# 第一章 前 言

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 为首的七家公司于 1994 年推出了 USB（Universal Serial Bus，通用串行总线）总线协议，专用于低、中速的计算机外设。目前，USB 端口已成为了微机主板的标准端口；而在不久的将来，所有的微机外设，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，都将通过 USB 与主机相连。这种连接较以往普通并口和串口的连接而言，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。

作为一个硬件厂商或是开发者，最关心的便是如何去开发 USB 外设。一般的 USB 设备都使用一片微控制器作为其核心部件。通过微控制器强大的控制和运算功能，开发者可以很容易地实现 USB 设备的智能化。

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 68HC05BD9A，用于鼠标的 68HC05JB2，以及用于键盘的 68HC08KL8 和 68HC08KH12 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。68HC05JB4 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。

国外在近两年已出现了不少的 USB 外设，但目前在国内市场上我们仅发现了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。国内近年来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。

此 USB 手写系统，采用汉王公司的传感器获得笔画信息，传给 68HC05JB4，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。

# 第二章　USB 总线协议

USB（Universal Serial Bus）总线协议是以Intel为主，并有Compaq，Microsoft，IBM，DEC，Northern Telecom以及日本NEC等共七家公司共同制定的串行接口标准，1994年11月制定了第一个草案，1996年2月公布了USB规范版本1.0。USB可把多达127个外设同时联到你的系统上，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1999年2月发布的USB规范版本2.0草案中，已建议将12Mbps的带宽提升到120-240Mbps。）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。

以下简单介绍USB总线的结构、原理，以使读者对USB有大致的了解，如果需要了解其协议细节，请查阅USB总线规范，这可以从www.usb.org下载。

## 2.1　总线拓扑结构

USB总线的物理连接是一种分层的菊花链结构，集线器(hub)是每个星形结构的中心。PC机就是主机和根Hub，用户可以将外设或附加的Hub与之相连，这些附加的Hub可以连接另外的外设以及下层Hub。USB支持最多5个Hub层以及127个外设。图2.1描述了USB的物理拓扑结构，从中可以看出每一段的连接都是点对点的。

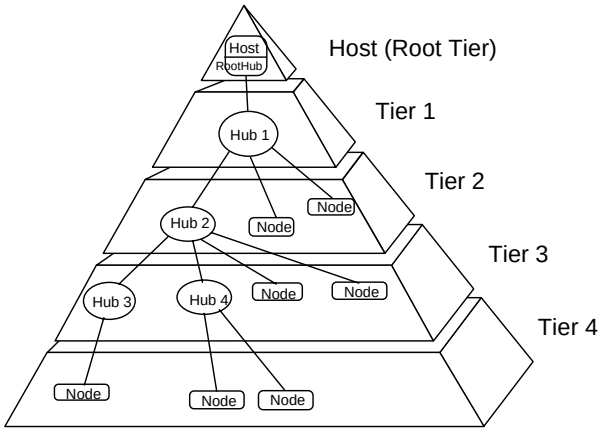


图2.1　USB总线拓扑

## 2.2 USB的物理层

USB的物理接口包括电气特性和机械特性。

USB通过一个四线电缆来传输信号与电源，如图2.2所示。

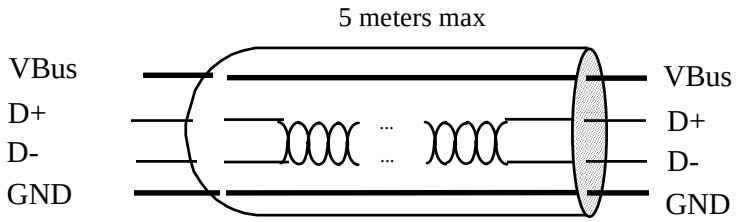


图2.2　USB电缆定义

其中D+和D-是一对差模的信号线，而VBus和GND则提供了＋5V的电源，它可以给一些设备(包括Hub)供电，当然要有一定的条件限制。

USB提供了两种数据传输率：一种是12Mb的高速(full speed)模式，另一种是1.5Mb的低速模式。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要求不高的设备的成本，比如鼠标、键盘等等。

USB信号线在高速模式下必须使用带有屏蔽的双绞线，而且最长不能超过5m；而在低速模式时中可以使用不带屏蔽或不是双绞的线，但最长不能超过3m。这主要是由于信号衰减的限制。为了提供信号电压保证，以及与终端负载相匹配，在电缆的每一端都使用了不平衡的终端负载。这种终端负载也保证了能够检测外设与端口的连接或分离，并且可以区分高速与低速设备。

所有的设备都有上行的接口。上行和下行的接头是不能互换的，这保证了不会有非法的连接出现。插头与插座有两个系列，分别为A和B，系列A用于基本固定的外围设备，而系列B用于经常拔插的设备，这两个系列是不能互换的。

## 2.3 USB 设备

USB设备包括Hub和功能设备，而功能设备又可以细分为定位设备、字符设备等等。为了进一步叙述，我们给出端点(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以作为设备的控制管道。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。

一个USB设备可以分为三个层（图2.3）。最底层是总线接口，用来发送与接收包。中间层处理总线接口与不同的端点之间的数据流通。一个端点是数据最终的使用者或提供者，它可以看作数据的源或接收端。最上层就是USB设备所提供的功能，比如鼠标或键盘等。

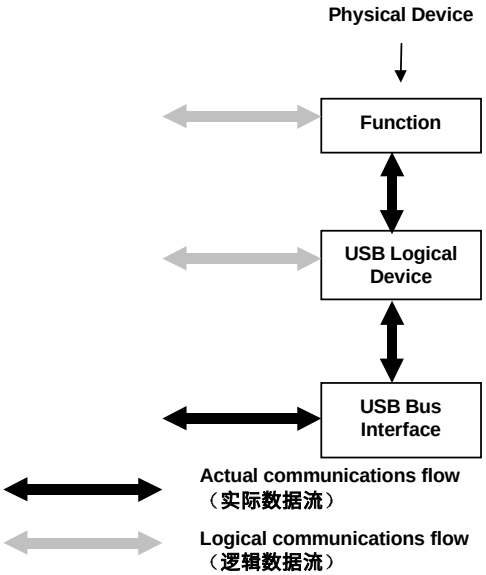


图2.3 USB设备结构层次

### 2.3.1 Hub

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。Hub可以检测出每一个下行端口的状态，并且可以给下端的设备提供电源。图2.4是一个典型的Hub。

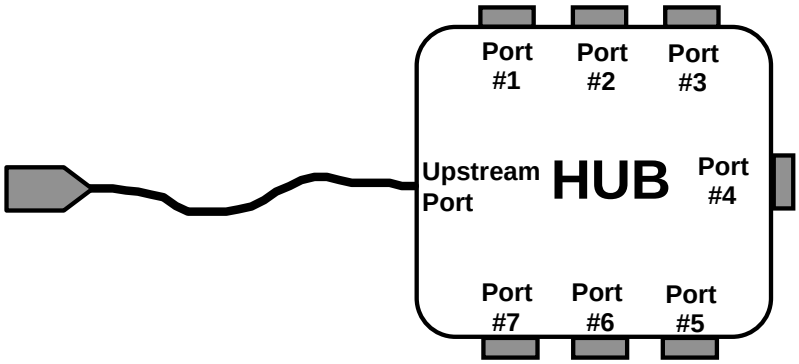


图2.4　典型的USB集线器（Hub）

### 2.3.2 即插即用

USB设备可以即插即用，但在可以使用之前，必须对设备进行配置。一旦设备连接到某一个USB的节点上，USB就会产生一系列的操作，来完成对设备的配置，这种操作被称为总线枚举过程：

