分类号 _____ 密级_____

UDC _____

华中科技大学

硕士学位论文

基于 MO 的数码相机嵌入式存储控制器

学位申请人：邓 剑

学 科 专 业：微电子学与固体电子学

指 导 教 师：杨晓非 教授

论文答辩日期 2005 年 5 月 13 日 学位授予日期 年 月 日

答辩委员会主席 李佐宜 评阅人 李佐宜 刘三清

A Thesis Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Engineering

The Embedded Storage Controller for Digital Camera Based on the MO

Candidate : **Deng Jian**

Major : **Microelectronics and Solid State Electronics**

Supervisor : **Prof. Yang Xiaofei**

Huazhong University of Science and Technology

Wuhan, Hubei 430074, P. R. China

May, 2005

独创性声明

本人声明所呈交的学位论文是我个人在导师的指导下进行的研究工作及取得的研究成果。近我所知，除文中已标明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：邓剑

日期：2005 年 5 月 13 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在_____年解密后适用本授权数。

本论文属于

不保密□。

（请在以上方框内打“ √ ”）

学位论文作者签名：邓剑　　指导教师签名：杨晓非

日期：2005 年 5 月 13 日　　日期：2005 年 5 月 13 日

摘 要

随着 USB 接口在消费类数码产品中的进一步推广，为实现数码设备直接连接，嵌入式系统中 USB 接口之间的通讯技术研究也就越来越迫切。受日本富士通研究所委托，研制了一套基于 MO 的数码相机嵌入式存储控制系统。该系统能够直接驱动 USB 接口的数码相机和 MO，实现数码相机照片到 MO 的直接备份。

USB 开发分为主机端开发和外设端开发，该系统属于主机端。传统的主机端开发主要是在 Windows 平台下进行，底层由 Windows 操作系统提供。脱 PC 机系统的开发必须从底层开始，其技术难点是必须开发一个嵌入式系统中使用的 USB 主机端协议栈。

主机端协议栈面向的对象是各种不同类型的 USB 设备。针对不同类型的设备，对应有不同的 USB 类协议以及相应的控制、管理方式。待管理的数码相机和 MO 一般属于海量存储类设备，除了 USB 海量存储类本身规定了数据传输方式外，还要考虑存储设备使用的透明控制命令集以及 FAT 文件系统管理等问题。

论文重点描述该系统由底层到顶层的实现过程，给出了 USB 主机端协议栈的一般结构。虽然系统使用的 USB 控制芯片是飞利浦公司的 ISP1362，但是除了控制寄存器的使用不同外，USB 协议栈的实现却是相通的。论文的研究成果对其它类似系统的开发具有参考价值。

最后，论文分析了该系统在嵌入式 USB 主机端开发以及系统扩展升级上的优点：该系统可进一步发展成为能够兼容多种 USB 设备的数据交换平台，分析了实现该系统需要做的相应技术工作。

关键词：USB 主机端　　嵌入式系统　　USB 海量存储类　　FAT 文件系统　　ISP1362

Abstract

It is urgent to develop a new communication technology for direct data transfer between embedded systems through the USB interfaces, which are widely employed in consuming digital products. This work develops an embedded storage management system based on MO for digital camera, which is consigned by Fujitsu Corp. The system can drive digital cameras and MOs through USB interface and it can read out the photos from digital camera then store them on MO disk directly.

There are two kinds of USB interfaces, host and slave. This system is a kind of host device, which was commonly developed under Windows system with the bottom layer support by the operating system. The key point of the system design is to develop a host USB data transfer protocol stack in embedded system, which must be built from the ground.

The host USB data transfer protocol stack faces a variety of USB devices. Digital cameras and MOs are USB mass storage class device mostly. It should be concerned that control codes used in MO and the management of the FAT file system besides obeying data transfer protocol defined in USB mass storage class specification.

This thesis describes the storage management system from bottom to top and gives a common structure of the protocol stack used in USB host device. Although ISP1362 is employed as the USB control chip here, the protocol stack has the same structure besides the slight difference between the control registers when other chips used. This thesis is a guide to the developers who want to add USB host interface in embedded system.

End of this thesis discussed the advantages of this system for easy USB host interface development in embedded systems and good expand ability. The system goal is to be a kind of common data interchange platform through USB interface finally.

Keywords: USB Host Embedded System USB Mass Storage Class FAT File System ISP1362

目 录

1 序 论

1.1 引言

USB 接口产生的最初目的是为 PC 机提供一种统一方便的外部总线扩展机制，为此，它采用了一种主机端、外设端分开设计的方法，并规定传输只能发生在主机端和外设端之间。以前，各种 USB 外设使用外设端协议，主机端集成在 PC 机上，这样所有设备以 PC 机为核心建立连接。但是，随着 USB 接口在消费类数码产品中的进一步推广，用户更希望数码设备之间能够直接建立连接。

从功能上看，消费类电子产品有这样两类：产生或使用数字数据类、保存数字数据类。纯粹保存数字数据的设备主要有：U 盘、移动硬盘、MO 等，这类设备存储容量通常比较大。产生或使用数字数据的设备主要有：数码相机、MP3、DV、掌上影院等，这类设备的本身带有一定存储功能，但是存储容量相对有限，经常需要拷贝或备份数据。在这两种设备之间的建立数据连接是一个很好的切入点。

USB 协议规定：USB 接口由 Host 端来发出控制命令，Slave 端被动的相应 Host 端的命令，数据传输只发生在 Host 和 Slave 之间，不允许 Host 到 Host 或 Slave 到 Slave 的传输[1]。以前，Host 角色由 PC 机充当，数码设备主要使用 USB Slave 的功能。所以，传统的只采用了 USB Slave 技术的数码设备不能直接通讯。

USB 数码设备必须围绕 PC 机才能充分发挥作用，显然是一个缺陷。这种情况下，USB 协议的制定者发布了针对 USB 设备脱 PC 机应用的 OTG（On-The-Go）协议，并在 USB 论坛 www.usb.org 上专门开了一个 OTG 版面来发布 OTG 的消息。芯片制造商也很快在 2001 年底推出了第一款的 OTG 控制芯片。

可以预见，数码设备的脱 PC 机传输必然是消费类电子的一个重要发展方向。对应于技术上来说，USB OTG 技术也就是 USB 接口开发的新方向。[2]

1.2 本文研究的主要内容

2002 年底，日本富士通研究所要求我们实验室开发一套基于 MO 的数码相机转存系统。要求该系统能够直接驱动数码相机和 MO，在脱离 PC 机的环境下实现数码相机照片到 MO 盘片的备份。由于现在的 MO 和数码相机都是使用 USB 接口的数码产品，所以该系统实际是 USB 接口的开发。

USB 的英文全称 Universal Serial Bus。它包含了 USB 接口最显著的两个特点：通

用和串行。“串行”表明接口的物理层比较简单，接口非常小，USB 接口只有 4 根线，用于电源管理的 Power 和 GND，以及用于数据传输的 D＋和 D－。“通用”意味着这个接口可以适合任何类型的设备。从操作系统上看，它适合于任何字符设备和任何块设备。从应用上看，它可以统一设备的接口，让设备连接非常方便。

为了达到“通用”的目的，USB 规范把设备分成 Host 端和 Slave 端两类：Slave 被动响应来自 Host 的各种命令，Host 负责设备类型识别、相关类驱动程序的调用、通讯过程中的检错纠错等。这样分类的以后，Slave 端的固件设计比较清晰，占用的系统资源比较小，非常适合嵌入式系统。Host 端虽然比较复杂，占用的资源也比较多，但是由于 Host 一般集成在 PC 机主板上，可以利用的资源非常丰富，所以 Host 系统不会成为传输瓶颈。

