USB 系统研究

USB System Study

(同等学力申请清华大学工学硕士学位论文)

院（系、所）：工 程 物 理 系

专　　业：核 技 术 及 应 用

申　请　人：王 云 飞

指 导 教 师：邵 贝 贝　教 授

2001 年 5 月

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得清华大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签　名：_____日　期：_____

关于论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

(保密论文在解密后应遵守此规定)

签　名：_____导师签名：_____日　期：_____

中文摘要

通用串行总线（USB）是 PC 体系中的一套全新的工业标准，它支持单个主机与多个外设同时进行数据交换。

论文首先会介绍 USB 的体系结构和特点，包括总线特征、协议定义、传输方式和电源管理等等。这部分内容会使 USB 开发者和用户对 USB 有一个整体的认识。

接下来论文会讨论 USB 系统的一般开发方法和技术特点，分设备端硬件、设备端软件和主机端软件三个部分。

然后论文会介绍几个 USB 项目的研发过程和技术细节，包括 USB 手写识别输入系统、USB 通用设备开发平台、USB 安全钥和 USB 在线编程设备等等。论文会详细介绍 USB 设备的硬件和软件开发的技术细节，包括 USB 设备协议栈的编写方法，同时也会讨论在 Windows 98 下开发 USB 内核驱动程序和用户应用程序的一般方法。

论文最后还会介绍 Motorola 的一些相关开发技术，主要是在线编程技术。

论文对广大的 USB 设备开发人员和技术人员具备较高的参考价值，可以帮助他们尽快掌握 USB 设备的特点，以及硬件电路设计和软件编程中的注意事项。

关键字：通用串行总线　微控制器　协议栈　设备驱动程序　在线编程

Abstract

The Universal Serial Bus （USB） is specified to be an industry standard extension to the PC architecture. USB is a serial cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals.

First, the thesis describes the architecture and features of USB, including the bus attributes, the protocol definition, types of transactions, power management, and so on. It tries to give developers and users a snapshot of USB.

After that, the thesis will talk about how to develop the USB system, including: device hardware, device software and host software.

Then the thesis describes some development and technique details of several USB projects: USB tablet, USB univeral development tool, USB security key and USB ISP (In System Program) device. I will give you the details of the hardware and software design of device, including how to write USB device protocol stack, and talk about the kernel drivers and user program in Windows 98.

Finally I will talk more about some technique of Motorola microcontroller, like ISP and so on.

The thesis is very useful to USB device implementers and developers. These can help them rapidly grasp the main feathers and important place in hardware design, programming of the USB device.

Key words:USB microcontroller protocol stack device driver ISP

第一章 引言

1.1 USB 技术

当今的计算机外部设备，都在追求高速度和高通用性。为了满足用户的需求，以 Intel 为首的七家公司（Intel，Compaq，Microsoft，IBM，DEC，Northern Telecom 以及日本 NEC）于 1994 年 11 月推出了 USB（Universal Serial Bus，通用串行总线）协议的第一个草案，专用于低、中速的计算机外设。USB 可把多达 127 个外设同时连到用户的系统上，所有的外设通过协议来共享 USB 的带宽，其 12Mbps 的带宽对于键盘，鼠标等低中速外设是完全足够的。（注：在 2000 年发布的 USB 规范版本 2.0 中，已经将 USB 支持的带宽提升到 480Mbps。）USB 允许外设在主机和其它外设工作时进行连接、配置、使用及移除，即所谓的即插即用（Plug & Play）。同时 USB 总线的应用可以清除 PC 上过多的 I/O 端口，而以一个串行通道取代，使 PC 与外设之间的连接更容易。自从 1996 年 2 月 USB 规范版本 1.0 发布，短短几年间，USB 不光成为了微机主板上的标准端口，而且还成为了所有微机外设（包括键盘、鼠标、显示器、打印机、数字相机、扫描仪和游戏柄等等）与主机相连的标准协议之一。这种连接较以往普通并口（parallel port）和串口（serial port）的连接而言，主要的优点是速度高、功耗低、支持即插即用（Plug & Play）和使用维护方便。

由于 USB2.0 还不成熟，而且本文涉及的项目又全都基于 USB1.1，所以本文以后对 USB 的描述将全部基于 USB 的 1.1 规范版本。凡属于 USB2.0 的内容，都会另外注明。

1.2 项目来源及概述

早在 1998 年，Motorola 公司就要求我们中心（清华大学 Motorola 单片机应用开发研究中心）用 Motorola 公司的 8 位单片机（当时是 68HC05JB4）开发支持 USB 协议的计算机外设。在调研了当时市场上的 USB 设备情况以后，我们选择了与中科院自动化所汉王公司合作开发国内第一套 USB 手写识别输入系统。由于种种原因，汉王公司最终并没有加入到这个项目的开发工作中来，而我独自承担

了所有的开发工作。

这个项目开发成功后，我们中心的 USB 技术实力得到了各方面（包括 Motorola、汉王公司和业内其他开发团队）的认同。这以后我继续用 Motorola 的 8 位单片机（68HC08 系列）为 Motorola 开发了 2 个使用 USB 技术的项目：USB 安全钥和 USB 在线编程设备。同时我也根据技术的发展和个人的爱好研究了使用其他公司的 USB 芯片（如美国国家半导体的 USBN9602）开发 USB 设备的技术和方法，与人合作设计了 USB 通用设备开发平台。

开发一个 USB 设备需要做很多硬件和软件的工作，包括设计 USB 设备的硬件，编写设备协议栈和 Windows 下的驱动程序等等。通过这些工作，我对 USB 协议的整个体系有了一个非常清晰的概念和认识。

本文下面的内容，就将以介绍 USB 协议和项目为主。

第二章 USB 协议

2.1 USB 技术背景

传统的计算机外部设备一般都是使用并口（Parallel Port）和串口（Serial Port）与计算机相连。这两种端口在计算机上使用了很多年，物理层的协议已经相当成熟，但却已经无法满足目前计算机设备不断提高的速度和使用要求。

串口使用的协议是 RS－232 串行通讯标准，适用于设备之间的通讯距离不大于 15 米，传输速率最大为 20KB/s 的场合。它的优点主要是开发方便，几乎所有的硬件开发人员都能很容易地使用串口实现设备与计算机间的通讯。并口使用的是并行通讯协议，在开发和使用上都不是很方便，但它的速度较串口有很大的提高，要达到每秒 1M 字节以上的传输速度并不困难。

这些传统的端口和总线协议都比较简单，一般只对物理层做了一些定义和约束，并没有涉及到设备和计算机的具体通讯行为和网络模型。这使得开发者在开发时有了很大的灵活性，但也带来了兼容性的问题。举例来说，由于串口有很多用于设备和计算机握手的线（如 RTS、CTS 等等），如果设备不需要使用这些握手线，就可以设法利用这些握手线从计算机的电源得到 12V 的直流电，以供设备内部使用。如果某台计算机的主板和电源能提供足够的电流给设备，那么这个串口设备是可以正常工作的；但如果某台计算机的主板和电源不能支持，那么有可能不光是这个串口设备不能使用，连主板和电源也无法正常工作。

以上所提到的种种原因，促使硬件开发商不得不推出一些新型的总线协议，以适应当前计算机和设备的需要。从 20 世纪 90 年代以来，有很多串行总线问世，其中最成功的，便是以 Intel 为首的七家公司于 1994 年推出的 USB（Universal Serial Bus，通用串行总线）协议，以及美国电气和电子工程师协会（IEEE）推出的 IEEE1394 总线协议。

IEEE1394 主要的特点是高速，但它的成本很高，所以目前 IEEE1394 只用于一些专用系统和打印机、扫描仪等数据传输速度要求较高的设备，而不能普遍用于诸如鼠标、键盘等价格较低、速度要求也不高的设备。所以确切地说，IEEE1394 只适于高速的计算机外部设备。

USB 具有速度高、成本低、功耗低、支持即插即用（Plug & Play）和使用维护方便等优点，在协议成熟以后，迅速地占领了计算机低、中速外部设备的市场，大有取代传统串口和并口之势。

2.2 USB 总线优势

总的来说，USB 总线的主要优势体现在以下几个方面：

- 速度
- 总线拓扑体系
- 即插即用
- 低功耗
- 标准接口和外设

本文以下就从这五个方面简单地介绍 USB 的协议内容和特点：

2.2.1 USB 的速度

在不少开发者和使用者的眼中，速度是总线协议的一个最重要的指标，所以媒体在对新技术的宣传过程中，也一直喜欢反复地强调其速度如何如何。在 USB 发展的这几年中，媒体向用户过分强调了 USB 的速度，说它比传统的串口和并口都快。在实际的使用过程中，虽然 USB 较串口在速度上有一定的提高，但它的速度比不上并口。其实在 USB 的规范版本中一早就已指出，USB 适于低、中速设备。

在 USB 的 1.1 规范版本中，USB 支持两种总线数据传输率：一种是在全速（Full Speed）模式下的 12Mbps，另一种是低速模式（Low Speed）下的 1.5Mbps。这两种模式可以同时存在于一个 USB 系统中，而引入低速模式主要是为了降低对速度要求不高的设备的成本，比如鼠标、键盘等等。这里所说的 12M 和 1.5M 只是总线在传输数据时使用的时钟频率，并不是有效数据的实际传输速度。实际上由于软件协议的限制，1.5M 的低速设备所能达到的有效数据传输速度只在几 K 字节/秒左右。

在 USB 的 2.0 规范版本中，USB 支持了一种新的总线数据传输率：在高速（High Speed）模式下的 480Mbps。这个速度较全速和低速设备而言是一个飞跃。之所以要支持这样一个速度，就是因为 USB1.1 无法满足用户对速度的需要。USB1.1 在低、中速设备上是如此成功，直接导致了 USB 开发组希望将它也推广到高速设

备中去。但是总线速度的提高，并不是简单地改改协议书就能实现的。USB1.1 的成功在很大程度上是由于它低廉的成本，而 USB2.0 要想同时实现高速度和低成本，这实际上是非常困难的。USB2.0 已经发布了一年多了，但用户却迟迟看不到成熟的芯片和方案。不少业内人士都担心 USB2.0 会走上 IEEE1394 的老路，在成本降不下来的情况下成为少数专用系统才会采用的方案。

所以如果用户需要开发 USB 设备，目前仍然只能使用 1.5Mbps 和 12Mbps 这两种总线数据传输率。

2.2.2 USB 的总线拓扑体系

与传统的串口和并口不同，USB 支持一个分层的菊花链结构（如图 2.1）。

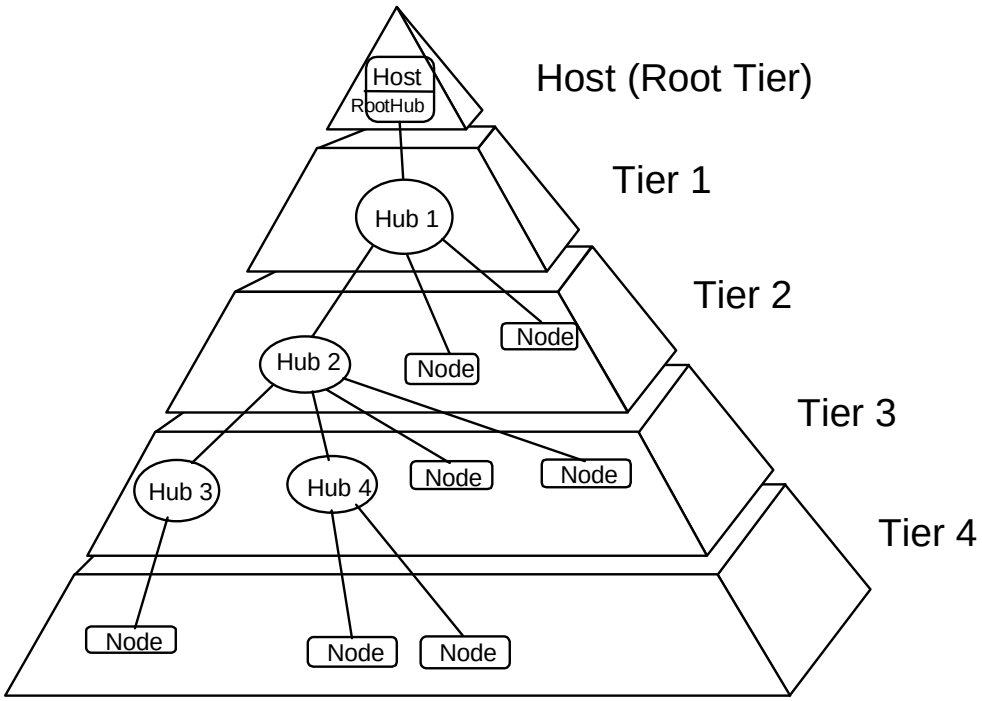


图 2.1 USB 总线拓扑体系

如图所示，整个 USB 总线拓扑体系由三个元素组成：主机（Host）、集线器（Hub）和设备（Node）。Hub 是每个星形结构的中心，用于连接设备和主机。在目前的 PC 应用中，PC 就是主机（Host）和根集线器（Root Hub），用户可以将外设或附加的 Hub 与之相连，这些附加的 Hub 可以连接其他的外设以及下层 Hub。在 USB1.1 规范版本中，USB 在一个拓扑网络中支持最多 4 个 Hub 层以及 127 个

外设。

Hub在USB结构中是一个关键，它提供了附加的USB节点，这些节点被称为端口（Port）。根据端口在总线拓扑体系中的位置和功能的不同，端口又分为上行端口（Upstream Port）和下行端口（Downstream Port）。Hub的上行端口用于与上层Hub或根Hub相连，下行端口则用于与下层设备相连。Hub可以检测出每一个下行端口的状态，并且可以给下层的设备提供电源。图2.2是一个典型的Hub。

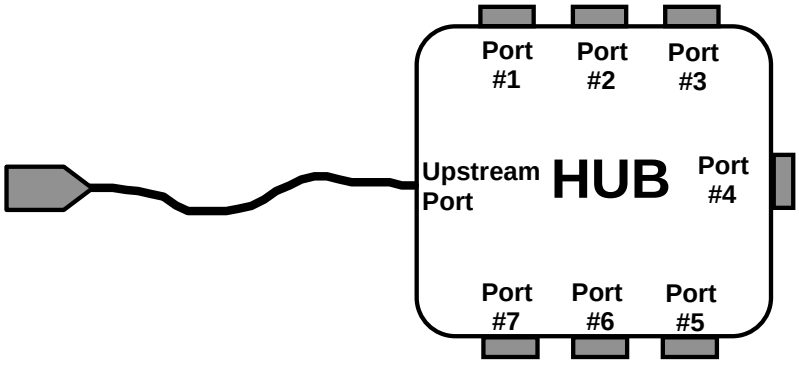


图2.2　典型的USB集线器（Hub）

Hub 可以是独立的硬件设备，也可以嵌入到 USB 设备中去成为普通 USB 设备的一部分，这类设备的特点是除了具有与上层 Hub 相连的一个上行端口以外，还同时提供与下层设备相连的下行端口。

对于目前的计算机应用，127 个外设的上限是完全够用了。不少的开发者甚至希望利用 USB 的总线拓扑体系构建一些简单的网络。在实际的应用中，最能发挥 USB 这一优势的当属办公室网络和家庭网络。在这些网络中，Hub 可以安装在墙上，外形看起来和 Ethernet 或电话线的插座一样，而不同的设备（如打印机、传真机、扫描仪、摄像头等等）可以分布在房间的各处，通过墙上的 Hub 与 PC 相连。用户通过一台 PC 就可以控制房间里的所有设备，而不必将所有的设备都堆放在 PC 周围。

但 USB 总线拓扑体系也不是什么领域都能使用的，它最大的不足在于整个体系必须有一个主机作为控制中心，所有设备对传输介质的使用都必须得到主机的授权，而且设备只能与主机通讯，并不能进行设备间的通讯。这一点决定了 USB 总线并不能用于普通的设备间或主机间的联网。国外有几家公司开发出了一种

USB 连接器，可以将两台 PC 通过 USB 总线对接。它实际上是在设备内部将两个 USB 系统相连，起到一个桥梁的作用，并没有从本质上改变 USB 的总线拓扑体系。

2.2.3 USB 的即插即用

所谓即插即用（Plug & Play），主要包含两个方面的内容：一是热插拔，一是自动配置。热插拔决定于物理层的实现，而自动配置则主要依靠软件协议。

众所周知，传统的串口并不支持热插拔，因为串口在热插拔时会产生很强的电流，容易烧毁硬件电路或接口芯片。USB 在这方面做了很多努力，使得热插拔在 USB 设备成为可能。

USB采用四线电缆来传输信号与电源，如图2.3所示。

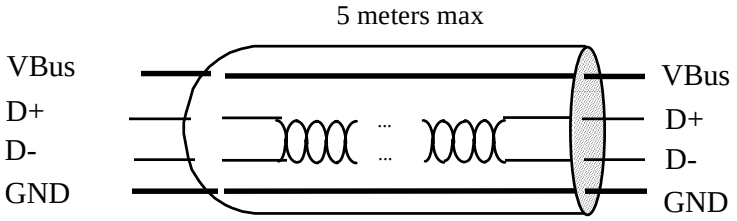


图2.3 USB电缆定义

其中 D＋和 D－是一对差模的信号线，使用 3.3V 的电平，而 VBus 和 GND 则提供了＋5V 的电源。而 USB 正是在电缆和连接点的设计上做了处理，才使得热插拔产生的强电流可以被安全地吸收

至于自动配置，主要是指设备在插入 Hub 的下行端口后能够被主机自动识别并进行信息交换，最终使设备在整个 USB 网络中可以正常工作。这一功能主要依靠 USB 的总线枚举（Bus Enumeration）过程来实现。

首先，USB 总线应该能够自动探测出新设备的插入。图 2.4 是 USB 设备与 Hub 的连接示意图：

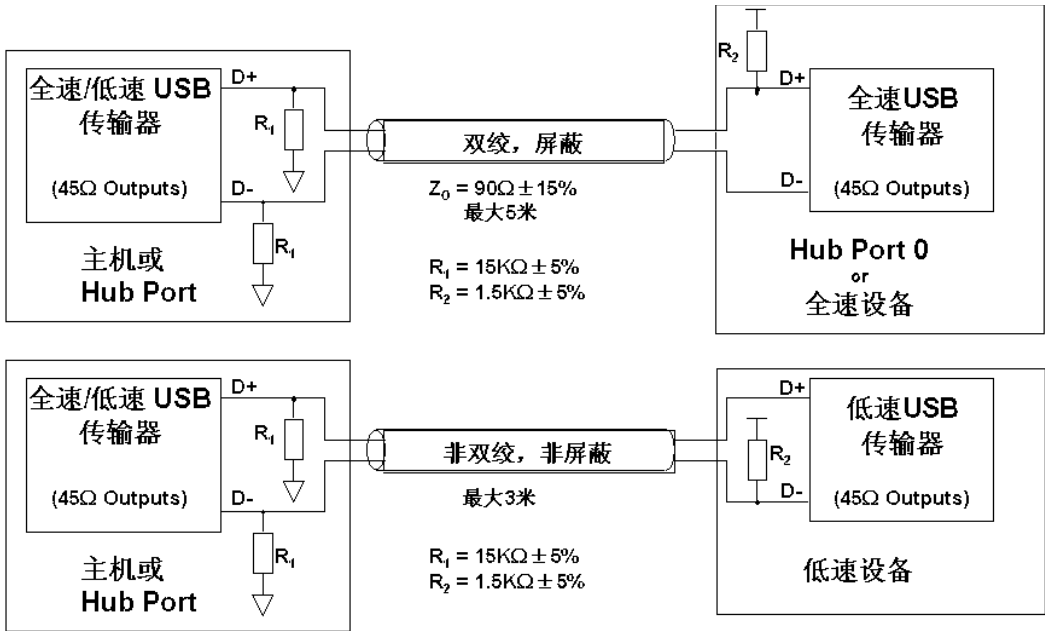
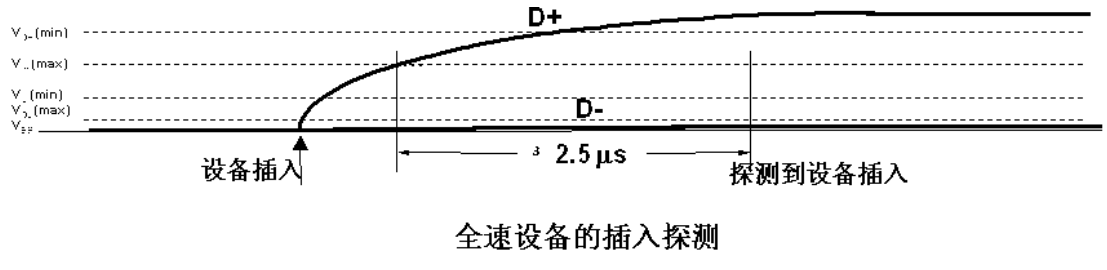
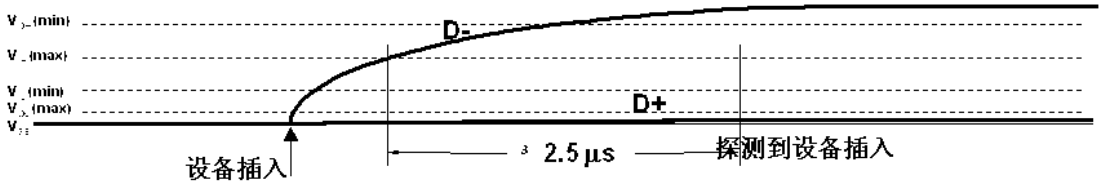


图 2.4　USB 设备连接示意图

如图所示，主机或 Hub 的下行端口（图左）的 D+与 D－都用 15KΩ 电阻接地，使得在没有设备插入的时候，D+与 D－上的电平始终为低。全速设备的上行端口的 D+通过 1.5KΩ 电阻接到 3.3V，而低速设备的上行端口的 D－通过 1.5KΩ 电阻也接到 3.3V，使得在设备插入主机或 Hub 的时候，D+或 D－上会产生一个上升沿，如图 2.5。



全速设备的插入探测



低速设备的插入探测

图 2.5　USB 设备的插入探测

主机或 Hub 就依靠这个上升沿判断出设备的插入和设备的类型（全速还是低速），并开始与设备通讯。具体的步骤如下：

a) 设备插入后自动复位，并将自己的地址设置为 0；

b) 主机向地址为 0 的设备发送命令，请求设备描述符；

c) 设备发回设备描述符；

d) 主机向地址为 0 的设备发送命令，分配新地址（不为 0）；

e) 设备存储新地址；

f) 主机向新分配地址的设备请求设备描述符；

g) 设备发回设备描述符；

h) 主机请求配置和报告描述符；

i) 设备发回所有描述符；

j) 总线枚举过程结束，设备开始正常工作。

在这个过程中，主机会为设备分配包括地址、数据堆栈等一系列的资源，并装载相应的驱动程序。

同样的，在设备拔出的过程中，PC 和 Hub 也会检测到 D＋或 D－上电平的变化（由高到低），然后收回为设备分配的资源，并卸载相应的驱动程序。

有了即插即用这一特点，USB 设备可以随时插入和拔出，在使用上极其方便，这是 USB 最吸引普通用户的地方。

2.2.4 USB 的低功耗

USB 设备的供电方式有两种：自供电（Self-powered）和总线供电（Bus-powered）。

所谓自供电就是由设备自己提供电源，设备不需要从 VBus 上取得电流，这类设备的功率不受 USB 协议的限制，设计时只需要将 VBus 用电容连接到 GND 就可以了。

总线供电设备完全从 VBus 上取得电流，它们的功率受 USB 协议的限制，一般不能超过 500mA。总线供电设备有两种工作状态：一是正常工作（Normal）状态，一是挂起（Suspend）状态。USB 协议规定：如果总线供电设备在 3ms 内没有进行总线操作（即总线处于空闲态），设备需要自动进入挂起状态，而挂起的

设备从总线上吸收的电流必须小于 500μA。而实际上协议规定的 500μA 包括了主机端 15KΩ 的电缆终端匹配电阻（见图 2.4）的电流（通常为 220μA），所以对于使用总线电源的设备而言，进入挂起状态通常便意味着设备的总电流功耗不能超过 280μA，这个功耗值是非常低的。

总线供电设备在进入挂起状态以后，可以通过唤醒（Resume）操作恢复到正常工作状态。唤醒操作既可以由主机发送唤醒或复位信号来触发，也可以由设备自行通过远程唤醒操作来完成。所谓远程唤醒是指一个被挂起的 USB 设备发送信号给处于挂起状态的主机，使得主机醒来，处理设备端的突发事件。

2.2.5 USB 的标准接口和外设

USB 协议体系中的外设都是非常标准的，从底层的物理和电气特性，到上层的软件协议、数据通讯，都有明确的定义。USB 将设备和主机都看作不同的对象，而这些对象又是由不同的模块和子模块组成的。就这样，USB 根据设备和主机各部分的功能和实现的不同，将整个 USB 系统划分成了很多的层次和模块。

USB 首先根据设备功能的不同，将设备划分为不同的子类：例如人机接口设备（HID）类，包括鼠标、键盘等与人的交互较多的设备；音频设备（Audio）类，包括音箱、话筒等；还有通讯设备（Communication Device）类，大容量存储器（Mass Storage）类，打印机（Printer）类和图象设备（Imaging）类等等。

在不同的子类中，设备使用不同的通讯协议（包括数据包的格式等等），使用不同的主机端驱动程序。而同一类的设备往往可以共享部分通讯协议和主机端的驱动程序，它们的对象模型和具体的功能模块是相似的。比如同属于 HID 类的鼠标和跟踪球，它们都向 PC 提供鼠标的移动信息，所以它们都有功能相同的坐标捕捉模块，采集意义相同的坐标偏移量（X 和 Y 方向），并以相同格式的数据包发往 PC，而 PC 在收到它们的数据以后，也可以用同一个驱动程序进行处理并实现光标的移动。这样就在同一类设备中实现了软件和硬件资源的共享，大大降低了系统的开销，也减少了开发者的工作量。

同时 USB 又将单一的 USB 设备或主机划分为不同的层次和模块，如图 2.6。图中的黑箭头表示实际的数据流，灰箭头表示逻辑上的数据流。

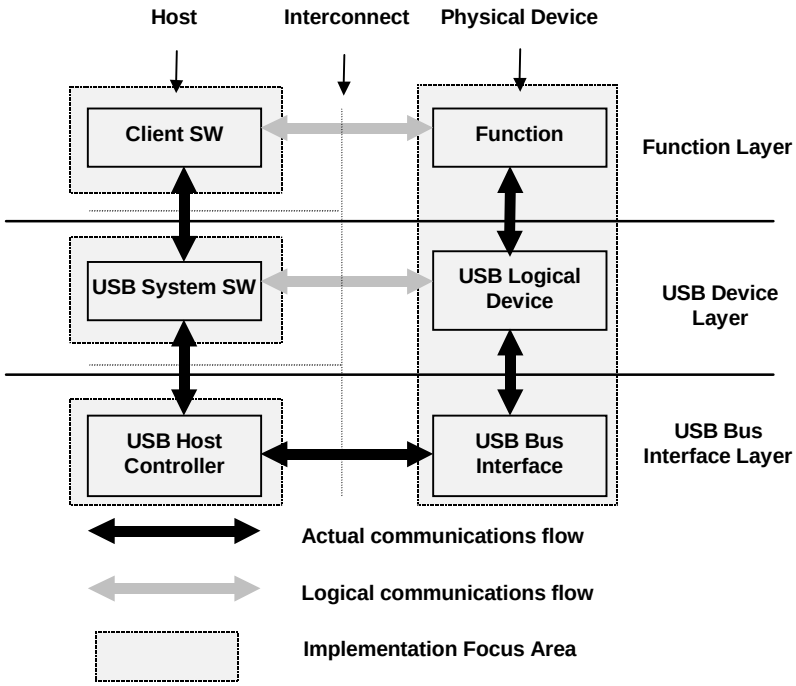


图 2.6　USB 对象模型

设备端虽然分成了总线接口层、设备层和功能层，但在一般的 USB 设备实现中，这 3 层仍然是一个整体，它们之间并没有很明确、很标准的接口定义。

主机端由于使用了大型的操作系统（如 Windows、Linux 等等），又需要同时处理多个 USB 设备的接入，所以不论是在定义上，还是在具体的实现上，都分成了如图所示的主机控制器层、系统软件层、用户软件层。3 层之间的接口是非常标准的，不能由开发者任意更改。