1.设备所连接的Hub检测出端口上有设备连接，通过状态变化管道向主机报告；

2.主机通过询问Hub以获取确切的信息；

3.主机这时知道设备连接到哪个端口上，于是向这个端口发出复位命令；

4.Hub发出的复位信号结束后，端口被打开，Hub向设备提供100mA的电源，这时设备上电，所有的寄存器复位，并且以缺省地址0以及端点0响应命令；

5.主机通过缺省地址与端点0进行通讯，赋予设备一个独一的地址，并且读取设备的配置信息；

6.最后主机对设备进行配置,该设备就可以使用了。

当该设备被移走时，Hub依然要报告主机，并且关闭端口。一旦主机接到设备移走的报告，就会改写当前结构信息。

### 2.3.3 设备的电源

USB设备的电源可以由USB总线供给，也可以自备电源。一个USB设备可以具有这两种供电方式，但同一时刻只能由一种方式供电。这两种供电方式是可以切换的。

### 2.3.4 设备的挂起

为了节电，当设备在指定的时间内没有总线传输，USB设备自动进入挂起状态。如果设备所接的Hub的端口被禁止了，设备也将进入挂起状态(称之为选择挂起)。当然主机也可以进入挂起状态。

USB设备当总线活动时，就会离开挂起状态。一个设备也可以通过电信号来远程唤醒进入挂起状态的主机。这个能力是可选的，如果一个设备具有这个能力，主机有能力禁止或允许使用这种能力。

## 2.4 USB主机

USB主机在USB系统中处于中心地位，并且对USB及其连接的设备有着特殊的责任。主机控制着所有对USB的访问，一个外设只有主机允许才有权力访问总线。主机同时也监测着USB的结构。

USB主机包括三层（如图2.5）：设备驱动程序，USB系统软件，USB主控制器(主机的总线接口)。另外，还有两个软件接口：USB驱动（USBD）接口，主机控制驱动(HCD)接口。

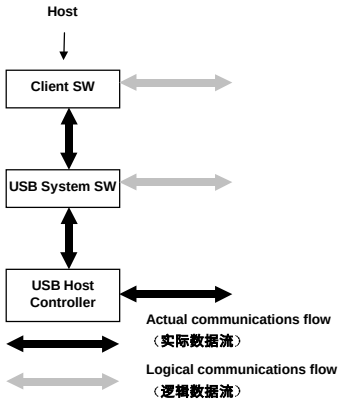


图2.5　USB主机结构层次

## 2.5 USB 数据流

图2.6描述了USB数据传输的过程。

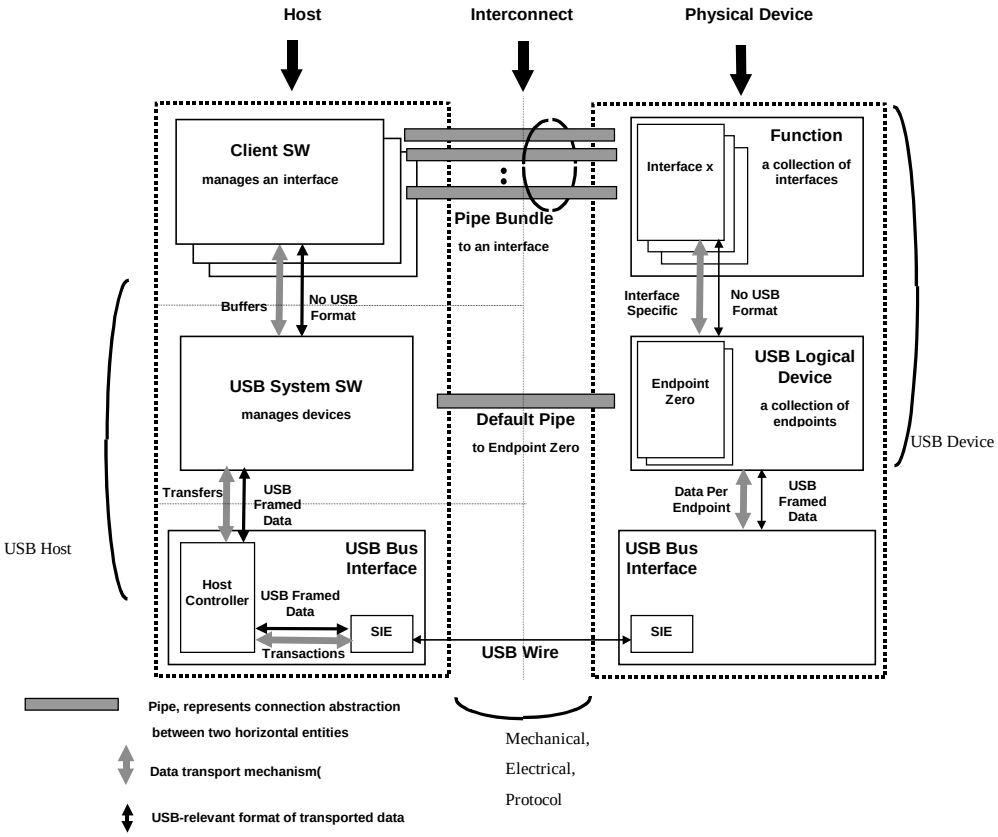


图2.6　USB数据流

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。

**帧：**USB总线将1ms定义为一帧，每帧以一个SOF包为起始，在这1ms里USB进行一系列的总线操作。引入帧的概念主要是为了支持与时间有关的总线操作。

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:

**控制传输(Control Transfer）**

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。

**同步传输(Isochronous Transfer)**

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%。

**中断传输(Interrupt Transfer)**

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。

**批传输(Bulk Transfer)**

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。

图 2.7 描述了输入输出请求（IRP）、传输（transfer）与操作（transaction）之间的关系。数据传输的具体格式详见 2.6.3。

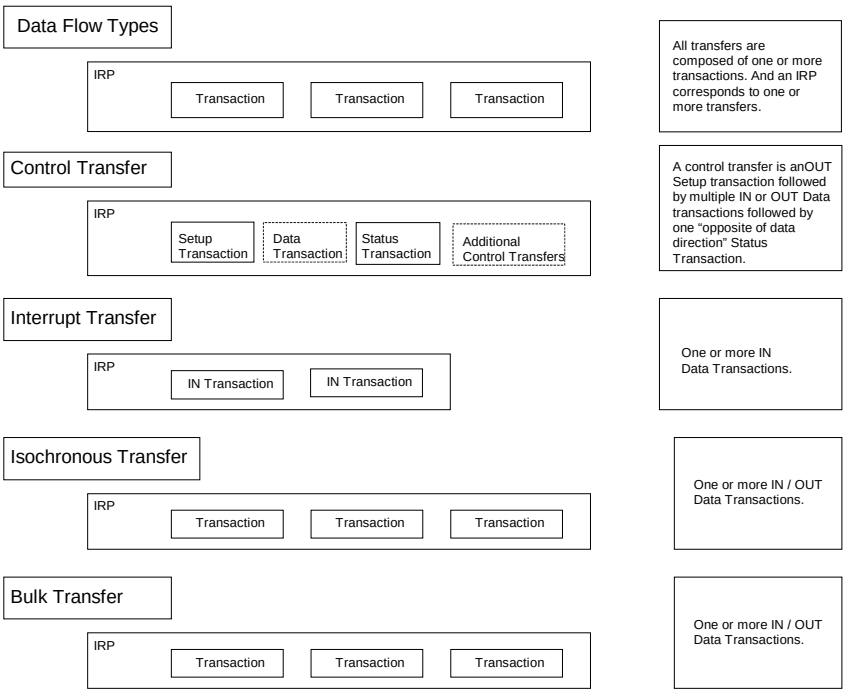


图 2.7 USB 数据传输

# 2.6 USB总线协议

所有总线操作都可以归结为三种包的传输。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)。然后在令牌中指定的数据发送者发出一个数据包或者指出它没有数据可以传输。而数据的目的地一般要以一个确认包(Handshake Packet)作出响应以表明传输是否成功。

## 2.6.1 域的类型：

☆**同步域**(SYNC field)：所有的包都起始于SYNC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，SYNC域将被省去。

☆**标识域**(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，外设将不作出响应。

☆**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独一的地址。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XFF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0一1023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。