由于越来越多数码移动设备采用 USB 接口，在移动设备上集成 USB Host 功能的要求也就越来越迫切，这样产生了 OTG 规范。从芯片设计上看，OTG 芯片实际上是在一个裁减过的 Host core 和 Slave core 基础上扩展一个 OTG core，OTG core 的作用主要是协调和切换 Host 和 Slave 功能[3]。OTG 规范推出没多久，芯片也就上市了，但是成熟的 OTG 软件迟迟没有规范化。原因很明显，以前的 USB Host 端的软件一般是在 Windows 操作系统下开发，而 Windows 的底层驱动的源代码是不公开的。现在的 OTG 驱动程序没有统一的标准，唯一的依据就是 USB 规范和各式各样的 USB 类规范以及相应的协议。

我们要实现的系统实际上是在嵌入式系统中开发 USB OTG 功能，不过比较特殊的是该系统不是移植在 MO 或数码相机上，而是一种可以管理 USB 数码移动设备的独立的新型 USB 设备，相当于一个 USB 设备的数码交换平台。

该系统主要涉及的技术内容如下：

第一，系统的核心是实现 USB Host 功能。从芯片应用上看，OTG 芯片实际是在 USB Slave 芯片上集成 USB Host 最主要的功能，从而让采用该芯片的数码移动设备可以互相通讯[4]。显然，要开发一个 OTG 设备，软件上必须包含 Slave 端和 Host 端的驱动程序。目前，移动设备的 USB Slave 技术已经比较成熟，OTG 实现的关键就是不断完善嵌入式 USB Host 技术。如果是仅仅实现富士通公司的要求，其实只要使用 Host 技术就可以了。但是为了考虑到以后的进一步应用，我们仍然选择了一款 OTG 芯片 ISP1362。

传统的 USB Host 开发在 PC 机的操作系统环境下进行，USB 的各种底层驱动都由 Windows 等操作系统提供，开发都是集中在应用层上。由于 USB 协议的复杂性等原因，现在各种流行的嵌入式操作系统基本上也不支持或者仅仅部分的支持 USB

Host。所以要开发在移动设备上的 USB Host 或 OTG 功能一般要从底层开始设计[5]。进行 USB Host 开发，不仅仅要了解 Host 需要做什么，还要关心 Slave 设备会怎么响应，所以，开发者要对整个 USB 体系比较熟悉。

第二，我们系统要控制的是带有存储功能的设备：数码相机和 MO。一般情况下，这些设备都是使用 FAT 文件系统。由于系统是从底层开始设计的，所以系统内部要实现一个简单的 FAT 文件管理系统。

第三，我们系统要控制两个 USB 设备，这必须用到 USB Hub。我们采用的 ISP1362 本身内置了 USB Hub，能管理 2 个 USB 设备。ISP1362 中，两个 USB 接口使用同一个缓冲器，同时管理两个设备的时候，要考虑到两个设备的协调工作。

由于系统本身不关心外面是个什么设备，更关心的是设备使用什么接口和相关协议来进行通讯。所以该系统完成后，我们发现，它可以管理所有采用 USB 接口的，使用 USB 海量存储类协议设计的数码设备。这类设备有：U 盘，MP3，移动硬盘，MO 和大部分的数码相机等等。也就是说，只要这些设备按照 USB 海量存储类标准协议来设计，我们系统就可以实现两个设备间的数据传输。

2 嵌入式 USB Host 系统设计

这个系统最初用 ISP1362 的 PCI 型开发包进行开发，开发软件为 TC3.0。后来制作样机，采用了现在比较流行的 ARM7 嵌入式硬件系统。系统没有移植任何嵌入式操作系统，系统软件结构完全自己定义。本章主要说明在制作样机过程中，系统在硬件上和软件上的一些全局性考虑。

2.1 硬件框架

我们设计的整机的硬件系统框图如图 2－1 所示。下面对硬件上的考虑做一些说明。

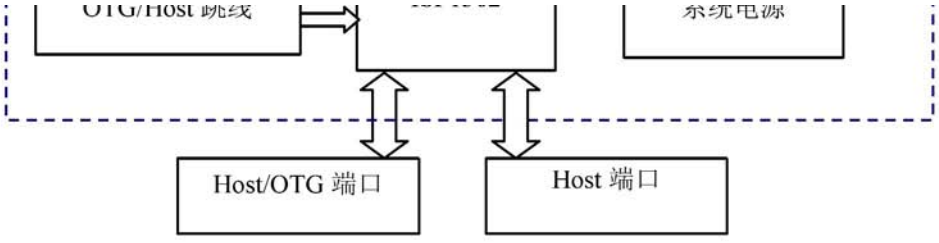


图 2-1 系统硬件模块简图

2.1.1 中央处理系统

这里所说的中央处理系统是指：MCU、Flash 和 SDRAM。

考虑到系统的扩展性和处理速度，对样机试验板初步要求主要有：有足够大的内存，方便数据的腾挪转移以及以后可能做的数据处理；MCU 速度要快，不能成为系统的瓶颈；系统扩展性要比较好，可以扩展出多种接口与 USB 设备通讯。

这里选用的 MCU 为 ARM7 核心的 S3C44B0X，它本身带有 IDE 等多种接口，运行速度也在 60MIPS 左右。S3C44B0X 的功耗低；自身集成了多种功能，综合成本比较低[6]；C 语言编译器使用 ARM STD 2.51[7]，通过 JTAG 接口烧写 Flash 程序，使用 JTAG 接口在线调试或者串口调试[8]。这些相关软硬件资料在网上非常多，而且免费使用。

2.1.2 USB 控制芯片——ISP1362

USB 的控制芯片从系统设计上看有两种类型：内置 CPU 和非内置 CPU。Cypress 公司的 USB 芯片是内置 CPU 的典型代表；Philips 公司的 USB 芯片一般不内置 CPU，是外部总线扩展的典型代表[9] [10]。出于系统灵活性的考虑，我们选用了 ISP1362。

ISP1362 是 Philips 公司生产的 USB1.1 的 OTG 控制芯片，这里只说明一些适合我们系统的主要特点，具体参数和其它特点可以参考 Datasheet[11]：

- 一个 OTG 接口和一个 Host 接口，OTG 接口可以通过跳线设定成 Host 模式、Slave 模式和 OTG 模式。
- 芯片内部集成一个 USB Hub，可同时控制两个 USB 设备进行工作。
- 即将上市的 USB2.0 OTG 芯片 ISP1761 和它一脉相承，可以方便的升级。

我们系统主要是使用 Host 功能，这里保留 OTG 和 Host 跳线，先实现 Host，再升级到 OTG。

2.1.3 外围电路设计

这里的外围电路主要有：电源、LCD 、LED 和控制键盘。

电源

系统中各个地方使用的电压：LCD 和 USB 接口规定的 5v；ISP1362 使用 3.3V；S3C44B0X 使用 3.3v 和 2.5v。其中 ISP1362 和 S3C44B0X 对电压的稳定性要求比较高。

如果使用 2 节电池供电，供电电压为 2.5v 到 3v 左右。需要使用直流升压芯片，先把电压升到 5V，然后使用降压芯片把电压降下来，降到稳定的 2.5v 和 3.3v。

如果使用 4 节电池供电，供电电压 5v 到 6v 左右。这个电压是 USB 接口和 LCD 可以接受的电压范围，所以直接把电压降到稳定的 2.5v 和 3.3v 即可。

2 节电池供电，由于有一个升压过程，会让系统耗电量变大，成本增加；直接降

压的情况下，耗电量相对比较小，所以这里采用 4 节电池供电。

5V 到 3.3V 选用 Alpha 公司的 AS1117_330，它输出电流最大值为 800mA.。5V 到 2.5V 选用 AS1117_250，最大输出电流也是 800mA。