这些复杂的层次和模块划分主要有两个好处：一是可以规范设备和主机的行为，消除不同硬件厂商的区别，减少兼容性的问题；二是可以将 USB 设备开发的工作量细化，硬件工程师可以集中精力设计设备的总线接口层，而软件工程师可以只需要考虑优化用户界面。

Windows 98 就专为 HID 设备编写了一个类驱动程序（对应于主机的系统软件层），用于处理 HID 设备通用的命令和请求，用户只需要编写一个简单的用户界面程序，就可以与自己的 USB 设备进行各种标准的 HID 数据通讯了。也就是说，开发一个 HID 设备，只需要考虑实现 USB 设备和主机的用户界面，而不需要考虑主机的底层硬件和驱动程序，这样开发者的工作量就大大减少了。

但是目前各个类所能获得的支持是不同的，比如 HID 类设备能在 Windows 98 下找到驱动程序，而大容量存储类设备就不能，它只能在 Windows 2000 下才能找到驱动程序。也就是说，虽然 USB 设计的初衷是希望所有的设备在各个层次上都能得到支持，但目前还没能实现，所以建议开发者在设计自己的方案的时候，如果可能的话，尽量采用能够获得的支持较多的方案。

相信随着 USB 的发展，对 USB 各类设备的支持也将越来越完善，到那时开发者只需要在把精力集中在设备的与众不同之处，稍做开发，就能使设备在整个系统中正常工作了。

2.2.6 结论

USB 的技术优势主要就集中在这五个方面。平心而论，虽然 USB 在具体的实现中还有很多不足，但它相比以往的 RS-232 串口而言，的确有了本质的提高。USB 正在不断地占领 PC 外设的市场，成为了 PC 外设的主流接口。在自己的产品中使用 USB，已经成为了一种潮流。如果希望产品被市场接受，开发者往往不得不使用 USB 将设备与 PC 连接。这一点才是最致命的，也才是 USB 受到开发者和用户普遍关注的根本原因。

2.3 USB 软件通讯协议

这里提到的软件通讯协议主要是指设备和主机间的数据流协议以及其他相关协议。

2.3.1 USB 数据流

首先需要提一下端点（endpoint）和管道（pipe）的概念。

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

初始化参数等，端点1、2等一般用作数据端点，存放主机与设备间往来的数据。

- 管道： 一个USB管道是主机端驱动程序的一个数据缓冲区与一个外设端点的连接，它代表了一种在两者之间移动数据的能力。一旦设备被配置，管道就存在了。管道有两种类型，数据流管道（其中的数据没有USB定义的结构）与消息管道（其中的数据必须有USB定义的结构）。管道只是一个逻辑上的概念。

所有的设备必须支持端点0以构筑设备的控制管道。通过控制管道可以获取完全描述USB设备的信息，包括：设备类型、电源管理、配置、端点描述等等。只要设备连接到USB上并且上电，端点0就可以被访问，与之对应的控制管道就存在了。在上文提到的总线枚举过程中，设备就是通过端点0与主机完成各种数据交互的。

下面的图2.7描述了USB数据传输的过程。

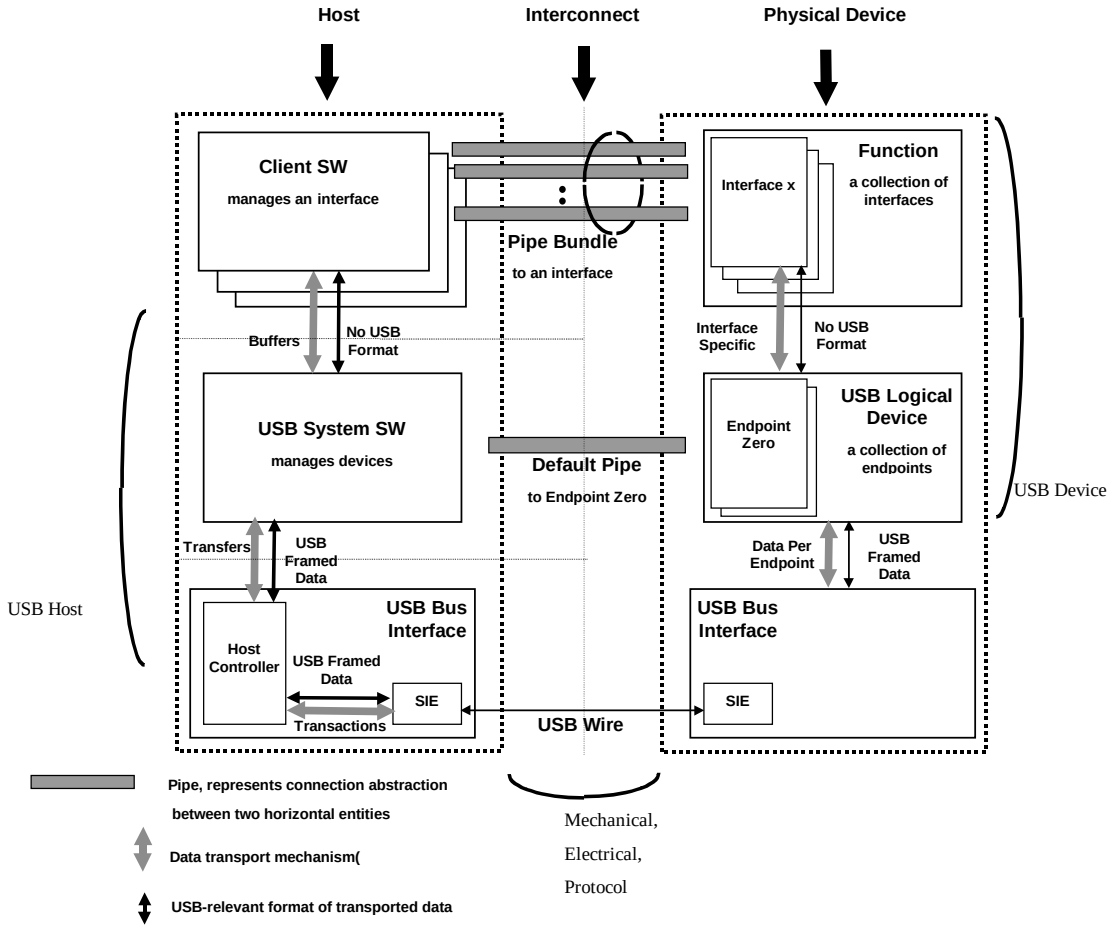


图2.7 USB数据流

从逻辑上讲，USB数据的传输是通过管道进行的。USB系统软件通过缺省管道（与端点0相对应）管理设备，设备驱动程序通过其它的管道来管理设备的功能接口，实现普通数据的交互。实际的数据传输过程是这样的：设备驱动程序通过对USBD接口(USB Driver Interface)的调用发出输入输出请求（IRP，I/O Request Packet）；USB驱动程序接到请求后，调用HCD接口（Host Controller Driver Interface），将IRP转化为USB的传输（Transfer），一个IRP可以包含一个或多个USB传输；然后HCD将USB传输分解为总线操作（Transaction），由主控制器以包（Packet）的形式（见2.3.2）发出。需要注意的是：所有的数据传输都是由主机开始的，任何外设都无权开始一个传输。

IRP是由操作系统定义的，而USB传输与总线操作是USB规范定义的。为了进一步说明USB传输，我们引出帧（frame）的概念。

USB总线将1ms定义为一帧，在这1ms里USB进行一系列的总线操作，但没有任何一次总线操作是跨帧进行的。引入帧的概念主要是为了支持与时间有关的总线操作。

2.3.2 USB 数据单元

所有总线操作（通讯过程）都可以归结为三种包的传输：令牌包、数据包和应答包。任何操作都是从主机开始的，主机以预先排好的时序，发出一个描述操作类型、方向、外设地址以及端点号（这将在以下部分给予解释）的包，我们称之为令牌包（Token Packet）。然后由在令牌中指定的数据发送者发出一个数据包（Data Packet）或者报告它没有数据可以传输。而数据的目的地一般要以一个应答包（Handshake Packet）做出响应以表明传输是否成功。

还有一种包称为特殊包，用于低速设备，详细的说明见2.3.5。

2.3.2.1 域

所有的包都是由不同的域组成的：

- 同步域(SYNC field)：所有的包都起始于同步域，它被用于本地时钟与输入信号的同步，并且在长度上定义为8位。SYNC的最后两位作为一个记号表明PID域(标识域)的开始。在以后的叙述中，同步域将被省去。
- 标识域(PID，Packet Identifier Field)：对于每个包，PID都是紧跟着SYNC的，PID指明了包的类型及其格式。主机和所有的外设都必须对接收到的PID域进行解码。如果出现错误或者解码为未定义的值，那么这个包就会被接收者忽略。如果外设接收到一个PID，它所指明的操作类型或者方向不被支持，外设将不做出响应。
- 地址域(Address Field)：外设端点都是由地址域指明的，它包括两个子域：外设地址和外设端点。外设必须解读这两个域，其中有任何一个不匹配，这个令牌就会被忽略。外设地址域(ADDR)指定了外设，它根据PID所说明的令牌的类型，指明了外设是数据包的发送者或接收者。ADDR共6位，因此最多可以有127个地址。一旦外设被复位或上电，外设的地址被缺省为0，这时必须在主机枚举过程中被赋予一个唯一的地址。而0地址只能用于缺省值，而不能分配作一般的地址。端点域(ENDP)有4位，它使设备可以拥有几个子通道。所有的设备必须支持一个控制端点0(endpoint 0)。低速的设备最多支持2个端点：0和一个附加端点。高速设备可以支持最多16个端点。
- 帧号域(Frame Number Field)：这是一个11位的域，指明了目前帧的排号，每过一帧(1ms)这个域的值加1，到达最大值0xFF后返回0。这个域只存在于每帧开始时的SOF令牌中。SOF令牌在下面将详细介绍。
- 数据域(Data Field):范围是0－1023字节，而且必须是整数个字节。
- CRC校验：包括令牌校验和数据校验。

2.3.2.2 包

上文已经提到，USB 的总线操作使用了 4 种包：

1.令牌包(Token Packet)：

令牌包有IN(输入)、OUT(输出)、SETUP(设置)、和SOF(Start of Frame，帧起始)四种类型。其中IN、OUT、SETUP的格式如图2.8所示。

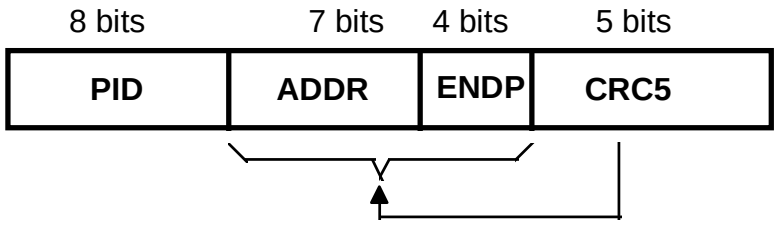


图2.8 IN、OUT、SETUP数据格式

对于OUT和SETUP来说，ADDR和ENDP中所指明的端点将接收到主机发出的数据包，而对IN来说，所指定的端点将输出一个数据包。

Token和SOF在三个字节的时间内以一个EOP(End of Packet)结束。如果一个包被解码为Token包但是并没有在3个字节时间内以EOP结束，它就会被看作非法或被忽略。

对于SOF包，它的格式如图2.9所示。主机以一定的速率(1ms±0.05一次)发送SOF包。SOF不引起任何操作。

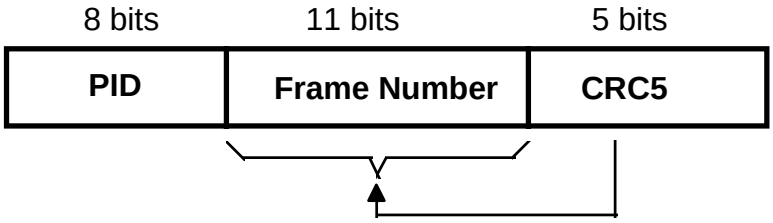


图2.9 SOF数据格式

2.数据包（Data Packet）：

数据包有Data0和Data1两种类型。这两种包的定义是为了支持数据触发同步。数据包包含了PID、DATA和CRC三个域（图2.10）。

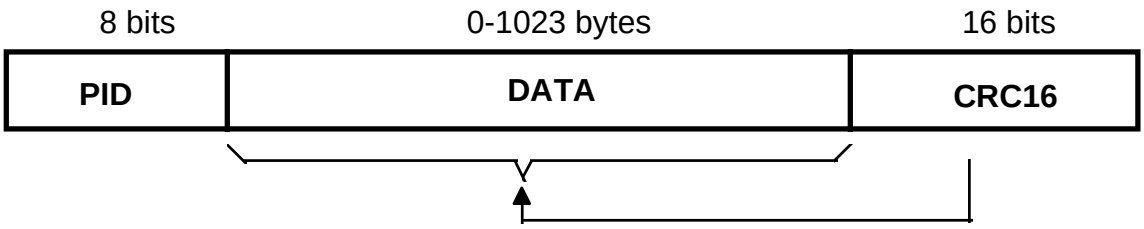


图2.10 DATA数据格式

3.应答包（Handshake Packet）：

应答包用来报告数据传输的状态，有三种类型：

- 确认包ACK：表明数据接收成功。
- 无效包NAK：指出设备暂时不能传送或接收数据，但无需主机介入，可以解释成设备忙。
- 出错包STALL：指出设备不能传送或接收数据，但需要主机介入才能恢复。

NAK和STALL不能由主机发出。

应答包仅包含一个PID域（图2.11）。只有支持流控制的传输类型（控制、中断和批传输）才能返回应答包。

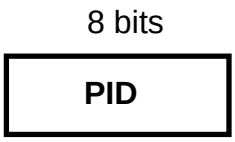


图2.11 PID数据格式

4.特殊包（Special Packet）：

特殊包的PID名称为PRE（preamble）,用于低速操作（详见2.3.5）。

2.3.3 USB 总线传输

为了满足不同外设和用户的要求，USB 提供了四种传输方式：控制传输；同步传输；中断传输；批传输。它们在数据格式、传输方向、数据包容量限制、总

线访问限制等方面有着各自不同的特征：

2.3.3.1 控制传输(Control Transfer)

控制传输的特点是：

1. 通常用于配置/命令/状态等情形；
2. 其中的设置操作（setup）和状态操作（status）的数据包具有 USB 定义的结构，因此控制传输只能通过消息管道进行；
3. 支持双向传输；
4. 对于高速设备，允许数据包最大容量为 8，16，32 或 64 字节，对于低速设备只有 8 字节一种选择；
5. 端点不能指定总线访问的频率和占用总线的时间，USB 系统软件会做出限制；
6. 具有数据传输保证，在必要时可以重试。

控制操作主要包括两个操作阶段（transaction stage）:设置和状态。图 2.12 给出了设置操作的细节，如果数据没有正确接收，那么设备就会忽略它，而且不返回应答包。

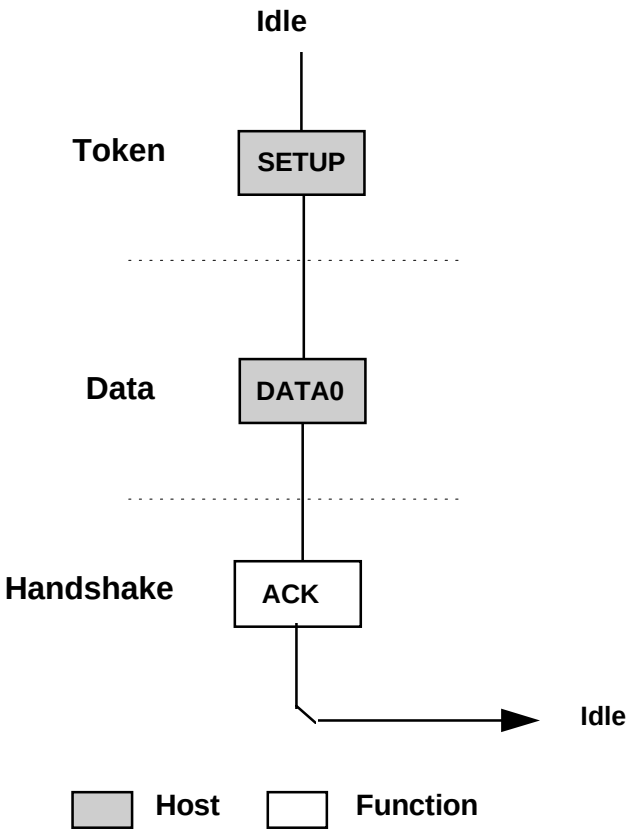


图 2.12 控制操作流程

下面是控制操作的详细描述（图 2.13），其中我们要注意数据包 PID 的使用。

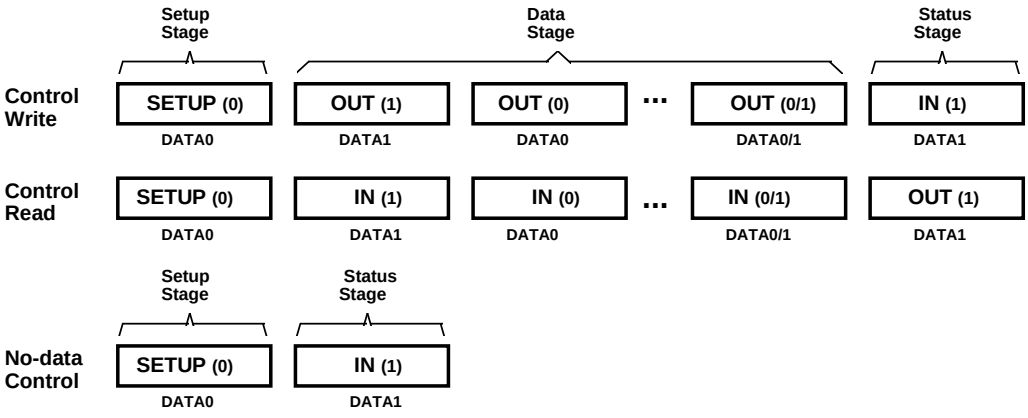


图 2.13 控制操作读写过程

2.3.3.2 同步传输(Isochronous Transfer)

同步传输的特点是：

1. 是一种周期的、连续的传输方式，通常用于与时间有密切关系的信息的传输；
2. 数据没有 USB 定义的结构（数据流管道）；
3. 单向传输，如果一个外设需要双向传输，则必须使用另一个端点；
4. 只能用于高速设备，数据包的最大容量可以从 0 到 1023 个字节；
5. 具有带宽保证，并且保持数据传输的速率恒定（每个同步管道每帧传输一个数据包）；
6. 没有数据重发机制，要求具有一定的容错性；
7. 与中断方式一起，占用总线的时间不得超过一帧的 90%。

同步操作不同于其他类型，只包含两个阶段：令牌和数据（图 2.14）。因为同步传输不支持重发的能力，所以没有应答阶段。另外它也不支持数据的触发同步与重试。

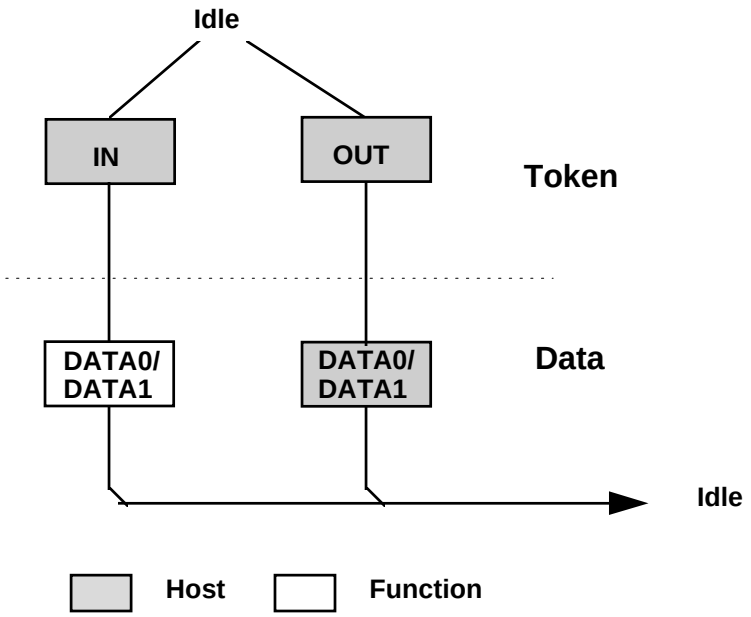


图 2.14 同步操作流程

2.3.3.3 批传输(Bulk Transfer)

批传输的特点是：

1. 用于大量的、对时间没有要求的数据传输；
2. 数据没有 USB 定义的结构（数据流管道）；
3. 单向传输，如果一个外设需要双向传输，则必须使用另一个端点；
4. 只能用于高速设备，允许数据包最大容量为 8，16，32 或 64 字节；
5. 没有带宽的保证，只要有总线空闲，就允许传输数据（优先级小于控制传输）；
6. 具有数据传输保证，在必要时可以重试，以保证数据的准确性。

批操作包括令牌、数据、应答三个阶段，如图 2.15 所示。对于输入操作，如果设备不能返回数据，那么必须发出 NAK 或 STALL 包；对于输出，如果设备不能接收数据，也要返回 NAK 或 STALL。

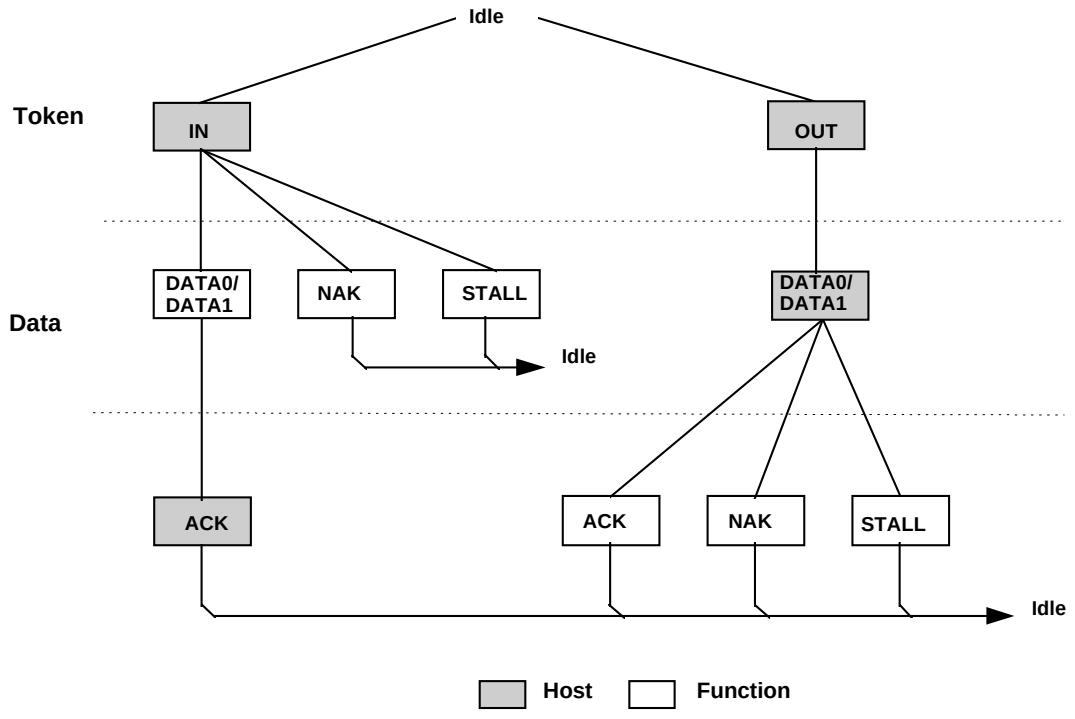


图 2.15 批操作流程

图 2.16 描述了批操作的读、写过程以及序列位(sequence bit)和数据包 PID 的使用（详见 2.3.2）。

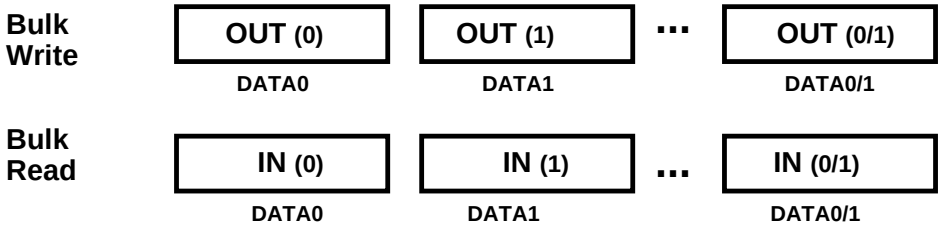


图 2.16 批操作读写过程

2.3.3.4 中断传输(Interrupt Transfer)

中断传输的特点是：

1. 用于非周期的、自然发生的、数据量很小的信息的传输，如键盘、鼠标等；
2. 数据没有 USB 定义的结构（数据流管道）；
3. 只有输入这一种传输方式（即外设到主机）；
4. 对于高速设备，允许数据包最大容量为小于或等于 64 字节，对于低速设备只能小于或等于 8 字节；
5. 具有最大服务周期保证，即在规定时间内保证有一次数据传输；
6. 与同步方式一起，占用总线的时间不得超过一帧的 90%；
7. 具有数据传输保证，在必要时可以重试。

中断操作只有输入这一个方向，具体格式与批操作的输入情形类似（图 2.17）。

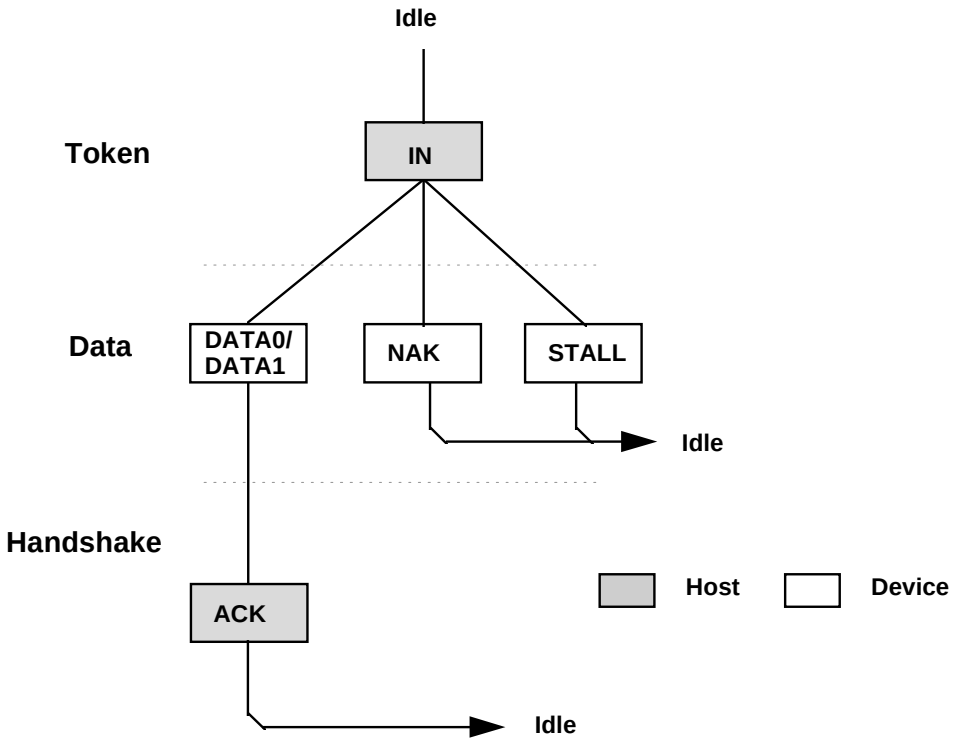


图 2.17 中断操作流程

2.3.3.5 结论

控制操作在所有的 USB 设备中都需要使用，因为主机对 USB 设备的配置命令都需要通过控制操作来发送，而设备的描述信息也需要通过控制操作传递给主机。至于同步操作，适用于实时性要求较高，而准确性要求较低的场合，比如音频或视频设备，要求语音或图象不能有明显的滞后，而如果传输的某些字节出错，人耳和人眼也无法察觉，这时就使用同步操作来传输音频和视频数据流。中断操作和批操作都属于异步传输方式，它们的主要区别在于传输数据的速度不一样，一般来说，批操作比中断操作传输数据要快得多。