### 2.6.2 包的类型：

☆**令牌包**(Token Packed)：

其中包括：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。

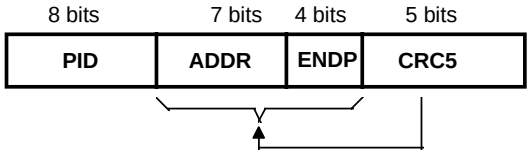


图2.8 IN、OUT、SETUP数据格式

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。

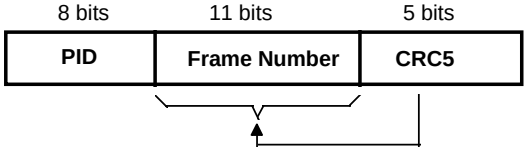


图2.9 SOF数据格式

☆**数据包**：包括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）。

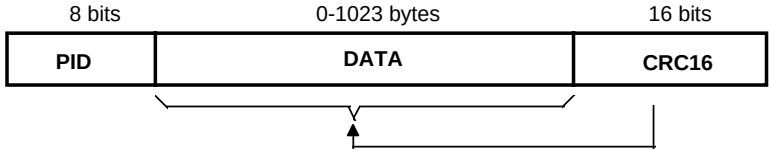


图2.10 DATA数据格式

☆**应答包**(Handshake Packet)：仅包含一个PID域（图2.11）。Handshake用来报告数据传输的状态。只有支持流控制的传输类型（控制、中断和批传输）才能返回Handshake。

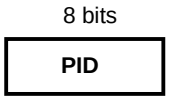


图2.11 PID数据格式

Handshake包有三种类型：

(1)确认包ACK：表明数据接收成功。

(2)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。

(3)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。

☆**特殊包**（Special）:PID名称为PRE（preamble）,用于低速操作的情形（详见6.5）。

### 2.6.3 总线操作的格式

**批操作（bulk transaction）:**

批操作包括令牌、数据、应答三个阶段，如图 2.12 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。

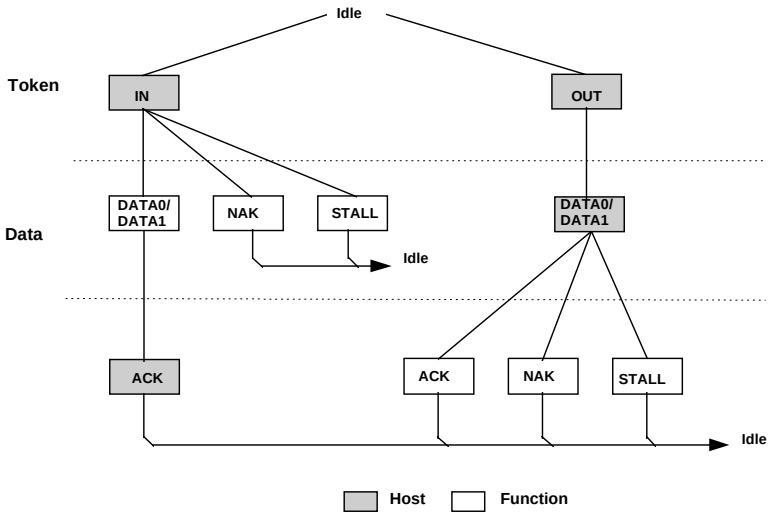


图 2.12 批操作流程

图 2.13 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 2.6.3）。

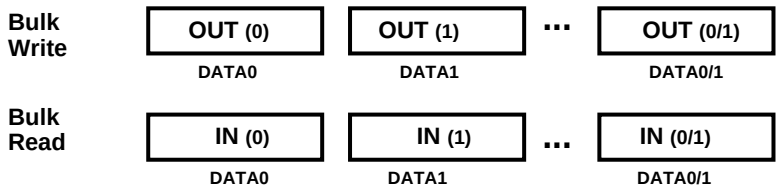


图 2.13 批操作读写过程

**控制操作(control transaction):**

控制操作（control transfer）主要包括两个操作阶段（transaction stage）:设置和状态。图 2.14 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。

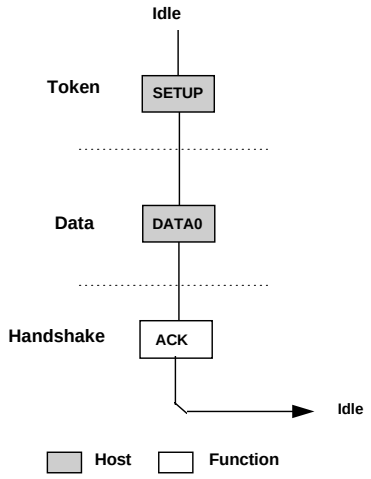


图 2.14 控制操作流程

下面是控制操作的详细描述（图 2.15），其中我们要注意数据包 PID 的使用。

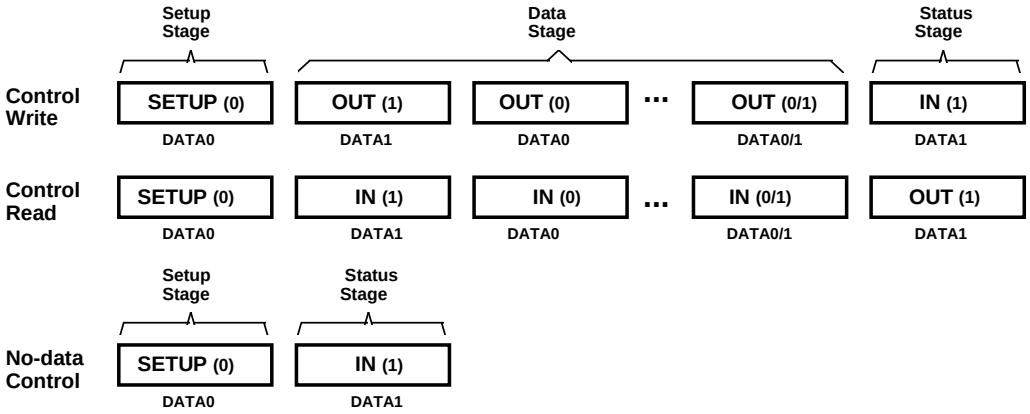


图 2.15 控制操作读写过程

**中断操作(interrupt transaction):**

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 2.16）。

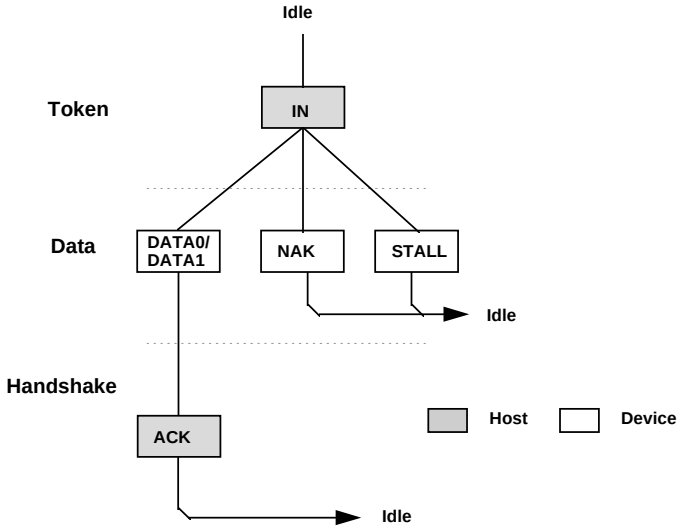


图 2.16 中断操作流程

**同步操作(isochronous transaction):**

同步操作不同于其他类型，只包含两个阶段：令牌和数据（图 2.17）。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。