LCD：选用显示 2X20 个英文字母的 CM1662（它与 S3C44B0X 都是 Samsung 的产品，兼容性比较好）。使用 S3C44B0X 内部集成的 LCD 驱动模块驱动。

LED：用来指示系统运行情况，使用 S3C44B0X 的通用 PORT E GROUP 的高 4 位控制 4 盏 LED 指示灯。

键盘：初步采用一键传输，直接使用 Reset 引脚。（以后按照需要可进行扩展）

2.2 软件框架

2.2.1 实现 USB Host 系统的基本要素

从不同的层次上对 USB 通信流进行考察，完整的 USB Host 系统应该包括以下三种基本要素[12] [13]：

- 主控物理/数据链路层(Host physical/data link layer)，这一层主要负责 USB 物理信号的生成和数据包的发送与接收，并进行物理校验。
- 主控系统栈（Host system stack），包括协议管理，Slave 设备列举、事务调动及状态处理等。它控制较底层的活动，这一层针对所有类型的 USB 设备，并为 USB 设备驱动的连接提供一个共用接口。
- 设备驱动（Device driver），主要负责与某一种特定的类型的 USB Slave 设备相对应。USB IF 已经把包括海量存储在内的多种 USB 外设标准化，归入各种 USB 类中。

对于我们的系统，主要是针对海量存储类设备，还要涉及一个文件系统。现在数码移动设备，大多数采用的是 FAT 文件系统。所以这里还要编写一个 FAT 文件管理系统。

如果从硬件控制上算起，由下向上，软件控制的关键为：实现数据总线的访问－>实现寄存器的访问－>实现 4 种基本传输方式－>实现 11 种标准控制传输命令－>实现海量存储类设备控制命令的发出－>FAT 文件管理系统[14]。

2.2.2 系统软件结构

我们开发的嵌入式 USB Host 系统中，驱动程序的模块的组成结构如图 2-2 所示。

位于最底层的是 ISP1362 固件设计，负责 ISP1362 的硬件控制。这一层与 ISP1362

芯片共同实现上面的“主控物理/数据链路层”。

往上的一层是 USB 基本驱动。主要是按照 USB 协议设计 11 个标准请求，获取设备基本信息和列举设备。这一层实现“主控系统栈”的功能。

嵌入式系统中不会支持所有的 USB 设备，应该根据需要，设计相应类设备驱动，类设备驱动层即上面提到的“设备驱动”。这里是 USB 海量存储类设备。

再往上就是封装相应的 API 函数。我们系统中，API 函数主要是为 FAT 文件系统服务。设备一般使用 SCSI 命令中 read10 命令和 write10 命令来读写 FAT 文件系统绝对地址的扇区。我们可以按照 FAT 文件管理系统的要求，提供相应的绝对地址或相对地址扇区的读写 API 函数。

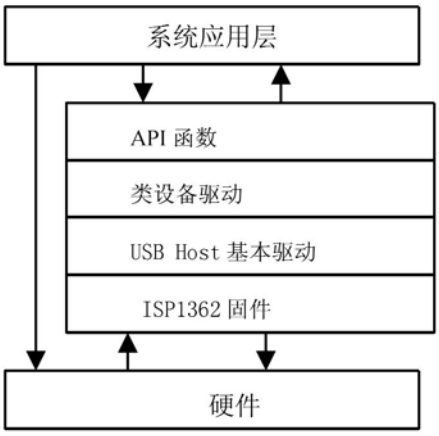


图 2-2　Host 端软件结构图

2.2.3　速度分析

传输速度是 USB 开发非常重要的一个问题，很多开发完成后都发现传输速度不够理想，下面对 USB 系统的传输速度做一个简单分析。

整个系统的速度由 USB Slave 设备速度，USB Host 设备速度和 USB 协议本身速度综合决定。因为控制命令的发出和传输的状态检查都是 USB Host 系统的工作，Host 的负担远远大于 Slave，所以，不论在 PC 机上还是在嵌入式系统中，USB Host 的系统速度基本上决定了整个 USB 传输的速度。

而 USB Host 系统的瓶颈，主要集中在对的 USB 控制芯片缓冲区的读写上，一般都要采用 DMA 方式来访问缓冲区。另外一个比较重要的因素就是 MCU 的速度，只要成本许可，MCU 越快越好。同时，软件上的代码优化和采用 Ping/Pong 传输也可以明显提高传输速度。

这里提供我们系统的传输速度供大家参考： S3C44B0X 在 60MIPS 的情况下，整

个系统的传输速度大约为 150Kbyte/Sec。

这里还有一个问题，我们系统需不需要移植一个嵌入式操作系统？分析如下：

移植嵌入式操作系统后，最大的好处是可以进行进程调度，充分合理的使用硬件资源[15]。但是我们的系统实际上只有一个 USB 传输进程。USB 接口的简单化和通用化是以 USB 协议的复杂性为代价的，USB 传输特别是 Host 的传输比较耗费 MCU 资源，即使在 PC 机的 Windows 系统下，如果进行 USB 传输，操作系统都会明显变慢。出于速度考虑，这个系统最好不要移植操作系统，全力处理 USB 传输即可。

3 ISP1362 固件设计

所有烧入 Flash 的系统控制程序都可以称为固件（Firmware）。整个系统的固件除了这一层，还应包括后面的 USB 协议栈以及 FAT 文件管理系统等。本章的固件设计，主要针对 ISP1362 硬件。主要说明在我们设计的嵌入式硬件系统中，如何通过访问 ISP1362 的控制寄存器，产生 USB 规范中的四种基本传输：控制传输，实时传输，中断传输和批量传输。

3.1 系统总线的读写

S3C44B0X 作为主控制器，它与 ISP1362 连接如图 3-1。

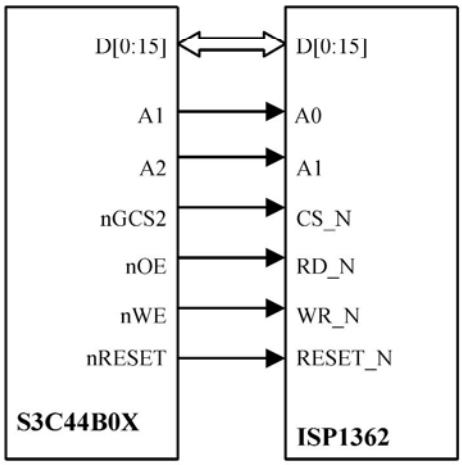


图 3-1 MCU 与 ISP1362 连接图

首先编写 outport 和 inport 函数来读写 ISP1362 对应地址的 16 位数据线：

```
//写规定地址的 16 bit 数据
void outport(U16 address,U16 ndata)
{ __asm
  { mov   r0,#0;add   r0,r0,ndata ; strh r0,[address];}
}
//读取规定地址的 16 bit 数据
U16 inport(U16 address)
{ U16 data_return;
```

```
 __asm
  {ldrh data_return,[address];}
  return(data_return) ;
}
```

这里 outport 和 inport 函数是仿照了 TC3.0 中的计算机总线读写的同名函数来设计的[16]。在 ISP1362 的 PCI 型开发包中，就是用 TC 中的这两个函数访问开发包上的 ISP1362 芯片[17]。

nGCS2 片选信号在 S3C44B0X 中是作为 SDRAM 扩展用的，我们这里用作 IO 扩展，需要禁止该地址的 Cache。

3.2 ISP1362 寄存器的访问

ISP1362 读/写寄存器过程为：输入寄存器地址—>读/写寄存器的低 16 位数据—>读/写寄存器高 16 位数据（仅用于 32 位寄存器）。

这样，系统数据总线上的数据有两种类型：要访问的寄存器地址值和该寄存器的实际读写值。ISP1362 上的 A0 和 A1 用于区分数据总线上要读写数据的类型。A0 区分数据总线上要读写的是寄存器地址还是寄存器值。A1 用来区分读写 Host 寄存器还是 Slave 寄存器。只要正确设定 A0 和 A1 的值，在 inport 函数和 outport 函数基础上就可以编写寄存器的读写控制函数[18]。

nGCS2 作为片选信号，它在 S3C44B0X 中对应地址是从 0x04000000 开始，所以 ISP1362 上的 A0 和 A1 选中的 4 个地址定义为：

```
#define hc_data 0x04000000    // Host 寄存器数据输入输出阶段
#define hc_com 0x04000002    // Host 寄存器地址写入阶段
#define dc_data 0x04000004    // Slave 寄存器数据输入输出阶段
#define dc_com 0x04000006    // Slave 寄存器地址写入阶段
```