图 2.18 描述了输入输出请求（IRP）、传输（transfer）与操作（transaction）之间的关系。

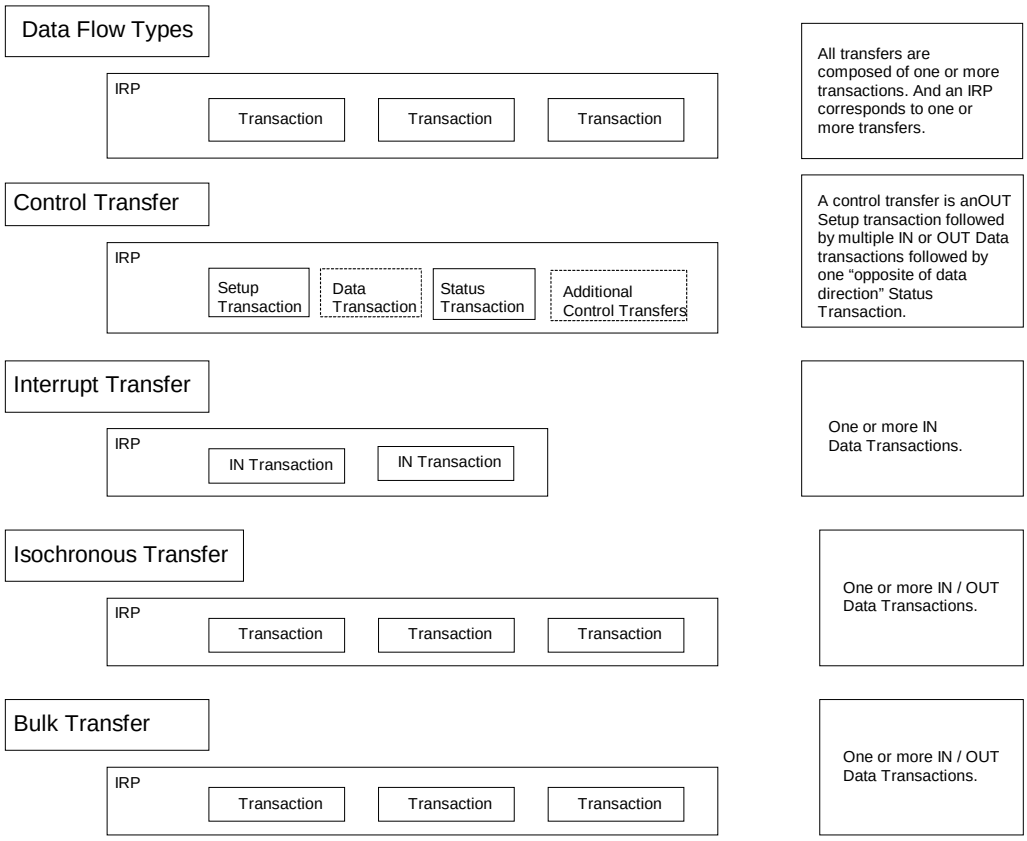


图 2.18 USB 数据传输

2.3.4 数据触发同步与重试

USB提供了保证数据序列同步的机制，这一机制确保了数据传输的准确性。这一同步过程是通过Data0和Data1的PID以及发送者与接收者上的数据触发序列位（data toggle sequence bit）来实现的。接收者的序列位只有当接收到一个正确的数据包时(包括正确的PID)才能被触发。而发送者的序列位只有当接收到确认包ACK时才能被触发。在总线传输的开始，发送者与接收者的序列位必须一致，这是由控制命令来实现的。同步传输方式不支持数据触发同步。

图2.19，2.20，2.21说明了数据触发同步的基本原理。

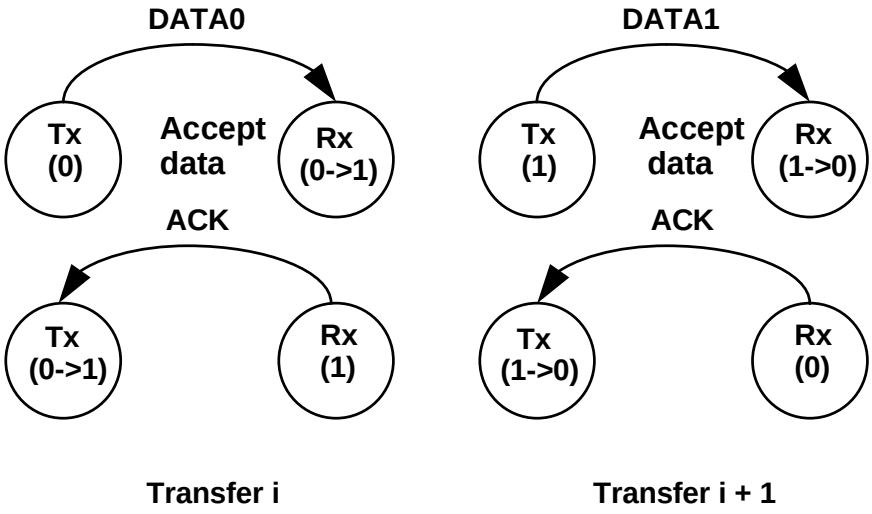


图2.19 数据触发与同步（一）

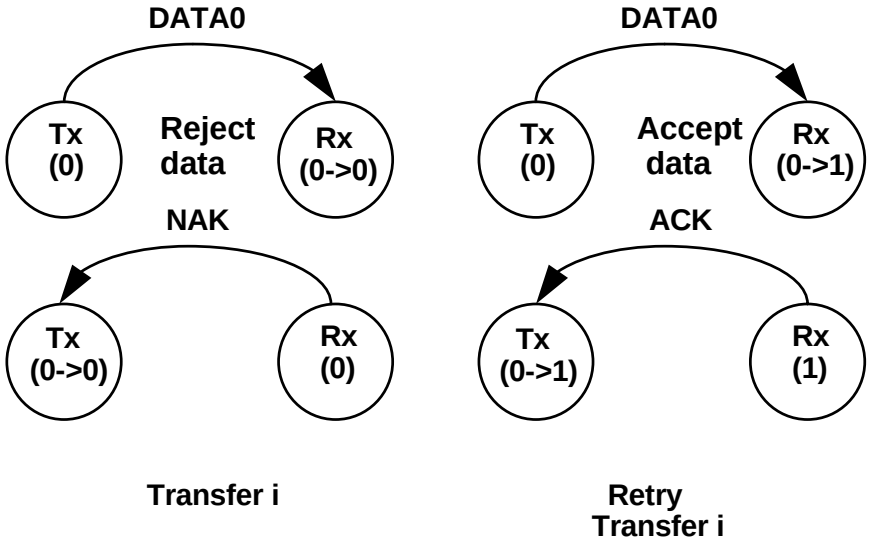


图2.20 数据触发与同步（二）

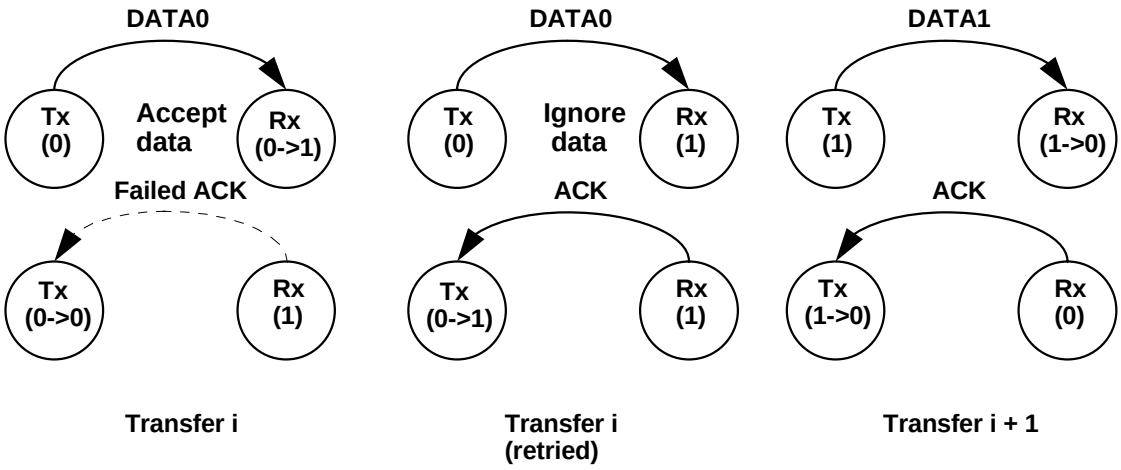


图2.21 数据触发与同步（三）

每次总线操作，接收者将发送者的序列位(被译码成数据包PID的一位，即Data0或Data1)与本身的相比较。如果数据不能接收，则必须发送NAK。如果数据可以被接收，并且两者的序列位匹配，则该数据被接收并且发送ACK，同时，接收者的序列位被触发。如果数据可以被接收，但两者的序列位不匹配，则接收者只发出ACK而不进行其它操作。对于发送者来说，在接收到NAK时或在规定时间内没有接收到ACK，则将上一次的数据重发。

2.3.5 低速操作

Hub具有禁止高速信号进入低速设备的能力，这既防止了电磁干扰的发生，又保护了低速设备。图2.22是一次低速的输入操作，主机发送令牌与应答包并且接收了一个数据包。

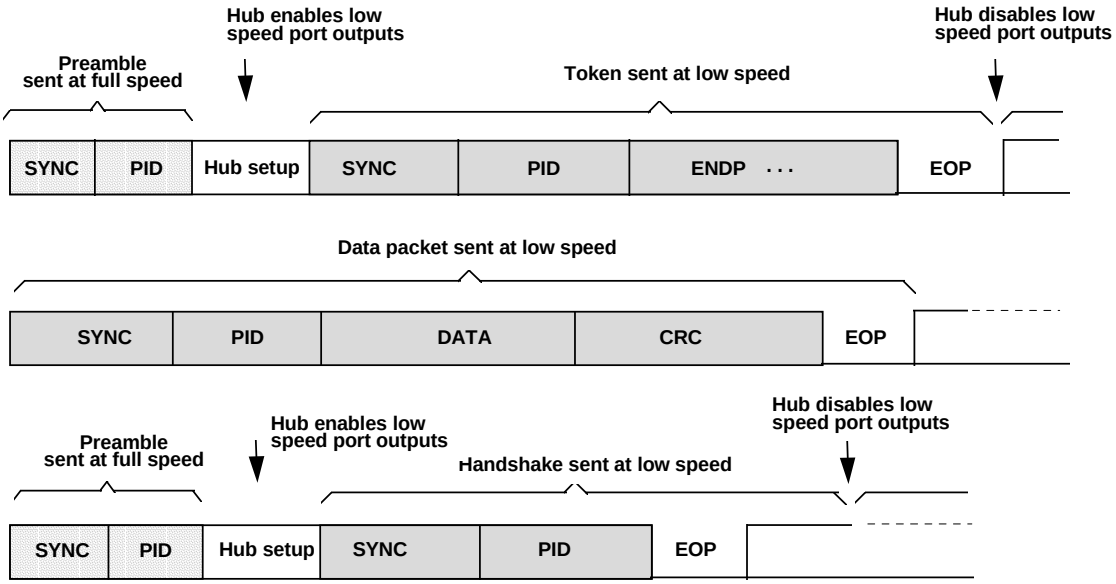


图2.22 低速方式的输入操作

所有下行的低速传输的包，必须先发送一个PRE包。Hub必须解释PRE包，而所有其它的USB设备必须忽略这个包。主机在发送完PRE包后，必须等待至少4位的时间，而在这个期间，Hub完成必要的设置，使之能接收低速的信号。在接收到EOP信号之后，Hub关闭低速设备的端口。

上行的操作则没有上述的行为，低速与高速是一样的。

低速操作还有其它的限制：

(1)数据包最大限制为8个字节。

(2)只支持中断和控制传输方式。

2.3.6 错误检验与恢复：

USB具有检查错误的能力，并且可以根据传输类型的要求进行相应的处理。例如，控制传输的需要很高的数据准确度，因此支持所有错误检验与重试来实现端对端的数据完整传输。而同步传输不允许重试，因此必须具有一定的容错性。

USB这种检查错误的能力包括：PID检验、CRC检验、总线时间溢出以及EOP错误检验等等。

2.4 结论

以上我们介绍了 USB 的基本结构和原理，只涉及了 USB 规范 1.1 版和 2.0 版的一些章节，如果想深入了解 USB，必须仔细阅读 USB 规范。这一规范从总体上描述了 USB 的结构和原理，而对于每一类 USB 设备（包括 HID 设备、音频设备等等），还相应制定了描述这类设备的 USB 规范，如果想开发 USB 设备，还必须对这种描述设备的规范有所了解。

USB 是 1994 年出现的，推出 USB 主要有三个目的：一是使安装、使用设备更加容易：使用 USB，几乎所有的中低速设备都可以用相同的电缆和接头与 PC 相连，即使是不懂得硬件知识的人也可以安全的安装和使用 USB 设备，USB 所具有的即插即用特性，更是体现了它的便捷；二是扩展 USB 的 I/O 能力：从理论上讲，USB 最多可以支持 127 个外设，总带宽达 12Mbs，可以满足几乎所有的中低速设备的要求，如果用户想增加一个外设，只需将它插到某个 Hub 的一个端口即可；三是支持声音和压缩影象等实时数据的传输：这为集成语音、电话等功能提供了一个简单的途径，这也是 USB 将得以发展的一个重要因素。可以支持 USB 的外设十分广泛，比如：鼠标、键盘、游戏杆、显示器、扫描仪、打印机、麦克风、数字相机等等。

USB 从推出到现在已经有好几年了，USB 的应用已经越来越广泛，成为标准的设备接口之一。学习、使用 USB 总线技术已经成为了所有硬件开发者的需要。

第三章　USB 项目开发技术

USB 的项目开发主要包括：设备端的硬件、软件开发，以及主机端的软件开发 3 个部分。

3.1　USB 设备端硬件开发

设计 USB 设备端的硬件需要使用专用的 USB 接口芯片。国内外的厂家推出了数百种支持 USB 设备端接口的芯片，主要分为两类：内含 CPU 的和不内含 CPU 的。Motorola 的产品都是在微控制器内部集成 USB 模块，属于内含 CPU 的 USB 接口芯片。而美国国家半导体（NS）公司推出的 USB 接口芯片（如 USBN9602）需要一个 CPU 芯片对它进行控制，属于不内含 CPU 的 USB 接口芯片。

这两种方案在与 USB 接口相关的硬件设计上没有什么区别。一般的 USB 接口芯片都会提供 3 个引脚用于 USB 接口的硬件连接：

- 3.3V 的电平输出脚
- D＋
- D－

以 Motorola 的 MC68HC05JB4 为例，硬件连接的原理图（图 3.1）如下：

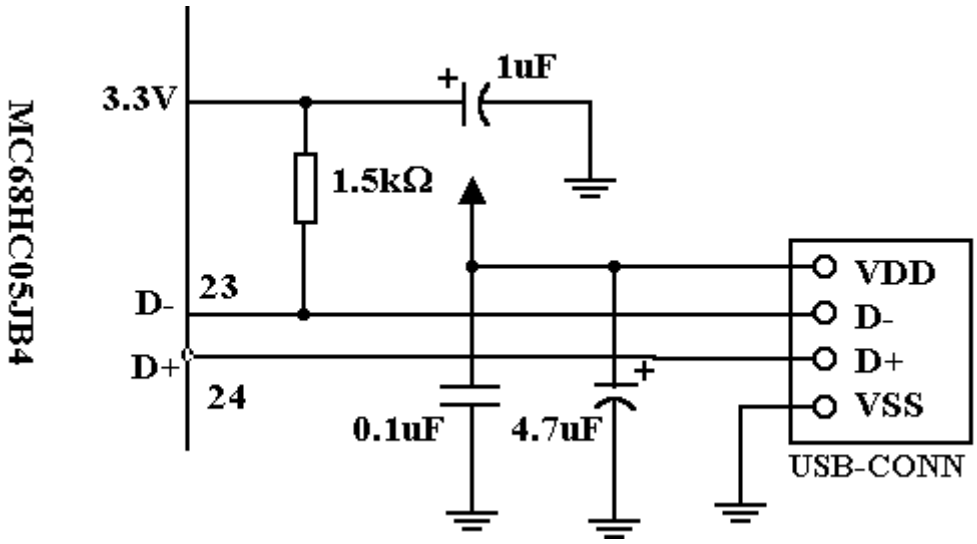


图 3.1　USB 硬件连接原理图

这里需要注意的是 D－与 3.3V 之间的 1.5KΩ 的电阻，阻值范围必须是 1.5KΩ±5%，否则设备将无法被 PC 或 HUB 识别。

为了保护芯片，有时芯片厂商会推荐在芯片的 D＋和 D－与 USB 插座（USB-CONN）的 D＋和 D－之间各串联上一个小电阻，以起到限制电流的作用。

与 USB 相关的硬件设计还包括时钟频率的选择，目前大部分的 USB 方案都是将一个标准的时钟源经过四分频以后用于 USB 总线的数据传输，也就是说，全速设备（总线速度为 12Mbps）使用 48MHz 的外部时钟源（例如美国国家半导体公司出品的 USBN9602），而低速设备（总线速度为 1.5Mbps）使用 6MHz 的外部时钟源（例如 Motorola 的 MC68HC908JB8 等）。

USB 的插座和电缆都是电子市场上非常标准的器件，开发者可以直接购买，而不需要做其他任何相关的工作。

3.2 USB 设备端软件开发

USB 设备端的软件开发实际上包含着很大的工作量，因为开发者需要在自己的平台上实现一个 USB 设备端的协议栈。

简单说来，USB 设备端的软件主要需要两个功能：一是监视设备的状态，控制设备的行为，自动产生状态信息和用户命令信息；二是完成主机与设备之间的 USB 总线通讯，自动处理主机的控制和查询命令。实现前一个功能的软件模块我们称为 USB 设备的通用模块，而实现后一个功能的软件模块我们称为 USB 设备的协议模块。

3.2.1 USB 设备通用模块的软件开发

通用模块的实现与设备的具体工作方式有关，与一般非 USB 设备的实现方法大体一致。唯一的区别是开发者需要预先定义设备所属的类别和使用的协议。原因是：如前文（见 2.2.5USB 的标准接口和外设）所述，USB 协议将设备分为不同的类型，每个设备类型都定义了一组功能相似的设备的共同行为和协议。这样主机与 USB 设备之间的通信就可以通过一些标准格式的数据包来完成。USB 开发者论坛发布了一系列 USB 设备的类型定义，并配以相应的开发和使用说明。开发者需要预先了解 USB 对设备的分类，并为自己的项目或产品选择合适的类别，然后在后续的开发过程中，就应该完全按照 USB 通用协议和该子类协议的内容去实

现自己的 USB 设备。

以 HID 子类设备为例，由于 HID 子类协议规定设备与主机之间的数据通讯是以报告（Report）为单元进行的，所以设备的数据都必须填写在一个标准的报告表格内发送。例如 USB 鼠标，需要向主机报告鼠标在 X 和 Y 方向的相对移动坐标和按键的情况，它就可以定义这样的一个报告表格：第一个字节是按键状态，第二个字节是 X 坐标，第三个字节是 Y 坐标。在鼠标插入 PC 并进行总线枚局过程的时候，鼠标需要将这个报告表格的格式和内容告诉 PC。在鼠标每次发送数据的时候，它都需要将数据写入这三个字节，并按顺序发送到主机。

开发者还应考虑的是主机的工作平台，比如 Windows 98/NT、Unix 等等，原因是不同的系统对不同的设备的支持程度不同。例如在 Windows 98 下，系统除了提供通用的 USB 设备的底层驱动以外，还单独提供了少数 HID 设备（如鼠标）的完整驱动，也就是说，开发者如果想实现一个 USB 鼠标，是不需要在 Windows 98 下开发自己的驱动程序的，而如果是想实现一个 USB 手写板，就必须得在通用的底层驱动基础上开发自己的设备驱动程序。所以设备类型的定义也直接影响着开发的难度和时间。

3.2.2 USB 设备协议模块的软件开发

协议模块的实现较为困难。USB 总线上传递的信息有两种：一是由数据线的差分信号传递的三种包（令牌包、数据包和应答包）；一是经过定义的特殊的数据线信号，如复位信号、唤醒信号和包结束（EOP）信号等。同时如上文（2.3.3）所述，USB 还定义了四种总线操作方式：控制传输、同步传输、中断传输和批传输。要同时实现对所有这些信号和传输的支持，开发者需要做大量的工作。

一般来说，USB 的芯片都会提供几个标准的 USB 端点，每个端点都支持单一的总线操作方式。其中端点 0 必须支持控制传输，而其他端点则可以支持同步、中断和批传输中的任意一种总线操作方式。控制这些端点需要通过相应的控制寄存器、状态寄存器、中断寄存器和数据寄存器。控制寄存器用于设置端点的工作模式，启动端点的功能等等；状态寄存器用于查询端点的当前状态；中断寄存器用于设置端点的中断触发和响应功能；而数据寄存器则是设备与主机交换数据时使用的缓冲区。

例如 Motorola 的 8 位微控制器 MC68HC05JB4 中的 USB 模块就提供了 3 个端

点，其中端点 0 通过控制传输与主机通讯，而端点 1 和端点 2 则使用中断传输。对应于 3 个端点，MC68HC05JB4 提供了 3 个控制寄存器，2 个中断寄存器（端点 1 和端点 2 共用 1 个），同时为端点 0 提供了 8 个数据发送/接收寄存器，为端点 1 和端点 2 提供了 8 个共用的数据发送寄存器。其他在 USB 模块中提供的寄存器还包括一个地址寄存器和一个状态寄存器。

软件开发的主要工作是编写 USB 的中断服务例程，其功能是处理 USB 发送/接收的不同的通讯信息（如令牌、数据或应答等），从端点 0 获得主机的控制信息，或是向端点 0 发送设备的状态信息，以及向端点 1 或端点 2 发送完整的数据报告等。

USB 中断的触发条件有很多种，大致可分为：

- 接收缓冲区满
- 发送缓冲区空
- SETUP 令牌传输
- OUT 令牌传输
- IN 令牌传输
- 主机发送唤醒（RESUME）信号
- 主机发送包结束（EOP）信号

中断服务例程的任务就是分辨各种触发条件，然后转入相应的处理例程。整个中断处理过程可以分成三个步骤：

1. 通过访问 USB 芯片的寄存器了解中断触发的原因，以及其他相关信息；
2. 根据步骤 1 收集的信息，转入相应的协议处理过程；
3. 根据协议处理的结果，控制 USB 芯片完成相应的通讯任务。

第 1 个和第 3 个步骤与芯片的结构直接相关，需要开发者在熟悉芯片的基础上小心处理。

最复杂的是第 2 个步骤，但它与硬件没有直接关系，开发者往往可以参照其他 USB 设备的实现方法来实现。其中又以三个令牌传输（SETUP、OUT 和 IN）的响应部分最为复杂。根据设备类型的不同，主机发送的令牌的种类和数量都会不同。例如，USB 手写板属于 HID 设备，在响应主机命令的时候，除了需要处理普通的 USB 命令（如 Set Address、Set/Get/Clear Feature 和 Get Status 等等）以外，还需要处理 HID 设备特定的一些命令，如 Set/Get Report、Set/Get Idle

和 Set/Get Protocol 等等。中断服务例程需要在对令牌传输响应的过程中一一分辨出这些命令，并完成相应的工作。

下表（表 3.1）是我在实现 USB 手写板设备端软件的时候使用的 USB 命令和处理函数的对照表。开发者可以参照这个表格编写中断处理过程中的命令解析代码。

表 3.1 USB 命令和处理函数对照表

	命令号	命令名	处理子例程
USB设备一般命令	0	GET_STATUS	SRQ_GET_STATUS
	1	CLR_FEATURE	SRQ_CLR_FEATURE
	2	(reserved)	PST_STALL
	3	SET_FEATURE	SRQ_SET_FEATURE
	4	(reserved)	PST_STALL
	5	SET_ADDRESS	SRQ_SET_ADDRESS
	6	GET_DESC	SRQ_GET_DESC
	7	SET_DESC	SRQ_SET_DESC
	8	GET_CONFIG	SRQ_GET_CONFIG
	9	SET_CONFIG	SRQ_SET_CONFIG
	10	GET_IF	SRQ_GET_IF
	11	SET_IF	SRQ_SET_IF
	12	SYNC_FRAME	SRQ_SYNC_FRAME
HID设备特有命令	命令号	命令名	处理子例程
	0	(reserved)	PST_STALL
	1	GET_REPORT	HCR_GET_REPORT
	2	GET_IDLE	HCR_GET_IDLE
	3	GET_PROTOCOL	HCR_GET_PROTOCOL
	4	(reserved)	PST_STALL
	5	(reserved)	PST_STALL
	6	(reserved)	PST_STALL
	7	(reserved)	PST_STALL
	8	(reserved)	PST_STALL
	9	SET_REPORT	HCR_SET_REPORT
	10	SET_IDLE	HCR_SET_IDLE
	11	SET_PROTOCOL	HCR_SET_PROTOCOL

3.2.3 设备的挂起与唤醒

除了上述两大模块，在程序的主循环里系统还有很多工作要做，其中最复杂、也是最重要的，便是实现 USB 设备特有的挂起（SUSPEND）与唤醒（RESUME）功能。

USB 协议规定，当总线处于空闲态超过 3ms 时，设备必须进入挂起状态，而挂起的设备从总线上吸收的电流必须小于 500μA。（见 2.2.4）开发者可以通过设置 USB 芯片的寄存器使设备进入挂起状态，以 Motorola 的 8 位微控制器为例，MC68HC05JB4 的挂起可以通过设置 USB 端点 0 的中断寄存器中的挂起标志位来实现。

设备的挂起虽然实现了整个 USB 系统的低功耗，但它对设备的工作往往会产生不利的影响。以 Motorola 的 8 位微控制器 MC68HC05JB4 为例,协议规定的 500μA 包括了主机端的电缆终端匹配电阻的电流（通常为 220μA），所以对于使用总线电源的设备而言，进入挂起状态通常便意味着总电流功耗不能超过 280μA，这实际上是要求 MC68HC05JB4 进入 STOP 模式。而当芯片处于 STOP 模式时，很多的外设模块都无法正常工作了。开发者如果需要设备不进入挂起状态，通常有两种方法。一种是通过主机周期性地向设备发送包结束（EOP）信号，间隔时间小于 3ms，这样设备将永远处于正常状态；另一种方法是在设备挂起时唤醒它，具体的实现方法是由设备向主机发出远程唤醒信号，在主机认可后设备即结束挂起状态。开发者可以在 MC68HC05JB4 的外中断端口上连接 RC（阻容）电路，在设备进入挂起状态时利用电路的充放电时间产生滞后的外中断信号，再在中断发生时向主机发送远程唤醒信号，就可以使设备自动定时恢复到正常的状态。

