mV。该设备一般通过确保它提供的负载电容的改变不超过 $10\mu F$ 来满足这个要求。

浪涌电流的限制也是USB兼容性测试中很重要的一项，如果板级设计没有考虑上述要求，很有可能导致兼容性测试失败。

## 1.1.2 驱动特性

USB的硬件信号都是由USB的物理层来处理的，USB 2.0的物理层收发器包括一个接收器和一个发送器，由于USB规范的向下兼容性要求，USB 2.0的物理发送器同时包括一个全/低速驱动器和一个支持高速传输的电流驱动器。接收器同样包括一个支持全/低速的差分接收器和两个单端的接收器，还包括一个支持高速传输的差分接收器和高速设备连接断开接测器。

USB 2.0的物理层收发器结构如图1-5所示。

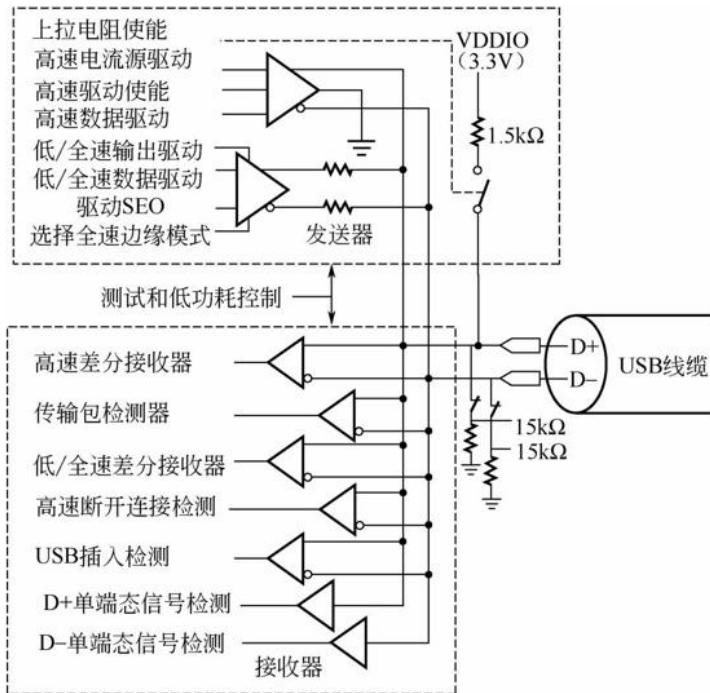


图1-5 USB 2.0的物理层收发器结构

在低速和全速模式下，当差分信号线上拉电压为3.6V，并在接有 $1.5\text{k}\Omega$ 负载上拉电阻时，如果要求驱动器输出为低时，差分信号线上的输出电平须低于 $V_{OL}(\text{max})$ (0.3V)；当差分信号线上接有 $15\text{k}\Omega$ 负载下拉电阻到地，如果要求驱动器输出为高时，输出电平必须高于 $V_{OH}(\text{min})$ 2.8V。此外，驱动器还有其他更多的要求，如驱动器须控

制斜率（上升或下降），斜率须满足眼图要求，使得辐射和串扰能够达到最小的程度。驱动器的输出还必须是三态输出，用以满足双向半双工操作。

USB在低速和全速模式下的传输速度分别只有 $1.5\text{Mb/s}$ 和 $12\text{Mb/s}$ ，使用电压驱动的方式能够满足其数据传输速度的要求。而高速模式支持的传输速度高达 $480\text{Mb/s}$ ，当USB 2.0规范在2000年刚推出时，电压高速驱动方案还并不成熟，所以高速模式采用恒流源模式和低摆幅输出（ $400\text{mV}$ ）的电流驱动方式来满足USB 2.0高速传输的速度需求。USB高速的电流信号源自一个内部的高速电流驱动器，通过电流转换流到D+或D-上，以产生高速差分信号。这个电流源的标称值为 $17.78\text{mA}$ ，而由于USB主机和USB设备在高速模式下分别挂载一个 $45\Omega$ 的等效终端电阻，两个终端电阻并联之后接到地的数据线上，因此数据线上标称高电平（ $V_{H\$OH}$ ）为 $400\text{mV}$ （ $17.78 \times 45/2$ ）。

## 1.1.3 USB信号特性

USB的数据信号线由差分驱动D+和D-两根数据线构成，根据驱动与否，差分数据线在数据传输过程中呈现静止态、差分态（0或者1）和单端态（0或者1）这几种状态。此外，在高速设备检测过程中还会出现一种特殊的电平信号，Chirp J状态和Chirp K状态，我们也会在后续章节中介绍。上述几种信号是USB协议的基本信号，所有其他的复杂数据信号状态都是由这几种基本信号组合而成的，如包开始（Start of Packet，SOP）和包结束（End of Packet，EOP）信号。

### 1.USB的基本信号

差分态、静止态和单端态这几种基本信号是了解USB中所有信号的基础，下面分别介绍低速、全速和高速模式下的这几种基本信号。

#### 1) 静止态信号（空闲态）

USB信号的静止态也就是USB的数据信号线在没有任何驱动情况下的状态。

在低/全速模式下，在USB主机或集线器的下行端口（Downstream Facing Port，DFP）的USB数据线上（D+和D-）都有一个 $15\text{k}\Omega$ 的下拉电阻。所有USB设备的上行端口（Upstream Facing Port，UFP）的USB数据线（D+或者D-中的一个）上有一个 $1.5\text{k}\Omega$ 的上拉电阻。当USB设备连接到USB主机或集线器上，且数据线没有被驱动时，由于数据线上拉电阻的存在使得该数据线的电压高于 $2.8\text{V}$ ，而另外一根没有挂载上拉电阻的数据线上的电压接近 $0\text{V}$ ，这种状态就是低/全速下的静止态，也称为空闲态。全速J/K波形图如图1-6所示，最后一条竖线右侧的波形就是全速模式的空闲态。

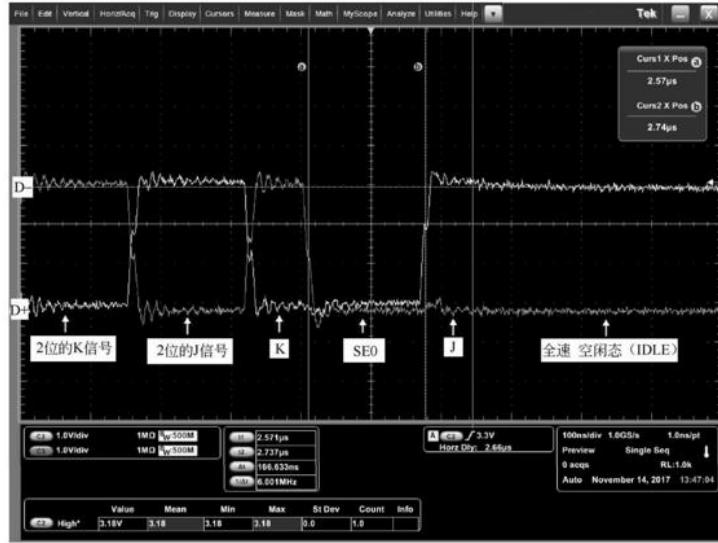


图1-6 全速J/K波形图

在高速模式下，由于驱动方式是电流驱动，使得USB的数据线在没有任何驱动情况下，D+和D-均会保持低电平状态。高速模式空闲态波形图如图1-7所示，最开始和结束一段波形就是高速信号的空闲态。

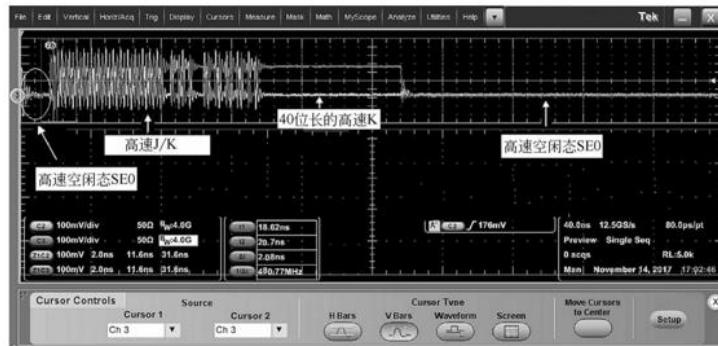


图1-7 高速模式空闲态波形图

## 2 ) 单端态信号

当D+或者D-的电平都低于主机或者集线器端口的单端低电压阈值时，称为单端态0（Single-Ended Zero，SE0），为方便起见，我们将本书中的单端态0称为SE0，此时D+与D-同时都是逻辑低电平。如图1-6所示，标线ab之间的波形就是一个两位的SE0信号。

USB的数据信号线上还有一种特殊的信号，单端态1（Single-Ended One，SE1），它和SE0是完全相反的信号，此时D+与D-都是逻辑高电位。由于该信号并没有在当前USB协议规范中被使用，本书将不详细阐述。

### 3) 数据差分态信号

USB数据信号线由D+和D-两根差分驱动数据线构成，因此USB传输数据的逻辑状态是由D+和D-两根数据线上电压的差值来表示的。

在低速或者全速模式下，当接收端检测到D+信号线的电压比D-信号线的电压高200mV时，表示差分信号“1”；当D-信号线的电压比D+信号线的电压高200mV时，表示差分信号“0”。在高速模式下，当接收端检测到D+信号线的电压比D-信号线的电压高360mV时，表示差分信号“1”；当D-信号线的电压比D+信号线的电压高360mV时，表示差分信号“0”。

此外，USB 2.0的规范中还定义了两种差分信号状态，J状态和K状态。全速和高速模式下的J状态对应差分信号“1”，而K状态对应差分信号“0”；低速模式下的J状态对应差分信号“0”，而K状态对应差分信号“1”。

在高速模式下，还存在两种特殊的信号，分别是Chirp J和Chirp K。这两种信号只会在高速模式的速度检测握手协议中出现。其基本原理如下：高速J/K信号由一个标称值为 $17.78\mu A$ 的电流源流向D+数据线或D-数据线产生，当USB主机和USB设备都进入高速模式后，两端都会挂载 $45\Omega$ 等效终端电阻，并且会断开D+数据线上 $1.5k\Omega$ 的上拉电阻，这两个终端电阻并联之后阻值为 $22.5\Omega$ ，因而可以在 D+或者 D-数据线上形成电压为 $400mV$  ( $17.78 \times 22.5$ ) 的高速J/K信号。而USB高速设备在初始连接到USB主机时默认状态为全速模式，此时USB设备端 $45\Omega$ 终端电阻并未被挂载而是挂载了 $1.5k\Omega$ 上拉电阻到D+数据线上。USB主机端 $45\Omega$ 终端电阻和USB设备D+上的 $1.5k\Omega$ 上拉电阻并联后阻值约为 $45\Omega$ ，此时将会在 D+或者 D-数据线上形成电压为 $800mV$  ( $17.78 \times 45$ ) 的差分信号，这两种差分信号就是Chirp J状态和

Chirp K状态，如图1-8所示。后面会有专门的章节对高速模式下的速度检测握手协议进行详细讲解。

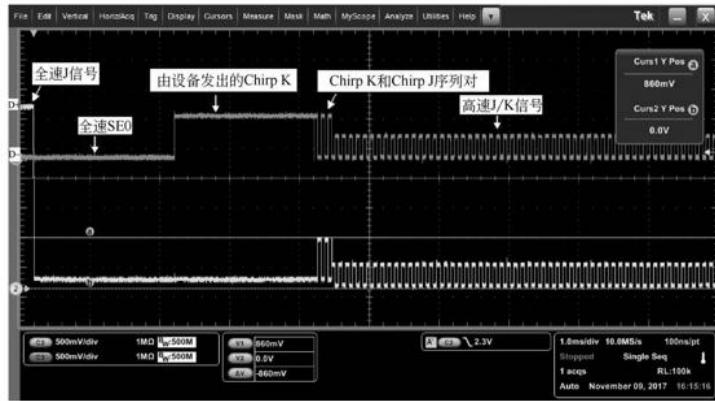


图1-8 高速J/K波形图

## 2.典型复合信号

USB中所有复杂数据状态都由前述几个基本信号组合而成，可以说复合信号就是由一个或者多个简单信号加上一个特定的时序来构成的，本节将对一些常用的信号进行介绍。

### 1 ) 低/全速模式中的包开始和包结束

USB的数据传输的基本单位就是包（Packet），USB规范定义了包开始（Start of Packet，SOP）和包结束（End of Packet，EOP）信号，使得信号接收方能够明确地鉴别一个包的开始和结束。

低/全速模式下的包开始就是一个从空闲状态（J状态）到K状态的变化过程，低速模式中包开始信号是D-从高电平变成低电平，D+从低电平变成高电平；全速模式中包开始信号是D+从高电平变成低电平，D-从低电平变成高电平。低/全速模式中包结束信号的定义是持续2位（bit）时间的SE0信号和1位时间的J状态，因为低速和全速的单个位持续时间不一样，所以包结束的SE0和J状态的持续时间也不一样。

- 对于全速传输，发射器包结束的SE0脉宽必须在160ns到175ns之间；

- 对于低速传输，发射器包结束的SE0脉宽必须在 $1.25\mu s$ 到 $1.50\mu s$ 之间。

如图1-6所示，标线ab之间两位的SE0信号和紧随其后的一位J信号构成了一个全速包结束信号。从图1-6中可以看出，标线ab之间两位的SE0信号的时间是166ns。

## 2 ) 高速模式中的包开始和包结束

正如前文驱动类型中所描述，对于高速信号，USB采用电流驱动方式来驱动。高速模式的空闲状态就是D+和D-均没有灌入电流，总线保持SE0状态。高速模式中的包开始就是从SE0状态切换到K/J状态的过程，由于包开始信号和同步信号（SYNC）是连续的，图1-9是高速模式下同步信号的包开始信号，高速模式下同步信号共有32位序列，其中包含了15个K/J序列和之后的2个K，其中标出了高速模式的包开始正好就是一个从SE0切换到高速传输同步模式（K）的过程。

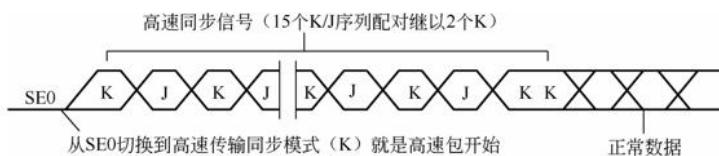


图1-9 高速模式下同步信号的包开始信号

在高速信号传输中，故意产生位填充错误来表示包结束，因此接收端需要将任何位填充错误解释为高速包结束。有些读者此时也许会有这样的疑问：如何判断一个位填充错误是真的位错误还是包结束呢？这个就由包里面的循环冗余校验码（Cyclic Redundancy Check，CRC）来判断。如果CRC校验正确，则说明这个位填充错误是高速包结束；否则就是传输错误。高速包结束的信号长度是无位填充的8位，即 $8'b01111111$ ，其中故意制造位填充错误让接收端识别出高速包结束信号。高速包结束开始于包结束之前的最后一个数据位的相反数据。如果包结束前的最后一个数据是差分信号“1”（J状态），包结束的数据就是“0”（K状态）。反之，如果包结束前的最后一个数据是“0”（K状态），那包结束的数据是“1”（J状态），如图1-10所示。需要注意的是，图1-9展示了从SE0到K的转换，而图1-10展示了从J到K的转换。

意的是，高速包结束的长度为8位只是在不是帧开始（Start of Frame，SOF）包的情况下，高速帧开始的包结束包长度有40位。

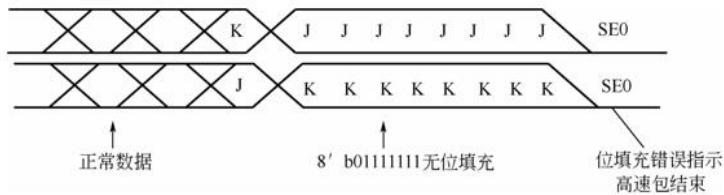


图1-10 非帧开始的高速包结束

当高速包结束完成后，驱动器停止向D+或D-线输入电流，数据线重回到高速空闲态。如图1-10所示，实际高速总线上存在以上两种不同的高速包结束，但是都是由K/J状态切换回SE0状态，所以USB协议上规定从K/J状态切换回SE0状态代表高速包结束。

高速帧开始的包结束是一个特殊的例子，帧开始的包结束是由之间没有任何转变的40个K或者J组成，是5个字节，同样无须位填充，即 $40' \text{ b}01111111\ 11111111\ 11111111\ 11111111\ 11111111$ 。因此，如果包结束之前的最后一个数据是差分信号“1”（J状态），包结束将是40位连续的K信号。反之，如果包结束之前的最后一位数据是差分信号“0”（K状态），包结束将是40位连续的J信号。这样做的目的是用来做高速断开检测，高速帧开始的包开始被延长到连续的40位且中间没有任何转换，这个长度足以保证检测的电压倍增，也只有这样才能满足高速断开的条件（差分电压差 $\geq 625\text{mV}$ ），从而能够正确检测出高速断开信号。

### 3 ) 保持连接（Keep Alive）信号

USB主机在全速和高速模式下用帧开始来防止系统进入挂起（Suspend）状态，而在低速模式下使用保持连接信号来防止USB总线进入挂起状态，低速模式下的保持连接信号就是一个低速模式下的包结束信号。全速和高速帧开始的周期分别是1ms和 $125\mu\text{s}$ ，而保持连接信号周期和全速帧开始信号周期一样，也是1ms。图1-11是低速模式下KEEP ALIVE波形图，其中的 $1.38\mu\text{s}$ 就是SE0脉宽长度（两位低速SE0的宽度）。

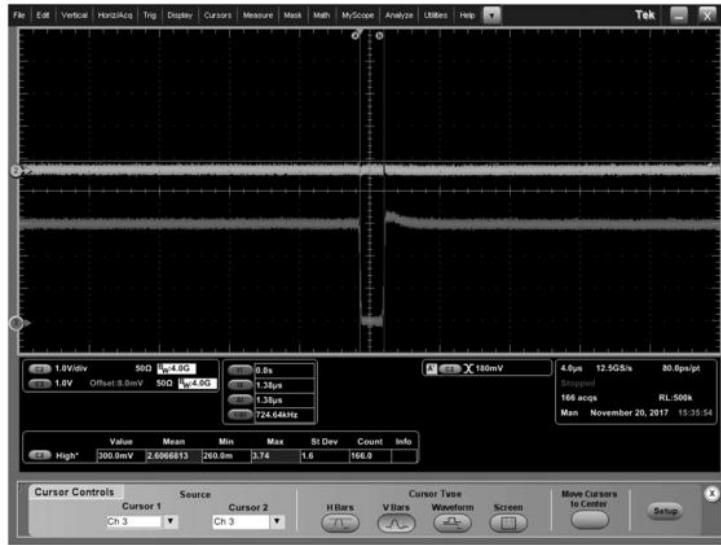


图1-11 低速模式下KEEP ALIVE波形图

#### 4) 复合信号中的SE0

如前文所述，复合信号是由一个或者多个简单信号加上一个特定的时序来构成的。USB总线复位和低/全速下行端口设备拨出都是通过检测SE0信号来判断的，只是这两个信号的持续时间不同而已。如果SE0信号产生并持续10ms以上，会被认为是USB总线的复位。如果SE0产生并持续 $2.0\mu s$ 以上，低/全速主机或者集线器下行端口就会认为设备已经拔出。总线复位的信号是由主机主动发出来的，由设备来检测的。设备拔出时产生的SE0信号是由设备的拔出产生的，是由低/全速主机或者集线器下行端口来检测这个信号的。所以这两个信号的处理对象不一样，是不会被误处理的。USB总线对时序有很严格的要求，如果信号没有满足这些时序的要求，可能会导致设备工作不正常。

### 3. 低/全速模式信号

表1-1给出了低/全速模式D+和D-的信号线状态。

表1-1 低/全速模式D+和D-的信号线状态

状态	D+	D-	(D+) - (D-) 的绝对值
差分“1”	D+ > V <sub>OH</sub> (min)	D- < V <sub>OL</sub> (max)	(D+) - (D-) > 200mV
差分“0”	D- > V <sub>OH</sub> (min)	D+ < V <sub>OL</sub> (max)	(D-) - (D+) > 200mV
单终端 0 (SE0)	D+ < V <sub>OL</sub> (max)	D- < V <sub>OL</sub> (max)	N/A
单终端 1 (SE1)	D+ > V <sub>OSE1</sub> (min)	D- > V <sub>OSE1</sub> (min)	N/A
全速数据 J 状态	D+ > V <sub>OH</sub> (min)	D- < V <sub>OL</sub> (max)	(D+) - (D-) > 200mV
全速数据 K 状态	D- > V <sub>OH</sub> (min)	D+ < V <sub>OL</sub> (max)	(D-) - (D+) > 200mV
低速数据 J 状态	D- > V <sub>OH</sub> (min)	D+ < V <sub>OL</sub> (max)	(D-) - (D+) > 200mV
低速数据 K 状态	D+ > V <sub>OH</sub> (min)	D- < V <sub>OL</sub> (max)	(D+) - (D-) > 200mV
全速空闲状态	D+ > V <sub>IHZ</sub> (min)	D- < V <sub>IH</sub> (min)	N/A
低速空闲状态	D- > V <sub>IHZ</sub> (min)	D+ < V <sub>IH</sub> (min)	N/A
恢复状态	同 K 状态	同 K 状态	同 K 状态
包开始 (SOP)	从空闲转换成 K 状态	从空闲转换成 K 状态	N/A
包结束 (EOP)	SE0 近似 2 位时间，继以 1 位时间的 J 状态	SE0 近似 2 位时间，继以 1 位时间的 J 状态	N/A

续表

状态	D+	D-	(D+) - (D-) 的绝对值
设备拔下 (下行端口)	SE0 产生并持续 2.0μs 以上	SE0 产生并持续 2.0μs 以上	N/A
设备插上 (下行端口)	空闲产生并持续 2.5μs 以上	空闲产生并持续 2.5μs 以上	N/A
总线复位	SE0 产生并持续 10ms 以上	SE0 产生并持续 10ms 以上	N/A

注：V<sub>OL</sub>是信号逻辑输出低电平电压值，最大值0.3V；

V<sub>OH</sub>是信号逻辑输出高电平电压值，最小值2.8V；

V<sub>OSE1</sub>是单终端接收器的门限电压值，最小值0.8V；

V<sub>IHZ</sub>是输入浮动高电平电压值，最小值2.7V，最大值3.6V；

V<sub>IH</sub>是输入驱动高电平电压值，最小值2.0V。

#### 4. 高速模式信号

表1-2给出了高速模式D+和D-的信号线状态。

表1-2 高速模式D+和D-的信号线状态

状态	D+	D-	(D+) - (D-) 的绝对值
高速差分 “1”	$V_{HSOH}(\min) \leq D+ \leq V_{HSOH}(\max)$	$V_{HSOL}(\min) \leq D- \leq V_{HSOL}(\max)$	需符合眼图模式定义
高速差分 “0”	$V_{HSOH}(\min) \leq D- \leq V_{HSOH}(\max)$	$V_{HSOL}(\min) \leq D+ \leq V_{HSOL}(\max)$	需符合眼图模式定义
高速数据 J 状态	同高速差分 “1”	同高速差分 “1”	同高速差分 “1”
高速数据 K 状态	同高速差分 “0”	同高速差分 “0”	同高速差分 “0”
Chirp J 状态	N/A	N/A	AC 差分电平： 目标连接器的 D+ 和 D- 差分电平必须 $\geq 300mV$
Chirp K 状态	N/A	N/A	AC 差分电平： 目标连接器的 D- 和 D+ 差分电平必须 $\geq 300mV$
高速空闲状态	N/A	N/A	DC 差分电平： $V_{HSOI}(\min) \leq (D+, D-) \leq V_{HSOI}(\max)$ AC 差分电平： 目标连接器的差分信号必须 $\leq 100mV$

续表

状态	D+	D-	(D+) - (D-) 的绝对值
高速开始包开始 (HS SOP)	从空闲态到 K/J 状态	从空闲态到 K/J 状态	N/A
高速结束包开始 (HS EOP)	从 K/J 状态到空闲态	从 K/J 状态到空闲态	N/A
高速断开状态	N/A	N/A	当差分电压 $\geq 625mV$ 时， 必须指示断开

注： $V_{HSOH}$  是高速数据传输高电平电压值，最小值 360mV，最大值是 440mV；

$V_{HSOL}$  是高速数据传输低电平电压值，最小值 -10.0mV，最大值是 10.0mV；

$V_{HSOI}$  是高速空闲状态电平电压值，最小值 -10.0mV，最大值是 10.0mV。

## 1.1.4 数据包编码

上一节我们讨论了由D+和D-数据线上的信号来构成USB的单个数据逻辑，本节将讨论如何由多个数据逻辑来组成一个USB最小传输单元—包，以及讨论数据传输字节序、编解码、同步域和位填充。

### 1.USB传输数据字节序

USB传输数据字节序是小端格式（Little Endian），在总线上先传输一个字节的最低有效位（LSB），再传输最高有效位（MSB）。当有多个字节需要传输时，总线先传输低地址的数据，然后再传输高地址的数据。

需要特别注意的是，如果处理器存储数据的字节序与USB传输字节序不一致时，一定要进行相应的字节序转换，否则USB设备将无法正常工作。

### 2.USB编解码方式

在USB总线设计中，只有D+和D-两个数据差分信号线负责传输，且并没有像其他总线那样利用一根额外的时钟线来同步数据。这是因为对于高速设备而言，时钟线和数据线同步的微小偏差，就会造成接收端有时无法满足数据采样的建立时间，进而导致接收到的数据出错。为了避免这种问题，USB使用了巧妙的编码方式将时钟信号和数据信号在相同的数据线上同时传输，这种编码格式就是反向不归零点编码（Non-Return-to-Zero Inverted Code，NRZI），为方便起见，我们将本书中的反向不归零点编码称为NRZI编码。

除了总线的空闲状态和包结束外，USB都采用NRZI编码方式，数据的发送方是NRZI的编码方，接收方就是数据的解码方。

NRZI的编码规则：

- 当遇到数据位为“1”时，数据位不作转换；
- 当遇到数据位为“0”时，数据位才作转换。

NRZI数据编码如图1-12所示。

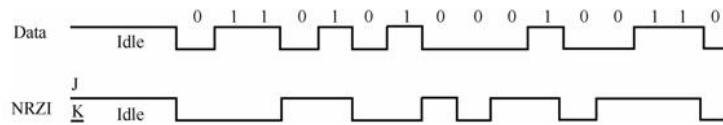


图1-12 NRZI数据编码

如图1-12所示，NRZI编码在一个时钟周期内传输一个位，按照编码规则，遇到“0”时作数据转换，而这个规则可以被接收端和发送端用于产生和确定同步时钟。

### 3.USB的同步域

因为USB并没有时钟线，信号接收端在缺少时钟同步信号情况下很难采样接收端的USB数据，而USB采用了一种方法来解决这个问题。就是让发送者先发送一个同步头，内容是8'b01010101（NRZI编码后的）的方波，接收者通过这个同步头计算出发送者的频率，从而使得接收方和发送方的采样频率达到同步，此时就可以采样得到实际的数据。图1-13是同步域的NRZI编码，编码前的数据是8'b00000001，通过NRZI编码后就成了一串方波（根据NRZI编码原则：遇0翻转遇1不变）。

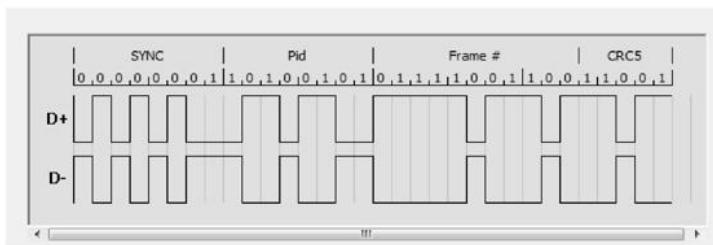


图1-13 同步域的NRZI编码

### 4.USB的位填充

USB通过在一个包的最前面加一个SYNC域来同步之后的数据信号，从而解决了大部分问题。但是仍然还存在一个问题，就是虽然接收者可以和发送者的频率匹配，但是两者之间总会有误差。举一个比较极端的例子，假如数据信号是1000位连续的逻辑1，经过USB的NRZI编码之后，就是很长一段没有变化的电平，在这种情况下，即使接收者的频率和发送者相差千分之一，也有可能会把数据采样成1001个或者999个1。

USB对这个问题的解决办法就是强制插0，被称为位填充（Bit-stuffing），发送前在第6个1后面会强制插入一个0，让发送的信号强制翻转，从而强制接收者进行频率调整。而在接收方，只要删除6个连续1之后的0，就可以恢复原始的数据了。

## 1.2 连接和断开的检测

USB协议支持热插拔的特性决定了USB主机必须能够动态地检测USB设备的连接和断开，这是USB设备和USB主机协同工作的前提。

### 1.2.1 连接状态的检测

对于USB设备或者USB主机，如果不能检测到对方的连接，USB的功能也就无从谈起，所以USB设备检测是否连接到主机以及USB主机检测是否有USB设备连接是USB协议能够进行后续工作的前提。USB设备和USB主机的整个连接过程可以分为两个阶段：连接前的初始化过程和建立连接的过程，如图1-14所示。

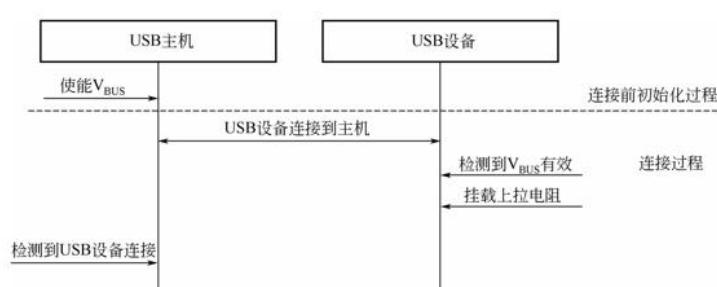


图1-14 USB设备连接流程

#### 1.连接前的初始化

对于USB设备和USB主机，检测连接的必要条件就是某些状态发生了变化。所以连接前的初始状态就非常重要，如果初始状态不对，其状态在USB插入的过程中不会发生改变，就会导致无法检测到对方的连接。

##### 1) 设备端

USB 2.0协议规范规定，任何时候USB设备都不能向V<sub>BUS</sub>供电，确保在USB设备连接到USB主机之前其V<sub>BUS</sub>小于400mV。当USB设备连接到USB主机时，USB设备就可以通过检测V<sub>BUS</sub>的变化来确定连接状态。

对于不支持V<sub>BUS</sub>检测的USB设备，USB 2.0规范并没有定义其连接的检测机制。在实际开发过程中可以使用USB主机发出的第一个复

位信号 (RESET) 来进行检测。在这种情况下，USB 设备端上拉电阻必须要预先挂载，同时需保证：

- 当  $V_{BUS}$  被移除或者低于有效值  $V_{BSVLD}$  (4.01V) 时，D+或者D-需要保持SE0状态；
- 当  $V_{BUS}$  达到有效值  $V_{BSVLD}$  (4.01V) 时，D+或者D-上的电压要达到主机检测到设备连接的最低电压2.0V。

## 2 ) 主机端

在正常工作过程中，无论USB设备是否连接，USB主机都需要维持  $V_{BUS}$  有效，并保证USB设备接入前D+和D-保持SE0状态（一般而言，主机端的下拉电阻能够保证这一点）。这样设备就可以通过  $V_{BUS}$  来检测到连接事件，进而双方开始建立连接。

## 2. 建立连接过程

当USB设备和USB主机初始化并处于正确的初始状态后，USB设备连接到USB主机上的过程如下：

(1) 由于主机端始终维持  $V_{BUS}$  有效，使得USB设备端的  $V_{BUS}$  从0V变化到5V。

(2) 对于没有  $V_{BUS}$  检测能力的 USB 设备，由于其上拉电阻已经预先挂载，当  $V_{BUS}$  达到有效值4.01V后，D+或者D-线上能够测量到2.0V以上的电压。对于有  $V_{BUS}$  检测能力的USB设备，当设备检测到  $V_{BUS}$  变化后，设备端就能够确定当前已经连接到主机，就会挂载上拉电阻，使得D+或者D-线上能够测量到2.0V以上的电压。设备就可以通过  $V_{BUS}$  确定当前已经连接到主机。

(3) 由于设备端上拉电阻的作用使得 D+或者 D-线上电压从 0V 变化到3V，USB主机一旦检测到这样的变化就能确定有设备连接。

(4) USB主机一旦检测到有设备连接，将会发出一个复位(RESET)信号让设备进入初始状态，对于没有 $V_{BUS}$ 检测能力的USB设备就可以使用这个复位信号作为设备连接的检测机制。

全速设备连接到主机时 $V_{BUS}$ /D+/D-的电压变化如图1-15所示。

在A点，USB设备和主机都处于初始状态，在USB设备上， $V_{BUS}$ 为0V，D+和D-处于SE0状态。

在B点，也就是USB设备插入USB主机时，由于USB主机一直提供 $V_{BUS}$ ，USB设备端的 $V_{BUS}$ 从0V升高到大约5V，D+和D-线保持不变。

在C点，USB设备的D+线从0V被拉高到大约3V。这一变化就是由于USB设备端挂载上拉电阻导致。该设备有 $V_{BUS}$ 检测能力，当设备检测到 $V_{BUS}$ 从0V升高到大于 $V_{BSVLD}$ (4.01V)时，会挂载D+线上的上拉电阻，图1-15中所示B点到C点间所用的时间大概为3.1ms，即为检测并挂载上拉电阻所用时间。在C点，有 $V_{BUS}$ 检测能力的USB设备已经完成连接检测。一旦USB主机检测到D+线上拉超过2V即可认为有USB设备接入，然后在C点后很短的时间内，USB主机可以完成其连接检测。

在D点，USB主机发出第一个复位信号，该信号可被没有 $V_{BUS}$ 检测能力的USB设备使用作为其连接到主机的标志。

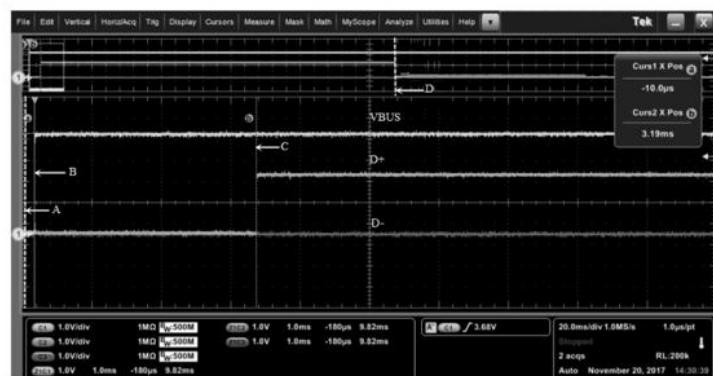


图1-15 全速设备连接到主机时 $V_{BUS}$ /D+/D-的电压变化

对于低速设备连接到USB主机的情况，其连接检测过程与全速设备的过程基本一致，唯一的区别就是全速设备的上拉电阻会挂载在D+线上，而低速设备的上拉电阻会挂载在D-线上。因此，对于低速设备，在C点时D-线上会拉。

高速设备在连接到主机时，初始阶段将会工作在全速模式并和主机建立连接，所以其连接过程和全速设备完全相同。当建立连接后，USB主机通过高速设备握手协议切换到高速模式，详细信息参考1.3.2节。

## 1.2.2 断开状态的检测

USB主机和USB设备同样需要具有检测断开的能力。USB主机如果不能检测到USB设备的移除将会导致其可用资源越来越少，比如可分配的USB设备地址，可分配的电源资源及可用的总线带宽等。而一个USB设备（特别是自供电USB设备）如果不能检测其与USB主机的断开，同样也有可能引发一些问题，如再次连接时可能不能正常工作等。USB设备断开流程图如图1-16所示。

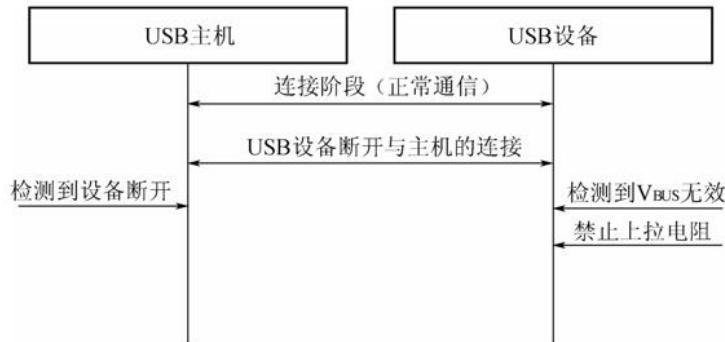


图1-16 USB设备断开流程图

### 1.USB设备端断开的检测

与USB设备端连接检测相同，设备端也是通过对 $V_{BUS}$ 的检测来实现断开的检测的。对于只使用USB总线供电的USB设备，当该设备从USB主机中移除时，设备就会处于断电状态且无法继续工作。在这种情况下，讨论断开检测是毫无意义的。后续的讨论都是建立在当设备从USB主机移除后设备还能继续工作的前提下。

对于不支持 $V_{BUS}$ 检测的USB设备，与连接检测相同，可以通过断开时D+和D-上信号变化所产生的中断再辅以实时D+和D-信号的状态来进行检测。

对于支持 $V_{BUS}$ 检测的USB设备，一旦检测到 $V_{BUS}$ 上电压低于 $V_{BSVLD}$ (4.01V)时即可认为该设备已经从USB主机上断开。此时，

USB设备端需要移除D+或D-上的上拉电阻来确保下次连接检测的初始状态是正确的。

图1-17是全速设备断开时 $V_{BUS}$ /D+/D-的电压变化，在A点， $V_{BUS}$ 开始下降后，表明USB设备跟主机的连接已经断开；在B点，USB设备检测 $V_{BUS}$ 变化后，把D+上的上拉电阻移除，整个断开检测过程完成。

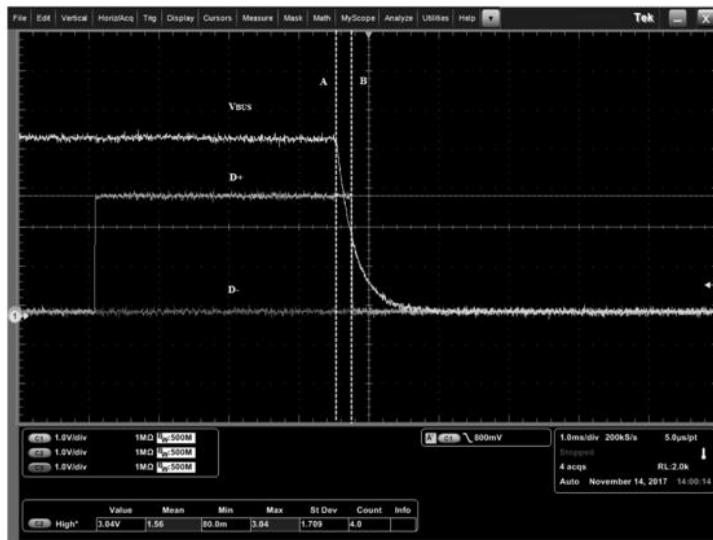


图1-17 全速设备断开时 $V_{BUS}$ /D+/D-的电压变化

对于低速USB设备，其过程与全速设备基本一致，唯一的区别是上拉电阻挂载在D-上，所以在检测断开时也是移除D-上的上拉电阻。而对于高速USB设备，由于在高速模式下，其上拉电阻已被移除，所以不需要再移除上拉电阻，详细信息参考1.3.2节。

在实际的开发过程中，USB设备端的 $V_{BUS}$ 上可能会旁接一个电容，而当电容较大时，快速拔除和插入USB设备就会出现设备不能正常工作的情况，其根本原因在于设备移除时 $V_{BUS}$ 下降比较缓慢，导致 $V_{BUS}$ 不能快速下降到 $V_{BSVLD}$ (4.01V)以下，当再次与USB主机相连时，设备端仍然处于连接状态。虽然USB主机能够检测到该设备，但设备端无法正常响应USB主机发送过来的请求。这是在开发过程中需要特别注意的。

## 2.USB主机端断开的检测

### 1) 全速和低速设备

USB设备从USB主机移除后，USB主机端的D+或D-由于主机端下拉 $15\text{k}\Omega$ 电阻的存在将会变为0V。USB 2.0协议规定，当主机端D+或D-的电压小于0.8V，并持续 $T_{DDIS}$ （最小值为 $2\mu\text{s}$ ）时间长度，USB主机就认为USB设备已经从USB主机的端口上断开。图1-18是一个全速设备从主机断开时，主机端  $V_{BUS}/D+/D-$ 状态变化图。大约在白色虚线处，D+电压降为 0.8V，主机这时就能够检测到设备的断开。低速设备断开时，主机端的D+、D-及 $V_{BUS}$ 变化与全速设备断开时是类似的，差别在于低速设备的D-会降为0V，D+一直保持低电平。

### 2) 高速设备

当USB设备工作在高速模式时，D+和D-上的信号输出电压，如高速空闲状态电压（High-speed Idle Level）、高速数字信号高电平电压（High-speed Data Signaling High），跟全速设备是不一样的，所以在检测高速USB设备断开时，D+和D-的检测标准跟全速设备是不一样的。



图1-18 主机端  $V_{BUS}/D+/D-$  状态变化图

USB 2.0协议规定，对于连接高速设备的高速主机，当D+和D-的差分信号电平差不小于625mV时，高速主机的断开检测模块就必须认为USB设备已经断开；当D+和D-的差分信号电平差不大于525mV时，高速主机的断开检测模块一定不能认为USB设备已经断开。USB主机会检测每个高速帧开始的包结束信号，当包结束电压大于检测电压，就表示有设备断开。需要说明的是，实际应用中，高速设备的断开检测电压可能在525mV到625mV之间，并且不同主机的断开检测电压可能不同。

由于主机是通过检测帧开始的包结束来判断设备是否断开，而帧开始的间隔是 $125\mu s$ ，所以，当设备被断开后，最多在 $125\mu s$ 内主机就可以检测到设备已经断开。

## 1.3 速度检测

当USB设备和主机建立连接后，下一步需要确定USB设备的速度类型。在USB 2.0协议规范中，按照传输速率划分，定义了三种USB速度模式：低速模式、全速模式和高速设备。

一般而言，低速模式下的设备称为低速设备，全速模式下的设备称为全速设备，高速模式下的设备称为高速设备，高速设备在一定条件下可以工作在全速模式。

USB主机分为全速主机和高速主机两种，但不管是哪一种主机，对连接的任意USB设备都应该能正常枚举和配置，使之能够正常工作。

不同速度模式下，USB差分信号线上的电气特性是不一样的，只有当主机和设备都工作在相同速度模式下时，其电气特性才能相互匹配，相互间发送的数据才能够被解析。因此USB设备连接到USB主机后的速度检测对后续的数据通信至关重要。

### 1.3.1 低速和全速设备识别

USB设备在开发之前就已经确定其工作模式：低速、全速还是高速。低速和全速设备在工作过程中无法改变其速度模式，而高速设备在高速握手协议失败后会工作在全速模式下，这样使得一个高速设备即使连接到一个全速的USB主机也能正常工作。全速设备线缆和电阻连接示意图如图1-19所示，低速设备线缆和电阻连接示意图如图1-20所示。全速设备在其D+线上有一个上拉电阻Rpu，如图1-19所示；而低速设备的上拉电阻Rpu在D-上，如图1-20所示。设备上的上拉电阻Rpu的阻值是 $1.5\text{k}\Omega \pm 5\%$ ，主机或hub的端口上的下拉电阻Rpd是 $15\text{k}\Omega \pm 5\%$ 。

当USB主机与全速或者低速设备连接后，因为上拉电阻的存在，D+或D-的电压会上拉到3V左右。USB主机根据哪根差分线上有上拉来判断接入USB设备的类型，如果D+上有上拉，主机就把USB设备识别为全速设备；如果D-上有上拉，主机就把USB设备识别为低速设备。

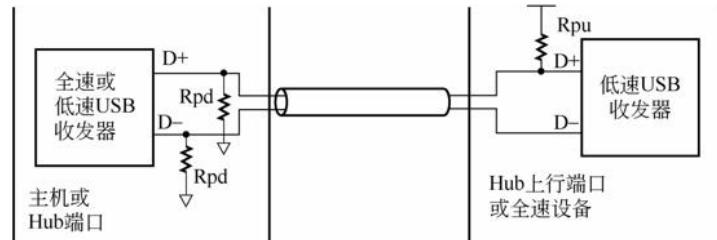


图1-19 全速设备线缆和电阻连接示意图

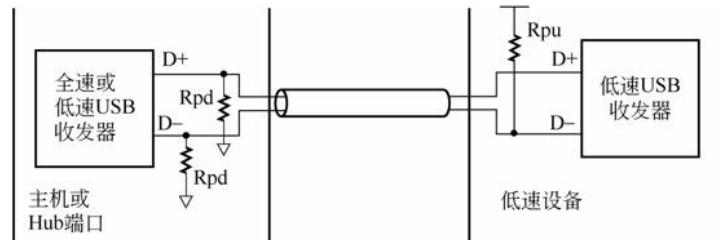


图1-20 低速设备线缆和电阻连接示意图

### 1.3.2 高速设备识别

在高速模式下，不管是USB设备还是USB主机，在D+和D-数据线上都要有一个 $45\Omega$ 等效对地电阻。

高速设备在初始连接时，是以全速模式连接到主机的，即高速设备的D+上挂载了上拉电阻，主机检测到全速模式的设备连接后，会对该设备进行复位（RESET）。工作在全速模式下的USB设备收到复位信号后，会主动发起高速模式握手协议进行速度识别。如果主机支持高速设备，会和该设备交互完成高速模式握手协议，此时USB主机和USB设备都工作在高速模式下；如果主机不支持高速设备，握手协议会失败，该设备也不会切换到高速模式，此时USB主机和USB设备都工作在全速模式下。同样，一个全速设备连接到一个高速主机，由于设备无法发起高速模式握手协议，最终主机和设备都会工作在全速模式下。

需要注意的是，USB设备连接到主机后，主机必须在第一时间对USB设备进行复位，复位信号至少持续10ms，整个高速握手协议是在复位过程中完成的，且在复位信号结束之前完成。USB主机和USB设备在复位结束后，都必须确定自己的速度模式。

图1-21是高速模式下握手信号波形图。图1-22是对图1-21信号的局部放大，主要集中于主机发送的3对KJ这一部分，图1-22中的C和D与图1-21中的C和D指向的是同一位置。

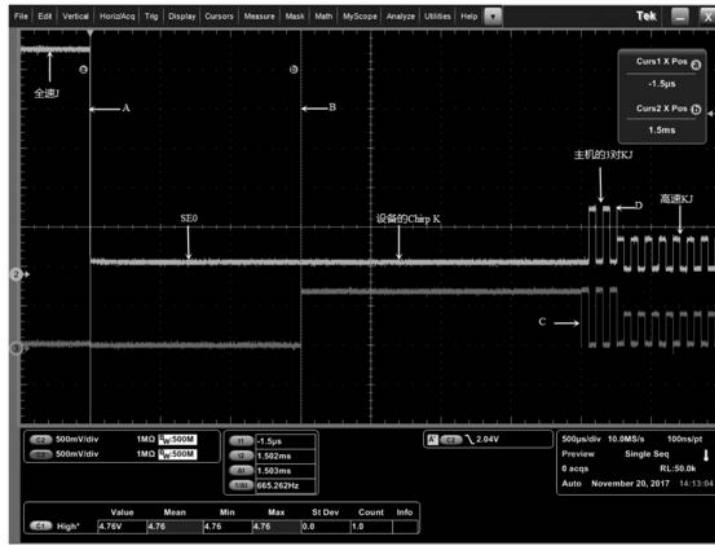


图1-21 高速模式下握手信号波形图

### 1 ) 全速J状态

在A时间点之前，USB设备完成其连接检测流程，其D+上的上拉电阻挂载，因此总线状态处于全速J状态。

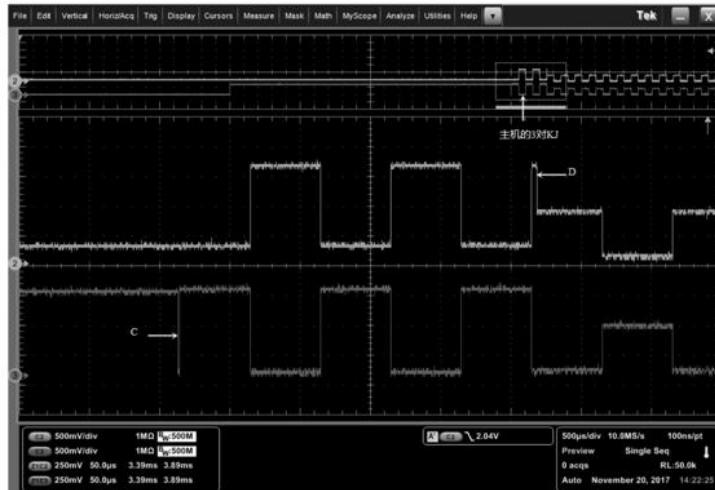


图1-22 主机KJ序列放大图

### 2 ) SE0状态

在 A 点，连接检测完成后，USB 主机发出复位信号，使总线处于 SE0 状态。USB 设备必须在检测到总线 SE0 状态后不少于  $2.5\mu s$  且不超过

3ms的时间内发起高速模式握手协议。图1-21中A时间点和B时间点之间就是SE0状态，持续时间大约1.5ms。

### 3 ) 设备产生Chirp K

在开始握手协议前，设备确保其D+线上的上拉电阻已经挂载，并且 $45\Omega$ 等效对地电阻没有挂载。之后，设备端向 D-输入 17.78mA 的电流，主机端 $45\Omega$ 等效对地电阻与主机的下拉电阻 ( $15k\Omega \pm 5\%$ ) 并联，并联后电阻大约还是  $45\Omega$  电阻，总线上就会产生一个约 800mV ( $17.78 \times 45$ ) 的电压，这就是Chirp K信号。Chirp K信号持续至少1ms，并且在复位信号开始后的7ms内结束。

根据USB协议，USB设备在检测到SE0后的 $2.5\mu s$ 和3ms之间开始高速握手信号，根据示波器测量，该高速USB设备是在检测到复位信号大约1.5ms后，开始产生Chirp K信号，Chirp K幅度大约为750mV，持续时间大约为2ms。从图1-22中可以看到，在C点有一个脉冲，这个脉冲是一个SE0状态，表示设备的Chirp K信号结束，根据示波器测量，该SE0信号持续时间大约为 $1.2\mu s$ ，主机检测到SE0，就认为设备的Chirp K结束，然后主机会发送连续的KJ序列。

### 4 ) 主机产生KJ序列

如果主机能够检测到的Chirp K信号持续时间不少于 $2.5\mu s$ ，则主机认为该Chirp K有效。如果检测不到设备的Chirp K，则主机必须一直驱动总线为SE0，直到复位信号结束后。高速模式握手协议结束后，主机和设备都将工作在全速模式。

根据USB协议，在检测到USB设备Chirp K信号结束后的 $100\mu s$ 内，主机必须发送交替变换的Chirp K 和Chirp J信号，信号必须连续，中间不能有空闲状态。每一个单独的Chirp K和Chirp J信号的持续时间为 $40\mu s$ 到 $60\mu s$ 之间。如果USB总线上持续3ms没有活动，USB设备就会进入挂起状态，为了避免总线在空闲态的时间太长导致设备进入挂起状态，Chirp JK序列需要持续发送直到复位信号结束前的 $100\mu s$ 到 $500\mu s$ 为止。

从图1-22中可以看出，在C点的SE0持续了大概 $1.2\mu s$ ，也就是说主机是在设备的Chirp K结束后 $1.2\mu s$ 后开始发送Chirp KJ序列的，从图1-22中可以测量到K或J信号的持续时间是 $50\mu s$ 左右。

主机端完成Chirp KJ序列后，要一直驱动总线为SE0状态，直到复位信号结束。设备完成Chirp K信号后，要等待主机的Chirp信号，设备要至少检测到3对KJ信号，才会认为这是一个有效的Chirp信号。

### 5 ) 设备挂载高速终端电阻

在设备端检测到3对有效的KJ序列后，在 $500\mu s$ 内，必须移除D+的上拉电阻，并挂载高速 $45\Omega$ 等效对地电阻，进入高速模式。此时，USB设备高速 $45\Omega$ 等效对地电阻与原来主机端的 $45\Omega$ 等效对地电阻并联后，其等效电阻变为 $22.5\Omega$ ，此时D+和D-线上的电平大概为 $400mV$  ( $22.5 \times 17.78$ )。

从图1-22中可以看到，D点的J信号很短，这是因为协议规定，Chirp J或者K有效的保持时间至少是 $2.5\mu s$ ，根据示波器测量，D点J信号的持续时间为 $3.8\mu s$ ，所以设备端在J信号持续约 $3.8\mu s$ 后才认为J信号有效。而D点的J信号是设备检测到的第三个Chirp J信号，USB设备在检测到第三个Chirp J信号有效后，立刻移除其D+的上拉电阻，并挂载高速 $45\Omega$ 等效对地电阻，导致电压幅度降为 $400mV$ 左右，因此主机发送的后续信号序列的幅值为 $400mV$ 左右，变为正常的高速JK信号。至此，整个高速模式握手协议完成。

如果USB设备检测不到3对KJ信号，则在USB设备完成自己的Chirp K信号后的 $1ms$ 到 $2.5ms$ 内，该设备要恢复到全速模式的默认状态，直到复位信号结束。最终USB设备和主机都将工作在全速模式下。

## 第2章 深入理解USB协议

第1章介绍了USB 2.0信号相关的内容，在此基础上，本章借助USB协议分析仪的日志详细地介绍了USB 2.0协议规范的内容，并对实际开发过程中可能会碰到的具体问题进行了专项讨论，以加深读者对协议的理解。本章主要包括枚举、端点停止（ENDPOINT STALL）、挂起和恢复，以及OTG相关内容。

## 2.1 通信协议

USB通信协议是一个主从关系的协议，多个USB设备通过USB总线及集线器（Hub）连接到一个USB主机上，形成一个星状网络，图2-1是USB网络拓扑。在这个网络中，USB主机管理和配置接入的每一个设备，从逻辑层面来说，每一个加入的USB设备（除集线器外）都只能看到USB主机，其他的设备都不可见，所以相互之间不能直接通信。

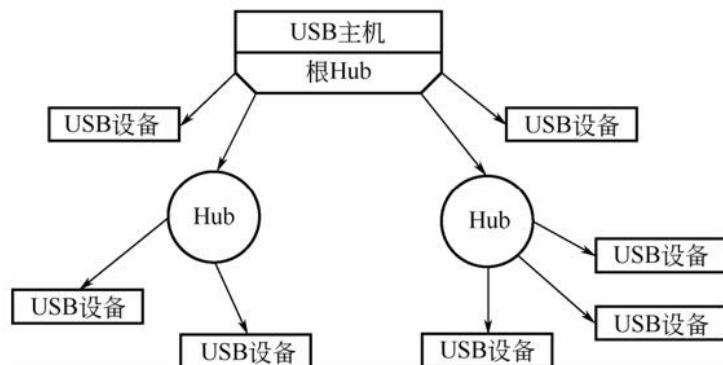


图2-1 USB网络拓扑

USB主机会给每一个接入的设备分配一个唯一的地址，发送给USB设备的数据会在总线上广播，而USB设备会对总线上的数据进行过滤，使其只接受发送给自己的数据。一个USB主机最多只能分配127个地址（1~127），并且最多只能连接5层集线器，如图2-2所示。需要注意的是，第5层Hub只能接USB设备，不能再接Hub。

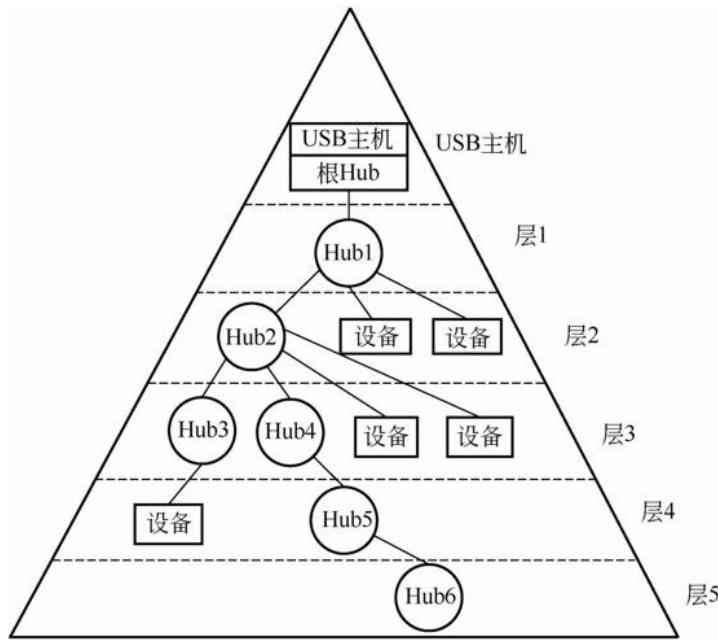


图2-2 集线器层次

USB主机和USB设备间的通信是通过管道（Pipe）进行的，如图2-3所示。在USB协议中，管道是一个逻辑概念，其在USB主机这一端实际上就是一组缓冲区，用于管道中数据的收发；而在USB设备这一端，管道对应着一个特定的端点（Endpoint，ENDP），每一个端点都是一个<索引，方向>二元组。USB设备地址、端点索引和端点方向的组合可以唯一确定USB主机和USB设备间的通信。

在USB协议中，在一个管道中传输的基本单元就是包（Packet），它由多个逻辑0和1（也就是差分信号“0”和“1”）构成。多个包构成了一个事务（Transaction），多个事务构成了一个传输（Transfer）。本节将基于包、事务和传输来讲解USB的相关协议，主要包括各类型包及其组成、各类型事务、各类型传输、数据翻转、USB设备相关请求（Request）。

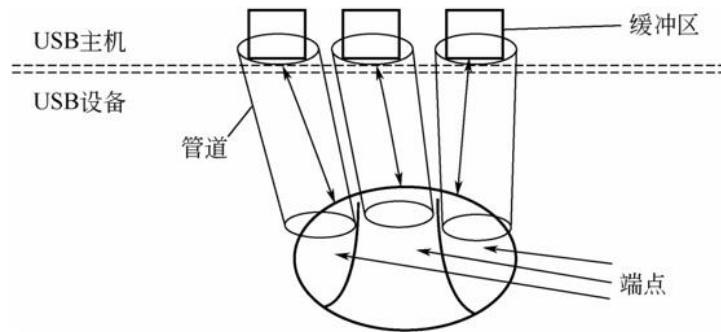


图2-3 USB管道和端点

## 2.1.1 概要

USB协议层定义了不同类型的包，以及每种包的格式和用途，这些包是USB协议的最小单位。USB协议中定义了三大类包，包括令牌（Token）包、数据包和握手包。各类型包通过包标识符（PID）作进一步区分，为简化起见，我们将PID分别为Setup、IN、OUT、Ping等的令牌（Token）包称为Setup包、IN包、OUT包、Ping包等。相应地，数据包和握手包也按此命名，如数据包包括DATA0包、DATA1包等，握手包包括ACK包、NAK包、STALL包等。