最后，以 32 位 Host 寄存器的读函数为例子说明：

```
U32 r32(U8 reg_no)
{
 U16 result_l,result_h;
 U32 result;
  outport(hc_com, reg_no);//要访问的 Host 地址输入
  result_l=inport(hc_data);//读出的寄存器值的低 16 位
  result_h=inport(hc_data); //读出的寄存器值的高 16 位
```

```
    result = result_h;
    result = result<<16;
    result = result+result_l;
    return(result);
}
```

3.3 USB 传输事务

所有的 USB 通讯传输都可以归结为三种包的传输：令牌包（Token Packet），数据包（Data Packet）和应答包（Handshake Packet）。通过这三种包的组合，就构成 USB 的基本传输单位——传输事务（transaction）。

传输事务组成：Host 首先发送一个令牌包，主要描述操作类型、方向、外设地址和端点号等（这就是任何操作都必须从 Host 开始的原因）；然后数据源部件发送数据包进行数据传输或者报告没有数据传输；最后数据的目的地给出一个应答包表示传输是否成功[19] [20]。

3.3.1 域

一个包由不同的域组成，域的类型如下：

- 同步域（**SYNC field**）

用于本地时钟与输入信号的同步，定义为 8 位长的二进制串，为 NRZI 编码的二进制串“KJKJKJKK”。最后的 2 位是同步域结束的记号，并且标志了包标志域（PID）的开始。

- 包标志域（**PID，Packet Identifer field**）

指名了包的具体类型。结构如图 3-2。表示的类型见表 3-1。USB 固件开发，一般需要开发者自己设定 PID 参数。

- 地址域（**Address field**）

包括两个子域，外设地址（port address）和端口地址（endpoint address）。外设地址的作用是找到唯一的 USB 设备，端口地址的作用是找到设备上对应的的端口(USB 传输中把一个大缓冲区分成若干小缓冲区，每个小缓冲区称为一个端口)。外设地址共 6 bit，可以支持 127 个设备。端口地址 4 bit，最多支持 16 个端口。

- 帧号域（**Frame Number field**）

USB 把每 1ms 的传输称为一帧，它是 USB 传输占用总线的最小时间单位。帧号域是一个 11 bit 的字段，主机每过一帧就将其内容加 1。它仅在 SOF 包中被发送。

- 数据域（**Data field**）

可以在 0 到 1,023 Byte 之间变动，但必须是整数个字节。

- **CRC** 校验

循环冗余校验（CRC）用来校验所有的非 PID 字段。

表 3-1　PID 类型

PID 类型	PID 名	PID[3:0]	描述
标记（Token）	输出（OUT）	0001B	在主机到功能部件的事务中有地址+端口号
	输入（IN）	1001B	在功能部件到主机的事务中有地址+端口号
	帧开始（SOF）	0101B	帧开始标记和帧号
	建立（SETUP）	1101B	在主机到功能部件建立一个控制管道的事务中有地址+端口号
数据（DATA）	数据 0（DATA0）	0011B	偶数据包 PID
	数据 1（DATA1）	1011B	奇数据包 PID
握手（Handshake）	确认（ACK）	0010B	接收器收到无措数据包；
	不确认（NAK）	1010B	接收设备部不能接收数据，或发送设备不能发送数据；
	停止（STALL）	1110B	端口挂起，或一个控制管道请求不被支持。
专用（Special）	前同步（PRE）	1100B	主机发送的前同步字。打开到低速设备的下行总线通信。

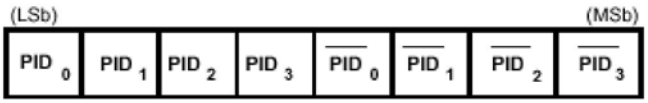


图 3-2　PID 结构图

3.3.2 包

USB 总线使用了四种包，不同包的域如下：

（1）令牌包

由表 3－1 知道，令牌包有：OUT，IN，SETUP 和 SOF 四种类型。其中 OUT、IN、SETUP 的包的格式如图 3-3。

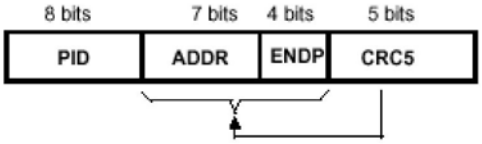


图 3-3 令牌包结构

SOF 包的格式如图 3-4。

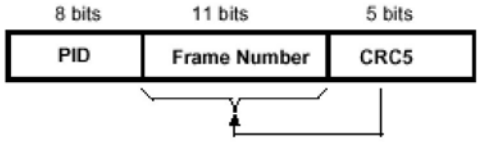


图 3-4 SOF 包结构

（2）数据包

数据包由PID，包括至少0个字节数据的数据区和CRC构成，如图3－5。有2种类型的数据包，根据不同的PID （DATA0和DATA1）来识别。2种数据包PID是为了支持数据切换同步（Data Toggle Synchronization）。

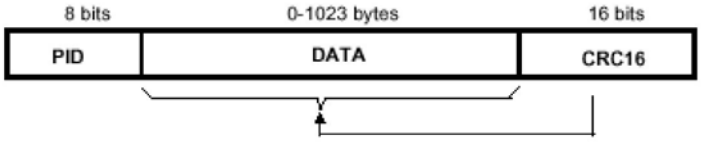


图 3-5 数据包结构

（3）握手包

如图3-6所示，握手包仅由PID构成。握手包用来报告数据事务的状态：命令的接收或拒绝。只有支持流控制的事务类型（控制、中断和批量传输）才能返回握手信号。握手包由1个字节的EOP确定界限。如果包被解读为合法的的握手信号，但没有以1个字节的EOP终止，则它被认为是无效的，且被接收机忽略。

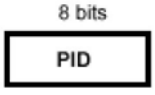


图 3-6 握手包结构

有3种类型的握手包：**ACK**表示数据包没有数据字段上的CRC错，并且数据PID被正确收到。**NAK**表示Slave不会从Host接受数据（对于输出事务），或者Slave没有传输数据到Host（对于输入事务）。**STALL**作为输入标记的回应，或者在输出事务的数据时相之后由Slave返回。STALL表示Slave不能传输，或者正在接收数据，或者不支持一个控制管道请求。在任何条件下都不允许Host返回STALL。

Slave的STALL握手有两种情况：

第一种情况，当设置了与端口相联系挂起特征（Halt feature）的时候，称为“功能STALL（functional stall）”。功能STALL的特殊情况是“命令STALL（commanded stall）”。如果slave的端口被挂起，则Slave必须继续返回STALL，直到引起停止的条件通过Host干涉而被清除。在后面要用到的Bulk Only传输中，Slave系统的状态报错就是通过它来实现。

第二种情况称为“协议STALL（protocol stall）”。协议STALL和功能STALL在意义和持续时间上是不同。协议STALL在控制传输的数据或状态阶段自动产生并返回，并且，STALL条件在下一个控制传送的建立事务阶段自动终止。