3.2.4 USB 设备端软件整体流程

以上程序模块，是依据其各自功能的不同和调试的先后来划分的。实际上在正常的程序运行条件下，各个模块往往是交叉调用，相互协调工作的。所以在最后的流程图中，已经看不到严格独立的各个模块了。

图 3.2 是根据我们编写的 USB 手写板的程序精简出来的一个典型的程序流图，开发者可以参考编写不同设备的代码。正如上文所述，最关键、也是最困难的，还是中断处理程序的编写，开发者需要格外重视。

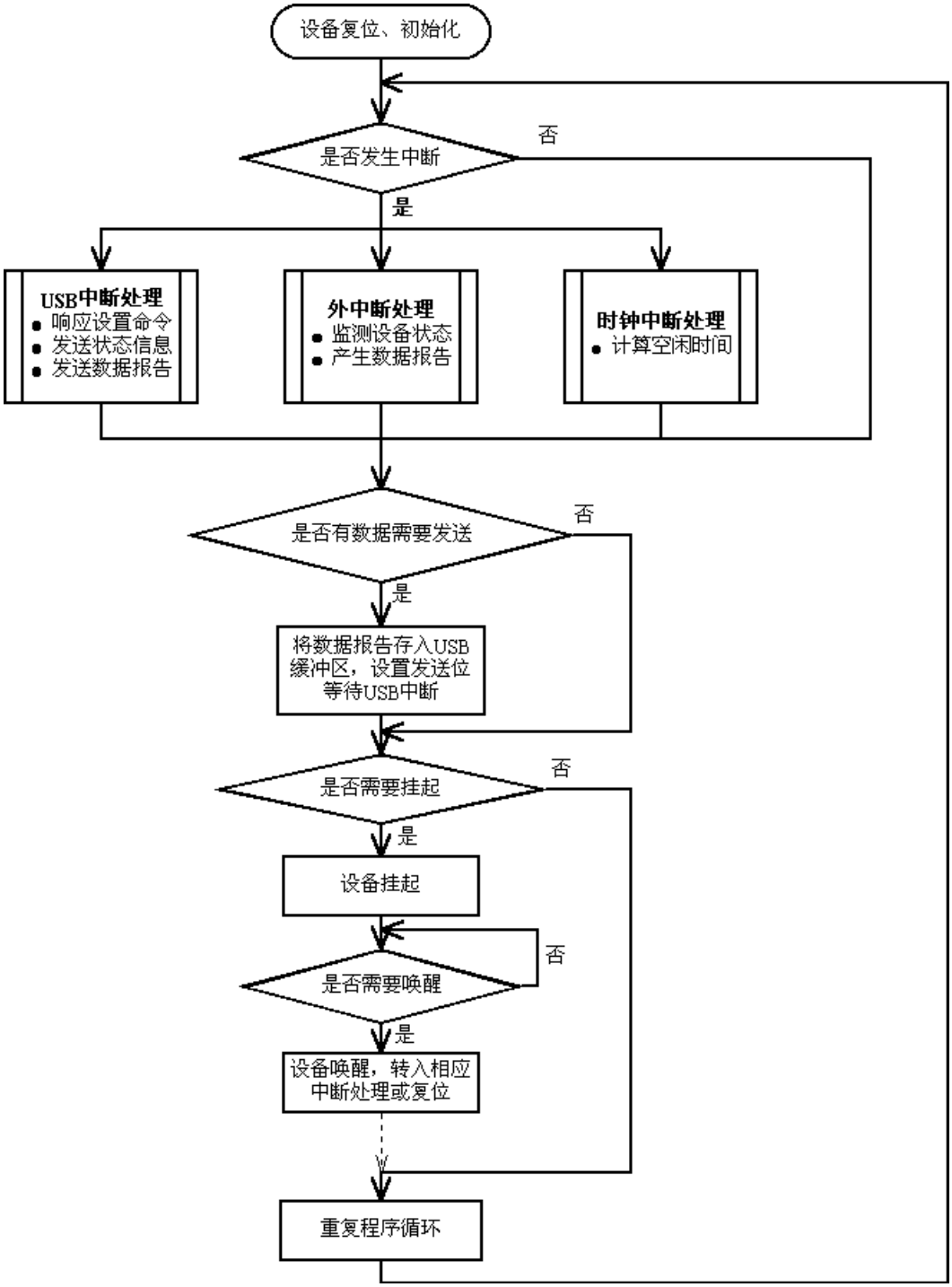


图 3.2　USB 设备端软件流程图

3.3　USB 主机端软件开发

USB 主机端的软件开发方法与用户使用的操作系统平台有很大的关系，目前对一般用户而言，主流的操作系统一般有两大类：一是微软推出的 Windows 系列，包括 Windows 95、Windows 98、Windows NT 4.0、Windows 2000 和 Windows ME

等；二是 UNIX 和 LINUX 系列。

在对 USB 的支持方面，微软比 LINUX 做得要好一些，目前大多数 USB 设备也只能在微软的 Windows 下才能使用，但也不是所有的 Windows 版本都能很好地支持 USB。由于 USB 的草案是在 1994 年发布，而成熟的 USB 1.0 版本发布更是在几年以后，所以 Windows 95 和 NT 4.0 都没有来得及为 USB 做出支持。在 Windows 95 的升级版本中，用户可以通过使用微软提供的 USBSUPP.EXE 为 Windows 95 添加一些基本的 USB 驱动程序。从 Windows 98 开始，微软开始在操作系统中为 USB 提供越来越完善的驱动支持，USB 驱动程序开始成为 Windows 标准的 WDM（Windows Driver Model）驱动程序体系的一部分。

本文涉及的几个 USB 项目都是基于普通的 PC 和 Windows 98 平台，所以下面将主要介绍在 Windows 98 下的主机端软件开发。本章主要讲述一般的主机端软件开发方法，技术细节和具体的实现方法请参看第四章对各项目在主机端软件开发工作的讲述。

3.3.1 Windows 98 下的驱动程序结构

从 Windows 3.x 到 Windows 95，Microsoft 的 Windows 操作系统取得了巨大的成功。相比起来，无论是在性能还是可靠性上都大大优于 Windows 95 的 Windows NT 反而不那么耀眼了。但性能和可靠性，始终是操作系统追求的主要目标。所以 Microsoft 最终还是把 Windows NT 定为了 Windows 操作系统的发展方向。这也就是为什么 Windows 98 虽说是 Windows 95 的升级产品，却沿袭了 Windows NT 的驱动程序结构。

Windows 98 下主要存在两种驱动程序（图 3.3）：一是为了保证对 Windows 95 下软件的兼容而保留的 Vxd（虚拟设备驱动）等模式的驱动程序；另一类驱动程序符合 WDM 1.0 标准，而这种标准同样也是 Windows NT 4.0、Windows 2000 及以后的 Windows ME 等操作系统所采用的驱动程序标准。从系统的发展趋势来看，Windows 98 推荐开发者使用符合 WDM 1.0 标准的驱动程序。本书涉及的 USB 项目最终采用的驱动程序结构和设计方法，都符合 WDM 1.0 标准。

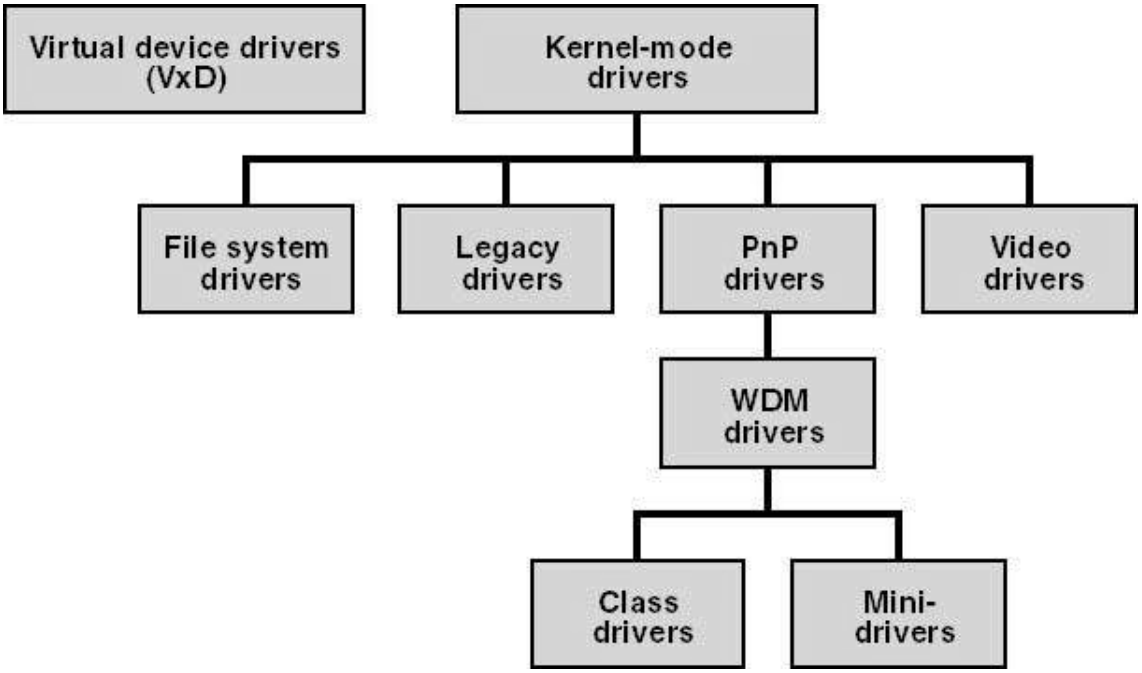


图 3.3　Windows 98 下的驱动程序类型

驱动程序可以工作在两种模式下（图 3.4），一是内核模式（Kernel Mode），此类模式的驱动程序以后缀名为 sys 的文件存在于系统目录下；一是用户模式（User Mode），此类模式的驱动程序以可执行文件（.exe）或动态连接库（.dll）的形式存在，可以从任何路径运行。用户模式下的驱动程序较内核模式下的相对简单一些，但能实现的控制和通讯功能也相对较少一些。所以究竟设计哪种模式下的驱动程序，还需要开发者根据具体的情况选择。我在调试 USB 手写板时曾同时编写了内核模式和用户模式下的驱动程序，后来为了系统安装和使用的方便，去掉了内核模式下的驱动程序，只保留了用户模式下的驱动程序。

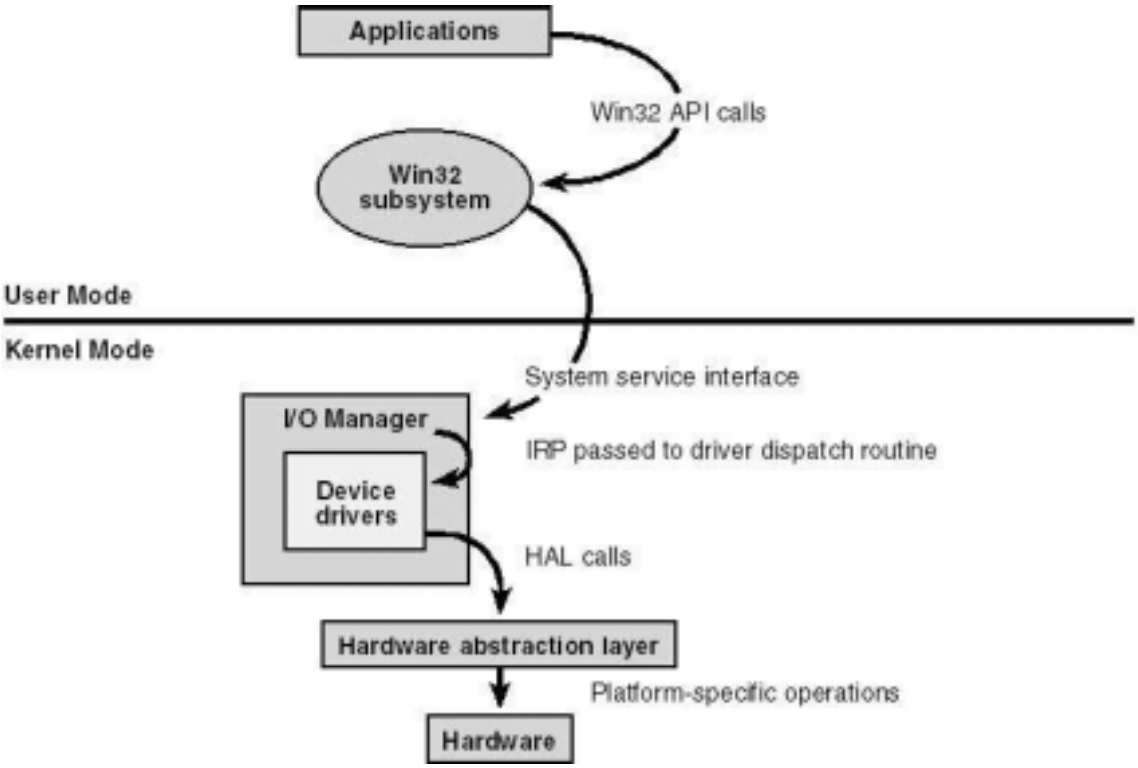


图 3.4　WDM 驱动程序体系

整个驱动程序体系的流程可以这样来描述：应用程序通过 WIN 32 提供的 API 调用向 WIN 32 子系统发出命令；WIN 32 子系统通过标准的系统调用与内核层的 I/O 管理器通讯，将用户程序的 API 调用转换成 IRP（I/O Request Package）包；I/O 管理器将 IRP 包传递给指定的设备驱动程序；设备驱动程序再将 IRP 转换为相应的硬件抽象层 HAL（Hardware abstraction layer）调用并传递给 HAL 层；最后由 HAL 与硬件直接打交道，得到用户程序需要的数据，并沿原路通过设备驱动程序、I/O 管理器和 WIN 32 子系统返回到用户程序。

3.3.2　Windows 98 下的 USB 设备驱动程序体系

Windows 98 下的 USB 设备驱动程序遵循的是 WDM 1.0 标准，有用户模式驱动程序和内核模式驱动程序两种，而内核模式驱动程序同时也包含类驱动程序（Class Driver）、客户驱动程序（Client Driver）和微驱动程序（Mini-driver）等类型。

如上文所述，USB 对设备的定义是非常复杂和多样化的，要在一个操作系统

中同时实现所有的 USB 设备的驱动支持，是非常困难的。从 Windows 98 开始，到 Windows 2000，到 Windows ME，微软一直在不断地为 USB 的设备添加驱动程序支持。但直到现在，微软的 Windows 系统仍只能对少数的几类 USB 设备提供完善的支持，如鼠标、键盘等 HID 设备，Windows 98 在内核层和用户层都为它们编写了通用的驱动程序，用户在内核层或是用户层都可以通过函数调用与设备通讯，完成想要完成的任务。而其他很多 USB 设备就只在内核层得到了支持，开发商可以在 Windows 为所有 USB 设备提供的一个最通用的内核层驱动程序——USB 类驱动程序（USBD.sys）的基础上，为自己的设备开发驱动程序。

图 3.5 是 Windows 98 下的 USB 设备驱动程序结构图的一部分，它主要描述了 HID 子类设备的驱动程序结构。

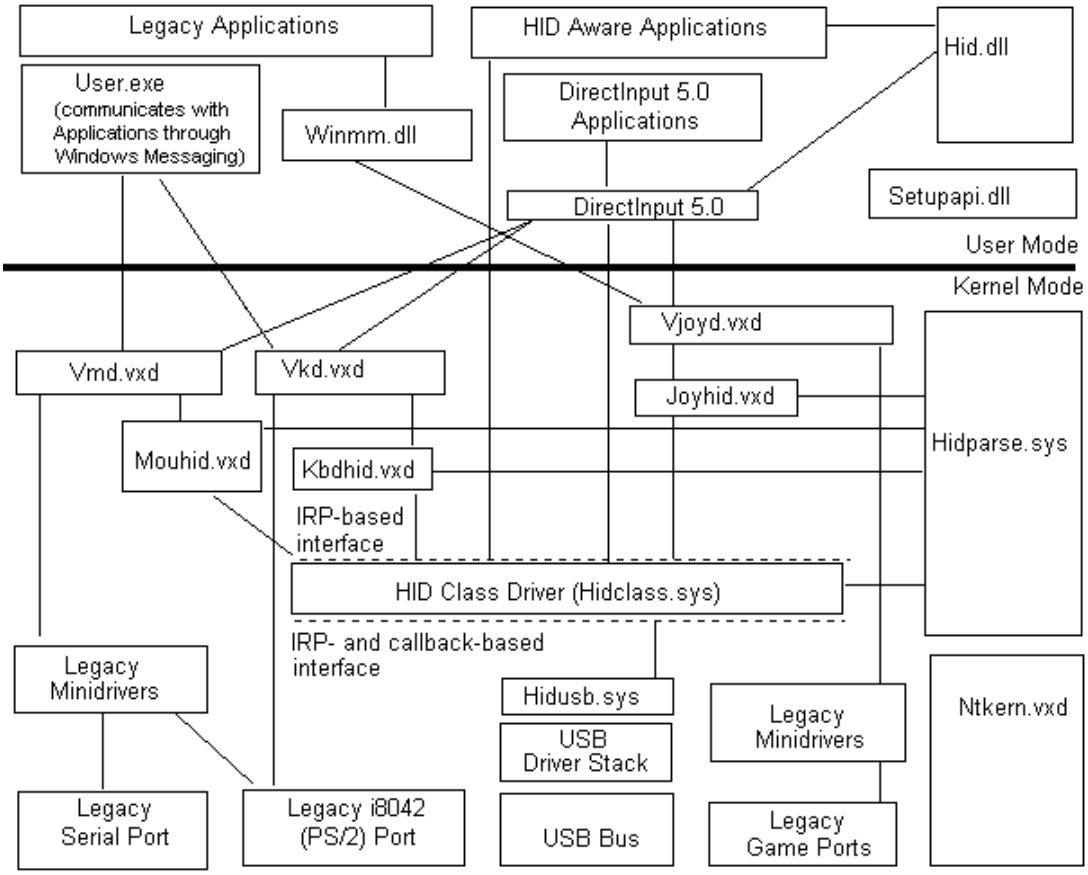


图 3.5　HID 设备驱动程序结构图

如图所示，最核心的驱动程序便是 Windows 提供的 HID 类驱动程序

（Hidclass.sys），它向下与 USB 底层驱动程序（包括 USBD.sys）打交道，向上对 HID 子类的所有上层驱动程序提供支持。还值得一提的是用户层的 Hid.dll，它在用户层的作用就象 Hidclass.sys 在内核层的作用一样，提供 HID 设备的通用命令支持。用户编写的内核层驱动程序可以直接和 HID 类驱动程序通讯，而用户层的应用程序则一般直接调用 Hid.dll 提供的 API 函数。

详细的 USB 内核层驱动程序开发技术请参看 4.2.2.3。

3.3.3 Windows 98 DDK 使用

在 Windows 98 下编写驱动程序，必须使用特定的编译器。Microsoft 为开发者提供了一个软件包——Device Driver Kit，简称为 DDK，其中包含了驱动程序的编译器和调试工具，以及帮助文档和一些范例。开发者可以在 Microsoft 的站点免费下载。

DDK 代码编写使用 C/C++语言，只是编译、连接需要使用 Visual C++和 DDK 的工具，所以使用十分简单。DDK 为开发者提供了各类驱动程序的源代码，对开发者编写自己的驱动程序很有参考价值。因为所有微软的技术文档，包括 DDK 的帮助文档在内，常常对许多技术细节含糊其词，而在 DDK 的源代码内，开发者往往能找到和自己的设备或驱动相类似的情况和解决方法。

国外的几个公司也为 Windows 开发了其他编写驱动程序的工具，如 Numega 公司推出的 DriverStudio，它可以在用户设置的选项基础上自动生成大量的代码，但它生成的代码与微软在 DDK 中提供的代码在可靠性和效率方面还是有一定差距的。

3.3.3.1 Windows 98 DDK 系统需求

使用 Windows 98 DDK 的最小要求包含以下软件和硬件：

- 微软发布的 Windows 98 操作系统
- 微软的 Visual C++ 4.2 或 5.0 版本系统（注意 Visual C++ 6.0 在早期的 Windows 98 DDK 版本中还不受支持，在运行过程中会提示 Visual C++环境变量设置有误）
- Win32 SDK 工具

完全安装需 82 M 硬盘空间（最小安装需 32 M 的硬盘空间，这不包含范例驱动程序的源代码、帮助文件或其他文档文件）。

需要开发者注意的是，一旦开始构造自己的驱动程序，开发者可能需要用到某些头文件，但这些文件在 Windows 98 DDK 中无法找到，它们存在于 SDK 平台中。开发者可以把所需的头文件或文件从 SDK 平台拷贝到适当目录下如 %98DDK% 或 %VCPPDEV%,或者安装 SDK 平台并在 %98DDK%\BIN 下编辑 Setenv.bat 文件来运行安装在%MSTOOLS%目录下的 Setenv.bat 文件。

3.3.3.2 Windows 98 DDK 的安装

1. 在 Windows 98 DDK 安装盘上运行 Setup.exe 可执行文件。Setup.exe 是基于 Windows 的应用程序,开发者可以按 Start/Run 按钮运行或是从 MS-DOS 方式运行。
2. 点击 NEXT 安装 DDK。
3. 读软件许可协议，如果开发者同意它的条件，选择“I accept the Agreement”并且点击 Next。
4. 选择开发者需要的安装功能，然后点击 Next。
5. 选择开发者安装 Windows 98 DDK 的目标驱动器和目录,然后点击 Next。(Windows 98 DDK 不能被安装在不符合 8.3 命名约定的目录下。另外，Windows 98 DDK 不能在安装时覆盖前一版本的 Windows 98 DDK。这将引起构造环境和 Visuall C++所包含的命令行工具问题。)
6. 选择开发者要安装的组件，点击 Detail 按钮选择或撤销子组件。(当开发者选择某一安装组件，安装程序会默认开发者选择了所有的子组件)然后点击 Next。
7. 核对要安装的组件，然后点击 Next。
8. 当组件的安装完成后点击 Next，然后点击 Exit。

安装程序在指定目录下安装选定的 Windows 98 DDK 组件和创建程序文件夹后，开发者应该增加环境空间。因为 Windows 98 DDK 的编译器和汇编程序（以及可选的 SDK 平台）都要用到环境空间来存储查找路径。

添加下列命令到 Config.sys 文件 (或改变已经存在的命令行)，就可以增加环境空间:

```
shell=c:\command.com /p /e:4096
```

这个命令将创建 4,096 字节的环境空间。如果开发者收到“out of environment space”消息，还应该进一步增加环境空间。

3.3.3.3 建立和使用 Windows 98 驱动程序构造环境

开发者可以使用 SETENV.BAT 建立驱动程序构造环境。在已经创建了的名为“Development Kits/Windows 98 DDK”的程序文件夹中包含了 Free Build Environment 和 Checked Build Environment 快捷方式。每次在开发者重启操作系统以后，如果需要构造驱动程序，都必须先点击这些程序文件夹项中的一个。这些快捷方式调用在 %98DDK%\BIN 目录下的 Setenv.bat，用正确的环境变量创建驱动程序构造环境。

3.3.4 Windows 98 下的 USB 设备应用程序开发

Windows 98 下的 USB 设备的应用程序开发就比较灵活了，可以使用大量的软件开发工具来进行，比如微软提供的 Visual Basic 和 Visula C++等等。

需要指出的是，应用程序在需要访问 USB 设备信息和数据时，可以通过 I/O 管理器访问开发者自行编写的驱动程序，也可以通过微软提供的动态连接库（如果微软提供了的话)，如 hid.dll，来访问微软提供的驱动程序。

3.4 结论

USB 协议并没有给设备的硬件开发带来太多的麻烦，开发者为 USB 而进行的硬件工作是比较少的。主要的问题在于芯片和方案的选择，因为目前支持 USB 的芯片太多，但真正很好地实现了 USB 协议的并不多，相当一部分芯片都存在着这样那样的问题，这是开发者在进行方案设计的时候需要认真考虑的。

USB 设备开发的工作量大部分集中在软件上，包括设备端的协议栈实现和 Windows 下的驱动程序设计等。开发者在进行这些工作时需要参阅大量的技术文档和程序范例。由于 USB 体系非常复杂，文档的描述很难做到详尽而又条理清楚，程序范例反而能更好地说明问题。

其实 USB 的项目开发是非常复杂的，但限于篇幅，本文只能简单地描述这其中的部分概念和大致的开发过程。

第四章　USB 项目简介

从 1998 年到 2000 年，我先后开发了 USB 手写识别输入系统、USB 通用设备开发平台、USB 安全钥和 USB 在线编程设备等几个项目。它们的特点大致如下：

USB 手写识别输入系统：

- Motorola 单片方案（MC68HC05JB4）
- 低速 USB 设备，属于 HID 子类
- 主要工作量：设备端硬件、设备端协议栈、Windows 98 下用户层驱动程序

USB 通用设备开发平台：

- Motorola 微控制器（MC68HC08GP32）＋USB 接口芯片(USBN9602)，双片方案
- 全速 USB 设备，属于通用 USB 设备
- 主要工作量：设备端硬件、设备端协议栈、Windows 98 下内核层驱动程序和用户层动态连接库

USB 安全钥：

- Motorola 单片方案（MC68HC08JB8）
- 低速 USB 设备，属于 HID 子类
- 主要工作量：设备端硬件、设备端协议栈、Windows 98 下用户层驱动程序、加密算法和安全认证协议、服务器端管理程序

USB 在线编程设备：

- Motorola 单片方案（MC68HC08JB8 和 MC68HC08LD64）
- 低速 USB 设备，属于通用 USB 设备
- 主要工作量：设备端协议栈、在线编程、Windows 98 下内核层驱动程序和用户层动态连接库

本章后面的内容将逐一介绍这些项目的背景、原理、技术内容和成果等情况。

4.1 USB 手写识别输入系统

4.1.1 背景

MOTOROLA 公司是目前世界上最大的微控制器供应商，其 8 位微控制器的全球市场份额达到了 30%左右。MOTOROLA 公司将其 8 位微控制器归类为用户定制的集成电路（CSIC），为客户提供了 MPU、RAM、EPROM、SPI、SCI、定时器和 USB 等多种模块，用量大的客户可以根据自己的需要选择不同的模块来构筑自己的微控制器。MOTOROLA 公司从 1996 年开始，陆续推出了一系列含有 USB 模块的 8 位微控制器，用于支持 USB 总线协议的设备，如最早的用于显示器的 MC68HC05BD9A，用于鼠标的 MC68HC05JB2，以及用于键盘的 MC68HC08KL8 和 MC68HC08KH12 等等。通过微控制器内含的 USB 模块，用户可以很方便地实现 USB 总线上的数据通讯。MC68HC05JB4 最初是用于开发 USB 游戏柄的，后来也常被用于其他一些 USB 外设的开发。