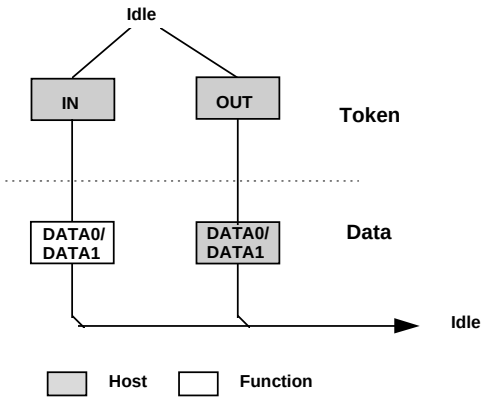


图 2.17 同步操作流程

### 2.6.4 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2.18，2.19，2.20说明了数据触发同步的基本原理。

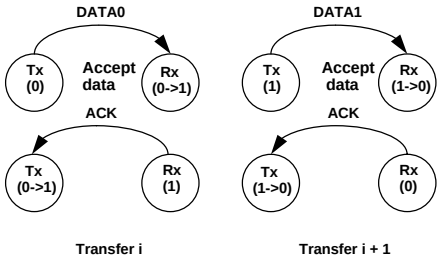


图2.18 数据触发与同步（一）

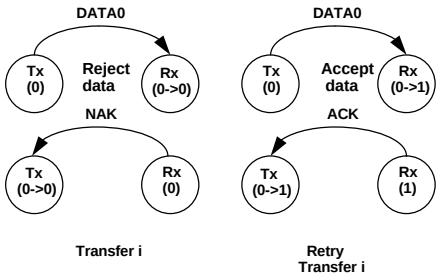


图2.19 数据触发与同步（二）

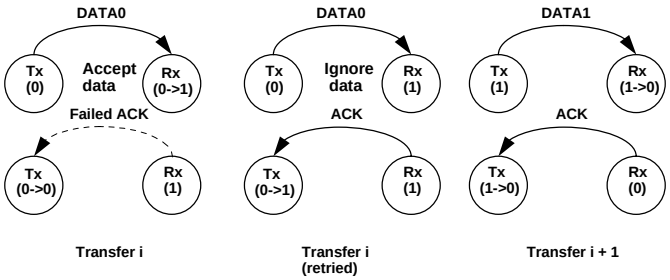


图2.20 数据触发与同步（三）

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。

### 2.6.5 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2.21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。

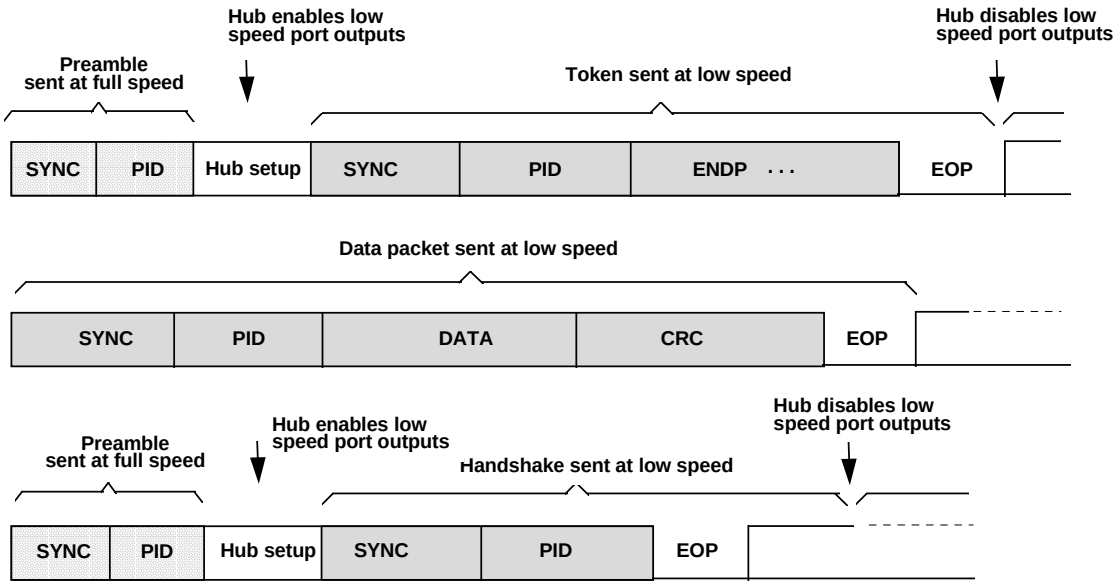


图2.21 低速方式的输入操作

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。

### 2.6.6 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。

## 2.7 结束语

以上我们介绍了 USB 的基本结构和原理，只涉及了 USB 规范 1.0 版的一些章节，如果想深入的了解 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，除此之外 USB 将设备分为不同的类型，每个设备类型都定义了类似功能设备的共同行为和协议。例如 HID（Human Interface Device）人机接口设备主要指用于人控制计算机系统操作的器件，而电源设备（Power Device）则被定位为 HID 的子系统之一。对设备进行分类是为了抹除不同硬件厂商之间的差异，以便于主机(PC)对设备进行方便、统一的管理。相同类型的设备都由一组标准定义的功能模块组成。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。对于每一类设备，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。本项目设计的手写板属于 HID 设备，在软件设计方面有许多 HID 设备特有的要求和实现方法，在本文后面的章节中还会详细叙述。

USB 是 1996 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。（另外，还有一种高速的串行接口 IEEE1394，传输率可达 100Mb,200Mb 乃至 400Mb，主要用于高速设备的数据传输，如硬盘、光驱等。）

USB 从推出到现在已经快两年了，到目前虽然还没有得到广泛的应用，但已经有许多软、硬件产品支持 USB 了。自从厂家把 USB 所需的控制芯片加入到外围设备的 ASIC（专用 IC）中，PC 对 USB 的支持只需要增加成本不到 1 美元的插座。这大大刺激了 USB 的发展。Microsoft 推出的 PC98 和 PC99 系统已宣布将 USB 和 HID 作为其支持的工业标准之一。可以预见，USB 将在今后的几年里得以发展，它将成为标准的设备接口。

# 第三章　硬件设计

## 3.1　整体硬件描述

本项目以“汉王”手写识别系统为基础，通过自制的实验板收集输入板的数据，经过芯片 MC68HC(7)05JB4 的处理，再通过 USB 总线传入 PC 主机，最后由驱动程序发给“汉王”汉字识别程序，完成整个数据的传递过程（图 3.1）。

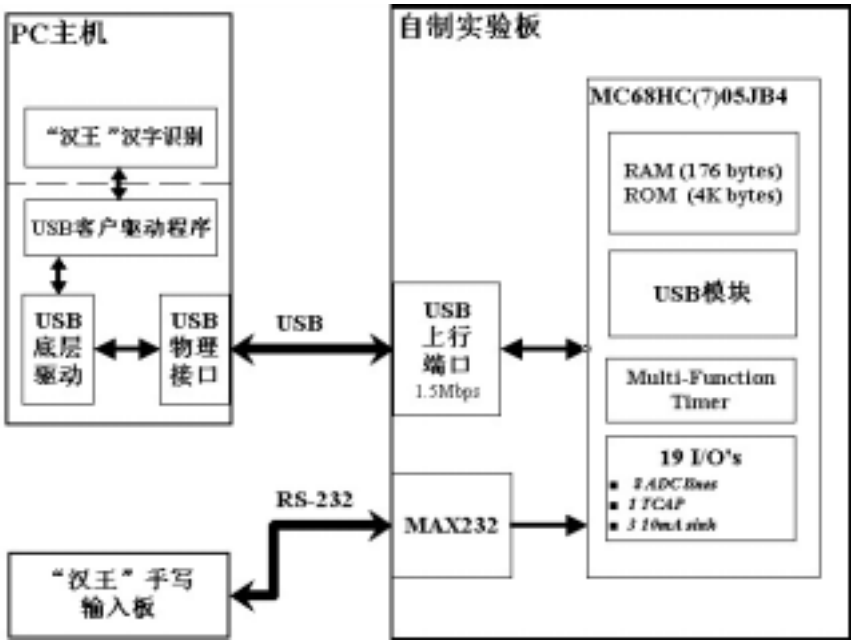


图 3.1　系统结构框图

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，以及单片机汇编程序和 WINDOWS 98 下的驱动程序编写。在硬件设计上，单片机的选择以及如何实现向“汉王”手写输入板的供电和输入板数据的接收是几个最主要的部分；而在程序实现上，编写手写输入板数据的接收程序和 WINDOWS98 下的 USB 客户驱动程序，

则是设计的基本任务。本章将详细讲述硬件部分的设计和实现，软件描述将由下一章来完成。

## 3.2 各部分硬件电路说明

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，称为 SCI（异步串行通讯接口）接口电路；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。

### 3.2.1 SCI 接口电路

SCI 接口电路使用的芯片是 MAX232，它可将微控制器使用的+5V 电压传换为+12V，供“汉王”手写板使用，同时将手写板的发送数据传给微控制器。（图 3.2）