USB协议层进一步定义了如何使用不同的包的组合来完成一个事务，根据事务中令牌（Token）包的类型，将事务分为 Setup 事务、IN 事务、OUT 事务、Ping 事务等。

在USB协议中，IN和OUT从USB主机的角度来表示数据流动的方向，IN 表示对于 USB 主机来说数据的流动是“IN”的方向，也就是我们通常说的USB主机接收数据、USB设备发送数据的情况；OUT表示对于USB主机来说数据的流动是“OUT”的方向，也就是我们通常说的USB主机发送数据、USB设备接收数据的情况。

所有事务全部由USB主机发起，并以一个令牌包开始，如USB主机使用Setup包来开始一个Setup事务，然后使用DATA0 数据包来发送数据，接着由USB设备使用ACK包来完成对数据的确认，最终实现一个带有握手机制的Setup事务来保证数据传输的正确性。事务可以实现传输数据、发送命令等功能，各种不同事务的组合实现了USB协议层中定义的传输（Transfer）。

图 2-4 是传输、事务和包的关系图，其给出了一次成功的控制传输的示例，控制传输一般用来实现一条USB协议定义的请求（REQUEST）。从图2-4中可以看出，一个控制传输包括多个事务，Setup事务用来将请求发送给USB设备；IN事务用于USB设备向USB主机返回请求的响应结果，或者向USB主机传输请求所要求的数据；

OUT事务使用握手机制对之前的数据事务的正常结束进行确认。每个事务包括多个包，每个包由不同的位域构成，后续章节会对各种包、事务、传输等有更加详细的讲解。

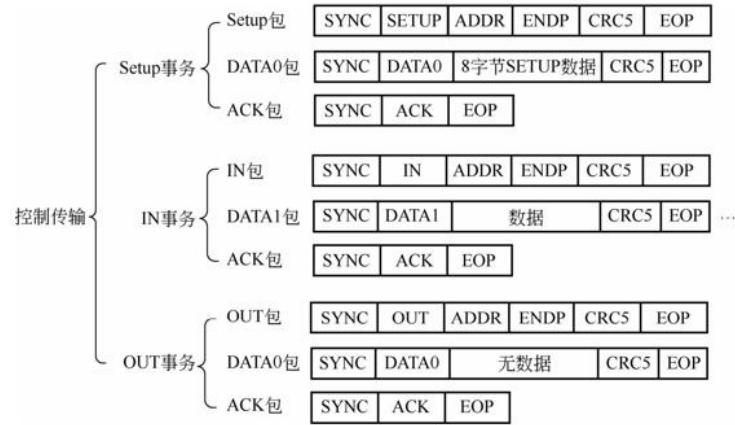


图2-4 传输、事务和包的关系图

## 2.1.2 包

### 1. 包的组成

下面将介绍 USB 基本传输单元—包。USB 协议中定义了多种类型的包、包括令牌 (Token) 包、数据包和握手包。令牌包全部由 USB 主机发出，其他类型包的方向根据实际情况而定。所有包都以同步位域SYNC开始，并以包结束 (EOP) 信号结束。各类型包组成位域各不相同，主要包括包标识符 (PID)、包目标地址 (ADDR)、包目标端点 (ENDP)、数据、帧索引和循环冗余校验码 (CRC)，下面将对各个位域进行详细讲解。

#### 1 ) 包标识符

包的类型通过长度为8位的包标识符指定，其中包括4位的包类型位域和与其对应的4位校验位域，校验位域是包类型位域的补码。表2-1是各类型包的PID位域。其中特殊用途包包括一些应用于特定场合下的令牌包和握手包。

表2-1 各类型包的PID位域

包 类型	包 名 称	PID 值
令牌包	OUT	4'b0001
	IN	4'b1001
	SOF	4'b0101
	SETUP	4'b1101
数据包	DATA0	4'b0011
	DATA1	4'b1011
	DATA2	4'b0111
	M DATA	4'b1111
握手包	ACK	4'b0010
	NAK	4'b1010
	STALL	4'b1110
	NYET	4'b0110
特殊用途包	PRE	4'b1100
	ERR	4'b1100
	SPLIT	4'b1000
	PING	4'b0100

#### 2 ) 包目标地址

每个USB设备都有一个由USB主机管理分配的地址，USB设备在被USB主机分配一个地址前将会使用默认的地址0。在收到USB主机分配的地址后，USB设备将会使用这个新的非0地址，直到USB设备被拔出、掉电或者复位。

USB主机发送的包将在总线上广播，所有在总线上的设备需要使用自己的设备地址对广播的令牌包进行过滤，如果令牌包的地址和端点号与其自身地址不匹配，该包将不会被设备忽略。

包目标设备的地址长度只有7位，这也是为什么一个USB主机最多只能管理127个USB设备的原因。所以USB设备的地址由地址位域ADDR和端点位域ENDP构成，如果令牌包的地址和端点号与USB设备的地址不匹配，USB设备会忽略该令牌包。同时，发送给一个未初始化的端点的令牌包也会被忽略。

地址位域的长度是7位，每个USB设备只有一个唯一的地址，USB枚举过程中使用默认的地址0。在收到设置地址请求后，USB设备会被分配一个新的非0地址，如果USB设备不被拔出、掉电或者复位，该地址将一直存在。

### 3 ) 包目标端点

USB设备和USB主机间的通信建立在一个个单独的管道上，每个管道在USB设备上都对应一个端点（Endpoint），因此在总线上传输的每一个包都需要指定其目标端点号。同时端点也是区分方向的，USB主机与USB设备端点1的IN方向建立的通信管道和与端点1的OUT方向建立的通信管道是两个不同的通信管道。端点号在包中由4比特表示，因此USB设备最大可以支持16个双向端点。其中端点0专门用于控制传输，被称为控制端点，任何USB设备都支持一个默认的控制端点0，其他端点和具体的功能相关。

### 4 ) 数据

数据长度的范围是 0 ~ 1024 字节，不同传输类型在不同速度模式下的数据位域的长度各不相同。

### 5 ) 帧索引

帧索引的长度是11比特，该位域的初始值为0，由USB主机对其进行递增，当达到最大值2047时则重新从0开始计数。详细信息见2.1.3节。

### 6 ) 循环冗余校验码

USB协议规定只有令牌（Token）包和数据包具有循环冗余校验码，其他的包没有循环冗余校验码。另外，令牌包使用 5 位循环冗余校验码，数据包使用16位循环冗余校验码。

## 2.包格式

USB 2.0协议中定义了4种包，各类型包的格式各不相同，下面将介绍令牌包、数据包、握手包和帧开始（Start of Frame，SOF）包的具体格式。

### 1 ) 令牌包

令牌包包括包标识符（PID）、包目标设备地址（ADDR）、包目标端点（ENDP）和CRC5校验位域。图2-5是Setup令牌包格式，包括同步头和包结束符，其中包标识符是0xB4，表明这是一个Setup包，目标设备地址和目标端点都是0，CRC5是包目标设备地址和包目标端点的校验值。

Sync	SETUP	ADDR	ENDP	CRC5	EOP
00000001	0xB4	0	0	0x08	233.330ns

图2-5 Setup令牌包格式

令牌包必须由 USB 主机发出，指明了其后续的数据包的目标地址和端点。

## 2 ) 数据包

数据包包括包标识符 ( PID ) 位域、数据位域和CRC16校验位域。图2-6是数据包格式，其中包标识符是0xC3，表明这是一个DATA0数据包，数据位域包含8个字节的有效数据，CRC16是数据位域的校验值。

Sync	DATA0	Data	CRC16	EOP
00000001	0xC3	8 bytes	0xBB29	250.000ns

图2-6 数据包格式

由于数据包中并没有包含其传输的目标地址信息和端点信息，所以数据包必须紧跟在令牌包之后。

## 3 ) 握手包

握手包中只有包标识符 ( PID ) 位域，图2-7是ACK握手包格式，其中PID是0x4B，表明这是一个ACK握手包。

Sync	ACK	EOP
00000001	0x4B	266.660ns

图2-7 ACK握手包格式

握手包跟随在令牌包或数据包之后，组合成一个完整的事务，是对一次事务完成的确认，USB主机或者USB设备会根据事务的完成状态返回相应的握手包。根据包标识符的不同，握手包可分为ACK包、NAK包、STALL包、NYET包和ERR包，其含义如表2-2所示。

表2-2 各类型握手包含义

握手包类型	含 义
ACK	数据被正确接收
NAK	因某些原因数据无法被接收，如内存空间不足
STALL	设备请求不支持该请求，或者端点被挂起
NYET	仅用于高速模式中，表示本次数据被成功接收，但是无法接收下一次数据
ERR	在 Split 事务中使用，表明出现错误

#### 4) 帧开始 (SOF) 包

在USB拓扑结构中，USB主机会每隔一定的时间向所有连接上的USB设备广播SOF包，两个SOF包之间被称为一个帧或者微帧。对于低速和全速模式，使用的是帧；对于高速模式，使用的是微帧。帧的时间间隔是  $1.00 \pm 0.0005$  ms，微帧的时间间隔是  $125 \pm 0.0625\mu s$ 。SOF包的作用如下：①USB主机不断向总线上广播 SOF 包，以防止 USB 设备进入挂起状态；②所有的事务必须在一个帧/微帧中进行，一个事务不能跨越两个帧或者微帧，通过控制帧/微帧中事务的个数，可以实现USB带宽控制。③用作USB主机和USB设备间时钟同步、调整的一种机制。

SOF包包括包标识符（PID）、帧索引和5位循环冗余校验码（CRC5）位域及EOP位域。图2-8是SOF包格式，其中PID位域是固定值0xA5，帧索引当前值是512，CRC5是帧索引的循环冗余校验码。

Sync	SOF	Frame#	CRC5	EOP
00000001	0xA5	512	0x02	250.000ns

图2-8 SOF包格式

USB主机中的根集线器会将SOF包广播给所有的USB设备，所以SOF包并不需要包含目标设备地址和目标端口信息。

## 2.1.3 事务

USB 通信中的另一个重要的单元就是事务。事务包括 3 种包、分别是令牌（Token）包、数据包和握手包。其中数据包是可选的，如用于设置地址的事务就只含有令牌包和握手包。令牌包的方向一定是 USB 主机到 USB 设备，数据包的方向和令牌包的类型有关，握手包的方向和数据包的方向相反。如果事务中没有数据包，握手包的方向是 USB 设备到 USB 主机。

单独的包并没有错误检测机制，传输过程中可能出现各种情况导致接收方出现错误，事务实际上就是利用令牌包、数据包和握手包实现了一个带有错误反馈机制的通信，使得 USB 传输成为一个安全、可靠的传输。

### 1. Setup 事务

图2-9是一次成功的Setup事务，对应Transaction 0，其作用是USB主机向USB设备发送用于获取设备描述符的标准USB设备请求。该Setup事务包含：

- USB 主机发送给 USB 设备的一个 Setup 包（Packet 28），该包用以指明本事务是一个 Setup 事务（0xB4），以及事务通信的对象为指定设备的端点0（图2-9中的 ADDR 字段和ENDP 字段）；
- USB 主机发送给 USB 设备的用来传输特定请求的数据包（Packet 29），数据长度为固定的8字节，且 Setup 事务必须使用 DATA0 包；
- USB 设备返回给 USB 主机的表示所有数据已经全部收到的 ACK 包（Packet 30）。

Transaction	F	SETUP	ADDR	ENDP	T	D	Tp	R	bRequest	wValue	wIndex	wLength	ACK
0	S	0xB4	0	0	0	D->H	S	D	0x06	0x0100	0x0000	64	0x4B
		Packet	H	F			Sync	SETUP	ADDR	ENDP	CRC5	EOP	
G0	28			S			00000001	0xB4	0	0	0x08	233.330 ns	
G0	29	Packet	H	F			Sync	DATA0		Data	CRC16	EOP	
G0	30			S			00000001	0xC3	80 06 00 01 00 00 40 00	0xBB29	250.000 ns		
		Packet	↑ D	F			Sync	ACK		EOP			
				S			00000001	0x4B		266.660 ns			

图2-9 一次成功的Setup事务

图2-10是Setup事务的处理机制，其中图2-10（a）是USB设备视图，图2-10（b）是USB主机视图。对于图2-10（a），USB主机向USB设备发送一个Setup包和DATA0包，如果USB设备接收到DATA0包没有任何错误（如位填充或者CRC错误），则会向USB主机返回一个ACK包。如果在此过程中，Setup包或DATA0包出现错误，USB设备会丢弃收到的包并且不作任何反应。

对于图2-10（b），如果USB主机向USB设备发送的Setup包或DATA0包出现错误，USB主机就不会收到USB设备返回的ACK包，USB主机会等待一定时间直到产生超时，此时USB主机将对该Setup事务进行重传。

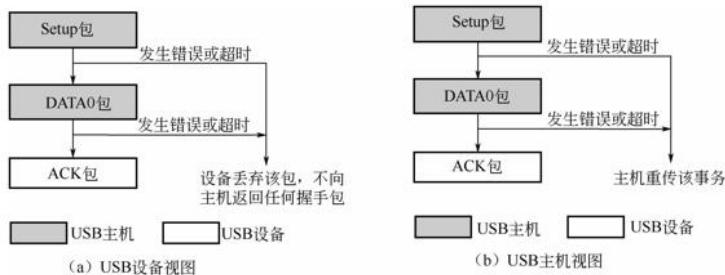


图2-10 Setup事务的处理机制

## 2.OUT事务

图2-11是OUT事务的处理机制，其中图2-11（a）是USB设备视图，图2-11（b）是USB主机视图。对于图2-11（a），USB主机向USB设备发送OUT包和数据包，如果USB设备接收到的数据包没有任何错误（如位填充或者CRC错误），并且数据包的PID和自身包序列匹配，则USB设备返回ACK包；如果低速或全速的USB设备因自身内存空间不够等原因无法接收USB主机发送来的数据，则会向USB主

机返回一个NAK包；如果USB设备的OUT端点被停止（Halt），那么USB设备向USB主机返回一个STALL包；对于控制传输和批量传输，如果高速的USB设备成功接收到USB主机发送来的数据但无法接收下一个数据包中的数据，则会向USB主机返回一个NYET包（后续章节会有关于 NYET 包的详细介绍）；需要注意的是，在此OUT事务中，如果OUT包或者数据包发生错误，USB设备会忽略它们并且不会作任何反应。

对于图2-11（b），USB主机会按照上述各种情况收到ACK包、NAK包、STALL包或者NYET包。如果USB主机向USB设备发送的OUT包或数据包出现错误，USB主机就不会收到USB设备返回的任何握手包，USB主机会等待一定时间直到产生超时，此时USB主机将对该OUT事务进行重传。同时，如果USB设备向USB主机返回一个NAK握手包，USB主机也会对该OUT事务进行重传。需要注意的是，在高速模式下的OUT事务中，USB主机收到USB设备返回的NAK包后，USB主机会转而使用Ping事务。

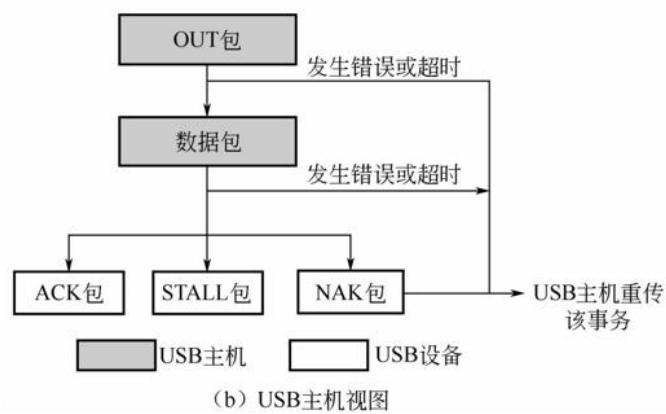
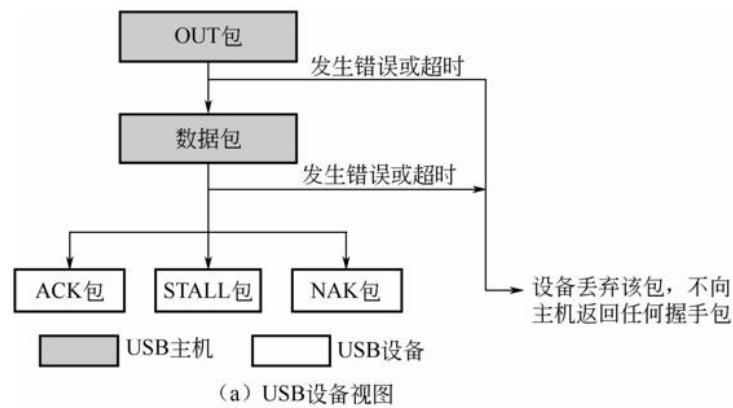


图2-11 OUT事务的处理机制

图2-12是一次成功的OUT事务，对应Transaction 25242，其作用是USB主机向USB设备发送数据。该OUT事务包含：

- USB主机发送给USB设备的一个OUT包（Packet 75805），该包用以指明本事务是一个OUT事务（0x87），以及事务通信的对象为指定设备的端点2（图2-12中的ADDR字段和ENDP字段）。
- USB 主机发送给 USB 设备的用来传输实际数据的数据包（Packet 75806），此例子中USB主机发送给USB设备的数据长度是8字节。

需要说明的是，数据长度不能超过端点所支持的最大包长度。

- USB 设备返回给 USB 主机的表示所有数据已经全部收到的ACK 包（Packet 75807）。

Transaction	F	OUT	ADDR	ENDP	T	Data	ACK
25242	S	0x87	4	2	T	0 8 bytes	0x4B
		Packet	H	F	Sync	OUT ADDR ENDP CRC5 EOP	
		75805	S		00000001	0x87 4 2 0x00	250.000 ns
		Packet	H	F	Sync	DATA0 Data CRC16 EOP	
		75806	S		00000001	0xC3 12 34 56 78 90 AB CD EF 0xF544	266.660 ns
		Packet	↑D	F	Sync	ACK EOP	
		75807	S		00000001	0x4B	266.660 ns

图2-12 一次成功的OUT事务

### 3.IN事务

图2-13是IN事务的处理机制，其中图2-13（a）是USB设备视图，图 2-13（b）是USB主机视图。对于图2-13（a），USB主机向USB设备发送一个IN包，USB设备向USB主机发送数据包。在 USB主机成功收到后，USB主机向USB设备返回一个ACK包。在该IN事务中，如果IN包或数据包发生错误，USB主机并不会返回任何握手包。USB主机向USB设备发送IN包后，如果USB设备没有数据要发送给USB主机，那么USB设备会直接向USB主机返回一个NAK包。如果USB设备中用于接收IN包的端点被停止，那么USB设备直接向USB主机返回一个STALL包。

对于图2-13 ( b ) , 如果USB 主机向USB 设备发送的IN包出现错误 , USB主机就不会收到USB设备返回的任何握手包和数据包 , USB主机会等待一定时间直到产生超时 , 此时 USB 主机将对该 IN 事务进行重传。另外 , 如果 USB 设备返回给 USB 主机的数据包出现错误 , USB 主机接收不到该数据包并一直等待直到超时 , 此时USB主机也会对该IN事务进行重传。

图2-14是一次成功的IN事务 , 对应Transaction 25243 , 其作用是USB主机向USB设备请求数据。该IN事务包含 :

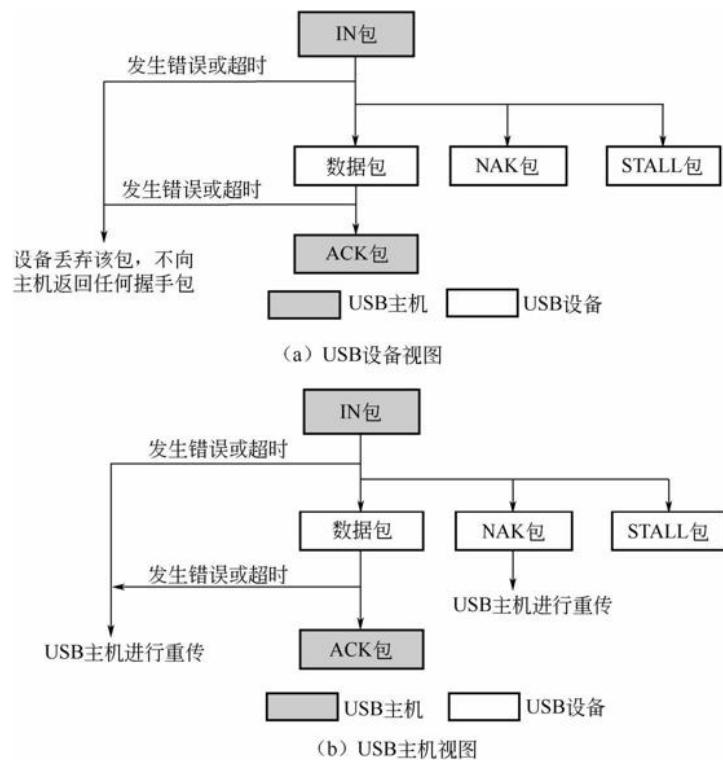


图2-13 IN事务的处理机制

Transaction	F	IN	ADDR	ENDP	T	Data	ACK
25243	S	0x96	4	1	0	8 bytes	0x4B
	O	Packet	H	I	F	Sync	IN ADDR ENDP CRC5 EOP
	O	75809				00000001	0x96 4 1 0x19 266.660 ns
	O	Packet	D	I	F	Sync	DATA0 Data CRC16 EOP
	O	75810				00000001	0xC3 12 34 56 78 90 AB CD EF 0xF544 266.660 ns
	O	Packet	H	I	F	Sync	ACK EOP
	O	75811				00000001	0x4B 266.660 ns

图2-14 一次成功的IN事务

- USB主机发送给USB设备的一个IN包（Packet 75809），该包用以指明本事务是一个IN事务（0x96），以及事务通信的对象为指定设备的端点1（图2-14中的ADDR字段和ENDP字段）。

- USB设备发送给USB主机的用来传输实际数据的数据包（Packet 75810），此例子中USB设备发送给USB主机的数据长度是8字节。需要说明的是，数据长度不能超过端点所支持的最大包长度。

- USB主机返回给USB设备的表示所有数据已经全部收到的ACK包（Packet 75811）。

#### 4.Ping事务

在低速和全速模式下，USB主机使用控制传输或者批量传输向USB设备发送数据时，如果USB设备因某些原因（如内存空间不足）无法接收数据，USB设备会向USB主机返回NAK包。如果USB设备一直无法接收USB主机发送来的数据，那么USB主机会一直向USB设备发送OUT包和数据包，USB设备会一直返回NAK包。这样，USB总线上会一直存在OUT包、数据包和NAK包，总线利用率会很低。在低速或者全速模式下，最大传输包长度只有64字节，但是对于高速设备，最大传输包长度可以达到512字节，总线带宽会被大量地浪费。因此，在高速模式下，使用一种新的事务—Ping事务，来解决这个问题。USB主机在发送OUT事务之前先使用Ping事务来确定USB设备是否能够接收数据，如果设备不能接收数据，USB主机会在一定时间后再次使用Ping事务，直到USB设备能够接收数据为止。由于Ping事务只有令牌包和握手包，与原有的OUT事务相比，能够节省大量的总线带宽，从而提高了总线利用率。

USB主机首先会向USB设备发送一个OUT包和包含要发送的数据的数据包，USB设备有可能会向USB主机返回一个ACK包、NAK包或者NYET包。如果USB设备返回NAK包或者NYET包，之后USB主机就会开始使用Ping事务。

图2-15是Ping事务的处理机制，其中图2-15（a）是USB设备视图，图2-15（b）是USB主机视图。对于图2-15（a），USB主机向USB设备发送一个Ping包。如果此时USB设备能够接收USB主机将要发来的数据，则会向USB主机返回一个ACK包；如果USB设备无法接收USB主机将要发送来的数据，则会向USB主机返回NAK包；如果Ping包发生错误，USB设备不会返回任何握手包。

对于图2-15（b），如果USB主机向USB设备发送的Ping包出现错误，USB主机就不会收到USB设备返回的任何握手包，USB主机会等待一定时间直到产生超时，此时USB主机将对该Ping事务进行重传。

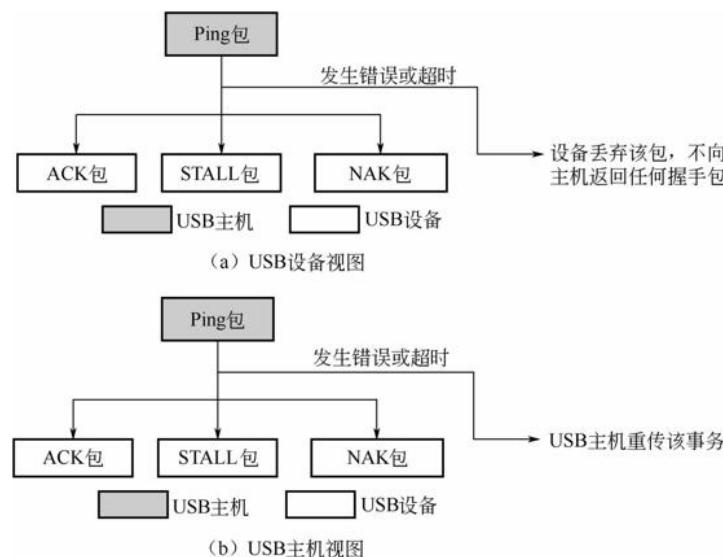


图2-15 Ping事务的处理机制

图2-16是用于发送数据的Ping事务，包括USB设备返回NYET包的OUT事务（Transaction 70）和USB设备返回ACK包的Ping事务（Transaction 73），其中OUT事务包含：

- USB主机发送给USB设备的一个OUT包（Packet 3314），该包用以指明本事务是一个OUT事务（0x87），以及事务通信的对象为指定设备的端点2（图2-16中的ADDR字段和ENDP字段）；
- USB主机发送给USB设备的用来传输实际数据的数据包（Packet 3315），此例子中USB主机发送给USB设备的数据长度是31

字节，需要说明的是，数据长度不能超过端点所支持的最大包长度；

- USB设备返回给USB设备NYET包（Packet 3316），表明USB设备可以接收当前USB主机发送来的数据，但是无法接收USB主机之后发送来的数据。

此时，USB主机开始使用Ping事务来检查USB设备是否已经可以接收数据，该Ping事务包含：

- USB主机发送给USB设备的一个Ping包（Packet 3325），该包用以指明本事务是一个 Ping 事务（0x2D），以及事务通信的对象为指定设备的端点2（图2-16中的ADDR字段和ENDP字段）；
- USB设备返回给USB主机的一个ACK包（Packet 3326），表示USB设备已经可以接收之后USB主机发送来的数据。

在此之后，USB主机将继续使用OUT事务向USB设备发送数据。

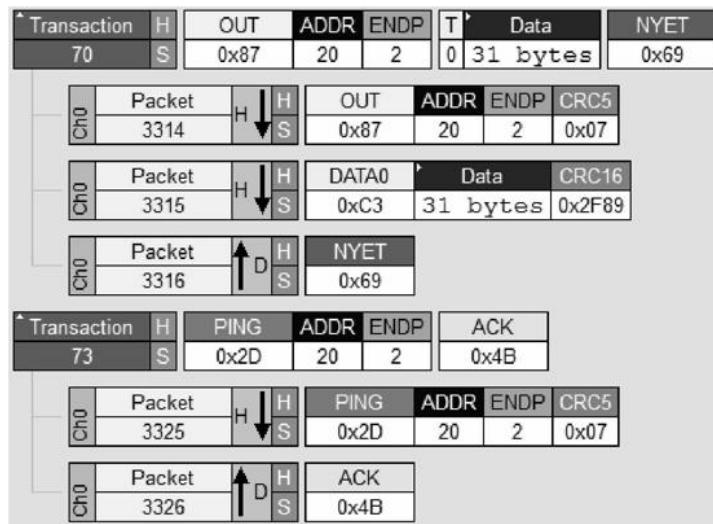


图2-16 用于发送数据的Ping事务

## 5.Split事务

当一个低速或全速的USB设备连接到一个高速的Hub上时，高速USB主机需要使用 Split 事务来解决数据从高速模式到低速和全速模式

的转换，Split事务分为SSplit（Start Split）事务和CSplit（Complete Split）事务两种类型。一个SSplit事务和一个CSplit事务两者联合使用来完成一个低速或全速模式下的常规的事务。

图2-17是发送数据的Split事务，包括高速USB主机和高速Hub之间的一个SSplit事务和一个CSplit事务，两个事务完成高速USB主机通过高速Hub向低速或全速USB设备发送数据。其中，SSplit事务用于高速USB主机通过高速Hub向低速或全速USB设备发送数据，CSplit事务用于高速Hub将USB设备返回的握手包返回给USB主机。

该SSplit事务包含：

- USB主机发送给高速Hub的一个Split包（Packet 18848），该包用以指明本事务是一个Split事务（0x87）。同时图2-17中Split包的S位域（值为0）进一步指明了这是一个SSplit包，因此协议分析仪进一步将这个Split事务解析为SSplit事务。另外，图2-17中Split包中的Hub Addr和Port指明了USB主机的事务通信的对象是哪个高速Hub的哪个端口。
- USB主机发送给高速Hub的一个OUT包（Packet 18849），该包用以指明高速Hub和低速/全速USB设备之间是一个OUT事务（0x87），以及高速Hub的下行端口的事务通信的对象为指定设备的端点0（图2-17中的ADDR字段和ENDP字段）。
- USB主机发送给高速Hub的用来传输实际数据的数据包（Packet 18850），此例子中USB主机发送给USB设备的数据长度是0字节，需要说明的是，数据长度不能超过端点所支持的最大包长度。
- 高速Hub返回给USB设备的表示所有数据已经全部收到的ACK包（Packet 18851）。

高速USB主机和高速Hub之间完成此次SSplit事务后，高速Hub会将SSplit事务的OUT包和数据包转发给指定端口的USB设备，USB设备

根据接收情况向高速Hub返回指定的握手包。高速USB主机开始使用CSplit事务来获取这个握手包，该CSplit事务包含：

- USB主机发送给高速Hub的一个Split包（Packet 18852），该包用以指明本事务是一个Split事务（0x87）。同时图2-17中Split包的S位域（值为1）进一步指明了这是一个CSplit包，因此协议分析仪进一步将这个Split事务解析为CSplit事务。另外，图2-17中Split包中的Hub Addr和Port位域指明了USB主机的事务通信的对象是哪个高速Hub的哪个端口。
- USB主机发送给高速Hub的一个OUT包（Packet 18853），该包用以指明高速Hub和低速/全速USB设备之间是一个OUT事务（0x87），以及事务通信的对象为指定设备的端点0（图2-17中的ADDR字段和ENDP字段）。
- 高速Hub返回给USB主机ACK包（Packet 18854）。需要注意的是，这个ACK包是低速/全速USB设备的，具体是ACK还是NAK等与设备和Hub之间的执行结果有关。



图2-17 发送数据的Split事务

图2-18是接收数据的Split事务，用于高速USB主机接收来自低速或全速USB设备的数据，也包括SSplit事务和CSplit事务，其执行流程与高速USB主机发送数据流程类似，此处不再赘述。

Transaction		H	SSplit	Hub Addr	Port	IN	ADDR	ENDP	ACK		
684	S	Bulk	23	1	0x96	26	1			0x4B	
O	Packet	H	H	SPLIT	SC	Hub Addr	Port	S E ET CRC5 Pkt Len			
O	18893			0x1E	0	23	1	0 0 Bulk 0x09 9			
O	Packet	H	H	IN	ADDR	ENDP	CRC5	Pkt Len			
O	18894			0x96	26	1	0x11	8			
O	Packet	H	D	ACK		Pkt Len					
O	18895			0x4B		6					
Transaction		H	CSplit	Hub Addr	Port	IN	ADDR	ENDP	T	Data	
685	S	Bulk	23	1	0x96	26	1	1	13 bytes		
O	Packet	H	H	SPLIT	SC	Hub Addr	Port	S U ET CRC5 Pkt Len			
O	18896			0x1E	1	23	1	0 0 Bulk 0x12 9			
O	Packet	H	H	IN	ADDR	ENDP	CRC5	Pkt Len			
O	18897			0x96	26	1	0x11	8			
O	Packet	H	D	DATA1		Data	CRC16	Pkt Len			
O	18898			0xD2	13 bytes	0x91CB	21				

图2-18 接收数据的Split事务

在Split事务中，如果高速Hub无法处理Split事务，高速Hub会向USB主机返回一个NYET包；如果低速或全速USB设备和高速Hub之间的事务一直没有完成，高速Hub也会向USB主机返回一个NYET包；当高速Hub端口下的低速或全速USB设备出现错误时，高速Hub会向USB主机发送一个ERR包。

## 2.1.4 传输

包是USB传输的最小单位，事务是USB带反馈机制的可靠传输的最小单位，基于事务，USB协议定义了传输（Transfer）用于完成一组具有特定目的的事务，其中任意一个事务失败，则整个传输都会失败。

USB协议中定义了四种传输类型，包括控制传输、中断传输、批量传输和同步传输，表2-3是各类型传输（Transfer）支持的最大包长度（字节），USB设备在所有的速度模式下都支持控制传输，而低速模式不支持同步传输和批量传输。

表2-3 各类型传输（Transfer）支持的最大包长度（字节）

传输类型 \ 速度模式	低速	全速	高速
控制传输	8	8/16/32/64	64
同步传输	不支持	1023	1024
中断传输	0~8	0~64	0~1024
批量传输	不支持	8/16/32/64	512

下面将对各类型传输进行详细介绍。

### 1. 控制传输

一个控制传输（Control Transfer）一般用于完成一个特定的请求（Request），通常这些特定的请求由USB规格说明书定义，特别是USB设备枚举过程，全部都由控制传输来完成，其完成的请求（Request）也全部都由USB规格说明书所定义。后续章节会详细讲解其中每一个请求的含义及用法，这里我们只选取第一个获取设备描述符GetDescriptor（）的请求来介绍控制传输的相关知识。

一个控制传输包括三个阶段，分别是设置阶段（Setup Phase）、数据阶段（Data Phase）和状态阶段（Status Phase），其中数据阶段是可选的。设置阶段包含一个Setup事务；数据阶段是零个、一个或多个

的IN或者OUT事务；状态阶段是一个数据长度为0的IN或者OUT事务。这里需要注意的是，如果数据阶段是IN事务，则状态阶段是一个OUT事务；如果数据阶段是OUT事务，则状态阶段是一个IN事务；如果控制传输没有数据阶段，则状态阶段是一个IN事务。

简单地说，设置阶段由USB主机向USB设备发送特定的请求（Request）。对于某个特定的请求，其数据阶段中需要传输的数据大小和方向都是确定的。但不同的请求，其数据阶段也不尽相同，有可能是USB主机向USB设备发送数据，也有可能是USB设备向USB主机回传数据，也有可能该请求不需要发送数据，这时就没有数据阶段。状态阶段则是对数据阶段的反馈，这也是为什么状态阶段数据流的方向与数据阶段数据流方向相反的原因。

如果在数据阶段中USB主机需要从USB设备端获取数据，所传输的数据量大小是由USB主机所指定的，但是有可能USB设备并没有足够的数据提供给USB主机，所以数据阶段可以在以下两种情况下结束：

- USB主机接收到其所指定大小的数据；
- USB主机接收到一个数据包，其数据量大小小于端点0的最大包长度。

对于第二种情况还有一种更为特殊情况就是USB设备端所发送的数据量小于USB主机所指定的大小，但是USB设备端所发送的最后一个数据包刚好是端点0的最大包长度。例如，USB主机想要从USB设备端获取255个字节，但是USB设备端只有64个字节的数据，并且USB设备端点0的最大包长度刚好也是64。在这种情况下，USB设备端必须要多发送一个0字节长度的数据包通知USB主机没有更多的数据可以发送。

图2-19是获取设备描述符的控制传输，这个控制传输用于USB主机获取USB设备的设备描述符。其中包含一个Setup事务、一个IN事务

和一个OUT事务，分别对应于一个控制传输的设置阶段、数据阶段和状态阶段。

在本例中，控制传输的设置阶段就是一个 Setup 事务，即图 2-19 中的 Transaction 0。该事务用于向USB设备发送用于特定的请求。数据阶段只包含一个IN事务，即图2-19中的Transaction 4。实际上数据阶段可以包含多个IN事务，在数据阶段中IN或者OUT事务的多少取决于需要传输的数据量和USB设备所支持的控制传输的最大包长度。例如，在本例中，需要传输的数据量是18个字节，而USB设备所支持的控制传输中数据长度是64，所以只需要一个IN事务即可处理。若传输的数据长度多于64，则需要多个IN事务才能完成。需要注意的是，当一个控制传输中有多个事务时，最后一个事务之前的事务的数据长度都是端点所支持的最大包长度，最后一个事务的数据长度小于或等于端点所支持的最大包长度。如果最后一个事务的数据长度小于端点支持最大包长度，则认为数据传输完。

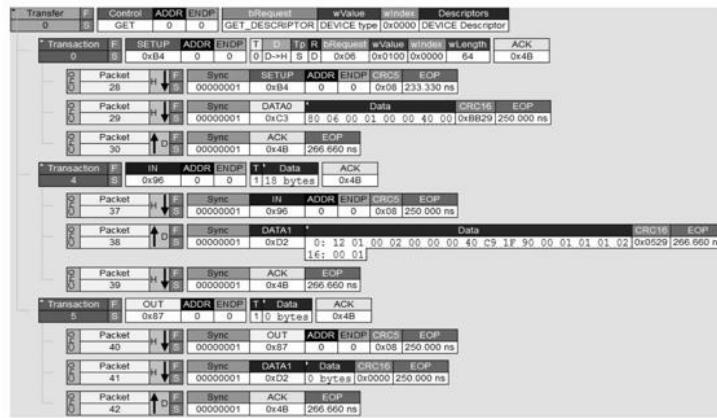


图2-19 获取设备描述符的控制传输

在本例中，由于数据阶段是IN事务，所以状态阶段就是一个OUT事务，并且OUT事务的数据长度一定是0。另外，需要注意的是，控制传输的设置阶段的数据包的PID必须是DATA0，数据阶段的数据包的PID必须以DATA1开始并且不断在DATA0和DATA1之间切换，状态阶段的数据包PID必须是DATA1，关于这一部分内容将在2.1.6节进行详细介绍。

对于控制传输，我们需要考虑一种非常特殊的情况，如果控制传输的状态阶段是IN事务，在这个IN事务中，如果USB主机向USB设备返回的ACK包失败了怎么办？对于USB主机，其会认为这个ACK握手包已经被USB设备成功接收，从而继续开始发送Setup包执行下一个控制传输。对于USB设备，它虽然没有接收到这个ACK包，但是它在接收到USB主机发送来的下一个Setup包时就会默认上一个没有收到的ACK包已经被收到，从而正常结束上一个控制传输，并正常响应下一个控制传输。

所有的USB设备都需要及时处理主机的命令请求，否则就会导致主机产生超时错误。USB协议规定，任何命令的上限执行时间不能超过5s。但是对于某些命令，5s是不适用的，这些命令必须在更短的时间内完成。USB协议虽然规定了命令可以在最多5s内完成，但是USB设备最好不要使用这个最大值，否则会导致命令请求处理过程消耗过多的时间，用户体验会很差。

对于没有数据阶段的标准设备请求，设备必须在收到请求后的50ms内完成请求的状态阶段；对于需要USB设备向USB主机发送数据的标准设备请求，设备必须在收到请求后的500ms内把第一个数据包发送给主机，如果接下来还有数据包，设备必须在前一个数据包成功传输完成后的500ms内发出数据包。在发出最后一个数据包后，设备必须在50ms内完成状态阶段；对于需要USB主机向USB设备发送数据的标准设备请求，设备必须在最多5s内完成数据阶段和状态阶段。

## 2. 中断传输

中断传输一般用于传输数据量小、具有周期性且要求响应速度较快的数据，如键盘、鼠标或者游戏手柄等。图 2-20 是一次成功的中断传输，用于USB主机向USB设备发送数据。其只有一个OUT事务，该事务中数据包的数据位域包含了USB主机所发送的8个字节的数据12 34 56 78 90 AB CD EF。需要注意的是，因为该例中只传输了8个字节的数据，没有超过端点支持的最大包长度，因此一个中断传输（Transfer）中只要一个事务即可。如果传输的数据长度大于端点支持的最大包长度，则一个中断传输内需要多个事务。在传输数据时，如

果最后一个事务的数据长度小于端点支持最大包长度，则认为数据传输完。

### 3. 批量传输

批量传输用于传输数据量大、非周期性、对实时性没有要求的数据，一旦有多余的USB总线带宽，批量传输会立即执行，但当带宽比较紧张时，批量传输会把带宽让给其他类型的传输。图2-21是一次成功的MSD类设备的批量传输，用于USB设备向USB主机发送512字节的数据。这个批量传输包含了8个IN事务，每个IN事务传输了64字节的数据。

Transfer	F	Interrupt	ADDR	ENDP	Bytes Transferred
12	S	OUT	4	2	8
* Transaction					
25242	F	OUT	ADDR	ENDP	T' Data ACK
	S	0x87	4	2	0 8 bytes 0x4B
	O	Packet	H	F	Sync OUT ADDR ENDP CRC5 EOP
	D	75805	S		00000001 0x87 4 2 0x00 250.000 ns
	O	Packet	H	F	Sync DATA0 T' Data CRC16 EOP
	D	75806	S		00000001 0xC3 12 34 56 78 90 AB CD EF 0xF544 266.660 ns
	O	Packet	H	F	Sync ACK EOP
	D	75807	S		00000001 0x4B 266.660 ns

图2-20 一次成功的中断传输

Transfer	F	Bulk	ADDR	ENDP	Mass	Bytes Transferred
101	S	IN	11	1	Storage	512
* Transaction						
4928	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	0 64 bytes	0x4B
4931	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	1 64 bytes	0x4B
4933	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	0 64 bytes	0x4B
4936	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	1 64 bytes	0x4B
4938	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	0 64 bytes	0x4B
4941	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	1 64 bytes	0x4B
4943	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	0 64 bytes	0x4B
4946	F	IN	ADDR	ENDP	T' Data	ACK
	S	0x96	11	1	1 64 bytes	0x4B

图2-21 一次成功的MSD类设备的批量传输

对于CDC类USB设备，其也可以使用批量传输（Transfer）与USB主机之间传输数据。当CDC类USB设备向USB主机发送数据时，USB主机发送IN包，USB设备返回包含数据的数据包，USB主机在成功接收时返回ACK包。对于批量传输的最后一个事务，如果USB主机接收的数据包中的数据长度小于端点支持的最大包长度，那么USB主机将知道USB设备已经将数据发送完；如果USB主机接收的数据包中的数据长度等于端点支持的最大包长度，那么USB主机并不知道USB设备已经将数据发送完。在这个时候，USB设备就需要向USB主机发送一个数据字段长度为0的数据包，来告诉USB主机其已经将数据发送完。

#### 4. 同步传输

同步传输用于传输周期性、低延时性但不需要保证传输质量的数据，如音频或者视频数据。因此，不论是同步传输的IN事务还是OUT事务都没有握手阶段。图2-22是一次成功的同步传输，用于USB设备向USB主机发送数据。其中包含了多个IN事务，每个事务中的数据包的数据位域包含了USB设备发送给USB主机的数据。从事务中可以看到，同步传输并没有握手包。需要注意的是，当一个同步传输中有多个事务时，最后一个事务之前的数据长度都是端点所支持的最大包长度，最后一个事务的数据长度小于或等于端点所支持的最大包长度。如果最后一个事务的数据长度小于端点支持最大包长度，则认为数据传输完。另外，对于全速模式的同步传输，USB设备或者USB主机应可以接收DATA1包或者DATA0包，但只能发送DATA0包。

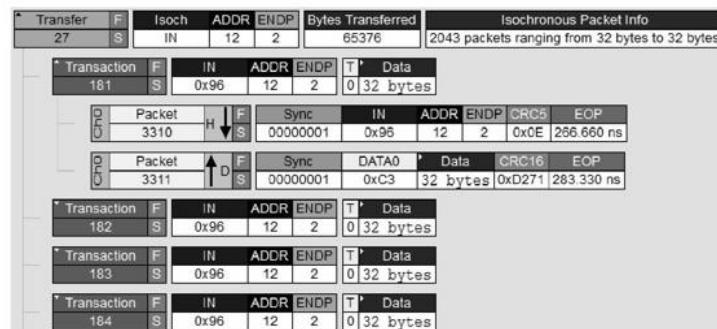


图2-22 一次成功的同步传输

## 2.1.5 数据翻转

USB协议提供了USB主机和USB设备之间的数据同步机制，并通过使用数据翻转（DATA TOGGLE）来实现。USB主机和USB设备之间使用数据翻转（DATA TOGGLE）来实现多个事务的数据传输同步，USB主机和USB设备各自维护一个数据包PID，根据PID来发送数据包。数据接收方根据数据包的接收情况来改变自身的数据包PID，而数据发送方根据数据接收方的反馈（握手包）来改变自身的数据包PID。数据翻转使数据发送方按照一定的PID序列（PID SEQUENCE）发送数据包，如交替使用DATA0/DATA1包。如果USB主机或USB设备没有更改自身PID，则当前数据包会被重传。

需要注意的是，由于同步传输不涉及握手包，因此同步传输不使用数据翻转机制。通信双方并不能保证数据包是否收发成功，当数据接收失败时并不会重传。发送方仅仅按照一定的PID序列发送数据包，接收方根据接收到的数据包PID序列是否正确来进行数据同步。

### 1. 成功情况

图2-23是IN事务中返回ACK的数据翻转，初始状态下，USB主机和USB设备的自身数据包PID都是DATA0。USB主机首先向USB设备发送一个IN包，USB设备向USB主机返回一个DATA0包，USB主机成功收到后将自身的数据包PID从DATA0变成DATA1，然后向USB设备返回一个ACK包。USB设备成功收到ACK包后，将自身的数据包PID从DATA0变成DATA1，此时一次成功的IN事务（事务i）完成。USB主机继续向USB设备获取数据，此时向USB设备发送一个IN包，USB设备返回一个DATA1包，USB主机成功收到后将自身的数据包PID从DATA1变成DATA0，然后向USB设备返回一个ACK包。USB设备成功收到ACK包后，将自身的数据包PID从DATA1变成为DATA0，此时另一次成功的IN事务（事务i+1）完成。如果USB主机还有数据需要从USB设备获取，则重复此过程。

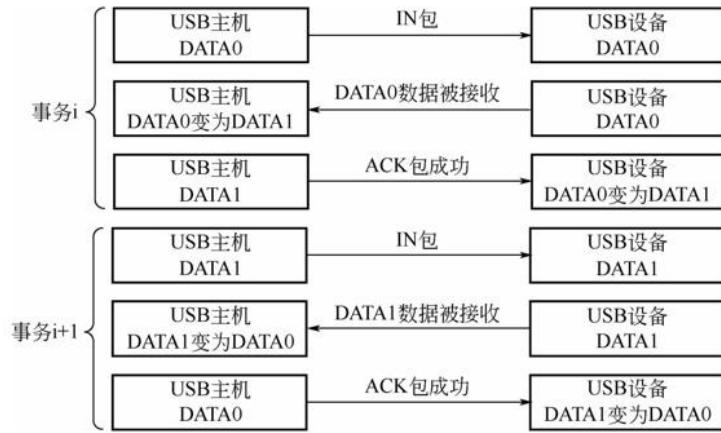


图2-23 IN事务中返回ACK的数据翻转

图2-24是OUT事务中返回ACK的数据翻转，其数据翻转的处理和前文的IN事务中返回ACK的情况类似。

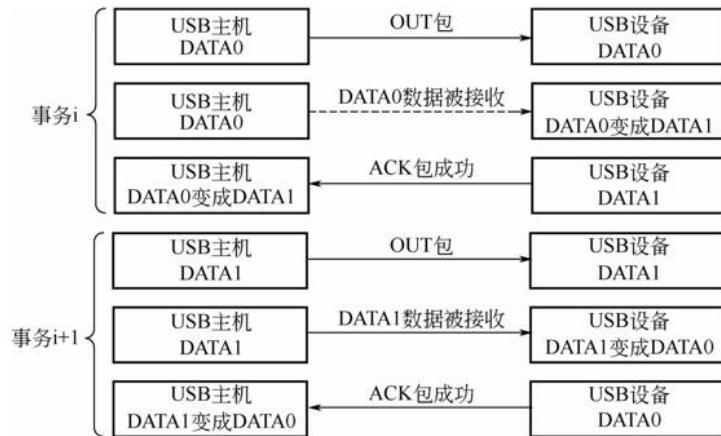


图2-24 OUT事务中返回ACK的数据翻转

## 2. 返回NAK包

图2-25是IN事务中返回NAK的数据翻转，初始状态下，USB主机和USB设备的自身数据包PID都是DATA0。USB主机首先向USB设备发送一个IN包，USB设备向USB主机返回一个DATA0包，但由于某些原因USB主机没有成功收到该数据包，此时保持自身的数据包PID不变，仍为DATA0，然后USB主机向USB设备返回一个NAK包。USB设备成功收到NAK包后，也维持自身的数据包PID不变，仍为DATA0，此时一次失败的IN事务（事务i）完成。USB主机继续向USB设备获取

数据，继续向USB设备发送一个IN包，USB设备返回一个DATA0包，USB主机成功收到后将自身的数据包PID从DATA0变成DATA1，然后向USB设备返回一个ACK包。USB设备成功收到ACK包后，将自身的数据包PID从DATA0变成为DATA1，从而完成该IN事务重传。此时USB主机和USB设备双方的数据包PID保持一致，都为 DATA1，如果USB主机还需从USB设备获取数据，则继续后续的IN事务，且USB设备使用DATA1作为数据包的PID。

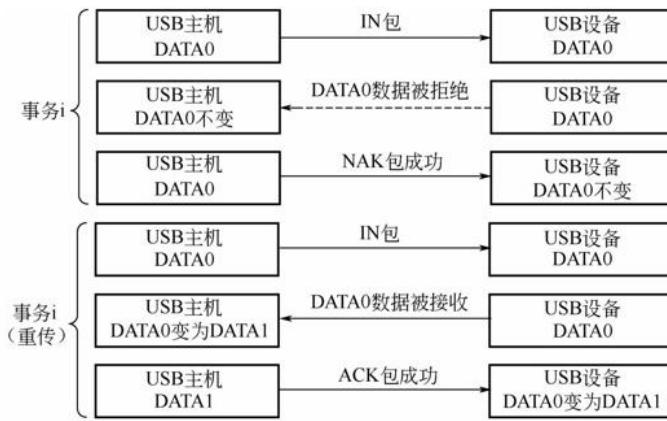


图2-25 IN事务中返回NAK的数据翻转

图2-26是OUT事务中返回NAK的数据翻转，其数据翻转的处理和前文的IN事务中返回NAK的情况类似。

### 3. 返回ACK包失败

图2-27是IN事务中ACK失败的数据翻转，初始状态下，USB主机和USB设备的自身数据包PID都是DATA0。USB主机首先向USB设备发送一个IN包，USB设备向USB主机返回一个DATA0包，USB主机成功收到该数据包后将自身的数据包PID由DATA0变为DATA1，然后向USB设备返回一个ACK包。由于某些原因USB设备没有收到这个ACK包，则保持自身的数据包PID不变，仍为DATA0，此时一次IN事务（事务i）完成。USB主机继续向USB设备获取数据，继续向USB设备发送一个IN包，USB设备返回一个 DATA0 包。USB 主机检查当前自身的数据包 PID ( DATA1 ) 和收到的DATA0包不匹配，则忽略该 DATA0包且不改变自身的数据包PID，同时向USB设备返回一个ACK

包。USB设备成功收到ACK包后将自身的数据包PID从DATA0变成DATA1，此时完成该IN事务重传。

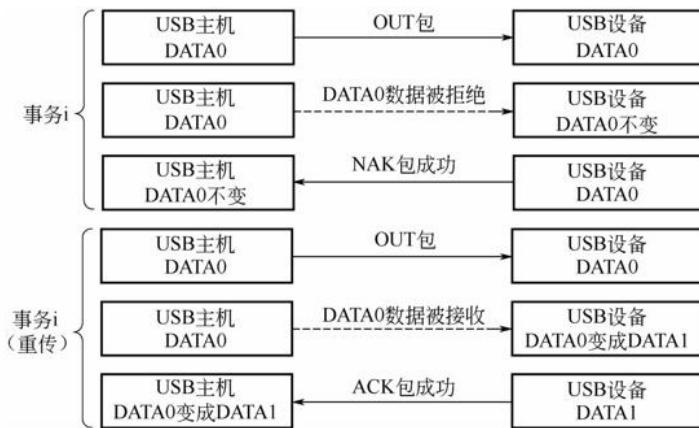


图2-26 OUT事务中返回NAK的数据翻转

此时USB主机和USB设备双方的数据包PID保持一致，如果USB主机还需从USB设备获取数据，则继续后续的IN事务，USB设备使用DATA1作为数据包的PID。

如果在进行数据传输时IN事务只有一个，那么就没有下一个IN事务来进行错误检测。正如2.1.5节提到的，控制传输的状态阶段只有一个IN事务，当这个IN事务中的ACK失败时，USB主机不会继续发送IN包，而是开始发送新的Setup包。

图2-28是OUT事务中ACK失败的数据翻转，其数据翻转的处理和前文的IN事务中ACK失败的情况类似。

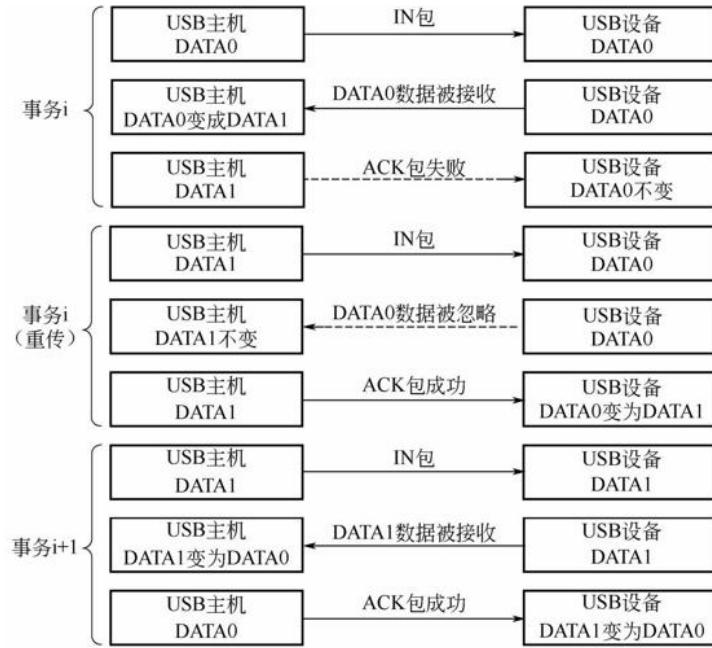


图2-27 IN事务中ACK失败的数据翻转

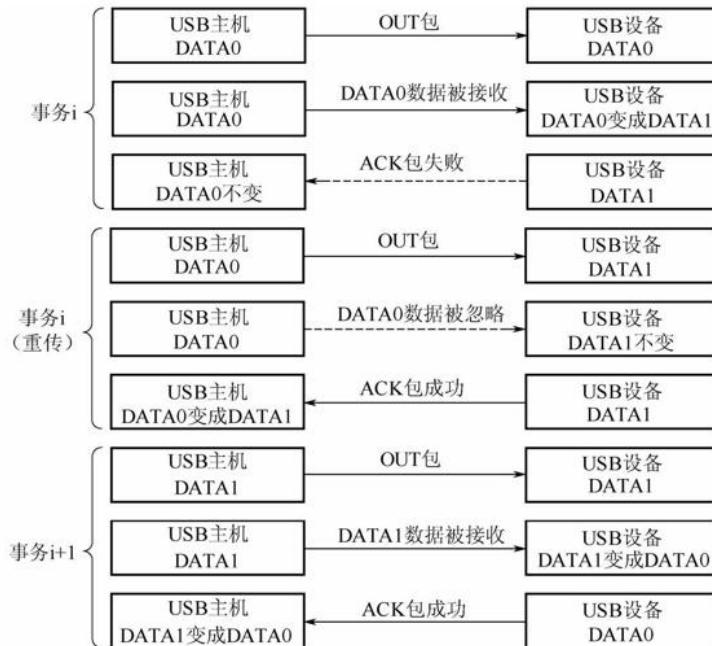


图2-28 OUT事务中ACK失败的数据翻转

#### 4.PID序列

下面介绍各类型传输的PID序列。

## 1 ) 控制传输的PID序列

图2-29是控制传输的PID序列，包括写入数据、读取数据和无数据阶段的控制传输。图中每一行都是一个控制传输，每一个方格都是一个事务。

写入数据时，USB主机在Setup事务中向USB设备发送一个DATA0包，然后在后续OUT事务中向USB设备发送交替的DATA1包和DATA0包，直到数据发送完。最后状态阶段必须使用一个数据长度为0的DATA1包。

读取数据和写入数据的包序列是一样的，无数据阶段的控制传输只有Setup事务中的DATA0包和IN事务中的DATA1包。

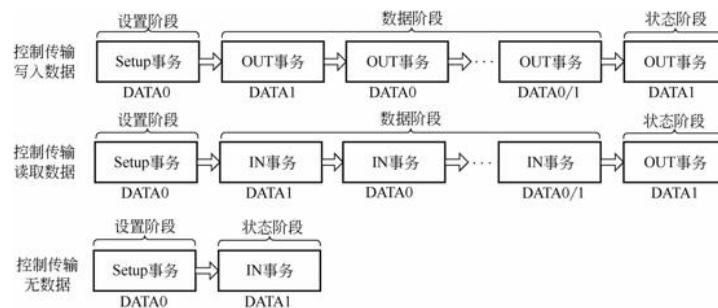


图2-29 控制传输的PID序列

## 2 ) 中断和批量传输的PID序列

图2-30是批量传输的PID序列，读取数据和写入数据的PID序列一致。端点在初始化后，从DATA0开始，每成功执行一个事务，翻转一下（由DATA0变成DATA1或者由DATA1变成DATA0）。数据翻转与传输个数没有直接联系，只由端点在初始化后处理的事务总数决定。端点的数据翻转只有在端点重新初始化或者ClearFeature ( endpoint halt ) 后才会恢复为DATA0。

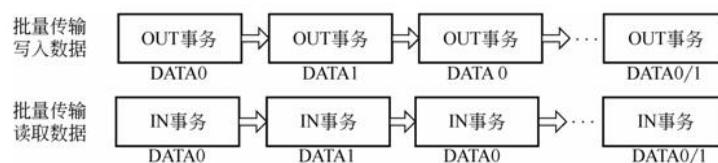


图2-30 批量传输的PID序列

### 3 ) 高速模式下同步传输的PID序列

全速模式下的同步传输数据发送方只使用DATA0包，因此正常情况下接收方只会收到DATA0包。在高速模式下，数据发送方会使用DATA0、DATA1、DATA2、MDATA包，下面重点介绍高速模式下同步传输的PID序列。