国外在 1998 年的时候已出现了不少的 USB 外设，但在国内市场上还只有台湾生产的摄像头等少数几类高速 USB 外设，低速 USB 设备还是一个空白。同时国外开发的 USB 设备多集中在鼠标、键盘等少数几类设备上，诸如 USB 手写板等设备就是在国外也很少见。当时国内的计算机非键盘输入技术发展很快，在汉字、英文和数字的手写识别方面已有相当基础。本项目之目的，就是吸收 USB 总线和 MOTOROLA 微控制器的先进技术，与中科院自动化所汉王公司的手写识别技术相结合，在汉王笔的基础上，设计生产出自己的新一代 USB 手写输入系统。

4.1.2 原理

该 USB 手写识别输入系统，采用汉王公司的传感器获得笔画信息，传给 MC68HC05JB4，经过整理后通过 USB 总线发送到 PC，再由自行编写的驱动程序接收，最终转给汉王公司的文字识别软件识别。（图 4.1）。

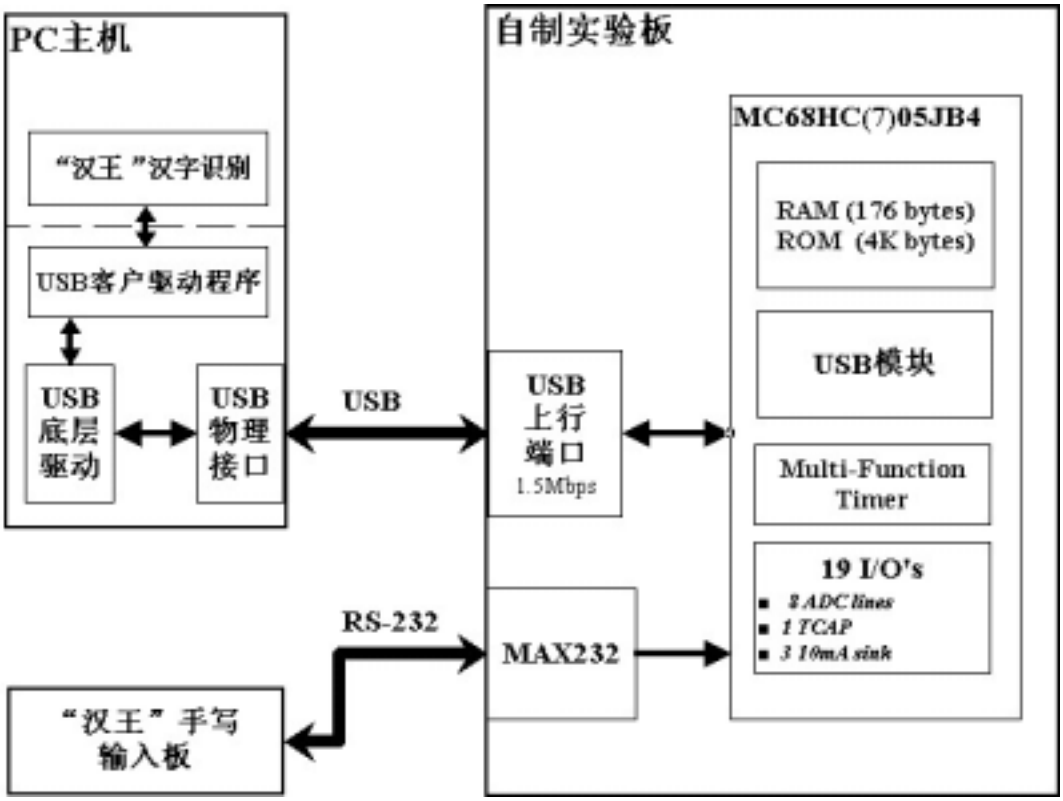


图 4.1　系统结构框图

笔触数据的采集和最终的文字识别都已由“汉王”的现成产品实现，本项目完成的工作主要是自制实验板的硬件电路设计，以及微控制器汇编程序（USB 设备端协议栈）和 WINDOWS 98 下的驱动程序编写。

4.1.2.1 设备端硬件设计

系统硬件的核心是 MOTOROLA 的微控制器 MC68HC05JB4，它需要两个接口电路分别与“汉王”手写板和主机通讯。与“汉王”手写板的接口是通过传统的 RS-232 协议实现的；与主机的接口则是通过 USB 协议实现的。同时辅助微控制器工作的还有一个晶体振荡电路。（图 4.2）

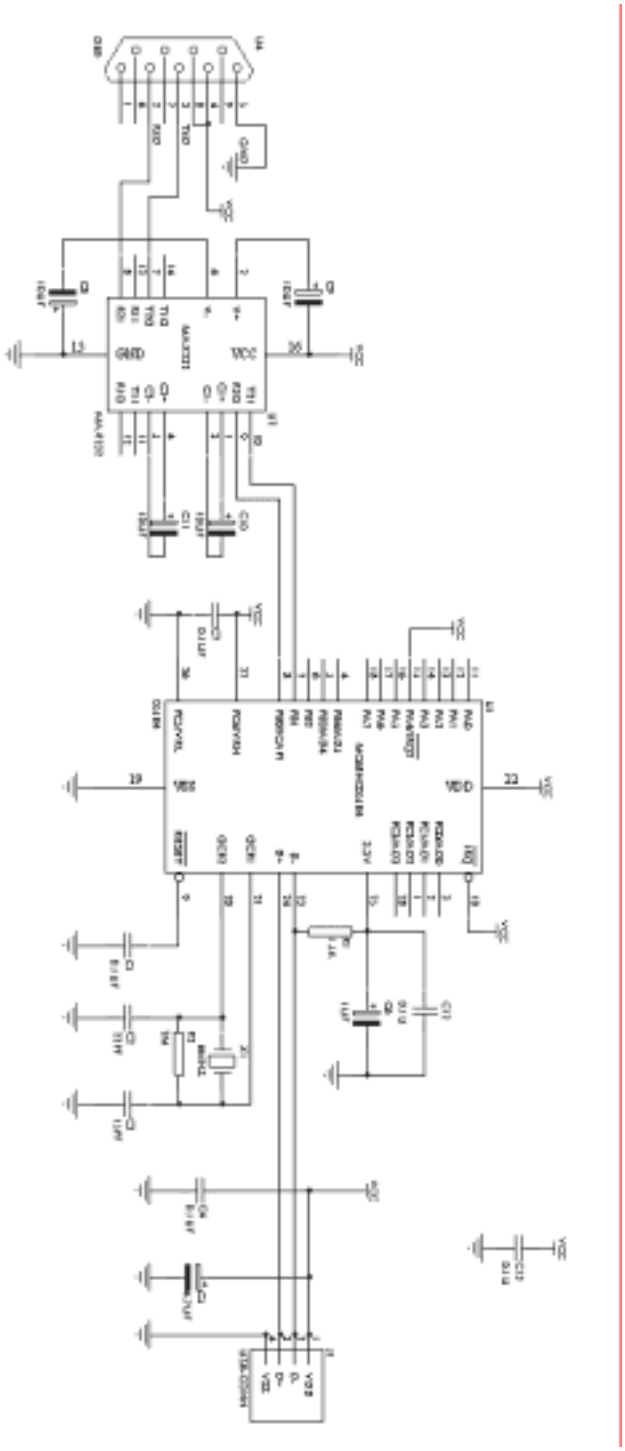


图 4.2　USB 手写识别输入系统整体电路原理图

在设计印制电路板时主要考虑的是减小信号之间的交叉干扰、电源干扰，降低噪声对电路的影响，提高整个系统的可靠性。

在本系统的电路板上，主要是晶体振荡电路对噪声比较敏感，因此在设计这部分电路时，特别注意使晶振、电阻、电容等相关器件与微控制器尽可能靠近，在布线时使这部分电路的信号线不与其他任何信号线交叉。

此外还采用了一些常规的降低噪声和干扰影响的手段，包括尽量增加地线和电源线的宽度，使用去耦电容，以及尽量减小元器件引脚长度等等。

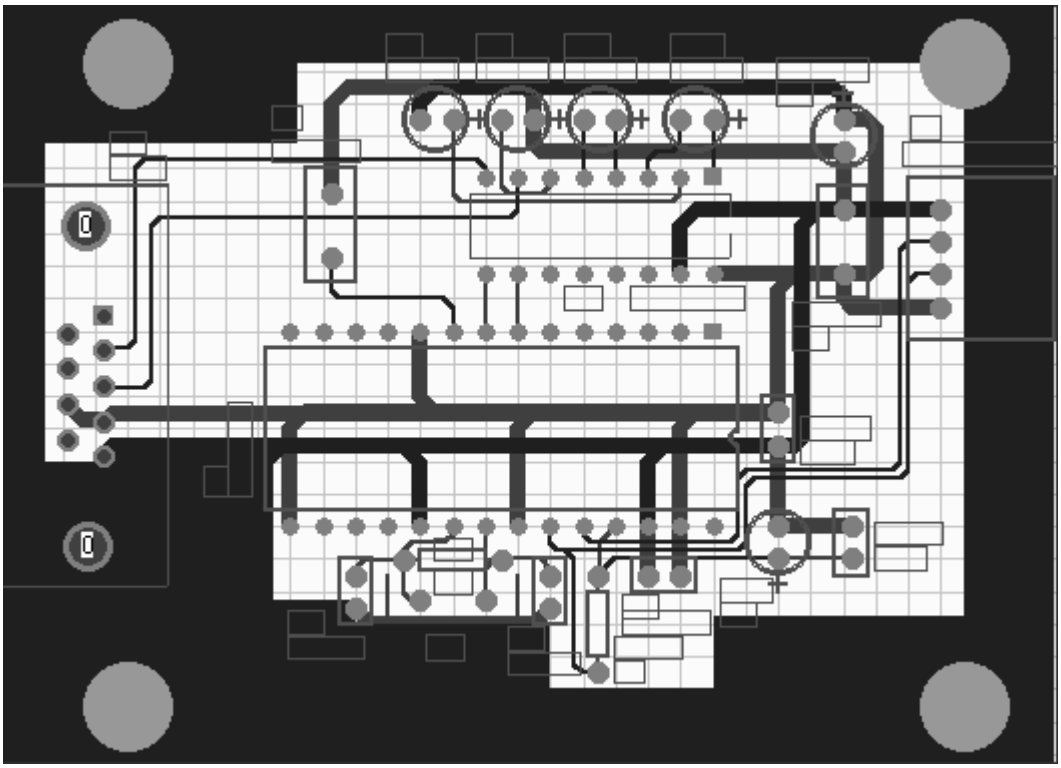


图 4.3 USB 手写识别输入系统印制电路板元件分布与布线图

4.1.2.2 设备端软件设计

微控制器的汇编程序任务主要是监视设备的状态，自动产生状态信息和用户命令信息，同时完成主机与设备之间的 USB 总线通讯，自动处理主机的控制和查询命令等等。最终编译后的机器代码总量在 2K 左右，按其功能可大致分为四个模块：

- USB 中断服务例程
- 命令处理器
- 手写板输入模块
- 报告处理器

USB 中断服务例程处理 USB 的不同的通讯信息（如令牌、数据或应答等），发送端点 0 的 SETUP、IN、OUT 等控制信息给命令处理器，另外协助报告处理器发

送待决的报告给中断端点 1。而手写板笔画数据报告的内容，则由手写板输入模块生成。

当 USB 设备第一次连接到总线上，它被指定为一个特定的地址。然后主机发送命令要求来检测设备特性并且选择不同的设备参数。命令处理器模块分析这些命令要求，按所要求的描述符和参数响应。由于 USB 手写板被定位为人机接口设备（HID），它不仅需要响应标准的 USB 协议要求，还要响应 HID 子协议的要求。同时为了完成手写板笔画信息的传输，设备还必须至少支持一种中断端点。另外为了使数据能被 BIOS 正确解释，USB 手写板必须按照报告定义的格式输入。报告处理器负责按规定格式转换手写板输入模块生成的数据（笔画信息），并请求中断服务例程通过中断管道发送报告。

手写板输入模块则随时准备接收“汉王”手写板发来的数据，修改报告数据字节，待一个完整的数据包（包括按键信息和笔点的 X、Y 绝对坐标）接收完成后，即通知报告处理器。

4.1.2.3 主机端软件设计

WINDOWS 98 下的 HID 设备驱动程序任务主要是与手写板通讯，并将数据以 WINDOWS 消息的方式发给“汉王”手写识别软件。该项目开始调研时，有关 WINDOWS 98 下驱动程序的资料还很少，至于 USB 一般设备或是 HID 设备的驱动程序文档就几乎是零了。后来随着工作的进行，曾一度因为没有详细的文档而无法知道某些技术细节。我曾尝试向中国微软寻求帮助，但他们对许多技术细节也不清楚。所以本项目的驱动程序，可以说就是在不断的假设和试验的过程中完成的。

该项目中最终的实现只有一个用户模式下的可执行文件，它可以可分为 HID 设备管理和 WINDOWS 消息发送两大模块。

HID 设备管理模块大量调用了 WINDOWS 98 系统提供的设备专用函数，完成与设备缓冲区的连接、对设备的识别、对设备数据报告的获取等功能。

鉴于篇幅有限，程序源代码无法提供，这里只能将各函数调用步骤和流程表示如下，各参数的赋值不再赘述。在 WINDOWS 98 驱动程序的编写中，开发者需要特别留意的，是内存和数据的占用和释放。一旦用户非法占用了某部分系统资源，或是占用了某些资源而未正常释放，WINDOWS 98 系统都有可能因此而瘫痪

甚至崩溃。

首先需要在程序执行时为 WINDOWS 98 中已安装的所有 HID 设备建立一个信息链表，这个链表为驱动程序的其他部分提供各个 HID 设备的接口。图 4.4 仅仅是这一步骤中最关键部分的流程和函数调用介绍，供读者参考。

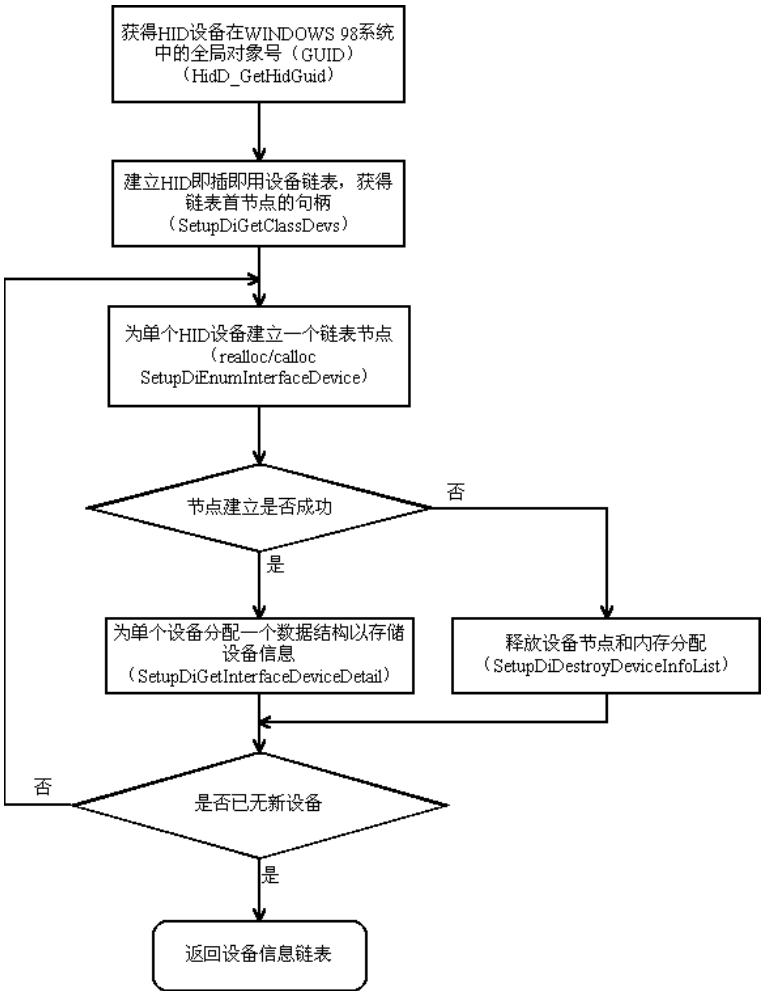


图 4.4　HID 设备链表的建立

第二步是获得单个 HID 设备的信息（图 4.5），不但有利于驱动程序正确地读取设备的数据报告，也可以使驱动程序从（可能是）众多的 HID 设备中分辨出自己要控制的设备，然后即可建立与该设备的直接连接，为以后读取数据报告作准备。当程序执行到图 4.5 的最后一步时，驱动程序已经获得了数个结构，存储或是指向 HID 设备的各项设定和属性。这时程序可以很简单地将设备与预先的设定

进行比较，以判断是否是自己要控制的设备。

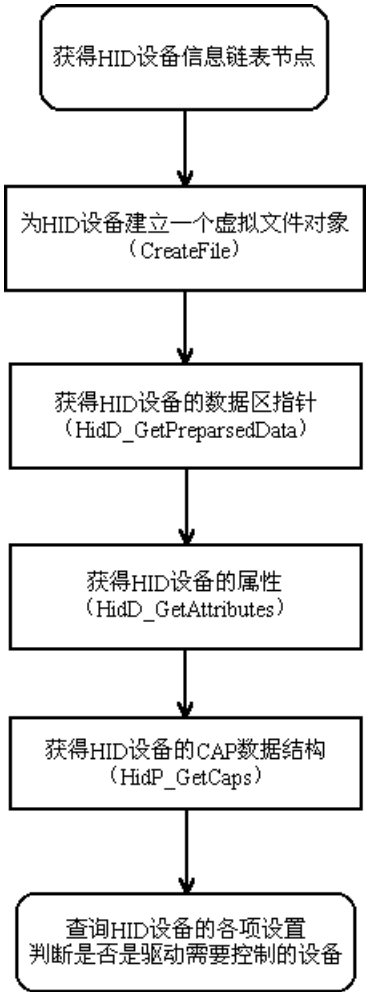


图 4.5　查询和判断 HID 设备信息

在与 HID 设备建立直接连接之后，驱动程序就可以直接读取设备缓冲区内的数据了。这里有一个同步的问题，就是驱动程序读取缓冲区的速度需要和设备发送数据的速度相当。如果驱动程序读取数据较慢，则有可能造成 WINDOWS 98 系统分配给设备的缓冲区被充满，而这时如果设备再向主机发送数据，就会发生数据的丢失；如果驱动程序读取数据较快，那么则有可能因为缓冲区无数据而使读取缓冲区的函数无法返回，最终造成驱动程序的停顿和系统资源的浪费。WINDOWS 98 DDK 提供了一个解决方法：改变系统分配的缓冲区的大小。但即使是在 1999 年 2 月份发布的 WINDOWS 98 DDK 正式版中，仍未提供可改变缓冲区大小的函数。所以本系统为了不遗失手写板数据，只好以牺牲部分系统资源为代价，始终保持

对设备缓冲区的读取状态。由于 WINDOWS 98 的进程调度遵从抢占式多任务调度，当缓冲区无数据时，系统不会始终让驱动程序占据 CPU 时间，所以虽然系统资源有所浪费，但使用效果还是让人满意的。

Windows 消息发送模块就负责在数据读取成功后，将数据以 WINDOWS 用户自定义消息的方式发送给“汉王”手写识别程序。对应着不同的按键信息和笔画信息，消息的 ID 和参数都是不同的，具体的内容详见表 4.1：

表 4.1　消息的 ID 和参数

设备	动作	消息 ID	消息参数 wParam	消息参数 lParam
左键	按下/释放一次	0X353	0	0
		0X356	0	0
		0X357	0	0
右键	按下/释放一次	0X358	0	0
		0X359	0	0
		0X353	0	0
笔	下笔	0X351	0	0X00yy00xx
	运笔	0X353	0	0X00yy00xx
	提笔	0X352	0	0X00yy00xx
注：“0X”表示 16 进制，“yy”表示笔点的垂直坐标，“xx”表示笔点的水平坐标				

WINDOWS 98 下发送消息有两个函数可以实现：SendMessage(…) 和 PostMessage(…)。SendMessage 发送消息时，发送方进程会停顿下来，直至接收方进程处理完消息并返回；而 PostMessage 发送消息后立即返回，发送方不理会消息是否被接收方进程处理。由于驱动程序比“汉王”手写识别程序优先级高，如果让驱动程序等待识别程序处理消息，往往造成双方都处于等待状态，使整个识别系统的响应速度降低。所以本项目中使用 PostMessage，实际运行结果证明，响应速度是让人满意的。

4.1.3 结论

该项目制作完成后通过了 MOTOROLA 公司和“汉王”公司专家的评定，都对最后的运行结果（图 4.6）表示满意。汉王公司从此选择了 Motorola 的微控制器作为它的 USB 手写板的核心控制器，并在该项目的基础上进行后续开发。同时该项目也获得了清华大学“挑战杯”科技作品竞赛二等奖。

图 4.6　USB 手写识别输入系统运行界面

4.2　USB 通用设备开发平台

4.2.1　背景

随着 USB 技术在国内外的迅速推广，从事 USB 设备开发的人员也越来越多。不过可惜的是，开发者所能获得的 USB 的技术支持都十分有限。目前国内外的厂家都只能提供处理物理层协议的芯片，开发者需要在设备端和主机端开发庞大的软件以处理 USB 复杂的协议请求。一套良好的易用的 USB 设备开发平台已成为所有 USB 开发者的迫切需要。

我和一位精仪系的硕士研究生（刘丁）在做各自的 USB 项目的过程中，都感觉到 USB 技术对普通的开发者而言具有相当的难度。如果能够有一套通用的 USB 平台，开发者在上面只需要稍做改动就能定制成自己的设备，那么 USB 设备的开

发就会简单多了。所以我们决定做一套标准的 USB 方案，推广给所有的 USB 开发者。这套方案的初衷是希望在硬件方面为开发者提供评估板和原理图；而在软件方面为开发者提供标准的设备端协议栈和主机端驱动程序的模板。

我们分析了当时的 USB 低速设备的方案，最终选择用 Motorola 的 8 位微控制器 MC68HC05JB4 来实现通用的低速设备方案。这个方案主要以 USB 手写识别输入系统为基础，对软件和硬件设计进行模块化的整理。由于这个方案使用的技术与 USB 手写识别输入系统没有太大的不同，所以在这里就不再赘述了。

在设计全速设备的方案时，我们发现很难找到好的单片方案，而在开始这个项目之前，刘丁已经使用 8051 控制 USBN9602 实现了一个全速 USB 设备的方案，测试结果还比较令人满意。所以我们又分别用 8051 和 MC68HC08GP32 控制 USBN9602 实现了通用的全速 USB 设备平台。

同时我还采用 Motorola 的 USB HUB 芯片 MC141555 设计实现了一个 USB 的 HUB，以方便对 USB 设备的调试，也使得 USB 设备开发平台成为一套完整的系统。

在所有工作完成后，该通用 USB 设备开发平台具有的特点是：

- 支持两种 USB 总线传输速率：低速（1.5Mbps）和全速（12Mbps）；
- 支持两大主流微控制器 CPU 内核：Intel 的 8051 和 Motorola 的 68HC08；
- 在设备端和主机端同时提供底层协议栈和驱动程序的源码和函数库；
- 硬件设计和软件开发的完美解决方案；
- 对 USB 通用设备和 HID 子类设备提供支持；

下面将主要讲述使用 MC68HC08GP32 控制 USBN9602 实现通用全速 USB 设备的原理和关键技术。

4.2.2 全速 USB 设备设计原理

该全速 USB 设备平台采用 MC68HC08GP32 作为设备端的控制核心，它通过 USBN9602 实现 USB 总线接口。PC 端安装了我们编写的客户驱动程序和动态连接库，使得用户程序可以与设备进行数据交换。（图 4.7）

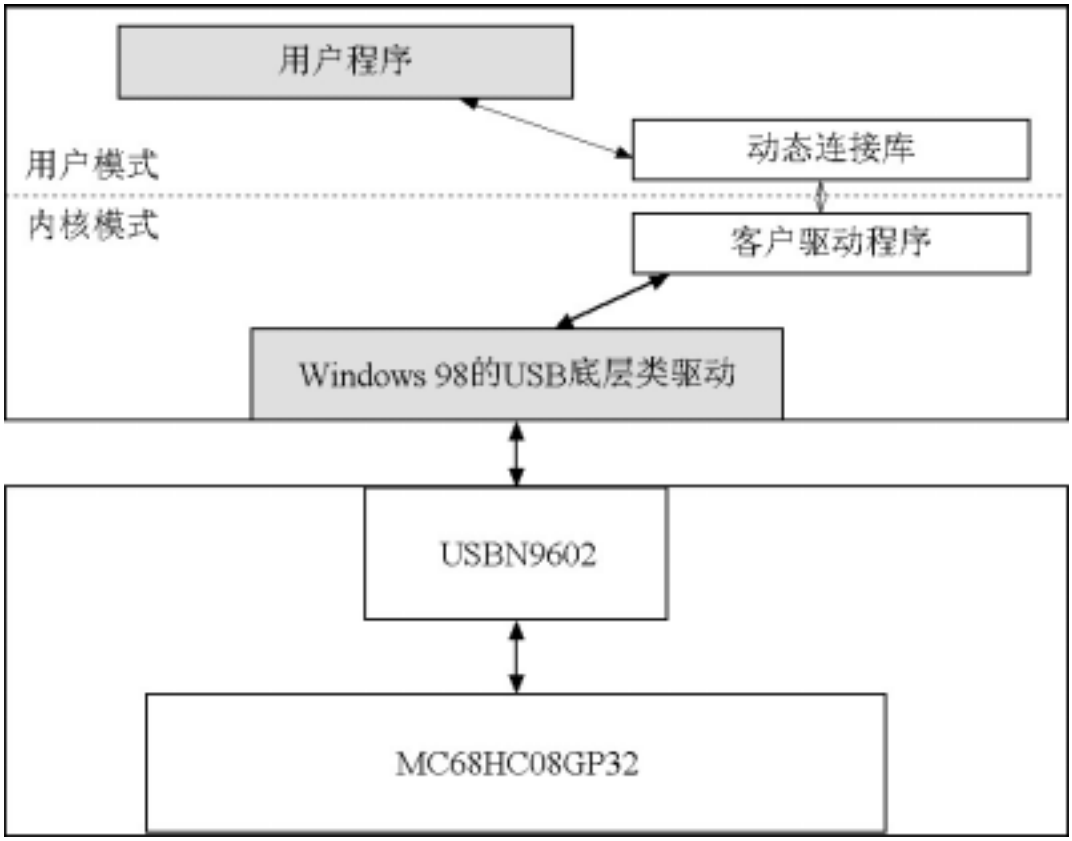


图 4.7　通用全速 USB 设备原理框图

4.2.2.1 设备端硬件设计

系统硬件的核心是 MOTOROLA 的微控制器 MC68HC08GP32，它用 I/O 脚模拟了一套外部总线接口与 USBN9602 通讯，而 USBN9602 则通过 USB 总线接口直接与 PC 相连。MC68HC08GP32 使用的时钟是 4.9MHz 的晶振，而为了产生 12MHz 的 USB 总线时钟，USBN9602 使用的是 48MHz 的晶振。（图 4.8,4.9）