（4）特殊包（Special Packet）

特殊包的PID称为PRE（Preamble），用于低速传输。

3.4 利用 ISP1362 产生 USB 传输事务

USB 传输的最小单位是传输事务（Transaction）。ISP1362 使用 PTD（Phillips Transfer Descriptor）来产生各种传输事务[21]。

3.4.1 PTD

ISP1362 的 PTD 大小为 8 字节，结构如图 3－7。

Bit	7	6	5	4	3	2	1	0
Byte0	ActualBytes[7:0]							
Byte1	CompletionCode[3:0]				Active	Toggle	ActualBytes[9:8]	
Byte2	MaxPktSize[7:0]							
Byte3	EndpointNumber[3:0]				B3_3	Speed	MaxPktSize[9:0]	
Byte4	TotalBytes[7:0]							
Byte5	B5_7	B5_6	B5_5	B5_4	DirToken		TotalBytes[9:8]	
Byte6	Reserved	FunctionAddress[6:0]						
Byte7	B7[7:0]							

图 3-7 ISP1362 的 PTD 结构图

这个 PTD 不仅仅包含了包的基本参数，也包含了传输状态的显示以及 4 种传输方式中的一些特别设定。各个区域的具体功能如下：

ActualBytes：已经传输数据的字节数。占用 10bit，可标志 0－1023 个字节。

Toggle：Data0 还是 Data1。

Active：如果为 1，表示 ISP1362 允许该传输进行。传输完成后，芯片会把该位置 0。

CompletionCode：反映这次传输是否成功，如果出错，则反映出错代码。对应的含意以及我们系统的纠错处理见下表 3－2。

上面四个参数在传输中是不断变化的，它们的更新由硬件自动完成。

MaxPktSize：每个传输数据包数据段的大小。

Speed：USB 速度。0 表示 USB1.1 的速度，1 表示 USB1.0。

EndpointNumber：终端地址。

TotalBytes：总共需要传输的数据量。

DirToken：令牌包的类型，0 是 SETUP，1 为 OUT，2 是 IN

FunctionAddress：外设地址。

B？_？：针对不同的传输方式有不同的含意。

表 3-2　错误代码及纠错方法一览表

代码值（**Bin**）	代码含意	我们系统中的处理方法
0000	传输成功	继续下次传输
0001	CRC 出错	重新上次传输
0010	数据包长度出错	重新上次传输
0011	Toggle 不匹配	找出正确的 toggle，重新上次传输。
0100	Stall 错误	如果是协议 Stall，等待 1ms，重新上次传输。 如是功能 Stall，清除端点的 stall 状态，重新传输
0101	设备没相应	等待一段时间，重新发送命令。如果多次重试失败，重新列举设备。
0110	PID 检查失败	重新上次传输
0111	无法识别的 PID	重新上次传输
1000	传输包长度大于 MaxPktSize 值	重新上次传输
1001	传输包长度小于 MaxPktSize 值。	重新上次传输
1100	IN 传输时候，传输数据速度快过了系统写入 buffer 速度。	重新上次传输
1101	OUT 传输时候，传输数据速度快于系统写入 buffer 速度	重新上次传输
其他值	保留值	重新上次传输

3.4.2 传输事务的生成

按照 PTD 结构图，我们首先用 C 语言构建一个 PTD 数组。

```
//PTD 要使用的各个参数
struct ptd_struct
{
 U16 c_code;    U16 active_bit;    U16 toggle;      U16 actual_size;    U16 endpoint;
 U16 last_ptd;  U16 speed;         U16 max_size;    U16 pid;            U16 total_size;
 U16 format;    U16 func_addr;     U8 fm;
}
ptd2send;
//生成 PTD 的函数
void new_make_ptd(U16 *rptr,U8 token,U8 ep,U16 max,U8 tog,U8 addr,U8 port, U16 total)
{
//基本参数设定
 ptd2send.c_code=0x0F;
 ptd2send.active_bit=1;
 ptd2send.toggle=tog;
 ptd2send.actual_size=0;
 ptd2send.endpoint=ep;
 ptd2send.last_ptd=0;
 ptd2send.speed=port_speed;
 if(port==1) {ptd2send.speed=port1speed;}
 if(port==2) {ptd2send.speed=port2speed;}
 ptd2send.max_size=max;
 ptd2send.total_size=total;
 ptd2send.pid= token;
 ptd2send.format=0;
 ptd2send.fm=0;
 ptd2send.func_addr=addr;
//构造一个 8 byte 的 PTD 数组
 *(rptr+0)=      (ptd2send.c_code              &0x0000)<<12
                |(ptd2send.active_bit      &0x0001)<<11
```

```
                |(ptd2send.toggle          &0x0001)<<10
                |(ptd2send.actual_size     &0x03FF);
*(rptr+1)=      (ptd2send.endpoint         &0x000F)<<12
                 |(ptd2send.last_ptd       &0x0001)<<11
                |(ptd2send.speed                 &0x0001)<<10
                |(ptd2send.max_size              &0x03FF);
*(rptr+2)=      (0x0000                    &0x000F)<<12
                |(ptd2send.pid                   &0x0003)<<10
                |(ptd2send.total_size       &0x03FF);
*(rptr+3)=      (ptd2send.fm                  &0x00FF)<<8
                |(ptd2send.format           &0x0001)<<7
                |(ptd2send.func_addr         &0x007F);
}
```

注：这实际上是一个用于控制传输和批量传输的 ATL 区的 PTD 结构，用于中断传输的 INTL 区和用于实时传输的 ISTL 区的 PTD 构建函数省略了，有关 ATL、INTL 和 ISTL 的说明见 **3.6.1** 章节。

下面对 new_make_ptd()函数中各个参数做个说明

生成的 PTD 放在*rptr 的指针所指向的数组中。这个 PTD 数组，包含了一个传输事务的所有参数。一个传输事务由主机发出令牌包，信息源发出数据包和接收器发出的握手包构成。

令牌包的结构图见图 3-3。该函数中的“token” 参数实际是指令牌包的 PID 域，有 SETUP、IN 和 OUT 三个选项；“ep”参数对应 ENDP 域；“addr”参数对应 ADDR 域；“port”参数是为了管理 hub 方便而引入的一个变量，在一般情况下等于“addr”。

数据包的结构见图 3－5。参数“tog”对应于它的 PID，有两个选择 Data0 或 Data1；“max”是这里 Data 区的大小。对应不同的传输方式，有它规定的值。

握手包和上面两个包的 CRC 域由 ISP1362 硬件完成。

最后一个“total”参数是在四种基本传输过程中，传输的数据总量。用它除以“max”参数，就可以推算出这次基本传输中有多少个相同的传输事务。

3.5 USB 的四种基本传输方式

为了满足不同外设和用户的要求，USB 提供了四种基本传输方式：控制传输，实时传输，中断传输和批量传输。它们的数据格式、传输方向、数据包容量限制、总线访问限制等各不相同[22]。通过传输事务（transaction）的组合，就构成了 USB 的四种基本传输（Transfer）。