图2-31是高速模式下同步传输读取数据时的PID序列。

- 当一个微帧中只有1个事务时，USB设备只使用DATA0包向USB主机发送数据，直到数据读取完。
- 当一个微帧中有2个事务时，USB设备先向USB主机发送一个DATA1包，再发送一个DATA0包，之后将继续在DATA1包和DATA0包之间交替，直到数据读取完。
- 当一个微帧中有3个事务时，USB设备先向USB主机发送一个DATA2包，然后再向USB主机发送一个DATA1包，接着再发送DATA0包，之后将继续按顺序在DATA2、DATA1和DATA0包之间交替，直到数据读取完。

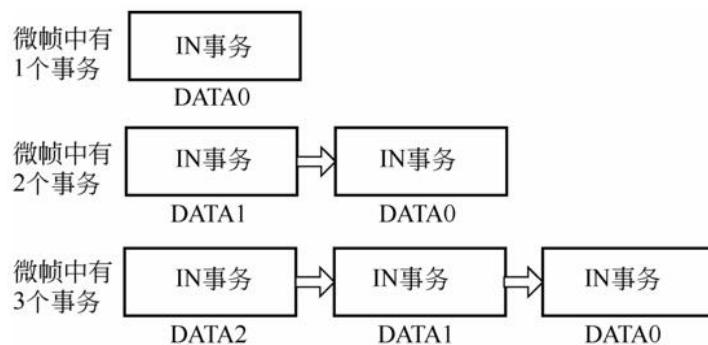


图2-31 高速模式下同步传输读取数据时的PID序列

图2-32是高速模式下同步传输写入数据时的PID序列。

- 当一个微帧中只有1个事务时，USB主机只使用DATA0包向USB设备发送数据，直到数据写入完。
- 当一个微帧中有2个事务时，USB主机先向USB设备发送一个MDATA包，再发送一个DATA1包，之后将继续在MDATA包和DATA1包之间交替，直到数据写入完。
- 当一个微帧中有3个事务时，USB主机先向USB设备发送一个MDATA包，然后再向USB设备发送一个MDATA包，接着再发送DATA1包，之后将继续按顺序在MDATA包、MDATA包和DATA1包之间交替，直到数据写入完。

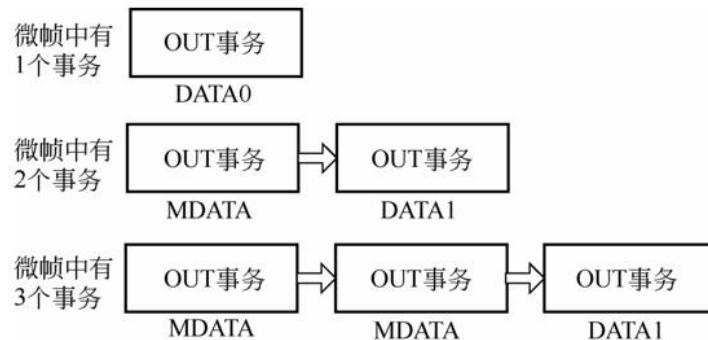


图2-32 高速模式下同步传输写入数据时的PID序列

## 2.1.6 标准USB设备请求

USB 主机通过向 USB 设备发送设备请求来获取 USB 设备的信息和对USB设备进行相关配置。设备请求由长度为8字节的Setup数据指定，方向总是从USB主机向USB设备。表2-4是Setup数据各位域，各位域值的具体含义请参考USB协议规范说明书。

表2-4 Setup数据各位域

偏移量	位 域	长度（字节）	位 域 描 述
0	bmRequestType	1	第 7 位：数据传输方向，0—主机到设备；1—设备到主机 第 5 到 6 位：请求类型，0—标准设备请求；1—类请求；2—制造商 第 0 到 4 位：接收者，0—设备；1—接口；2—端点
1	bRequest	1	特定功能的设备请求
2	wValue	2	根据设备请求的不同而不同
4	wIndex	2	根据设备请求的不同而不同
6	wLength	2	数据阶段要传输的数据长度

其中bRequest位域指明了USB设备请求的类型，包括标准设备请求、设备类请求和制造商自定义请求。标准设备请求包括获取描述符GetDescriptor（）、设置地址Set\_Address（）、设置设备的配置描述符Set\_Congfiguration（）、获取状态Get\_Status（）等。其中获取描述符包括获取USB设备的设备描述符、配置描述符、接口描述符、端点描述符及可选的字符串描述符等。对于获取各个描述符的请求(request)，其bmRequestType位域是一样的(8'b10000000)，确定了数据阶段的数据传输方向是USB设备到USB主机、请求类型是标准设备请求、接收者为USB设备；其bRequest位域确定了该请求是用于获取描述符(0x06)；其wValue位域确定了请求的描述符类型，如0x01代表设备描述符、0x02代表配置描述符、0x03代表字符串描述符等；wIndex位域确定了请求的描述符的语言ID；wLength位域确定了请求的描述符的长度。

表2-5给出了在USB 2.0协议中所有的USB标准设备请求。

表2-5 在USB 2.0协议中所有的USB标准设备请求

bmRequestType	bRequest	wValue	wIndex	wLength	返回数据
8'b00000000 8'b00000001 8'b00000010	CLEAR_FEATURE	特性选择字	0、 接口号、 端点号	0	无
8'b10000000	GET_CONFIGURATION	0	0	1	Configuration 值
8'b10000000	GET_DESCRIPTOR	描述符类型和索引	0或语言 ID	描述符长 度	描述符

续表

bmRequestType	bRequest	wValue	wIndex	wLength	返回数据
8'b10000001	GET_INTERFACE	0	接口号	1	接口的可交换 设置值
8'b10000000 8'b10000001 8'b10000010	GET_STATUS	0	0、 接口号、 端点号	2	设备、 接口、 端点状态
8'b00000000	SET_ADDRESS	USB 设备地址	0	0	无
8'b00000000	SET_CONFIGURATION	Configuration 值	0	0	无
8'b00000000	SET_DESCRIPTOR	描述符类型和索引	0	描述符长 度	描述符
8'b00000000 8'b00000001 8'b00000010	SET_FEATURE	特性选择字	0、 接口号、 端点号	0	无
8'b00000001	SET_INTERFACE	接口的可交换 设置值	接口号	0	无
8'b10000010	SYNCH_FRAME	0	端点号	2	帧号

对于SET\_FEATURE和CLEAR\_FEATURE这两种请求，其特性选择字（Feature Selector）如表2-6所示。

表2-6 特性选择字

特性选择字	接收者	值
ENDPOINT_HALT	端点	0
DEVICE_REMOTE_WAKEUP	USB 设备	1
TEST_MODE	USB 设备	2

## 2.2 USB描述符

描述符相当于设备的名片，描述了该USB设备所有的属性和可配置信息，如设备所属于的类（Class）、接口（interface）信息、端点（endpoint）信息等。可以说，获得了设备的描述符，就知道了该设备的类型和用途、通信的参数等，主机就可以对它进行配置，从而使得通信双方使用相同的参数工作。

标准的USB设备有6种常用的USB描述符：设备描述符、配置描述符、字符串描述符、接口描述符、端点描述符、设备限定描述符。另外，还有一种特殊的描述符称为接口关联描述符，用于将一组有关的描述符关联起来共同描述一个特定的功能。图2-33是USB描述符结构框图。

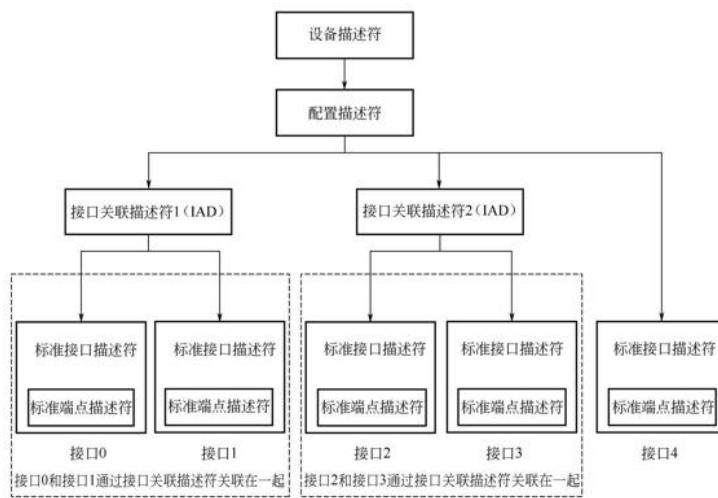


图2-33 USB描述符结构框图

在图2-33中，接口0和接口1通过接口关联描述符1关联在一起共同描述一个功能；接口2和接口3通过接口关联描述符2关联在一起共同描述另外二个功能；而接口4单独描述其他的一个功能。设备描述符指明了该设备有几个配置描述符，每个配置描述符都分别指明了该配置描述符中的接口描述符，而接口描述符指明了该接口有几个端点描述符。当主机需要获取配置描述符时，该配置描述符拥有的接口描述符和端点描述符的所有信息都一并返回。在同一个配置描述符中的多个功能（由一个或者多个接口描述符描述）能够同时工作，但是如果USB设备存在多个配置描述符，USB主机会通过SetConfiguration（）使用其中一个，其他的配置描述符中所描述的功能则不能工作。

## 2.2.1 设备描述符

一个设备有且只有一个设备描述符。设备描述符描述了设备的基本属性，USB设备描述符的位域如表2-7所示。

表2-7 USB设备描述符的位域

偏移量	字段	长度(字节)	字段描述
0	bLength	1	设备描述符长度
1	bDescriptorType	1	描述符类型为设备描述符
2	bcdUSB	2	该设备描述符的USB Spec版本
4	bDeviceClass	1	USB设备的类号，该位域是零代表配置描述符中的接口描述符确定各自的类属性
5	bDeviceSubClass	1	USB设备的子类号，该位域是零代表配置描述符中的接口描述符确定各自的子类属性
6	bDeviceProtocol	1	该设备的协议
7	bMaxPacketSize0	1	端点零的最大包长度（只有8、16、32、64是有效的）
8	idVendor	2	供应商ID
10	idProduct	2	产品ID
12	bcdDevice	2	该USB设备的版本号
14	iManufacturer	1	制造商的字符串描述符索引
15	iProduct	1	产品的字符串描述符索引
16	iSerialNumber	1	设备的序列号字符串索引
17	bNumConfigurations	1	该设备支持的配置数目

在设备描述符中只会给出这个设备所支持的配置描述符（Configuration Descriptor）的数量，设备的配置描述符的索引从1开始，比如当设备的配置描述符有两个时，这两个配置描述符的索引分别是1和2。USB主机就是使用这个索引作为GetDescriptor（Configuration）的参数来分别获取对应的设备的配置描述符的。

## 2.2.2 配置描述符

配置描述符定义了设备的一种配置信息，一个设备可以有一个或者多个配置描述符。USB配置描述符的位域如表2-8所示。

表2-8 USB配置描述符的位域

偏移量	字段	长度(字节)	字段描述
0	bLength	1	配置描述符长度为9字节
1	bDescriptorType	1	描述符类型为配置描述符
2	wTotalLength	2	配置描述符的总长度，包括接口描述符和端点描述符
4	bNumInterfaces	1	该配置支持的接口数目
5	bConfigurationValue	1	主机用 SetConfiguration()选择该配置时配置号
6	iConfiguration	1	描述该配置的字符串索引
7	bmAttributes	1	配置属性
8	bMaxPower	1	USB设备在该配置下所消耗的最大电流值

配置描述符包括了配置的基本信息，如该设备的接口描述符的个数、配置描述符的长度、供电属性等。配置描述符的长度

(wTotalLength) 描述了该配置描述符的总长度，即从该描述符的 bLength 位域开始一直到最后一个端点描述符结尾为止的总长度，包括该描述符本身的长度，以及配置描述符中所有接口描述符和端点描述符的总长度。接口数目 (bNumInterfaces) 表示这个配置有多少个接口描述符，这些接口是控制接口和非控制接口的合集。该配置描述符的配置号信息由 bConfigurationValue 决定，主机就是根据这个值决定在 SetConfiguration ( ) 时给设备选择设置哪个配置描述符，并让设备处于该配置的工作状态。bmAttributes 代表配置的某些属性，第5比特代表该设备是否支持远程唤醒 (Remote Wakeup)，第6比特代表该设备是否支持自供电。bMaxPower 代表设备在该配置下所消耗的最大电流值，以2mA为单位，如当bMaxPower的值是25时，代表设备在该配置下所消耗的最大电流值为50mA。USB设备多个配置描述符中接口数目不尽相同，bMaxPower也有可能不一样。USB 主机会根据设备在某种配置下 bMaxPower 的值来查看其是否有足够的电流可以提供，最终决定该设备的某种配置是否可用，如果某个设备的所有配置因为USB 主机无法提供足够电流而不可用，该设备将无法正常工作。

当主机需要获取配置描述符时，该配置描述符所拥有的接口描述符和端点描述符都一并返回。

## 2.2.3 接口描述符

接口描述符位于配置描述符中，指明了某个特殊的USB类的接口。接口描述符通过端点完成数据传输，实现特定类的特定功能。例如，一个设备既有扬声器又有麦克风的功能，那它就至少有两个音频接口。USB接口描述符的位域如表2-9所示。

表2-9 USB接口描述符的位域

偏移量	字段	长度(字节)	字段描述
0	bLength	1	配置描述符长度为9字节
1	bDescriptorType	1	描述符类型为接口描述符
2	bInterfaceNumber	1	接口描述符号
3	bAlternateSetting	1	该接口的可替换设置号
4	bNumEndpoints	1	除了端点零外，该接口所使用的端点数目
5	bInterfaceClass	1	该接口实现的USB类
6	bInterfaceSubClass	1	该接口实现的USB子类
7	bInterfaceProtocol	1	该接口实现的协议
8	iInterface	1	描述该接口的字符串索引

接口描述符包括该接口的识别号（bInterfaceNumber）、接口的可替换设置号（bAlternateSetting）、端点数目、类和子类、协议等。接口的替换设置号主要在音频和视频类中使用，一般用于描述不同声道、采样率或者帧率、分辨率的组合。接口的可替换设置0不包含传输端点，其他的可替换设置包含传输端点。之所以要有一个不包含任何传输端点的可替换设置0，是由USB音频、视频设备的特性决定的，该设置用于在设备没有被使用时作为设备的默认设置。当USB设备存在多个可替换设置时，USB主机通过SetInterface（）来指定某个接口的可替换设置（Alternate Setting）。如果USB主机没有使用SetInterface（）来指定接口的可替换设置，通信双方默认使用可替换设置0。端点数目（bNumEndpoints）就是该接口描述符所有支持端点的数目，不包括控制端点0。

## 2.2.4 端点描述符

端点描述符包括该端点地址、端点属性、支持的最大包长度、传输时间间隔等。在主机获取配置描述符时，端点描述符和接口描述符一起返回。USB端点描述符的位域如表2-10所示。

表2-10 USB端点描述符的位域

偏移量	字 段	长 度（字节）	字 段 描 述
0	bLength	1	端点描述符的长度
1	bDescriptorType	1	描述符类型为端点描述符
2	bEndpointAddress	1	端点地址
3	bmAttributes	1	端点的属性
4	wMaxPacketSize	2	端点的最大传输包长度
6	bInterval	1	端点的数据传输间隔

端点描述符包括该端点地址、端点属性、支持的最大包长度、传输时间间隔等。端点地址 ( bEndpointAddress ) 的第 0 ~ 3 比特代表该端点所使用的地址，第7比特代表该端点是用作输入 ( IN ) 还是输出 ( OUT ) 端点。端点属性 ( bmAttributes ) 的第0和第1比特代表该端点的支持的四种传输属性。如果该端点属于ISO端点，属性的第2 ~ 3比特代表该ISO端点的数据同步属于同步、异步、自适应还是无同步类型，属性的第4 ~ 5比特代表该ISO端点属于数据传输端点还是反馈端点，对于ISO端点的具体介绍参见第5章。传输时间间隔代表该端点的数据传输间隔了几个帧/微帧，计算方法为  $2^{b\text{Interval}-1}$  个帧/微帧。

## 2.2.5 字符串描述符

字符串描述符是可选的。如果一个设备不支持字符串描述符，需要将设备描述符、配置描述符、接口描述符中的字符串索引值设成零，字符串描述符用UNICODE编码。

主机获得设备的某个字符串描述符分两条命令：首先主机发送USB标准命令GetDescriptor（），其中所使用的字符串的索引值为0，设备返回一个零字符串描述符的位域，如表2-11所示。

表2-11 零字符串描述符的位域

偏移量	字段	长度（字节）	字段描述
0	bLength	1	描述符的长度
1	bDescriptorType	1	描述符类型为字符串描述符
2	wLANGID[0]	2	字符串的语言 ID 0
...	...	...	...
N	wLANGID[x]	2	字符串的语言 IDx

wLANGID[0] ~ [x]代表该设备支持的语言，可以从USB设备语言ID规范中获得具体值，典型值0x0409代表英语。

主机根据自己是否支持该语言，再次发出USB标准命令GetDescriptor（），指明所要求得到的字符串的索引值和语言。这次设备所返回的是UNICODE编码的字符串描述符，UNICODE字符串描述符的位域如表2-12所示。

表2-12 UNICODE字符串描述符的位域

偏移量	字段	长度（字节）	字段描述
0	bLength	1	描述符的长度
1	bDescriptorType	1	描述符类型为字符串描述符
2	bString	N	UNICODE 字符串

## 2.2.6 接口关联描述符

对于复合USB设备的接口描述符，可以在每个类（Class）要合并的接口描述符之前加一个接口关联描述符（Interface Association Descriptor，IAD），其作用就是把多个接口定义成一个类设备。

接口关联描述符的位域如表 2-13 所示，bFirstInterface 代表起始的接口编号，bInterfaceCount 代表属于这个 IAD 的接口数目，编号中间不能有间隔。在一个类的所有合并接口都结束之后，第二个类的所有需要合并的接口又以 IAD 开始。如图 2-33 所示，IAD1 的 bFirstInterface 为 0，bInterfaceCount 为 2；IAD2 的 bFirstInterface 为 2，bInterfaceCount 为 2。对于音频类的 IAD，音频流接口号必须以一个连续的顺序放在对应的音频控制接口号后面。

表2-13 接口关联描述符的位域

偏移量	字 段	长 度 (字节)	字 段 描 述
0	bLength	1	接口关联描述符的长度
1	bDescriptorType	1	描述符类型为接口关联描述符
2	bFirstInterface	1	该接口关联描述符所关联的第一个接口号
3	bInterfaceCount	1	接口关联描述符所拥有的连续接口数
4	bFunctionClass	1	接口关联描述符的功能所实现的 USB 类
5	bFunctionSubClass	1	接口关联描述符的功能所实现的 USB 子类
6	bFunctionProtocol	1	接口关联描述符的功能所实现的 USB 协议
7	iFunction	1	功能的字符串索引

## 2.2.7 设备限定描述符

设备限定描述符（Device Qualifier Descriptor）用于描述一个能够同时支持高速和全速模式的USB设备工作在另外一个模式时的设备信息。例如，当设备工作于全速模式时，设备限定描述符返回它工作于高速模式下的信息。反之，如果设备工作于高速模式时，设备限定描述符返回它工作于全速模式下的信息。如果一个设备能够同时支持高速和全速模式，并且其在高速模式下和全速模式下信息有所不同，则它必须支持设备限定描述符。

主机同样发出USB标准命令GetDescriptor（），指明所需要获得的设备限定描述符。如果一个只支持全速模式的设备收到一个获取设备限定描述符的命令，需要告诉主机这是个错误请求。设备限定描述符的位域如表2-14所示。

表2-14 设备限定描述符的位域

偏移量	字段	长度（字节）	字段描述
0	bLength	1	接口关联描述符的长度
1	bDescriptorType	1	描述符类型为设备限定描述符
2	bcdUSB	2	USB 规范版本号
4	bDeviceClass	1	设备限定描述符所实现的 USB 类
5	bDeviceSubClass	1	设备限定描述符所实现的 USB 子类
6	bDeviceProtocol	1	设备限定描述符所实现的 USB 协议
7	bMaxPacketSize0	1	对于其他速度最大包长度
8	bNumConfigurations	1	其他速度的配置数目
9	bReserved	1	保留

## 2.2.8 其他速度模式下的配置描述符

其他速度模式下的配置描述符（Other\_Speed\_Configuration Descriptor）与普通描述符是完全相同的，一般与设备限定描述符一起使用来描述在其他速度模式下设备的配置信息，在此就不再赘述。

## 2.3 枚举

当USB设备的速度类型确定之后，USB通信双方将会工作在相同的速度模式下，随后USB枚举才会开始。USB枚举的本质就是USB主机获取USB设备的参数信息并且对于可配置参数进行配置的过程。当枚举结束时，USB设备将会使用在枚举过程中USB主机所配置的参数进行工作，从而确保通信双方使用相同的参数。同时，需要指出的是，可配置的参数可以在后续的通信过程中进行修改。

## 2.3.1 设备状态

枚举完成之前，USB设备要经过一系列的状态变化，才能最终完成枚举。这些状态是连接状态（attached）、供电状态（powered）、默认状态（default）、地址状态（address）、配置状态（configured）、挂起状态（suspended）。当设备状态变为配置状态时，即可认为USB设备和USB主机间的枚举完成。图2-34是设备端的状态变化。

### 1.连接状态

正如1.2节所述，USB设备需要和主机先建立物理上的连接。如果USB设备连接到主机，就处于连接状态。

### 2.供电状态

USB设备可以从USB的 $V_{BUS}$ 上获取电源，或者通过外部电源获取电源。通过外部电源供电的设备称为自供电（Self-powered）设备，通过 $V_{BUS}$ 供电的设备称为总线供电（Bus-powered）设备。对于自供电设备，在连接到主机之前，设备已经通电，但此时设备并不是USB协议定义的供电状态，只有 $V_{BUS}$ 有电后，才进入供电状态。

当USB设备不是直接与USB主机相连，而是通过集线器间接连接到USB主机时，只有对应的集线器端口被主机配置使能之后，集线器端口才会向USB设备的 $V_{BUS}$ 线供电。当集线器端口复位时，端口在被重新配置之前都不会向 $V_{BUS}$ 线供电，相应的USB设备的状态也将变为连接状态。

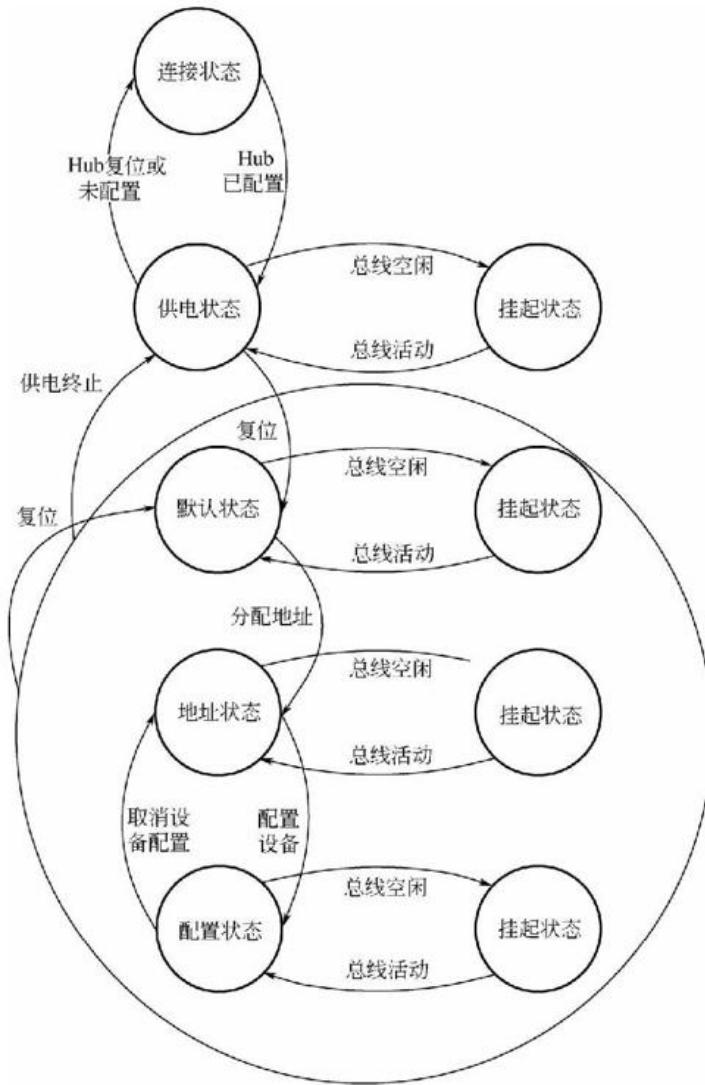


图2-34 设备端的状态变化

### 3.默认状态

USB设备进入供电状态后，在被复位之前，不能响应总线上的任何事务（Transaction）。只有当USB设备被复位，处于默认状态后，才会响应主机发送过来的请求。

### 4.地址状态

在USB设备被复位后，且在USB主机给USB设备设置一个新的地址之前，所有的USB设备使用默认的0地址与主机通信。USB设备收到

主机发送的设置地址 SetAddress( ) 请求后，USB 设备就会得到一个唯一的地址。USB 设备会保存并使用该地址与主机通信，直到设备被复位或者断开。即使 USB 设备在获取唯一的地址之后进入挂起状态，设备依然会保留这个地址，并在总线恢复后继续使用该地址。在设备挂起期间，主机也不能把该地址分配给其他的 USB 设备。

指定了新的地址的 USB 设备在收到 USB 主机发出的复位信号后会进入默认状态，之前所指定的地址也将无效，USB 设备需要 USB 主机重新分配地址。

## 5. 配置状态

在 USB 设备的功能能够使用之前，USB 设备和 USB 主机必须协商确定功能相关的配置项，所有的配置项以描述符的形式提供（详见 2.2 节）。USB 主机通过获取描述符 GetDescriptor( ) 请求获得 USB 设备相关的描述符，通过设置接口 SetInterface( ) 和设置配置 SetConfiguration( ) 来设置设备的相关配置项。一旦相关功能参数被配置后，USB 设备就能正常工作了。

正常工作的 USB 设备在收到 USB 主机发出的复位信号后会进入默认状态，之前所指定的地址、所配置的接口（Interface）和配置项（Configuration）都将无效，需要由 USB 主机重新指定设备地址，获取描述符并配置相关配置项，才能使设备再次工作。

## 6. 挂起状态

挂起/恢复是 USB 协议实现低功耗的一种机制，本书将在 2.4 节详细介绍挂起与恢复的相关机制。总体上说，除了设备的连接状态，在其他状态下，当 USB 总线持续 3ms 没有活动时，设备就会自动进入挂起状态。进入挂起状态后，USB 设备要维持所有的内部状态，如软件的状态机，以及设备的地址和配置项等。

## 2.3.2 枚举流程

USB设备状态从初始的连接状态到最终的配置状态的变化过程就是整个枚举流程。在USB 2.0协议中，USB设备的状态变迁都是由USB主机发起的请求（Request）所触发，而这些请求实际上就是一个个的控制传输（Control Transfer）。不同主机实现的枚举在细节上会略有不同，但大体流程是一样的。图2-35是枚举的数据包图，是一台运行Windows 7系统的PC在枚举一个全速设备时，通过协议分析仪抓取的。本节以此为例详细介绍USB设备的枚举流程，特别对不同主机上枚举流程的细微差异进行深入分析，让读者对USB协议有更深入的理解。

Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
0	GET	0	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
1	Packet	49		Reset			
2							64.876 ms
3	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex wLength
4	SET	0	0	SET_ADDRESS	New address 9	0x0000	9
5	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
6	GET	9	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
7	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
8	GET	9	0	GET_DESCRIPTOR	CONFIGURATION type, index 0	0x0000	4 Descriptors
9	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
10	GET	9	0	GET_DESCRIPTOR	STRING type, LANGID codes requested	Language ID 0x0000	Lang Supported
11	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
12	GET	9	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	HID MOUSE DEVICE
13	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex NotEnough Data Descriptors
14	GET	9	0	GET_DESCRIPTOR	DEVICE_QUALIFIER type	0x0000	0 bytes DEVICE_QUALIFIER Descriptor
15	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
16	GET	9	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
17	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
18	GET	9	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	CONFIGURATION Descriptor
19	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex Descriptors
20	GET	9	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	4 Descriptors
21	Transfer	Control	ADDR	ENDP	bRequest	wValue	wIndex wLength
22	SET	9	9	SET_CONFIGURATION	New Configuration 1	0x0000	0

图2-35 枚举的数据包图

### 1. 获取设备描述符

枚举的第一步是获取设备描述符，主机通过GetDescriptor（Device type）请求来获取设备的设备描述符。对于高速设备和低速设备，用于控制传输的端点0支持的最大包长度是确定的，分别是64字节和8字节（见表2-3）。但是对于全速设备，端点0支持的最大包长度可能是8字节、16字节、32字节或者64字节，在USB主机获取到USB设备的设备描述符之前，USB主机不能确定USB设备端点0的最大包长度，这样USB主机获取USB设备描述符的请求就有可能不能正常完成。

下面首先介绍当USB主机不能确定USB设备端点0的最大包长度时，USB主机与USB设备通信可能存在的问题，然后介绍对应的解决方案。

在USB主机和USB设备通信前，USB主机必须要指定端点0上的最大包长度，以便正确检测控制传输中数据阶段的结束（详见 2.1.4 节）。假设USB主机端指定USB端点0的最大包长度为64字节，而实际USB设备端点0的最大包长度为8字节。由于USB的设备描述符长度为固定的18字节，USB主机发送的第一个GetDescriptor（Device type）请求将会要求USB设备端发送18字节的数据。USB设备收到该请求后，准备发送18字节的数据，由于其最大包长度为8字节，所以18字节的数据将会分成3个事务完成，分别发送8字节、8字节和2字节。但是由于USB主机端指定的最大包长度为64字节，当USB主机接收到第一个8字节的事务时，就认为数据阶段已经完成（因为8字节的包长度小于USB主机端指定的最大包长度），而USB设备端此时还在准备发送第二个8字节的事务，导致这个控制传输由于通信双方的状态不同步而无法正常完成。

在不同的主机系统上，使用了不同的解决方案。下面以Windows 7操作系统和Linux Ubuntu系统为例介绍其解决方案。

在Windows 7操作系统上，其获取设备描述符的方式如图2-36所示。

Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
0	S	GET	0	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
Transaction	F	SETUP	ADDR	ENDP	T D Tp R	bRequest	wValue	wIndex
0	S	0xB4	0	0	0 D->H S D	0x08	0x0100	0x0000
Transaction	F	IN	ADDR	ENDP	T * Data	ACK		0x4B
2	S	0x96	0	0	1 8 bytes			
Transaction	F	OUT	ADDR	ENDP	T * Data	ACK		
3	S	0x87	0	0	1 0 bytes			0x4B
Packet		49	H	Reset				
CnC 54.876 ms								
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
1	S	SET	0	0	SET ADDRESS	New address 9	0x0000	0
Transaction	F	SETUP	ADDR	ENDP	T D Tp R	bRequest	wValue	wIndex
2	S	0xB4	9	0	0 D->H S D	0x08	0x0100	0x0000
Transaction	F	IN	ADDR	ENDP	T * Data	ACK		0x4B
8	S	0x96	9	0	1 8 bytes			
Transaction	F	IN	ADDR	ENDP	T * Data	ACK		0x4B
10	S	0x96	9	0	0 8 bytes			
Transaction	F	IN	ADDR	ENDP	T * Data	ACK		0x4B
12	S	0x96	9	0	0 8 bytes			
Transaction	F	IN	ADDR	ENDP	T * Data	ACK		0x4B
14	S	0x96	9	0	1 2 bytes			
Transaction	F	OUT	ADDR	ENDP	T * Data	ACK		0x4B
15	S	0x87	9	0	1 0 bytes			

图2-36 Windows 7获取设备描述符的方式

主机要求设备返回64字节的设备描述符并默认USB设备端点0的最大包长度为64字节。如果设备端点0的最大包长度不小于标准设备描述符的长度（18字节），设备就可以一次性将18字节的设备描述符发送给主机，通信双方状态保持同步；如果设备的端点0最大包长度小于设备描述符的长度，那么此时设备要使用多个事务向主机发送一个完整的设备描述符，前述的不同步问题会出现。对此，Windows 7系统在第一次获取设备描述符后，直接进行复位（Reset），将设备状态重置为初始状态。而由于设备端所支持的最小的最大包长度为8，此时主机一定能获取到设备描述符的前8个字节；而设备描述符的第8个字节（详见2.2.1节）就是设备端点0所支持的最大包长度。主机可以在后续的控制传输中使用这个设备端的最大包长度来保证双方正常通信。图2-36中的“Reset”就是这个解决方案的关键点。

在Ubuntu上，其获取设备描述符的方式如图2-37示。

Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
					GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
* Transaction F SETUP ADDR ENDP T D Tp R bRequest wValue wIndex wLength ACK								
3	S	0xB4	12	0	0 D->H S D	0x06	0x0100	0x0000 8 0x4B
* Transaction F IN ADDR ENDP T Data ACK								0x4B
5	S	0x96	12	0	1 8 bytes			
* Transaction F OUT ADDR ENDP T Data ACK								0x4B
6	S	0x87	12	0	1 10 bytes			
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
					GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
7	S	0xB4	12	0	0 D->H S D	0x06	0x0100	0x0000 18 0x4B
* Transaction F IN ADDR ENDP T Data ACK								0x4B
9	S	0x96	12	0	1 18 bytes			
* Transaction F OUT ADDR ENDP T Data ACK								0x4B
10	S	0x87	12	0	1 10 bytes			

图2-37 Ubuntu获取设备描述符的方式

主机第一次获取设备描述符时只获取其前8字节。此时，不管USB设备端点的最大包长度是多少，USB设备只能发送8字节的数据，同时通信双方同步结束数据阶段。之后USB主机读取USB设备描述符的第8个字节，得到设备端点0的最大包长度，并用于后续的控制传输。这个解决方案的关键点就是第一次获取设备描述符时只获取其前8个字节。

这两种解决方案都能解决实际的问题，但是解决问题的思路有所不同，Windows 7的解决方案是一种异常处理的方式，而Linux Ubuntu的解决方案则体现了一种精细控制、避免出错的思想。读者可以根据自己的实际情况选择相应的解决方案。

## 2.设置设备的地址

如图2-35所示，在Transfer 1中主机发送SetAddress（）请求来设置设备的地址，在本例中，SetAddress（）请求的Setup数据内包含主机为设备分配的新地址。设备收到地址后，进入地址状态，在该设备从主机上被断开或者复位之前，设备要一直使用该地址与主机通信。在本次传输中，USB主机和USB设备都仍然使用地址0进行通信。

设备收到SetAddress（）请求后，必须要在50ms内处理该请求并完成该请求的状态阶段。在完成状态阶段后，可以有2ms的恢复时间。在该恢复时间内，USB主机不能使用其刚刚指定的地址与USB设备通信。在2ms后，设备必须使用该新地址与USB主机进行通信。同

时，设备不能对发送到旧地址的任何请求产生任何回复，除非新旧地址是一样的。

### 3. 获取完整设备描述符

当设备的新地址设置完成后，USB主机将会使用新的地址与USB设备进行通信。如图2-35所示，在Transfer 2中，USB主机使用新地址9与USB设备进行通信，以获取完整的18个字节的USB设备描述符。

USB设备描述符的第18个字节表示该设备配置描述符的数目，当USB主机获取到完整的设备描述符后，USB主机就可以获取USB设备的配置描述符了。

### 4. 获取配置描述符

由于每个设备配置描述符的长度都不一样，USB主机无法提前得知USB设备的配置描述符长度。USB设备的配置描述符的第3个和第4个字节表示USB设备的配置描述符长度，而更重要的是，USB设备的配置描述符的第9个字节代表该配置下USB设备的最大消耗电流值（详见2.2.2节）。所以USB主机会先获取配置描述符的前9个字节，以获取该配置描述符的总长度和最大消耗电流值，如果该配置下的最大消耗电流值超过USB主机可提供的额度，该配置将不可用。如果设备中所有的配置都不可用，该设备将不可用，枚举的流程将中止。

在不同的主机系统上，获取配置描述符上采取不同的策略。  
Windows 7操作系统获取配置描述符的过程如图2-38所示。

运行在 Windows 7 上的主机在第一次获取配置描述符时，在 Transfer 3 中要求设备发出最多255字节的配置描述符。之后，在 Transfer 8 中主机再次发出请求，获取设备配置描述符的前 9 个字节。获取到配置描述符的总长度后，USB 主机在 Transfer 9 中使用该长度作为参数发出 GetDescriptor ( Configuration ) 请求，要求USB设备提供完整的配置描述符。获取到设备的配置描述符后，主机通过解析配置描述符就可以知道该设备的具体功能。

Transfer	#	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
3	S	GET	9	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	4 Descriptors
					T	D	R	
					0 D->H	S	D	0x06 0x0200 0x0000 255 0x4B
*Transaction	F	SETUP	ADDR	ENDP	T	D	R	
16	S	0xB4	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
18	S	0x96	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
20	S	0x96	9	0	018 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
22	S	0x96	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
24	S	0x96	9	0	018 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
26	S	0x96	9	0	112 bytes			ACK
								0x4B
*Transaction	F	OUT	ADDR	ENDP	T	D	R	
27	S	0x87	9	0	10 bytes			ACK
								0x4B
Transfer	#	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
8	S	GET	9	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	CONFIGURATION Descriptor
					T	D	R	
					0 D->H	S	D	0x06 0x0200 0x0000 9 0x4B
*Transaction	F	SETUP	ADDR	ENDP	T	D	R	
55	S	0xB4	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
57	S	0x96	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
59	S	0x96	9	0	01 byte			ACK
								0x4B
*Transaction	F	OUT	ADDR	ENDP	T	D	R	
60	S	0x87	9	0	10 bytes			ACK
								0x4B
Transfer	#	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
9	S	GET	9	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	4 Descriptors
					T	D	R	
					0 D->H	S	D	0x06 0x0200 0x0000 34 0x4B
*Transaction	F	SETUP	ADDR	ENDP	T	D	R	
61	S	0xB4	9	0	0 D->H	S	D	0x06 0x0200 0x0000 34 0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
63	S	0x96	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
65	S	0x96	9	0	08 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
67	S	0x96	9	0	18 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
69	S	0x96	9	0	08 bytes			ACK
								0x4B
*Transaction	F	IN	ADDR	ENDP	T	D	R	
71	S	0x96	9	0	12 bytes			ACK
								0x4B

图2-38 Windows 7操作系统获取配置描述符的过程

与Windows 7相比，运行在Ubuntu上的主机获取设备描述符的方式的差别在于Ubuntu并不会要求设备发送最多255字节的配置描述符，其只会先获取配置描述符的前9字节，然后再获取完整的配置描述符。因其过程与Windows 7主机基本一致，在此就不再赘述。

## 5. 获取字符串描述

当USB主机遍历USB设备所有的配置描述符，确定有可用的配置后，主机通过GetDescriptor ( String type ) 标准请求获取该设备的一些字符串格式的信息，如设备制造商、产品信息和序列号等。这些字符串格式的信息是用字符串描述符的形式提供的，而这些信息所对应的字符串描述符的索引是在USB设备的设备描述符中提供（详见2.2.1节）。

如果2-35所示，USB主机首先在Transfer 4中获取了字符串所使用的语言，本例中是用的语言是英语，其语言ID为0xxxx；之后在Transfer 5中获取了设备的xxx信息。

## 6. 获取限定描述符

如图2-35所示，在Transfer 6中USB主机也会尝试去获取USB设备的限定描述符，以获取USB设备在其他速度模式下的信息。

USB设备不一定会支持限定描述符，在这种情况下，USB设备会回复STALL表示设备不支持该描述符。这种控制传输的失败并不影响设备的正常枚举和使用。USB主机会再次获取USB设备的设备描述符（Transfer 7）、配置描述符（Transfer 8）并进行后续的流程。

## 7. 配置设备的配置描述符

USB主机获取USB设备的某个配置描述符后，会解析该配置描述符中所有的接口信息和端点信息，之后主机就可以发出SetConfiguration（）请求来通知USB设备使用哪一个配置。即使USB设备只有一个配置描述符，这个SetConfiguration（）请求也不能被省略。该命令中包含一个配置值，这个配置值是主机从设备的配置描述符中解析得来的。

USB设备收到SetConfiguration（）请求后，会判断配置值是否合法。如果合法，设备就进入配置状态。设备会根据配置值选择对应的配置描述符，进行初始化操作，如初始化该配置中所有接口中所有的端点。初始化完成后，设备就具备与主机进行数据传输的能力，此时枚举阶段结束。

如图2-35所示，在Transfer 10中，USB主机使用SetConfiguration（）配置成功后，整个枚举流程结束。后续的USB通信将主要发生在配置描述符的接口/端点描述符所描述的端点上。

## 2.4 挂起和恢复

为了降低 USB 设备的整体功耗，使 USB 设备有更长的续航时间，在USB 2.0 规格说明书中定义了两种电源管理事件：挂起和恢复。挂起会导致USB设备进入低功耗模式并且相应的功能不可用；恢复则相反，会使USB设备退出低功耗模式同时恢复相应的功能。

## 2.4.1 挂起

从时间角度来说，一个USB设备的挂起分为两个阶段：挂起事件的产生和进入挂起状态。

当检测到USB总线空闲状态持续超过3ms时，设备的USB控制器就会产生挂起事件。设备必须在收到事件后的7ms内进入实际的挂起状态，即低功耗模式。挂起状态的主要表象是设备从 USB 的  $V_{BUS}$  上汲取的电流不超过 $500\mu A$ 的电流。一个例外是，如果高功耗USB设备的远程唤醒功能被使能了（如果该设备支持远程唤醒功能），那么该设备在总线挂起后从 USB 的  $V_{BUS}$  上汲取的电流不能超过 $2.5mA$ 。

图2-39是低速模式下USB总线的挂起状态（通道3是D-信号，通道2是D+信号）。

在进入挂起状态后，USB设备必须保持D+或D-的电阻上拉状态，以保证USB总线处于空闲J状态，从而使得USB主机或Hub能够检测到设备仍处于连接状态。

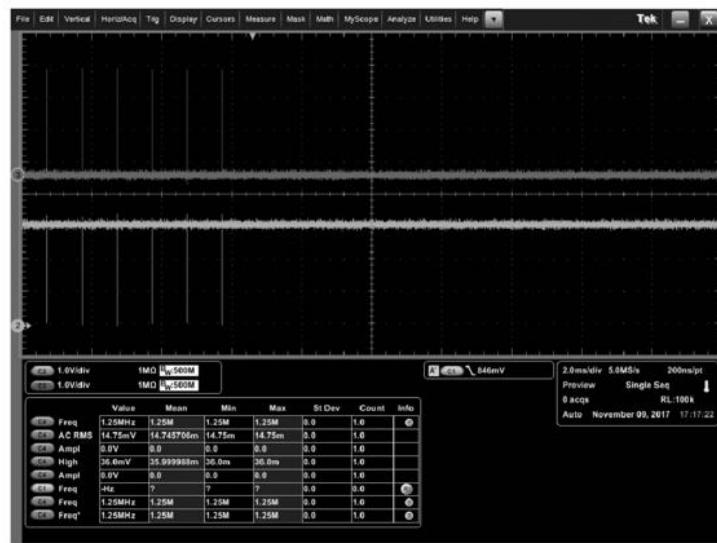


图2-39 低速模式下USB总线的挂起状态

当挂起总线时，USB主机首先根据USB设备是否支持远程唤醒且系统是否需要开启USB设备的远程唤醒功能来决定是否开启设备的远程唤醒功能。其次，USB主机通过停止总线的所有活动使得总线空闲至少3ms。从总线挂起开始到总线恢复过程中，主机必须一直保持 $V_{BUS}$ 状态，以确保设备不会断开连接。

这里有一点是需要解释的，对于高速模式的设备在进入挂起状态后，它将切换到全速模式（挂起后的空闲状态和全速模式一样）。

挂起分为两种情况：全局总线挂起和部分总线挂起。

## 1.全局总线挂起

当没有任何事务需要处理时，USB主机为了降低整个系统的功耗会停止总线的所有传输。USB总线的每个设备在收到挂起事件后，各自进入挂起状态。

USB主机通过根端口实现全局总线挂起，所以在有无Hub的情况下，主机软件对于全局总线挂起的操作流程基本类似。

## 2.部分总线挂起

当主机系统需要有选择性地让总线上的某些设备进入挂起状态时，主机通过向设备所对应的Hub发送SetPortFeature (PORT\_SUSPEND) 命令。Hub收到命令后将对应下行端口和设备之间的总线挂起（停止转发任何数据，使下行端口的总线一直处于空闲状态），该设备在收到挂起事件后进入挂起状态。部分总线挂起主要针对有Hub的情况，主机通过控制设备连接Hub的端口实现对设备的挂起。

在实际开发过程中，经常有开发者在挂起总线时将 $V_{BUS}$ 禁止导致USB设备不是进入挂起而是进入断开状态。

是否进入挂起状态完全由USB主机决定，USB设备无法自行进入挂起状态，目前也没有任何已定义的命令或者方式能够让USB设备主

动通知 USB 主机使其能够让该设备进入挂起状态。

## 2.4.2 恢复

在挂起后，总线上任何的非空闲状态都可以唤醒总线，如被动的SE0信号（如移除USB设备等）、复位信号和恢复信号等。本节主要介绍使用恢复信号的方式唤醒总线。

恢复信号主要是指由主机或者设备发起的用于将总线从挂起状态唤醒。恢复信号时序由两部分组成：不小于20ms的全速/低速的K状态（20ms的K信号主要是用于确保总线上的所有设备都能够被唤醒）和结束标志。在不同速度模式下，结束标志是不同的。当总线原本（挂起前）处于全速/低速模式时，结束标志是两位的低速SE0状态及一位的当前速度的J状态（两位的SE0状态持续时间在1.25μs到1.5μs之间），如图2-40所示。而如果总线挂起前处于高速模式，结束标志是高速的空闲状态SE0，如图2-41所示。



图2-40 全速/低速模式下的恢复信号



图2-41 高速模式下的恢复信号

图2-42是恢复信号的错误EOP波形。USB主机发出的用于唤醒低速设备的恢复信号的两位SE0状态持续时间大约为10μs。该值已经远远超过USB规范规定的1.25μs到1.5μs的范围。在实际测试中发现这个错误的EOP会导致大部分的USB设备不会被唤醒。

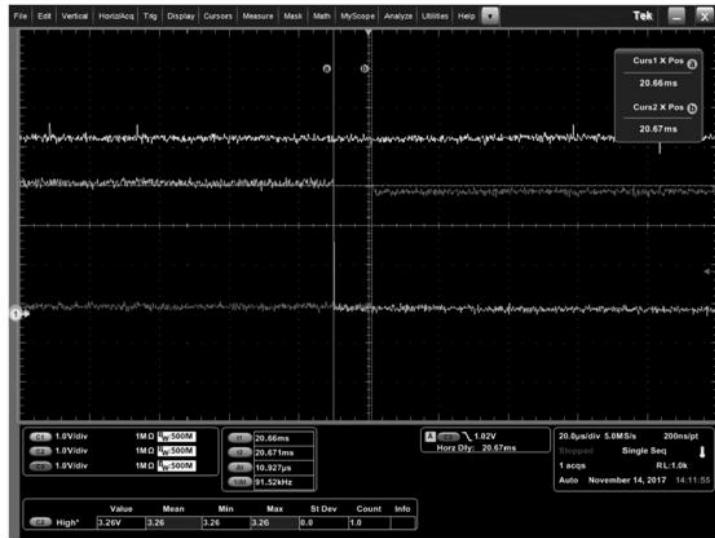


图2-42 恢复信号的错误EOP波形

由于恢复信号的发起者不同，主机和设备在整个总线恢复中起到的角色和介入的时间各有不同。以下主要从两个方面讨论：主机主动发起的恢复和设备主动发起的恢复。同时由于Hub的存在，本节将从上述两个方面分别讨论Hub在总线恢复中起到的作用。

## 1. 主机主动发起的恢复

由于总线挂起情况不同，主机唤醒总线的方式也是不同的。本节将分别从两种总线挂起状态讲述恢复。

### 1) 全局总线挂起

在全局总线挂起的情况下，主机通过直接驱动根端口产生恢复信号唤醒总线。不管网络拓扑结构中有无Hub，主机处理都是相同的。当Hub检测到上行端口的恢复信号时，Hub无条件将信号转发给所有使能的下行端口，如图2-43所示。

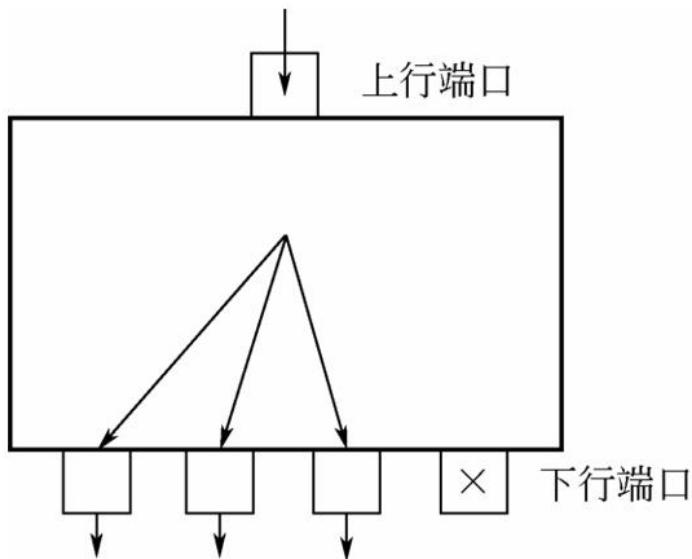


图2-43 上行端口的恢复信号在Hub上的流向

## 2 ) 部分总线挂起

主机首先找到被唤醒设备所在网络拓扑结构中的离设备最近的处于活动状态（非挂起状态）的 Hub。然后，向该 Hub 发送 ClearPortFeature ( PORT\_SUSPEND ) 命令。Hub 在收到命令后向该命令所对应的端口发送恢复信号。此时，该 Hub 的行为和全局总线挂起的 USB 主机唤醒总线的行为类似。在收到主机的端口恢复请求后，Hub 通过驱动总线向该端口发送复位信号，从而唤醒目标端口所连接的部分总线。

## 2. 设备主动发起的恢复

设备主动发送恢复信号需要同时满足以下三个前提条件：

- 设备支持远程唤醒功能（配置描述符的bmAttributes属性的Bit5如果是1）。
- 主机开启了设备的远程唤醒（SetFeature ( DEVICE\_REMOTE\_WAKEUP ) ）功能。

- 设备检测到总线空闲状态至少持续了5ms ( 3ms的挂起时间及2ms的等待时间 , 以保证Hub进入挂起状态并准备好转发恢复信号 )。

由于全局总线挂起和部分总线挂起在设备主动发起恢复信号的情况下 , 主机、Hub和设备的各自行为是一致的 , 所以本节将统一介绍。

设备产生恢复信号至少需要持续1ms , 但是不能超过15ms。在停止发送恢复信号后 , 设备保持它的信号引脚处于高阻态。如果主机和设备之间存在挂起状态的Hub , 那么该Hub在收到恢复信号后需尽可能快地 ( 必须在1ms之内 ) 将信号传播给它的上行端口和其他使能的下行端口 , 如图2-44所示。同时该Hub在15ms内 ( 从检测到上行端口的恢复信号开始计时 ) 必须停止向上行端口发送恢复信号 , 且开始将上行端口的恢复信号转发到所有使能的下行端口。设备产生的恢复信号将一直向上行端口传播 , 直到到达主机或者非挂起的Hub。当主机或者活动状态的Hub检测到恢复信号后 , 其必须在1ms以内产生恢复信号 , 同时驱动信号的持续时间不得小于20ms。这里需要强调的是 , 如果主机和设备之间存在活跃的Hub , 该Hub将充当主机的角色 , 即驱动USB总线产生恢复信号并不会向上传播下行端口的恢复信号。

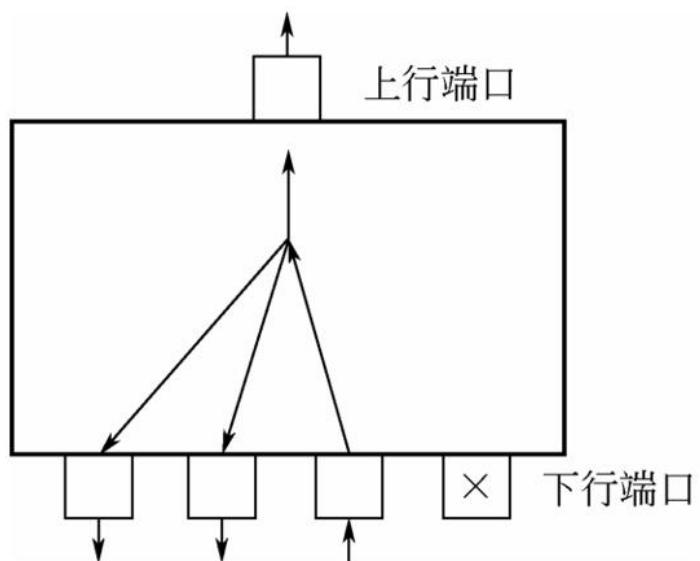


图2-44 下行端口的恢复信号在挂起的Hub上的流向

图2-45是全速设备主动发起的恢复信号示例，在该示例中，局部设备被挂起并且该设备设置了远程唤醒功能。Hub A和Hub B是高速设备，挂起的设备是全速设备。Hub A与Hub B所连接的总线被系统挂起，所以Hub B是挂起状态，Hub A是活动状态。 $t_n$ 是挂起的设备远程唤醒总线的过程中的主要时间节点，具体各时间节点的具体事件如下：

$t_0$ ：挂起的设备在该时刻通过驱动USB总线产生信号K开始远程唤醒USB总线。

$t_1$ ：挂起的Hub B在收到下行端口的K状态后将恢复信号传播到上行端口和所有使能的下行端口。当前示例中，Hub B从检测到K信号到开始驱动恢复信号的时间间隔是900 $\mu$ s。

$t_2$ ：活动状态的Hub A在挂起的端口（Hub B所连接的端口）上检测到K状态后驱动总线向Hub B发送K信号。当前示例中，Hub A从检测到K信号到开始驱动恢复信号的时间间隔是900 $\mu$ s。

$t_3$ ：在驱动USB总线产生的K信号持续一段时间（如10ms）后，挂起的设备停止驱动总线。

$t_4$ ：Hub B在驱动USB总线产生的K信号持续一段时间（如10ms）后，停止驱动总线并将下行端口的信号转播给所有使能的端口。

$t_5$ ：Hub A在驱动总线产生恢复信号一段时间后（不小于20ms）产生恢复结束信号。由于Hub B工作在高速模式，Hub A驱动的恢复信号的结束阶段是高速的SE0。而Hub B在检测到高速SE0后获知总线恢复进入了结束阶段，它根据每个端口连接设备的速度发送对应速度模式的结束信号。例如，端口处于全速或低速模式，那Hub B将会驱动总线产生两位的低速SE0及一位该端口速度模式下的J信号。如果端口处于全速模式，那J信号就是全速J信号。否则，J信号就是低速J信号。如果端口处于高速模式，那Hub B将会驱动总线产生高速SE0。

在恢复信号时序完成后，主机或者非挂起的Hub必须在3ms内产生SOF包，以防止总线再次进入挂起状态。同时，USB 主机系统必须提供 10ms 的恢复时间，在此期间主机不能访问任何刚刚恢复的设备。

图2-46是两种典型的全局总线挂起和恢复流程图。主机在挂起总线前，将所有支持远程唤醒功能的设备全部使能。图2-46的左侧是主机主动恢复总线的流程图，右侧是设备主动恢复总线的流程图。

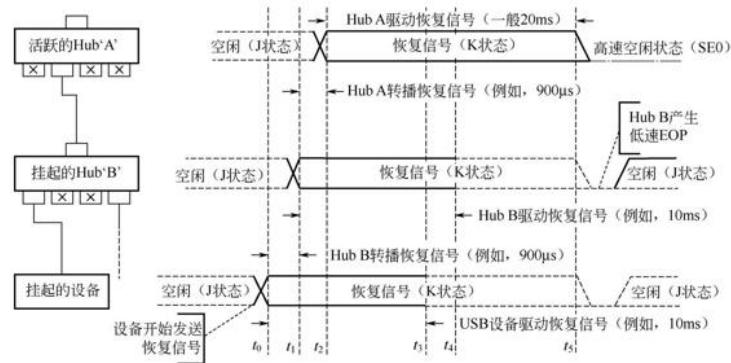


图2-45 全速设备主动发起的恢复信号示例



图2-46 两种典型的全局总线挂起和恢复流程图

另外，本节开始也提到过设备移除事件也会唤醒总线。如果设备直接连接主机，处于睡眠状态的主机会被一个设备移除事件唤醒；如果设备通过Hub连接主机，那么所连接的Hub会收到一个设备移除事件。这里存在两种情况：Hub处于挂起状态和Hub处于活动状态。

### 1 ) Hub处于挂起状态

当设备移除时，如果Hub的远程唤醒功能在挂起前已经被系统使能了，那么Hub会在被设备移除事件唤醒后向上行端口发送恢复信

号，此情形和设备主动发起的恢复信号类似。至于Hub在向上行端口发送恢复信号的同时会不会向所有的下行端口发送恢复信号由具体的Hub决定。不管如何实现，有一个结果可以确定—USB网络中的每一个设备收到的恢复信号都不会小于20ms。

如果Hub的远程唤醒功能没有被系统使能，那么Hub可能可能会被唤醒，也可能不会，由具体的Hub决定，但是Hub一定不会将恢复信号转播给上行端口和任何下行端口。在这种情形下，设备的移除不会被系统感知，也不会唤醒总线。

## 2 ) Hub处于活动状态

如果设备移除，主机会收到Hub的端口状态变化通知。在收到端口变化消息后，USB主机通过获取端口状态获知设备被移除，然后开始设备移除事件的相关处理。

### 2.4.3 高速模式下的挂起信号和复位信号的区别

在高速模式下，复位信号和挂起信号的起始部分是相同的。也就是说，高速设备在检测到持续空闲状态（SE0）时，并不知道该信号的具体含义。因此，USB 设备在检测到连续的空闲（3ms）时，它必须在第 3.125ms 内恢复到全速模式（移除高速终端电阻，并且用 $1.5\text{k}\Omega$ 的电阻将D+上拉）。在恢复到全速模式后，高速设备在  $100\mu\text{s}$  到  $875\mu\text{s}$  之间开始检测总线状态。如果总线处于全速的J状态，则设备认为这是一个总线挂起事件。设备开始总线挂起相关的处理。如果总线是全速的 SE0 状态，则设备认为这是一个复位事件，开始高速检测握手逻辑。高速设备检测到总线 3ms 的空闲后进入挂起状态，其波形如图 2-47所示。高速设备在进入高速模式后收到总线复位事件，其波形如图 2-48所示。高速设备在检测到总线空闲3ms后通过移除高速终端电阻加载全速上拉电阻，开始检测总线状态。如果总线是全速J状态，则USB设备认为收到的是总线挂起事件。否则USB设备认为收到的是总线复位事件。