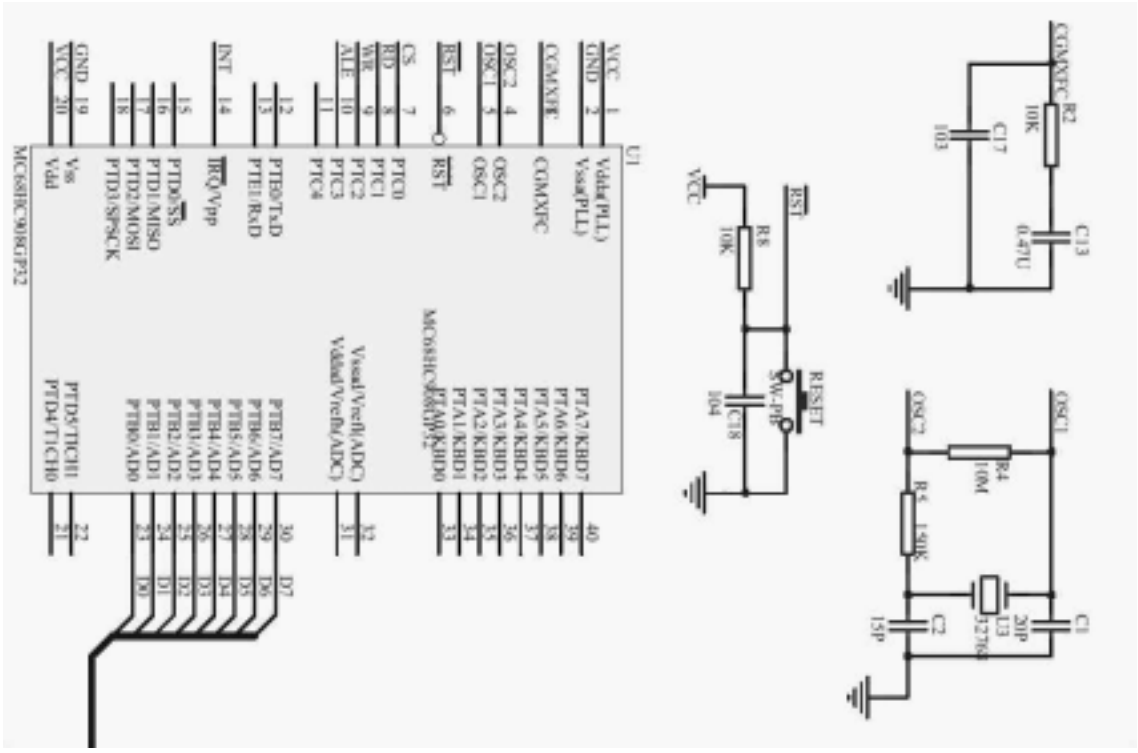


图 4.8　全速 USB 设备整体电路原理图一

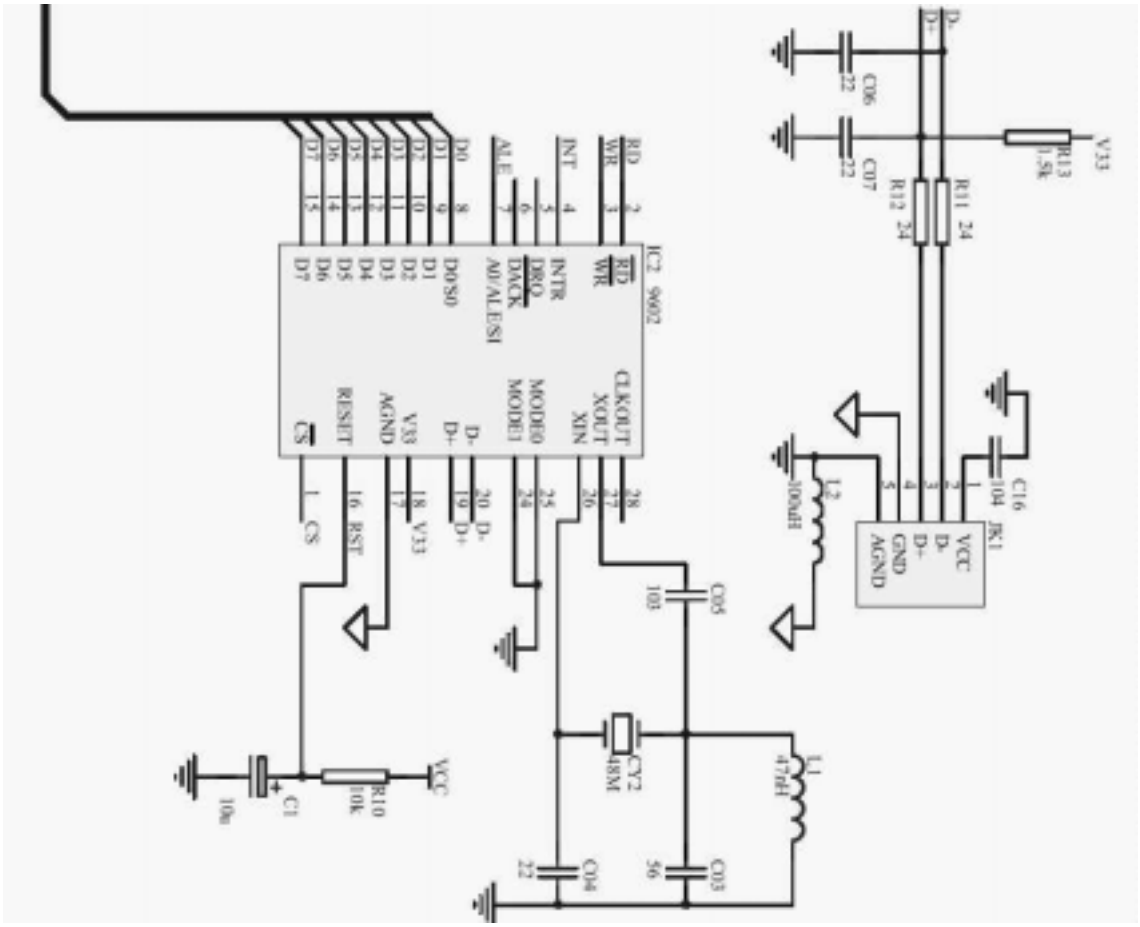


图 4.9　全速 USB 设备整体电路原理图二

由于本系统使用了 48Mhz 的时钟源，所以在布线时需要格外注意电磁兼容性。同时由于 USBN9602 本身不是太稳定，所以推荐用户使用有源晶振，而不使用普通的无源晶体。

4.2.2.2 设备端软件设计

由于我们要实现的只是一个通用的 USB 设备平台，所以软件方面就只需要实现一个通用的 USB 设备协议栈。

协议栈的结构和内容和 USB 手写识别输入系统非常类似，并且由于这个项目实现的设备属于通用设备类，不需要再处理 HID 子类的命令，所以较 USB 手写识别输入系统而言更为简单，在本文中就不再赘述了。

MC68HC08 系列微控制器与 MC68HC05 系列相比，最大的优点就是添加了堆栈

操作的指令，从而使开发者可以用 C 语言编写芯片的程序。这个项目的所有设备端代码都是用 C 语言书写的，具有很好的可移植性。

4.2.2.3 主机端软件设计

由于设备属于通用 USB 设备类，所以 Windows 下的内核层客户驱动程序就是不可缺少的了，这也成为了本项目最复杂和困难的部分。

限于篇幅，内核层 USB 客户驱动程序的流程和结构无法在本文中详细地叙述出来，我只能将通过 USB 类驱动程序访问 USB 设备的一般过程在这里列举一下，详细的资料请参阅 Windows DDK 的技术文档、Chris Cant 著的《Writing Windows WDM Device Drivers》（中译本名为《Windows WDM 设备驱动程序开发指南》，机械工业出版社出版）和 Walter Oney 著的《Programming the Microsoft Windows Driver Model》。

如前文所述（3.3.2），USB 类驱动程序提供了一套 USB 驱动程序接口（USBDI），客户驱动程序通过一系列的内部 IOCTL（表 4.2）与 USB 类驱动程序通讯。

表 4.2 USBDI 的内部 IOCTL

IOCTL_INTERNAL_USB_SUBMIT_URB	发出 URB
IOCTL_INTERNAL_USB_RESET_PORT	复位并重新启动一个端口
IOCTL_INTERNAL_USB_GET_PORT_STATUS	得到端口状态位
IOCTL_INTERNAL_USB_ENABLE_PORT	重新启动一个禁用的端口
IOCTL_INTERNAL_USB_GET_HUB_COUNT	（由集线器驱动程序在内部使用）
IOCTL_INTERNAL_USB_GET_ROOTHUB_PDO	（由集线器驱动程序在内部使用）
IOCTL_INTERNAL_USB_GET_HUB_NAME	得到 USB 集线器的设备名
IOCTL_INTERNAL_USB_GET_BUS_INFO	填写 USB_BUS_NOTIFICATION 结构 （仅用于 Windows 2000）
IOCTL_INTERNAL_USB_GET_CONTROLLER_NAME	得到主机控制器名 （仅用于 Windows 2000）
IOCTL_INTERNAL_USB_CYCLE_PORT	模拟设备拔出和再次插入

其中最重要的内部 IOCTL 是 IOCTL_INTERNAL_USB_SUBMIT_URB，它发出 USB 请求块（URB），并交由 USB 类驱动程序处理。有 30 多个不同的 URB 功能代码。USB 客户驱动程序就是使用这些 URB 完成它们的大部分工作的。为了方便客户驱动程序构造合适的 URB，Windows 提供了各种构造宏（如

UsbBuildGetDescriptorRequest），它们会为 URB 分配内存并填写。

USB 客户驱动程序可以使用以下的步骤向 USB 类驱动程序发送内部 IOCTL：

1) 构造一个 IRP，并将正确的内部 IOCTL（如 IOCTL_INTERNAL_USB_SUBMIT_URB）填入其中；
2) 构造适当的 URB；
3) 在 IRP 的下一个堆栈单元中存储 URB 的指针；
4) 将 IRP 传递给 USB 类驱动程序；
5) 等待 IRP 处理完毕，得到处理结果。

在具体实现时，USB 客户驱动程序首先应该初始化与 USB 设备的连接，这包括下面几步工作：

1) 检查设备是否已经启用。如果必要，复位并重新启用设备；
2) 在设备支持的所有配置（Configure）中选择一个接口（Interface）；
3) 读取设备的其他描述符，如字符串描述符、类描述符等；（这一步也可以略过）
4) 访问设备，发出相关的命令，读取设备状态，以及初始化管道（Pipe）。

在初始化完成后，客户驱动程序就可以通过管道与设备对话了。

上面所讲的，主要是如何与 USB 类驱动程序通讯，这是客户驱动程序下沿的工作。同时 USB 客户驱动程序还应该向用户层应用程序提供访问的接口。它需要象 USB 类驱动程序一样，向用户层应用程序显露一个设备接口，允许应用程序读取设备的数据，响应应用程序发出的各种 IOCTL。在本项目中，客户驱动程序支持应用程序发送打开设备、读取数据、发送数据和关闭设备等 IOCTL。

我编写的用户层动态连接库就是通过发送 IOCTL 与内核层客户驱动程序通讯的，同时这个动态连接库又向其他用户的应用程序提供了 4 个函数接口，它们是：DeviceOpen（打开设备）、DeviceClose（关闭设备）、ReadDevice（读取设备数据）和 WriteDevice（向设备发送数据）。

这样用户在进行开发时，只需要在自己的应用程序中使用这 4 个函数就可以完成与 USB 设备的通讯了。USB 的主机端软件工作变得非常简单。

4.2.3 USB HUB（集线器）设计原理

为了向开发者提供全套 USB 系统方案，我们使用 Motorola 的 USB HUB 专用

芯片 MC141555 设计实现了一个有 4 个下行口的 USB HUB。

由于 Motorola 已经将 USB HUB 的代码掩模在 MC141555 中，所以设备端不需要进行任何软件开发；同时 Windows 也为 USB HUB 编写了标准的驱动程序，所以在主机端也没有任何软件开发的工作要做。我需要完成的只是硬件电路的设计和调试。（图 4.10、4.11、4.12）

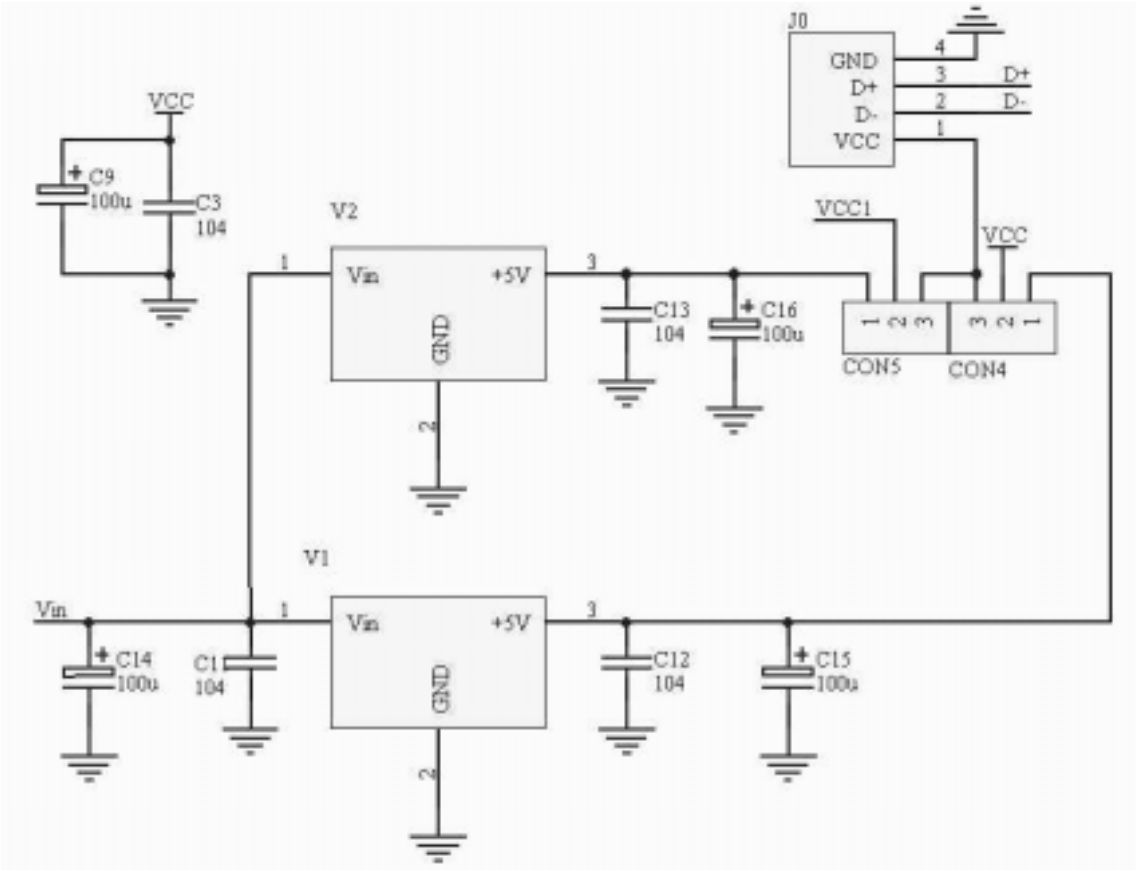


图 4.10 USB HUB 电路原理图一

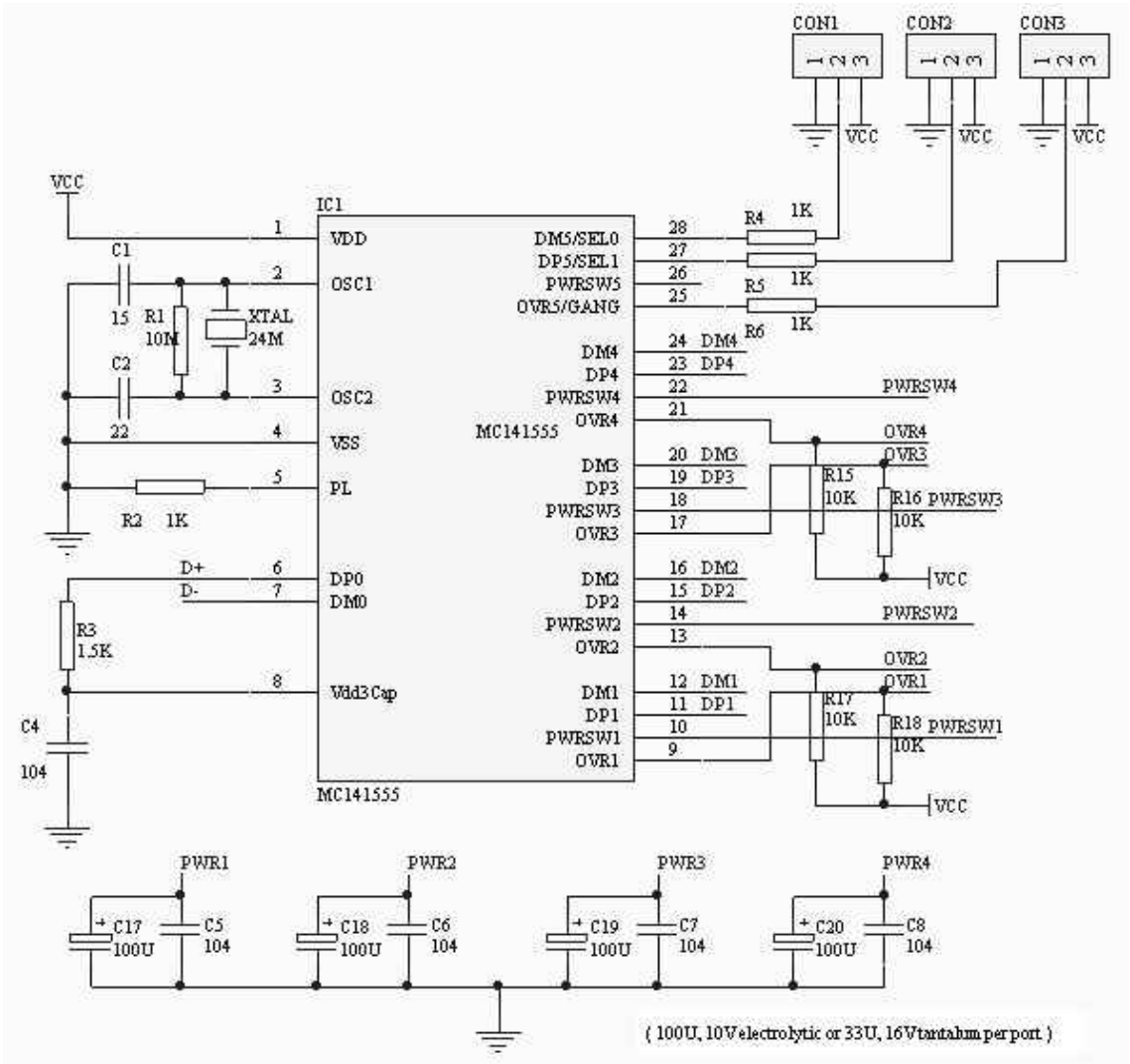


图 4.11 USB HUB 电路原理图二

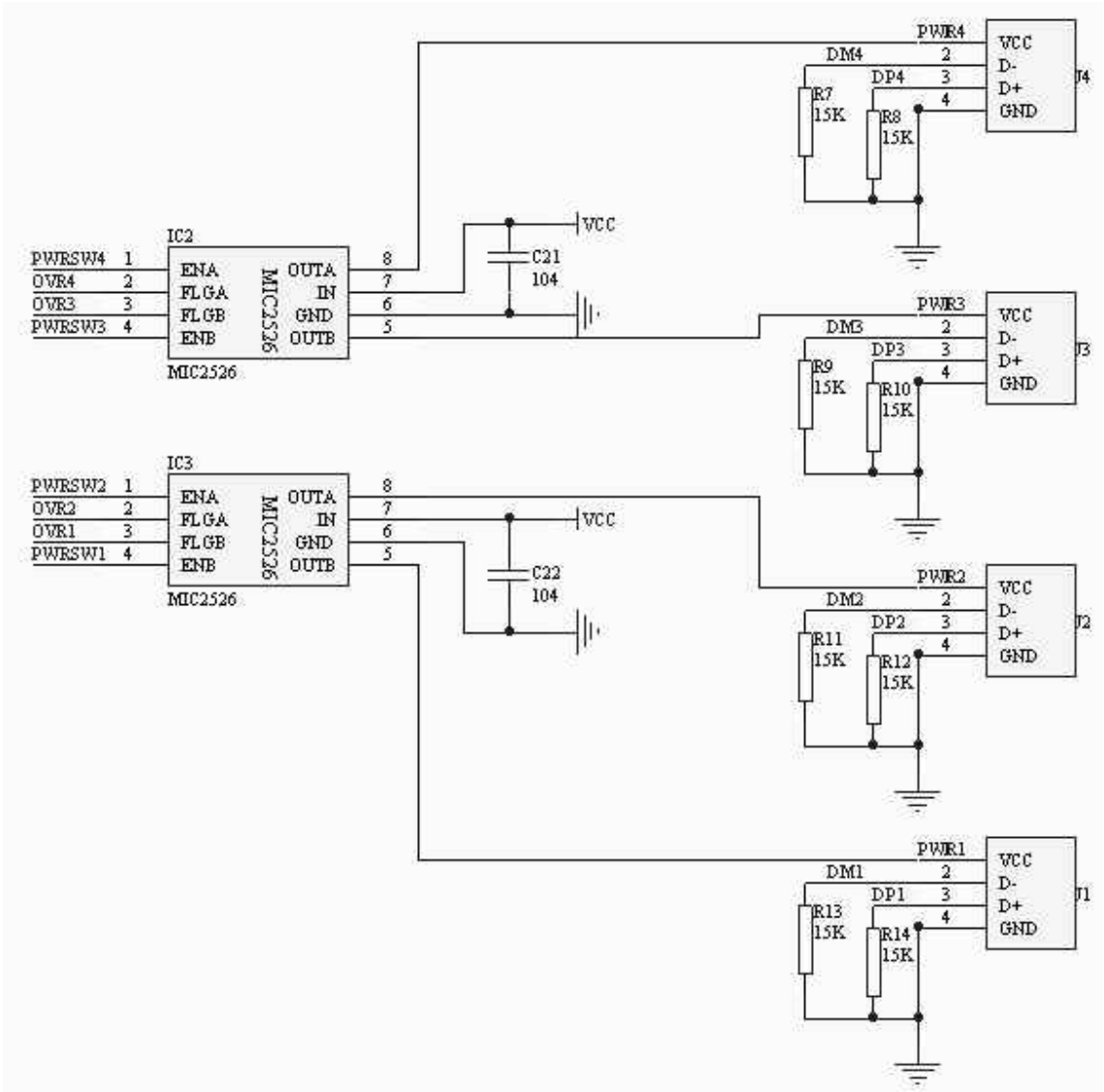


图 4.12 USB HUB 电路原理图三

MIC2526 是专为 USB HUB 设计的电源开关，它可以在 USB 下行端口拉出的电流过大时自动切断该端口的供电，起到保护 HUB 和整个 USB 系统的功能。MC141555 也可以直接通过它的使能信号脚（ENA 和 ENB）打开或关闭下行端口的电源供应。

MC141555 支持总线供电和自供电两种方式，我在电路板上提供了跳线（CON1、CON2 和 CON3）让用户选择设置。它们同时还决定了 HUB 的其他一些功能设置（表 4.3）。

表 4.3 USB HUB 跳线设置

	自供电&各端口独立	总线供电&各端口独立	总线供电&各端口相关
CON1	0	1	1
CON2	0	0	0
CON3	-	1	0

4.2.4 结论

USB 协议还在不断更新，而 USB 开发者的任务也随之不断加重。这套 USB 通用设备开发平台将普通 USB 设备开发周期从半年缩短到一个月以内，大大减少了 USB 开发者的工作量，为产品的推出节省了宝贵的研发时间，一经推出就受到大量 USB 设备开发人员的关注。

在 2000 年的清华大学“挑战杯”科技作品竞赛上，这个项目受到了所有师生的好评，获得了一等奖的荣誉。

4.3 USB 安全钥

4.3.1 背景

在目前的电子商务模式中，身份认证大致可以通过两种方法来实现：一是交易双方通过协商选定一个协议，用来在网络上确认对方的身份。最简单的方法便是使用密码（Password），只有能正确说出密码的人才能参与交易。二是交易双方请求第三方机构代为确定对方的身份。这个第三方机构必须具有足够的权威性，是交易双方所共同信任的。各种商业银行都可以充当这一角色。

不论采用何种方式，安全性和方便性都是不可缺少的。但是在目前的电子商务模式中，这两项要求都没有得到很好的满足。

国内的网络终端大多是 PC 机，运行 WINDOWS 操作系统。然而 WINDOWS 系统并不是一个足够安全的系统，系统级的缺陷使得在它上面运行的应用软件都可以被很容易地跟踪、破解。为了克服这一缺点，用户不得不在 PC 上安装尽可能复杂的加密软件，在交易过程中采用尽可能多的认证程序，虽然这样能稍微增强一些系统的安全性，但是交易的方便性却受到了极大的影响。最明显的一点便是，用户无法随意选择一台网络终端（PC）进行交易，因为用户必须确保自己使用的

网络终端上已经安装了完善的加密软件，和具有足够的数据安全性（也就是说，其他人无法轻易地得到这台网络终端上储存的数据）。

可惜的是，WINDOWS 系统并不是我们自己开发的，要提高它的安全性也不是我们能轻易做到的。唯一的解决办法便是借助于硬件的支持，将身份认证过程中的关键部分（加密算法和个人数据）从 PC 移植到硬件中去。目前最常用的硬件设备按功能分为两类：一是用于加密，例如 PC 内安装的加密卡、网络中安装的安全网关等等；二是用于个人数据存储，例如 IC 卡，磁卡等，在银行系统使用十分广泛。PC 机内安装的加密卡等硬件虽然大大地增强了系统的安全性，但它们价格昂贵，安装、使用不方便，是普通的 PC 用户无法接受的；而 IC 卡、磁卡等存储卡则需要在 PC 机上安装读卡器，读卡器价格昂贵，兼容性差，也不是任意一台 PC 机就能轻易安装得上的。

随着硬件技术的飞速发展，USB 安全钥（USB Security Key）出现在了电子商务的硬件大家庭中，它集数据加密和数据存储两大功能于一身，成为了推动电子商务发展的强大动力之一。

在我刚开始这个项目的时候，国外有一家叫 Rainbow 的公司已经在全球范围内推出了它们的 USB 钥，但提供的功能只限于软件和文件加密等不涉及网络应用的功能。我推出的 USB 安全钥主攻的方向便是网络的身份认证和电子商务领域，它具有以下特点：

- 体积小：和普通钥匙相当，可以挂在钥匙串上随身携带；
- 兼容性好：由于设备本身符合 USB 各项协议定义，在不同的计算机平台（PC、MAC 等）和操作系统（WINDOWS、LINUX 等）中均能使用；
- 使用方便：具有即插即用功能，用户可以在交易过程中将安全钥插入网络终端，使用完毕后可以立即拔出，不需要中止程序或操作系统的运行；
- 功能强大：同时具有数据加密和数据存储两大功能，加密算法可以支持 DES、MD5 和 RSA 等等，数据存储容量一般都在 1K 字节以上；
- 安全性好：在使用过程中将网络终端的功能弱化为一个数据传输的媒介，所有有关安全性的操作都在 USB 安全钥和服务器内部完成，极好地保护了安全数据。在这一前提下，用户可以携带 USB 安全钥在任意一台网络终端上进行交易，而不必担心会有任何重要数据留

在使用过的网络终端上；

- 价格便宜：USB 安全钥实际上就是一个带有 USB 接口的微控制器，这类微控制器的成本在 1～1.5 美元之间。

在这个项目中，我一方面设计实现了 USB 安全钥所必需的硬件和软件，另一方面利用 USB 安全钥，设计制作了一个网络身份认证的演示系统。

4.3.2 原理

本项目的底层设计与 USB 手写识别输入系统类似，也是选择实现一个 HID 类设备，这样可以在主机端省去编写内核层客户驱动程序的工作量，也为用户在自己的系统中安装设备提供了方便。与 USB 手写识别输入系统相比，USB 安全钥的独特之处在于：