3.5.1 控制传输

控制传输主要用于传输设备控制命令，状态信息和确认命令。它是所有 USB 设备必须支持的传输方式。

控制传送最少有2个事务阶段：建立和状态。数据阶段为可选。下图是控制传输的3种基本传输情况。

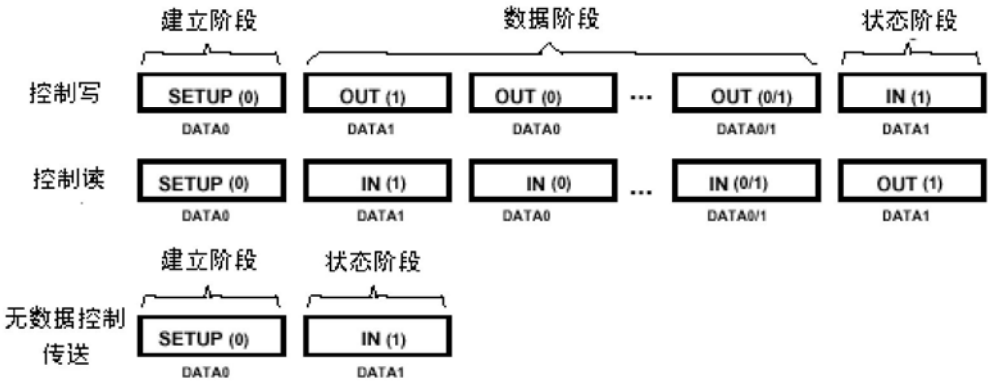


图 3-8 控制传输读写序列

建立阶段，由SETUP令牌包，8 字节的数据包，和一个ACK握手包组成（SETUP事务）。SETUP令牌包用于向功能部件的控制端口传输信息。8 字节的数据包使用DATA0 PID，用于发送控制命令，比如后面提到的11种标准请求的代码就这在这里发出的。最后是ACK应答，如果数据被损坏，则丢弃数据且不返回握手。

数据阶段，如果有的话，由一个以上的输入或输出数据事务构成，所有的数据阶段里的事务都必须有相同的方向（全部输入或者全部输出）。每个输入输出事务是由IN/OUT令牌包，设定长度的数据包（8、16、32或64字节），以及一个握手包组成。在数据包时相中要发送的数据的数量和其方向在建立阶段里被指定。

状态阶段是序列中的最后一个操作。由一个IN/OUT令牌包，零长度的数据包和一个握手包组成（零长度数据事务）。

3.5.2 批量传输

批量传输主要是用在大量的，对时间没有严格要求的设备上。数码相机、MO和U盘等与存储设备有关的USB设备一般都要使用批量传输。如图所示，从形式上看，它相当于一个控制传输的数据阶段。

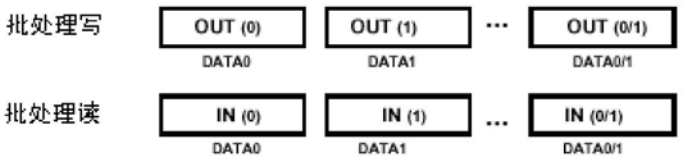


图 3-9　批量传输读写序列

3.5.3　中断传输

中断传输的典型应用就是HID设备，如鼠标、键盘等。特点是非周期的，自然发生的，数据量比较小。

中断传输包的结构和事务格式与批量传输差不多。由IN/OUT令牌包，设定长度的数据包（1到64中的任意值），以及一个握手包组成。

3.5.4　实时传输

实时传输有时翻译为同步传输。主要用于对数据正确性要求不高而对时间比较敏感的设备。如摄像头、麦克风等。

实时事务有令牌包和数据包，没有握手包。实时传输不支持握手时相或重试能力。

实时传输时，设备或主机控制器都应该能接受 DATA0 和 DATA1。如果设备或主机控制器只发送 DATA0，数据仍然会被接受。实时传输不支持切换时序。

3.6　ISP1362 产生四种基本传输

任何一款 USB 控制芯片的作用都是通过硬件产生 USB 的 4 种基本传输，然后在此基础上，依照相关的协议设计相应的 Host 端或者 Slave 端的驱动程序。[23]

虽然有些 USB 芯片只支持其中的部分传输，例如鼠标的芯片可能只使用控制传输和中断传输，但是像 ISP1362 这种 Host 芯片，面向所有 USB 设备，必须支持所有的基本传输。ISP1362 作为 Host 端的控制芯片，它主要是产生 4 种传输的而不像 Slave 端那样去响应 4 种传输。我们系统中没有用到实时传输，这里就省略了。

3.6.1 Slave 设备的终端与 ISP1362 的 Host 缓冲区分区

Slave 设备的终端（endpoint）与 4 种传输方式密切相关。

USB 是一种串行通讯的规范，串行通讯一般需要一定的数据缓冲区，USB 把缓冲区还进行了专业化的划分，把一个大的缓冲区划分成若干小的缓冲区，USB 把这些小的缓冲区称为终端。

一个 USB 的传输事务都是以一个设备的终端开始，或者以一个终端为目的来进行的。Host 没有终端，终端是针对 Slave 设备的概念。USB 规范定义一个 Slave 设备的终端为“一个 USB 设备的唯一可寻址部分，用来作为 Host 和 Slave 之间通讯流的信息源或接收器”。[24]

零终端是一个特殊的终端，它是唯一可以传输 SETUP 事务并进行控制传输的终端，也是所有 USB 设备必须支持的终端，所以该终端又称为默认终端（default endpoint）和控制终端（control endpoint）。除了零终端是一个双向的终端外，其他的终端都是单向的（虽然协议这么规定，现在的芯片好像也有做成双向终端的，这个终端必须在保证数据同步的问题上做充分考虑）。这样，除了零终端，其它终端要么是 IN 终端，要么就是 OUT 终端。

芯片设计时，由于不同的传输方式的传输机理不同，往往一个终端对应一种固定的基本传输方式。比如，我们实验中 MO 的 USB 芯片终端情况如下：它共有 4 个终端，终端 0 用于控制传输，终端 1 用于批量传输的 OUT，终端 2 用于批量传输的 IN，终端 3 用于中断传输的 IN。

ISP1362 Host 端的数据传输缓冲区共有 4K，通过设置相关的寄存器参数，把它划分成 3 种类型的区域，分别为：用于控制传输和批量传输的 ATL 区；用于实时传输的 ISTL 区；用于中断传输 INTL 区。

每个区域的前 8 个 byte 用于存放 PTD，PTD 之后是待传输的数据。通过寄存器把 PTD 和需要传输的数据发送到不同缓冲区分区后，就可以使用该缓冲区对应的 USB 传输方式来传输数据。

3.6.2 控制传输

ISP1362 的控制传输和批量传输的 PTD 格式基本相同，所以使用同一个缓冲区（ATL 区）来管理这两种传输方式。控制传输的读写顺序见图 3－8，分为 3 个或 2 个阶段。最先的建立阶段由一个 SETUP 事务组成；中间是可选的数据阶段，由“total”除以“max”数量的 IN 或 OUT 事务构成。最后是一个 0 byte 的 OUT 或 IN 数据事务。

由协议知道，建立阶段的 toggle 必然是 0，最后一个状态阶段的 toggle 必须是 1。

中间数据阶段的 toggle 由 1/0 交互变化。

控制传输是使用的零终端（endpoint），可以传输 IN、OUT 和 SETUP 事务。

控制传输中的数据包的大小为 8，16，32 或 64 字节，一般取传输数据量最接近的值比较好。如果为了统一格式，可以取 64，这对那些传输数据不到 64 字节的情况可能有一些微微的浪费。

控制传输的流程图见图 4－2。(为了比较控制传输、后面用到的 BO 和 CBI 传输方式，这里把控制传输流程图放在后面一章)

下面是一个典型控制 OUT 传输的程序片断：

```
//SETUP stage，在零终端上生成一个 SETUP 事务
 new_make_ptd(cbuf,SETUP,0,64,0,addr,active_port,8);//SETUP 阶段的 PTD
 array_app(cbuf+4,stage1,4);//SETUP 阶段要发送的数据
 new_send_ptd(cbuf,rbuf,10,0);//通知寄存器，产生该传输事务
//DATA stage，在零终端上生成若干个 OUT 事务
if(length)
{ new_make_ptd(cbuf,OUT,0,64,1, addr,active_port,length);//传输数据长度为 length byte
 array_app(cbuf+4,stage2,length/2);
 new_send_ptd(cbuf,rbuf,4+length/2,0);
}
//STATUS stage，在零终端上生成一个 IN 事务
 new_make_ptd(cbuf,IN,0,64,1, addr,active_port,0);
   new_send_ptd(cbuf,rbuf,4,0);
```