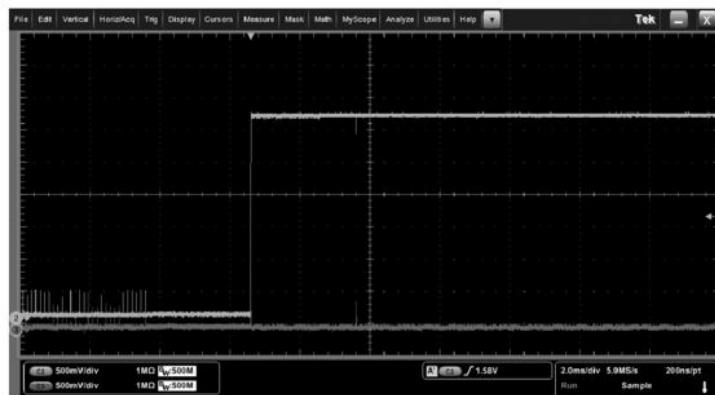


图2-47 高速模式下设备进入挂起状态的波形

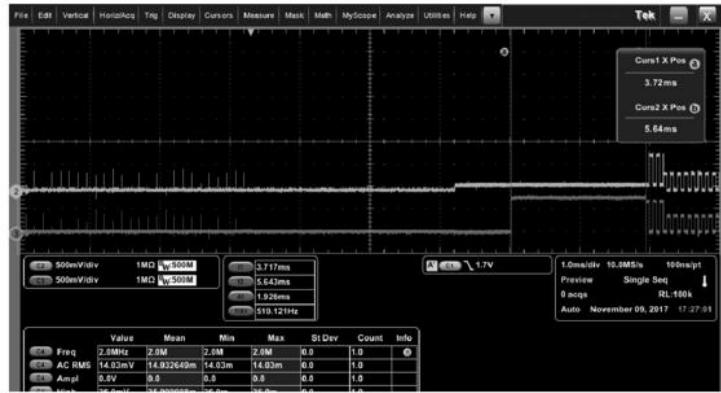


图2-48 高速设备的复位信号波形

这里需要补充说明的是，一般的USB控制器在设计时，为了降低功耗，正常状态和低功耗模式状态使用了不同的电源和时钟电路。无论主机（使能了远程唤醒功能）还是设备，在进入低功耗模式前都需要确保其恢复信号监测模块的时钟和电源都已使能，以确保主机或者设备能够正确地检测到对方的恢复信号。在实际开发中，往往有部分开发者忽视主机低功耗模式的电源和时钟使能，导致USB主机无法检测到USB设备发出的恢复信号，从而导致唤醒不成功。

## 2.5 端点停止

握手包中有一个STALL包，由USB设备方发出，用来表示某个端点不能发送或者接收数据，或者表示不支持控制传输的某种请求。端点一旦发出STALL包，表示其处于停止状态，在停止状态被清除前，该端点不能行使其功能。同时，只有USB设备可以返回STALL包，USB主机在任何情况下都不能发送STALL包。针对USB设备的不同端点，STALL包在两种不同的情况下使用：功能停止（Function Stall）和协议停止（Protocol Stall）。

## 2.5.1 功能停止

功能停止（Function Stall）是指USB设备端点的停止（Halt）特性被设置导致该端点不能行驶其功能。设备端点的停止特性被设置有两种情况：USB设备设置和USB主机设置。功能停止一般不会在控制端点0中使用。

需要注意的是，在功能停止的情况下，同一端点的不同方向上的停止特性相互独立，如IN端点1处于停止状态不影响OUT端点1的通信。

### 1. USB设备设置停止特性

对于非控制端点，如果USB设备无法正常响应USB主机发出的请求，USB设备主动把该端点置为停止（Halt）状态，并使用STALL包通知主机。USB主机可以通过ClearFeature（ENDPOINT\_HALTI）请求来清除该端点的停止状态。在该端点的停止状态被清除前，对于主机对该端点的任何请求，USB设备会一直回复STALL包。

图2-49是设备设置端点停止特性示例。由于某些原因，大容量存储设备主动设置端点为停止状态，主机收到STALL包后，通过ClearFeature（）清除端点的停止状态并继续通信。其流程如下：



图2-49 设备设置端点停止特性示例

(1) USB主机发送一个Inquiry请求，希望设备返回长度为0xff的数据，而当前的USB设备可以返回数据的长度不足，设备根据纯批量传输协议（Bulk-Only Transport），把IN端点1主动置为停止（Halt）状态并返回STALL包。

(2) USB主机收到IN端点1返回的STALL包后，使用ClearFeature（ENDPOINT\_HALTI）清除IN端点1的停止状态。

(3) USB主机通过READ CAPACITY(10)命令成功地获取了设备的信息。可以看到，IN端点1恢复了正常状态并可以返回相应的数据完成与USB主机的正常通信。

## 2.USB主机设置停止特性

当发生错误时，USB主机也可以通过SetFeature（ENDPOINT\_HALTI）请求主动将USB设备的端点设置为停止状态。在错误被处理后，USB主机可以通过ClearFeature（ENDPOINT\_HALTI）请求将该端点的停止状态清除。在该端点的停止特性被清除前，对主机发送给该端点的任何请求，USB设备会一直回复STALL包。

图2-50是主机首先设置端点停止特性，其次再清除停止特性状态，最后恢复正常通信的例子。USB主机在Transfer 15中通过SetFeature（ENDPOINT\_HALTI）

图2-50 主机设置停止特性示例

将设备的OUT端点2设置为停止状态。之后在Transaction 138中，当主机试图与被停止的端点通信时，USB设备回复STALL包。由于本次事务(Transaction 138)未传输成功，协议分析仪无法将其解析成一个完整的传输(Transfer)，因此事务(Transaction 138)以一个单独的事务显示。主机在Transfer 16通过ClearFeature(ENDPOINT\_HALTI)清除掉OUT端点2的停止状态后，USB设备在Transfer 17中就能正常响应主机，并完成传输。

需要注意的是，无论是设备设置停止特性还是主机设置停止特性，对于中断和批量端点，在收到ClearFeature ( ENDPOINT\_HALT ) 的请求后，该端点上的数据PID值应该置为DATA0，否则就可能因主机和设备之间不同步而导致错误。

## 2.5.2 协议停止

协议停止（Protocol Stall）只能在控制端点0上使用。如果设备不支持USB主机发送的某个请求，或者该请求的参数不合法，或者存在其他与USB规范冲突的行为，设备会在控制传输（Control Transfer）的数据或者状态阶段返回STALL包。对于协议停止，其与功能停止的不同之处在于：

- 当USB设备在收到USB主机发送的下一个Setup包时，USB设备的控制端点0的停止状态要被自动清除并正常响应该Setup包。
- 在收到下一个Setup事务前，设备对于控制端点0上的通信，无论IN还是OUT方向都会持续回复STALL包。

图2-51是协议停止示例，其流程如下：

（1）在Transaction 27中，主机通过Get\_Descriptor（DEVICE\_QUALIFIER）请求获取设备的限定描述符（Device Qualifier Descriptor），设备回复确认（ACK）握手包表示已经收到该请求。由于设备并没有限定描述符，无法响应该请求，设备将控制端点0设置为停止状态。

（2）在Transaction 29中，主机发IN包获取设备的限定描述符，设备回复了STALL包，同时设备自动清除控制端点0的停止状态。

（3）在Transaction30中，主机通过Get\_Descriptor（DEVICE）请求获取设备的设备描述符，设备回复ACK包表示已经收到该请求且其停止状态位已被清除。

（4）在Transaction 32和Transaction 33中，设备正常响应主机的通信，完成USB主机获取设备描述符的请求。

Transfer	F	Control	ADDR	ENDP	bfRequest	wValue	wIndex	NotEnough Data	Descriptors	STALL
7	S	GET	19	0	GET_DESCRIPTOR	DEVICE_QUALIFIER	type 0x0000	0 bytes	DEVICE_QUALIFIER Descriptor	0x08
27	S	SETUP	ADDR	ENDP	T D Tp R bfRequest	wValue wIndex wLength	wLength	ACK		
29	S	IN	ADDR	ENDP	STALL				设备在端点0回复stall	
Transfer	F	Control	ADDR	ENDP	bfRequest	wValue	wIndex	Descriptors		
30	S	GET	19	0	GET_DESCRIPTOR	DEVICE type 0x0000	DEVICE Descriptor			
30	S	SETUP	ADDR	ENDP	T D Tp R bfRequest	wValue wIndex wLength	wLength	ACK		
32	S	IN	ADDR	ENDP	T' Data	ACK			设备通过端点0发送数据	
33	S	OUT	ADDR	ENDP	T' Data	ACK				

图2-51 协议停止示例

## 2.6 OTG简介

目前，越来越多的设备使用 USB 作为其传输接口，这样就出现了许多不具有经典PC全部功能的电子设备需要直接和USB外设进行数据交互的问题。例如，相机直接控制打印机、手机直接读写U盘、平板电脑连接鼠标和键盘等。

这些非PC的USB产品具有部分管理USB主机资源的能力，但是它们和标准PC是不同的。虽然能够为某些USB设备提供主机功能，但是它们不一定会支持所有的USB设备。例如，虽然相机可以控制打印机，但是相机厂商不会认为它同时需要识别鼠标。

OTG ( On-The-GO ) 协议就是在这个背景下产生的。OTG产品是一种便携的使用单个Mirco-AB接口的设备，它既可以做主机，又可以做设备。当连接到标准的USB主机时，OTG设备只能工作在USB设备模式。

同时，OTG设备可以互相连接。OTG规范能够使得OTG设备在无须修改USB电缆方向的情况下，实现USB主机和设备之间的角色切换。

## 2.6.1 角色

OTG使用了两种角色：A设备和B设备。

A设备主要是指具有标准的A接口或者其Mirco-AB插入了一个Micro-A接头的设备。A设备为V<sub>BUS</sub>提供电源且在会话开始时使能的是主机功能。如果A设备是一个具有OTG功能的设备，在具有OTG功能的B设备连接的条件下，A设备可能会放弃主机功能。

B设备是一个具有以下接口的设备，如标准B、Mini-B、Micro-B和Mirco-AB等接口。B设备在会话开始时使能的是USB设备功能。如果B设备是一个具有OTG功能的设备，在连接到具有OTG功能的A设备的条件下，B设备可能会使能为主机功能。

插入A插头的OTG设备就是A设备。插入B插头的OTG设备就是B设备。在没有插头的情况下，OTG设备就是B设备。在实际应用中，支持OTG的设备可以通过Mirco-AB接口的ID引脚状态来判断是什么角色的设备。如果ID引脚是高电平，那么设备就是B设备。如果ID引脚是低电平，那么设备就是A设备。

## 2.6.2 协议

OTG 规格说明书定义了三种协议，会话请求协议（Session Request Protocol , SRP）、主机角色协商协议（Host Negotiation Protocol , HNP）和连接检测协议（Attach Detection Protocol , ADP）。当B设备连接到A设备时，A设备作为USB主机开始枚举B设备。A设备可以通过单独请求B设备的OTG配置描述符（GetDescriptor ( OTG )）来获取B设备的能力。或者，B设备在回复GetDescriptor ( Configuration ) 请求时将OTG描述符一并返回。B设备将它是否支持SRP、HNP或者ADP的信息通过它的OTG描述符返回给A设备。如果不支持OTG描述符，B设备应该在控制传输的数据阶段返回STALL。如果支持，B设备应该在所有的USB配置下都支持OTG描述。

### 1.会话请求协议

为了节约电能，A设备允许在USB总线没有任何设备连接的情况下关闭 $V_{BUS}$ 。B设备可以通过SRP请求A设备打开 $V_{BUS}$ ，从而开始建立会话。

任何的A设备（包括PC或者笔记本）都可以响应SRP请求。任何的B设备（包括只支持USB设备模式）都可以发起SRP请求。同时，支持HNP的B设备必须要支持SRP。

当一个A插头插入时，如果目标主机关闭了 $V_{BUS}$ ，那它需要响应SRP请求。反之，如果目标主机一直打开 $V_{BUS}$ ，那它就不用响应SRP请求。

图2-52是A设备在SRP流程中的时序要求。图2-53是B设备在SRP流程中的时序要求。在会话结束后，A设备禁止 $V_{BUS}$ 。OTG协议要求 $V_{BUS}$ 从4.0V下降到 $V_{OTG\_VBUS\_LKG}$ 的时间 $T_{SEND\_LKG}$ （ $\leq 0.7V$ ）最大不超过1s。当会话结束后，如果B设备需要建立新会话，那么它需要确

保上次会话结束到再次发起SRP请求的时间间隔不能小于  $T_{B\_SEND\_SRP}$  ( $\geq 1.5s$ )，同时还要满足USB总线SE0状态至少持续  $T_{B\_SE0\_SRP}$  ( $\geq 1s$ )。在上述条件满足后，B设备通过挂载电阻上拉 D+，上拉持续时间  $T_{B\_DATA\_PLS}$  (5ms到10ms)，然后移除上拉电阻，使得总线再次进入SE0状态。A检测到在总线的D+线的脉冲后使能  $V_{BUS}$ ，同时必须在  $T_{B\_SRP\_FAIL}$  (5 ~ 6s) 内将  $V_{BUS}$  拉升到  $V_{OTG\_SESS\_VLD}$ 。在会话有效后 ( $V_{BUS}$  电压达到  $V_{OTG\_SESS\_VLD}$ )，B 设备必

须在时间  $T_{B\_SVLD\_BCON}$  ( $\leq 1s$ ) 内将 D+ 电压上拉到  $V_{IH}$ 。而 A 设备必须在时间  $T_{A\_VBUS\_RISE}$  ( $\leq 100ms$ ) 内将  $V_{BUS}$  拉升到 4.4V。之后，A 设备一直等待 B 设备连接 (等待总线的 J 状态)，等待时间为  $T_{A\_WAIT\_BCON}$  ( $\geq 1.1s$ )。在检测到 J 状态且 J 状态持续时间超过  $T_{A\_BCON\_LDB}$  ( $\geq 100ms$ ) 后，A 设备复位总线并开始枚举 B 设备。

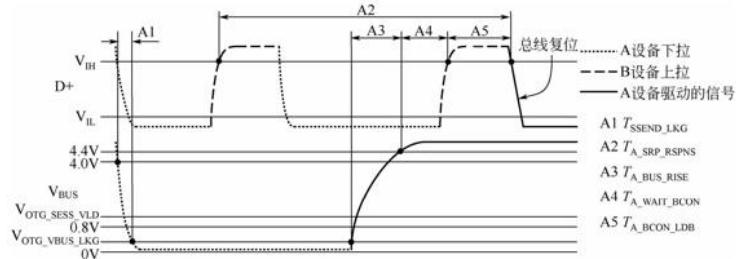


图2-52 A设备在SRP流程中的时序要求

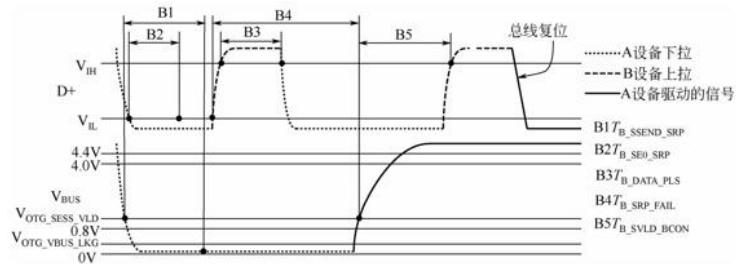


图2-53 B设备在SRP流程中的时序要求

## 2. 主机角色协商协议

两个OTG设备连接在一起后，A设备默认是USB主机，B设备默认是USB设备。在支持HNP的条件下，B设备可以通HNP转换为USB主机。而此时A设备就变成USB设备。相反，A设备可以再次切回USB主机模式。HNP主要用于两个直接连接的OTG设备之间在无用户修改USB线缆方向的情况下进行USB主机角色切换。典型的HNP应用是应用程序或者用户在OTG功能的B设备主动发起的。不论A设备是否是USB主机，它始终为USB接口供电( $V_{BUS}$ )。同时，A设备需要无条件的使能B设备的HNP功能，以确保B设备在任何时候都能切到主机模式。同样地，无论任何时候，A设备需要切回USB主机模式时，主机模式的B设备都需无条件停止主机功能。

表2-15列出了OTG协议所能支持的A设备和B设备的HNP组合的所有情况。从表2-15中可以看出，如果B设备能够切到USB主机模式，那么A设备一定支持HNP协议。另外，OTG协议不强制OTG设备都支持HNP协议。

表2-15 OTG协议支持的HNP列表

OTG协议是否支持	A设备是否支持HNP	B设备是否支持HNP
支持	否	否
支持	是	否
支持	是	是
不支持	否	是

在开始建立会话时，A设备默认是USB主机。在会话建立后，A设备和B设备可以通过HNP不限次数地来回切换USB主机和设备角色。

图2-54是HNP的时序图，是A设备和B设备的操作时序在D+线上的复合波形。具体流程如下：

(1) A设备在完成所有总线事务后停止总线的所有传输，如通过挂起总线停止总线。

(2) B设备在总线空闲后不小于 $T_{B\_AIDL\_BDIS}$  ( $4ms \leq T_{B\_AIDL\_BDIS} \leq 150ms$ ) 的时间后通过禁止D+上拉开始HNP请求。之后，总线进入SE0状态。

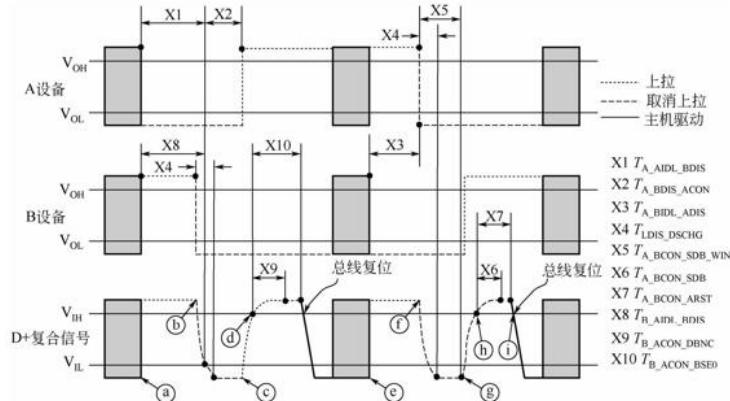


图2-54 HNP的时序图

(3) 当检测到总线的SE0状态后，A设备认为B设备发起了HNP请求。A设备在 $T_{A\_BDIS\_ACON}$  ( $\leq 150\text{ms}$ ) 时间内通过挂载D+上拉电阻响应B设备的请求。如果A设备在 $T_{A\_AIDL\_BDIS}$  ( $\geq 200\text{ms}$ ) 时间内没有检测到设备移除事件，那么该设备停止检测并终止当前会话。

(4) B设备在等待 $T_{LDIS\_DSCHG}$  ( $\geq 25\mu\text{s}$ ) 时间后检测总线是否为J状态。此次等待是为了避开B设备上拉D+所引起的电源剩余效应产生的误判。如果检测到有效的J状态且J状态持续时间超过 $T_{B\_ACON\_DBNC}$  ( $\geq 2.5\mu\text{s}$ )，那么B设备认为 A 设备识别了 HNP 请求。此时，B 设备切换到主机模式并且在 $T_{B\_ACON\_BSE0}$  ( $\leq 150\text{ms}$ ) 时间内产生总线复位。

(5) 在使用完USB总线后，B设备停止总线活动。如果检测到全速的总线空闲状态，那么B设备可以上拉D+。

(6) 当检测到总线空闲超过 $T_{A\_BIDL\_ADIS}$  ( $155\text{ms} \leq T_{A\_BIDL\_ADIS} \leq 200\text{ms}$ ) 后，A设备移除D+的上拉电阻。此时如果没有任何事务需要处理，A设备可能会直接禁止 $V_{BUS}$ 并结束当前会话。

(7) B设备上拉D+电阻。如果在 $T_{A\_WAIT\_BCON}$  ( $\geq 1.1\text{s}$ ) 之前B设备还没有连接，那么A设备可以停连接检测。

(8) A设备在等待 $T_{LDIS\_DSCHG}$  ( $\geq 25\mu s$ ) 时间后检测总线是否为J状态。此次等待是为了避开A设备上拉D+所引起的电源剩余效应产生的误判。如果检测到有效的J状态，那么A设备认为B设备已经连接并且以USB设备模式准备好响应USB主机请求。

(9) 在A设备检测到B设备的D+上拉信号后且信号持续时间超过 $T_{A\_BCON\_SDB}$  ( $\geq 2.5\mu s$ ) 后，A设备可以切换到主机模式并且在 $T_{A\_BCON\_ARST}$  ( $\leq 30s$ ) 时间内产生总线复位或者直接结束当前会话。

### 3.HNP状态轮询

HNP状态轮询是一种允许当前作为USB主机的OTG设备决定其他请求变成USB主机的OTG设备（当前运行在USB设备模式）在何时变成USB主机的机制。

在以USB设备角色运行的OTG设备连接到以主机角色运行的OTG设备上时，OTG主机需要定期轮询设备状态，以判断设备是否想转换到主机模式。在主机看到设备想切换到主机模式时，OTG主机需要尽可能快地满足该USB设备的需求。

不论任何时候，当两个OTG设备连接在一起后，如果B设备支持HNP功能，主机以 $T_{HOST\_REQ\_POLL}$  ( $1s \leq T_{HOST\_REQ\_POLL} \leq 2s$ ) 周期地通过GetStatus()命令获取USB设备保存在OTG状态信息中的主机请求状态。当检测到主机请求被设置后，主机必须在 $T_{HOST\_REQ\_SUSP}$ （不超过2s）内允许USB设备切换到主机模式。

B设备成功切换到USB主机模式需要同时满足以下几个条件：

- B设备支持HNP功能；
- B设备设置了HNP请求状态（GetStatus(OTG status)）；
- A设备使能了B设备的HNP的功能（SetFeature(b\_hnp\_enable)）。

## 4.连接检测协议

ADP协议主要用于一个本地设备检测是否有外部设备插入。该本地设备可以是OTG A设备、OTG B设备、嵌入式主机或者支持SRP但只支持USB设备模式的B设备。外部设备可以是任何USB设备。

ADP 是基于两个设备在插入或移除时的 V<sub>BUS</sub>线的电容变化的原理工作的。支持ADP的本地设备首先通过对V<sub>BUS</sub>线放电检测到电容，然后测量用已知电流将V<sub>BUS</sub>线充电到已知电压所用的时间。通过测量的时间变化来检测电容的不同。

如果连接在一起的A设备和B设备都支持ADP，那么A设备需要做ADP检测而B设备需要做ADP感知。所谓ADP感知就是检测B设备需要V<sub>BUS</sub>线上是否有ADP活动。

## 2.6.3 OTG设备状态迁移

本节主要介绍B设备如何从USB设备模式转为USB主机模式。

### 1.B设备

图2-55是B设备的状态机。

OTG设备在上电后，通过检测ID引脚的状态判断其是A设备还是B设备；如果ID是高电平，设备就是一个B设备；如果ID是低电平，设备就是一个A设备。

#### 1)b\_idle

当ID为高电平时，B设备进入b\_idle状态。如果B设备检测到ID为低电平时，B设备转变为A设备并转到a\_idle状态。

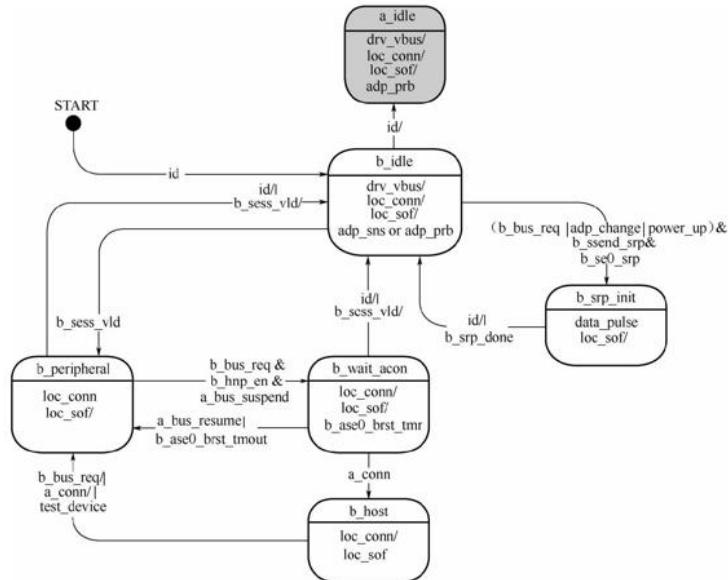


图2-55 B设备的状态机

在当前状态，B设备一直检测 $V_{BUS}$ 是否高于会话有效阈值 $V_{OTG\_SESS\_VLD}$  ( $0.8V \leq V_{OTG\_SESS\_VLD} \leq 4.0V$ ) 并且置 **b\_sess\_vld** 为

真。如果条件有效，B设备进入b\_peripheral状态并开始一个连接（loc\_conn为真）。在实际应用中，如果是全速设备，通过在DP引脚加载终端电阻来实现这一连接。同时，由于硬件原因，实际应用中 $V_{OTG\_SESS\_VLD}$ 可能是0.8V到4.0V之间的任意值。

从当前状态转入b\_srp\_init需要同时满足以下条件：

- 应用或者用户主动开始一个会话（b\_bus\_req为真），或者检测到ADP变化（adp\_change为真），或者B设备刚刚上电（power\_up为真）；
- 检测到 $V_{BUS}$ 低于会话结束阈值（低于 $V_{OTG\_SESS\_VLD}$ ）的时间超过 $T_{B\_SEND\_SRP}$ （b\_ssend\_srp为真）；
- BUS在SE0状态的持续时间至少超过 $T_{B\_SE0\_SRP}$ （b\_se0\_srp为真）。

## 2)b\_srp\_init

在进入当前模式后，B设备开始使用SRP进行会话初始化。一旦SRP完成（b\_srp\_done为真），B设备立即返回b\_idle状态等待A设备驱动 $V_{BUS}$ 高于 $V_{OTG\_SESS\_VLD}$ 。

## 3)b\_peripheral

在当前状态下，B设备以USB设备的身份与A设备通信，开始正常的USB事务。

如果ID引脚为低电平或者 $V_{BUS}$ 低于 $V_{OTG\_SESS\_VLD}$ ，B设备从当前状态返回b\_idle状态。

当以下条件同时满足时，B设备进入b\_wait\_acon状态：

- 应用程序想以主机的模式开始一个会话（b\_bus\_rep为真）；

- A设备通过SetFeature ( b\_hnp\_enable ) 允许B设备的请求 ( b\_hnp\_en为真 ) ；

- 总线处于挂起状态 ( a\_bus\_suspend为真 ) 。

#### 4)b\_wait\_acon

进入当前状态后，B设备断开DP引脚上的上拉电阻，开始计时 ( b\_ase0\_brst\_tmr )，同时等待A设备连接。

同样地，如果ID引脚为低电平或者 $V_{BUS}$ 低于 $V_{OTG_SESS_VLD}$ ，B设备从当前状态返回b\_idle状态。

当以下任一条件达成时，B设备退回到b\_peripheral状态：

- 如果等待超时，

$b_{ASE0\_BRST\_TMR} \geq T_{B\_ASE0\_BRST}$  ( b\_ase0\_brt\_tmout为真 ) ；

- 如果B设备检测到K状态，表明A设备正在发送恢复信号 ( a\_bus\_resume为真 ) 。

如果在时钟 $b_{ASE0\_BRST\_TMR}$ 超时前检测到A设备的连接 ( B设备的D+引脚被上拉 )，B设备进入b\_host状态。B设备必须在检测到D+上拉的 $T_{B\_ACON\_BSE0}$  ( $\leq 150ms$ ) 时间内进入b\_host状态。

#### 5)b\_host

在进入当前状态后，B设备复位USB总线，并且开始产生SOF。然后B设备开始正常的USB事务。当B设备不支持A设备时，B设备需要将信息报告给用户。

当以下任一条件达成时，B设备将返回b\_peripheral状态：

- 当B设备完成了它的任务 ( b\_bus\_req为假 ) ；

- 当B设备检测到A设备移除了 ( a\_conn为假 ) ；

- 30s内B设备检测到插入的是一个测试设备（test\_device为真）。

## 2.A设备

图2-56是A设备的状态机。

### 1)a\_idle

当ID引脚为低电平时，OTG设备就是一个A设备。当同时满足以下条件时，A设备进入a\_wait\_vrise状态：

- 应用程序没有关闭总线电源（a\_bus\_drop为假）；
- 以下任一条件为真：
  - ◆ 应用程序请求使用总线（a\_bus\_req为真）；
  - ◆ 检测到B设备的SRP（a\_srp\_det为真）；
  - ◆ 检测到ADP变化（adp\_change为真）；
  - ◆ A设备刚刚上电（power\_up为真）。

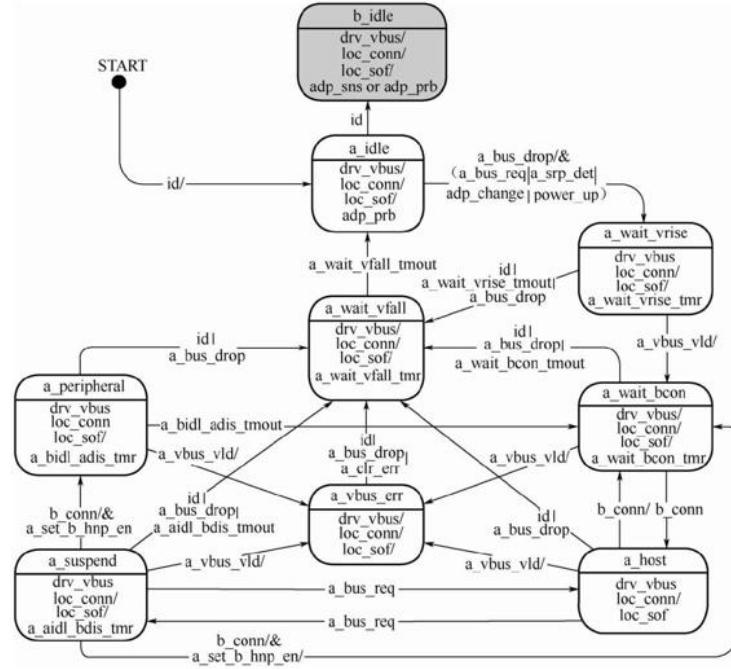


图2-56 A设备的状态机

另外，如果OTG设备不支持ADP功能，ID 引脚由高电平到低电平的变化也会触发a\_bus\_req为真。

## 2)a\_wait\_vrise

在当前状态，A 设备主要检测 V<sub>BUS</sub> 的电压是否正常，并且用定时器a\_wait\_vrise\_tmr判断检测是否超时。

当V<sub>BUS</sub>在T<sub>A\_VBUS\_RISE</sub> ( $\leq 100\text{ms}$ ) 时间内有效 (a\_vbus\_vld为真)，那A设备进入 a\_wait\_bcon 状态。如果 ID 引脚为高电平、等待超时 (a\_wait\_vrise\_tmout 为真) 或者应用程序需要关闭总线电源 (a\_bus\_drop 为真)，那么A设备进入a\_wait\_vfall状态。

## 3)\_wait\_bcon

当进入当前状态后，A 设备检测 B 设备是否连接，并且用定时器a\_wait\_bcon\_tmrv判断等待是否超时。

类似地，如果 ID 引脚为高电平、等待超时 ( a\_wait\_bcon\_tmout 为真 ) 或者应用程序需要关闭总线电源 ( a\_bus\_drop 为真 )，那么 A 设备进入 a\_wait\_vfall 状态。

如果 B 设备连接上 ( b\_conn 为真 )，那么 A 设备进入 a\_host 状态，必须在  $T_{A\_BCON\_ARST}$  ( 不超过 30s ) 时间内复位总线，并且开始 USB 事务。

如果检测到  $V_{BUS}$  过流事件，则 A 设备需要关闭  $V_{BUS}$  并且进入 a\_vbus\_err 状态。

#### 4) a\_host

进入该状态后，A 设备开始枚举 B 设备。如果检测到 B 设备并不支持，A 设备设置 a\_bus\_req 为假。

如果能够支持 B 设备，那 A 设备就开始正常的 USB 事务。如果 A 设备和 B 设备都支持 HNP，那么 A 设备就以  $T_{HOST\_REQ\_POLL}$  的周期轮询主机请求状态标志位。如果 B 设备断开连接 ( b\_conn 为假 )，A 设备进入 a\_wait\_bcon。类似地，如果 ID 引脚为高电平或者 a\_bus\_drop 为真，那么 A 设备进入 a\_wait\_vfall 状态。如果检测到  $V_{BUS}$  过流事件，A 设备需要关闭  $V_{BUS}$  并且进入 a\_vbus\_err 状态。

如果不再需要使用总线 ( a\_bus\_req 为假 )，A 设备进入 a\_suspend 状态。所谓不在需要总线有两种可能：应用程序主动释放总线，或者检测到主机请求标志位为真。如果检测到主机请求标志位为真，那么 A 必须在  $T_{HOST\_REQ\_SUSP}$  ( 不超过 2s ) 内通过命令 SetFeature ( b\_hnp\_enable ) ( 同时设置 a\_set\_b\_hnp\_en 为真 ) 允许 B 设备的请求并且挂起总线。

#### 5) a\_suspend

如果 B 设备的 HNP 请求已经批准 ( a\_set\_b\_hnp\_en 为真 )，那么 A 设备开启一个定时器 a\_aidl\_bdis\_tmr ( 超时时间  $T_{A\_AIDL\_BDIS}$  ) 用于检

测B设备的移除。

类似地，如果ID引脚为高电平、等待超时（`a_aidl_bdis_tmout`为真）或者应用程序需要关闭总线电源（`a_bus_drop`为真），那么A设备进入`a_wait_vfall`状态。

如果B设备移除了且`a_set_b_hnp_en`为真，则A设备进入`a_peripheral`状态。如果应用程序通过设置`a_bus_req`再次请求总线，A设备通过恢复总线返回`a_host`状态。如果远程唤醒功能已经使能，那么A设备在检测到恢复信号后设置`a_bus_req`。如果B设备移除了（`b_conn`为假）且`a_set_b_hnp_en`为假，则A设备进入`a_wait_bcon`状态。

#### 6) `a_peripheral`

在当前状态下，A设备以USB设备的身份与B设备通信，开始正常的USB事务。

类似地，如果ID引脚为高电平或者应用程序需要关闭总线电源（`a_bus_drop`为真），那么A设备进入`a_wait_vfall`状态。如果在 $T_{A\_BIDL\_ADIS}$ 的最大时间（200ms）内，A设备检测到连续的总线空闲，其将进入`a_wait_bcon`状态。如果由于 $V_{BUS}$ 过流，A设备进入`a_vbus_err`状态。

#### 7) `a_wait_vfall`

当进入当前状态后，A设备关闭 $V_{BUS}$ 并且等待 $V_{BUS}$ 降到 $V_{OTG\_VBUS\_LKG}$ （ $\leq 0.7V$ ）以下。

在 $T_{SSEND\_LKG}$ （ $\leq 1s$ ）时间后，A设备进入`a_idle`状态。

#### 8) `a_vbus_err`

进入当前状态后，A设备等待 $V_{BUS}$ 过流状态恢复。

如果ID引脚为高电平、系统软件清除了错误或者应用程序需要关闭总线电源（`a_bus_drop`为真），那么A设备进入`a_wait_vfall`状态。

## 2.6.4 OTG工作流程

本节以双方都支持HNP协议的OTG设备为例，简单地介绍OTG设备之间的工作流程，如图2-57所示。基本的工作流程如下：

(1) OTG设备在上电后根据ID引脚的状态初始化相应的角色。在默认状态下，OTG设备双方没有连接任何线缆。所以两个设备都初始化为B设备。

(2) 在其中一个设备检测到Micro AB插头后，该设备切换到A设备模式。在进入A设备模式后，该设备使能 $V_{BUS}$ 用于检测是否有设备连接（主要用于检测不支持SRP的设备）。由于USB线缆未连接，A设备未检测到USB设备（检测超时），并禁止 $V_{BUS}$ 。

(3) 在USB线缆连接后，由于 $V_{BUS}$ 无效，B设备未检测到连接事件。此时，系统软件主动发起一次SRP请求以发起会话连接。在检测到有效的SRP后，A设备使能 $V_{BUS}$ 。

(4) 在检测到有效的 $V_{BUS}$ 后，B设备使能USB设备功能。A设备检测到USB设备后开始枚举设备并开始USB事务。同时，A设备开始轮询B设备的主机请求标志。

(5) 当需要切换主机模式时，B设备设置主机请求标志。在检测到主机请求标志被置位后，A设备发送SetFeature ( b\_hnp\_enable ) 请求使能B设备的HNP。

(6) 在请求成功后，A设备以挂起总线的方式释放总线。在检测到总线挂起后，B设备禁止上拉D+线的电阻并使能主机功能。在检测到B设备移除后，A设备使能USB设备功能。

(7) B设备检测到USB设备后开始枚举设备并开始USB事务。

(8) 在使用完成总线后，B设备以挂起总线的方式释放总线。在检测到总线挂起后，A设备禁止上拉D+线的电阻并使能主机功能。在检测到A设备的移除后，B设备使能USB设备功能。

(9) A设备检测到USB设备后再次开始枚举设备并开始USB服务。



图2-57 OTG设备之间的工作流程

## **第3章**

# **USB硬件控制器**

在前面的章节中介绍了 USB 2.0 协议规范的内容，详细讲解了 USB 协议，本章将结合恩智浦公司微控制器中两种常用的 USB 控制器，以“理论联系实践”来讲解如何在实际的硬件控制器上实现 USB 协议，并对实际开发过程中可能会碰到的具体问题进行讨论，以加深读者对协议的理解。

### 3.1 增强主机控制器接口

增强主机控制器接口（Enhanced Host Controller Interface，EHCI）规范是由Intel公司主导设计，并于1999年发布的用于实现USB 2.0规范的寄存器级的主机控制器接口规范。从1999年发布的0.8a版本到2002年发布的1.0版本，EHCI规范经历多次更新，这些版本更新使EHCI规范趋于稳定。该规范定义了系统软件与USB主机控制器之间的访问接口。图3-1是USB主机系统框图，灰色部分为EHCI规范所能覆盖的范围。

EHCI规范中只定义了USB主机控制器接口部分并没有定义USB设备的控制器接口规范，所以对于USB设备的接口，不同厂家实现的接口和方式都可能不同。恩智浦公司推出的MK65FN2M0CAC18不仅包括高速主机功能，还包括高速设备功能。本章将以该芯片为例讲解USB主机和USB外设的控制器行为及软硬件间的交互流程。

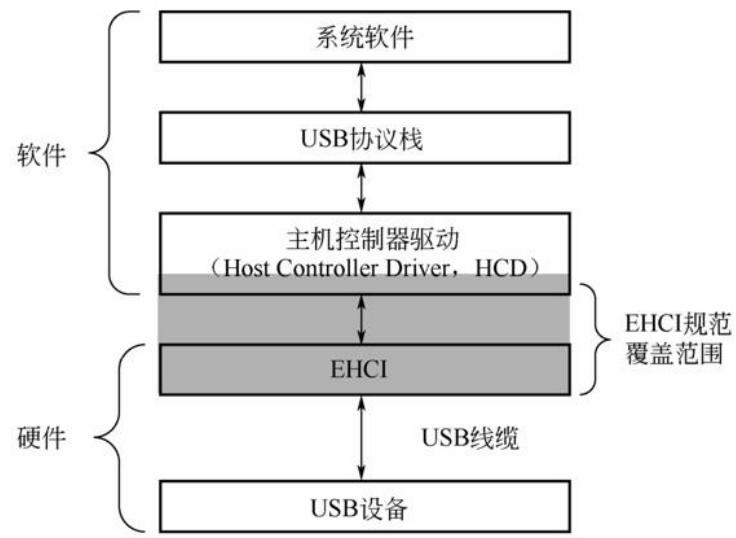


图3-1 USB主机系统框图

### 3.1.1 寄存器接口

由于 EHCI 主机和设备控制器的寄存器较多，本节只列出本书中涉及的寄存器，如表3-1所示。如果读者想了解其他寄存器，请参考 EHCI 规范和恩智浦公司 MK65FN2M0CAC18 的芯片手册。

表3-1 EHCI主机和设备控制器的寄存器

寄存器名称	位	位域	描述
HCCPARAMS	1	PFL	帧列表长度可编程标志： 0—帧列表长度不可变，固定为 1024 1—帧列表长度可变，由 USBCMD[FS] 和 USBCMD [FS2] 决定
USBCMD	0	RS	控制器运行/停止控制位 主机模式： 清除该位时，主机在完成当前传输事务后停止调度。 USBSTS[HCH] 表示控制器是否处于停止状态 设置该位后，主机执行调度。只有主机处于停止状态时 (USBSTS[HCH] 为 1)，软件才能设置该位 设备模式： 清除该位时，控制器会禁止设备功能 使能该位时，控制器开始设备功能
	2~3	FS	该位只有在主机模式时有效。和 FS2 结合起来用于控制帧列表长度（以 4 字节为一个单位）： 帧列表长度 = $1024 \div 2^{\text{FS}} \div 2^{\text{FS2}+4}$

续表

寄存器名称	位	位域	描述
USBCMD	13	SUTW	该位不是 EHCI 规范定义的，且只有在设备模式有效。该位作为一个信号量使用，用于保证软件能够获取一个完整的 Setup 数据
	15	FS2	该位不是 EHCI 规范定义的。用法见 FS 位描述
USBSTS	12	HCH	该位是一个只读位，且只有在主机模式时有效。该位主要用于显示当前控制器的状态。当 RS 位设置时，该位自动清除
PERIODICLISTBASE	12~31	PERBASE	该位域主要用于存储周期性传输列表的基地址（高 20 位）。该位只在主机模式有效
FRINDEX	0~13	FRINDEX	当前帧号
ASYNCLISTADDR	5~31	ASYBASE	用于保存异步传输链表中正在执行的 QH 地址。该位只在主机模式有效
EPLISTADDR	11~31	EPBASE	用于保存异步传输列表的首个 dQH 的地址。该位只在设备模式有效
OTGSC	19	BSVIS	该位是一个中断标志位，表示 BSV 发生了变化
	11	BSV	该位是一个状态位： 0—VBUS 低于 B 会话有效的阈值 1—VBUS 高于 B 会话有效的阈值
DEVICEADDR	24	USBADRA	该字段提供了一种提前写入 USB 设备地址的机制。在收到 SetAddress() 请求后，软件将设备地址提前写入 USBADR，同时设置该位。在状态阶段完成后，硬件自动使用 USBADR 更新设备地址
	25~31	USBADR	设备的当前地址
EPCRn	0	RXS	接收端点（OUT 端点）是否处于 stall 状态： 0—正常状态 1—stall 状态
	6	RXR	该位用于复位接收端点的 data toggle 位。软件设置该位后，data toggle 复位为 0
	16	TXS	发送端点（IN 端点）是否处于 stall 状态： 0—正常状态 1—stall 状态
	22	TXR	该位用于复位发送端点的 data toggle 位。软件设置该位后，data toggle 复位为 0
PORTSC	0	CCS	当前连接状态位： 0—设备不存在（主机模式），断开状态（设备模式） 1—设备存在（主机模式），连接状态（设备模式）
	1	CSC	连接变化状态位。该位只支持主机模式。用于反映 CCS 位发生了变化。当 CCS 位变化后，该位被置位或该位被设置为 1

续表

寄存器名称	位	位域	描述
PORTSC	6	FPR	<p>该位不是 EHCI 规范定义的。</p> <p>主机模式： 软件通过设置 FPR 产生恢复信号。该位在恢复信号结束后会被硬件自动清除</p> <p>设备模式： 在远程唤醒功能使能后，软件设置 FPR 产生恢复信号。在一段时间（如 10ms）后，软件主动清除该位</p>
	7	SUSP	<p>主机模式： 当该位设置后，主机停止总线活动，让总线处于挂起状态</p> <p>设备模式： 该位是一个只读位，用于反映总线是否处于挂起状态： 0—端口没有处于挂起状态 1—端口处于挂起状态</p>
	8	PR	<p>主机模式： 该位用于控制总线产生复位信号。 当软件设置该位时，总线产生 USB 复位信号。在复位完成后，硬件自动清除该位。该位和 EHCI 规范定义的行为有所不同</p> <p>设备模式： 该位是一个只读位，用于反映 USB 总线是否处于复位状态： 0—端口不处于复位状态 1—端口处于复位状态</p>
	9	HSP	<p>高速模式标志： 0—全速或者低速 1—高速</p>
	23	PHCD	该位不是 EHCI 规范定义的。该位控制 PHY 进入低功耗模式
	26~27	PSPD	端口速度，该位不是 EHCI 规范定义的。该位和 HSP 功能类似： 0—全速 1—低速 2—高速 3—未定义

### 3.1.2 EHCI连接/断开检测

正如1.2节描述，连接和断开是USB协议的一个重要组成部分，是实现USB设备热插拔的必要前提。因此，EHCI规范提供了连接和断开检测功能特性。所以，在设计硬件时就需要考虑连接和断开检测功能特性。由于USB主机和设备的连接/断开的检测机制是不同的，所以本节将分别从USB主机和设备两个方面进行介绍。

#### 1. USB主机的连接和断开检测

USB 2.0协议规范规定了USB主机对设备连接和断开的检测时序。在EHCI规范中，这些检测过程都由硬件实现，对软件并不可见，软件只需要读取最终的检测结果，这样的设计大大简化了软件开发的复杂度。

EHCI规范在端口状态和控制寄存器（Port Status and Control Registers，PORTSC）中定义了两个位域：连接状态变化位（Connect Status Change，CSC）和当前连接状态位（Current Connect Status，CCS），软件可以根据这两个位域的值来判断当前的连接状态。

当前连接状态位是一个只读位，反映了当前是否有设备连接。当设备连接时，其值为1；当没有设备连接时，其值为0。

连接状态变化位反映了连接状态是否发生了变化。USB主机控制器在检测到设备的连接状态发生变化后，设置连接状态变化位并产生中断。在其中断处理函数中，软件通过连接状态变化位和当前连接状态位来判断是否产生了设备连接和移除事件。同时通过写1清除连接状态变化位状态；否则，USB主机控制器将会持续不断地产生中断导致其功能无法正常完成。

图3-2是USB主机的连接和断开检测流程。

#### 2. USB设备的连接和断开检测

USB设备主要通过  $V_{BUS}$  的状态和变化来检测设备的连接和断开。USB协议规范规定  $V_{BUS}$  不小于4.01V时视为  $V_{BUS}$  有效。而在实际硬件实现时，从成本和易实现角度考虑，使用 B 设备会话有效 ( $V_{B\_SESS\_VLD}$  ,  $0.8V \leq V_{B\_SESS\_VLD} \leq 4.01V$ ) 作为USB设备连接和断开检测的标的。即当  $V_{BUS}$  上升到  $V_{B\_SESS\_VLD}$  以上时，设备认为  $V_{BUS}$  有效；当  $V_{BUS}$  下降到0.8V以下时，设备认为  $V_{BUS}$  无效。

MK65FN2M0CAC18的USB设备控制器也实现了类似的机制：B会话有效中断状态 (B Session Valid Interrupt Status, BSVIS) 和B会话有效 (B Session Valid, BSV)。

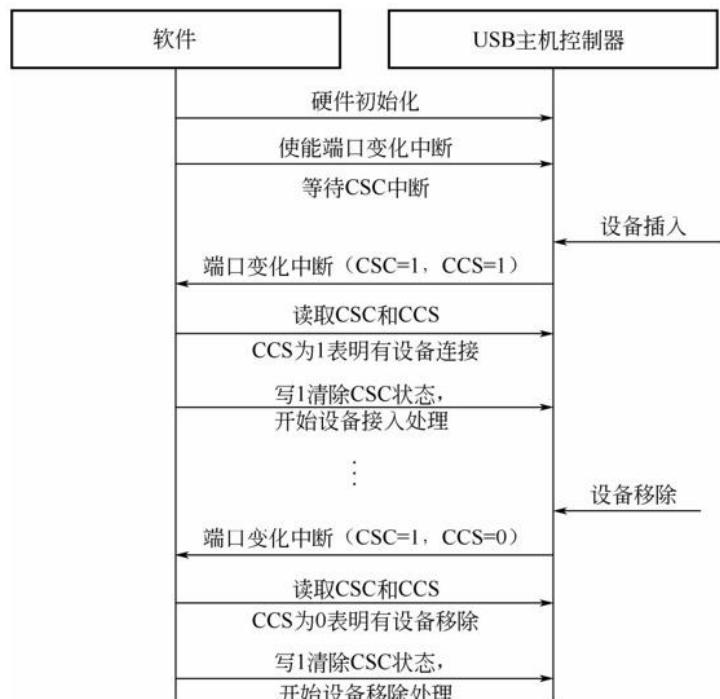


图3-2 USB主机的连接和断开检测流程

BSV是一个只读位，反映了当前  $V_{BUS}$  是否大于B会话有效的阈值。当  $V_{BUS}$  大于  $V_{B\_SESS\_VLD}$  时，BSV 为 1；当  $V_{BUS}$  小于  $V_{B\_SESS\_VLD}$  时，BSV 为 0。

BSVIS反映了BSV位是否产生了变化。USB设备控制器检测到 $V_{BUS}$ 由有效变无效或者由无效变有效后，设置BSVIS位并产生中断。在中断处理函数中，软件通过BSVIS和BSV位判断设备的 $V_{BUS}$ 是否有效，从而判断设备是否连接到USB主机。同时通过写1清除BSVIS状态。

图3-3是USB设备的连接和断开检测流程，显示了USB设备和软件对于设备连接和移除事件之间的交互。在实际应用中，当USB设备检测到 $V_{BUS}$ 有效时，软件通过置USBCMD寄存器的RS位（设置该位后设备控制器会使能D+引脚的上拉电阻，使得主机能够检测到设备）使能USB设备的功能，开始USB的枚举流程。当 $V_{BUS}$ 无效时，软件清除USBCMD寄存器的RS位。

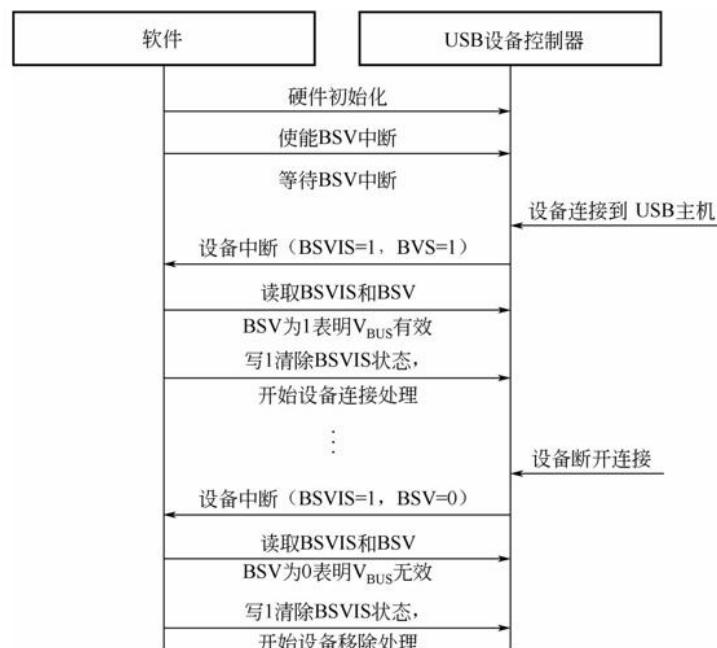


图3-3 USB设备的连接和断开检测流程

### 3.1.3 EHCI速度检测

速度检测是USB主机和USB设备能够正常协同工作的前提。不仅USB主机需要识别设备的速度，设备也需要知道其当前工作在哪种速度模式下。因为在不同速度模式下，USB设备的配置可能不同。例如，在全速模式下，大容量存储类USB设备的批量传输端点支持的最大包长度是64字节。但是，在高速模式下，大容量存储类USB设备的批量传输端点支持的最大包长度是512字节。

#### 1. USB主机的速度检测

EHCI规范提供PORTSC[HSP]位域用于获取当前端口连接的设备是否工作在高速模式下。该位域只有一位，只表示设备是否工作在高速模式。但主机无法获知设备是全速设备还是低速设备。为此，MK65FN2M0CAC18 在PORTSC 寄存器增加了一个两位的位域PORTSC[PSPD]，用于获取当前端口的速度模式。具体值定义是0—全速，1—低速，2—高速。

USB主机在复位设备后，通过寄存器PORTSC的PSPD位域获取设备接入设备的速度。值得注意的是，USB主机必须在复位流程结束后才能获取设备的速度，以保证获取的USB设备速度是有效的。在实际应用中，HCD 通过设置PORTSC[PR]产生复位信号复位USB总线。USB主机控制器在检测到PORTSC[PR]后驱动USB 总线产生复位信号，在复位结束后将PORTSC[PR]清除。所以，HCD在获取设备速度前需要一直轮询PORTSC[PR]，直到该位被清除，然后通过PORTSC[PSPD]获取当前USB 设备的速度。图 3-4是USB主机的速度检测流程。



图3-4 USB主机的速度检测流程

## 2.USB设备的速度检测

对于USB设备模式而言，在收到端口变化的中断消息时，软件可以通过PSPD或者HSP位获取当前设备的速度模式。根据当前速度，USB设备软件配置相关参数并初始化控制管道，等待接收USB主机的控制请求。图3-5是USB设备的速度检测流程。

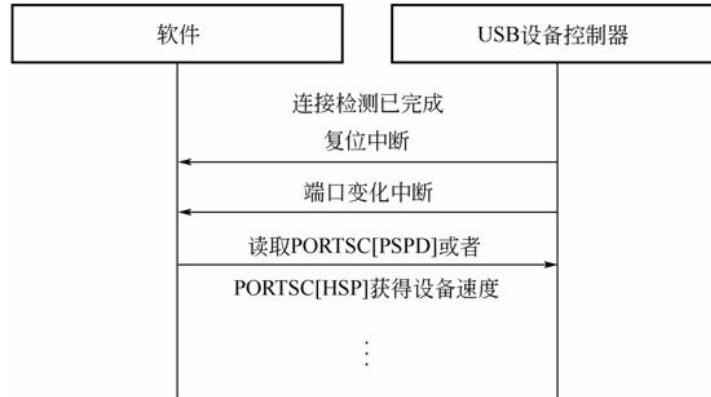


图3-5 USB设备的速度检测流程

### 3.1.4 USB主机传输调度机制

根据传输性质的不同，EHCI规范定义了两种传输列表：周期传输列表和异步传输列表。周期传输列表主要用于管理周期性的传输，包括中断传输和同步传输。异步传输列表主要用于管理非周期性的传输，包括控制传输和批量传输。

#### 1. 周期传输列表

对于周期性的传输管道，主机控制器需要每隔一定的时间检查各个管道是否有事务需要开始传输。对于检查的时间间隔，不同的端点可能各不相同，其主要由端点描述符的bInterval字段决定。EHCI规范根据这种特性设计了周期传输列表，用于简化软硬件设计，同时尽可能地保证高实时性。

周期传输列表是一个由软件申请、软件维护、硬件访问的数组。周期传输列表必须是4KB对齐的，以保证硬件能够通过PERIODICLISTBASE[31-12]和FRINDEX[N-3]寄存器值索引到对应的周期性传输单元。数组的成员由4个字节表示，即每个项占4个字节。数组的长度可以由软件自定义，最大为1024个项（4096个字节）。如果硬件不支持长度自定义（HCCPARAMS[PFL]为0），那么长度固定为1024。否则，长度由USBCMD[FS]字段决定。EHCI规范定义了3种长度：1024、512和256。即使最小的长度（256）对于内存空间的需求也高达1KB，对于嵌入式应用这是难以接受的。为此，MK65FN2M0CAC18对于该字段进行了扩展（增加了字段USBCMD[FS2]），以降低对硬件资源的占用。当USBCMD[FS2]为0时，长度可以为1024（N=12）、512（N=11）、256（N=10）和128（N=9）。当USBCMD[FS2]为1时，长度可以为64（N=8）、32（N=7）、16（N=6）和8（N=5）。

周期传输列表数组的每个成员都是一个指针，指向用于周期传输的数据结构。如图3-6所示，同步传输的传输描述符在前，中断传输的

描述符在后。只有任意两帧都不共享同一同步传输描述符，才能满足这样的要求。这就意味着任意两个数组成员不可能指向同一个同步传输的传输描述符表。然而，由于中断传输存在握手阶段，如果USB设备没有准备好，那么USB主机会在下一个周期再次尝试该传输。

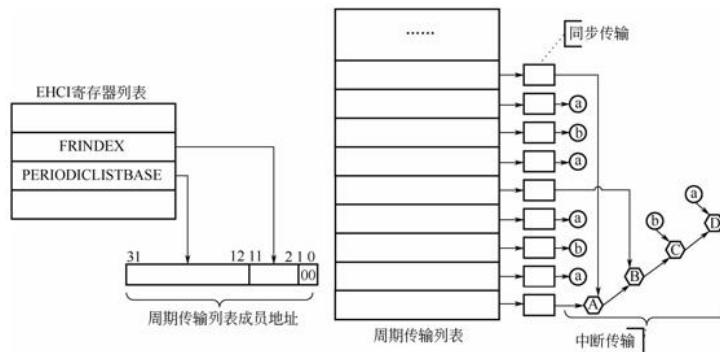


图3-6 周期传输结构框图

根据周期传输的调度特性，离列表越近的描述符传输周期（bInterval）越大。在一个SOF开始，硬件会根据PERIODICLISTBASE[31-12]和FRINDEX [N-3]索引到对应的传输列表，然后一一开始处理所连接的传输描述符。对于全速和低速模式，在1ms内周期传输列表的对应项只会被调度一次。而对于高速模式，在1ms内周期传输列表的对应项会被调度8次。假设图3-6是一个高速USB主机的周期传输列表状态，FRINDEX为0x00 ~ 0x07（0 ~ 1ms内），那么A、B、C和D节点会被调度8次。

如图3-6所示，A传输的周期是8ms，B传输的周期是4ms，C传输的周期是2ms，D传输的周期不超过1ms。在实现USB主机驱动时，软件需要保证所有的周期性传输管道的周期必须是2的指数倍（保证中断传输链表的拓扑结构是一棵二叉树结构），以保证周期传输列表的链表结构不被破坏。例如，一个接入鼠标的中断传输周期是9ms，那么在初始化该传输的管道时软件需要将周期调整为8ms。

### 1) 中断传输

USB主机根据队列头（Queue Head，QH）和队列元素传输描述符（Queue Element Transfer Descriptor，qTD）进行非同步类型的传输调

度。QH是一个qTD链表的头。通常情况下，一个方向的管道会与一个QH一一对应。qTD是实际用于传输的数据结构。一个transfer会包含多个qTD。对于多个transfer，软件需要将所有的qTD根据transfer的先后顺序链接起来形成一个qTD链表，链表的头节点就是QH。图3-7是一个中断传输描述符链表结构图。