- 外形要求高，电路板的尺寸要尽量小；
- 使用了 Motorola 最新推出的 MC68HC908JB8，取代 MC68HC05JB4；
- 为了存储用户信息，使用了 MC68HC908JB8 的在线编程技术；
- 将加密算法用微控制器的汇编语言来实现等等。

4.3.2.1 设备端硬件设计

USB 安全钥的所有工作都是在芯片内部完成的，外围硬件非常简单，只需要芯片必需的时钟电路和 USB 接口电路就行了。（图 4.13，图 4.14）

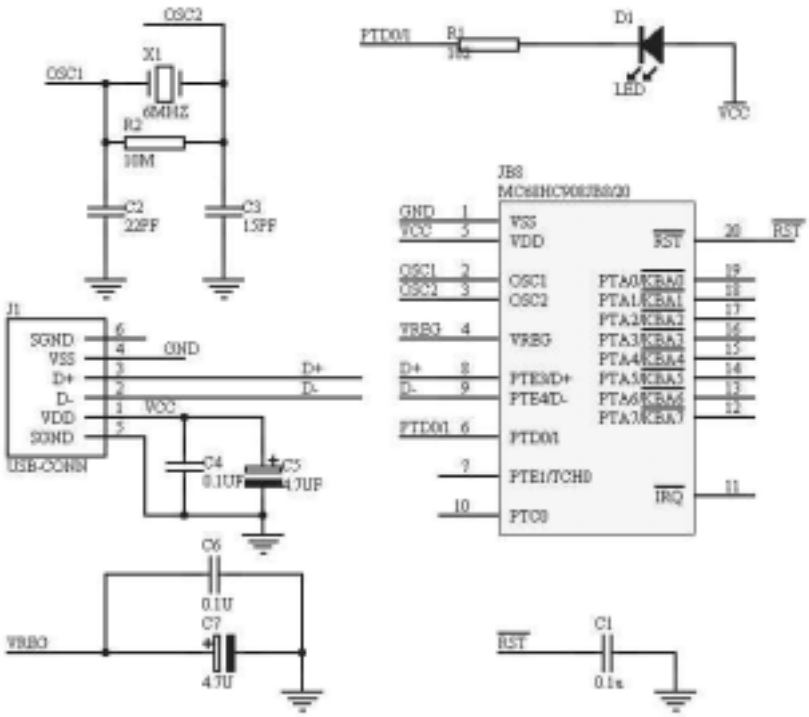


图 4.13 USB 安全钥电路原理图

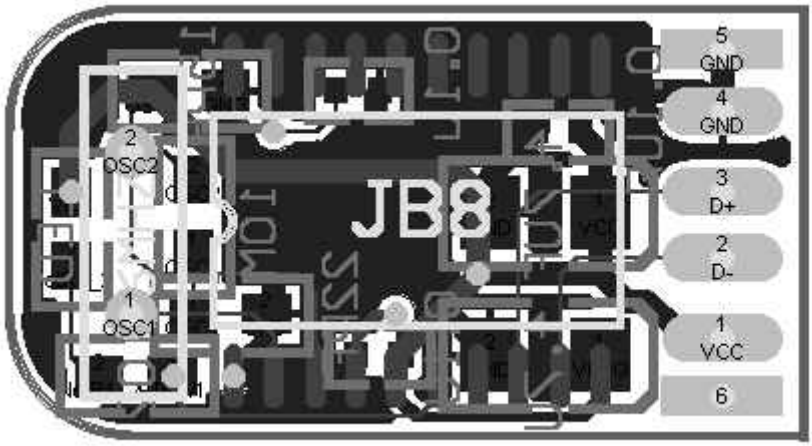


图 4.14 USB 安全钥印制板图

4.3.2.2 设备端软件设计

设备端软件设计的任务除了需要实现 HID 设备的协议栈以外，还需要实现：

- DES 和 MD5 的算法
- 对 FLASH 的在线编程

HID 设备协议栈的内容已经在 USB 设备端软件开发（3.2）一节中叙述过，这里就不再赘述了。

DES 和 MD5 的算法实现是在 Motorola 香港总部的工程师的协助下完成的。这两个算法都是国际上标准的算法，C 语言的实现是不难找到的，但要利用微控制器有限的 RAM 和 ROM 来实现，却有着相当的难度。由于这一部分涉及的技术过细，限于篇幅，就不能在这里多加叙述了。

Motorola 在 FLASH 技术已相当成熟的时候才推出集成片内 FLASH 的 8 位微控制器，在应用方便性和可靠性上均有不俗的表现，具体体现在：

- 单一电源电压供应：FLASH 在正常的只读情况下，只需要用户为其提供普通的工作电压（如+5V）。如果用户要对 FLASH 编程，需要同时提供高出正常工作电压的编程电压（如+9V）。Motorola 通过在片内集成电荷泵，只需要用户提供单一+5V 工作电压，便可以在片内产生出编程电压。这使得用户无需为 FLASH 的编程而在目标板上增加额外的电源。
- 可靠性高：Motorola 片内 FLASH 的存储数据可以保持 10 年以上，可擦写次数也在 1 万次以上。
- 擦写速度快：Motorola 片内 FLASH 的整体擦除时间可以控制在 5ms 以内，对单字节的编程时间也在 40μs 以内。

最重要的是，Motorola 片内 FLASH 支持在线编程（In-Circuit Program），允许微控制器内部运行的程序去改写 FLASH 存储内容。这一技术大大增加了 Motorola 微控制器的应用范围和使用方便性，也使得用 MC68HC908JB8 存储用户数据在技术上成为了可能。

MC68HC908 系列微控制器的片内 FLASH 可以在两种模式下进行在线编程：监控模式（Monitor Mode）和用户模式（User Mode）。USB 安全钥使用的是用户模式下的在线编程技术。它是在微控制器正常工作的过程中，通过 USB 接口从主机获得编程数据，再利用程序的代码转至对 FLASH 的编程操作，以完成对用户信息的存储。

Motorola 在 MC68HC908JB8 的监控 ROM 中已经固化了对 FLASH 进行擦除和写

入的子程序，用户程序只需要设置好各个寄存器和变量的值，然后用 JUMP 指令调转到相应的子程序入口执行就可以了。有关这些子程序的说明，可以在 Motorola 的应用文档 AN1831 中找到。

有关 Motorola 微控制器在线编程技术的详细说明，请参看 5.1.1.1。

4.3.2.3 主机端软件设计

由于 Windows 98 和 Windows 2000 已经为 HID 设备提供了完善的驱动程序，所以不需要为 USB 安全钥编写任何驱动程序。我主要的工作是设计并实现了一个网络身份认证的演示系统，以说明 USB 安全钥的用途。

这套系统的使用流程如图 4.15。

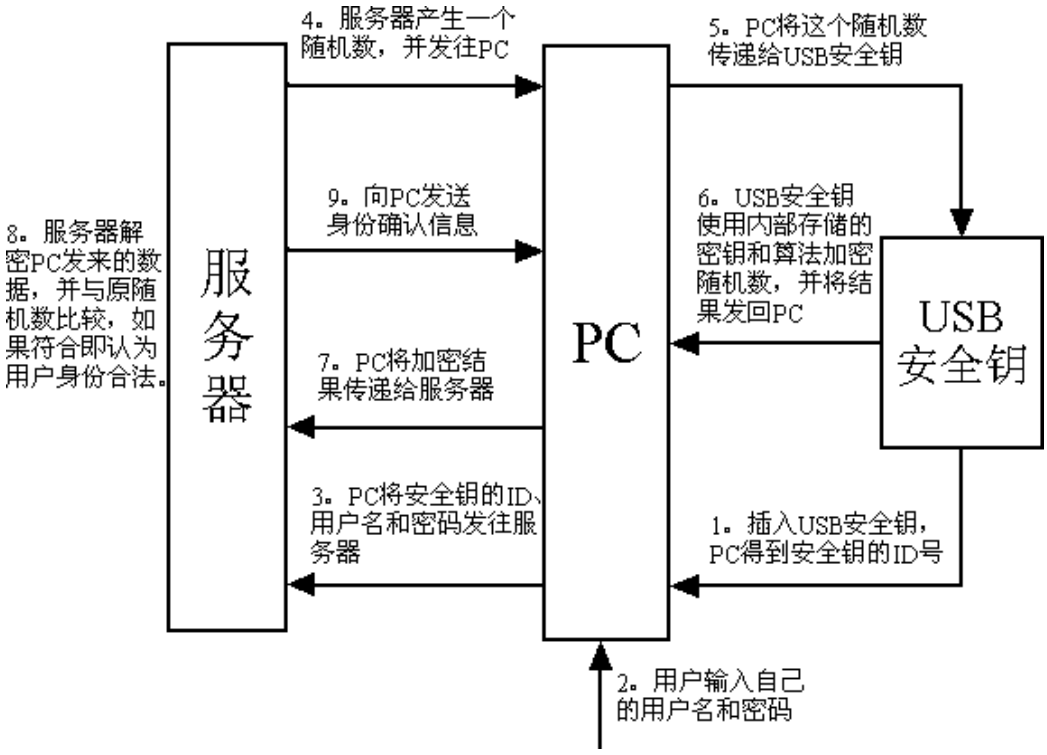


图 4.15 USB 安全钥演示系统流程图

首先，用户需要将 USB 安全钥插入 PC，让 PC 读出安全钥的 ID 号（步骤 1）；然后用户需要输入自己的用户名和密码（步骤 2）；PC 将安全钥的 ID、用户名和密码一起发往服务器，并请求服务器对用户进行身份认证（步骤 3）；在这个应

用中，服务器存储着所有 USB 安全钥的密钥，从 PC 得到安全钥的 ID、用户名和密码后，服务器便可以从数据库中提出正确的密钥，同时服务器产生一个随机数，并发往 PC（步骤 4）；PC 将收到的随机数原封不动地发给 USB 安全钥（步骤 5）；USB 安全钥使用内部存储的密钥和算法加密随机数，并将结果发回 PC（步骤 6）；PC 再将加密结果原封不动地传递给服务器（步骤 7）；服务器使用正确的密钥解密 PC 发来的加密数据，如果结果与原随机数相符，就认为用户的身份合法（步骤 8）；最后，服务器将用户身份的认证结果发往 PC（步骤 9）。

普通 PC 上的应用程序通过调用 Windows 提供的标准的 API 与 HID 设备（USB 安全钥）通讯，同时利用 SOCKET 技术与服务器通过 TCP/IP 协议通讯。服务器端的软件只需要维护一个数据库，支持一个加密算法和随机数产生算法就可以了。在 Windows 的环境下要实现这些功能都是非常简单的，所以在这里也就不再赘述了。（图 4.16，4.17）

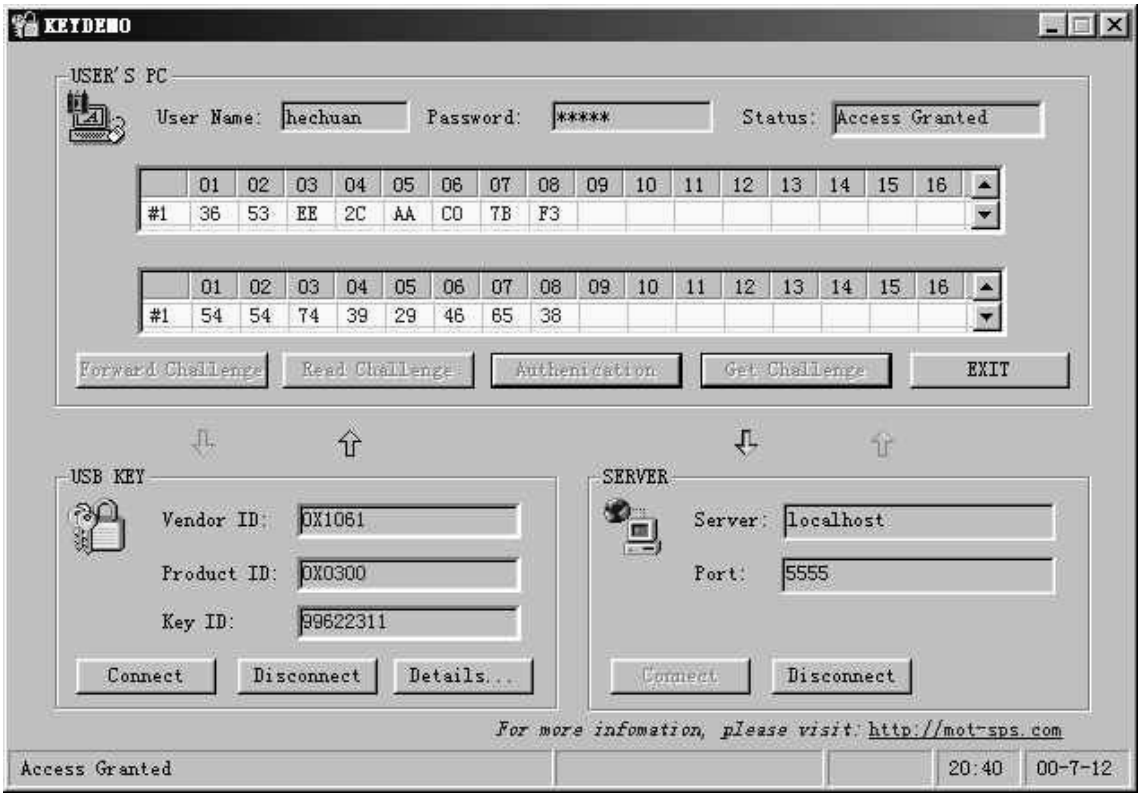


图 4.16 USB 安全钥演示系统 PC 软件界面

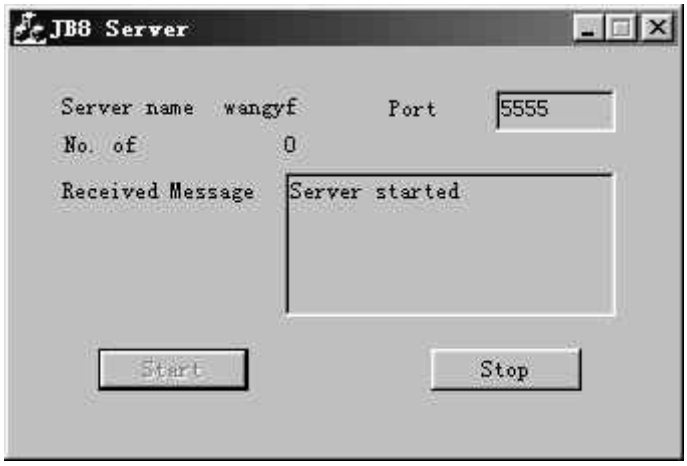


图 4.17 USB 安全钥演示系统服务器端软件界面

4.3.3 结论

目前这个方案已经交由 Motorola 向客户推广，TCL 等许多厂家都在开始使用这一方案制作自己的产品。

USB 安全钥的应用领域很广，几乎覆盖了 IC 卡和磁卡的所有应用（包括银行 ATM 取款卡、信用卡等），同时又由于其全新的即插即用功能在 PC 的相关应用中扮演了重要的角色。针对不同的应用，USB 安全钥可以由用户定制，嵌入不同的加密模块。目前我国政府和权威的安全机构都特别强调使用有中国自己的知识产权的加密算法和软件，对于比较敏感的系统和部门，都需要使用自行开发的加密算法，所以国内的相关公司都在注意开发自主知识产权的电子商务算法和软件。

随着 USB 安全钥的发展，电子证书等传统电子商务的核心技术都可以和它相结合。USB 安全钥一定能在未来的电子商务应用中发挥越来越重要的作用。

4.4 USB 在线编程设备

在完成了前面的项目以后，我实际上已经掌握了：

- USB 设备端通用协议栈和 HID 子类协议栈的编写方法
- 主机端 HID 设备的用户层应用程序编程方法
- 主机端通用 USB 设备的内核层客户驱动程序和用户层应用程序的编写方法

在有了这些技术做基础以后，很多 USB 项目就只需要简单的把某些技术整合

在一起就可以了。USB 在线编程设备就是这样实现的。

4.4.1 背景

一年前，Motorola 准备推出一款支持 USB 显示器的 8 位微控制器 MC68HC08LD64。这款芯片内置近 64K 的 FLASH，支持在线编程技术。有关在线编程技术的具体描述，请参看 5.1.1.1。

显示器芯片内部一般都会存储一些与显示有关的算法程序，而这些算法在不同的显示器型号里是不一样的。厂家在推出某一批显示器的时候，芯片内的算法程序是一定的，如果发现问题需要更改，必须将显示器从市场上收回并重新换上写入新程序的芯片。使用 Motorola 推出的这款芯片，就可以随时更改显示器里的芯片程序了。即使是在用户购买显示器以后，也可以由用户自己完成芯片内代码的更新和升级。同时芯片内没有使用的 FLASH 还可以用于存储显示器的设置信息，这样就可以永久保存用户对显示器的设置了。

项目确定以后，Motorola 的工程师负责编写固化在芯片内部的与在线编程有关的程序，而我负责编写 Windows 下的 USB 客户驱动程序和用户程序。

实际上在线编程技术在很多领域都可以有很好的应用，所以 Motorola 希望我们也帮 MC68HC908JB8 实现一个在用户模式下进行在线编程的演示系统。这个项目在设备端和主机端的软件开发都是由我完成的，下面也将以它为重点介绍。

4.4.2 原理

在 USB 在线编程设备正常工作时，与在线编程有关的监控代码是不会被执行的，用户程序负责与主机通讯（对 USB 显示器而言，使用的是 HID 子类协议），完成正常的工作。（图 4.18）

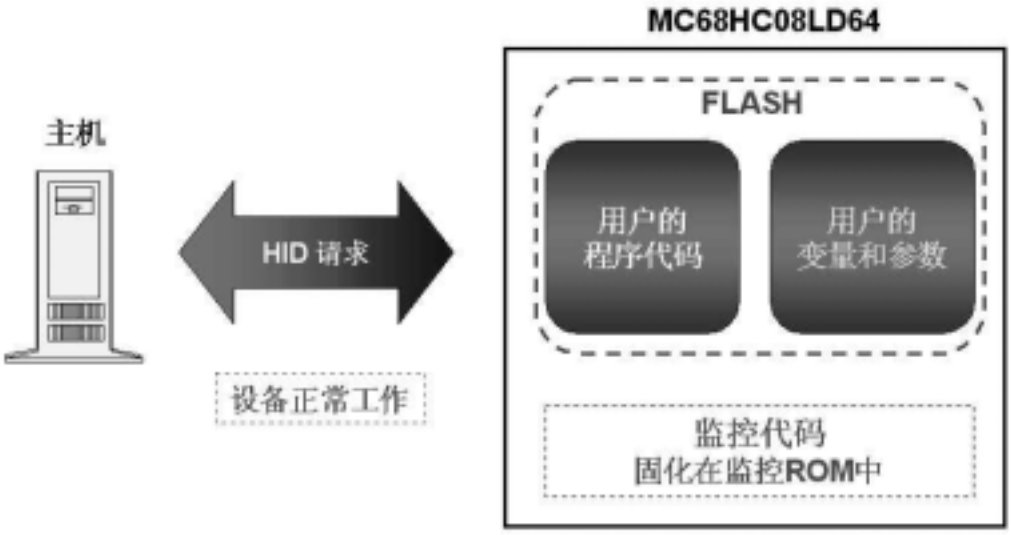


图 4.18 MC68HC08LD64 正常工作时

一旦用户需要对芯片内的程序代码和参数进行更新，可以控制设备进入监控模式，这时芯片会自动执行监控 ROM 中的代码。监控代码将把设备配置为一个新的 USB 设备，主机会认为原来的 USB 显示器已经被用户拔除，新插入的是一个支持在线编程的 USB 通用设备。

接下来，用户就可以通过我的用户程序和内核层客户驱动程序对芯片内的 FLASH 进行在线编程了。（图 4.19）

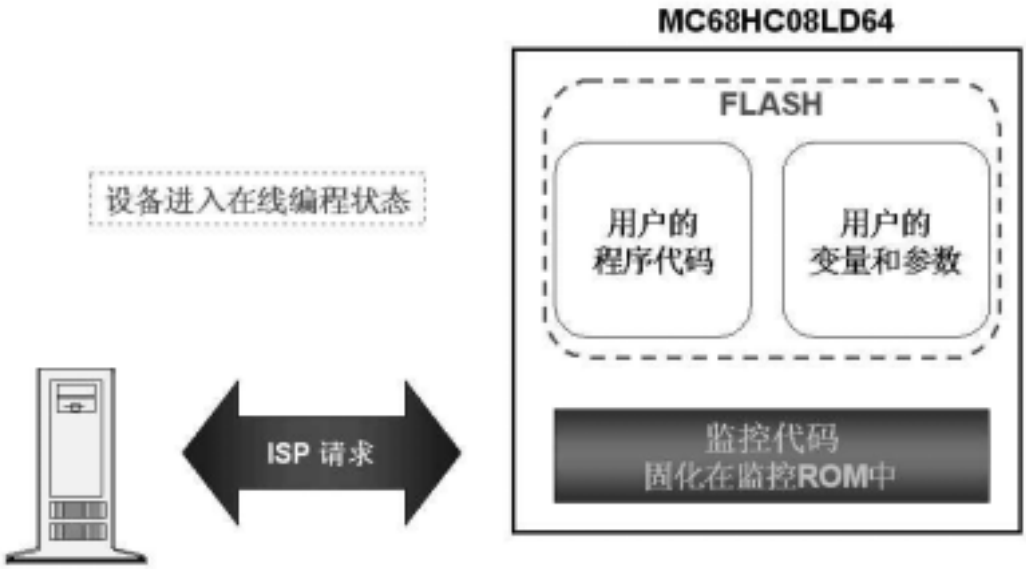


图 4.19 MC68HC08LD64 进入在线编程状态后

MC68HC08JB8 在监控 ROM 中没有驻留在线编程设备的代码，所以我将 MC68HC08JB8 的 FLASH 区域分为了两个部分，一个部分（地址较高）存储在线编程设备的代码，另一个部分存储用户的程序代码和参数。正常工作时，与在线编程有关的代码同样是不会被执行的。（图 4.20）

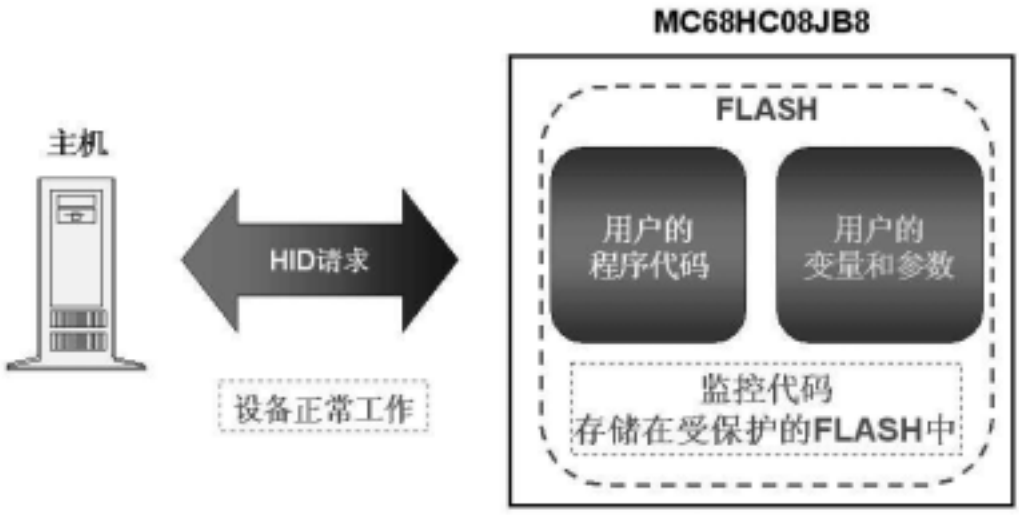


图 4.20 MC68HC08JB8 正常工作时

一旦用户需要对芯片内的程序代码和参数进行更新，可以控制设备转入对监控代码的执行。由于此时芯片仍处于用户模式下，没有任何硬件手段可以支持这种从用户代码到监控代码的跳转，所以在 MC68HC08JB8 这个项目中，需要用户程序提供跳转到监控代码的指令。用户程序可以探测用户有否按下某个按键或是 PC 有否发出转到监控代码的命令，如果有，用户程序应该立刻关闭所有中断，并跳转到监控代码的入口处开始执行。监控代码将把设备配置为一个新的 USB 设备，主机会认为原来的 USB 显示器已经被用户拔除，新插入的是一个支持在线编程的 USB 通用设备。

接下来，用户就可以通过我的用户程序和内核层客户驱动程序对芯片内的 FLASH 进行在线编程了。（图 4.21）

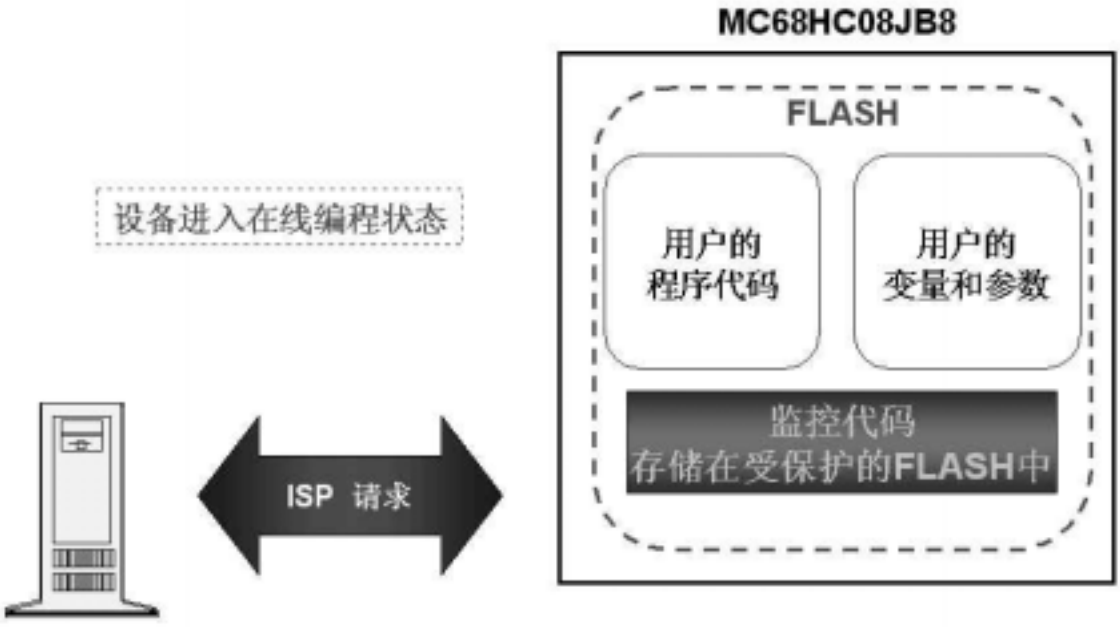


图 4.21 MC68HC08JB8 进入在线编程状态后

4.4.2.1 设备端软件设计

在用 MC68HC08JB8 实现在线编程设备的时候，需要编写一个通用 USB 设备的协议栈。这个协议栈与以往用 Motorola 的 8 位微控制器实现的协议栈有两个不同的地方。

一是这个协议栈始终是在中断关闭的情况下运行的。由于芯片的中断向量是用户程序编写的，所以我无法在监控代码里编写任何中断处理程序，也不允许在监控代码的执行过程中发生任何中断，因为这都会导致芯片转入对用户代码的执行。我只能不断查询 USB 中断寄存器的状态，判断是否有中断被触发，然后转到相应的处理程序中去。