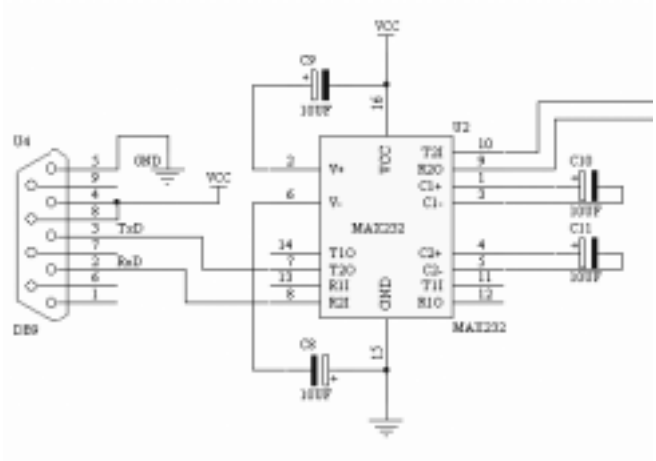


图 3.2　SCI 接口电路

### 3.2.2 USB 接口电路

MC68HC05JB4 提供了两个端口分别与电缆的 D+、D-相连；同时提供了一个 3.3V 的参考电压，与 D-相连。典型连接如图 3.3。其中 1.5K Ω 的电阻要求较高， 阻

值范围必须是 1.5KΩ±5%。一般的 USB 设备（包括本项目）由于电缆长度有限，类似电缆负载之类的工作都不需要开发者去考虑。

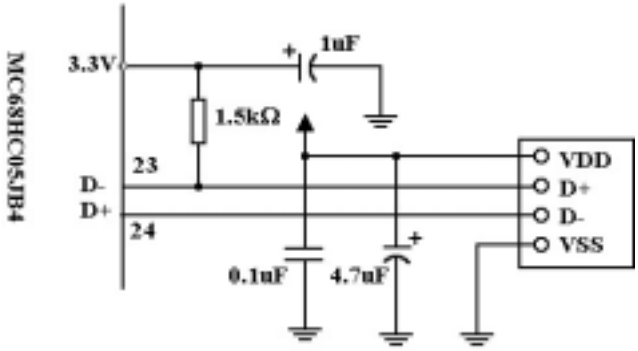


图 3.3　USB 接口电路

### 3.2.3　晶体振荡电路

晶体振荡电路在微控制器系统中非常重要，它决定了整个微控制器系统能否稳定工作。为了产生 1.5M 的 USB 总线速率，系统使用 6MHz 的晶振。电路连接如图 3.4，其中 C2、C3 和 R2 的接入都是为了晶振更容易起振。一般要求 C2 和 C3 的容值在 20PF 上下，而 R2 可用 2M 或是 10M。