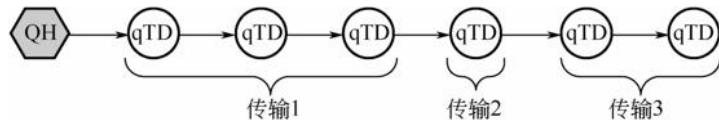


图3-7 中断传输描述符链表结构图

QH保存了管道所对应的USB设备和端点的信息，包括端点支持的最大包长(wMaxPacketSize)、是否是控制类型的传输、设备速度、端点号、USB设备地址等。在创建管道时，软件需要申请一个QH，并根据对应设备和端点的信息将其填充。在QH填充完成后，软件将该QH地址加入相应的传输列表。

对于中断传输，软件根据端点的周期插入相应的位置。例如，当前端点的周期是2ms，那么软件会将QH插入周期为2ms的位置。将该QH(节点F)插入图3-6所描述的QH链表后的状态图如图3-8所示。对于节点C和F，两者的周期相同。所以，F节点可以插入节点B和C之间，也可以插入节点C和节点D之间。根据二叉树的特性，距离根节点的路径长度越短，插入节点对二叉树的修改越少。所以节点F建议插入节点C和节点D之间。这样有助于降低软件设计复杂度，提高软件执行效率。

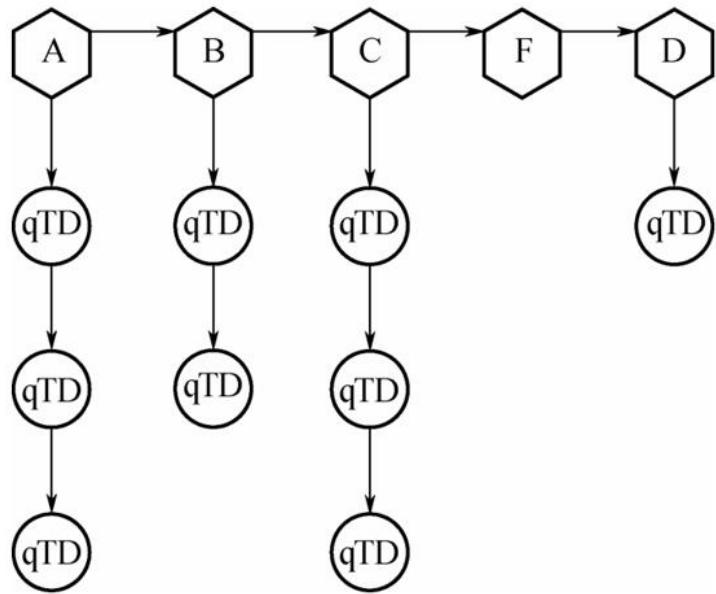


图3-8 中断传输QH插入状态

另外，需要特别提出的是，由于硬件会将每个执行完成后的qTD从QH的传输链表中移除，在准备传输时（申请qTD并将其插入链表），每个transfer需要记录各自的qTD首节点地址，这样有助于HCD在取消transfer时可以快速地找到各自的qTD。同时，每个QH需要额外保存整个管道qTD链表的首节点地址。在传输完成后，HCD可以快速地通过管道的首节点地址找到整个qTD链表。另外，对于qTD来说，HCD需要实现一个从qTD到QH的映射关系，使得在处理传输完成的qTD时，HCD可以快速地找到当前的transfer。

## 2 ) 同步传输

USB主机根据同步传输描述符（Isochronous Transfer Descriptor，iTID）和分片事务同步传输描述符（Split Transaction Isochronous Transfer Descriptor，siTID）进行同步类型的传输调度。iTID用于高速设备同步类型的传输，而siTID用于高速主机与全速设备之间的同步传输。但是如果USB主机工作在全速模式，那么同步传输也是使用iTID来实现。

由于同步传输没有类似于QH的机制，所以USB设备和对应端点的信息都保存在iTID/siTID中。在实现同步传输时，iTID/siTID都是在上层

应用请求数据传输时，HCD才开始分配、填充并且将其连接到周期传输列表的链表中。

图 3-9 显示软件在准备了一个同步类型的传输后的周期传输列表、iTД和待传输缓冲区关系图。上层应用准备的一次传输可能会被划分成多个iTД，同时一个iTД只能通过唯一一个帧号索引。

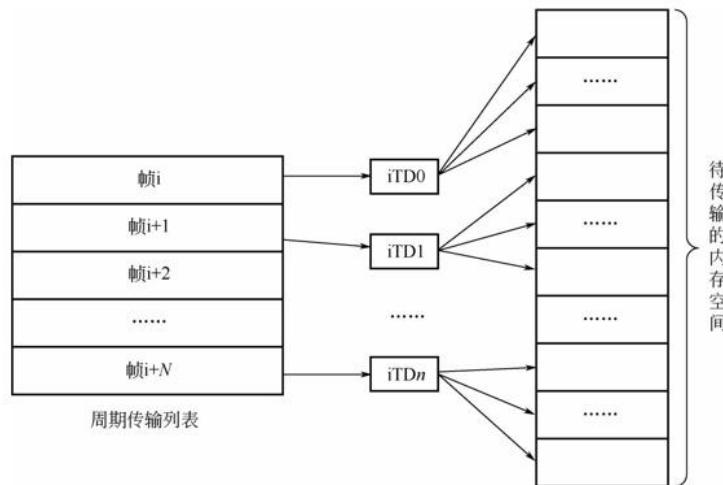


图3-9 一个典型的周期传输列表、iTД和待传输缓冲区关系图

siTD主要是与USB规范中定义的Split传输对应。siTD的原理就是在同一个帧中的某些微帧中发送SSplit，而在其后的某些微帧中发送CSplit。siTD主要是在iTД的基础上增加了分片传输的控制。软件通过当前带宽分配情况、待传输数据的长度、待传输管道的方向和待传输管道的周期等信息计算出SSplit和CSplit。例如，当前的帧号是i，待传输的数据长度是512字节，最大包长是1023字节，传输周期是1ms等。如果传输方向是IN，那么可能的情况是SSplit=8'b00000001，CSplit=8'b01111100。如果传输方向是OUT，那么可能的情况是SSplit=8'b00000111，CSplit=8'b00000000。同时待插入的帧号可能是i+1。这里存在一个问题，如果待插入的帧号是i+1，那就意味着当前的传输可能很危险。因为当siTD链接到周期传输列表时，硬件可能已经处理过了SSplit所指定的微帧，这就使得SSplit并未执行，从而导致异常。那如果待插入的帧号是i+2，就会出现输入的数据并不连续的现象，对于音频类的设备，就会存在噪声的问题。所以在实际应用中，HCD会将上层应用的请求的传输插入先于当前帧的周期传输列表

中。一般设计HCD时待插入帧与当前帧的时间间隔是2ms ( N=2 , 可配置 )。例如 , 当前帧的帧号是 i , 那么待插入帧号是 i+3 。同时 , 这样设计就要求上层应用最少需要准备 N+1 个传输 , 以保证不存在漏帧的情况。即对于同一个同步传输管道 , 在任何时候至少有N+1个有效的iTД/siTД被链接到周期传输列表。

## 2. 异步传输列表

异步传输列表是一个循环链表 , 用于管理控制传输和批量传输。当周期传输结束后 , 主机控制器根据寄存器 ASYNCLISTADDR 开始异步传输调度 , 如图 3-10 所示。

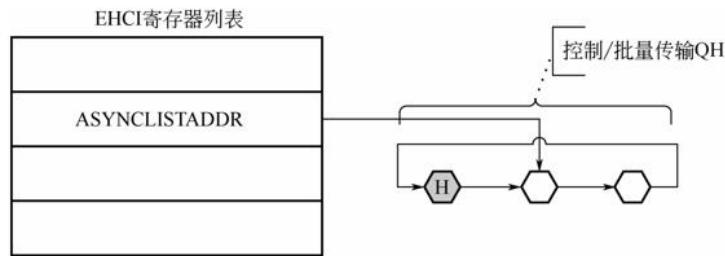


图 3-10 异步传输结构框图

异步传输链表是控制或者批量传输的 QH 链表。在管道初始化时 , HCD 为每个管道申请一个 QH 结构体 , 在配置完成结构体成员后 , 将结构体插入链表 , 完成管道初始化。如果异步列表为空 , 那么当前插入的 QH 的 H 位需要设置为 1 。在实际开发过程中 , 硬件控制器初始化时 , 软件可以添加一个冗余的 QH ( H 位置为 1 ) , 以简化软件设计 , 同时保证硬件正常运行。

### 1 ) 控制传输

和中断传输类似 , USB 主机控制器也使用 QH 和 qTD 管理控制传输。与中断传输不同的有 :

- 中断传输的 QH 插入周期传输列表 , 而控制传输的 QH 是插入异步传输链表。

- 对于Split传输，控制传输QH的SSplit和CSplit都是0。同时控制传输的Data Toggle由每一个qTD决定，而中断传输的Data Toggle由QH控制。

- HCD 只会为每个 USB 设备的控制管道准备一个 QH。即控制传输的发送和接收管道共用一个QH，具体的传输方向由qTD指定。

- 当有控制传输需要发起时，HCD将控制传输的三个阶段分别使用三个qTD实现。

这样设计具有如下优点：

- 降低系统资源开销（节省一个QH空间）。
- 控制传输的时序可控，且能够简化 HCD。如果两个方向分别使用不同的QH，那么一次只能准备控制传输的其中一个阶段。当传输成功后，需要软件再次干预（插入新的qTD），才能开始下一阶段的传输。

图3-11显示了控制传输读写时序，主要列举了控制传输qTD链接的三种示例。

## 2 ) 批量传输

与控制传输类似，USB主机控制器也使用QH和qTD管理批量传输。HCD对批量传输的管理基本和控制传输相似。

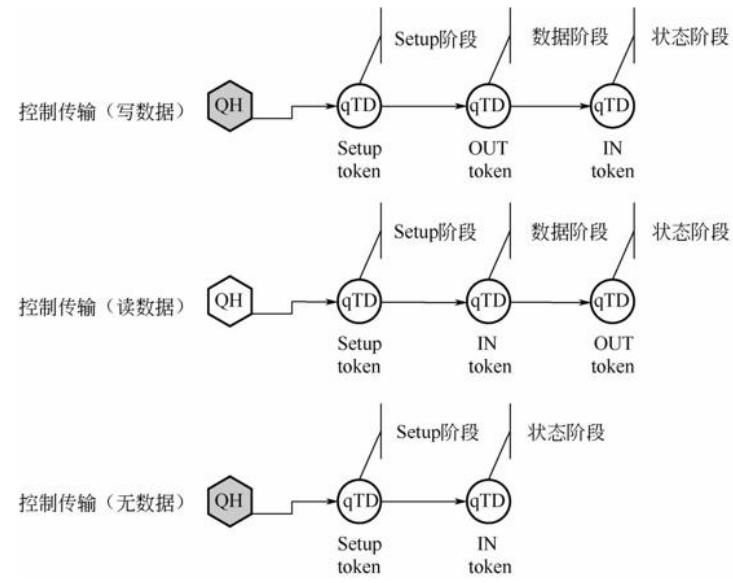


图3-11 控制传输读写时序

### 3.1.5 USB设备传输调度机制

与USB主机模式的传输控制机制不同，USB设备模式的传输控制不区分描述符类型，使用设备端点队列头（Device endpoint Queue Head，dQH）和设备传输描述符（Device Transfer Descriptor，dTД）控制所有类型的传输。USB设备传输端点结构框图如图3-12所示。端点dQH列表是一个由设备控制器驱动（Device Controller Driver，DCD）申请的数组结构列表，保存了所有端点的dQH。DCD可以通过端点号和端点方向索引到对应的dQH。

从软件实现角度上看，dQH和dTД与USB主机的QH和qTD基本相似。值得一提的是，硬件在dQH保留了8个字节的空间（dQH.SetupBuffer）用于保存USB主机发送的Setup数据。当端点0初始化后，USB设备硬件会将USB主机发送来的Setup数据保存到dQH.SetupBuffer中，然后产生一个Setup传输完成的中断消息。在中断处理函数的过程中，DCD从dQH.SetupBuffer获取之前保存的Setup数据。在读取Setup数据时，需要严格按照图3-13所示流程。

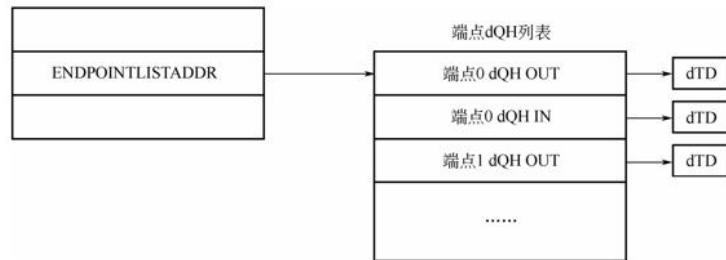


图3-12 USB设备传输端点结构框图

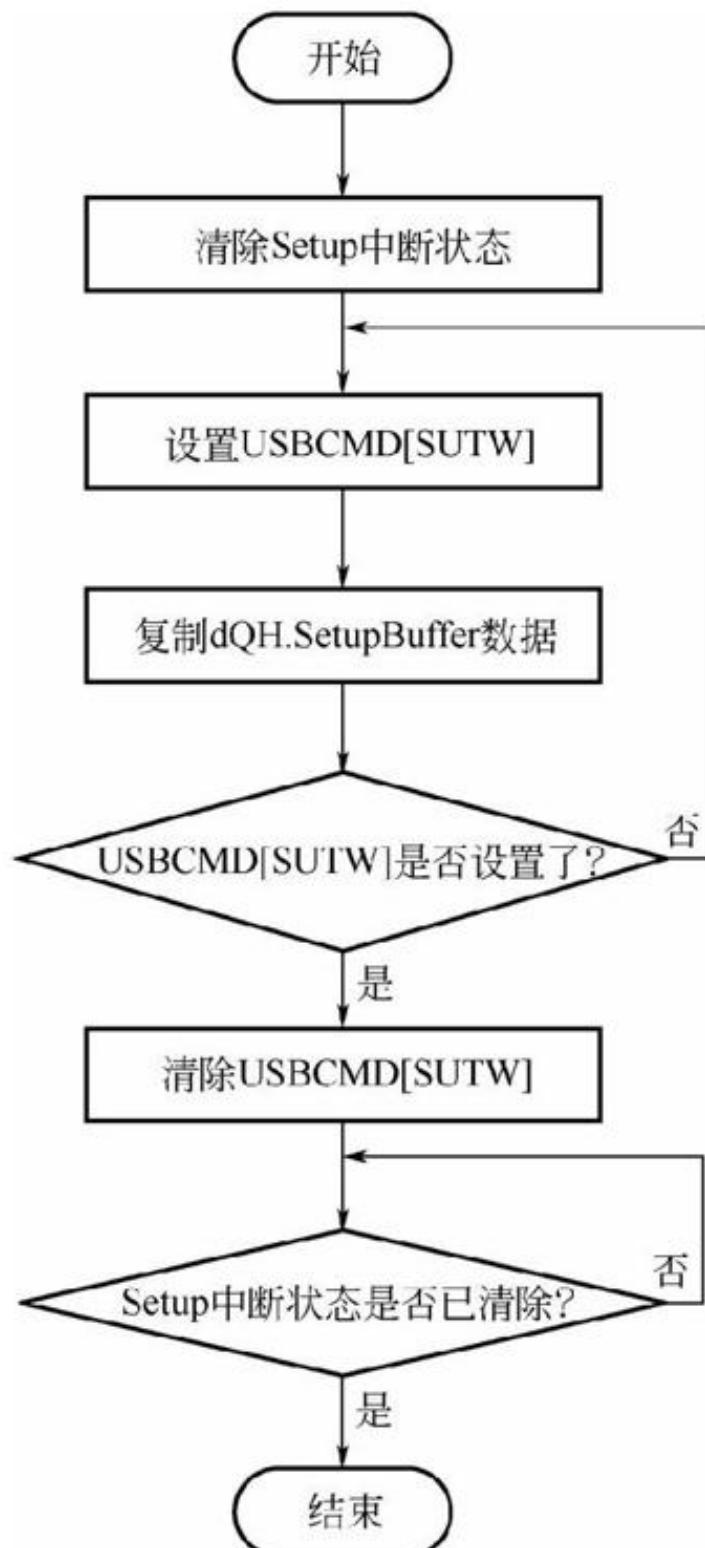


图3-13 Setup数据读取流程

### 3.1.6 EHCI枚举的软件实现

正如2.2节所述，枚举的本质就是USB主机获取USB设备参数信息并配置信息的过程。这也是USB设备能够实现热插拔的重要条件所在。USB规范决定了USB主从结构的特性，任何传输的发起都必须从USB主机开始，USB设备只能被动地响应主机的请求。所以本节的枚举主要是从USB主机的角度出发。

枚举主要出现在设备连接的开始阶段。一般情况下在枚举之前，USB主机会先发送USB复位信号，使得目标USB设备处于默认（default）状态。

当USB设备收到USB复位事件后，系统软件需要将USB设备状态机设置为默认状态，并且初始化控制传输管道，然后准备好接收USB主机发送的Setup请求。当前USB设备控制器有一个特性就是在控制传输管道初始化结束后，硬件会自动准备好（用于保存 Setup 数据的内存空间是管道数据结构的一个域，详见3.1.3节）接收USB主机的Setup请求数据。

对于设置地址请求（SetAddress（）），USB主机在完成该请求（状态阶段传输成功）后，需要等待2ms，以保证USB设备能够恢复到正常状态。而USB设备对于设置地址请求有两种处理机制：

- 在收到地址请求的Setup数据后，直接将地址写入地址寄存器；
- 在状态传输成功后，更新地址寄存器。

对于设置地址请求，当前的USB设备控制器提供了一种软件先设置地址硬件后更新地址的机制。即USB设备在收到设置地址的请求后，将地址直接写入寄存器DEVICEADDR[USBADR]域，同时设置位DEVICEADDR[USBADRA]。图3-14是枚举的部分流程并包含了软件对设置地址请求的处理流程。这样做可以使软件实现简单，无须额外维护这些状态。



图3-14 枚举的部分流程

### 3.1.7 EHCI挂起和恢复

本节将从软硬件两个方面来描述EHCI挂起和恢复的功能是如何实现的。

#### 1.挂起

一般来说，系统软件为了降低系统功耗，会将整个USB总线或者部分总线挂起。本节主要讨论全局总线挂起情况，部分总线挂起不在本节讨论。在总线挂起后，HCD会使能USB控制器的异步唤醒功能（检测USB设备移除、USB恢复信号等）。在将USB控制器设置为异步唤醒源后，系统进入低功耗模式。

当上层软件请求挂起总线时，HCD首先在停止异步传输和同步传输后通过设置PORTSC[SUSP]将总线挂起。在最少等待3ms后，HCD通过清除USBCMD[RS]停止USB主机功能。通过设置PORTSC[PHCD]和USB PHY的PWD寄存器将USB PHY设置为低功耗状态。其次，在USB PHY的CTRL[UTMI\_SUSPENDM]置位后，HCD使能USB控制器的异步唤醒功能，同时关闭USB PHY的时钟。最后，系统软件进入低功耗状态。具体的USB主机挂起总线流程如图3-15所示。

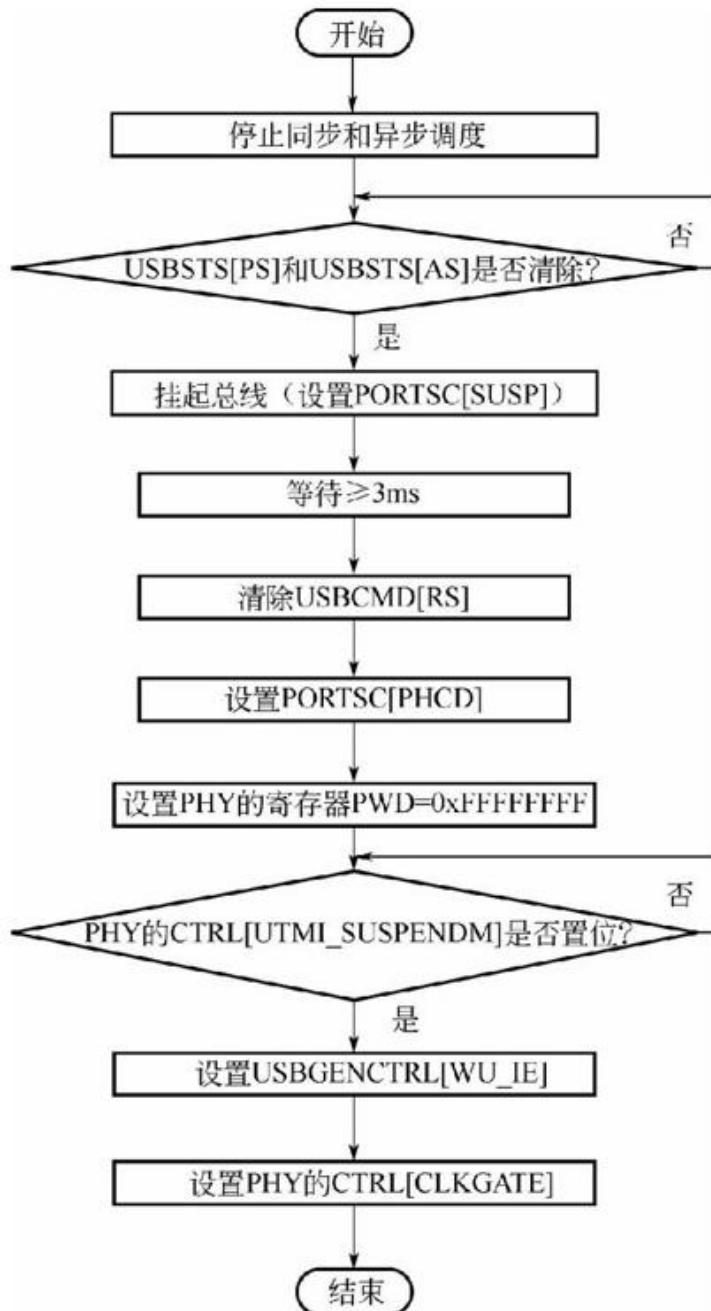


图3-15 USB主机挂起总线流程

对于USB设备，在检测到总线挂起事件后，除了禁止周期和异步传输和清除USBCMD[RS]位外，其DCD设置USB设备控制器的流程和USB主机类似。

## 2.恢复

由于总线恢复根据恢复信号发起源的不同而不同，本节将从USB主机发起的恢复信号和远程唤醒的恢复信号两个角度分别讲述。

### 1 ) USB主机发起的恢复信号

当系统软件需要唤醒USB总线时，HCD通过配置USB主机控制器产生恢复信号。在收到系统软件唤醒总线的请求后，HCD首先清除PORTSC[SUSP]和PORTSC[PHCD]位。如果USB设备还处于连接状态，那么HCD通过USBCMD[RS]使能USB主机功能，然后设置PORTSC[FPR]控制USB主机控制器产生恢复信号。当PORTSC[FPR]位自动清除后，HCD使能同步和异步调度。

### 2 ) USB设备发起的恢复信号

当系统软件需要远程唤醒USB总线时，DCD通过配置USB设备控制器产生恢复信号。在收到系统软件唤醒总线的请求后，DCD首先清除PORTSC[SUSP]和PORTSC[PHCD]位。然后DCD设置PORTSC[FPR]控制USB设备控制器产生恢复信号。在持续时间不小于10ms后，DCD清除PORTSC[FPR]位。

### 3.1.8 EHCI异常处理

本节主要介绍软件是如何通过USB控制器进行异常处理，主要包括两个部分：端点停止和取消传输。

#### 1. 端点停止

端点停止（Endpoint Stall）主要分为两种：功能停止（Function Stall）和协议停止（Protocol Stall）。功能停止可以通过SetFeature（endpoint\_halt）或者ClearFeature（endpoint\_halt）来设置或者清除端点的停止状态。而协议停止一般出现在控制传输的数据阶段。当USB设备不支持USB主机的Setup请求时，USB设备通过控制端点返回停止状态。

USB主机控制器可以通过qTD.Status的bit6获取当前qTD所对应的端点是否是Stall状态。如果USB设备需要返回停止或者收到了SetFeature（endpoint\_halt）请求，DCD将EPCRn寄存器的TXS（IN管道）或者RXS（OUT管道）位设置为1。对于控制传输管道，在返回停止状态后，USB设备控制器会自动将控制端点的TXS和RXS位清除，以准备好接收下一次的Setup传输。或者，当收到新的Setup包后，USB设备控制器会自动清除控制端点的TXS和RXS位。

在清除端点的停止状态后，USB主机和USB设备都需要将其保存的数据toggle状态清除（恢复到DATA0状态）。

对于USB主机，HCD只需要将中断传输和批量传输的数据toggle状态清除（同步传输的数据toggle永远是DATA0，控制传输的数据toggle由qTD控制），“即将对应管道的QH.DT位设置为0”。

对于USB设备，DCD可以通过EPCRn[TXR]或者EPCRn[RXR]将相应端点的数据toggle复位。

#### 2. 取消传输

一般在发生错误时或者由于上层软件管理的需要，HCD或者DCD会取消已经准备好的传输请求。HCD在移除某个或者某些传输时，先根据该传输的传输类型将对应的传输调度功能暂停，在移除该传输的所有传输描述符后，再次开启传输调度功能。例如，对于控制和批量传输，HCD会暂停异步传输调度功能。对于中断传输和同步传输，HCD会暂停周期传输调度功能。

对于USB设备来说，USB设备控制器的相关资料没有对这种情况有特别处理或者说明。所以在设计DCD时，一般只需要将对应传输的dTD移除即可。

## 3.2 Kinetis主机控制器接口

Kinetis主机控制器接口（Kinetis Host Control Interface，KHCI）控制器因在恩智浦Kinetis系列微控制器中被大量使用而得名，是一个设计简单但功能完整的USB控制器，可以工作在单一的主机和设备模式下，配合一些其他的芯片也可以支持OTG工作模式，具有内存占用小并且驱动程序易于开发的特性。KHCI控制器完全兼容USB 1.1和USB 2.0协议，具有如下特点：

- 支持16个双向的端点；
- 支持内部DMA接口；
- 支持低功耗；
- 支持带时钟恢复的IRC48M，但是只工作在设备模式，有利于减小板级晶振使用。

虽然KHCI控制器能支持OTG工作模式，但是本章只讨论KHCI用作主机和设备模式的情况，不讨论KHCI工作在OTG模式的情况。恩智浦公司推出的MK65FN2M0CAC18芯片集成了KHCI控制器，不仅包括主机功能，还包括设备功能。本节将以该芯片为例讲解 KHCI 控制器行为及软硬件间的交互流程。

### 3.2.1 外围硬件设计

KHCI控制器需要USB系统利用外部元器件来确保驱动输出阻抗，以满足眼图和V<sub>BUS</sub>电缆容错要求。印刷电路板（PCB）上KHCI控制器的D+和D-接口必须通过串联电阻（大约33Ω）连接到USB连接器。另外，为了提高信号质量，这两个33Ω电阻需安装得更靠近微控制器而不是连接器端。USB收发器包括：

- D+线上的内部 1.5kΩ 上拉电阻用于做全速设备（由 CONTROL [DPPULLUPNONOTG]或OTGCTL [DPHIGH]来控制）；
- D+和D-信号上的内部15kΩ下拉电阻；

对于内部的15kΩ下拉电阻问题，需要特别注意：

- 设备模式下，通过内部15kΩ下拉可让D+和D-差分线处于稳定状态，如可用在设备没有插入主机时。设备模式内部 15kΩ 下拉由 USBCTRL[PDE]来控制。
- 主机模式下，当CTL[HOSTMODEEN]被设成1后，内部的15kΩ 下拉会自动设置。

图3-16是KHCI主机典型连接图。主机模式下用户需要分配一个GPIO来控制电源芯片的V<sub>BUS</sub>的输出。

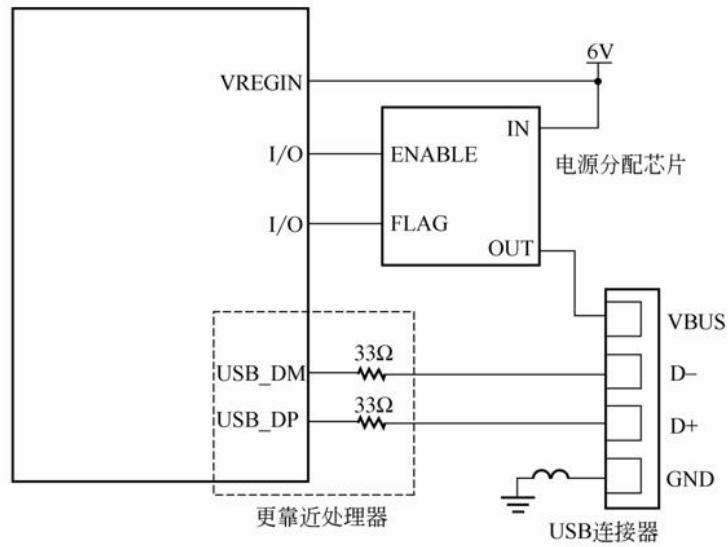


图3-16 KHCI主机典型连接图

图3-17是KHCI设备典型连接图。

### 3.2.2 控制器接口

KHCI控制器的编程接口是一套寄存器和一个共享内存区。软件通过特定的寄存器操作，完成对KHCI控制器的初始化和配置工作。

KHCI控制器和驱动程序通过一个缓冲区描述符表（ Buffer Descriptor Table , BDT ）来交换数据，而页面寄存器（ BDTPAGE ）用来指定缓冲区描述符表的起始地址。软件通过对缓冲区描述符表共享内存区和寄存器的操作，来驱动 KHCI 控制器完成 USB 的四种类型的数据传输。

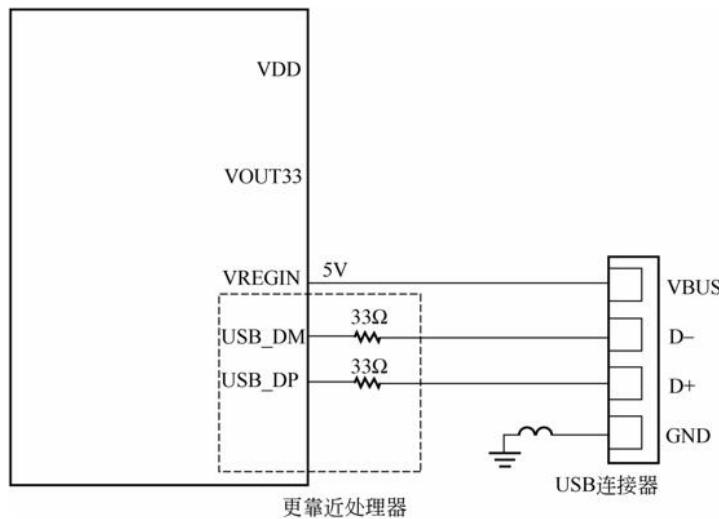


图3-17 KHCI设备典型连接图

#### 1. 缓冲区描述符表

为了有效地管理USB通信，KHCI控制器和软件驱动使用系统内存中的一块共享缓冲区来进行控制和信息交换。这块共享缓冲区就是缓冲区描述符表。缓冲区描述符表驻留在512字节对齐的系统内存中，其中含有32个缓冲区描述符（ Buffer Descriptor , BD ）。每个缓冲区描述符包含一个特定端点特定方向（ IN或者OUT ）上进行传输所需要的配置信息，实际上代表一个USB传输的事务（ Transaction ）。KHCI 控制器对每个端点特定方向的传输使用了双缓冲区机制（ 奇偶缓冲区描述符机制 ），允许KHCI控制器在处理一个缓冲区描述符时，微控制器可

以准备另外一个缓冲区描述符，使得USB数据传输的性能大为提高。由于每个端点索引可以有IN和OUT两个方向，而每个方向上又有2个缓冲区描述符，所以每个端点索引一共有4个缓冲区描述符。在KHCI控制器中最多允许16个端点索引，最多需要64个缓冲区描述符，而每个缓冲区描述符是8个字节，所以一般KHCI控制器的缓冲区描述符表就是512字节，并要求其地址为512字节对齐。

缓冲区描述符表在系统内存中分配，其地址并不固定，软件驱动程序在确定缓冲区描述符表的地址后需要更新KHCI控制器的BDTPAGE寄存器，使得KHCI控制器能够访问这块内存区域并进行USB通信传输。

每个缓冲区描述符中包含一个指向传输缓冲区的地址指针，由软件驱动程序在传输前进行配置，之后 KHCI 控制器从缓冲区描述符中取得并使用该传输地址进行实际的数据传输。KHCI控制器接口关系图如图3-18所示。

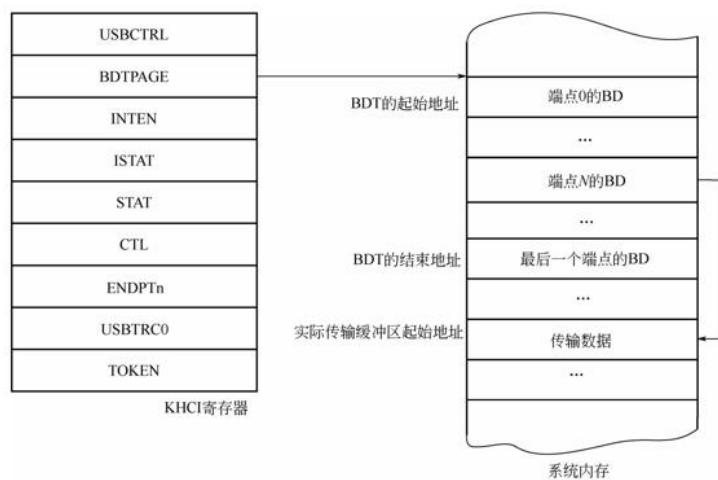


图3-18 KHCI控制器接口关系图

## 2. 缓冲区描述符

缓冲区描述符包含了在特定端点特定方向上进行传输所需要的所有的配置信息。低地址的4个字节指定其配置信息，高地址的4个字

节指定传输的缓冲区地址，该地址必须4字节对齐。缓冲区描述符如图3-19所示。

31:26	25:16	15:8	7	6	5	4	3	2	1	0
预留	BC (10位)	预留	OWN	DATA0/1	KEEP/ TOK_PID[3]	NINC/ TOK_PID[2]	DTS/ TOK_PID[1]	BDT_STALL/ TOK_PID[0]	0	0
Buffer Address (32位, 传输的缓冲区地址)										

图3-19 缓冲区描述符

由于缓冲区描述符表是一块系统共享内存，微控制器更新BDTPAGE寄存器，通知KHCI控制器缓冲区描述符表的首地址之后，微控制器和KHCI控制器都可以访问这一块内存。为了避免双方同时访问所引起的冲突，KHCI控制器定义了一套同步机制，其核心就是低4字节地址中的OWN位。

当缓冲区描述符OWN位为1时，表示KHCI控制器拥有该缓冲区描述符的访问权限，微控制器不得访问该缓冲区描述符除OWN位以外的其他信息。当缓冲区描述符OWN位为0时，表示微控制器拥有该缓冲区描述符的访问权限，KHCI控制器不得访问该缓冲区描述符除OWN位以外的其他信息。

OWN位的默认值为0，在传输开始前，微控制器拥有访问权限，所以可以对传输相关的位进行配置，如传输地址、传输长度、传输数据的PID等。当相关参数配置好后，将OWN位设为1，将缓冲区描述符的访问权限交给KHCI控制器。KHCI控制器使用软件驱动程序所配置的参数进行传输，在传输完成后将传输结果写回缓冲区描述符，将OWN位设为0，并触发中断通知传输已完成。最后软件访问缓冲区描述符得到传输结果，微控制器与KHCI控制器交互完成传输的过程结束。KHCI传输交互过程如图3-20所示。

需要注意的是，缓冲区描述符低32比特的配置信息中，第2位到第5位这4位字段是复用的。在传输前这4位用于配置一些传输相关信息，而在传输完成后，这4位用于保存USB主机发送过来的令牌包ID(PID)。同样，第16位到第25位的该字段也是同样的处理方式，

在传输前该字段表示驱动软件期望在本次传输中所传输的数据量大小，在传输完成后该字段表示本次传输实际完成的数据量大小。

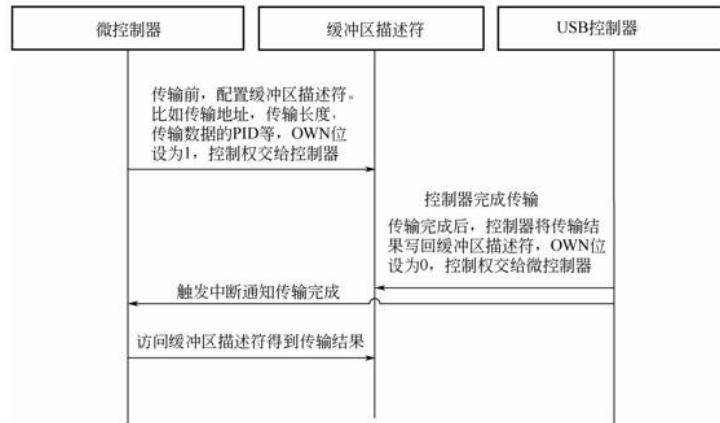


图3-20 KHCI传输交互过程

关于缓冲区描述符各字段的详细描述，请参考恩智浦公司MK65FN2M0 CAC18的芯片手册。

### 3. 定位缓冲区描述符

KHCI有16个端点，每个端点不同的方向上有4个缓冲区描述符，为了配置指定端点指定方向上的传输，首先需要寻址到正确的缓冲区描述符。缓冲区描述符地址计算表如3-21所示，23位的缓冲区描述符表的首地地址（BDTPAGE），加上4位的端点索引（Endpoint），1位的方向位（TX，0表示OUT方向，1表示IN方向），1位的奇偶位（ODD）组成了缓冲区描述符地址的高29位，缓冲区描述符地址的低3位为0（8字节对齐）。

这里需要特别指出是，驱动软件需要维护每个端点的每个方向上的ODD值，在驱动软件初始化时会把CTL[ODDRST]写1，把ODD的值清0，并且软件需要在每次传输完成后把ODD值反转，以保持和控制器同步。

31:24	23:16	15:9	8:5	4	3	2:0
BDT_PAGE_03	BDT_PAGE_02	BDT_PAGE_01[7:1]	Endpoint	TX	ODD	000

图3-21 缓冲区描述符地址计算表

## 4. 寄存器

由于KHCI控制器的寄存器较多，本节只列出本书中涉及的KHCI寄存器，如表 3-2 所示。读者若想了解 KHCI 控制器的其他寄存器，请参考恩智浦公司MK65FN2M0CAC18的芯片手册。

表3-2 KHCI寄存器

寄存器名称	位	位域	描述
USBCTRL	6	PDE	0—清除内部 15kΩ 下拉电阻 1—使能内部 15kΩ 下拉电阻
CONTROL	4	DPPULLUPNONOTG	0—清除内部 1.5kΩ 上拉电阻 1—使能内部 1.5kΩ 上拉电阻
OTGCTL	7	DPHIGH	0—OTG 模式清除内部 1.5kΩ 上拉电阻 1—OTG 模式使能内部 1.5kΩ 上拉电阻
CTL	0	USBENSOFEN	0—停止向总线发送帧开始 1—向总线发送帧开始
	2	RESUME	0—停止向总线发送恢复信号 1—向总线发送恢复信号
	3	HOSTMODEEN	0—控制器工作在设备模式 1—控制器工作在主机模式
	5	TXSUSPENDTOKENBUSY	在主机模式下，当 USB 模块忙于执行传输时，会设置该位。当该位置位时，驱动软件不能写更多的令牌 TOKEN 命令到令牌寄存器组。 在设备模式下，当控制器禁止数据包传输时，该位会被置上，清除该位允许控制器继续传输
	7	JSTATE	指示当前总线的状态
ADDR	6~0	ADDR	USB 设备的地址
TOKEN	3~0	TOKENENDPT	传输的端点地址。写入的四位值必须是有效的端点号
	7~4	TOKENPID	USB 模块执行的令牌类型。 0001 OUT 令牌 USB 模块执行 OUT (TX) 事务 1001 IN 令牌 USB 模块执行 In (RX) 事务 1101 SETUP 令牌 USB 模块执行 SETUP (TX) 事务

续表

寄存器名称	位	位域	描述
USBTRC0	5	USBRESMEN	0—清除异步恢复中断允许 1—使能异步恢复中断允许
INTEN	0	USBRSTEN	0—清除 USB 复位中断 1—使能 USB 复位中断
	5	RESUMEEN	0—清除 USB 恢复中断 1—使能 USB 恢复中断
	6	ATTACHEN	0—清除 USB 连接中断 1—使能 USB 连接中断

### 3.2.3 KHCI连接/断开检测

USB 的一个重要特性就是支持热插拔，本书第 2 章已经详细讨论了 USB 系统的连接和断开的各种情况，本章将讨论如何利用 KHCI 控制器来实现 USB 的连接和断开检测，以加深读者对 USB 总线连接和断开检测的理解。

#### 1. USB设备的连接和断开检测

##### 1 ) 设备连接检测过程

一般USB设备端通过监测 $V_{BUS}$ 的变化来判断是否连接到USB主机上，但是在恩智浦MK65FN2M0CAC18芯片上，KHCI控制器不支持检测 $V_{BUS}$ 的变化。所以在板级设计时需要将 $V_{REGIN}$ 和 $V_{BUS}$ 连在一起，使得 $V_{BUS}$ 与 $V_{REGIN}$ 同步变化，且芯片内部 D+的上拉电压由 $V_{REGIN}$ 提供。因为控制器不支持检测 $V_{BUS}$ 的变化，要求控制器驱动必须在设备连接到主机前先使能CONTROL [DPPULLUPNONOTG]，让D+上拉电阻预先连接。但由于芯片内部D+的上拉电压由 $V_{REGIN}$ 提供， $V_{BUS}$ 在连接到主机前是低电平， $V_{REGIN}$ 也是低电平，所以即使上拉电阻预先连接，也不会产生连接信号给主机。当 USB 设备连接到USB主机后， $V_{REGIN}$ 和 $V_{BUS}$ 同步变化，这时 $V_{BUS}$ 是高电平，D+上拉电压也是高电平，总线上产生了一个能被主机识别的设备连接信号。USB 主机一旦检测到有设备连接，将会发出一个复位 (RESET) 信号让设备进入初始状态，USB 设备就可以使用这个复位信号作为设备连接的检测机制。

##### 2 ) 设备断开检测过程

一般USB设备端也是通过监测 $V_{BUS}$ 的变化来判断是否从USB主机拔出。但是在恩智浦MK65FN2M0CAC18芯片上，KHCI控制器不支持检测 $V_{BUS}$ 的变化，所以 KHCI 控制器不能通过检测  $V_{BUS}$ 的变化来检

测设备断开。这里也和设备连接检测一样要求在板级设计时需要将  $V_{REGIN}$  和  $V_{BUS}$  连在一起，当 USB 设备从到 USB 主机拔出后， $V_{REGIN}$  和  $V_{BUS}$  同步变化，且芯片内部 D+ 的上拉电压由  $V_{REGIN}$  提供，这时  $V_{BUS}$  是低电平， $V_{REGIN}$  也是低电平，D+ 上拉电压也是低电平，D+ 上也是低电平。由 D+ 和 D- 上信号变化所会产生复位（RESET）信号。设备可以使用这个复位信号作为设备断开连接的检测机制。

图3-22是KHCI设备的连接和断开检测流程。

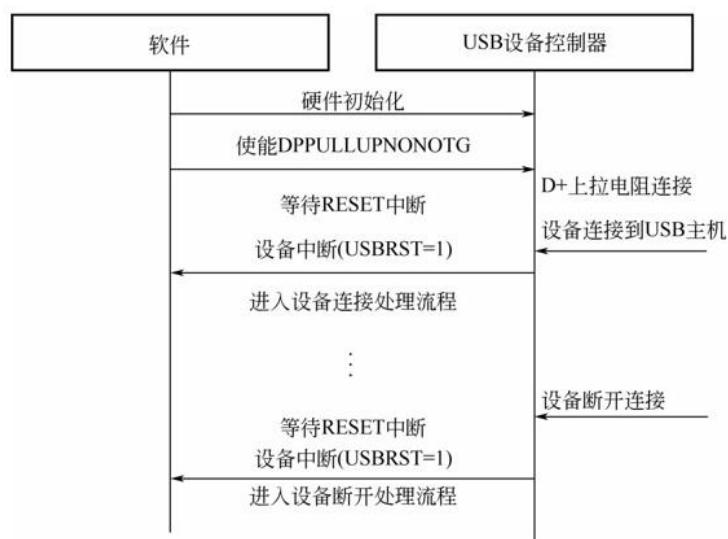


图3-22 KHCI设备的连接和断开检测流程

## 2. USB主机的连接和断开检测

本节将介绍基于KHCI如何实现主机的连接断开检测和处理。

### 1 ) KHCI主机连接检测

KHCI主机通过监测端口D+和D-的电压变化来检测设备的连接。为了监测D+和D-的变化，在USB主机端驱动软件需要在设备连接到主机前先使能INTEN[ATTACHEN]。当USB设备连接到USB主机后，D+和D-电压变化并且控制器寄存器 ISTAT[ATTACH]会变成1，表示设备连接上了USB主机，KHCI控制器会产生一个中断。KHCI控制器的设备驱动程序会在连接中断例程里处理设备的连接事件。

## 2 ) KHCI主机断开检测

KHCI主机并没有使用第1章提到的由D+和D-电压变化产生的断开连接信号来进行设备的断开连接检测，控制器也没有一个断开连接中断来指示USB设备断开连接，而是使用当设备从主机拔出来产生总线复位（RESET）信号来判断设备断开。为了监测D+和D-的变化产生的复位信号，在USB主机端驱动软件需要在连接时使能INTEN[USBRSTEN]。当USB设备从到USB主机拔出后，D+和D-电压发生变化，控制器会产生一个复位中断，并且控制器寄存器ISTAT[USBRST]会变成1，表示设备复位状态。KHCI控制器驱动在复位中断处理例程里进行一小段延时来判断USB的连接状态寄存器，如果此时KHCI控制器状态寄存器ISTAT[ATTACH]的值是1，指示设备是连接态，这时就当成总线复位来处理；如果ISTAT[ATTACH]的值是0，需要当成设备断开连接处理。

当USB主机收到一个设备断开连接的事件后，在断开连接事件的任务里，移除设备的连接信息，通知应用程序设备已经断开，最后重新使能设备连接检测中断，以便检测下一次设备的连接。图3-23是KHCI主机的连接和断开检测流程。

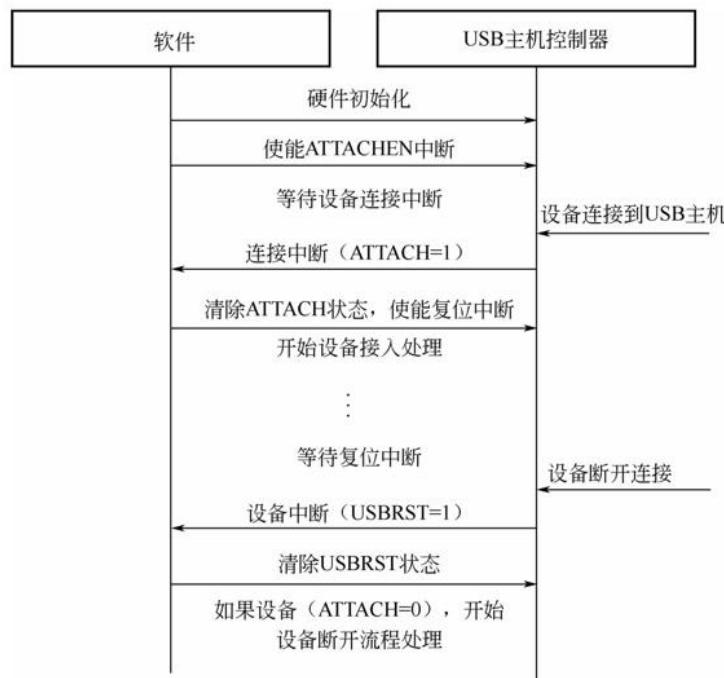


图3-23 KHCI主机的连接和断开检测流程

### **3.2.4 KHCI速度检测**

本节只讨论KHCI控制器的速度检测，分为USB设备的速度检测和USB主机的速度检测。

#### **1.KHCI设备的速度检测**

KHCI控制器当被用作设备时，只能作为全速（Full Speed）设备来使用，设备如果插入主机只能被识别成USB全速设备。

#### **2.KHCI主机的速度检测**

KHCI 主机使用 CTL[JSTATE]这个寄存器位域通过检测总线空闲状态来进行设备速度检测，这个寄存器的值会指示当前连接设备的速度。当检测到这个寄存器位域的值为 1 时，设备是一个全速设备；当检测到这个寄存器位域的值是0时，设备是一个低速设备。

### 3.2.5 KHCI传输实现

USB协议中定义了4种传输类型，每种传输类型有不同的特性，详细的细节请参看2.3节。KHCI控制器是一个相对比较简单的硬件控制器，在控制器内部并没有实现不同传输类型的这些特性，只是提供了一个简单的机制来实现一个通用传输模型，所有的4种传输都可以用这个模型来实现，而传输类型的不同特性就交由软件来实现。在3.2.2节中已经介绍过这个通用的传输模型，本节将介绍如何利用这个传输模型实现4种不同的传输类型。

#### 1. 控制传输

控制传输分为3个阶段：设置阶段、数据阶段和状态阶段，在实现上分别由3个事务来完成。

##### 1) Setup事务设置阶段

- 寻址端点缓冲区描述符

利用3.2.2节中图3-21所示的缓冲区描述符计算表时，由于控制传输都是在端点0上进行传输，其Endpoint为0；对于Setup事务来说，方向位TX为0；ODD为当前驱动维护的端点0的接收方向上的ODD值，这时即可计算出该事务的缓冲区描述符的地址。

- 配置缓冲区描述符

获得缓冲区描述符的地址之后，就可以对该缓冲区描述符进行配置，以完成相应的传输。缓冲区描述符项的缓冲区地址（Buffer Address）需要配置为接收的Setup包内存地址。BC位需要配置为8（Setup包的固定长度）。DATA0位需要配置成0。完成配置后将OWN位配置为1，使得USB控制器获得该缓冲区描述符的读写权限。最后清除控制器的CTL[TXSUSPENDTOKENBUSY]位，来通知USB控制器进行传输。

- 传输完成

当传输完成后，USB控制器会产生一个中断，微控制器收到中断，并检测ISTAT[TOKDNE]为1，确认这是一个传输完成中断；在继续后续处理前，需要清除ISTAT[TOKDNE]位（该位为写1清除位，需要对该位写入1清除该位）；之后，读取STAT寄存器得到传输完成的端点索引、方向及ODD值，计算出该 Setup 事务缓冲区描述符的地址；获得缓冲区描述符的地址后即可读取传输完成时的相关状态，如BC 位为 8，表明接收到 8 个字节的数据，TOK\_PID[n]为0xD，表明这是一个Setup包；并根据缓冲区描述符的包地址缓冲地址（Buffer Address）获取Setup包的内容。最后需要把ODD的值反转，以保持和控制器同步。至此，Setup事务完成。

## 2 ) IN或者OUT事务数据阶段

根据Setup包中请求的不同，可能需要进行0个或者多个IN/OUT的事务来完成控制传输的数据阶段。

- 寻址端点缓冲区描述符

与 Setup 事务处理相同，只是方向位需要根据实际传输的方向来确定。如果是IN事务则TX为1，OUT事务则TX为0。

- 配置缓冲区描述符

与Setup事务处理相同，只是BC位设置为实际传输的长度，DATA0位需要翻转，在有多个IN或者OUT事务的情况下，DATA0每次传输都需要翻转。

- 传输完成处理

与Setup事务处理相同。

## 3 ) IN或者OUT事务状态阶段

N或者OUT事务状态阶段和数据阶段的事务处理方式基本相同，区别在于：状态阶段数据传输的长度固定为0，发送的数据包的DATA0位固定为1。

## 2.批量传输

批量传输和控制传输的数据阶段使用的寻址端点缓冲区描述符、配置缓冲区描述符和传输完成处理的方法类似，在此不再赘述。

## 3.中断传输

中断传输和控制传输的数据阶段使用的寻址端点缓冲区描述符、配置缓冲区描述符和传输完成处理的方法类似。只是在KHCI控制器主机模式下，中断传输的周期性需要由微控制器来保证，即当收到设备端回复的NAK包后，微控制器需要在中断传输的传输周期达到之后，重新配置新的缓冲区描述符。

## 4.同步传输

同步传输和控制传输的数据阶段使用的寻址端点缓冲区描述符、配置缓冲区描述符和传输完成处理的方法类似。只是主机进行同步传输时需要把ENDPTn [EPHSHK]位域设成0，因为同步传输不需要握手信号。

## 5.注意事项

- 传输前需要配置ENDPTn[EPRXEN]或者ENDPTn[EPTXEN]使能端点的收发功能。
- KHCI 工作在设备模式时，配置好缓冲区描述符后通过操作CTL [TXSUSPENDTOKENBUSY]寄存器来使能USB控制器传输；而工作在主机模式时，是通过操作 TOKEN寄存器来使能 USB控制器传输。

- KHCI工作在主机模式时，同一时间只能处理一个事务，因此只需要使用第一个端点的4个缓冲区描述符，同时也只会使能第一个端点的ENDPTn寄存器。

### 3.2.6 KHCI枚举的软件实现

USB的枚举实际上就是一系列控制传输的组合，使得USB主机能够获取USB设备的描述符，并对USB功能进行配置。对控制传输的介绍请参看3.2.5节，本节不再赘述。本节只介绍在使用KHCI控制器时在枚举过程中需要注意的两点。

当USB设备收到USB复位事件后，系统软件需要将USB设备状态机设置为默认状态，并且初始化控制传输管道，准备接收USB主机发送的Setup请求。正如在第2章中所讨论的，Setup包不同于数据包，USB设备收到后不会回复NAK包。如果KHCI控制器没有及时准备好用于接收Setup包的缓冲区描述符，它将无法对USB主机发送的Setup包做出任何回复，如果连续多次均没有回复Setup包的话，会导致整个传输失败。

对于设置地址请求（SetAddress（）），USB主机在完成该请求（即状态阶段传输成功）后，需要等待2ms以保证USB设备能够使用新的地址进行工作。USB设备在完成该请求后，需要立即设置ADDR[ADDR]字段使得新地址能尽快生效。因为USB协议规定，如果设备完成了设置地址，主机会在下一个包使用新的地址。设备如果没有及时更新地址，会导致传输失败。

具体的KHCI枚举流程如图3-24所示。

### 3.2.7 KHCI挂起和恢复

第2章讨论了USB的挂起和恢复，本节将介绍如何使用KHCI实现挂起和恢复。



图3-24 KHCI枚举流程

#### 1.挂起

一般来说，系统软件为了降低系统功耗会将整个USB总线或者部分总线挂起。本节只会涉及控制器的挂起和唤醒，只讨论全局总线挂起情况，集线器的部分总线挂起不在本节讨论。在总线挂起后，主机驱动程序会使能USB控制器的异步唤醒功能（如检测USB设备移除、USB恢复信号等）。在设置USB控制器能够被异步唤醒源唤醒后，系统进入低功耗模式。

当需要挂起总线时，主机驱动程序在停止传输后通过清除寄存器CTL [USBENSOFEN]停止向总线发送帧开始（Start of Frame，SOF）包，设备会在没有收到帧开始包的3ms后进入挂起状态。在最少等待3ms后，主机设置寄存器INTEN [RESUMEEN]和USBTRC0[USBRESMEN]来打开USB的异步唤醒源功能。最后，系统进入低功耗状态。具体的KHCI主机挂起总线流程如图3-25所示。

对于USB设备的挂起流程，在检测到总线挂起事件后，USB设备控制器的流程和USB主机类似，在此不再赘述。

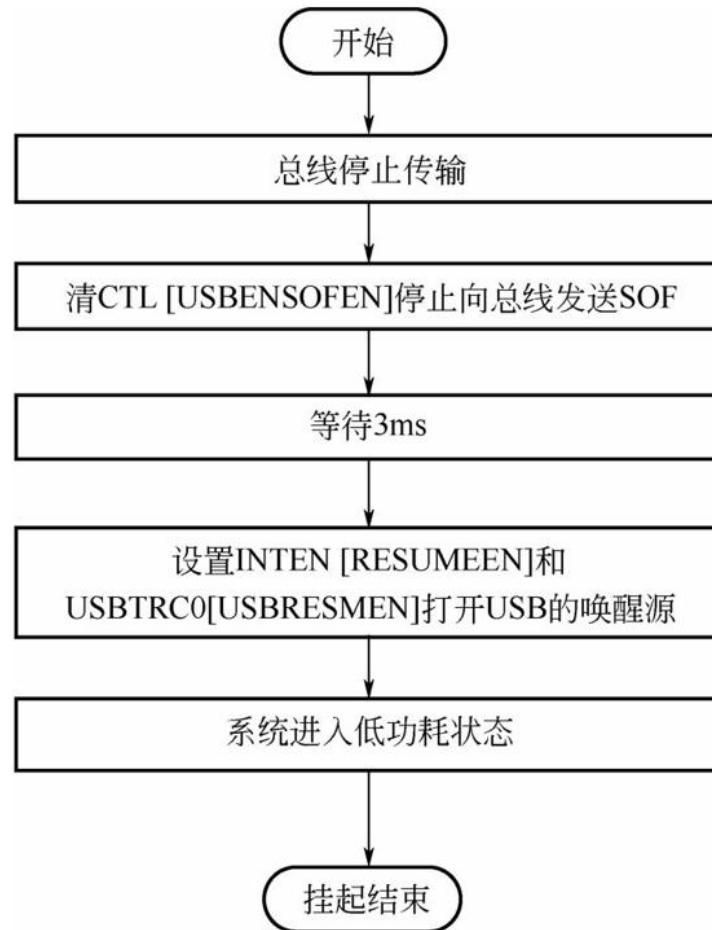


图3-25 KHCI主机挂起总线流程

## 2. 恢复

根据恢复信号发起源的不同，恢复信号可分为USB主机发起的恢复信号和USB设备远程唤醒的恢复信号，本节将从这两个角度分别介绍恢复信号。

### 1) USB主机发起的恢复信号

USB主机的系统软件需要唤醒USB总线时，主机控制器程序通过配置USB主机控制器产生恢复信号。在总线停止传输后，主机控制程序首先清除INTEN [RESUMEEN]和USBTRC0[USBRESMEN]位，以清

除USB的唤醒源，然后主机系统软件设置CTL [RESUME]控制USB主机控制器产生恢复信号。如果 USB 设备还处于连接状态，主机在等待 20ms 后，会清除寄存器CTL [RESUME]并设置寄存器CTL [USBENSOFEN]，以使USB 主机控制器回到正常工作状态。具体的 KHCI 主机恢复总线流程如图3-26所示。