3.6.3 批量传输

如图 3－9，批量传输由若干个同一个方向的 IN 或者 OUT 事务组成。

不同通道中的 toggle 是分开的，toggle 由 0/1 交互变化。如果在同一个通道进行传输，完成一次传输后需要保存当前 toggle 的值。举例说明，假设有一个批量 IN 终端 1，我们先在此终端上进行批量 IN 传输，以 DATA0 开始，以 DATA0 结束；此时，芯片切换到其它终端上进行操作；接下来，终端 1 又要开始一个批量 IN 传输，toggle 就应该是 DATA1，而不是 DATA0。这样，下面程序片断中表示批量 OUT 通道中的 tog_out 参数需要定义成一个静态变量。

批量传输中的数据包的大小为 8，16，32 或 64 字节，由于批量传输数据量比较大，一般取 64 比较好。传输数据拆成 64 字节的包，如果包的数量是个奇数，toggle 发生变化，如果是偶数，toggle 不变（中间的 toggle 变化由硬件完成，软件只设定 toggle

的初值）。当然，也可以在传输完成后，从 PTD 中读取现在的 toggle 值。

下面就是一个典型批量 OUT 程序了：

```
//在批量 OUT 终端上生成若干个 OUT 事务
new_make_ptd(cbuf,OUT,out_endpoint,64,tog_out,addr,active_port,length);
array_app(cbuf+4,data,length/2);//data 是要传输数据的指针
mycode=new_send_ptd(cbuf,rbuf,4+length/2,0);
//计算包的数量，看看 toggle 是否需要改变。
if (length==0){tog_out++;}
else ((((length－1)<<6)+1)&0x01){tog_out++;}//if(((length-1)/64+1)%2) {tog_out++;}
```

3.6.4 中断传输

中断传输同批量传输的事务格式差不多。只不过数据包的大小可以设定为 1 到 64 之间的任意值。

一个中断传输的往往有一个时间上的延迟，它的 PTD 格式与前面两种传输不同，ISP1362 使用专门的 INTL 缓冲区来管理中断传输。

在 PTD 的 Byte7 就是用来管理延时的的。Bit5－Bit7 表示循环等待指数（polling Rate），Bit0－Bit4 记录起始帧（Starting Frame）。Polling Rate＝N，那么它的等待时间就是 2 的 N 次方毫秒。USB 传输中，一毫秒等于一帧，所以该 PTD 构造的中断传输会在 Starting Frame＋2^N 毫秒后发出。

除了构建 PTD 时候多了一个延迟时间参数外，它的程序同批量传输一样，这里就不列出程序代码了。

3.7 USB 数据切换同步和重试

USB 提供了保证数据序列同步的机制，以确保数据传输的准确性。这一同步过程是通过 Data0 和 Data1 的 PID 以及发送器和接收器上数据触发序列（data toggle sequence bit）来实现。序列位的触发机理如下：

3.7.1 控制传输的同步

控制传送使用建立标记初始化主机和功能部件的时序位。如图3-10所示，主机向功能部件发送建立包，其后跟着输出事务。圆圈里的数代表发送器和接收器的时序位。功能部件必须接受数据并返回ACK。当功能部件接受事务的时候，它必须设置其时序位，以便主机和功能部件的时序位在建立事务的最后都等于1。

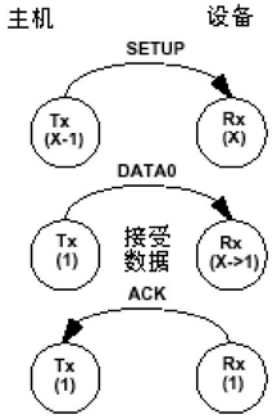


图 3-10　建立初始化

3.7.2　数据事务的同步

图3-11说明了成功的数据传输事务的情况。对于数据发送器，这表示它根据ACK的接收情况来切换其时序位。仅当接收器收到合法的的数据包，并且包的数据PID和其时序位的当前值相匹配的时候，才切换其时序位。发送器在它收到数据包的ACK时才切换其时序位。

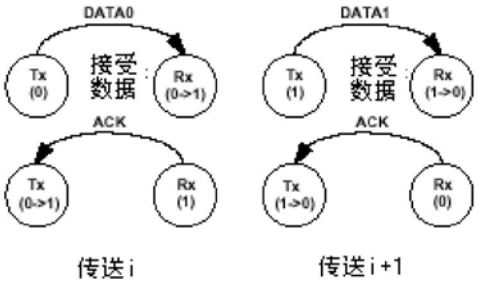


图 3-11　成功的连续传送

如果数据不能被接受，或者得到的数据包被损坏，接收器将根据情况发出 NAK 或 STALL 握手，或者超时（Timeout），并且，接收器将不切换其时序位。图 3-12 说明了事务被返回 NAK，然后被重试的情况。任何非 ACK 握手或是超时都将产生类似的重试动作。没有收到 ACK 握手的发送器，将不切换其时序位。其结果是失败的数据包事务使得发送器和接收器的时序位同步并不切换。然后事务将被重试，如果成功，将引发发送器和接收器时序位的切换。

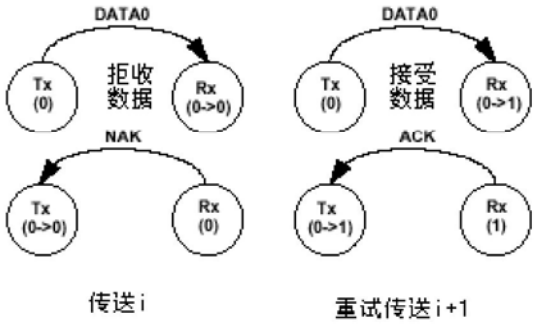


图 3-12　重试不被确认的事务

发送器是根据其收到ACK握手确切地知道事务是否成功的最后并且唯一代理。如图3-13所示，丢失或者损坏的ACK握手使得发送器和接收器之间的暂时失去同步。这里发送器在发出合法的数据包，且接收机成功地收到；但是ACK握手损坏。

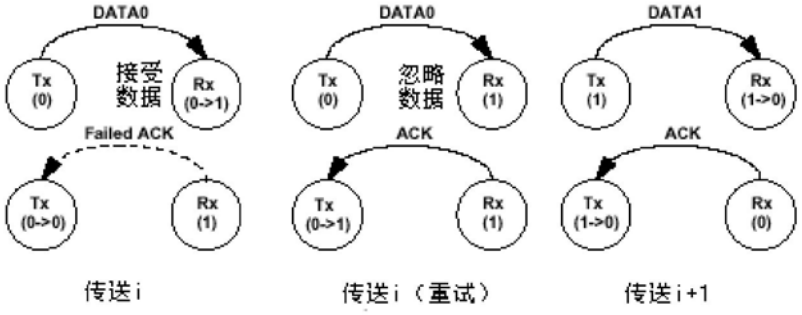


图 3-13　重试 ACK 损坏的事务

在事务i的最后，由它们各自的时序位间的失配可看出发送器和接收器暂时失去了同步。接收器已经收到了正确的数据，但是，发送器不知道它是否成功地发送了数据。在下一个事务中，发送器将重发使用DATA0 PID的先前的数据。接收器时序位和数据PID将不匹配，于是接收器知道它以前接受了这个数据。从而它丢弃此数据包且不切换其时序位。然后接收器发放ACK，使得发送器知道被重试的事务成功了。ACK的接收使得发送器切换其时序位。在事务i+1的开头，其时序位被切换，于是再一次同步了。

3.8　ISP1362 固件层的纠错

系统中的大部分错误都是偶然发生的，通过简单的重新发送该命令就可以纠正。系统的纠错程序，越靠近硬件越好[25]。在 USB 系统中，Slave 端和 Host 端相互独立，