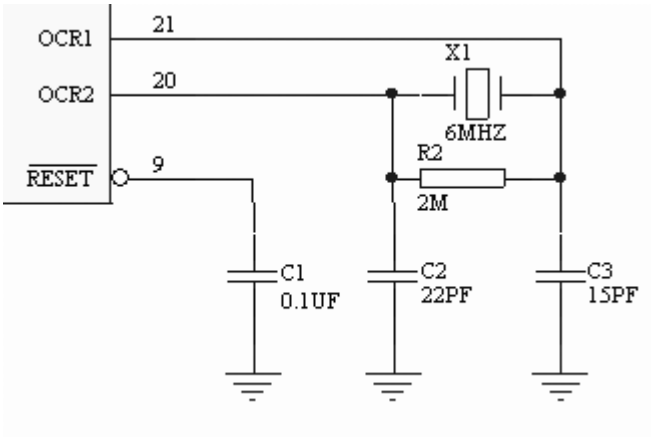


图 3.4　晶体振荡电路

## 3.3　整体电路原理图

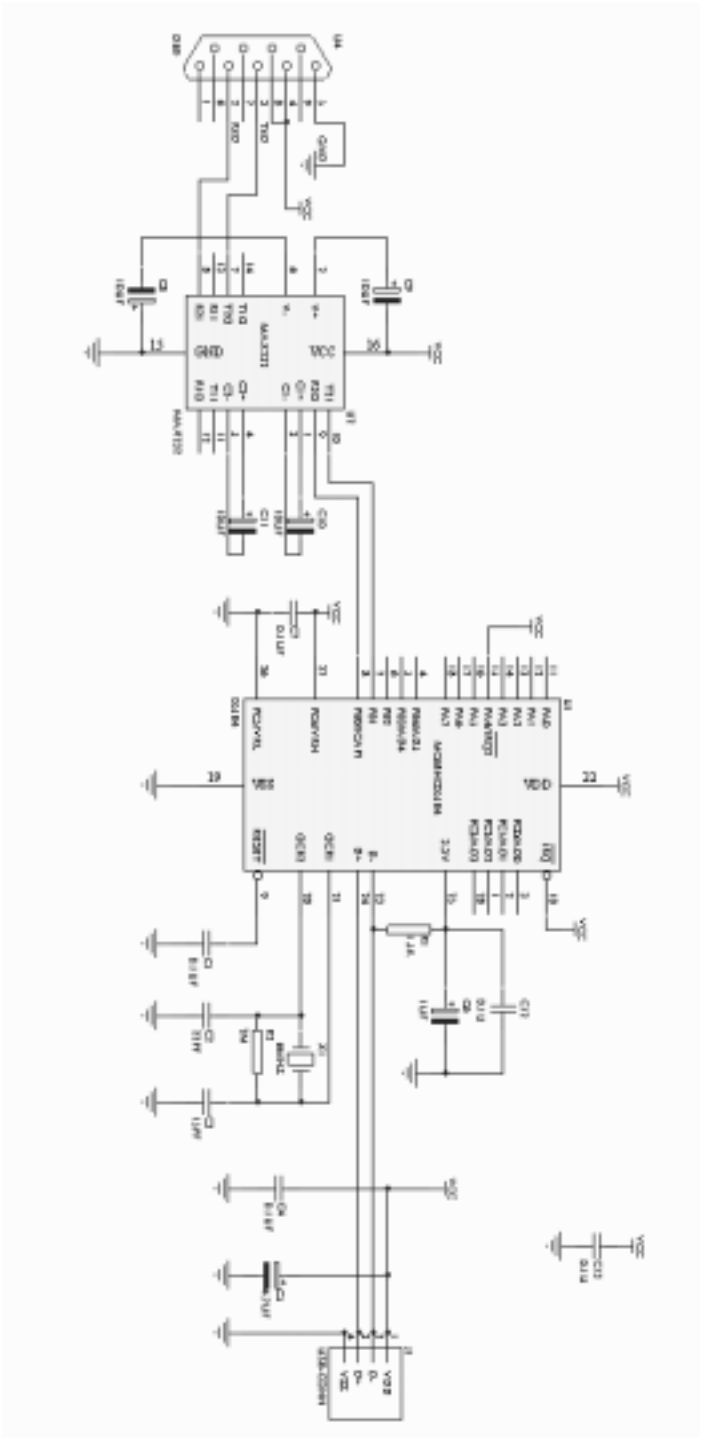

图 3.5　整体电路原理图

## 3.4 印制电路板设计实现

在设计印制电路板是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。

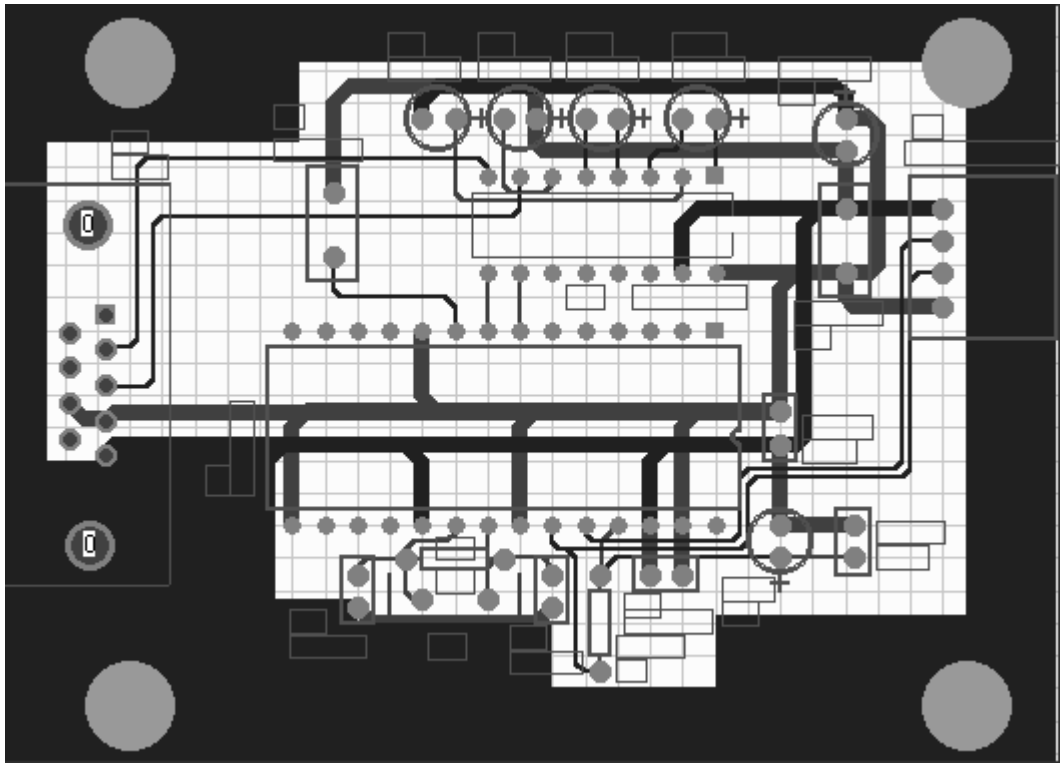

图 3.6 印制电路板元件分布与布线图

# 第四章　软件设计

本项目的软件设计是最困难的部分，主要包括微控制器 MC68HC05JB4 的汇编程序设计和 WINDOWS 98 下的 HID 设备驱动程序编写。其中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；而 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。

## 4.1　微处理器汇编程序设计

MC68HC05JB4 中的 USB 模块提供了 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。用户可以近似地把端点 0 看作是设备的控制和状态寄存器，而端点 1 和端点 2 则是设备的两个数据缓冲区。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。

由于 USB 协议和 HID 子协议比较复杂，所以这部分软件编写的工作量很大，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。

### 4.1.1 USB 中断服务例程

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，然后转入响应的处理例程。其中以三个令牌传输的响应最为复杂。USB 和 HID 协议都规定了许多主机对设备的命令请求，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来，再由命令处理器模块去完成相应的工作。

### 4.1.2 命令处理器

命令处理器需要处理的命令主要有两类：USB 设备的一般命令和 HID 设备的特有命令。程序实现过程中使用了不同的子程序以响应不同的命令请求，列表如

下：

| | 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 |

其中部分命令号在 USB 和 HID 的协议中尚未定义（以 reserved 表示），程序中调用 PST_STALL 去响应。例程 PST_STALL 只向主机返回一个 STALL 信号，此信号在主机方可以被解释为设备忙或是设备还未准备好等等。这样即使 USB 和 HID 协议在以后添加了新的命令，程序也能照常工作，提供了良好的兼容性。

### 4.1.3 手写板输入模块

由于 MC68HC05JB4 并没有 SCI 模块，为了正确接收“汉王”手写板发送的数据，需要用软件模拟实现 SCI 的接收功能。原理上主要是利用 MC68HC05JB4 内含的 16 位时钟的输入捕捉（ICAP）和输出比较（OCMP）功能，具体实现步骤如下（图 4.1）：

1．允许时钟的输入捕捉功能，设置为对下降沿敏感（触发）；
2．输入捕捉被触发，即标志起始位的开始，此时禁止输入捕捉功能，允许输出比较，并设置比较时间为 3/2 脉宽（使下次中断发生时正处于接收数据最低位的脉冲中央）；
3．在第一次输出比较中断发生时，对输入捕捉引脚电平进行三次采样（“0”或“1”），累加后取结果的第二位作为最终的采样结果（即接收数据的第 1 位），这样做可以减少波形毛刺对采样结果的影响，从而提高数据接收的正确率。同时设定下一次输出比较中断发生的时间，约为一位脉宽左右。
4．输出比较中断发生，按步骤 3 的方法对输入捕捉引脚电平进行采样，并设置接收数据的相应数据位。同时设定下一次输出比较中断发生的时间，约为一位脉宽左右。
5．重复步骤 4，直至接收数据的所有数据位都已接收完毕。

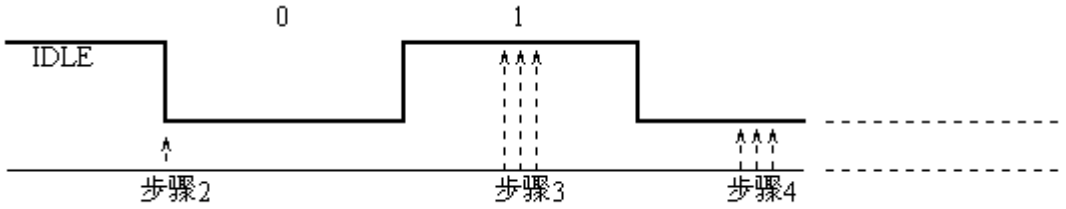


图 4.1　SCI 数据读入步骤

每次 SCI 数据接收后都存入缓冲区，待一个完整的数据包（共 5 字节）接收完毕，就设置标志位，通知报告处理器数据已准备好。如果 USB 通信陷入停顿，缓冲区有可能被充满，这时 SCI 的数据接收将被禁止。

### 4.1.4 报告处理器

报告处理器负责生成标准的 USB 报告，既包括“汉王”手写板发送的数据，也包括命令处理器相应主机命令时发送给主机的各种状态报告。它的实现较为简单。

### 4.1.5 设备的挂起与唤醒

除了上述各大模块，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。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。

但设备的挂起对设备的工作往往有不利的影响。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 280mA，这实际上是要求 MC68HC05JB4 进入 STOP 模式。但 MC68HC05JB4 在 STOP 模式下时钟被禁止，也就是说此时无法接收“汉王”手写板的数据，这显然不能满足项目设计的要求。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，既可以由主机发送唤醒或复位信号，也可以由设备自行远程唤醒，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 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，就可以自动恢复到正常的状态。本系统使用的是前一种方法，而后一种方法在 MOTOROLA 用 MC68HC05JB2 制作的鼠标里被具体实现过。