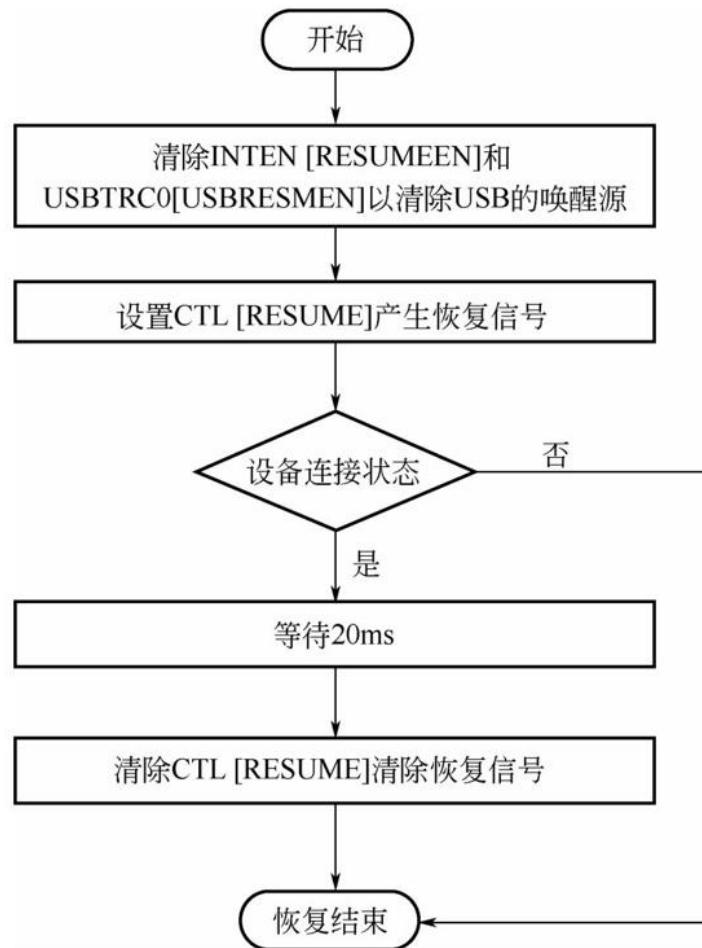


图3-26 KHCI主机恢复总线流程

## 2 ) USB设备发起的恢复信号

当USB设备的系统软件需要远程唤醒USB总线时，可以配置USB设备控制器产生恢复信号。USB设备被自身的唤醒按键唤醒后，设备先是设置CTL [RESUME]控制USB控制器产生恢复信号，在持续时间不小于10ms后，设备驱动程序再清除CTL [RESUME]位完成设备发起的恢复。

### 3.2.8 KHCI端点停止处理

端点停止（Endpoint Stall）主要分为两种：功能停止（Function Stall）和协议停止（Protocol Stall）。功能停止可以通过SetFeature（endpoint\_halt）或者ClearFeature（endpoint\_halt）来设置或者清除端点的停止状态。而协议停止一般出现在控制传输的数据阶段，当USB设备不支持USB主机的协议请求时，USB设备通过控制端点返回停止状态。

USB主机控制器可以通过读取传输完成后缓冲区描述符的TOK\_PID[n]位域来获取当前对应的USB设备的端点是否处于停止状态，如果TOK\_PID[n]为0xE，则对应的设备的端点处于停止状态。

如果USB设备需要返回停止状态或者收到了SetFeature（endpoint\_halt）请求，设备驱动软件会将当前端点的缓冲区描述符列表的缓冲区描述符第二位DT\_STALL位设置为1，这时端点会返回停止状态。但是对于控制端点，USB设备驱动软件需要将控制端点的BDT的第二位DT\_STALL位清除，以准备好接收下一次的Setup传输。如果是非控制端点，则需要主机发送ClearFeature（endpoint\_halt）请求来清除端点的停止状态。当一个端点进入停止状态时，微控制器会收到一个STALL中断。驱动软件会根据是否是控制端点0而采用不同处理流程。图3-27是KHCI控制器端点停止的中断处理流程。

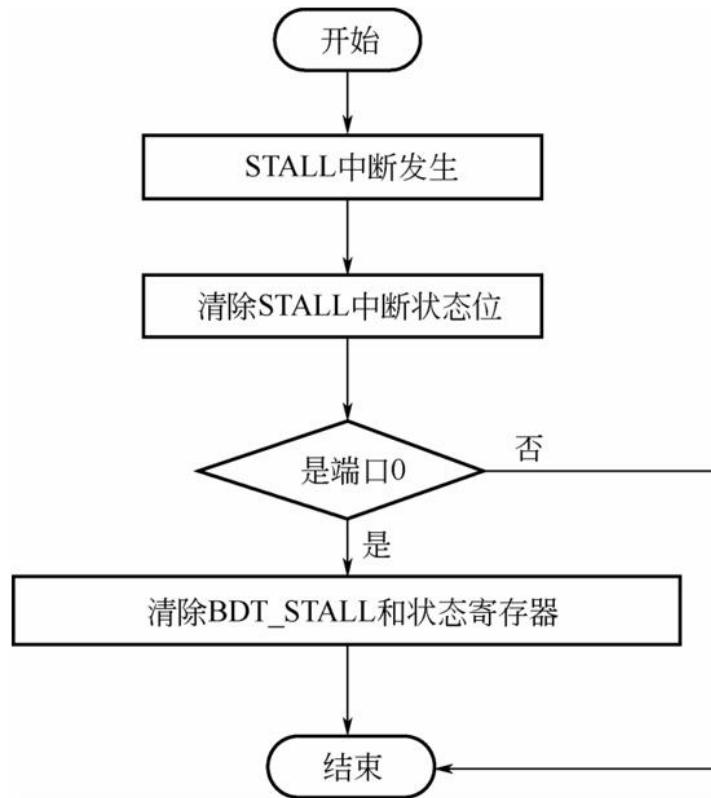


图3-27 KHCI控制器端点停止的中断处理流程

## 第4章

# USB音频演示程序

本书的前几章分别从USB信号、协议和硬件控制器的角度着重讲解了连接检测、速度检测、枚举、挂起和恢复等USB特性。本章将结合USB音频类的知识从源代码的角度来介绍如何实现这些特性。本章中的源代码来自恩智浦的 MCUXpresso 中 USB 扬声器音频演示程序，所以也会对其协议栈框架做一个简单的介绍，以便让读者更加容易理解USB这些特性的实现。

## 4.1 USB音频

USB音频的功能由功能实体组合得来，下面结合USB音频功能的拓扑结构来解析它的组成和音频流向。

## 4.1.1 USB音频功能实现拓扑

USB 音频功能一般由两种实体组成，分别是单元（ Units ）和终端（ Terminals ）。

以一个包含音频输入和输出的设备为例，音频设备结构拓扑如图 4-1 所示，它包含了单元、终端及音频流的流向。

单元实现了USB音频的基本功能，音频设备的整体功能就是由一个个单元连接来实现的。一般常用的单元有：特征单元（ Feature Unit , FU ）、混合单元、选择单元、处理单元和扩展单元等。以特征单元为例，它内部包含了不同的功能，如音量控制、静音、声道平衡，等等。

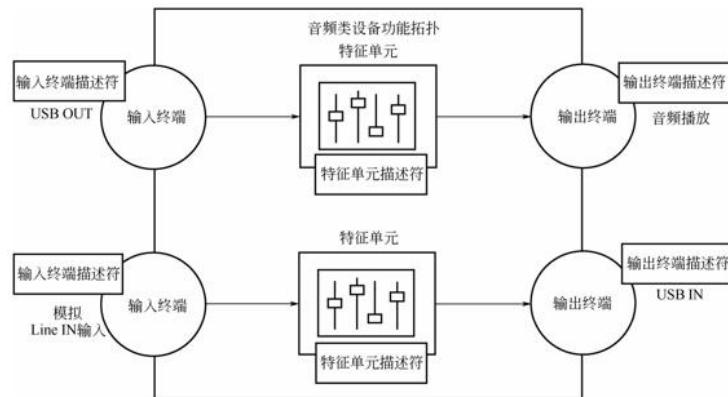


图4-1 音频设备结构拓扑

终端分两种，分别是输入终端（ Input Terminal , IT ）和输出终端（ Output Terminal , OT ）。输入终端代表音频功能内的音频通道的起点，而输出终端代表音频通道的结束点。从音频功能的角度来看，USB 端点（ Endpoint ）可以代表输入或输出终端作为音频功能的输入或输出。它作为输入终端提供音频数据流，作为输出终端消耗来自音频功能的数据流。当然，其他的外设也可以作为音频功能的输入或者输出终端，如线路输入（ Line-In ）或者线路输出（ Line-Out ）。当 USB 端点作为音频功能的输入或者输出终端时，主机必须通过 USB 音

频类配置描述符，获得该类的输入或者输出终端描述符以及与终端描述符对应的端点描述符，这样它才可以知悉该终端的特性和功能。和控制相关的功能参数存储在输入或者输出终端描述符中，和音频流相关的参数存储在端点描述符中。

## 4.1.2 音频控制接口和描述符

音频控制接口是USB设备提供给主机的一组接口，让主机可以控制音频设备内部的单元和终端的功能，以实现整体的音频功能。USB主机使用端点0向音频设备发送音频类控制请求来读取或修改音频设备的各个单元或终端的数据和状态。同时，音频控制接口可以包括一个可选中断传输类型的端点，主要用于USB设备向USB主机返回音频设备的状态。

USB音频设备规范定义了多种描述符用来表示控制接口中的各个模块，如音频类输入终端描述符、音频类输出终端描述符等。本节将着重介绍几种常用的音频类规范定义的描述符类型，包括标准接口描述符、音频类头接口描述符（Class-Specific AudioControl Interface Header Descriptor）、音频类输入终端描述符（Input Terminal Descriptor，ITD）、音频类特征单元描述符（Feature Unit Descriptor，FUD）和音频类输出终端描述符（Output Terminal Descriptor，OTD）。更多的描述符请参见USB音频设备规范。

音频控制接口描述符包括标准音频控制接口描述符（Standard AudioControl Interface Descriptor）和音频类控制接口描述符（Class-Specific AudioControl Interface Descriptor）两类。前者反映该音频接口的通用属性，后者反映该音频接口的特殊功能。音频类控制接口描述符实际上是音频类头接口描述符、输入终端描述符、输出终端描述符、特征单元描述符和其他单元描述符组合在一起的总称，没有具体的定义。下面一一介绍各描述符。

### 1. 标准音频控制接口描述符

表4-1给出了标准音频控制接口描述符。

表4-1 标准音频控制接口描述符

字 段	长 度 (字 节)	字 段 描 述
bLength	1	描述符的长度，标准音频控制接口描述符是 9 字节
bDescriptorType	1	描述符类型，对于标准音频控制接口必须是 0x04
bInterfaceNumber	1	接口号，编号从该接口所属配置的接口 0 开始递增
bAlternateSetting	1	该接口的可替换设置号
bNumEndpoints	1	除了端点 0 以外，该接口所使用的端点数目

续表

字 段	长 度 (字 节)	字 段 描 述
bInterfaceClass	1	音频接口所属类
bInterfaceSubClass	1	音频控制接口所属子类，必须是 0x01
bInterfaceProtocol	1	未被使用，设置成 0
iInterface	1	该音频控制接口的字符串描述符的编号

描述符的bInterfaceClass、bInterfaceSubClass和bInterfaceProtocol的组合表示当前描述符是音频接口描述符。bNumEndpoints只能为0或1，如果该接口使用中断端点返回设备状态，则bNumEndpoints为1，否则为0。

## 2. 音频类头接口描述符

音频类头接口描述符是音频类控制接口描述符的开始，它包含了该设备的音频规范标准、音频类控制接口的总长度以及和该控制接口相关的音频流接口等信息。音频类头接口描述符如表4-2所示。

表4-2 音频类头接口描述符

字 段	长 度 (字 节)	字 段 描 述
bLength	1	长度是 $(8+n)$ 字节， $n$ 为该控制接口包含的音频流接口数目
bDescriptorType	1	描述符类型，音频类接口必须是 0x24
bDescriptorSubType	1	描述符子类型，属于音频控制头子类，其值为 0x01
bcdADC	2	该设备所属的音频标准规范
wTotalLength	2	所有的音频类控制接口描述符的长度总和
bInCollection	1	该音频控制接口所拥有的音频流接口集合，总数为 $n$
baInterfaceNr (1)	1	所拥有的第一个音频流接口号
...	...	...
baInterfaceNr ( $n$ )	1	所拥有的最后一个音频流接口号

该描述符wTotalLength字段，是所有的音频类控制接口描述符的总长度，需要包括音频类头接口，输入、输出终端，各种单元描述符

的总长。bInCollection代表音频控制接口中所包含的音频流接口的总数，而baInterfaceNr字段需要一一列举 bInCollection 集合中音频流接口号。例如，该音频流接口中既有麦克风又有扬声器接口，则bInCollection为 2，如麦克风为第一个音频流接口，接口号为1，扬声器为第二个音频流接口，接口号为2，则baInterfaceNr (1) 为1，baInterfaceNr (2) 为2。

### 3. 音频类输入终端描述符

音频类输入终端描述符主要用于向USB主机提供有关输入终端功能的相关信息。音频类输入终端描述符如表4-3所示。

表4-3 音频类输入终端描述符

字 段	长 度 (字 节)	字 段 描 述
bLength	1	描述符的长度，音频类输入终端描述符长度是 12 字节
bDescriptorType	1	描述符类型，音频类接口必须是 0x24
bDescriptorSubType	1	描述符子类型，属于输入终端描述符子类，其值为 0x02
bTerminalID	1	该输入终端的 bTerminalID 号
wTerminalType	2	该输入终端的类型，详见表 4-4
bAssocTerminal	1	该输入终端的关联终端
bNrChannels	1	该输入终端所包含的输出逻辑通道数目
wChannelConfig	2	该终端所支持的逻辑通道的位图
iChannelNames	1	通道名称的字符串描述符的编号
iTerminal	1	该输入终端的字符串描述符的编号

音频类输入终端由描述符中的bTerminalID唯一标识，bTerminalID由用户自行定义。音频类控制接口的同一替换设置中不应有任何其他终端或单元用相同的bTerminalID。USB主机在请求中可以通过该值访问到该终端。

wTerminalType用于表示输入终端所代表的物理实体类型，该类型可以是USB OUT端点或者麦克风。表4-4列出了USB终端类型，该类型既可以作为输入，也可以作为输出。

表4-4 USB终端类型

终端类型	取 值	字段描述
USB Undefined	0x0100	未定义的 USB 终端类型
USB streaming	0x0101	USB 流终端
USB vendor specific	0x01FF	厂商自定义终端类型

表4-5列出了部分输入终端类型。

表4-5 部分输入终端类型

终端类型	取 值	字段描述
Input Undefined	0x0200	未定义的输入终端类型
Microphone	0x0201	一般麦克风类型
Desktop microphone	0x0202	桌面麦克风类型

例如，USB扬声器设备输入终端的wTerminalType一般是USB streaming（0x0101），而USB麦克风设备输入终端的wTerminalType是Microphone（0x0201）。

bNrChannels代表该输入终端的通道数。逻辑通道就是通常所谓的声道，是共有相同的采样率、位分辨率（bBitsolution）等一系列属性的音频流传输通道。通道0代表主通道，非0通道代表逻辑通道。在主通道上对静音、音量等音频设置的调节可应用于所有的逻辑通道，而对逻辑通道的设置只能应用于该逻辑通道本身。

wChannelConfig代表该音频设备支持的逻辑通道位图。根据逻辑通道被用户听到时的不同空间分布，一共有12种逻辑通道，用2字节表示，有左前通道（L）、右前通道（R）、左环绕通道（Ls）、右环绕通道（Rs）等，在此不再赘述。

#### 4.音频类输出终端描述符

音频类输出终端描述符主要用于向USB主机提供有关输出终端功能的相关信息。音频类输出终端描述符如表4-6所示。

表4-6 音频类输出终端描述符

字 段	长 度（字节）	字 段 描 述
bLength	1	描述符的长度，音频类输出终端描述符长度是 9 字节
bDescriptorType	1	描述符类型，音频类接口必须是 0x24
bDescriptorSubType	1	描述符子类型，属于输出终端描述符子类，其值为 0x03
bTerminalID	1	该输出终端的 bTerminalID 号
wTerminalType	2	该输出终端的类型，详见表 4-7
bAssocTerminal	1	该输入终端的关联终端
bSourceID	1	该输出终端前序连接其他哪个终端或单元
iTerminal	1	该输出终端的字符串描述符的编号

音频类输出终端由描述符中的字段 bTerminalID 唯一标识，bTerminalID 由用户自行定义。与音频类输入终端描述符中的字段 bTerminalID 类似，音频控制接口的同一替换设置中不应有任何其他终端或单元用相同的 bTerminalID。

wTerminalType 用于表示输出终端所代表的物理实体类型，该类型可以是 USB IN 端点或者扬声器。

表4-7列出了部分输出终端类型。

表4-7 部分输出终端类型

终 端 类 型	取 值	描 述
Output Undefined	0x0300	未定义的输出终端类型
Speakers	0x0301	一般扬声器类型
Headphones	0x0202	耳机类型

bSourceID 代表输出终端前序连接的实体（终端或单元）。如果它的前序连接实体是特征单元，就把特征单元的 bUnitID 作为该输出终端的 bSourceID。如果它的前序连接实体是终端，就把终端的 bTerminalID 作为该输出终端的 bSourceID。

## 5. 音频类特征单元描述符

音频类特征单元描述符描述了该特征单元前序连接的输入终端上每个通道可被配置的功能列表。

音频类特征单元描述符如表4-8所示。

表4-8 音频类特征单元描述符

字段	长度(字节)	字段描述
bLength	1	描述符长度, 7+终端上通道数×n 字节
bDescriptorType	1	描述符类型, 音频类特征单元必须是 0x24
bDescriptorSubType	1	描述符子类型, 属于特征单元描述符子类, 其值为 0x06
bUnitID	1	该特征单元的 bUnitID 号
bSourceID	1	该特征单元前序连接的实体
bControlSize	1	通道功能特征位图的大小: n
bmaControls (0)	n	通道 0 的功能特征位图描述
bmaControls (1)	n	通道 1 的功能特征位图描述
.....	.....	.....
bmaControls (ch)	n	通道 ch 的功能特征位图描述
iFeature	1	该特征单元的字符串描述符的编号

特征单元由描述符中的字段bUnitID唯一标识，bTerminalID由用户自行定义。与音频类输入终端描述符中的字段bTerminalID类似，音频控制接口的同一替换设置中不应有任何其他终端或单元用相同的bUnitID。

bSourceID与音频输出终端的bSourceID定义相同，不再赘述。

bControlSize是bmaControls的长度。bmaControls代表该通道的功能特征位图，每位都代表了一个功能。表4-9是通道的功能特征位图。

表4-9 通道的功能特征位图

位	位描述
0	静音
1	音量
2	低音
3	中音
.....	.....
10... (8×n-1)	保留

如某位的值为1代表支持该功能的控制。由于在规范中规定了10种功能，所以n最大值为2。

### 4.1.3 音频流接口和描述符

音频流接口主要用于主机和设备之间交换音频流数据，可以同时包含一个用于传输音频流的音频传输端点和一个用户传输同步信息的音频同步端点。一般情况下，为了支持多种音频流格式、采样率、传输带宽和其他属性，一个 USB 设备的音频流接口描述符都包含多个接口可替换设置（Alternate Setting）。

例如，有一个音频设备，其音频控制接口是用来控制音量，音频流接口是用作音频流数据传输的。音频流接口可以有多种可替换设置：

- 可替换设置0：单声道模式，16kHz采样率，24位量化。在该设置的音频类数据格式描述符表明该格式，在音频流端点上传输对应格式单通道的音频数据。
- 可替换设置1：立体声双声道模式，48kHz采样率，16位量化。在该设置的音频类数据格式描述符表明该格式，在音频流端点上传输对应格式双通道的音频数据。

音频流接口描述符包括标准音频流接口描述符（Standard AudioStreaming Interface Descriptor）和音频类音频流接口描述符（Class-Specific Audio Streaming Interface Descriptor）。

#### 1. 标准音频流接口描述符

标准音频流接口描述符如表4-10所示。

表4-10 标准音频流接口描述符

字 段	长 度(字节)	字 段 描 述
bLength	1	描述符的长度，标准音频流接口描述符长度是 9 字节
bDescriptorType	1	描述符类型，标准音频流接口必须是 0x04
bInterfaceNumber	1	接口号，编号从 0 开始递增
bAlternateSetting	1	该接口的可替换设置号
bNumEndpoints	1	除了端点 0 以外，该接口所使用的端点数目（除反馈端点外，最多只能有一个传输端点）
bInterfaceClass	1	音频接口所属类
bInterfaceSubClass	1	音频流接口所属于类，必须是 0x02
bInterfaceProtocol	1	未被使用，一般设为 0
ilInterface	1	该音频流接口的字符串描述符的编号

标准音频流接口描述符和标准音频控制接口描述符一样，不再赘述。

## 2. 音频类音频流接口描述符

音频类音频流接口描述符里面包含了该音频流数据格式，以及该音频流接口和哪个终端相连等。

音频类音频流接口描述符如表4-11所示。

表4-11 音频类音频流接口描述符

字 段	长 度(字节)	字 段 描 述
bLength	1	长度是 7 字节
bDescriptorType	1	描述符类型，音频类接口必须是 0x24

续表

字 段	长 度(字节)	字 段 描 述
bDescriptorSubType	1	描述符子类型，属于 AS_GENERAL 子类描述符，其值为 0x01
bTerminalLink	1	如果该音频流接口有音频传输端点，则 bTerminalLink 代表该端点所对应的终端号 (bTerminalID)
bDelay	1	数据通路上的延时
wFormatTag	2	音频流编码格式

如果该音频流接口有音频流传输端点，则 bTerminalLink 代表该端点所对应的终端号 (bTerminalID)。

wFormatTag代表该接口中传输音频流的编码格式，音频流编码格式如表4-12所示。

表4-12 音频流编码格式

数据格式	值
TYPE_I_UNDEFINED	0x0000
PCM	0x0001
PCM8	0x0002
IEEE_FLOAT	0x0003
ALAW	0x0004
MULAW	0x0005

### 3.音频类数据格式描述符

音频类数据格式描述符 ( Class-Specific AudioStreaming Format Type Descriptor ) 主要描述音频流的通道数、位分辨率、采样率等信息。

音频类数据格式描述符如表4-13所示。

表4-13 音频类数据格式描述符

字 段	长 度(字节)	字 段 描 述
bLength	1	长度是 $(8 + 3 \times n)$ 字节
bDescriptorType	1	描述符类型，音频类数据格式必须是 0x24

续表

字 段	长 度(字节)	字 段 描 述
bDescriptorSubType	1	描述符子类型，属于数据格式子类描述符，其值为 0x02
bFormatType	1	音频流格式类型
bNrChannels	1	该接口所拥有的物理通道个数
bSubframeSize	1	该接口的子帧中有几个字节的音频数据
bBitResolution	1	该接口的子帧中有几位的音频数据是有效的
bSamFreqType	1	音频流接口的同步传输端点支持几种频率类型
tSamFreq[1]	3	同步传输端点所支持的第 1 种采样率
.....	.....	.....
tSamFreq[n]	3	同步传输端点所支持的第 n 采样率

bFormatType 为音频流数据格式类型编码，可选值如表4-14所示。

表4-14 音频流格式类型编码

格式类型	值
FORMAT_TYPE_UNDEFINED	0x00
FORMAT_TYPE_I	0x01
FORMAT_TYPE_II	0x02
FORMAT_TYPE_III	0x03

音频流格式一共有3种，这里只介绍Type I。当音频流按照物理时序采样时使用 Type I，每一个采样点由一个数据表示，而最终的音频信号是将这些连续的采样数据进行DA转换后得到的波形。典型的Type I信号就是标准的PCM码。

音频子帧（ Audio Subframe ）是传输音频数据的最基本单位，一个音频子帧就是一个采样数据，其大小一般是1、2、3或者4个字节，用bSubframeSize表示。一个音频子帧以二进制数表示时，二进制的位数就是bBitResolution。需要注意的是，bBitResolution 和 bSubframeSize 存在一定的限制关系：  
 $bBitResolution \leq bSubframeSize \times 8$ 。

bSamFreqType 代表该音频流接口的 ISO 端点支持几种频率，根据频率数目，可以得到tSamFreq的布局。

#### 4. 标准音频传输端点描述符和标准音频同步端点描述符

标准音频传输端点描述符（ Standard AudioStreaming Isochronous Audio Data Endpoint Descriptor ）和标准音频同步端点描述符是一样的，与标准端点描述符并无太大区别，如表4-15所示。

表4-15 标准音频传输端点描述符

字 段	长 度 (字 节)	字 段 描 述
bLength	1	端点描述符长度是 9 字节
bDescriptorType	1	端点描述符类型，必须是 0x05
bEndpointAddress	1	该字段表征 USB 端点的地址，具体位描述如下： 第 7 位表示该端点的方向， 0 代表 OUT 端点，1 代表 IN 端点 第 4 到 6 位保留 第 0 到 3 位代表端点号
bmAttributes	1	该字段表征 USB 端点的属性，具体位描述如下： 第 2 到 3 位代表同步的方式。 2'b00 代表没有进行同步 2'b01 代表用异步方式进行同步信息 2'b10 代表用自适应方式进行同步信息 2'b11 代表用同步方式进行数据同步 第 0 到 1 位代表传输类型， 2'b01 代表同步传输端点
wMaxPacketSize	2	该端点发送/接收的包的最大字节数
bInterval	1	该端点的传输数据间隔
bRefresh	1	主机对同步端点的轮询频率，传输端点必须设成 0
bSynchAddress	1	该端点如果有同步端点，对应的同步端点号

需要注意的是，bmAttributes要反映音频传输端点的同步类型和用法。使用异步方式进行同步的标准传输端点，其bmAttributes为0x5；用同步调节时钟方式进行同步的标准传输端点，其bmAttributes为0xD；标准音频同步端点的bmAttributes为0x1。对于标准传输端点，wMaxPacketSize 代表端点在该配置下可以传输的最大包长度，对于标准同步端点，wMaxPacketSize 固定为 3。bInterval代表传输间隔为多少个帧/微帧，计算公式为  $2^{bInterval-1}$ ，bInterval为 1 则代表间隔 1 帧。对于标准传输端点，bRefresh只能取零值，对于同步端点，bRefresh取值为1 ~ 9。对于标准传输端点，bSynchAddress 代表其所用的同步端点号，如果不具备同步端点则该字段为0。对于同步端点，bSynchAddress 为0。

## 5. 音频类传输端点描述符

音频类传输端点描述符（Class-Specific AudioStreaming Isochronous Audio Data Endpoint Descriptor）描述了标准音频传输端点的属性，如表4-16所示。

表4-16 音频类传输端点描述符

字 段	长 度 (字 节)	字 段 描 述
bLength	1	端点描述符长度是 9 字节
bDescriptorType	1	描述符类型，音频类传输端点必须是 0x25
bDescriptorSubType	1	描述符子类型，属于音频类端点子类描述符，其值为 0x01
bmAttributes	1	该端点支持哪些控制操作
bLockDelayUnits	1	wLockDelay 的单位
wLockDelay	2	端点稳定产生/消耗音频数据前需等待时间，单位为 bLock DelayUnits

bmAttributes代表该端点支持哪些端点方面的操作。第0位代表是否支持采样率调整，第1位代表是否支持音高调整，第7位代表该端点是否只支持wMaxPacketSize的传输。bLockDelayUnits和wLockDelay代表该USB设备在主机设置后需要多久能达到时钟稳定，以固定速率产生/消耗音频数据，这里不做过多讲解。

## 4.1.4 USB音频描述符实例

由于USB音频相关的描述符比较繁复，本节提供一组音频相关的描述符供读者参考。以48kHz采样，左右双声道，声道支持静音和音量调节操作，音频数据为PCM格式，16位量化，音频数据流通过反馈端点进行同步的扬声器为例，各描述符实例如下。

### 1. 音频类头接口描述符

音频类头接口描述符中bInCollection为0x01，baInterfaceNr为0x01，表明该控制接口只有一个对应的音频流接口，且该音频流接口号为1。

---

```
/* Class-Specific AudioControl Interface Header Descriptor */

0x09,          /* bLength */

0x24,          /* bDescriptorType */

0x01,          /* bDescriptorSubtype */

0x00, 0x01,    /* bcdADC */

0x27, 0x00,    /* wTotalLength */

0x01,          /* bInCollection */

0x01,          /* baInterfaceNr */
```

---

### 2. 音频类输入终端描述符

在音频类输入终端描述符中bTerminalID为1，wTerminalType为0x0101，bNrChannels为2，wChannelConfig为0x0003，代表该输入终端的终端标号为1，其终端类型是一个USB端点，且支持左右双声道。

---

```
/* Audio Class Specific Input Terminal Descriptor */

0x0C,          /* bLength */

0x24,          /* bDescriptorType */

0x02,          /* bDescriptorSubtype */

0x01,          /* bTerminalID */

0x01, 0x01,    /* wTerminalType */
```

---

---

```
0x00,          /* bAssocTerminal */

0x02,          /* bNrChannels */

0x03, 0x00,    /* wChannelConfig */

0x00,          /* iChannelNames */

0x00,          /* iTerminal */
```

---

### 3.音频类特征单元描述符

音频类特征单元描述符中bSourceID为0x01，表明该音频类特征单元的前序连接的输入终端的标号为1，即为前一小节中音频类输入终端描述符所描述的终端。由于该终端支持左右双声道，音频类特征单元描述符中需要用分别用bmaControls(0), bmaControls(1)和bmaControls(2)描述通道上可被控制的功能。bmaControls(0)为0x03，代表主通道支持静音和音量调节的控制，而另外两个逻辑通道的bmaControls都为0x00，表明不能单独对逻辑通道进行相应的控制。

---

```
/* Audio Class Specific Feature Unit Descriptor */

0x0A,          /* bLength */

0x24,          /* bDescriptorType */

0x06,          /* bDescriptorSubtype */

0x02,          /* bUnitID */

0x01,          /* bSourceID */

0x01,          /* bControlSize */

0x03,          /* bmaControls (0) */
```

---

---

```
0x00,          /* bmaControls (1) */

0x00,          /* bmaControls (2) */

0x00,          /* iFeature */
```

---

## 4.音频类音频流接口描述符

音频类音频流接口描述符中 bTerminalLink 为 0x1 , 表明该接口中传输端点所对应的输入终端的标号为1 , 且音频流编码格式为PCM格式。

---

```
/* Audio Class Specific AudioStreaming INTERFACE Descriptor */

0x07,          /* bLength */

0x24,          /* bDescriptorType */

0x01,          /* bDescriptorSubtype */

0x01,          /* bTerminalLink */

0x01,          /* bDelay */

0x01, 0x00,    /* wFormatTag */
```

---

## 5.音频类数据格式描述符

音频类数据格式描述符中 bNrChannels 为 0x02 , 表明双声道。 bBitResolution 为0x10 , 代表16位量化。 tSamFreq为0x00BB80 , 代表48kHz采样率。

---

```
/* Audio Class Specific type I format INTERFACE Descriptor */

0x0B,          /* bLength */

0x24,          /* bDescriptorType */

0x02,          /* bDescriptorSubtype */
```

---

---

0x01,	/* bFormatType */
0x02,	/* bNrChannels */
0x02,	/* bSubframeSize */
0x10,	/* bBitResolution */
0x01,	/* bSamFreqType */
0x80, 0xBB, 0x00,	/* tSamFreq */

---

## 6. 标准音频传输端点描述符

标准音频传输端点描述符中bmAttributes为0x05，表明该音频流使用异步方式同步信息，所以该音频流接口包含一个同步端点。  
wMaxPacketSize是196，它是由采样率48kHz、16位编码、双通道设备一毫秒内传输的192字节再增加一个4字节的采样帧长度计算而来（详见4.2.1节）。bSynchAddress为反馈端点的地址0x82。

---

/* Standard AS Isochronous Audio Data Endpoint Descriptor */	
0x09,	/* bLength */
0x05,	/* bDescriptorType */
0x02,	/* bEndpointAddress */
0x05,	/* bmAttributes */
0xC4, 0x00,	/* wMaxPacketSize */
0x01,	/* bInterval */
0x00,	/* bRefresh */
0x82,	/* bSynchAddress */

---

## 7. 标准音频同步端点描述符

标准音频同步端点描述符中bmAttributes为0x01，代表该端点是同步端点。wMaxPacketSize长度为3个字节（10.14格式，详见4.2.1节）。

---

```
/* Standard AS Isochronous Synch Endpoint Descriptor */

0x09,          /* bLength */

0x05,          /* bDescriptorType */

0x82,          /* bEndpointAddress */

0x01,          /* bmAttributes */

0x03, 0x00,    /* wMaxPacketSize */

0x01,          /* bInterval */

0x05,          /* bRefresh */

0x00,          /* bSynchAddress */
```

---

## 4.1.5 音频设备请求

音频设备除了支持本书2.1.6节介绍的标准USB设备请求外，还支持音频类特定的请求。特定请求可分为两类：一类是音频控制请求，如对音量、静音、音调的调整；另一类是音频流请求，如采样率的调整。音频设备的请求格式和标准USB设备请求格式是相同的。

控制选择字（Control Selector，CS）代表对音频某个实体对象的控制功能的选择，实体对象可以是特征单元或者输入、输出终端。表4-17列出了部分特征单元的控制选择字，表4-18列出了端点的控制选择字。

表4-17 部分特征单元的控制选择字

控制选择字	值
FU_CONTROL_UNDEFINED	0x00
MUTE_CONTROL	0x01
VOLUME_CONTROL	0x02
BASS_CONTROL	0x03

表4-18 端点的控制选择字

控制选择字	值
EP_CONTROL_UNDEFINED	0x00
SAMPLING_FREQ_CONTROL	0x01
PITCH_CONTROL	0x02

由于实体单元多种多样，限于篇幅，本书只讨论实体对象是特征单元的情况。特征单元的音频类特定请求格式如表4-19所示。

表4-19 特征单元的音频类特定请求格式

bRequestType (1字节)	bRequest (1字节)	wValue (2字节)	wIndex (2字节)	wLength (2字节)
8'b00100001	SET_CUR SET_MAX SET_MIN SET_RES	高字节代表控制选择字，低字节代表逻辑通道	高字节代表实体的ID号，低字节代表接口端点号	数据传输阶段要发送/接收的长度
8'b00100010				
8'b10100001	GET_CUR GET_MAX GET_MIN GET_RES	高字节代表控制选择字，低字节代表逻辑通道	高字节代表实体的ID号，低字节代表接口端点号	数据传输阶段要发送/接收的长度
8'b10100010				

bRequestType的第7位为1 ' b0说明这是个设置请求，为1 ' b1说明这是个获取请求。第5到6位为2 ' b01说明这是个类请求。第0到4位为5 ' b00001说明请求的对象是音频控制接口和音频流接口，为5 ' b00010说明对象是传输端点。

bRequest 代表音频类的请求，常用的有设置当前值（SET\_CUR），设置最大/最小值（SET\_MAX/SET\_MIN），设置分辨率（SET\_RES）。常用音频类请求编码如表4-20所示。

表4-20 常用音频类请求编码

音频类请求	值
REQUEST_CODE_UNDEFINED	0x00
SET_CUR	0x01
GET_CUR	0x81
SET_MIN	0x02
GET_MIN	0x82
SET_MAX	0x03
GET_MAX	0x83
SET_RES	0x04
GET_RES	0x84

wValue的高字节是控制选择字，表示该请求是对音频设备的那个属性（音量、静音等）进行控制选择，控制选择字从表4-17中选择。低字节代表控制选择字所控制的设备逻辑通道。

如请求对象是音频接口，wIndex 高字节表示终端或单元的标识bTerminalID/bUIntID，bRequestType的第0到4位为5 ' b00001，低字节表示口号。如请求对象是音频传输端点，bRequestType第0到4位为5 ' b00000，wIndex高字节为0，低字节表示端点号。

wLength代表该请求数据阶段发送或接收数据的长度。

例如，要获取bUIntID为2的特征单元的通道1的当前音量。bRequestType为0xA1，代表请求对象是音频接口；bRequest为GET\_CUR；wValue为0x0201，wValue 是表 4-17 中 VOLUME\_CONTROL 和通道号的组合；wIndex 为0x0200，是bUIntID 和口号的组合，wLength为2。

## 4.1.6 USB音频类2.0简介

为了更好地支持高速音频设备，2006年USB音频类2.0标准被引入。

USB音频类2.0和1.0相比，最大的区别就是采样率的提升。音频类1.0在数据是双声道、24位编码的情况下，最高支持采样率为96kHz，计算得到 $96 \times 2 \times 3 = 576$ 字节。音频类2.0一般最高支持384kHz的采样率。在高速USB设备下，音频类2.0大大提升了音质和通道数，为用户提供了更好的体验。

音频类2.0的功能拓扑和1.0一样，都有输入终端、输出终端，特征单元、混合单元、选择单元、处理单元和扩展单元。音频类2.0引入了时钟实体（Clock Entity），为设备中所有音频功能模块提供采样时钟。时钟实体分为三类：时钟源、时钟选择单元和时钟倍频单元。时钟源对于该音频模块是必需的。在它的描述符里面，反映该时钟源的属性，是内部固定时钟，还是内部可调时钟，抑或是外部时钟。同时也会表征该时钟源是否可以被主机调节。另外，音频类2.0使用接口关联描述符（Interface Association Descriptor，IAD）来描述音频接口的集合。音频类2.0设备功能拓扑如图4-2所示。

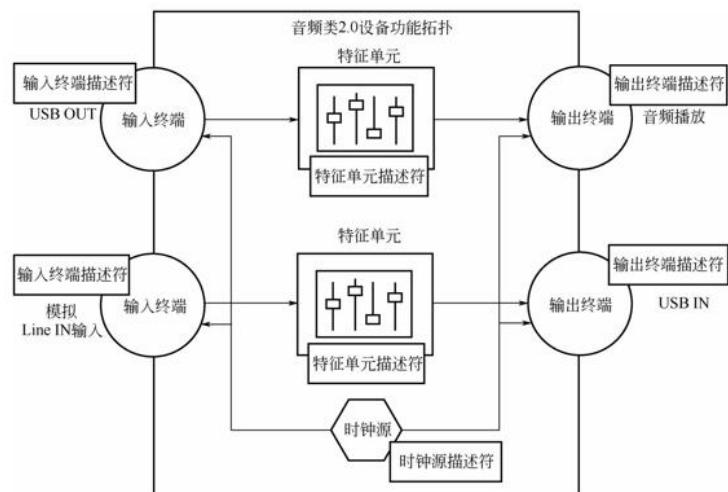


图4-2 音频类2.0设备功能拓扑

本节仍然只介绍音频类 2.0 中实体对象是特征单元的音频类请求。与音频类1.0相比，音频类2.0中请求编码发生了很大的变化。音频类2.0中请求编码如表4-21所示。

表4-21 音频类2.0中请求编码

音频类请求编码	值
REQUEST_CODE_UNDEFINED	0x00
CUR	0x01
RANGE	0x02

音频类1.0中SET\_CUR/GET\_CUR请求被CUR请求所代替，而其他SET\_MIN/GET\_MIN/SET\_MAX/GET\_MAX/SET\_RES/GET\_RES 等 6个请求被RANGE请求所代替。而通过bRequestType的第7位，设备端可以得知主机是想要获取还是设置CUR/RANGE。

另外，在音频类 2.0 中新增加一组只针对时钟实体控制选择字，其定义如表4-22所示。在其对应的音频类请求的wIndex高字节中，需要指定期钟源实体，在实际使用时需要区分针对时钟实体的请求和针对特征单元的请求。

表4-22 时钟实体控制选择字

字 段	值
CS_CONTROL_UNDEFINED	0x00
CS_SAM_FREQ_CONTROL	0x01
CS_CLOCK_VALID_CONTROL	0x02

根据主机发送请求的不同，设备端所返回的数据长度也会不同。例如，设备端对控制选择字为 CS\_SAM\_FREQ\_CONTROL 的请求返回的数据长度为4字节。

数据长度为4个字节的当前属性的结构如表4-23所示。

表4-23 当前属性的结构

字 段	长 度(字节)	描 述
dCUR	4	CUR 属性的返回值

数据长度为4个字节的范围属性的结构如表4-24所示，其总长度为(2+12×n)，n代表范围值的个数。

表4-24 范围属性的结构

字段	长度(字节)	描述
wNumSubRanges	2	一共支持多少种范围值
dMIN (1)	4	这个范围值的 MIN 属性
dMAX (1)	4	这个范围值的 MAX 属性
dRES (1)	4	这个范围值的 RES 属性

数据长度为1字节或2字节时，CUR属性和RANGE属性与4字节的结构类似，在此不再赘述。

需要注意的是，控制选择字为CS\_CLOCK\_VALID\_CONTROL的请求只支持对当前属性的获取，不支持对当前属性的设置和对范围属性的获取与设置。

容易混淆的是，针对时钟实体，控制选择字是CS\_SAM\_FREQ\_CONTROL的请求在音频类2.0下，其当前属性的数据长度为4字节。而在音频类1.0下，有一个非常类似的控制选择字为SAMPLING\_FREQ\_CONTROL（见表4-18），其实体对象是特征单元，且其数据长度为3字节，需要额外注意。

## 4.2 USB音频流同步

USB音频类中一个需要特殊处理的地方就是USB主机和设备间音频流的同步。由于USB主机时钟和USB音频设备的时钟不可能精确同步，在长时间工作的情况下，音频数据流会产生固定方向的漂移而导致不同步。USB音频规范中提供了多种方案用于音频流的同步。由于篇幅所限，本章只介绍USB设备端音频流同步的实现，不涉及USB主机端音频流同步的实现。

以音频扬声器为例，USB主机产生并发送数据，USB设备接收数据，如果USB主机产生数据过快，USB设备来不及接收，音频数据将会丢失，产生噪声；反之，USB主机产生数据过慢，USB设备接收不到数据，也会产生噪声。

有两种方式进行USB主机和设备间音频数据的同步：

- 设备根据缓冲区里剩余空间的大小，动态调整音频时钟频率，使USB主机和USB设备双方产生和消耗音频流速度一致。这种方式称为同步方式。
- 设备端通过反馈端点（Feedback Endpoint）通知USB主机调整音频流发送速度，使USB主机和USB设备双方产生和消耗音频流速度一致。这种方式称为异步方式。

下面将对这两种方式进行具体介绍。

## 4.2.1 反馈端点同步方案

图4-3是利用反馈端点进行同步的工作流程。

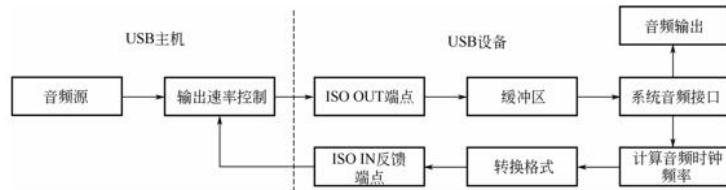


图4-3 利用反馈端点进行同步的工作流程

USB主机和USB设备通过音频输出端点传输音频流，设备端在第一个采样周期内统计它的音频时钟的实际频率，然后根据当前设备缓冲区剩余空间的大小按照一定的算法计算出期望的主机端音频流发送频率。随后将这个频率转换成相应的格式通过反馈端点发送给主机，最后主机将根据设备端的反馈调整音频流端点上音频流数据的长度，使得USB主机和设备端收发音频的速度一致。

### 1. 缓冲区大小和采样周期的选择

为了精确地统计出设备端音频时钟的实际频率，设备需要让系统音频模块先开始工作一段时间（一个采样周期），然后统计出这段时间内音频模块产生的中断数目，并以此计算出音频时钟的实际频率（中断数目与音频时钟实际频率存在一个确定的比例关系，具体算法与实际的音频配置相关，本书中不做讨论）。

如果采样周期太短，样本数目太少，统计出来的数据误差就比较大；而如果采样周期太长，就可能需要较大的缓冲区存放数据，以避免缓冲区溢出。在实际使用中，一般采样周期使用 $2^{10}$ 个SOF作为采样的周期，而缓冲区的大小一般设置为在当前配置下主机一毫秒发送音频数据量的 $2^n$ 倍，如32倍。

例如，音频流的采样率为48kHz，音频流数据采用16位编码，并且支持双通道，此时USB主机端在音频流端点上每毫秒发送音频数据

的长度为： $2 \times (16/8) \times 48 = 192$  字节。此时缓冲区大小一般设为  $(32 \times 192)$  字节即可。

## 2. 反馈频率

反馈频率是USB设备端反馈给USB主机，期望USB主机发送音频流数据所采用的频率。例如，主机端发送的音频流的采样率为 48kHz，但是由于设备端音频模块实际频率只有 47.9kHz，如果主机端一直采用 48kHz 的频率发送数据，设备端的音频流缓冲区工作一段时间后就会溢出。所以USB设备端会通过反馈端点给USB主机发送一个 47.9kHz 的频率，这个频率就是USB设备端期望USB主机发送音频流的频率。

反馈频率主要由USB设备端音频模块的实际频率决定。另外，设备端音频流缓冲区剩余空间的大小对于反馈频率也有所影响。例如，如果音频流缓冲区剩余空间还较多，反馈频率可以直接使用设备音频模块的实际频率；而如果单前缓冲区剩余空间已经不多，反馈频率则需要设置为比设备音频模块的实际频率更小的频率，以期尽快减少缓冲区使用比例，避免缓冲区溢出。

## 3. 反馈端点中频率的格式

在反馈端点中传输的数据是3个字节，采用10.14格式，其高10位代表反馈频率的整数部分，剩余部分代表反馈频率的小数部分，但小数部分最低4位不使用。反馈频率的单位是kHz。

仍以47.9kHz为例，将整数部分47转为10位二进制是10' b0000101111，将小数部分0.9转为14位二进制是14' b11100110100000，将整数部分左移14位与小数部分进行逻辑或之后转换成10.14的格式后得到 0x0BF9A0。

## 4. 音频流端点中数据长度

当USB主机通过反馈端点接收到设备端反馈的频率后，将会对音频流端点上的音频载荷进行微调，使得双方的收发音频流的速度一

致。但是，USB主机对于音频载荷的调整只能在一个有限的范围内。音频流编码位数和音频流通道数的乘积即为音频流的采样帧（Sample Frame），主机端对于音频数据长度的调整就是以采样帧的长度为单位的。

例如，音频流的采样率为48kHz，音频流数据采用16位编码，支持双通道。此时USB主机端在音频流端点上每毫秒发送音频数据的长度为： $2 \times (16/8) \times 48 = 192$ 字节，而该音频流的采样帧长度为： $(16/8) \times 2 = 4$ 字节，此时主机端会根据反馈端点发送过来的反馈频率对音频流端点上音频流数据进行微调，调整后的音频流数据大小为：原音频数据长度 $\pm$ 采样帧长度，在本例中为188字节或者196字节。

主机端并不会调整所有发送的音频流数据长度，而是根据设备端的反馈频率对一定比例的数据包长度做调整。具体的调整方式为：先算出一个固定周期内需要调整的数据量总大小，除以采样帧长度得出周期内需要进行长度调整数据包的个数。例如，音频流的采样率为48kHz，音频流数据采用16位编码，支持双通道，设备端反馈频率为47.9kHz。以一秒为一个周期，在一个周期内，主机端需要少发 $(48 - 47.9) / 48 \times 192 \times 1000 = 400$ 字节，因为采样帧长度为4，所以一个周期内需要调整长度的数据包的个数为 $400/4 = 100$ 个。主机会按照一定的算法将这100个调整过长度的数据包在下个周期内发出。

## 5.其他注意事项

由于反馈端点机制的作用，USB主机有可能会发出比原有的最大包长度更大的数据包。例如，音频流的采样率为48kHz，音频流数据采用16位编码，支持双通道，设备端反馈频率为48.1kHz，这时USB主机需要发出 $192 + 4 = 196$ 字节的数据包来满足音频设备端过快的时钟频率。这是在开发音频设备时需要特别注意的地方。

表4-15标准音频同步端点描述符中bRefresh字段代表主机轮询反馈频率值的周期，其值为 $2^{bRefresh}$ 毫秒。但是在Windows 7下使用该字段有一个限制，就是该字段的值必须大于等于4，就是主机轮询反馈频率值的周期必须大于 $2^4 = 16$ ms，主机才能正常工作。如果这个值小于

4，则主机会概率性地停止向USB音频设备发送音频流数据，这是开发过程中另外一个需要注意的地方。

最后需要指出的是，使用反馈端点异步调整主机发送数据流的长度仍然对USB音频设备的自身时钟有要求。例如，音频流的采样率为48kHz，音频流数据采用16位编码，支持双通道，主机端的最大调整幅度为 $4 \div 192 \times 100\% = 2\%$ ，如果设备端自身时钟偏差程度偏差超2%，再怎么调整反馈端点频率值，在长时间播放的情况下，仍然还是会有噪声。

## 4.2.2 调节时钟同步方案

对于调整音频时钟进行同步的方案，就是根据缓冲区剩余空间的大小，来动态调整音频设备的时钟频率。当缓冲区剩余空间较少时，需提高音频频率。当缓冲区剩余空间较大时，可适当降低音频频率。由于不需要反馈端点的介入，在这种情况下，该方案设备描述符中将不包含同步端点描述符。并且，其标准传输端点描述符中bmAttributes字段的第2到3位同步方式需要从异步（2'b01）改成同步（2'b11）。bRefresh 和 bSynchAddress 字段都需设成零。

## 4.3 USB音频演示程序代码分析

本节将结合恩智浦MCUXpresso SDK中USB扬声器演示程序的源代码，先介绍音频描述符的实现，然后从代码角度，讲解音频描述符的构建以及连接检测、速度检测、枚举、挂起和恢复等USB特性。

本书中所用的源代码可登录

<https://mcuxpresso.nxp.com/en/welcome>网站，选择TWR-K65F180M开发板即可下载。

本节所用例程路径为

boards\twrk65f180m\usb\_examples\usb\_device\_audio\_speaker。

### 4.3.1 连接/断开检测代码分析

由于本书使用的芯片的KHCI控制器不支持设备模式的连接/断开检测，所以本节着重讨论基于EHCI控制器的连接/断开检测的软件行为。

#### 1.连接检测代码分析

图4-4为USB设备连接检测流程，具体分析如下：

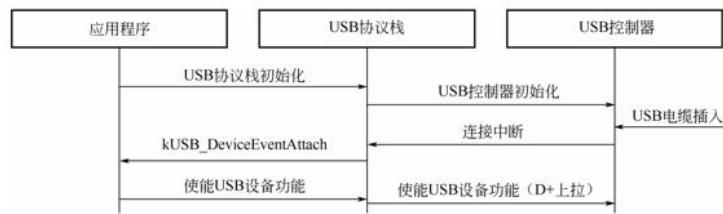


图4-4 USB设备连接检测流程

(1) 应用程序首先初始化USB协议栈。USB协议栈在初始化USB设备控制器时会将控制器的连接检测功能初始化。

具体功能和代码参考 `usb_device_ehci.c` 文件的 `USB_Device_Ehci Set DefaultState()` 函数。其中，连接检测代码如下：

```
static void USB_DeviceEhciSetDefaultState (usb_device_ehci_state_struct_t *ehciState)
{
    ...
    ehciState->registerBase->OTGSC = ehciState->registerBase->OTGSC &
    0x0000FFFF;
    ehciState->registerBase->OTGSC |= USBHS_OTGSC_BSVIE_MASK;
    ...
}
```

第一行代码用来清除中断状态寄存器。第二行代码主要用于使能OTGSC[BSV]变化中断。连接检测功能的原理、寄存器介绍和配置请

参考3.1节。

(2) 当USB设备接入USB主机时，USB设备控制器会产生B设备会话有效状态变化中断。USB协议栈的中断处理函数根据OTGSC[BSV]的状态判断该事件是连接还是断开。由于中断由设备连接产生，OTGSC[BSV]为真，USB协议栈将上报设备连接事件(kUSB\_DeviceEventAttach)。

具体功能和代码参考usb\_device\_ehci.c文件的USB\_DeviceEhciIsrFunction()函数。其中，中断处理代码如下：

---

```
void USB_DeviceEhciIsrFunction (void *deviceHandle)
{
    .....
    if (ehciState->registerBase->OTGSC & USBHS_OTGSC_BSVIS_MASK)
    {
        usb_device_callback_message_struct_t message;

        ehciState->registerBase->OTGSC |= USBHS_OTGSC_BSVIS_MASK;

        message.buffer = (uint8_t *) NULL;
        message.length = 0U;
        message.isSetup = 0U;

        if (ehciState->registerBase->OTGSC & USBHS_OTGSC_BSV_MASK)
        {
            /* Device is connected to a host. */
            message.code = kUSB_DeviceNotifyAttach;
            USB_DeviceNotificationTrigger (ehciState->deviceHandle, &message);
        }
    }
}
```

---

```
    }
    else
    {
    .....
    }
    .....
}
```

当中断产生后，中断处理函数通过OTGSC[BSVIS]判断是否产生了连接/断开中断。如果 OTGSC[BSVIS]为真，那么中断处理函数清除该状态并判断OTGSC[BSV]是否为真。如果 OTGSC[BSV]为真，那么中断处理函数将上报设备连接事件（kUSB\_DeviceEventAttach）。

(3) 在收到事件 kUSB\_DeviceEventAttach 后，应用程序通过 USB 协议栈使能USB设备功能。USB协议栈通过置USBCMD[RS]使能 USB设备功能（控制器内部会将D+上拉）。详细信息可参见本书3.1.2节。

具体功能和代码参考usb\_device\_ehci.c文件的 USB\_DeviceEhciControl( ) 函数。其中，USB设备使能代码如下：

```
usb_status_t USB_DeviceEhciControl ( usb_device_controller_handle ehciHandle ,
usb_device_control_type_t type, void *param)
{
.....
case kUSB_DeviceControlRun:
    ehciState->registerBase->USBCMD |= USBHS_USBCMD_RS_MASK;
.....
error = kStatus_USB_Success;
break;
.....
}
```

## 2.断开检测代码分析

图4-5为USB设备断开检测流程，具体分析如下：

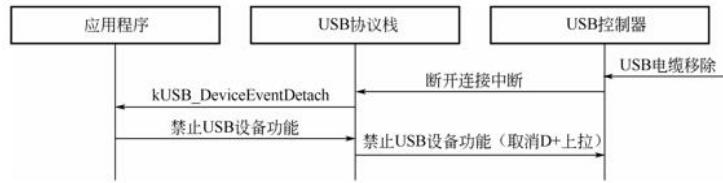


图4-5 USB设备断开检测流程

(1) 当USB设备从USB主机拔出时，USB设备控制器会产生OTGSC[BSV]变化中断(OTGSC[BSVIS]为真)。USB协议栈的中断处理函数根据OTGSC[BSV]的状态判断该事件是连接还是断开。由于中断由设备断开连接产生，OTGSC[BSV]为假，USB协议栈将上报设备连接事件(kUSB\_DeviceEventDetach)。

具体功能和代码参考usb\_device\_ehci.c文件的USB\_DeviceEhciIsrFunction()函数。其中，中断处理代码如下：

```
void USB_DeviceEhciIsrFunction (void *deviceHandle)
{
    .....
    if (ehciState->registerBase->OTGSC & USBHS_OTGSC_BSVIS_MASK)
```

```
{  
    usb_device_callback_message_struct_t message;  
  
    ehciState->registerBase->OTGSC |= USBHS_OTGSC_BSVIS_MASK;  
  
    message.buffer = (uint8_t *) NULL;  
    message.length = 0U;  
    message.isSetup = 0U;  
  
    if (ehciState->registerBase->OTGSC & USBHS_OTGSC_BSV_MASK)  
    {  
        .....  
    }  
    else  
    {  
        /* Device is disconnected from a host. */  
        message.code = kUSB_DeviceNotifyDetach;  
        USB_DeviceNotificationTrigger(ehciState->deviceHandle, &message);  
    }  
}  
.....  
}
```

(2) 当中断产生后，中断处理函数通过 OTGSC[BSVIS]判断是否产生了连接/断开中断。如果OTGSC[BSVIS]为真，那么中断处理函数清除该状态并判断OTGSC[BSV]是否为真。如果OTGSC[BSV]为假，那么中断处理函数将上报设备连接事件（kUSB\_DeviceEventDetach）。

(3) 在收到事件kUSB\_DeviceEventDetach后，应用程序通过USB协议栈禁止USB设备功能。USB协议栈通过清除USBCMD[RS]禁止USB设备功能（控制器内部会禁止D+上拉）。详细信息可参见本书3.1.2节。

具体功能和代码参考usb\_device\_ehci.c文件的USB\_DeviceEhciControl（）函数。其中，USB设备使能代码如下：

---

```
usb_status_t USB_DeviceEhciControl ( usb_device_controller_handle ehciHandle ,
usb_device_control_type_t type, void *param )

{

.....



case kUSB_DeviceControlRun:

    ehciState->registerBase->USBCMD &= ~USBHS_USBCMD_RS_MASK;

    error = kStatus_USB_Success;

    break;

.....



}
```

---

### 4.3.2 速度检测代码分析

图4-6为USB设备速度检测流程，具体分析如下：

(1) 当检测到总线的复位信号后，USB设备控制器通过复位中断将该事件上报。

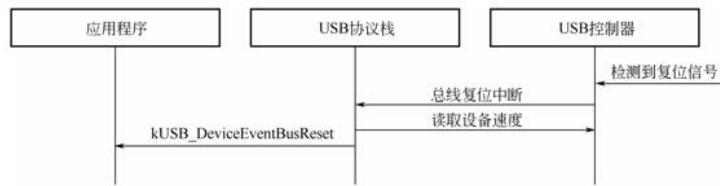


图4-6 USB设备速度检测流程

(2) 中断处理函数根据中断状态处理相应的中断。由于 KHCI 控制器作为设备时，系统只能工作在全速模式。在收到总线复位中断后，USB协议栈直接上报总线复位事件 (kUSB\_DeviceNotifyBusReset)。所以本节将着重介绍EHCI设备控制器的速度识别流程。

在检测到总线复位信号后，产生总线复位中断。在复位完成后，EHCI设备控制器产生端口复位清除中断。

对于总线复位中断处理，具体功能和代码参考 `usb_device_ehci.c` 文件的 `USB_DeviceEhciInterruptReset()` 函数。中断处理函数根据端口是否处于复位状态做不同处理。如果端口处于复位状态，那么置总线复位标志 `isResetting`；否则，直接上报复位事件。其中，部分代码如下：

---

```

static void USB_DeviceEhciInterruptReset (usb_device_ehci_state_struct_t *ehciState)
{
    .....
    /* Whether is the port reset. If yes, set the isResetting flag. Or, notify the up layer. */
    if (ehciState->registerBase->PORTSC1 & USBHS_PORTSC1_PR_MASK)
    {
        ehciState->isResetting = 1U;
    }
}

else
{
    usb_device_callback_message_struct_t message;
    message.buffer = (uint8_t *) NULL;
    message.code = kUSB_DeviceNotifyBusReset;
    message.length = 0U;
    message.isSetup = 0U;
    USB_DeviceNotificationTrigger (ehciState->deviceHandle, &message);
}

```

---

对于端口复位清除中断处理，具体功能和代码参考  
usb\_device\_ehci.c文件的USB\_DeviceEhciInterruptPortChange（）函数。其中，读取速度代码如下：