二是需要用软件模拟一个 USB 设备插入和拔出的过程。因为在芯片由对用户程序的执行转入对监控代码的执行以后，PC 不会觉察到设备的变化。本来在正常情况下，应该由用户将设备拔出，再重新插入，让 PC 自动对新的在线编程设备进行配置和分配资源。但由于使用 MC68HC08JB8 的 USB 设备大多是总线供电设备，所以热插拔会直接导致芯片的复位，而芯片一旦复位，将会自动执行用户程序，PC 一样不会发现在线编程设备。监控程序必须在自己的运行过程中模拟一

个 USB 设备插入和拔出的过程。

如前文所述（2.2.3），USB 的 HUB 是依靠 D－上有高电平来判断低速设备的插入的，而 D－上的高电平是通过在设备端上拉到 3.3V 的标准电平上获得的。MC68HC08JB8 支持用软件设置这个上拉电阻是否有效。当需要 PC 检测到设备的插入时，程序应该允许上拉电阻，使 D－上出现高电平；当需要 PC 认为设备已拔出时，程序应该禁止上拉电阻，是 D－在主机或 HUB 端被下拉到低电平。

在监控代码进行初始化的时候，它需要先禁止上拉电阻，使 PC 以为原 USB 设备已经拔除，在等待几秒以后（让 PC 有充足的时间释放为原有 USB 设备分配的资源，并准备好检测新设备的插入），再允许上拉电阻，使 PC 自动检测到在线编程设备的插入。

4.4.2.2 主机端软件设计

这个项目的 USB 客户驱动程序是直接从 USB 通用设备开发平台中移植过来的，所以相关的技术细节就不再赘述了，请参阅 4.2.2.3。

用户程序提供了一个类似编程器的界面（图 4.22），用户可以通过它设置目标芯片（目前支持 MC68HC08LD64 和 MC68HC08JB8），将新的程序代码（必须是 Motorola 的 S 记录格式的文件）写入芯片，修改芯片内存储的数据等等。

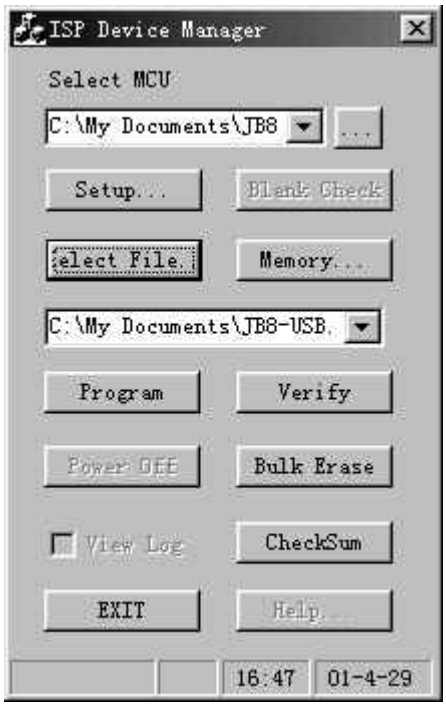


图 4.22 USB 在线编程用户程序界面

4.4.3 结论

这个项目主要是为了支持 Motorola 的芯片客户。项目完成后，所有的软件资料都作为芯片的附加软件包免费地提供给客户。项目的成果得到了 Motorola 和客户双方面的认可。

4.5 结论

以上的 4 个 USB 项目是我在两年内完成的，通过开发这些项目，我学会了：

- USB 设备和集线器（HUB）的硬件设计方法
- USB 设备端通用协议栈和 HID 子类协议栈的编写方法
- 主机端 HID 设备的用户层应用程序编程方法
- 主机端通用 USB 设备的内核层客户驱动程序和用户层应用程序的编写方法

这些技术合并在一起，就使得我对 USB 整个系统的结构有了清晰的认识。当然 USB 协议还有很多方方面面的细节技术是我所不了解的，但有了以上的这些基础，USB 的系统技术对我而言将不再是不可掌握的。

这些项目的立意都是直接以市场和用户的需要为基础的，都或多或少地产生了社会和经济效益。这一点也是比较让人满意的。

当然这些项目还有许多可以提高和完善的地方，例如 USB 安全钥，还有很多工作可以做。如果条件允许，我也会不断地努力去加强和完善它们的功能，弥补已有的不足。

第五章 其他工作

USB 技术不是凭空构筑出来的，而我对 USB 技术的学习和实践也不可能是只集中在 USB 协议周围的。我需要去学习硬件电路的通用设计方法，学习微控制器的编程和使用技术，学习 Windows 下的驱动程序和应用软件编程方法……种种技术，都是为了实现一个完整的系统而服务的。在学习和实践这些非 USB 技术的过程中，我也积累了一些心得和体会，希望在这一章中写出来与大家交流。

5.1 Motorola 微控制器的使用

从 MC68HC05JB4 开始，Motorola 的 8 位微控制器就与我结下了不解之缘。1999 年，Motorola 推出了 MC68HC908GP32，我为它设计制作了 MC68HC908GP32IDK 在线编程开发系统。而在 2000 年里，我则主要使用 MC68HC908JB8 和 MC68HC908LD64 开发了 USB 安全钥和 USB 在线编程设备。

在前面对 USB 技术和项目的叙述中，已经讲述了 MC68HC05JB4、MC68HC908JB8 和 MC68HC908LD64 的特点和开发方法。下面的篇幅将主要介绍 MC68HC908GP32 的特点和 MC68HC908GP32IDK 的设计思路。

5.1.1 在线编程技术和 MC68HC908GP32IDK

MC68HC908GP32 是 Motorola 在从 05 系列微控制器向 08 系列升级的过程中推出的一款硬件资源相当丰富的芯片，具有以下特点：

- 32K 字节片内 FLASH 存储器，支持在线编程
- PLL（锁相环电路），最大可以产生 8MHz 的内部总线时钟
- 增强型串行口通信口 SCI
- 串行外围接口 SPI
- 8 路 8 位 A/D 转换器
- 两个 16 位双通道定时器接口模块（TIM1 和 TIM2），每个通道可选择为输入捕获、输出捕获和 PWM，其时钟可分别选为内部总线的 1、2、4、8、16、32 和 64 的分频值
- 8 位键盘唤醒口

- 支持 C 语言
- 完全向上兼容 68HC05

利用片内 FLASH 支持在线编程的特点，我设计和制作了 MC68HC908GP32IDK 在线编程开发系统。

5.1.1.1 MC68HC908 系列微控制器的在线编程技术

FLASH（闪速存储器）是一种兼有紫外线擦除 EPROM 和电擦除 EEPROM 两者优点的新型非易失大容量存储器件，擦写速度快，集成密度和制造成本与普通 UVEPROM 相近。在微控制器内部集成 FLASH，并不是 Motorola 首创的技术，在 Microchip 的 PIC 系列、Atmel 的 AT89C51 系列等微控制器中早已引入了片内 FLASH 技术。但这类微控制器的片内 FLASH 较 ROM 或 EPROM 而言，可靠性和稳定性是最大的问题。Motorola 在 FLASH 技术已相当成熟的时候才推出集成片内 FLASH 的 8 位微控制器，在应用方便性和可靠性上均有不俗的表现，具体体现在：

- 单一电源电压供应：FLASH 在正常的只读情况下，只需要用户为其提供普通的工作电压（如+5V）。如果用户要对 FLASH 进行编程，需要同时提供高出正常工作电压的编程电压（如+9V）。Motorola 通过在片内集成电荷泵，只需要用户提供单一 5V 工作电压，便可以在片内产生出编程电压。这使得用户无需为 FLASH 的编程而在目标板上增加任何多余的硬件模块。
- 高可靠性：Motorola 片内 FLASH 的存储数据可以保持 10 年以上，可擦写次数也在 1 万次以上。
- 高的擦写速度：Motorola 片内 FLASH 的整体擦除时间可以控制在 5ms 以内，对单字节的编程时间也在 40μs 以内。

Motorola 片内 FLASH 的编程操作分为两种：擦除（Erase）和写入（Program）。擦除操作是将存储单元的内容由二进制的 0 变成 1，而写入操作恰好相反，是将存储单元的内容由二进制的 1 变成 0。擦除操作有整体擦除和页擦除两种具体操作方式。MC68HC908 系列微控制器提供了闪速存储控制寄存器（FLCR），FLASH 的写入和擦除操作都是通过设置 FLCR 中的控制位来完成的。

目前在 FLCR 中只有四位是有效的：

- HVEN — 高压允许位，用于将来自片内电荷泵的高压加到 FLASH 阵列上；
- MASS — 整体擦除控制位，用于选择擦除方式（=1，整体擦除 / =0，页擦除）；
- ERASE — 擦除控制位，用于选择擦除操作；
- PGM — 编程写入控制位，用于选择编程写入操作。

在对 FLASH 进行编程操作时，开发者需要注意两个单位：页（page）和行（row）。页和行的大小（字节数）随整个 FLASH 的大小变化而变化，但页的大小始终为行的两倍。例如 MC68HC908GP32 内含 32K 的 FLASH，每页的大小为 128 字节，每行的大小为 64 字节；而 MC68HC908JL3 片内 FLASH 仅有 4K，每页和每行的大小也分别变为 64 字节和 32 字节。对 FLASH 的擦除操作都是以页为基础的，而写入操作则是以行或页为基础的。

用户还可以选择对部分 FLASH 进行编程保护，保护区的首地址由 FLASH 块保护寄存器（FLBPR）设定，末地址则固定为$FFFF。受保护的 FLASH 单元在用户模式下将无法被编程擦除或写入。

对 FLASH 进行擦除或写入操作都需要遵循一定的时序和步骤。对于 MC68HC908 系列的微控制器，这些步骤都是一样的，但时序要求可能略有不同，用户应根据具体情况参照相应的芯片手册。

1. 页擦除操作

1) 置 ERASE 位为 1；
2) 读出 FLASH 块保护寄存器；
3) 向被擦除的 FLASH 页内任意地址写入任意值；
4) 延时 Tnvs；
5) 置 HVEN 位为 1；
6) 延时 Terase；
7) 清 ERASE 位为 0；
8) 延时 Tnvh；
9) 清 HVEN 位为 0；
10) 延时 Trcv 后，该 FLASH 页可以被正常读取。

2. 整体擦除操作

1) 置 ERASE 位和 MASS 位为 1；
2) 读出 FLASH 块保护寄存器；
3) 向 FLASH 存储区内任意地址写入任意值；
4) 延时 Tnvs；
5) 置 HVEN 位为 1；
6) 延时 Tmerase；
7) 清 ERASE 位为 0；
8) 延时 Tnvhl；
9) 清 HVEN 位为 0；
10) 延时 Trcv 后，整个 FLASH 存储区可以被正常读取。

3. 写入操作（在 MC68HC908 系列众多的微控制器中，有的以页为基础进行写入操作，如 MC68HC908GP32 等，有的以行为基础进行写入操作，如 MC68HC908JL3 和 MC68HC908JB8 等。用户在编程时需要根据实际选用的微控制器型号查阅相应的芯片手册。）

1) 置 PGM 位为 1；
2) 读出 FLASH 块保护寄存器；
3) 向页（或行）地址范围内（以 MC68HC908GP32 为例，即$XX00—$XX7F，或$XX80—$XXFF）任意 FLASH 单元写入任意值；
4) 延时 Tnvs；
5) 置 HVEN 位为 1；
6) 延时 Tpgs；
7) 向页内目标地址写入编程数据；
8) 延时 Tprog；
9) 重复 7)、8)，直至同一页内各字节编程完毕；
10) 清 PGM 位为 0；
11) 延时 Tnvh；
12) 清 HVEN 位为 0；
13) 延时 Trcv 以后，该 FLASH 页可以被正常读取。

下表是 MC68HC908GP32 的 FLASH 编程的各项参数，其他型号的微控制器参数请参阅相应的芯片手册。

表 5.1 MC68HC908GP32 的 FLASH 编程参数

参数	符号	最小值	最大值	单位
FLASH 行大小	—	64	64	字节
FLASH 页大小	—	128	128	字节
FLASH 读写周期	Fread	32K	8.4M	Hz
FLASH 页擦除时间	Terase	1	—	ms
FLASH 整体擦除时间	Tmerase	4	—	ms
FLASH 从设置 PGM 或 ERASE 位到 HVEN 建立时间	Tnvs	10	—	μs
FLASH 高电平维持时间（页擦除时）	Tnvh	5	—	μs
FLASH 高电平维持时间（整体擦除时）	Tnvhl	100	—	μs
FLASH 写入编程维持时间	Tpgs	5	—	ns
FLASH 写入编程时间	Tprog	30	40	μs
FLASH 返回只读状态时间	Trcv	1	—	μs
FLASH 累积高电平时间	Thv	—	25	Ms
FLASH 行擦除次数	—	10K	—	次数
FLASH 行写入次数	—	10K	—	次数
FLASH 数据保持时间	—	10	—	年

其中：

1. Terase 和 Tmerase 的最小时间也是擦除操作的最佳时间，如果擦除时间长于该最小值，FLASH 的寿命会受到影响。
2. Thv 指的是在下一次擦除操作执行以前，该 FLASH 行在写入编程操作中被加上高电平的累积时间。

MC68HC908 系列微控制器的片内 FLASH 可以在两种模式下进行在线编程：监控模式（Monitor Mode）和用户模式（User Mode）。在不同的模式下，对 FLASH 进行操作的时序和步骤都是一样的。

Motorola 推出的 MC68HC08 系列微控制器都可以工作在监控方式下。只要微控制器的复位向量（$FFFE-$FFFF）内容为“空”（$FFFF），或是在微控制器复位时在特定引脚上加上特定的电平，就可以使微控制器在复位后进入监控工作方式。在监控方式下，微控制器内部的监控 ROM 程序开始工作，并通过半双工串行口为主机（Host）的远程控制提供服务。主机程序可以利用监控 ROM 提供的少数

几个指令对微控制器内部地址进行读取、写入等基本操作，包括下载程序到 RAM 中并执行。在此基础上，主机可以通过主机程序或是下载到 RAM 中的程序完成对 FLASH 编程所需的一系列操作。

同时，在微控制器正常工作的过程中，程序也可以随时转入对 FLASH 的用户模式下的编程操作。此时只有对 FLASH 进行编程写入或擦除的程序是必须的，而不需要用户提供任何外部硬件条件（包括特定引脚电平和主机的存在）。如果需要，微控制器程序可以通过 SCI 串口从主机获得某些命令或数据，但开发者必须为此编写相应的串口通讯和命令服务程序。

两种模式各有优缺点。监控模式需要外部硬件支持，但不需要微控制器内部程序的存在，所以适合对新出产芯片进行编程写入，或是对芯片进行整体擦除或写入；用户模式可以在微控制器正常工作时进入，所以常用在程序运行过程中对部分 FLASH 单元进行修改，特别适合于目标系统的动态程序更新和运行数据存储。目前监控模式常被仿真器和编程器采用，而在实际的工程应用中，开发者往往只需要考虑和实现用户模式下的 FLASH 在线编程。

在用户模式下，开发者需要按上文描述的 FLASH 编程操作的步骤编写相应的 FLASH 编程操作子例程，然后在适当的时候由主流程调用。

开发者需要特别注意的是，这段完成编程步骤的子例程代码应该驻留在什么地方。在 Motorola 关于片内 FLASH 的某些宣传资料中，曾指出只要子例程代码与被编程的目标地址不在同一操作单元（页）中，对 FLASH 的任何操作就都是可行的。但在 Motorola 的许多技术文档中，却一再强调应将子例程在 RAM 中运行。实际上对于目前的 MC68HC908 系列微控制器而言，一旦 FLASH 控制寄存器（FLCR）的 HVEN 位为 1，整个 FLASH 都会被加上高于普通工作电平的编程电平，这时对 FLASH 内任何字节的读取都是不稳定的（包括对程序执行代码的读取），也就是说，微控制器此时执行的代码有可能是完全错误的。而一旦微控制器执行了错误的指令，系统的复位就是不可避免的了。我在 MC68HC908GP32 的使用过程中发现，如果直接调用 FLASH 中的编程子例程，在对 FLASH 进行页擦除时，操作是成功的，但会导致微控制器自动复位，而在对 FLASH 进行写入时，系统复位并导致操作完全失败。所以 FLASH 编程子例程必须存在于 RAM 或监控 ROM 中。如果用户使用的是自行开发的 FLASH 编程子例程，可以在需要对 FLASH 进行编程前将子例程复制到 RAM 中去，然后跳转到 RAM 中执行。

对于 RAM 区较少的微控制器，如 MC68HC908JL3，开发者必须利用监控 ROM 中固化的程序。一般来说，在这类微控制器的监控 ROM 中，都已经固化了标准的 FLASH 编程操作子例程。Motorola 的相关技术文档对这些子例程的接口和使用方法都有详细的介绍。

5.1.1.2 MC68HC908GP32IDK 在线编程开发系统

MC68HC908GP32IDK(In-system-program Development Kit)在线编程开发系统利用了单片机内含 32K FLASH 的特点，为开发者提供了一套简便的、友好的单片机在线编程开发系统。

一方面，用户可以简便地利用它来对芯片的各个功能和模块进行一些实验和测试。系统提供了良好的软件和硬件模块接口，用户可以直观地从目标评估母板上观察程序运行的结果，也可以从 PC 机查询单片机当前的状态。此功能最适于高校的教学实践，是高校单片机教学的理想实验系统。同时此功能也可以用作芯片功能的演示和测试，起到 MC68HC908GP32 评估板的作用。

另一方面，开发者可以利用我们提供的固化在单片机中的监控程序和外围硬件功能模块，开发自己的系统。本系统在调试过程中能实现 100%的在线实时仿真；用户可以通过 PC 机方便地查询芯片状态、修改寄存器内容和用户程序。同时目标评估母板为 MC68HC908GP32 设计的各个外围硬件功能模块，可以直接嵌入到用户的各种实际系统之中，成为开发者降低开发难度、缩短开发时间的有利工具。

它的特点是：

- 基于 MC68HC908GP32
- 内含不大于 8K 的监控程序
- 监控程序提供给用户各种基本的开发和调试功能：程序的下载和运行、汇编和反汇编、断点设置、内存显示修改等等
- 利用 MC68HC908GP32 的在线编程技术，实现了对用户程序的在线写入和对 FLASH 存储内容的随时修改
- 用户可以实现对自编程序的 100%在线实时仿真与监测
- 目标评估母板提供了大量外围功能模块，可辅助完成对并行 I/O 口、键

盘和外部中断、A/D、SPI、SCI 和时钟等模块的实验和测试，也可直接嵌入到用户应用之中

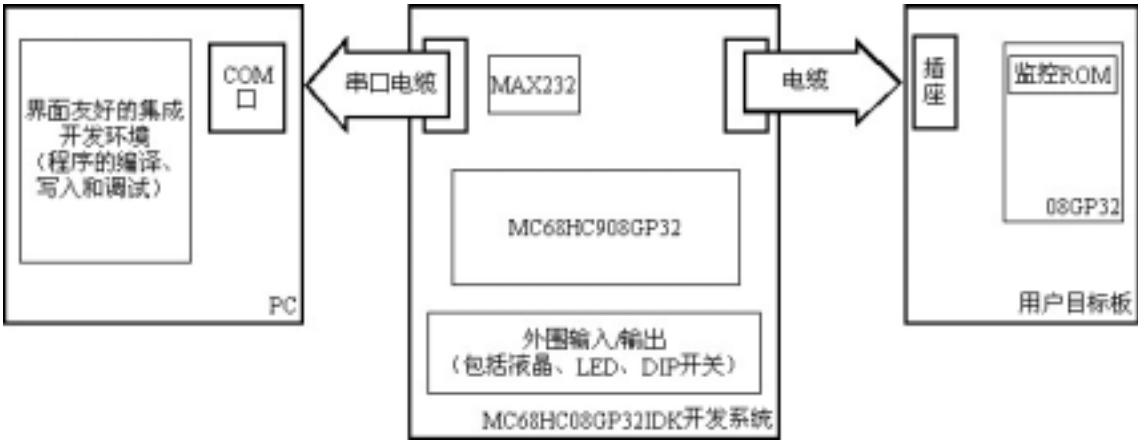


图 5.1　MC68HC908GP32IDK 原理框图

MC68HC908GP32IDK 硬件系统由核心子板与目标评估母板组成，两对 32 芯欧式连接插头座实现了电源和信号线二者的连接。

核心子板(图 5.2)提供了使 MC68HC908GP32 正常工作的最小系统，包括 GP32 及其复位电路、时钟电路、RS232 串口电路等几个部分。为了满足不同用户的需要，核心子板提供了 3 个跳线选择设置：复位电路选择、Txd 选择和 Rxd 选择。

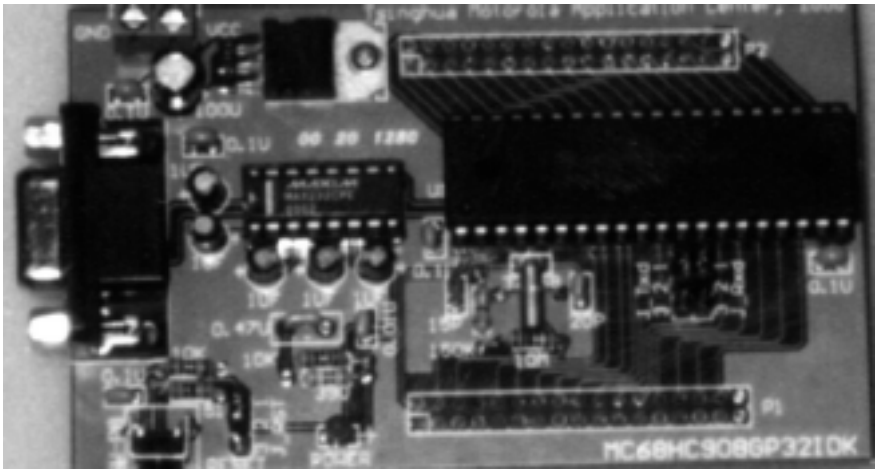


图 5.2　核心子板

目标评估母板（图 5.3）上,除 MC68HC908GP32IDK 模块外,还提供了多个外围硬件功能模块，包括：

- 8 位数字量的输入输出
- A/D 模拟电平输入
- 液晶显示
- 键盘和外中断信号输入
- SPI 和 SCI 输入输出

限于篇幅，各模块功能原理在这里不再赘述。用户在了解了各模块基本的电路原理以后，使用导线在模块之间或模块内部进行连接，便可以完成各种试验。母板上提供了许多单孔插座以方便用户完成这些接线。

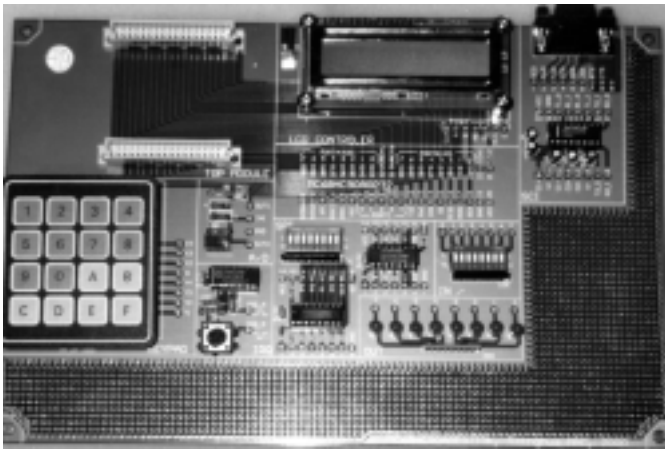


图 5.3　目标评估母板

有关 MC68HC908GP32IDK 的详细说明，请参看《MC68HC908GP32IDK 在线编程开发系统用户手册》。

5.1.1.3 结论

这套开发工具推出以后，得到了 Motorola 公司、各大高校和广大开发者的欢迎和认可，Motorola 公司也因此向我所在的 Motorola 单片机开发中心一次性购买了 500 套，并转送给十几所高校用于 Motorola 微控制器的教学和实验。

参考文献

1. 邵贝贝，刘慧银，等，微控制器原理与开发技术，清华大学出版社，1997
2. 陈章龙、韩光，MOTOROLA 单片机接口技术手册，复旦大学出版社，1993
3. Jeffrey Richter 著，郑全战，阿夏，译，Windows 95/Windows NT 3.5 高级编程技术，清华大学出版社，1996
4. Art Baker 著，科欣翻译组，译，Windows NT 设备驱动程序设计指南，机械工业出版社，1997
5. Chris Cant 著，孙义，马莉波，国雪飞，等译，Windows WDM 设备驱动程序开发指南，机械工业出版社，2000
6. MC68HC（7）05JB4 SPECIFICATION（GENERAL RELEASE），MOTOROLA INC.，1998
7. Universal Serial Bus Specification, Revision 1.0 ， USB Implementers’ Forum, January 15, 1996
8. Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.0 Final. USB Implementers’ Forum，1997
9. Universal Serial Bus HID Usage Tables, Release Candidate 1.0. USB Implementers’ Forum，October 30, 1997
10. The Windows NT Device Driver Book: A Guide for Programmers. Art Baker.
11. Windows 98 DDK Documentation. Microsoft Corporation, 1998.
12. Walter Oney 著，Programming the Microsoft Windows Driver Model，Microsoft Press，1999

致谢

首先感谢系里和实验室老师对我无微不至的关怀和指导，他们是邵贝贝教授、刘慧银副教授和吴凤茹老师。还有感谢实验室的同学在我做项目的过程中给予我的帮助，他们是王若鹏（已毕业）、陈勇（已毕业）、王暹辉（已毕业）、崔永刚（已毕业）、许庆丰、郑杨斌、龚光华、薛涛和马伟等。

在这里还要感谢 EDLAB 工作组和刘丁、黄锋等在技术上给过我指导的同行，没有他们的帮助和指导，我也不可能取得这些成果。

个人简历

个人资料：

姓名: 王云飞 **性别:** 男

出生日期: 1977 年 6 月 13 日 **婚姻状况:** 未婚

国籍: 中华人民共和国 **健康状况:** 良好

E-mail: wangyf_tshua@sina.com

电话: 86-10-62781445(O), 86-10-62775634(H)

呼机: 86-191-1287212

手机: 86-13910127712

地址: 北京市清华大学工程物理系 210 室

邮政编码: 100084

身份证号: 110108770613893

履历：

清华大学 1999 年 9 月~现在

学位: 核技术及应用专业工学硕士

论文: "USB 系统研究"

导师: 邵贝贝教授

清华大学 1995 年 9 月~1999 年 6 月

学位: 工程物理专业学士

论文: "USB 手写识别输入系统"

导师: 邵贝贝教授

曾获荣誉

- 校“光华”一等奖学金，2000
- 校“挑战杯”一等奖（USB 通用设备开发平台）, 2000
- 校学生实验室建设贡献二等奖，1999
- 校优良毕业生称号, 1999
- 校“挑战杯”二等奖（USB 手写输入系统）, 1999
- 通用电气奖学金, 1998
- 校级优秀学生称号, 1997
- 优秀学生一等奖学金, 1997
- 优秀学生一等奖学金, 1996
- 新生奖学金, 1995

发表论文

1. 《USB 电源管理》,发表于《电子产品世界》（国家一级科技期刊）1999 年第 1 期
2. 《MC68HC908 系列单片机片内 FLASH 在线编程》，发表于《电子产品世界》（国家一级科技期刊）1999 年第 12 期
3. 《USB 在数据采集系统中的应用》，发表于《电子技术应用》（中文核心期刊/科技论文统计源期刊）2000 年第 4 期（第一作者：刘丁）
4. 《用 MC68HC05JB4 开发 USB 外设》，发表于《电子技术应用》（中文核心期刊/科技论文统计源期刊）2000 年第 5 期
5. 《USB 安全钥在电子商务中的应用》，发表于《电子产品世界》（国家一级科技期刊）2001 年第 1 期（第一作者：邵贝贝）