### 4.1.6 微控制器整体软件流程

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 4.2）

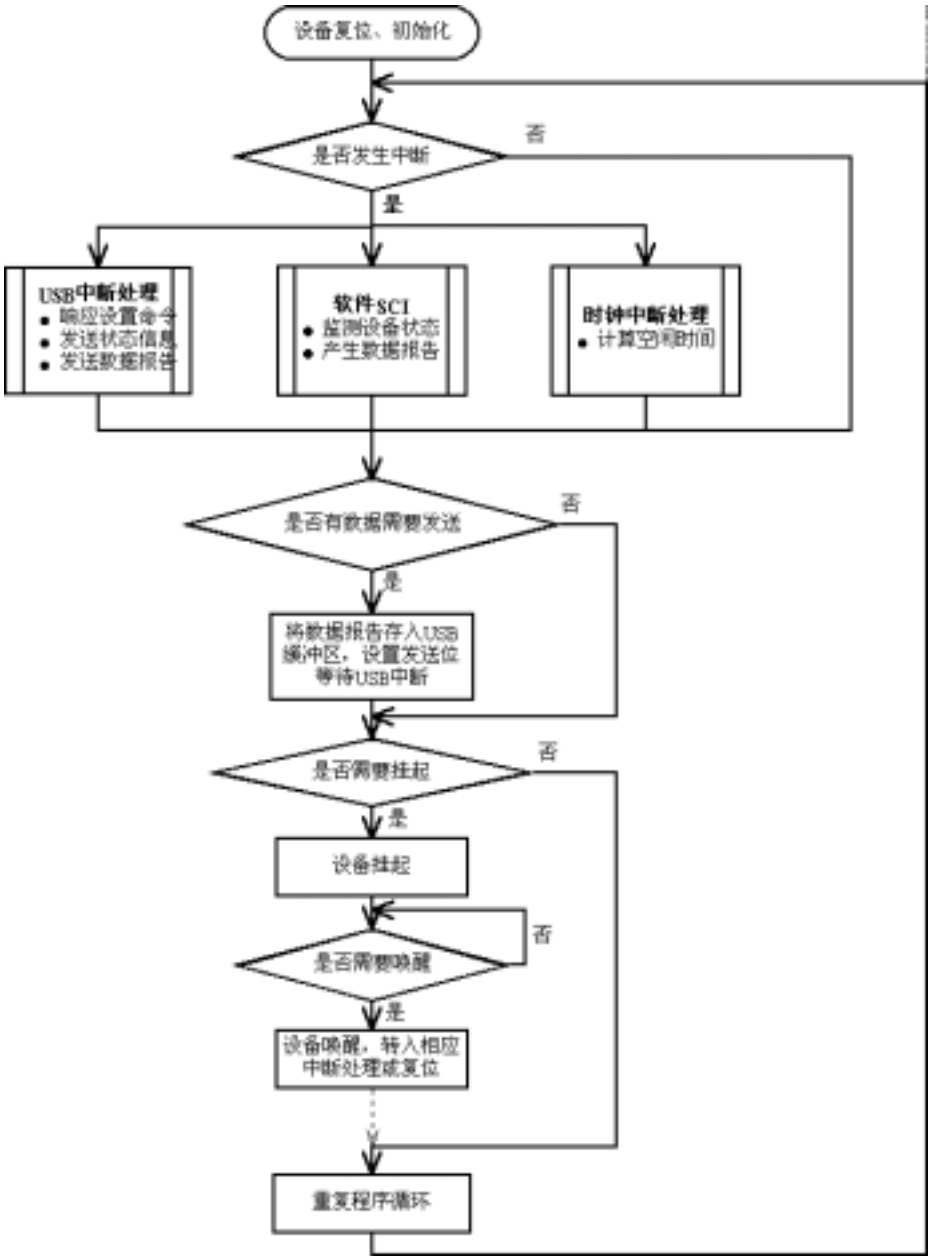


图 4.2　微控制器整体软件流程

## 4.2 HID 设备驱动程序编写

程序设计最终的难点是在 WINDOWS 98 下的驱动程序编写。本项目开始调研时，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。

### 4.2.1 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 下存在两种驱动程序：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，Microsoft 称它们为 Legacy（直译为：遗产、遗物等）Drivers；另一类驱动程序符合 WDM 1.0（Windows Driver Model 1.0）标准，而这种标准同样也是 Windows NT 4.0、Windows 2000 及以后的 Windows 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，因为毕竟谁也不知道 Legacy Drivers 还能被 Windows 系统支持多久。本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。

驱动程序可以工作在两种模式下，一是核心模式（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核心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核心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，去掉了核心模式下的驱动程序，只保留了用户模式下的驱动程序。

### 4.2.2 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。

### 4.2.3 HID 设备驱动程序

鉴于 USB 设备协议的复杂性和通用性，Windows 98 为开发者做了很多的支持工作，在两种模式下都提供了方便的编程接口。开发者无论是在哪个模式下开发驱动程序，都可以通过调用一系列 USB 设备专用的函数去完成主要的控制和通讯功能。用这种方式开发的驱动程序被称为客户驱动程序。

以 HID（Human Interface Device，USB 的功能子类之一）设备的驱动程序为例，Windows 98 提供的 HID 设备专用函数主要是在两个系统文件里实现：核心模式下使用的 Hidparse.sys 和用户模式下的 Hid.dll。通过 Windows 98 DDK 的编译器，开发者可以很容易地将自己的驱动程序里的调用同这两个文件联系起来。

图 4.3 是 Windows 98 下有关 USB 设备部分的驱动程序结构图，它描述了硬件层、核心驱动层、用户驱动层之间的关系。

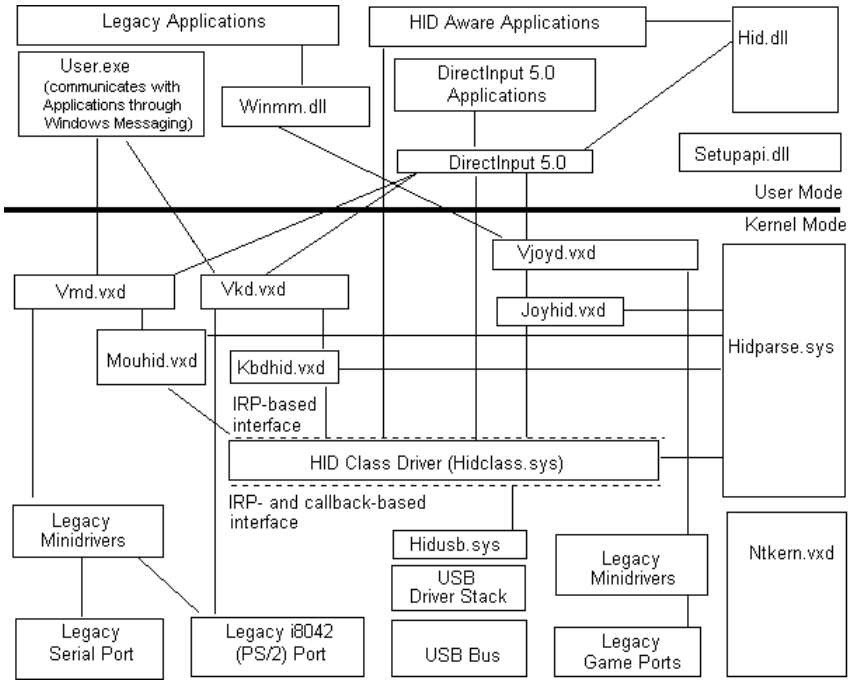


图 4.3　驱动程序结构

这些函数按功能可以分为两类。一类是用于初始化 USB 设备，建立该设备的一个客户驱动；另一类是用于完成设备数据的输入和输出。典型的函数例如 HidD_GetConfiguration（），可以将 HID 设备的配置信息传给驱动程序；HidD_SetConfiguration（），可以设置 HID 设备；HidP_GetUsage（），可以得到 HID 设备数据报告中的二进制值；……

在本项目中，最终的实现只有一个用户模式下的 EXE 文件，其功能主要是从 WINDOWS 98 底层驱动获得 HID 设备的信息和数据报告，然后以 WINDOWS 消息（MESSAGE）的方式发送给“汉王”的手写识别程序，可分为 HID 设备管理和 WINDOWS 消息发送两大模块。