---

```

static void USB_DeviceEhciInterruptPortChange (usb_device_ehci_state_struct_t
*ehciState)
{
    .....
    /* Whether the port is doing reset. */
    if (! (ehciState->registerBase->PORTSC1 & USBHS_PORTSC1_PR_MASK) )
    {
        /* If not, update the USB speed. */
        if (ehciState->registerBase->PORTSC1 & USBHS_PORTSC1_HSP_MASK)

```

---

```
    {
        ehciState->speed = USB_SPEED_HIGH;
    }
    else
    {
        ehciState->speed = USB_SPEED_FULL;
    }

    /* If the device reset flag is non-zero, notify the up layer the device reset
finished. */
    if (ehciState->isResetting)
    {
        message.code = kUSB_DeviceNotifyBusReset;
        USB_DeviceNotificationTrigger(ehciState->deviceHandle, &message);
        ehciState->isResetting = 0U;
    }
}
.....
}
```

在端口状态变化处理函数中，如果端口不处于复位状态，那么获取设备的前状态。此时，如果复位状态为真（isResetting 为 1），那么上报总线复位事件（kUSB\_DeviceNotifyBusReset）。

（3）在收到事件kUSB\_DeviceNotifyBusReset后，应用程序开始初始化控制传输端点并准备好接收主机的Setup请求。

### 4.3.3 枚举、挂起和恢复代码分析

本节将从实际出发介绍USB音频设备的枚举过程中软件行为和软硬件的交互。同时，本节将给出USB协议栈的执行流程和关键代码。另外，本节将进一步讨论USB协议栈对挂起和恢复的实现。

#### 1. 枚举

USB设备的枚举如第2章所述，相关操作按序为：获取设备描述符前8个字节、设置设备的地址、获取完整的设备描述符、获取配置描述符前9字节、获取完整的配置描述符、获取字符串描述符、设置设备的配置值。

图4-7为USB设备枚举流程，具体分析如下：

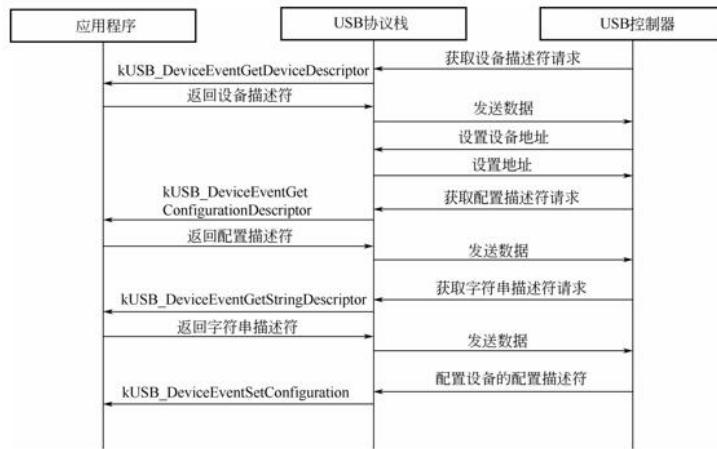


图4-7 USB设备枚举流程

(1) 当收到获取设备描述符请求时，USB协议栈会上报kUSB\_DeviceEventGetDeviceDescriptor事件获取设备描述符。应用程序在收到该事件后将相应的描述符地址和长度填充到该事件的相关字段，详细信息参考文件usb\_device\_descriptor.c的函数USB\_DeviceGetDeviceDescriptor( )。检查如果发现设备描述符已填充，USB协议栈通过数据发送接口将描述符发出，同时准备好接收主

机状态阶段的 0 长度数据包。如果描述符未填充或者无效，USB 协议栈通过端点停止接口将控制端点停止（返回 stall 状态）。详细信息参考文件 `usb_device_ch9.c` 的函数

`USB_DeviceControlCallbackFeedback()`。

（2）当收到设置地址请求时，USB 协议栈将地址保存并将当前状态设置为“地址设置状态”，并向 USB 主机发送状态阶段的 0 长度包。在该包传输成功（收到主机返回的 ACK 握手）后，USB 协议栈将先前保存的地址写入寄存器并将当前状态改为“地址状态”。详细信息参考文件 `usb_device_ch9.c` 的函数 `USB_DeviceCh9SetAddress()`。

（3）获取配置描述符请求和获取字符串描述符请求与获取设备描述基本相同。当收到获取配置描述符请求时，USB 协议栈会上报 `kUSB_DeviceEventGetConfigurationDescriptor` 事件。当收到获取设备描述符请求时，USB 协议栈会上报 `kUSB_DeviceEventGetStringDescriptor` 事件。

（4）当收到配置设备的配置描述符请求时，USB 协议栈会上报 `kUSB_DeviceEventSetConfiguration`，并向 USB 主机发送状态阶段的 0 长度包。应用程序或者类驱动根据请求携带的配置更新当前的配置。一般情况下，应用程序或者类驱动会复位旧的配置并设置新的配置。详细信息可参考文件 `usb_device_audio.c` 的函数 `USB_DeviceAudioEvent()`。

## 2. 挂起

图 4-8 为 USB 设备挂起流程，具体分析如下：

（1）在检测到总线挂起状态后，USB 设备控制器会以中断方式将该状态上报。



图4-8 USB设备挂起流程

(2) USB DCD收到控制器的挂起中断后，会将该事件一级级上报，直到应用程序收到该事件。

(3) 应用程序在收到总线挂起事件后将所有在低功耗状态无须工作的模块关闭并禁止相应的引脚（为了降低引脚电路漏电的风险）。同时，应用程序配置低功耗模块做好进入低功耗模式的准备。

(4) 如果设备支持远程唤醒功能且该功能已经被USB主机使能，那么应用程序需要配置本地唤醒源（如开关）。该功能主要用于能够使得用户在需要远程唤醒主机时设备能够被用户通过该唤醒源首先唤醒。

(5) 在完成所有配置后，应用程序通过USB协议栈API设置USB设备控制器进入低功耗模式。当收到该请求后，DCD在将USB设备作为异步唤醒源（主要用于检测USB主机的恢复信号并唤醒系统）后进入低功耗状态。

对于KHCI控制器，具体功能和代码参考usb\_device\_khci.c文件的USB\_DeviceKhciInterruptSleep()函数。异步唤醒配置如下：

---

```

khciState->registerBase->USBTRC0 |= USB_USBTRC0_USBRESMEN_MASK;
khciState->registerBase->USBCTRL |= USB_USBCTRL_SUSP_MASK;

```

---

对于 EHCI 控制器，具体功能和代码参考 `usb_device_ehci.c` 文件的 `USB_DeviceEhciControl( )` 函数。异步唤醒配置如下：

```
ehciState->registerBase->OTGSC |= 0x007F0000U;  
ehciState->registerPhyBase->PWD = 0xFFFFFFFF;  
while (ehciState->registerPhyBase->CTRL & (USBPHY_CTRL_UTMI_SUSPENDM_  
MASK))  
{  
    __ASM ("nop");  
}  
ehciState->registerBase->USBSTS |= USBHS_USBSTS_SRI_MASK;  
ehciState->registerBase->PORTSC1 |= USBHS_PORTSC1_PHCD_MASK;  
ehciState->registerBase->USBGENCTRL = USBHS_USBGENCTRL_WU_IE_  
MASK;  
ehciState->registerPhyBase->CTRL |= USBPHY_CTRL_CLKGATE_MASK;
```

(6) 在所有的设置成功后，应用程序设置低功耗模块使得系统进入低功耗模式。

### 3. 恢复

根据恢复信号的发起者不同，本节将从设备发起的恢复信号和主机发起的恢复信号两个部分进行介绍。

#### 1) 设备发起的恢复信号

图4-9为USB设备远程唤醒总线流程，具体分析如下：



图4-9 USB设备远程唤醒总线流程

(1) 当唤醒按键被按下后，系统会被异步唤醒机制唤醒。

(2) 系统被唤醒后等待系统时钟稳定。在时钟稳定后，应用程序恢复引脚设置和重新使能被关闭的模块。一旦全局中断被使能，CPU会被按键唤醒中断打断。在按键唤醒中断服务程序中，应用程序将会设置按键唤醒标志。所以在全局中断使能后，应用程序将会判断按键唤醒标志是否设置。如果标志被设置，那么应用程序将会通过USB协议栈驱动总线产生恢复信号。否则，应用程序认为系统是由恢复信号唤醒，并等待恢复事件。由于按键唤醒标志被设置，应用程序将会远程唤醒总线。

(3) 应用程序通过USB协议栈提供的接口驱动总线产生10ms的恢复信号。在执行结束后，应用程序将会等待恢复事件。

对于 KHCI 控制器，具体功能和代码参考 `usb_device_khci.c` 文件的 `USB_DeviceKhciControl()` 函数。设备产生恢复信号并保持10ms的代码如下：对于 EHCI 控制器，具体功能和代码参考 `usb_device_ehci.c` 文件的 `USB_DeviceEhciControl()` 函数。设备产生恢复信号并保持10ms的代码如下：

```

khciState->registerBase->CTL |= USB_CTL_RESUME_MASK;

startTick = deviceHandle->hwTick;

while ((deviceHandle->hwTick - startTick) < 10)

{

    __ASM ("nop") ;

}

khciState->registerBase->CTL &= ~USB_CTL_RESUME_MASK;

```

---

```

ehciState->registerBase->USBGENCTRL &= ~USBHS_USBGENCTRL_WU_IE_MASK;

ehciState->registerBase->PORTSC1 &= ~USBHS_PORTSC1_PHCD_MASK;

ehciState->registerBase->PORTSC1 |= USBHS_PORTSC1_FPR_MASK;

startTick = deviceHandle->hwTick;

while ((deviceHandle->hwTick - startTick) < 10)

{

    __ASM ("nop") ;

}

ehciState->registerBase->PORTSC1 &= ~USBHS_PORTSC1_FPR_MASK;

```

---

(4) 在收到恢复事件 (kUSB\_DeviceEventResume) 后，应用程序恢复正常状态。

## 2) 主机发起的恢复信号

图4-10为主机唤醒总线流程，具体分析如下：

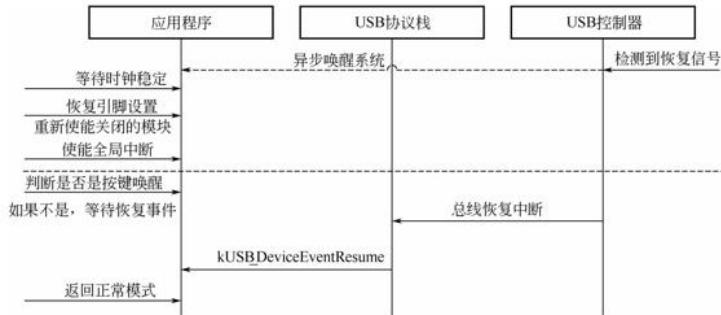


图4-10 主机唤醒总线流程

(1) 当检测到恢复信号后，USB设备控制器通过异步唤醒机制唤醒系统。

(2) 系统被唤醒后等待系统时钟稳定。在时钟稳定后，应用程序恢复引脚设置和重新使能被关闭的模块。在使能全局中断后，应用程序检测系统是否是被按键唤醒。由于按键唤醒标志未被设置，应用程序将会等待恢复事件。

(3) 在收到恢复事件 (kUSB\_DeviceEventResume) 后，应用程序恢复正常状态。

## 4.3.4 音频流同步代码分析

### 1. 反馈端点同步方案

在audio\_speaker.c文件的USB\_AudioFeedbackDataUpdate函数中，需要计算10.14格式的频率值，部分代码如下：

```
originFeedbackValue = ((g_UsbDeviceAudioSpeaker.audioSendCount-g_UsbDevice Audio Speaker.  
lastAudioSendCount) << 4) / (AUDIO_FORMAT_CHANNELS * AUDIO_FORMAT_SIZE);
```

上述代码是在一个采样周期结束时，通过获取当前音频模块发送的字节数（g\_UsbDeviceAudioSpeaker.audioSendCount）和上个周期结束时，音频模块发送字节数

（g\_UsbDeviceAudioSpeaker.lastAudioSendCount）的差值，除以每个采样帧长度

（AUDIO\_FORMAT\_CHANNELS\*AUDIO\_FORMAT\_SIZE），得到周期内处理帧的个数，然后左移4位变成10.14的格式。

需要注意的是，如4.2.1节所述，标准做法是先计算出音频时钟的频率值，把它的整数部分左移14位与小数部分进行逻辑或之后再转换成10.14的格式。但该计算过程需要使用双精度数据。为了简单起见，由于10.14格式的小数部分有14位，在得到1024( $2^{10}$ )个SOF中的处理帧个数后，左移 $14-10=4$ 位就是主机需要的反馈频率值。同理，如果采样周期为512( $2^9$ )个SOF时，需要把周期内处理帧个数左移5位作为反馈频率值。

在计算得到USB设备端音频模块的实际频率后，音频时钟在长时间工作的情况下仍会发生固定方向的偏移，这时需要计算缓冲区剩余空间的大小对反馈频率再次微调。

在audio\_speaker.c文件的USB\_AudioFeedbackDataUpdate函数中，计算缓冲区的剩余空间，在原先反馈频率的基础上进行动态调整，代码如下：

```
usedSpace= USB_AudioSpeakerBufferSpaceUsed();

if ( usedSpace >= AUDIO_BUFFER_UPPER_LIMIT (AUDIO_SPEAKER_DATA_WHOLE_BUFFER) )

{

    feedbackValue == (AUDIO_SAMPLING_RATE_KHZ / AUDIO_SAMPLING_RATE_16KHZ) * (AUDIO_ADJUST_MIN_STEP) ;

}

else if ( usedSpace < AUDIO_BUFFER_LOWER_LIMIT (AUDIO_SPEAKER_DATA_WHOLE_BUFFER) )

{

    feedbackValue += (AUDIO_SAMPLING RATE_KHZ /AUDIO_SAMPLING RATE_16KHZ) * (AUDIO_ADJUST_MIN_STEP) ;

}
```

AUDIO\_SPEAKER\_DATA\_WHOLE\_BUFFER 为音频缓冲区总长度。AUDIO\_BUFFER\_UPPER\_LIMIT 和 AUDIO\_BUFFER\_LOWER\_LIMIT 的定义为下列代码。

```
#define AUDIO_BUFFER_UPPER_LIMIT (x) ( ((x)*5) / 8)
#define AUDIO_BUFFER_LOWER_LIMIT (x) ( ((x)*3) / 8)
```

其目的为根据总缓冲区的大小，计算出上限和下限值。

如果发现缓冲区已使用空间超过了其最高门限值  
AUDIO\_BUFFER\_UPPER\_LIMIT (AUDIO\_SPEAKER\_DATA\_WHOLE\_BUFFER)，说明缓冲区快要耗尽，需要减少反馈频率，以避免缓冲区溢出。当已使用空间低于最低门限值时，实现原理基本类似，不再赘述。

AUDIO\_ADJUST\_MIN\_STEP 的值为 0x10，是 10.14 格式中主机可以分辨的最小单位。需要调整的频率变化值是  
AUDIO\_ADJUST\_MIN\_STEP 乘以实际采样率与 16kHz 基准采样率的比值。

## 2. 调节时钟同步方案

由于在 TWR-K65F180M 开发板上时钟频率不可动态调整，该同步方案在该开发板上并未实现。用户可登录 <https://mcuxpresso.nxp.com/en/welcome> 网站，选择LPCXpresso54608 开发板即可下载到含有调整时钟同步方案的源代码。

代码所在例程路径为

boards\lpcxpresso54608\usb\_examples\usb\_device\_audio\_speaker。

在恩智浦 LPC54S60X 芯片上，软件可以通过寄存器 FROCTL [FREQ TRIM]位域调节自激振荡器（Free Running Oscillator）改变音频时钟频率。当FROCTL[FREQTRIM]的值增减1时，音频时钟就会在原值的基础上增减大约0.1%。由于不能精确匹配主机端和设备端数据产生和消耗的速度，所以还需要在程序中周期性地判断缓冲区剩余空间的大小，继续调节 FROCTL [FREQTRIM]。

在audio\_speaker.c的USB\_AudioFSSync函数中，实现了根据缓冲区剩余空间大小，调节 FREQTRIM值的功能，部分代码如下：

---

```
    usedSpace = USB_AudioSpeakerBufferSpaceUsed();

    if (usedSpace >= AUDIO_BUFFER_UPPER_LIMIT (AUDIO_SPEAKER_DATA_
WHOLE_BUFFER) )

    {

        audio_trim_up();

    }

    else if (usedSpace < AUDIO_BUFFER_LOWER_LIMIT (AUDIO_SPEAKER_DATA_
WHOLE_BUFFER) )

    {

        audio_trim_down();

    }

    void audio_trim_up (void)

    {

        uint32_t val = SYSCON->FROCTRL;

        val = (val & ~ (0xff<<16) ) | ( ( (val>>16) & 0xFF) + 1) <<16) | (1UL

<<31) :
```

---

---

```
        SYSCON->FROCTRL = val;

    }

    void audio_trim_down (void)

    {

        uint32_t val = SYSCON->FROCTRL;

        val = (val & ~ (0xff<<16) ) | ( ( (val>>16) & 0xFF) - 1) <<16) | (1UL

<<31) :

        SYSCON->FROCTRL = val;

    }
```

---

AUDIO\_BUFFER\_UPPER\_LIMIT 和  
AUDIO\_BUFFER\_LOWER\_LIMIT 与上节定义一致。

如果发现缓冲区已使用空间超过了其最高门限值，说明缓冲区快要耗尽，需要调用audio\_trim\_up函数，提高设备端的时钟频率，加快USB设备消耗音频数据的速度。当已使用空间低于最低门限值时，实现原理基本类似，不再赘述。

### 4.3.5 USB音频示例演示

本节使用恩智浦推出的塔式评估板套件（包括 TWR-ELEV、TWR-K65 F180M 及 TWR-AUDIO-SGTL）演示 usb\_device\_audio Speaker 示例在 Windows 操作系统上的使用方法。主要分为如下几步：

- (1) 将示例编译完成后下载到目标板，将设备连接到PC。
- (2) 在设备管理器中可以发现一个显示信息为“USB AUDIO DEMO”的音频设备。
- (3) 右击任务托盘中的音频控制图标，然后选择播放设备（Playback Devices），如图4-11所示。



图4-11 选择播放设备

- (4) 在弹出的窗口中选择播放（Playback）标签。然后选择“USB AUDIO DEMO”扬声器点击属性（Properties）按钮，如图4-12所示。



图4-12 选择“USB AUDIO DEMO”扬声器点击属性按钮

(5) 在弹出的窗口中将音量调节到 100%，调节扬声器音量如图 4-13 所示。

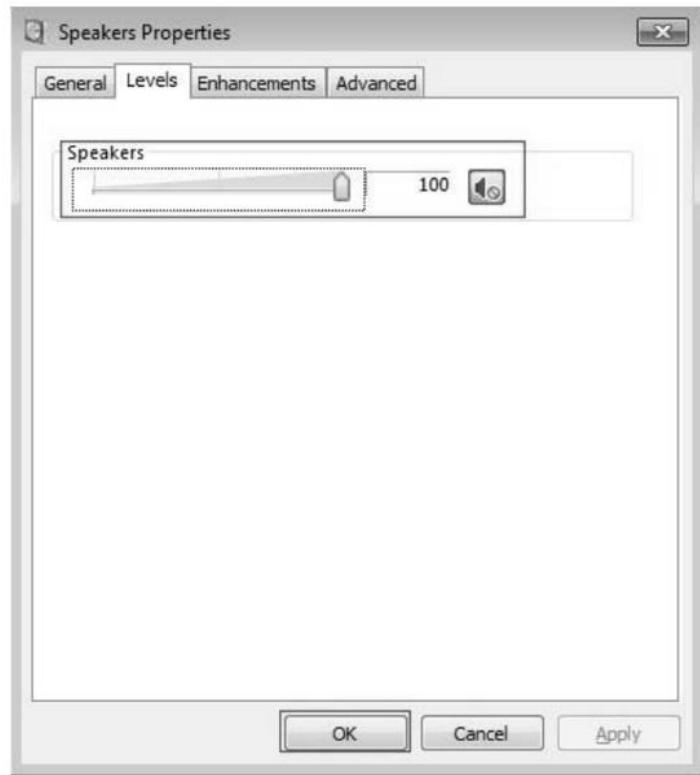


图4-13 调节扬声器音量

(6) 在返回第4步描述的窗口后，选择“USB AUDIO DEMO”扬声器为默认设备，如图4-14所示。



图4-14 选择“USB AUDIO DEMO”扬声器为默认设备

(7) 将耳机或者扬声器插入TWR-AUDIO-SGTL的J3后，使用音频播放软件播放音频文件。此时，音乐会从耳机或者扬声器播放。

## **第5章**

# **USB认证**

随着USB技术的快速发展，市场上USB的产品越来越多，USB的兼容测试认证变得越来越重要，USB认证已经成为一个USB产品的质量保证及认可的标志。本章将介绍USB 2.0认证相关的一些基本信息，并着重介绍获取USB 2.0认证证书的流程，最后对认证过程中典型的测试用例进行分析，让读者对USB 2.0认证以及相关的兼容性测试有基本的了解，并对读者实际的USB认证过程提供指导。

## 5.1 USB认证的简介

USB协会（USB Implementers Forum，USB-IF）是一个致力于推广并发展USB技术的非营利性组织，该组织制定标准的USB传输接口规格，标准化计算机与外围USB设备间的连接接口，同时为了确保USB产品的互联互通性，USB-IF定义了多项兼容性测试，任何USB产品必须要先通过USB-IF所规定的特定测试后才能使用USB标志。产品接受测试的方法有两种：参加USB-IF赞助的兼容性测试大会或私人测试实验室，可以从网站

（<http://www.usb.org/developers/compliance/labs/>）找到符合USB-IF要求的测试实验室。一旦产品通过兼容性测试，它就会列入USB整合厂商清单（Integrators List），同时获得一个协会测试号（TESTING ID，TID）并有权使用USB标志。

图5-1是一些典型的USB标志。



图5-1 典型的USB标志

下面对图中几个USB标志进行简单介绍。

左边第一个标志是USB全速/低速标志，用于12Mbps或者1.5Mbps的产品；左边第二个标志是带OTG功能的USB全速/低速标志，用于带OTG功能 12Mbps 或者 1.5Mbps 的产品；右边第二个标志是 USB 高速标志，用于480Mbps的产品，这个标志不能用于全速/低速的产品；右

边第一个标志是带OTG功能的USB高速标志，用于带OTG功能480Mbps的产品，这个标志不能用于全速/低速的产品。

USB兼容性认证主要包括下面几个认证项：

- 外围设备（Peripheral Device）；
- 集线器或组合设备（Hub/Compound Device）；
- 主机/系统（Host/System）；
- 嵌入式主机（Embedded Host）；
- USB OTG设备（USB On-The-Go）；
- 芯片类（Silicon Building Block）。

在微处理器领域，根据产品类型一般需要获得外围设备和嵌入式主机这两类认证。每一项测试内容应根据USB的版本、产品类型等来决定，大致分类如下。

- 电气性能测试：主要测试产品的电气信号，如眼图测试。
- USB协议功能测试：主要测试产品的功能是否完整，如USB Command Verifier（USB CV）测试，枚举是否成功，功能是否完整等。
- 在不同操作系统和不同控制器下的互用性测试（兼容性部分）：主要测试USB设备在不同的软件操作系统和不同硬件控制器下的兼容性。可能会使用“Gold Tree”上的一些设备来做互操作性测试，会测试设备和不同主机的兼容性，如设备在使用OHCI控制器的主机下能不能枚举成功等。

在5.3节会介绍典型的测试用例来说明每一项USB测试的主要内容。

## 5.2 USB认证的流程

USB兼容性测试认证的一般流程如下：

- 申请成为USB协会会员，获得供应商识别码；
- 准备USB兼容性测试清单；
- 送USB授权实验室进行测试；
- 获得合格报告及证书；
- 使用相应的USB 标志。

## 5.2.1 获得供应商识别码

要获取测试设备的供应商识别码（Vendor ID，VID），需登录申请供应商识别码的 USB-IF 协会网站

（<http://www.usb.org/developers/vendor>）下载并签署相应的协议书。包括以下两种协议。

- 成为 USB-IF 协会会员：每年会员费 4000 美元，成为会员可免费参加 USB-IF 协会所举办的专题讨论会等，还有一些其他相关的权益。
- 成为USB-IF非会员标志许可证持有人：费用是两年3500美元。

## 5.2.2 准备USB兼容性测试清单

表5-1和表5-2列出了USB协会所要求的兼容性测试项，5.3节将详细讲解其中几个典型的测试用例。本书并不涉及USB兼容性测试过程，如果读者想进一步了解USB兼容性预测试过程，请参考其他书籍或者USB-IF协会的官方文档。

表5-1是标准A接口的嵌入式主机的测试项，需要说明的是待测产品的USB接口必须是标准A接口；表5-2是Micro-B接口的外围设备的测试项，需要说明的是待测产品的USB接口必须要是Micro-B接口。两个表中用到了这些符号：

√必须要求

\*如果这个功能支持，这项必须要求

\*\*如果存在多个下行端口，这项必须要求

表5-1 标准A接口的嵌入式主机的测试项

USB-IF test▼ USB speed▼	Automated Test Ch6	Manual Test Ch7	Drop/Droop	DS LS SQT	DS FS SQT	DS HS Electrical
High Speed Host	√	√	√/**	*	*	√
Full Speed Host	√	√	√/**	*	*	
LOW Speed Host	√	√	√/**	√		

表5-2 Micro-B接口的外围设备的测试项

USB-IF test▼ USB speed▼	IOP Goldtree	Avg Current	USBCV	Back-Voltage	US HS Electrical	US FS SQT	Inrush	Automatd Test Ch6
FS B	√	√	√	√		√	√	√
HS B	√	√	√	√	√	√	√	√

测试设备和附件（包括测试线材、外设等）在送到测试实验室之前，需要准备一些文档，如USB兼容性测试清单（CheckList）和目标外设列表（Target Peripheral List，TPL）等文档，兼容性清单文档的模

板可以从USB-IF的网站上  
( [http://www.usb.org/developers/compliance/check\\_list/](http://www.usb.org/developers/compliance/check_list/) ) 下载。

需要完成的清单如下：

- Compliance Checklist for USB On-The-Go and Embedded Host Supplement Revision 2.0;
- USB Compliance Checklist Peripheral Silicon(Excluding Hub Silicon);
- USB Compliance Checklist Peripherals(Excluding Hubs);
- USB Compliance Checklist Systems.

目标主机不需要支持所有类型的USB外设。每个目标主机的制造商必须声明主机将支持哪些外设，并提供这些外设的列表，被称为目标主机的“目标外设列表”(TPL)。目标外设列表应准确地描述目标主机所支持设备的类别。目标外设列表仅列出目标主机支持设备类的几个典型外设，目标主机实际支持的外设可能比所声明的目标外设列表更多，TPL(目标外设列表)如下：

- TPL Form for USB On-The-Go and Embedded Host Supplement Revision 2.0;
- TPL Hub Form for USB On-The-Go and Embedded Host Supplement Revision 2.0.

最后在送到实验室测试之前，还需提供一个送测设备的操作说明文档，方便实验室测试人员测试送测设备，避免因测试方法不对而引起错误。

## **5.2.3 送USB授权实验室进行测试**

测试实验室是经过USB-IF授权，提供USB与连接器相关的兼容性测试服务与技术咨询的实验室。USB-IF希望测试实验室能够为厂商提供最实时的服务，来减少测试与认证所需时间。下面对测试实验室测试周期和送样数目进行简单介绍。

### **1. 测试周期**

每次测试周期为4~30个工作日不等，视产品而定。

### **2. 测试样品数量**

至少需提供1个测试样品，通常为避免因单机问题而造成测试失败，尽可能提供2个测试样品给测试实验室。

## 5.2.4 获得合格报告及证书

当实验室完成USB的兼容性测试后，会收到一个测试报告。如果测试通过，这个设备就会被列入 USB 整合厂商清单中，并且这款产品会获得一个协会测试号。通过这个协会测试号，可以在USB-IF协会网站上查到这个测试设备的相关信息，并有权使用USB 标志。如图5-2所示是恩智浦公司的一个产品的协会测试号。



图5-2 产品的协会测试号

## 5.3 典型的测试用例

兼容性测试包括很多测试用例，这里不能将所有的测试用例一一说明，我们将会选取一些典型的、容易导致测试失败的用例进行说明，所有测试用例的列表可以从USB官方网站下载得到。

### 5.3.1 不受支持的设备信息

对于不支持的设备插入测试主机时，测试主机应显示一个明确的不支持该设备的错误消息，如一个不支持打印机设备的测试主机连接了一个USB打印机设备时，测试主机应明确地显示一个“Unsupported device”的错误消息。

值得注意的是，如果测试主机不支持集线器，测试主机应该显示一个明确的错误消息来表明不支持集线器，而不是一个不受支持的设备。如不支持集线器的测试主机连接了一个集线器时，主机应显示一个“Unsupported Hub”的错误消息。

### 5.3.2 最大集线器层数

USB 2.0协议规定一个USB主机最多支持5层集线器级连。对于支持集线器的测试主机，需要测试其支持的最大集线器层数次，测试要求当集线器层数超过主机所支持集线器的最大层数时，测试主机应显示相应的错误消息。例如，支持最大五级集线器的测试主机，当再连接一个第六级集线器时，测试主机需要显示一个明确的类似于“Error : exceeding the maximum tier of hubs”的错误消息。

### 5.3.3 电源预算指示

当将大功率设备连接到总线供电的集线器的下游端口时，如果该设备的电流超过了主机的电流预算，测试主机应明确地显示合适的错误消息来指示电流超过预算，从而确保支持集线器的目标主机能够检测并防止总线供电的集线器上的过电流事件。例如，支持总线供电的集线器，其下游端口连接了一个高功率的设备并发生电流过流，此时应打印出“Error : over current happen”。

## 5.3.4 PET测试

USB-PET ( USB Protocol and Electrical Tester ) 是测试USB协议和电气的测试仪，用于USB规范中定义的测试及电池充电的测试。PET测试仪硬件是与USB-IF、USB OTG工作组和电池充电工作组合作设计的。本书并不涉及USB-PET 兼容性测试过程，如果读者想进一步了解USB-PET 测试过程，请参考其他书籍或者USB协会的官方文档。

USB-PET 测试本身就是基于协议标准来做的测试，所以 USB 协议栈并不需要为支持USB-PET增加新的功能。但是因为USB-PET测试设备供应商识别码（Vendor ID，VID）为0x1A0A 和产品识别码（Product ID，PID）为0x0200且没有提供明确的类和子类的信息，所以USB协议栈需要支持上述供应商识别码和产品识别码的设备的枚举。

### 5.3.5 USB测试模式

USB测试模式是一种高速模式下所特有的模式，在这个模式下USB不再正常工作，只是单纯地发送特定的信号，一般用作电器信号测试。在EHCI兼容的USB主机控制器上可以通过设置端口状态和控制寄存器（Port Status and Control Registers，PORTSC）寄存器的PTC位域来进入测试模式。PORTSC寄存器的描述如图5-3所示。

关于PTC寄存器位域的描述如表5-3所示，当PTC被写0x1后总线就进入J状态，如果PTC被写0x4，控制器就会向USB总线发送高速测试包（Test Packet），也就是我们通常说的高速眼图模式。

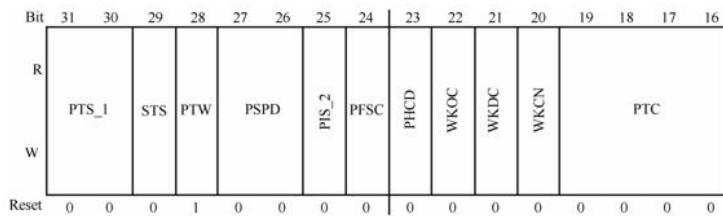


图5-3 PORTSC寄存器的描述

表5-3 PTC寄存器位域的描述

位	位域	描述
19-16	PTC	<p>端口测试控制 除0之外的任何值都表示端口在测试模式下运行。 注意：FORCE_ENABLE_HS 和 FORCE_ENABLE_LS 设置是测试模式的扩展支持EHCI规范。将PTC字段写入任何FORCE_ENABLE值强制端口以选定的速度进入连接和启用状态。然后清除PTC字段允许端口状态机从该点正常进行。</p> <p>4'b0000 不使能端口测试 4'b0001 J_STATE 4'b0010 K_STATE 4'b0011 SE0_NAK 4'b0100 Test_Packet 4'b0101 FORCE_ENABLE_HS 4'b0110 FORCE_ENABLE_FS 4'b0111 FORCE_ENABLE_LS 其他保留</p>

外围设备和嵌入式主机测试采用了两种不同的办法让被测设备进入测试模式。

#### 1. 外围设备进入测试模式

下面是USB测试设备作为外围设备（Peripheral Device）要进入测试模式的流程。首先按图5-4所示方法把测试设备连接到PC主机上。

再在PC主机上运行HSETT（High-Speed Electrical Test Tool）这个应用程序，HSETT可以从（[http://www.usb.org/developers/compliance/electrical\\_tests/](http://www.usb.org/developers/compliance/electrical_tests/) # usbhset）网站下载到相对应的测试程序版本，测试项界面如图5-5所示。

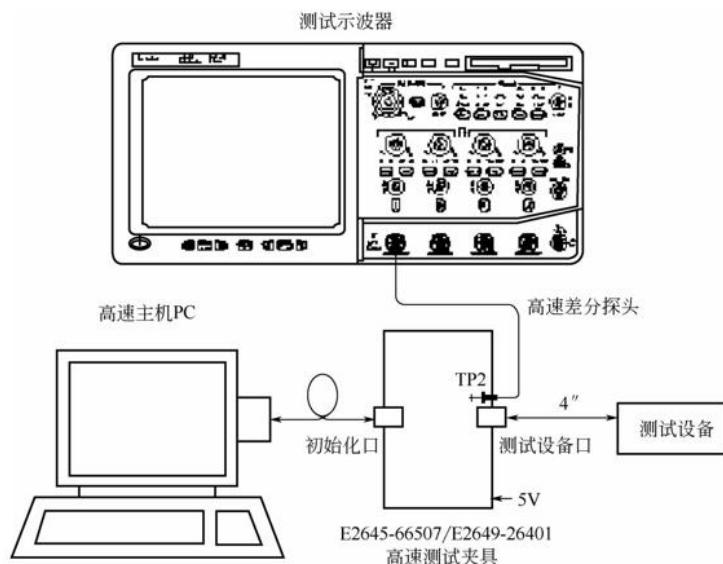


图5-4 外设信号测试环境

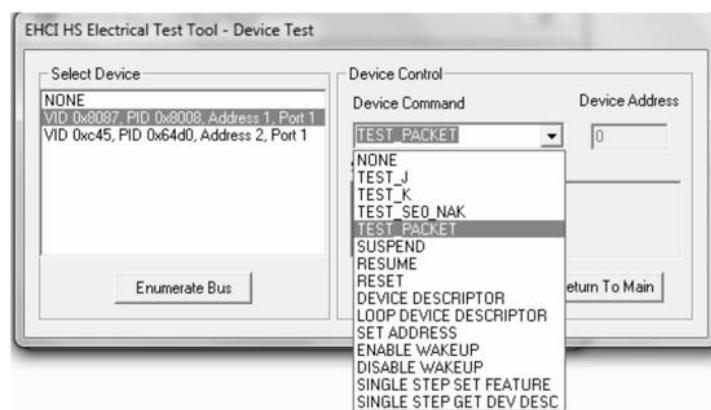


图5-5 测试项界面

从图5-5中可以看出，可以通过HSETT的界面选择不同的测试模式，当选择不同的测试项后，与待测设备的主机会向待测设备发送使能下行端口特性请求 SetPortFeature ( PORT\_Test ) 让设备进入所指定的测试模式，当测试设备收到 SetPortFeature ( PORT\_Test ) 命令后，会对 5.2 节提到的 PORTSC寄存器中PTC位域写入相应的值，使得USB控制器进入相应的测试模式。

如图5-5所示，当选择“TEST PACKET”后，待测设备的PORTSC寄存器中 PTC 位域会写进相应的值 ( 0x4 )，使得 USB 控制器进入相应的高速眼图模式。

需要支持测试模式的USB协议栈都应该实现SetPortFeature ( ) 来支持高速信号测试。

## 2.嵌入式主机进入测试模式

嵌入式主机进入测试模式的流程与外围设备进入测试模式的流程有所不同。USB 定义了一组特殊供应商识别码（固定为 0x1A0A）/产品识别码，当嵌入式主机检测到这些特殊的供应商识别码/产品识别码时，就会根据表5-4来进入相应的测试模式。例如，主机检测到供应商识别码为0x1A0A 和产品识别码为 0x0101 的测试设备时，会进入 Test\_SE0\_NAK的测试模式。

表5-4 USB主机测试模式列表

PID	测试模式
0x0101	Test_SE0_NAK
0x0102	Test_J
0x0103	Test_K
0x0104	Test_Packet
0x0105	保留
0x0106	HS_HOST_PORT_SUSPEND_RESUME
0x0107	SINGLE_STEP_GET_DEV_DESC
0x0108	SINGLE_STEP_GET_DEV_DESC_DATA

### 5.3.6 USB特殊单步测试用例

为了在测试过程中让示波器能够捕获特殊的高速信号以完成相关的高速信号电气特性测试，USB 协会定义了两种针对嵌入式主机的单步测试用例。

#### 1. 单步获取设备描述符

当主机枚举供应商识别码为0x1A0A且产品识别码为 0x0107的设备时，测试流程如图5-6所示。

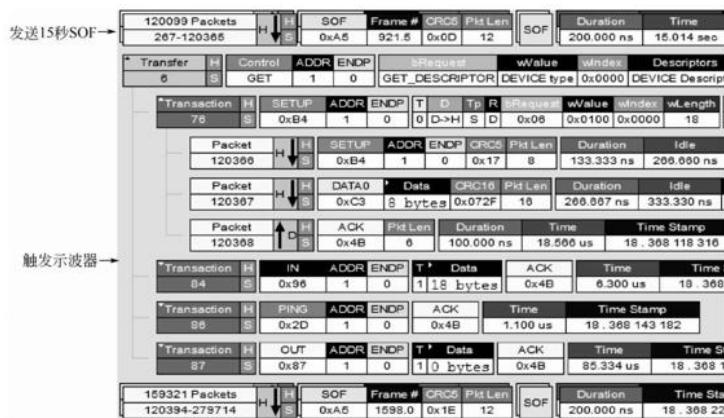


图5-6 单步获取设备描述符流程

(1) 主机枚举测试设备，读取设备供应商识别码（0x1A0A）和产品识别码（0x0107），然后完成其设备的枚举过程。

(2) 主机发出帧开始包（SOF）并持续 15s（Packet 267-120365），这时测试工程师将示波器的触发电压的范围升到帧开始电压水平以上一点点。

(3) 主机发送获取设备描述符的请求GetDescriptor（Device）。

(4) 当设备向主机返回Setup包ACK包（Packet 120368），此时触发示波器。这时主机继续发送IN事务（Transaction 84）和OUT事务（Transaction 87），完成整个设备描述符请求的传输。

要支持这个测试项，只要在主机完成供应商识别码（0x1A0A）和产品识别码（0x0107）设备的枚举后，做一个 15s 的延时再发一个 GetDescriptor（Device）命令给测试设备即可。

## 2. 单步获取设备描述符数据

当主机获取设备的供应商识别码（0x1A0A）和产品识别码（0x0108）时，测试流程如图5-7所示。



图5-7 单步获取设备描述符数据流程

(1) 主机枚举测试设备，读取供应商识别码（0x1A0A）和产品识别码（0x0108），然后完成其设备的枚举过程。

(2) 枚举设备后，主机发送获取设备描述符请求 GetDescriptor（Device）。

(3) 设备在收到USB主机发送来的Setup包（Packet 368）后向主机返回第一个ACK包（Packet 370）。

(4) 主机发出帧开始包（SOF）并持续 15s（Packet 371-120370），这时测试工程师将示波器的触发电压的范围升到帧开始电压水平以上一点点。

(5) 主机向设备发送IN包（Packet 120371）。

(6) 设备响应 IN 包，并使用数据包（Packet 120372）向主机发送设备描述符数据，此时触发示波器。

(7) 主机接收到IN数据包后，向设备发送ACK包（Packet 120373）。这时主机继续发送OUT事务（Transaction 78），完成整个设备描述符请求的传输。

下面简单介绍一下基于 EHCI 主机控制器实现单步获取设备描述符的数据传输过程：

完整的获取设备描述符的传输（Transfer）包括三个事务（Transaction）：Setup事务、IN事务和OUT状态事务。在3.1.3节详细讨论了如何使用QH和qTD将相关的事务连接到异步传输链表上。为了降低开销，在标准的流程中会一次性把三个qTD一起准备（Prime）好，并只设置一个IOC（Interrupt On Complete），当三个事务都完成了以后，处理器收到一个传输完成的中断消息，这时候整个标准获取设备描述符数据传输过程完成。标准获取设备描述符数据传输过程如图5-8所示。

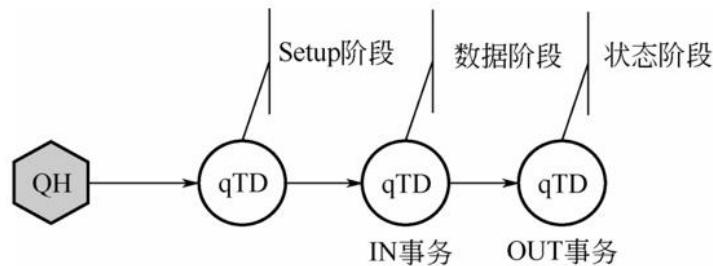


图5-8 标准获取设备描述符数据传输过程

对于单步获取设备描述符数据的传输（Transfer），需在 Setup 事务完成后延迟15s，让系统发送帧开始，再准备IN事务和OUT状态事务，这就需要先准备 Setup 事务的 qTD，当 Setup 事务完成后，再等待 15s 后再准备IN事务和OUT状态事务qTD。单步获取设备描述符数据传输过程如图5-9所示。

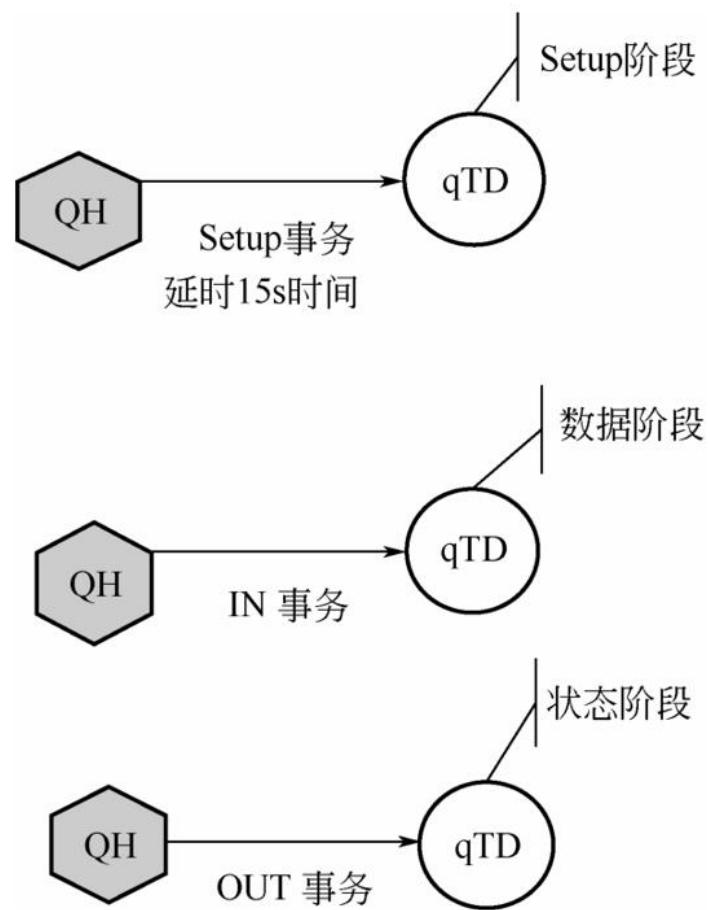


图5-9 单步获取设备描述符数据传输过程

## 第6章

# USB Type-C及供电协议

随着USB在各个领域中被广泛普及和应用，人们对USB不断提出新的需求，USB技术的发展也日新月异，USB Type-C接口则是USB组织最新推出的一种接口，与之前普遍使用的Type-A、Type-B、Mini-B和Micro-B接口相比，该接口具有以下新的特性：

- 从外形的角度来说它更小、更薄，更有利于产品的系统集成和外观设计；
- 统一的接口，USB主机和USB设备使用相同的接口；
- 支持正反插；
- 支持供电协议（Power Delivery，PD），能够实现动态、灵活地充放电管理及进行动态角色切换；
- 支持更多的可扩展性功能。

## 6.1 Type-C

在USB Type-C接口出现之前，USB规范所定义的接口与其功能是相互对应的，每一种接口一般都对应了一类特定的功能；同时每一种接口的正反面在外观和机械特性上都有明显的区别，这就使得正反插不可能实现。

USB Type-C接口完全颠覆了之前的USB接口的定义，其统一定义的接口使得接口不再与功能一一对应，同时也加入了对正反插的支持，使得USB的使用更为方便快捷。

## 6.1.1 接口定义

Type-C接口包含24个引脚，图6-1描述了Type-C接口和接头引脚。全功能的Type-C线缆会包含两个Type-C接头，线缆里面有22根导线连接两端的接头。两端接头的B6和B7之间没有线缆进行连接。图6-2是Type-C接口（左）和接头（右）的实物图。当插头正插入接口时，接头的A1~A12连接接口的A1~A12，接头的B1~B12连接接口的B1~B12；当插头反插入接口时，接头的A1~A12连接接口的B1~B12，接头的B1~B12连接接口的A1~A12。

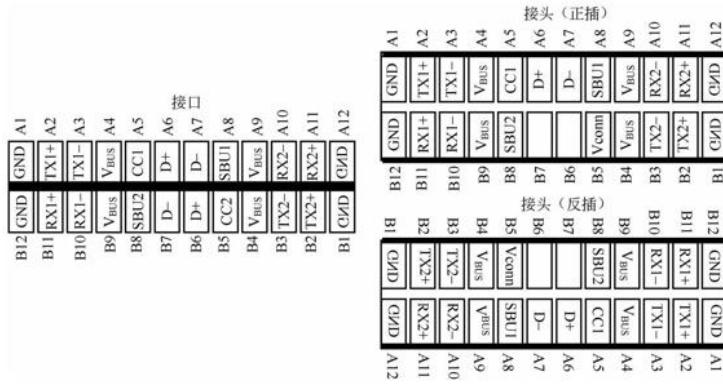


图6-1 Type-C接口和接头引脚

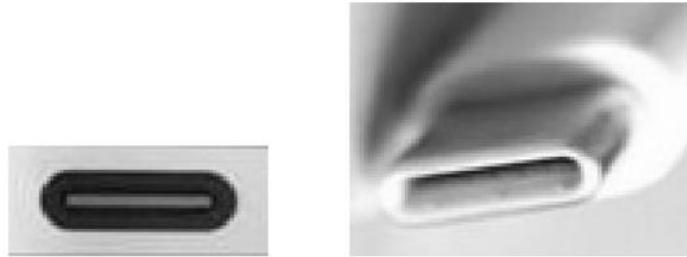


图6-2 Type-C接口（左）和接头（右）实物

表6-1是Type-C的引脚描述，表6-2是Type-C的引脚分类。24个引脚定义了USB 2.0和USB 3.1数据传输的信号，还定义了CC和SBU辅助信号。它们的功能会在后面进行介绍。

表6-1 Type-C的引脚描述

引脚	名 字	功 能	引脚	名 字	功 能
A1	GND	地	B1	GND	地
A2	TX1+	发送差分信号+	B2	TX2+	发送差分信号+
A3	TX1-	发送差分信号-	B3	TX2-	发送差分信号-
A4	V <sub>BUS</sub>	充放电引脚	B4	V <sub>BUS</sub>	充放电引脚
A5	CC1	配置通道 1	B5	CC2	配置通道 2
A6	D+	USB 2.0 数据	B6	D+	USB 2.0 数据
A7	D-	USB 2.0 数据	B7	D-	USB 2.0 数据
A8	SBU1	Sideband Use	B8	SBU2	Sideband Use
A9	V <sub>BUS</sub>	充放电引脚	B9	V <sub>BUS</sub>	充放电引脚
A10	RX2-	接收差分信号-	B10	RX1-	接收差分信号-
A11	RX2+	接收差分信号+	B11	RX1+	接收差分信号+
A12	GND	地	B12	GND	地

表6-2 Type-C的引脚分类

功 能	引 脚 组	描 述
USB 3.1	A2, A3, B2, B3, A10, A11, B10, B11	USB 3.1 数据传输
USB 2.0	A6/B6 (D+), A7/B7 (D-)	USB 2.0 数据传输
配置	CC1, CC2	接口功能配置, 拔插检测, 供电协议信息传输, Vconn 功能
辅助信号	SBU1, SBU2	辅助信号
供电	V <sub>BUS</sub> , V <sub>CONN</sub> , GND	供电线缆

Type-C接口的机械结构比较复杂，本书不进行具体介绍，详细信息可以参考Type-C规范文档。图6-3是Type-C和Type-A接口尺寸，其简单描述了接口形状和长度高度信息，接口长度是8.34mm，接口高度是2.56mm。与长度为15.7mm、宽度为7.5mm的标准Type-A接口相比较，此种尺寸更利于系统集成和外观设计。

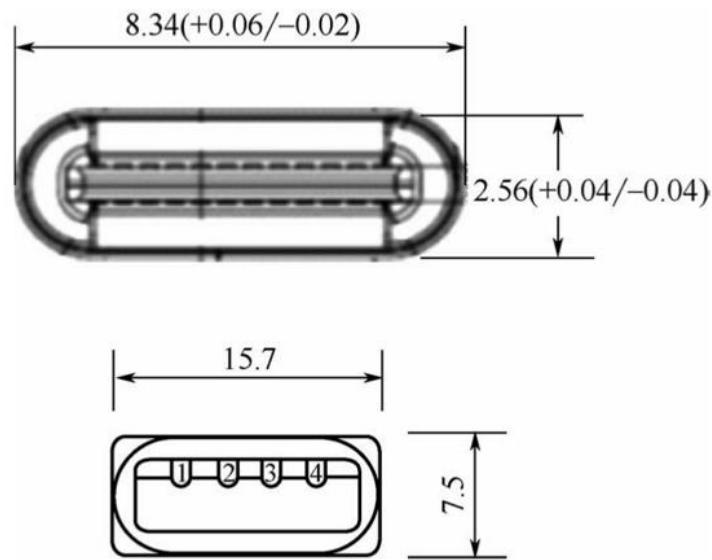


图6-3 Type-C和Type-A接口尺寸

## 6.1.2 角色定义

USB本质上是一个主从结构的总线协议，所以在两个连接的设备能够通信之前，需要先确定这两个设备在通信过程中的角色，是USB主机还是USB设备。在Type-C接口出现之前，当设备的角色确定时，其对外的USB接口也就确定了（OTG除外）。例如，USB下行端口（Downstreaming Face Port，DFP）一般为USB主机，且使用Type-A接口。USB上行端口（Upstreaming Face Port，UFP）一般为USB设备，且使用Type-B、Micro-B或Mini-B接口。这种方式的优点是用户可以通过接口分辨出设备的功能，缺点是需要使用USB转接线缆在不同的接口类型间做转接。新设计的Type-C接口使用统一的接口，使得所有设备之间都能通过标准Type-C线缆进行连接而不需要转接线缆。但是也带来了一个缺点：用户不再能够通过接口类型直接分辨出设备的功能，两个相连的Type-C设备是否能够工作则由这两个相互连接的设备的角色来决定。因此，角色的定义对于Type-C设备来说很重要。

Type-C规范中的DFP、UFP和USB规范中的DFP、UFP不完全相同。在USB规范中一个接口是DFP；则此接口具有USB主机的通信功能；一个接口是UFP，则此接口具有USB设备通信功能。在Type-C规范中一个接口具有DFP功能但不一定具有USB主机通信功能，一个接口具有UFP功能但不一定具有USB设备通信功能。

Type-C规范中定义了两类角色：一类就是我们所熟知的通信功能角色—DFP和UFP。此外，规范中还新增加了另一类供电角色的定义，可以说这个新增加的定义是整个Type-C规范和后续的供电协议（Power Delivery，PD）的核心。

Type-C规范定义了纯供电方（Source-Only）、默认供电方（Source-Default）、纯耗电方（Sink-Only）、默认耗电方（Sink-Default）、可切换的双重角色（DRP Toggling，Dual Role Port Toggling）、双重角色的可供电设备（DRP Sourcing Device）和双重角色的耗电主机（DRP Sinking Host）这七种类型。

- 纯供电方：只能工作为供电方（Source），不可通过供电协议切换成耗电方（Sink）。
- 默认供电方：默认工作为供电方，可通过供电协议切换成耗电方。
- 纯耗电方：只能工作为耗电方，不可通过供电协议切换成供电方。
- 默认耗电方：默认工作为耗电方，可通过供电协议切换成供电方。
- 可切换的双重角色：默认工作为双重角色（DRP），可通过供电协议切换供电角色。
- 双重角色的可供电设备：默认工作为双重角色，可通过供电协议切换供电角色，具有DFP和UFP功能但不能工作为USB主机，如Hub的上行端口（UFP）。
- 双重角色的耗电主机：默认工作为双重角色，可通过供电协议切换供电角色，具有DFP和UFP功能但不能工作为USB设备（device），如Hub的下行端口（DFP）。

为了更方便读者的理解，本节对角色进行了简化，主要通过表6-3所示的Type-C接口角色进行介绍。

表6-3 Type-C接口角色

供电角色	连接初始通信功能角色	描述
供电方 (Source)	下行端口（DFP）	接口进行供电
耗电方 (Sink)	上行端口（UFP）	接口进行充电
双重角色（DRP） (Dual Role Port)	双重角色 (DFP 或者 UFP)	接口既可以是供电方，也可以是耗电方。在连接上设备之前，此接口角色会在供电方和耗电方之间不断地切换。只有当连接了设备时才能确定。当连接的设备是供电方时，此接口就是耗电方；当连接的设备时耗电方时，此接口就是供电方；当连接的设备也是双重角色时，此接口可能是供电方也可能是耗电方

从供电的角度出发，接口可以是供电方（Source）或者耗电方（Sink）或是双重角色（DRP）。从通信功能的角度出发，接口可以是下行端口（DFP）或者上行端口（UFP）或是双重角色。对于功能角色，它是在连接时根据供电角色确定的，当供电角色是供电方时，则功能角色默认是DFP；当供电角色是耗电方时，则功能角色默认是UFP。

Type-C设备的角色一般在出厂时就已经确定好，表6-4是角色互连表，从中可以看出两个Type-C设备相连是否能够工作。“不工作”指设备相连之后，双方都不能识别对方，不会进行供电，也无法正常使用，但是不会导致设备损坏；“工作”指设备互连之后，双方可以识别对方，并进行供电和通信。从表6-4中可以看出，供电方不能和供电方相互连接工作，耗电方也不能和耗电方相互连接工作。双重角色的设备可以和任何设备相互连接，如果与供电方设备连接，则设备工作为耗电方；如果与耗电方设备连接，则设备工作为供电方。如果与另外一个双重角色设备相连，则设备随机的工作为供电方或耗电方。

表6-4 角色互连表

角 色	供 电 方	耗 电 方	双 重 角 色
供 电 方	不 工 作	工 作	工 作
耗 电 方	工 作	不 工 作	工 作
双 重 角 色	工 作	工 作	工 作

实际上Type-C接口的角色是通过两根CC进行设置和检测的，如图6-4所示。接口1作为供电方，接口的两根CC线上会加上上拉电阻（Rp）；接口2作为耗电方，接口的两根CC线上会加上下拉电阻（Rd）。当两个接口相连时，接口1识别出对方的Rd下拉时，则认为连接成功并且检测出对方的角色是耗电方；当接口2识别出对方的Rp上拉时，则认为连接成功并且检测出对方的角色是供电方。如果两个同是供电方或者耗电方的接口相连，那么双方都不能识别出想要的Rd下拉或者Rp上拉，双方都不能工作。对于双重角色（DRP）接口，两根CC上会在Rp上拉状态和Rd下拉状态这两种状态中不断地切换。当与非双重角色设备连接时，如果对方是Rp上拉，此接口会停在Rd下拉的状态，此时接口的供电角色为耗电方；如果对方是Rd下拉，此接

口会停在 $R_p$ 上拉的状态，此时接口的供电角色为供电方。当与双重角色设备相连时，双方会随机地停在 $R_p$ 上拉状态或 $R_d$ 下拉状态，当一方停在 $R_p$ 上拉状态且另一方停在 $R_d$ 下拉状态时，连接就此建立。停在 $R_p$ 上拉状态的一方是供电方，停在 $R_d$ 下拉状态的一方是耗电方。

不管是供电角色还是通信功能角色，其连接时确定的初始角色并不是始终不变的，可以通过供电协议进行切换，具体的细节会在 6.2 节进行介绍。例如，一个产品被设计为纯供电方并且通信功能是 UFP，因为连接时产品的供电角色只能是供电方，所以连接时通信功能是 DFP，连接之后此产品需要通过供电协议切换通信功能成UFP。

在Type-C规范中，供电方通过 $R_p$ 的阻值告诉耗电方自己默认的供电电流能力，详细信息如表6-5所示。

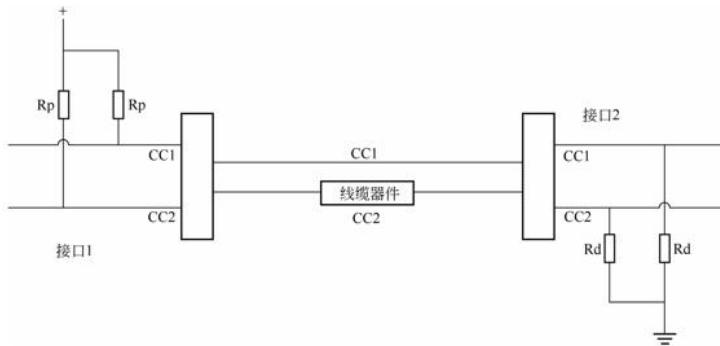


图6-4 Type-C接口相连

表6-5  $R_p$ 的定义

供电电流能力	$R_p$ 值 (电流驱动)	$R_p$ 值 (5V 驱动)	$R_p$ 值 (3.3V 驱动)
900mV	$80 \mu\text{A} \pm 20\%$	$56 \text{k}\Omega \pm 20\%$	$36 \text{k}\Omega \pm 20\%$
1.5A	$180 \mu\text{A} \pm 8\%$	$22 \text{k}\Omega \pm 5\%$	$12 \text{k}\Omega \pm 5\%$
3.0A	$330 \mu\text{A} \pm 8\%$	$10 \text{k}\Omega \pm 5\%$	$4.7 \text{k}\Omega \pm 5\%$

耗电方检测 $R_d$ 上的电压对供电方的供电能力进行识别，详细信息如表6-6和表6-7所示。

表6-6  $R_d$ 的定义

Rd 值

$5.1 \text{ k}\Omega \pm 10\%$

表6-7 检测供电能力

供电电流能力	Rd 电压值
900mV	0.15~0.61V
1.5A	0.70~1.16V
3.0A	1.31~2.04V

### 6.1.3 热拔插检测

Type-C的热拔插检测是通过检测CC1和CC2状态实现的。如图6-4所示，接口1所示Type-C接口是供电方，接口的CC1和CC2上有Rp上拉。接口2所示Type-C接口是耗电方，接口的CC1和CC2上有Rd下拉。正常的Type-C线缆两端的CC1引脚是连接导通的，CC2是断开的（CC2用于实现Vconn供电功能，会在6.1.5节介绍）。

当接口1和接口2通过Type-C线缆相连时，由于正反插情况的存在，接口1会检测到CC1或CC2有下拉电阻，就认为插入了设备。同样，接口2检测到CC1或CC2上有上拉电阻时，就认为插入了设备。随后，接口1会进行状态机的切换，当CC1和CC2的状态稳定并满足状态机时间的要求之后（状态机请参考Type-C规范文档），接口1会提供5V到V<sub>BUS</sub>上，并认为连接成功。同理，接口2的状态机检测到CC1和CC2稳定并V<sub>BUS</sub>有电之后，则认为连接成功。

当接口1和接口2的连接断开时，接口1会检测到CC1或CC2的下拉电阻移除，则认为连接断开。同理，当接口2检测到V<sub>BUS</sub>没有电或者CC1/CC2的上拉电阻移除后，则认为连接断开。

## 6.1.4 正反插检测

Type-C接口和接头上下两面是对称的，这种对称设计可以支持正反插，使得USB在实际的使用中更加方便快捷。正反插的检测也是基于CC1和CC2两个引脚。如图 6-5 所示，当相互连接的设备都是接口并通过一根两头都是接头的Type-C相连时，情况最为复杂，总共有四种Type-C线缆连接情况。当Type-C线缆插入接口1时，线缆的CC1连接到接口1的CC1，这样接口1能检测到 CC1 上有下拉，则认为是正插；相反，线缆的 CC1 连接到接口 1 的CC2，这样接口1能检测到CC2上有下拉，则认为是反插。对于接口2也是同理，检测到CC1上有上拉则认为是正插，检测到CC2上有上拉则认为是反插。

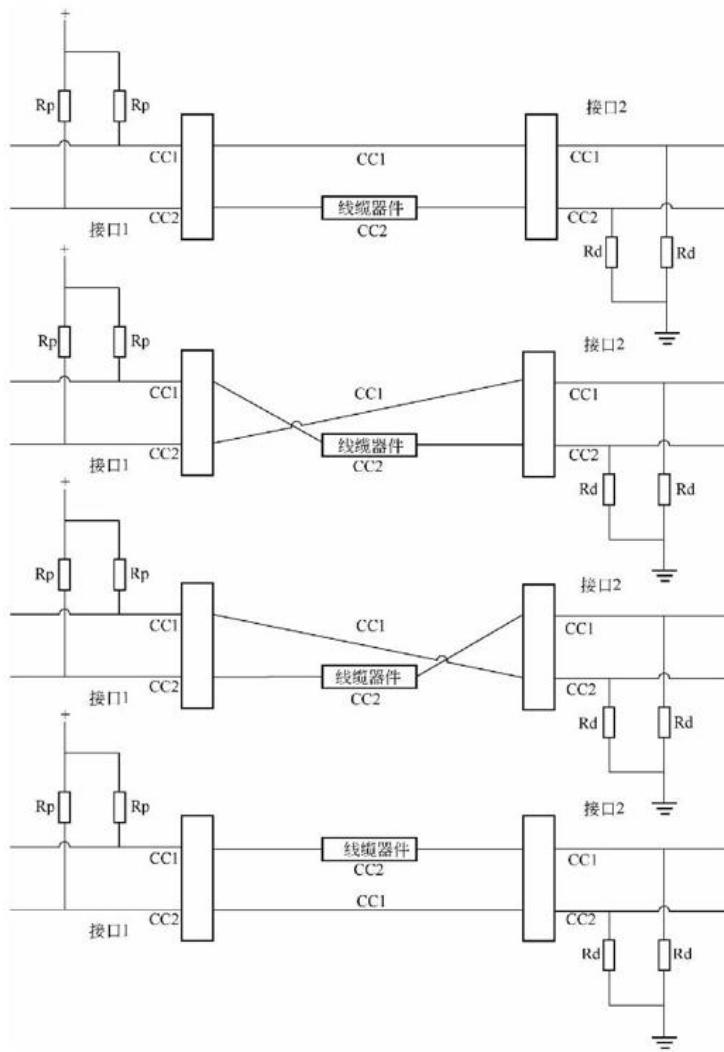


图6-5 正反插检测

检测完正反插之后，系统分配用于传输的数据线。图6-6为Type-c接口连接示例，描述了一根Type-C线缆的两端及USB 3.1数据传输线缆的连接。当正插时，插头和接口的对接是：插头的A1 ~ A12连接接口的A1 ~ A12，插头的B1 ~ B12连接接口的B1 ~ B12。接口使用接口的A2/A3进行USB 3.1信号的发送，使用接口的B10/B11进行USB 3.1信号的接收。当反插时，插头的A1 ~ A12连接接口的B1 ~ B12，插头的B1 ~ B12连接接口的A1 ~ A12。接口使用接口的B2/B3进行USB 3.1信号的发送，使用接口的A10/A11进行USB 3.1信号的接收。

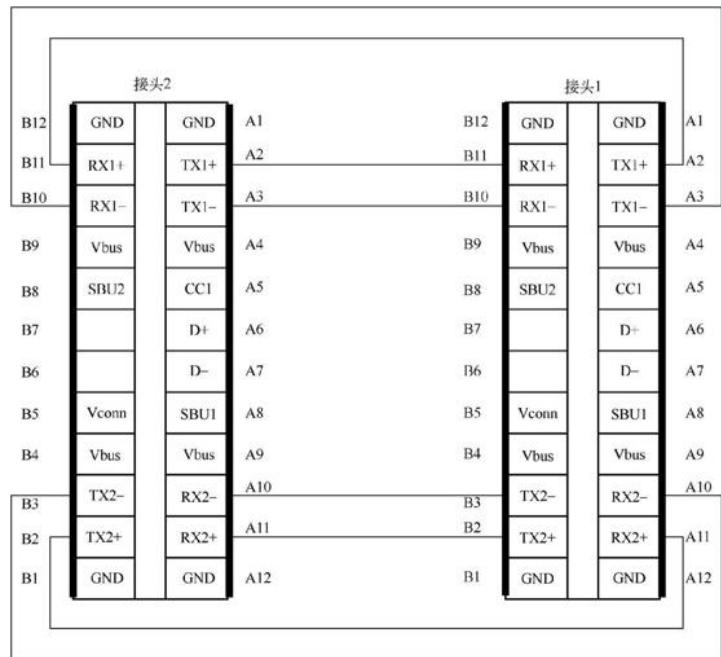


图6-6 Type-C接口连接示例

## 6.1.5 供电功能

在没有供电协议（PD）功能时，Type-C可以提供5V的电压和900mA、1.5A或者3A的电流。如表6-5所示，能提供的最大电流值，也即是电流能力由供电方（source）决定。供电方通过Rp上拉电阻值代表不同的电流能力，当连接上耗电方后，耗电方通过检测Rp的值就能确定供电方能够提供的电流能力。

在有供电协议功能时，Type-C可以提供更高、更灵活的供电能力，最高可提供20V、5A的供电能力，其详细功能会在6.2.2节进行介绍。

在Type-C规范中，线缆比旧有的USB线缆复杂得多，有些线缆里会有元器件进行信号调节、数据转换、辅助功能等功能。这时，对线缆中的元器件供电也需要被考虑进去。如果线缆有被供电的需求，那么这些线缆会在CC2上加一个Ra下拉（ $800\Omega \sim 1.2k\Omega$ ），如图6-7所示，当供电方检测到Ra时，会向CC2进行Vconn供电，电压恒定为5V。

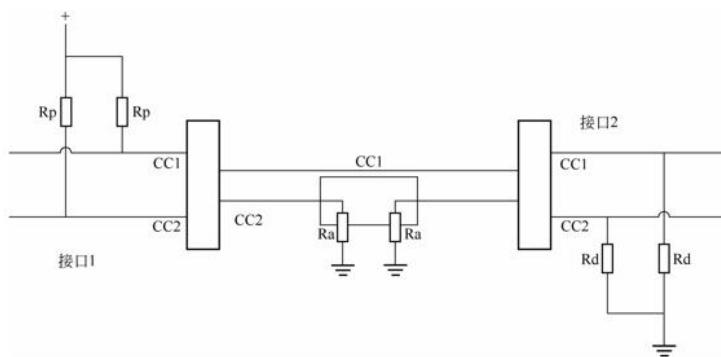


图6-7 Vconn供电

## 6.1.6 附属模式

Type-C 规范定义了 Type-C 接口可以实现两类附属模式（ Accessory Mode ）：音频（ Audio ）附属模式和调试（ Debug ）附属模式。

实现音频附属模式的设备会在CC1和CC2上都进行Ra下拉，当主机检测到连接设备两个CC上都是Ra时则认为连接了Type-C接口的音频设备。音频附属模式把Type-C接口引脚中的一些引脚定义成音频引脚功能，然后通过Type-C接口进行音频信号传输。音频附属模式引脚定义如表6-8所示，对于音频附属模式没有使用的引脚此表没有列出。

表6-8 音频附属模式引脚定义

引脚	Type-C 功能	音频功能	说明
A6/B6	USB 2.0	Right 信号	A6 和 B6 连接在一起
A7/B7	USB 2.0	Left 信号	A7 和 B7 连接在一起
A8	SBU1	Mic/AGND	对于 OMTP 是麦克风功能，对于 CTIA 是接地功能
B8	SBU2	AGND/Mic	对于 CTIA 是麦克风功能，对于 OMTP 是接地功能
A5	CC		连接 Ra 到地
B5	Vconn		连接 Ra 到地

实现调试附属模式的设备会在CC1和CC2上都进行Rd下拉，此处不进行详细介绍。

## 6.2 供电协议

供电协议（Power Delivery）定义了一套信息通信机制，基于此机制直接相连的两个接口扩展进行电压、电流、供电方向、通信功能角色和复用功能（Alternate Mode）等协商。它扩展了Type-C的如下功能：

- 供电动态协商和角色切换；
- 通信功能角色动态切换；
- 复用模式（Alternate Mode）动态协商。

## 6.2.1 供电协议规范定义

### 1.信号定义

供电协议不仅能运行在Type-C接口上，还能运行在Type-A、Type-B接口上，本书只介绍Type-C接口上的供电协议，不考虑其他接口。

当供电协议运行在Type-C 接口上时，供电协议信息通过CC1 线缆进行传输。信号的传输速率为300kbps，最小值是270kbps，最大值为330kbps。信号的编码为BMC（Biphase Mark Coding），8位b01010101经过BMC编码后的情况如图6-8所示。对于需要编码的每个位的起始信号都要发生跳变，如果该位的值为0，则信号保持一位（ $3.33\mu s$ 以300kbps来计算）的时间不变；如果该位的值为1，则在此位时间的中间信号发生跳变。

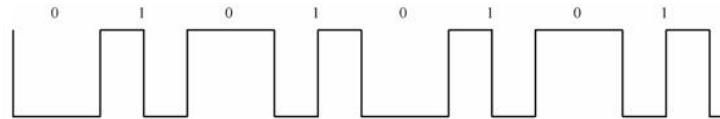


图6-8 BMC编码

供电协议数据编码格式是4b5b，即把4位的数据（0-F）编码成5位，编码表可以参考供电协议规范文档。表6-9提供了Sync-1、Sync-2、Sync-3、RST-1、RST-2和EOP的5位编码。

表6-9 4b5b部分编码

名 称	5b 编码数据
Sync-1	11000
Sync-2	10001
Sync-3	00110

续表

名 称	5b 编码数据
RST-1	00111
RST-2	11001
EOP	01101

通过它们的组合定义了特殊的功能序列，表6-10描述了常用的功能序列。

表6-10 常用的功能序列

名 称	序 列	说 明
Cable Reset	RST-1 Sync-1 RST-1 Sync-3	对线缆进行复位
Hard Reset	RST-1 RST-1 RST-1 RST-2	对接口和线缆进行复位
SOP	Sync-1 Sync-1 Sync-1 Sync-2	数据包开始标识，在发送给接口的信息中使用
SOP*	Sync-1 Sync-1 Sync-3 Sync-3	数据包开始标识，在发送给线缆的信息中使用
SOP**	Sync-1 Sync-3 Sync-1 Sync-3	数据包开始标识，在发送给线缆的信息中使用，如果线缆中有两个供电协议通信单元，则一个通过 SOP 通信另一个通过 SOP**通信

## 2.信息定义

供电协议定义了三类信息：控制信息、数据信息和扩展信息。信息格式如表6-11所示，包含前导码、SOP（包开始）、2字节的信息头、信息数据、CRC和EOP（包结束），前导码以0开始包含64个0和1的序列并且不进行4b5b编码（前导码是32对连续的01序列，并且不进行4b5b编码），SOP\*指SOP、SOP'或SOP”其中的一个，CRC是对信息头和信息数据的4字节校验，EOP是5bit的4b5b编码。

表6-11 信息格式

64 Bits	20 Bits	2 Bytes	n Bytes	4 Bytes	5 Bits
前导码	SOP*	信息头	信息数据	CRC	EOP

需要注意的是，SOP和EOP是4b5b编码后的长度，其他字段都是原始数据的长度（前导码除外），本书其他信息格式也按此种方法描述。

除了信息之外，供电协议还定义了硬复位（Hard Reset）和线缆复位（Cable Reset）信号，其信号格式可以参考表6-12。

表6-12 hard/cable reset信号格式

64 Bits	20 Bits
前导码	hard/cable reset 编码（参考表 6-10）

信息包中的信息头格式如表6-13所示。

表6-13 信息包中的信息头格式

位	名 称	描 述
15	Extended 扩展	此信息是否是扩展类型
14…12	Number of Data Objects 数据个数	后面数据的个数，每个数据是 4 字节 对于非扩展包，此位段的值为 0~7
11…9	MessageID 信息 ID	信息 ID，每发一次信息加 1
8	Port Power Role 接口供电角色	对于 SOP 信息，代表接口的供电角色
	Cable Plug 线缆或接口	对于 SOP' 和 SOP'' 信息，代表此信息是接口发送的还是线缆发送的
7…6	Specification Revision 版本	此信息使用的版本
5	Port Data Role 接口通信功能角色	对于 SOP 信息代表接口的通信功能角色
	Reserved	对于 SOP' 和 SOP'' 保留
4…0	Message Type 信息类型	信息类型

### 1 ) 控制信息

控制信息没有信息数据，只包含 2 字节的信息头，信息头中的数据个数为0。信息头的控制信息类型如表6-14所示。由于供电协议规范的不断更新，本节描述的信息类型可能不全，但不妨碍对供电协议的理解。

表6-14 信息头的控制信息类型

名 称	描 述
GoodCRC	收到信息的回复
GotoMin	供电调节到最小
Accept	接受对方的请求
Reject	拒绝对方的请求
PS_RDY	供电已经准备好
Get_Source_Cap	获取对方的供电能力信息
Get_Sink_Cap	获取对方耗电的需求
DR_Swap	通信功能角色切换
PR_Swap	供电角色切换
VCONN_Swap	Vconn 供电角色切换
Wait	回复对方的请求，让对方稍等再进行请求
Soft_Reset	软复位
Not_Supported	不支持对方发送的请求
Get_Source_Cap_Extended	获取对方的供电相关的扩展信息
Get_Status	获取对方的状态
FR_Swap	快速供电角色切换
Get_PPS_Status	获取供电方的额外信息
Get_Country_Codes	获取本地化信息

## 2 ) 数据信息

数据信息包含信息数据，信息头中的数据个数为1~7个。信息头的数据信息类型如表6-15所示。由于供电协议规范的不断更新，本节描述的信息类型可能不全，但不妨碍对供电协议的理解。

表6-15 信息头的数据信息类型

名 称	描 述
Source_Capabilities	供电方（source）的供电能力信息
Request	请求供电
BIST	自测
Sink_Capabilities	耗电方的耗电需求信息
Battery_Status	电池状态
Alert	警告
Get_Country_Info	获取本地化信息
Vendor_Defined	厂商定义信息

其中厂商自定义信息（Vendor Defined Message，VDM）分为非结构化（Unstructured）信息和结构化（Structured）信息。非结构化信息完全由厂商自己定义，这里不作讨论。而结构化信息会包含4字节的

VDM头信息，它属于信息的一个数据（每个数据是4字节），厂商自定义信息（VDM）格式如表6-16所示。

表6-16 厂商自定义信息（VDM）格式

64 Bits	20 Bits	2 Bytes	4 Bytes	n Bytes	4 Bytes	5 Bits
前导码	SOP	信息头	VDM 头	信息数据	CRC	EOP

厂商自定义信息（VDM）头格式如表6-17所示。

表6-17 厂商自定义信息（VDM）头格式

位	名 称	描 述
31…16	Standard or Vendor ID (SVID)	标准或厂商 ID
15	VDM Type VDM 类型	0: 非结构化 VDM 1: 结构化 VDM
14…13	Structured VDM Version Structured VDM 版本	版本
12…11	保留	

续表

位	名 称	描 述
10…8	Object Position 对象索引	该字段只对 Enter Mode、Exit Mode 和 Attention 命令有效 3'b000: 保留 3'b001~3'b110: 指定目标模式（Mode），目标模式通过 Discover Modes 获取 3'b111: 只能 Exit Mode 用，代表退出所有的模式
7…6	Command Type 命令类型	2'b00: 请求 2'b01: ACK 回复 2'b02: NAK 回复 2'b11: BUSY 回复
5	保留	
4…0	Command 命令	0: 保留 1: Discover Identity, 获取对方的产品信息 2: Discover SVIDs, 获取对方的厂商列表 3: Discover Modes, 获取某个厂商支持的模式 4: Enter Mode, 进入某个模式 5: Exit Mode, 从模式中退出 6: Attention, 通知对方自己的状态有改变 7~15: 保留 16~31: 厂商自己定义的命令

### 3) 扩展信息

扩展信息信息头中的扩展位为1。扩展信息包含扩展头，扩展信息格式如表6-18所示。信息头的扩展信息类型如表6-19所示。由于供电协议规范的不断更新，本节描述的信息类型可能不全，但不妨碍对供电协议的理解。

表6-18 扩展信息格式

64 Bits	20 Bits	2 Bytes	2 Bytes	n Bytes	4 Bytes	5 Bits
前导码	SOP	信息头	扩展信息头	信息数据	CRC	EOP

表6-19 信息头的扩展信息类型

名 称	描 述
Source_Capabilities_Extended	供电方（source）的扩展供电信息
Status	状态信息
Get_Battery_Cap	获取电池能力
Get_Battery_Status	获取电池状态

续表

名 称	描 述
Battery_Capabilities	电池能力信息
Get_Manufacturer_Info	获取厂商信息
Manufacturer_Info	厂商信息
Security_Request	认证请求
Security_Response	认证回复
Firmware_Update_Request	固件更新请求
Firmware_Update_Response	固件更新回复
PPS_Status	供电方信息
Country_Info	本地化信息
Country_Codes	本地化编码

扩展信息头可以参考供电协议规范文档，本书不展开介绍。

### 3.信息通信

图 6-9 是供电协议信息通信流程，发送方发送信息，接收方收到信息并判断信息的CRC是否正确，如果正确则认为接收成功，然后回复GoodCRC信息，发送方收到GoodCRC回复之后认为发送成功。如果超时（0.9 ~ 1.1ms）没有收到 GoodCRC 回复，供电协议规范规定进行两次重试，如果三次都收不到GoodCRC则认为发送失败。

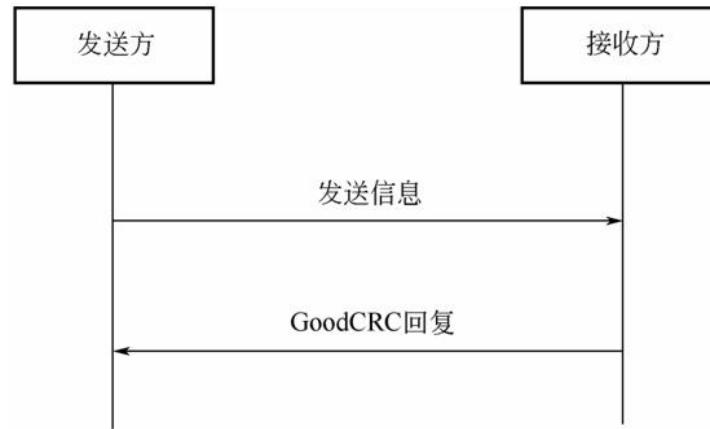


图6-9 供电协议信息通信流程

所有的信息发送都是由发送和GoodCRC回复组成，为了便于分析，本书其他涉及信息发送的流程，不描述 GoodCRC 的回复，默认收到了 GoodCRC 回复并发送成功。

## 6.2.2 供电规则

供电协议规范建议根据供电功率提供如表6-20所示的供电电压和电流。供电协议规范不限制提供其他电压，但必须要提供表6-20定义的电压和电流，并且提供的功率不能超过产品的供电功率。

表6-20 供电规则

供电功率	5V 的电流值	9V 的电流值	15V 的电流值	20V 的电流值
$0.5 \leq x \leq 15$	$x \div 5$	①	①	①
$15 < x \leq 27$	3	$x \div 9$	①	①
$27 < x \leq 45$	3	3	$x \div 15$	①
$45 < x \leq 60$	3	3	3	$x \div 20$
$60 < x \leq 100$	3	3	3	$x \div 20$

注：①不需要提供此电压。

### 6.2.3 供电协商

当Type-C端口连接起始时刻，供电协议协商供电的流程如下（见图6-10）：

（1）当两个Type-C端口通过Type-C线缆连接之后，一端工作为供电方，另一端工作为耗电方。

（2）此时供电方提供的电压是Type-C默认电压5V。耗电方可以通过CC1线缆上的电压获得供电方的默认供电电流能力。

（3）供电方向耗电方发送Source\_Capabilities数据信息（见5.2.1节）。

（4）Source\_Capabilities信息的数据部分描述了供电方的所有供电能力，如5V/3A和9V/3A。

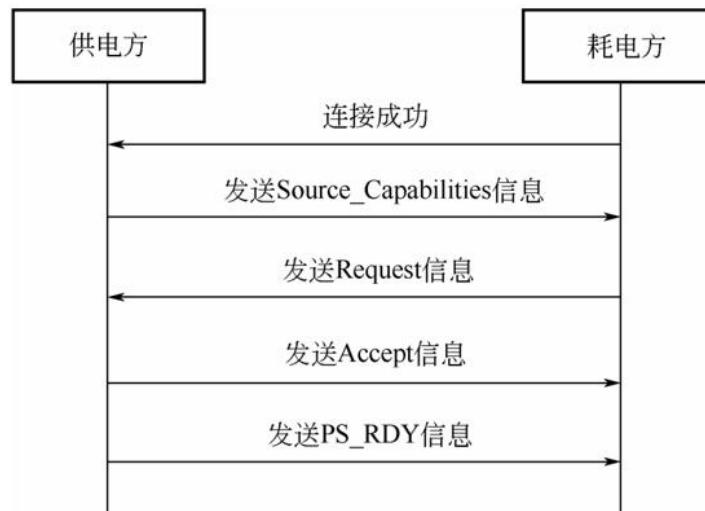


图6-10 初始状态供电协商流程

（5）耗电方收到之后，解析供电方的供电能力，然后请求自己需要的供电。

(6) 例如，耗电方的电源需求是9V/2A，耗电方解析到供电方有9V/3A的供电能力，能满足自己的需求，然后耗电方发送Request信息请求9V/2A的供电。

(7) 供电方收到Request信息之后，会判断是否可以满足对方的请求。

(8) 如果满足，执行第(9)步；如果不满足，执行第(11)步。

(9) 供电方会发送Accept信息，在Accept发送成功之后，供电方操作 $V_{BUS}$ 提供请求的供电，然后发送PS\_Rdy告诉耗电方已提供请求的供电。

(10) 耗电方收到PS\_Rdy之后开始使用请求的供电。

(11) 供电方回复Reject。

在耗电方工作过程中，当耗电方想改变工作供电时，协商的流程如下（见图6-11）：

(1) 耗电方根据前面接收到的Source\_Capabilities，选择其他的电压和电流。



图6-11 耗电方改变供电协商流程

(2) 发送Request信息请求需要的供电。

(3) 供电方收到Request信息之后，会判断是否可以满足对方的请求。

(4) 如果满足，执行第(5)步；如果不满足，执行第(7)步。

(5) 供电方会发送Accept信息，在Accept发送成功之后，供电方操作V<sub>BUS</sub>提供请求的供电，然后发送PS\_Rdy告诉耗电方已提供请求的供电。

(6) 耗电方收到PS\_Rdy之后开始使用供电。

(7) 供电方回复Reject。

在供电方工作过程中，当供电方的供电能力发生了变化时，协商的流程如下（见图6-12）：

(1) 供电方的供电能力发生变化，供电方重新发送Source\_Capabilities信息，包含自己更新后的供电能力。

(2) 耗电方收到之后，解析供电方的供电能力，然后发送Request信息请求自己需要的供电。

(3) 供电方收到Request信息之后，会判断是否可以满足对方的请求。

(4) 如果满足，执行第(5)步；如果不满足，执行第(7)步。

(5) 供电方会发送Accept信息，在Accept发送成功之后，供电方操作V<sub>BUS</sub>提供请求的供电，然后发送PS\_Rdy告诉耗电方已提供请求的供电。

(6) 耗电方收到PS\_Rdy之后开始使用供电。

(7) 供电方回复Reject。

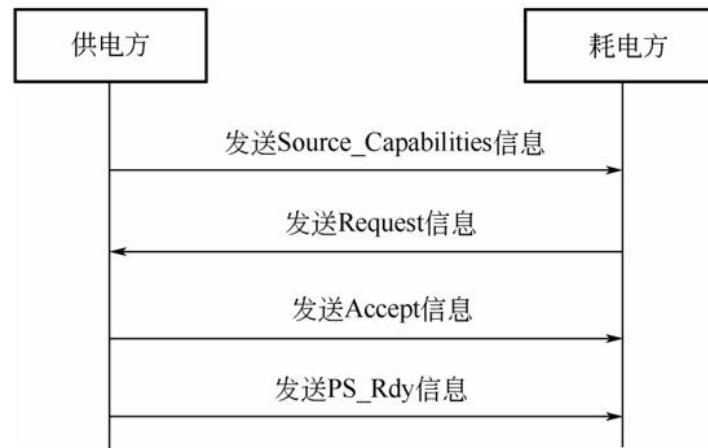


图6-12 供电方供电能力发生改变供电协商流程

## 6.2.4 供电切换

供电角色在端口运行过程中可以动态切换，供电方可以变成耗电方，耗电方可以变成供电方。在供电方工作过程中，供电方想切换成耗电方时，协商的流程如下（见图6-13）：

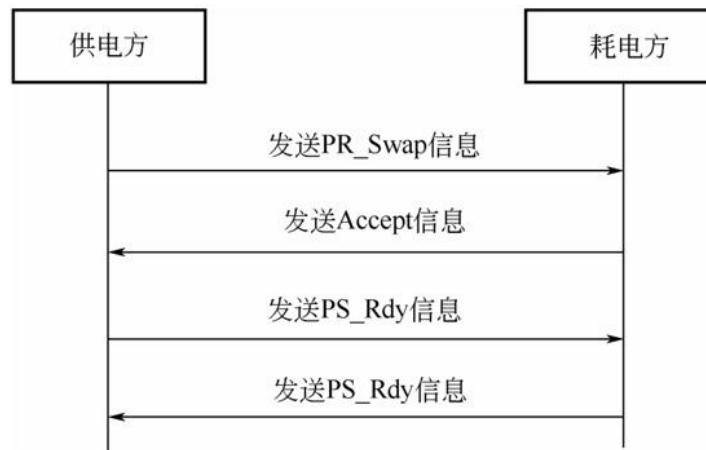


图6-13 供电方切换成耗电方流程

(1) 供电方发送PR\_Swap信息 (power role swap , 供电角色切换) 请求切换供电角色。

(2) 耗电方收到PR\_Swap信息之后，会判断是否允许切换。

(3) 如果耗电方允许切换 (例如，耗电方是有外部供电的，可以给对方供电)，执行第(4)步；如果不允许切换，执行第(8)步。

(4) 耗电方发送Accept信息。

(5) 供电方收到 Accept 信息之后，会关掉自己 V<sub>BUS</sub>上的供电，然后发送PS\_Rdy告诉耗电方自己已经准备好。

(6) 耗电方收到PS\_Rdy之后，会向V<sub>BUS</sub>提供默认5V电压，然后发送PS\_Rdy告诉供电方完成。

(7) 供电方收到 PS\_Rdy 之后，则认为供电角色切换完成。之后供电方变成了耗电方，耗电方变成了供电方。

(8) 耗电方发送 Reject 信息，供电角色切换失败。

在耗电方工作过程中，耗电方想切换成供电方时，协商的流程如下（见图6-14）：

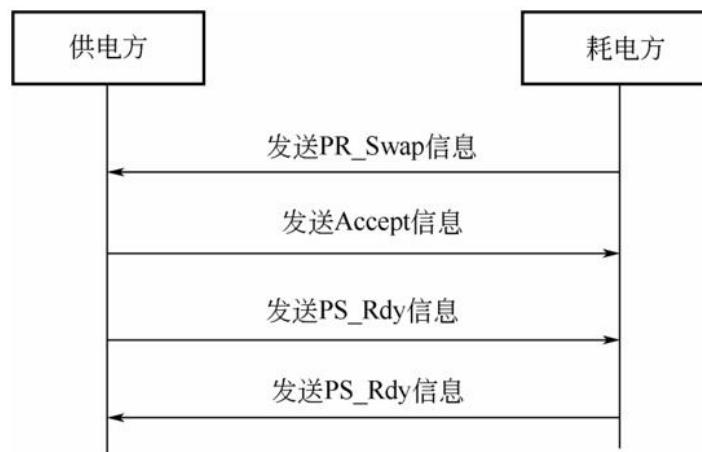


图6-14 耗电方切换成供电方流程

(1) 耗电方发送 PR\_Swap 信息请求切换供电角色。

(2) 供电方收到 PR\_Swap 信息之后，会判断是否允许切换。

(3) 如果供电方允许切换，执行第(4)步；如果不允许切换，执行第(8)步。  
(4) 供电方发送 Accept 信息，耗电方收到 Accept 信息之后，进入等待状态。

(5) 供电方在发送 Accept 成功后，会关掉自己 V<sub>BUS</sub> 上的供电，然后发送 PS\_Rdy 告诉耗电方自己已经准备好。

(6) 耗电方收到 PS\_Rdy 之后，会向 V<sub>BUS</sub> 提供默认 5V 电压，然后发送 PS\_Rdy 告诉供电方完成。

( 7 ) 供电方收到 PS\_Rdy 之后，则认为供电角色切换完成。之后供电方变成了耗电方，耗电方变成了供电方。

( 8 ) 供电方发送Reject信息，供电角色切换失败。

## 6.2.5 通信功能协商

在 Type-C 接口连接上时，如果接口是供电方则 USB 角色为下行端口（DFP），如果接口是耗电方则USB角色为上行端口（UFP）。下行端口工作为USB 主机，上行端口工作为 USB 外设。同样，供电协议提供了角色切换的功能。

当下行端口想切换成上行端口时，协商流程如下（见图6-15）：

- (1) 下行端口发送Dr\_Swap信息请求切换。
- (2) 上行端口收到请求后，判断是否允许切换。
- (3) 如果允许切换，执行第(4)步；如果不允许切换，执行第(5)步。
- (4) 上行端口发送Accept信息，当Accept发送成功后，上行端口变成了下行端口。当下行端口收到Accept后，切换完成，下行端口变成了上行端口。
- (5) 上行端口发送Reject信息。

同理，当上行端口想切换成下行端口时，协商的流程如下（见图6-15）：(1) 上行端口发送Dr\_Swap信息请求切换。



图6-15 通信功能切换

(2) 下行端口收到请求后，判断是否允许切换。

(3) 如果允许切换，执行第(4)步；如果不允许切换，执行第(5)步。

(4) 下行端口发送Accept信息。当Accept发送成功后，下行端口变成了上行端口。当上行端口收到Accept后，切换完成，上行端口变成了下行端口。

(5) 下行端口发送Reject信息。

## 6.2.6 复用模式

通过供电协议协商可以使 Type-C 线缆连接的设备工作在其他复用模式下，如DisplayPort（DP），整个协商流程如下（见图6-16）：

（1）当具有DP功能的设备通过Type-C线缆连接上之后，一端是DP主机，另一端是DP设备。

（2）DP主机如果不是下行端口（DFP），会进行通信角色切换成下行端口。

（3）DP主机发送Discover Identity信息获取连接设备的信息。

（4）DP设备回复Discover Identity ACK信息，包含自己的设备信息。

（5）DP主机发送Discover SVIDs信息获取连接设备支持的厂商SVID列表。

（6）DP设备端回复Discover SVIDs ACK信息，包含自己支持的SVID列表，其中DP的VID（0xFF01）包含在其中。

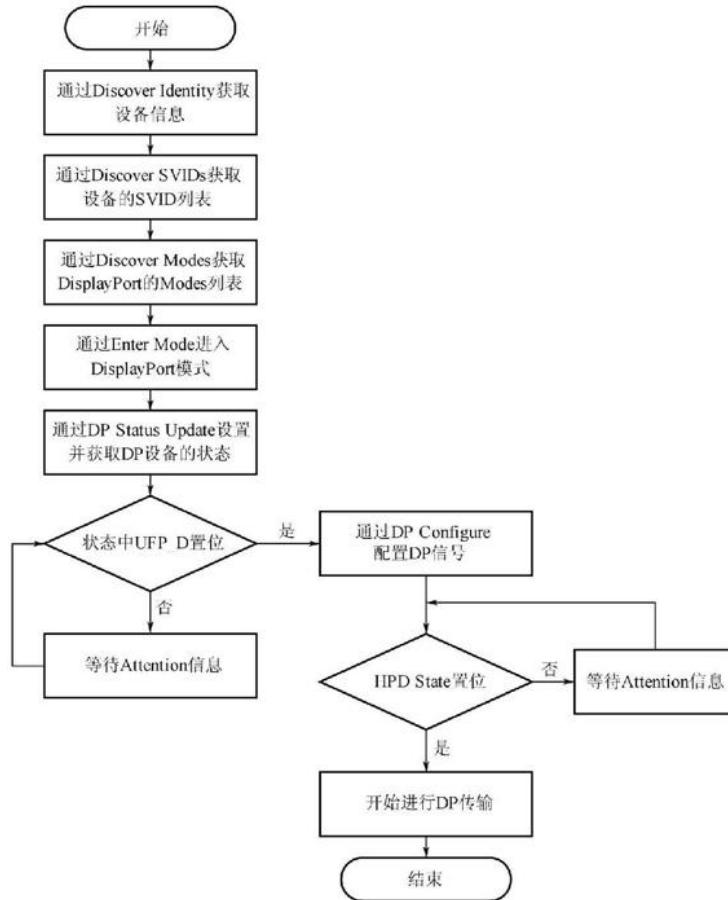


图6-16 DP复用模式协商流程

(7) DP主机收到回复后，发送Discover Modes获取DP设备的模式信息。此信息的数据头使用DP的VID，表示获取支持的DP模式列表。

(8) DP设备回复Discover Modes ACK信息，包含自己的模式列表。

(9) DP主机收到回复后，解析收到的所有模式，并从中选取一个合适的模式，作为Enter Mode的参数发送给对方。

(10) DP设备收到后，进入DP模式，开始工作为DP模式。

(11) DP主机发送Enter Mode成功后，也开始工作为DP模式。

( 12 ) DP主机发送DP Status Update信息设置DP设备的状态。

( 13 ) DP设备回复DP Status Update ACK信息，回复自己的状态。状态中包含DP上行端口功能连接状态 ( UFP\_D ) 和DP显示设备连接状态 ( HPD State )。对于DP适配器，DP上行端口连接状态 ( UFP\_D ) 始终是真。

( 14 ) 如果 DP 上行端口功能 ( UFP\_D ) 已经连接，DP 主机发送 DP Configure配置设备端DP通信信号。如果DP显示设备已经连接，则开始进行DP通信。

( 15 ) 如果DP上行端口功能没有连接，DP主机等待Attention信息报告连接变化。

( 16 ) DP设备会通过Attention通知自己的状态变化。

( 17 ) 当收到Attention指示DP上行端口功能已经连接时，DP主机通过DP Configure配置DP通信信号。

( 18 ) 如果DP显示设备没有连接，DP主机等待Attention信息报告连接变化。

( 19 ) DP设备会通过Attention通知自己的状态变化。

( 20 ) 当收到Attention指示DP显示设备已经连接 ( HPD State ) 时，开始进行DP通信。

DP主机端主动退出DP模式的流程如下（见图6-17）：

( 1 ) 主机端发送DP Configure配置设备端退出DP信号模式。

( 2 ) 主机端发送Exit Mode信息，请求退出DP模式。

( 3 ) 设备端回复Exit Mode ACK，并同意退出。

( 4 ) 主机端收到回复后，退出成功。

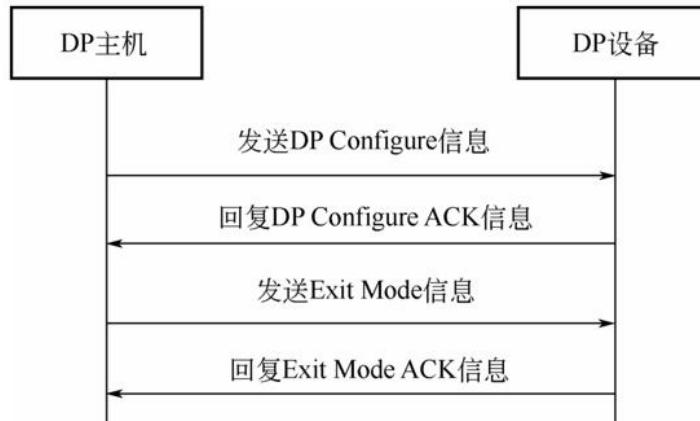


图6-17 退出DP复用模式

DP设备端请求退出DP模式的流程如下（见图6-18）：

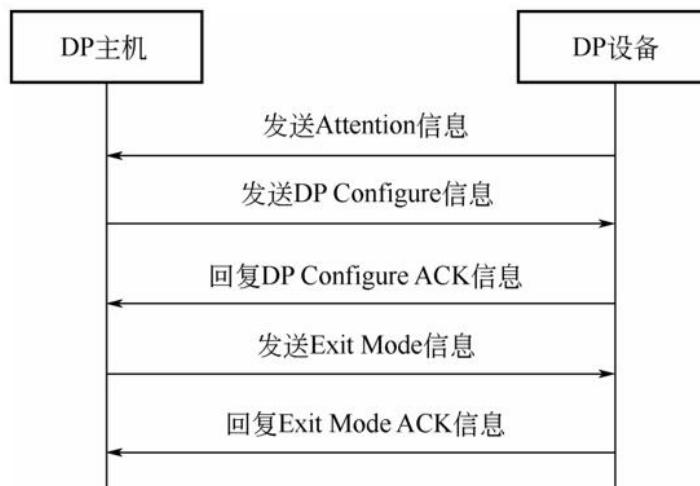


图6-18 退出DP复用模式

（1）设备端发送Attention信息请求退出DP模式。

（2）主机端收到 Attention 后，解析出设备端的退出请求。主机端发送DP Configure配置设备端退出DP信号模式。

（3）主机端发送Exit Mode信息，请求退出DP模式。

（4）设备端回复Exit Mode，并同意退出。

（5）主机端收到回复后，退出成功。

## 6.3 解决方案

### 6.3.1 硬件支持

恩智浦公司提供了基于扩展板（OM13588）的Type-C解决方案，如图6-19所示。此扩展板兼容Arduino<sup>TM</sup>引脚布局，可以插入兼容Arduino<sup>TM</sup>引脚布局主板上实现Type-C和供电协议（PD）功能。恩智浦Kinetis微控制器系列中的FRDM板（如FRDM-KL28Z）和LPC微控制器系列中的一些开发板（LPC54608）兼容Arduino<sup>TM</sup>接口，所以可以使用此扩展板实现SDK供电协议栈。

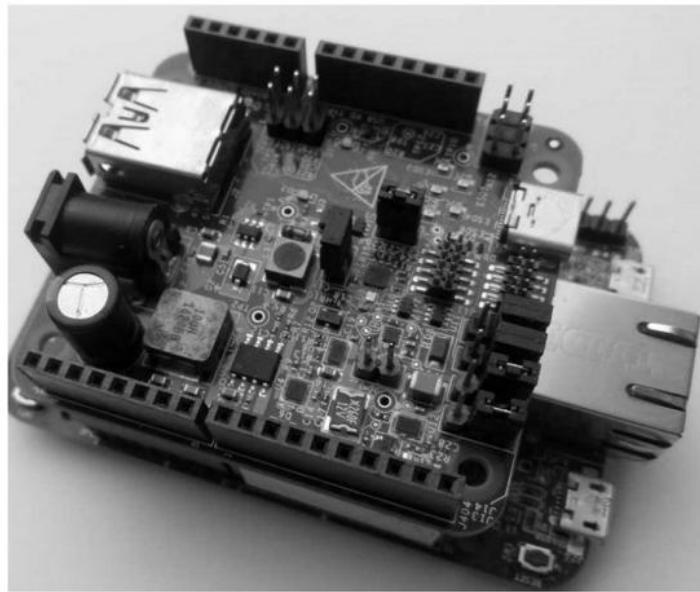


图6-19 Type-C扩展板

Type-C和供电协议的功能主要由此扩展板上的 PTN5110 芯片提供，此芯片兼容USB论坛定义的Type-C控制器接口（Type-C Port Controller Interface , TCPCI）规范。SOC通过I2C接口驱动此芯片实现供电协议和Type-C功能。

## 1. 供电协议栈

MCUXpresso SDK Type-C 供电协议栈实现了Type-C规范和PD 3.0 规范的基本功能。例如，Type-C热拔插检测状态机，供电协议信息发送接收功能。本协议栈提供易用的API和配置方法给用户。用户可以依据自己的需求对协议栈进行配置。

## 2. 协议栈架构

图6-20是MCUXpresso SDK Type-C供电协议栈架构。

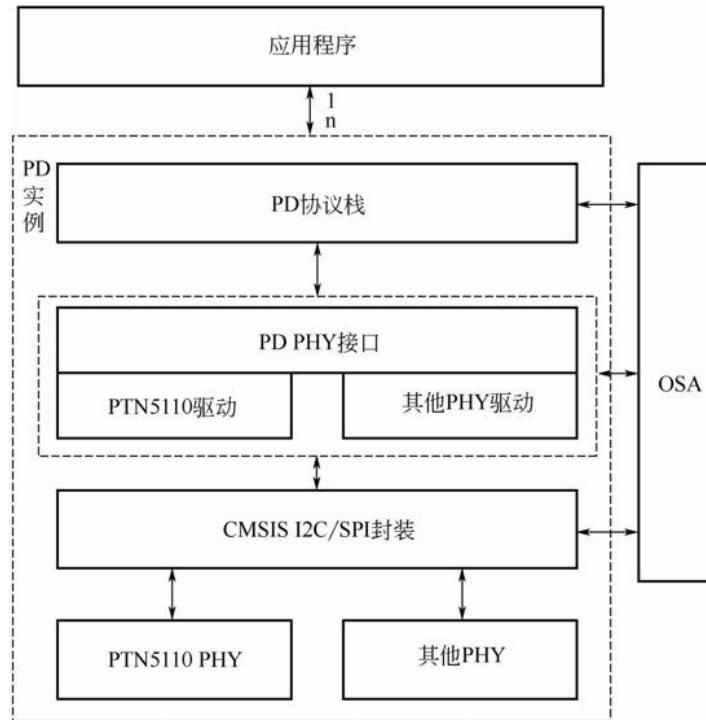


图6-20 MCUXpresso SDK Type-C供电协议栈架构

- **应用程序**：实现供电协议规范中的device policy manager功能。它管理和协商供电，如接受或拒绝请求。

- 供电协议栈：此组件实现了供电协议规范和 Type-C 规范的大部分功能。如供电协议规范的policy engine 和 protocol功能，Type-C规范的热拔插检测功能。它提供协议栈的接口给应用程序。

- PHY接口和PHY驱动：PHY接口对所有PHY是通用接口，每个PHY驱动都提供这个通用接口给供电协议栈，供电协议栈依据这个通用接口操作所有不同的PHY。

- CMSIS I2C/SPI封装：此组件封装了CMSIS I2C和CMSIS SPI，它提供了统一的接口给上层组件。上层组件不需要考虑I2C和SPI的不同，只需要调用统一的接口。

- PHY：Type-C硬件外设。

- RTOS 适配层：为了用一份代码支持不同的RTOSes，此协议栈使用RTOS适配层来封装不同的RTOSes。

需要注意的是，RTOS 适配层不支持在应用程序使用，从应用程序的角度RTOS适配层是不可见的。

### 3. 协议栈接口

供电协议栈API如表6-21所示。

表6-21 供电协议栈API

名 称	描 述
PD_InstanceInit	初始化协议栈实例
PD_InstanceDeinit	复位已初始化的协议栈
PD_Command	触发供电协议功能，比如进行供电角色切换
PD_Control	对供电协议栈进行控制和获取供电协议栈信息，比如获取供电角色
PD_InstanceTask	供电协议栈的任务函数，应用程序需要用此函数创建一个RTOS 任务
PD_PTN5110IsrFunction	PTN5110 中断处理函数，需要关联到 PTN5110 硬件中断上
PD_TimerIsrFunction	需要在 1ms 为间隔的中断中调用此函数

### 4. 协议栈初始化

图6-21是供电协议栈初始化流程，首先初始化Timer和PTN5110中断，然后调用PD\_InstanceInit进行协议栈初始化，最后使能中断和初始化协议栈的任务。

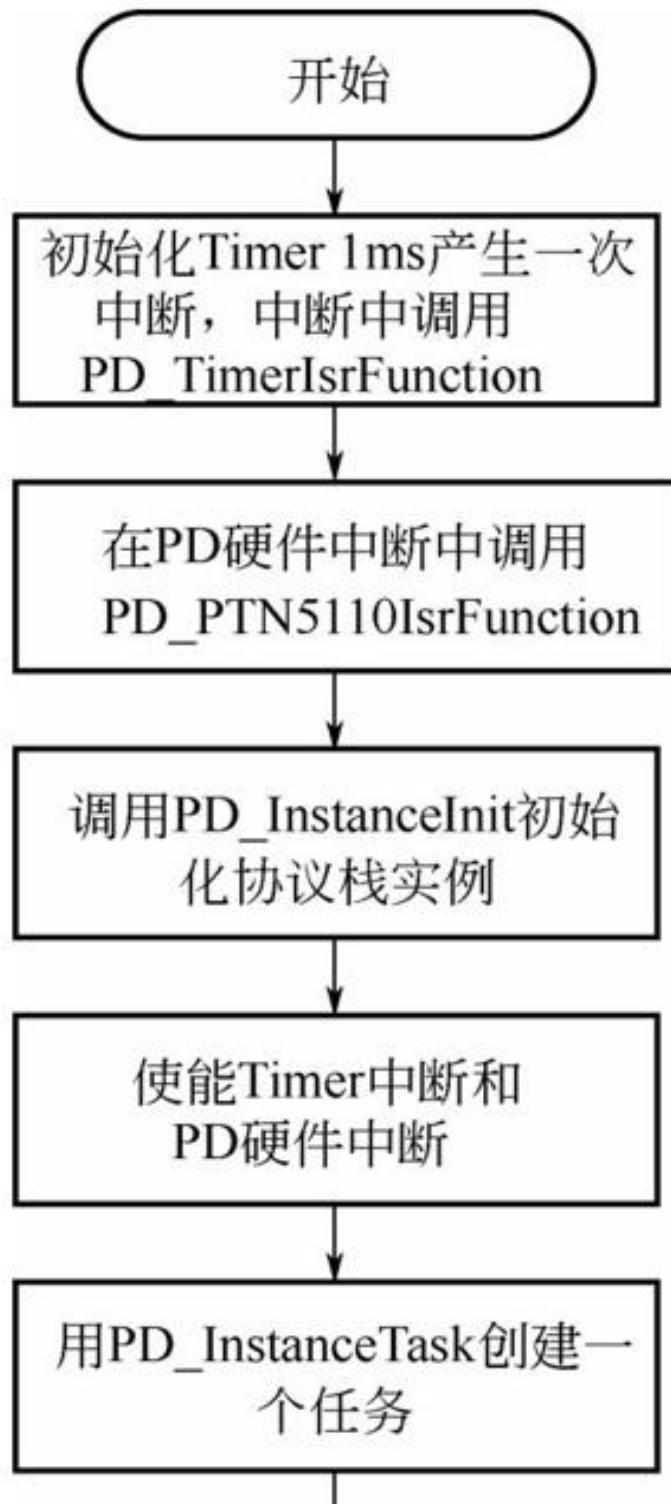




图6-21 供电协议栈初始化流程

## 5. 协议栈运行

在调用 PD\_InstanceInit 时会传入协议栈状态的回调函数和供电操作的回调函数，协议栈会把拔插检测和PD\_Command触发功能的执行流程通过协议栈状态的回调函数通知应用程序。图6-22描述了插入检测后的供电协议栈工作流程。

## 6.3.2 演示程序

MCUXpresso SDK包提供了四个应用程序，以供用户参考。

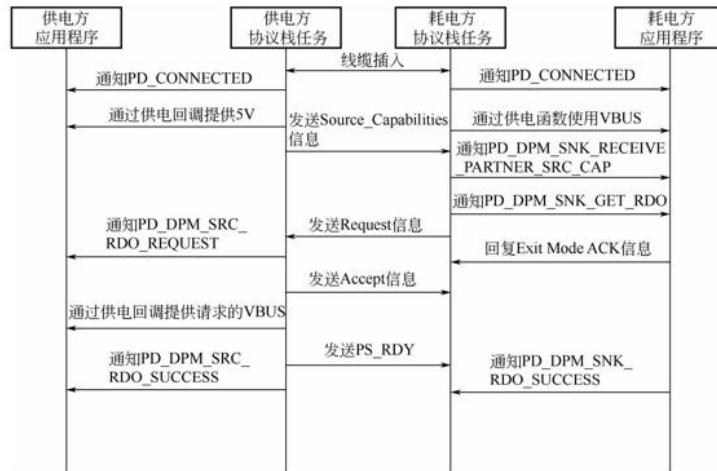


图6-22 供电协议栈工作流程

- usb\_pd：此应用程序不是实际的供电协议栈使用的用户实例，它以菜单和按键的形式向用户展示了供电协议栈所有的功能，用户可以通过此应用程序了解供电协议栈。

- usb\_pd\_charger\_battery：此应用程序模拟了一个有电池的产品，此产品既可以给插入的设备充电，也可以从插入的设备供电，如笔记本。当运行此应用程序板子的Type-C接口连接到一个充电器时，此接口会通过供电协议协商充电；当连接到一个耗电设备时，同样通过供电协议协商供电。

- usb\_pd\_sink\_battery：此应用程序模拟了一个只能被充电具有电池的产品。当运行此应用程序板子的Type-C接口连接到一个充电器时，此接口会通过供电协议协商充电。

- usb\_pd\_source\_charger：此应用程序模拟了一个Type-C接口的充电器。当运行此应用程序板子的Type-C接口连接到一个耗电设备时，此接口会通过供电协议协商供电。

本节通过usb\_pd应用程序演示恩智浦解决方案。图6-23是供电协议栈运行连接图，其中展示了两个Type-C接口相连。此扩展板可以提供5V和9V的电压。

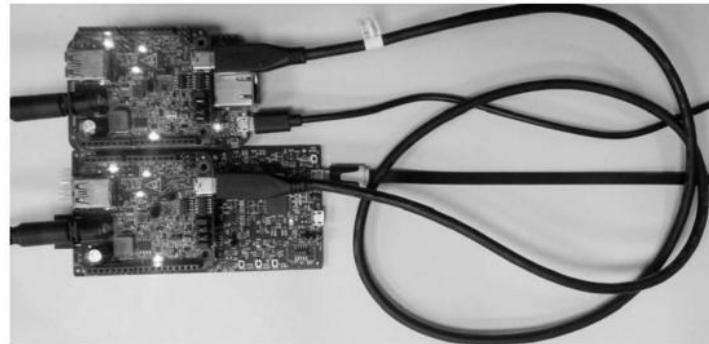


图6-23 供电协议栈运行连接图

(1) 连接之后状态。当通过Type-C线缆连接之后，终端会打印如图6-24所示的信息。连接之后的 $V_{BUS}$ 电压如图6-25所示。



图6-24 连接之后的信息

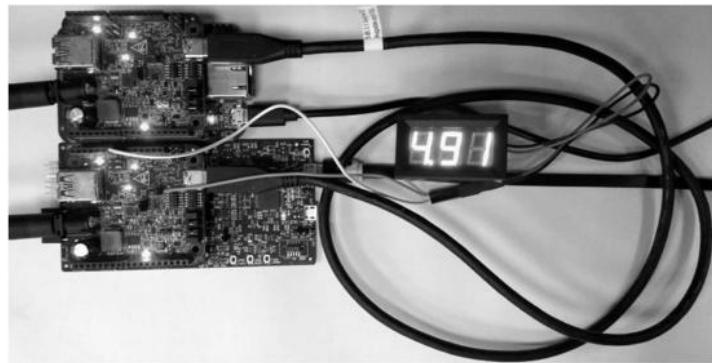


图6-25 连接之后的 $V_{BUS}$ 电压

(2) 菜单。在终端输入“0”之后会显示的菜单如图6-26所示。

```

pd init success
connected, power role:source, data role:DFP, vconn source:yes
partner sink's request 5V success
The menu is as follow for source:
0. print menu
a. source power change
b. goto min
d. hard reset
e. soft reset
f. power role swap
g. data role swap
h. vconn swap
i. get partner sink capabilities
j. standard structured VDM test (only DFP can send enter mode)
k. exit mode (only DFP)
l. send attention
m. test vendor structured VDM
n. test unstructured VDM
u. cable reset (not supported yet)

pd init success
connected, power role:sink, data role:UFP, vconn source:no
receive source capabilities:
1: fixed PDO; vol:5000mV, current:2700mA
2: fixed PDO; vol:9000mV, current:1500mA
sink request 5V success
The menu is as follow for sink:
0. print menu
a. get partner source capabilities
b. request 5V
c. request high voltage
d. hard reset
e. soft reset
f. power role swap
g. data role swap
h. vconn swap
i. get partner sink capabilities
j. standard structured VDM test (only DFP can send enter mode)
l. send attention
m. test vendor structured VDM
n. test unstructured VDM
u. cable reset (not supported yet)

```

图6-26 菜单

(3) 请求9V高电压。在右侧的终端输入“c.request high voltage”，请求高电压的终端信息如图6-27所示，高电压值如图6-28所示。

```

partner sink's request 9V success
u. cable reset (not supported yet)
c. request high voltage
sink request 9V success

```

图6-27 请求高电压的终端信息

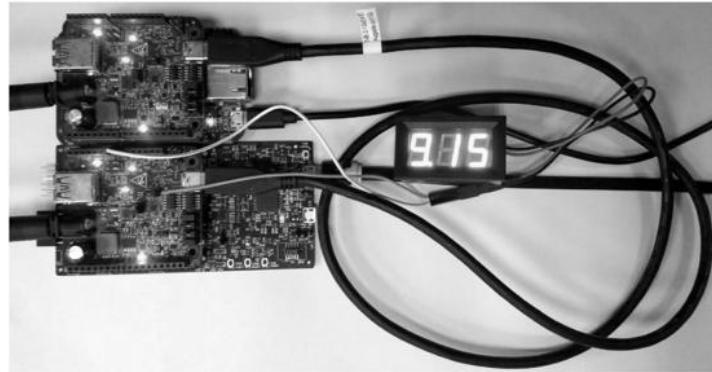


图6-28 高电压值

(4) 供电角色切换。在左侧或右侧输入“f.power role swap”，供电角色切换的终端信息如图6-29所示，电压也会回到默认电压5V。

```

f. power role swap
enter sink
receive source capabilities:
1: fixed PDO; vol:5000mV, current:2700mA
2: fixed PDO; vol:9000mV, current:1500mA
sink request 5V success

u. cable reset (not supported yet)
c. request high voltage
sink request 9V success
enter source
partner sink's request 5V success

```

图6-29 供电角色切换的终端信息

(5) 用户也可以进行其他菜单项的输入，然后观察终端输出和电压变化，从而了解供电协议栈。

## 参考文献

- [1] USB IF,Universal Serial Bus Specification Revision 2.0,2000.
- [2] NXP Semiconductors,K65 Sub-Family Reference Manual,Rev.2,2015.
- [3] USB IF,Interface Association Descriptor Engineering Change Notice to the USB 2.0 specification.
- [4] USB IF,Universal Serial Bus Mass Storage Class Bulk-Only Transport,Revision 1.0,1999.
- [5] USB IF,On-The-Go and Embedded Host Supplement to the USB Revision 2.0 Specification,Revision 2.0 version 1.1a,2012.
- [6] Intel Corporation,Enhanced Host Controller Interface Specification for Universal Serial Bus,Revision 1.0,2002.
- [7] USB IF,Universal Serial Bus Device Class Definition for Audio Devices,Release 1.0,1998.
- [8] USB IF,Universal Serial Bus Device Class Definition for Terminal Types,Release 1.0,1998.
- [9] USB IF,Universal Serial Bus Device Class Definition for Audio Data Formats,Release 1.0,1998.
- [10] USB IF,Universal Serial Bus Device Class Definition for Audio Devices,Release 2.0,2006.
- [11] NXP Semiconductors,UM10912 LPC54S60x/LPC5460x User manual Rev.0.5,15 July 2016.
- [12] USB IF,USB Logo Usage Guidelines,Final,2016.

[13] USB IF,USB On-The-Go and Embedded Host Automated Compliance Plan, Version 1.2, 2012.

[14] Agilent Technologies Inc, Agilent N5416A USB 2.0 Compliance Test Option, Version 03, 2014.

[15] USB IF, Universal Serial Bus Implementers Forum Full and Low Speed Electrical and Interoperability Compliance Test Procedure, Revision 1.3, 2004.

[16] USB 3.0 Promoter Group, Universal Serial Bus Type-C Cable and Connector Specification, Revision 2.1, 2016.

[17] USB IF, USB Power Delivery Specification Revision 2.0, Version 1.2, 2016.

[18] USB IF, USB Power Delivery Specification Revision 3.0, Version 1.0a, 2016.

[19] [www.arduino.cc](http://www.arduino.cc) [EB/OL].

[20] USB IF, USB-Port Controller Specification Rev1.0, v1.2 Final, 2016.