#### 4.2.3.1　HID 设备管理模块

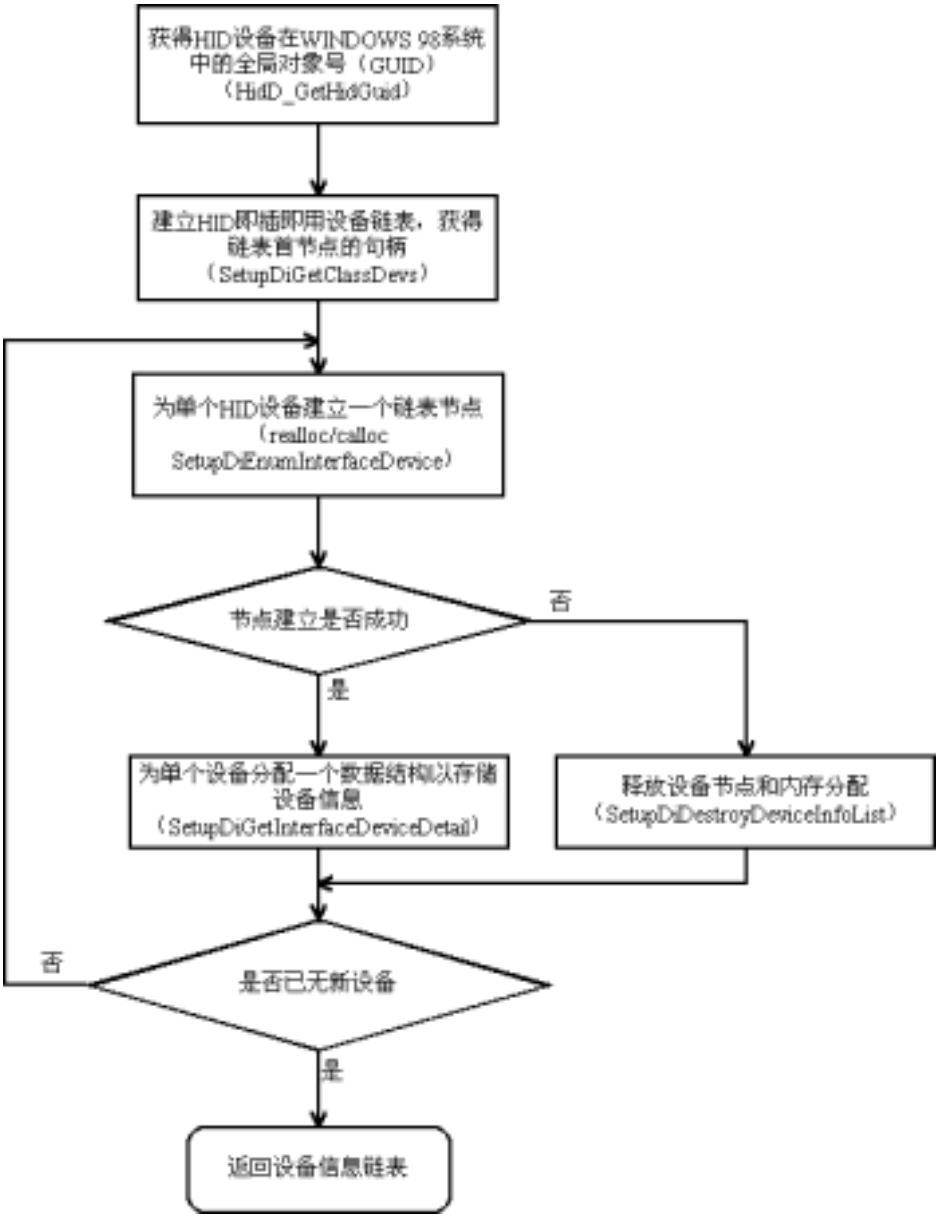


图 4.4　HID 设备链表的建立

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（图 4.~~图 4.），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。

第一部分是为 WINDOWS 98 中已安装的所有 HID 设备建立一个信息链表，这个链表就成为驱动程序的其他部分提供各个 HID 设备的接口。图 4.4 仅仅是这一部分中最关键部分的流程和函数调用介绍，供读者参考。

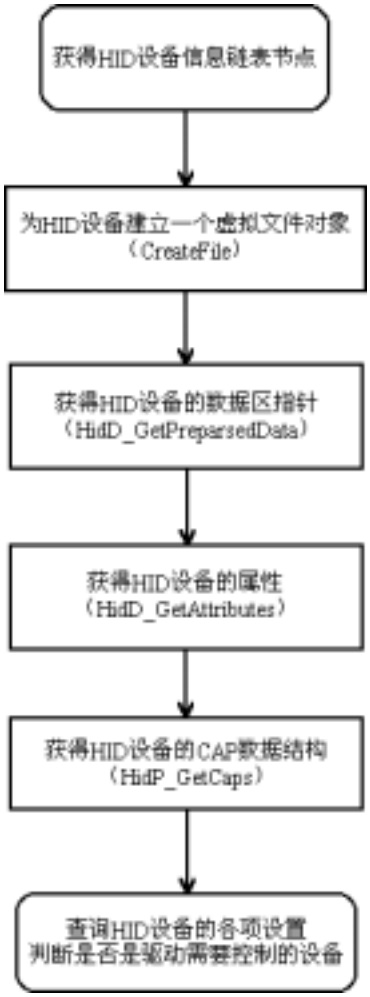


图 4.5　查询和判断 HID 设备信息

第二部分是获得单个 HID 设备的信息，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。

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。但即使是在二月份发布的 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。

#### 4.2.3.2 WINDOWS 消息发送模块

数据读取成功后驱动程序就以 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，具体的内容详见以下列表：

| 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。

# 第五章　调试运行与结论

本项目调试时采取先对设备端的软硬件进行测试，再联机调试驱动程序的方法。

在 USB 开发组织者的 INTERNET 站点上提供了对 USB 和 HID 设备的测试软件包——USB Compliance Test Suite。该软件包是由 MICROSOFT 和 INTEL 共同开发的，免费提供给 USB 开发者使用，不过使用者必须登记注册参加他们的开发者组织。软件包内共包括两个软件程序：HidView.exe 和 USBCheck.exe。图 5.1 是 HidView.exe 对本项目的 USB 手写板的测试报告，结果表明设备的协议兼容性良好，完全符合 USB 和 HID 协议的标准要求。

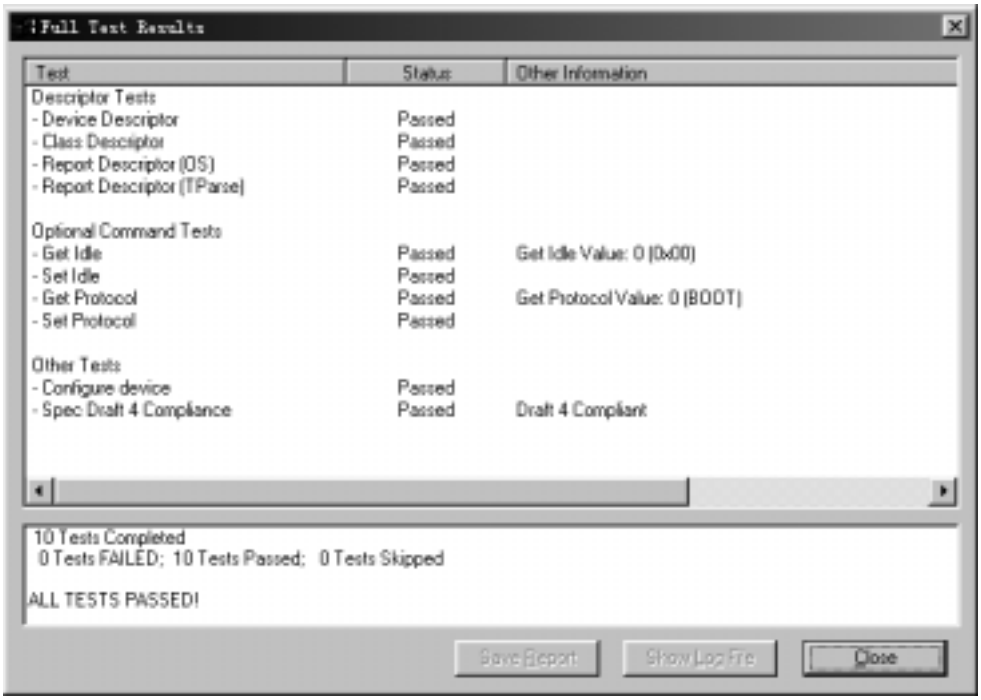


图 5.1　HidView 对设备的测试报告

驱动程序的调试过程相对较为困难，最初的识别系统响应速度始终很慢。后来

对代码进行了许多优化，又去掉了几个潜在的错误，系统的响应速度才达到了令人满意的程度。

本项目是得到了 MOTOROLA 公司技术和资金支持的演示项目，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 5.2）表示满意。


图 5.2　系统运行结果

本项目是对 USB 技术的有益尝试，为国内使用 MOTOROLA 微控制器开发 USB 设备作了一些基础性的研究。但毕竟只是演示项目，如果要真正应用到产品和市场中去，还有很多的工作要做。希望这次尝试，能对 USB 技术在国内的推广起到些微的作用，那本项目所做的工作也就算是有价值了。

# 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. MC68HC（7）05JB4 SPECIFICATION（GENERAL RELEASE），MOTOROLA INC., 1998
3. Universal Serial Bus Specification, Revision 1.0， USB Implementers' Forum, January 15, 1996
4. Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.0 Final. USB Implementers' Forum，1997
5. Universal Serial Bus HID Usage Tables, Release Candidate 1.0. USB Implementers' Forum，October 30, 1997
6. The Windows NT Device Driver Book: A Guide for Programmers. Art Baker.
7. Windows 98 DDK Documentation. Microsoft Corporation , 1998.
8. 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
9. 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