Slave 和 Host 只需要纠正自身系统的错误就可以了。

ISP1362 PTD 的 Completing Code 域负责报错，硬件上对相应报错代码的处理如下表：

表 3-3 ISP1362 错误响应表

致命错误	错误代码	**IN** 事务	**OUT** 事务
CRC	01	No ACK send	Not applicable
BitStuffing	02	No ACK send	Not applicable
ToggleMismatch	03	ACK send	Not applicable
Stall	04	No ACK send	Host received Stall from device
NonResponding	05	No ACK send	Host not received a handshake reply within 18 Bit time，or a bad SYNC pluse
PIDCheckFail	06	No ACK send	Not applicable
UnexpectedPID	07	No ACK send	Corrupt NAK， ACK， or Stall
DataOverRun	08	N AK send	Not applicable
下面是唯一的非致命错误，不会导致传输事务中断，只是在 PTD 中给出警告。			
DataUnderRun	09	ACK send	Not applicable

错误代码 12、13 不会引起传输事务，所以不在表中。在 IN 事务中，ISP1362 要送出握手包，表项内容是 ISP1362 检测到错误后，硬件上的握手包自动处理。OUT 事务的表项内容是 ISP1362 接收到的握手包。

ISP1362 固件层的纠错，目的是保证软件系统中 PTD 参数与实际数据传输中相应参数的一致性。主要是防止意外错误后，USB 传输中的 toggle 不匹配，也就是上面 3.7 章节所说的数据切换同步和重试的问题。纠错程序是结合表 3－3 和 3.7 章节的内容来设计的。

3.8.1 控制传输检错与纠错

控制传输由 2 或 3 个传输阶段组成。由规范知道，在数据阶段和状态阶段的处理过程中，为了不必要的干扰，不管出错与否，它会 Stall 该通道，只有收到一个设置阶段的 PID，Stall 才会终止。所以在数据阶段和状态阶段是不可能纠错的，纠错只能从一开始的 SETUP 阶段开始。

而不论 SETUP 包合理与否，只要是一个设置阶段的 PID，系统就必须接收，所以设置阶段是不报错，报错只能在后面两个阶段进行。

由于控制传输中的每个阶段 toggle 都指定了初始值，所以发生错误后，toggle 不会出错。

结论就是：控制传输的纠错无法在事务（transaction）级完成，只能在传输（transfer）级进行。直接重新发送一次该控制传输就可以完成纠错处理。

3.8.2 数据事务检错与纠错

批量或中断传输是由若干个 IN 或者 OUT 数据事务组成。一个 PTD，可以产生多个传输事务。比如，PTD 可以设定 ISP1362 传输 512 byte 的数据。而批量包的大小一般是 64 byte，这就可以产生 8 个传输事务。如果出错，可能是其中任何一个事务的问题，所以批量传输的纠错必须重新发送该次批量传输，而不是仅仅重新发送一个传输事务。

每次数据事务的初始 toggle 是由软件指定的，传输过程中的 toggle 由硬件自动更新，如果出错后，toggle 可以是任意值。这里 ISP1362 硬件上对数据切换的同步已经做了相应的处理，我们只需要从 PTD 中读出正确 toggle 值就可以了。

在后面提到的 bulk only 传输方式中，传输的指令、数据、状态信息都是由批量传输发送。出错后，重新发送命令纠错的最小单元应该是一个 SCSI 指令，而不是一次简单的批量传输。

3.9 小结

本章从访问系统的数据总线开始；在访问总线基础上实现寄存器访问；通过寄存器访问完成 USB 基本传输事务；最后以 USB 基本传输事务组成了 USB 的 4 种基本传输。后面实现 USB 协议栈的各种函数，都是采用这 4 种基本传输制作。

目前大多数 USB Host 控制芯片在硬件上的作用是：通过设置寄存器参数，产生 USB 的基本传输事务。在此基础上，可以依照相关的协议设计相应的传输方式以及 Host 端驱动程序。为了不失一般性，文章淡化了 ISP1362 的具体寄存器功能，详细说明了 USB 传输事务和各种传输方式基本原理，并给出了我们系统中对应的程序实现。希望能给使用其它 USB Host 芯片的开发者提供一定的参考。

后面紧接着的 USB 基本驱动层和类设备驱动层与硬件无关，是 USB 协议栈的一般设计方法。为了移植方便，应该尽量在这一层提供比较合适的接口函数。这样，如果要更换其它硬件环境，软件系统只须重新封装这一层中的部分函数。

4 USB 主机端协议栈的设计

USB 主机端协议栈可以分为“USB 基本协议层”和“类设备驱动层”。“USB 基本协议层”主要是设计 11 个标准请求，列举设备，打开设备全部访问权限，等待相应子协议的进一步操作，这层针对所有的 USB 设备。“类设备驱动层”主要是驱动 USB IF（Implementers' Forum）所规定的具体某一类设备，我们系统主要针对海量存储类。

4.1 控制传输和标准请求

标准请求是 USB Host 获取设备基本信息的途径。USB 协议规定“所有的设备必须对 11 个标准请求做出响应，如果不支持标准请求对应的功能，则返回一个 STALL”[26]。这些标准请求是驱动所有 USB 设备的基础。标准请求都是使用控制传输发送，所有的数据都送到 Slave 设备的控制终端。

4.1.1 控制传输的要素

控制传输有 3 个阶段：

第一阶段：发送一个 **SETUP** 事务。

该事务的数据包长度是固定的：8 Byte。我们要发送的标准请求就填写在这 8 byte 中。

bmRequest	brequest	wValue	wIndex	wLength

该内容分为 5 个字段。如下所示

bmRequestType：用于指定数据传输方向和请求的类型。

Bit7 表示传输方向：0 是从 host 到 Slave，1 是从 Slave 到 host。

Bit5 和 Bit6 指示请教类型：00B 表示标准请求；01B 是特定的类请求，比如海量存储类的 reset 命令和 CBI 传输方式中控制传输都是使用 01B 值；10B 表示用户自定义请求；11B 保留。

brequest：请求的类型。如果是标准请求，这里用来区分 11 个标准请求。

wValue：来自于主机端的设置值信息。比如发送标准请求 Set_Address 命令，这里就是主机端给设备分配的地址。

wIndex：用来区分命令针对的对象。比如，很多命令是针对终端（Endpoint）的，这里区分对那个终端发送命令。

wLength：指示后面数据阶段发送数据的长度，如果为 0，表示没有数据包。

第二阶段：数据交换

依照第一阶段的指示，进行相应的数据交换。数据长度在 **wLength** 字段中已经指定了。它由若干个 IN 事务或者 OUT 事务组成。这个阶段可能不存在。

第三阶段：状态阶段

作用是报告整个传输成功和失败的地方。它是一个零长度的数据包，由一个 PID 和 CRC 域组成，所以有状态检验的作用。

如果存在第二阶段(数据交换)，那么在第二阶段就已经完成状态检验，这个阶段完全可以不用。比如我测试用的数码相机，用的 CBI 协议，它的控制命令就是使用类控制传输发送，它就不存在状态阶段(见 4.5.2 节)。但是在标准请求中，不论第二阶段是否存在，都必须有第三阶段。

4.1.2 11 种标准请求

下表总结了 USB 的 11 种标准请求。所有的设备必须对这些请求作出响应（虽然相应有可能只是 STALL）。这里只给出的请求基本功能的简单描述，详细内容可参考 USB 协议。

如果要设计主机端的 USB 驱动程序，只有包含这 11 个标准请求，才有可能驱动所有的 USB 设备。

表 4-1 USB 中 11 个标准请求功能简表

请求号	请求	目的
00h	Get_Status	获得设备、端口或终端的特性
01h	Clear_Feature	使能设备、端口或者终端，比如可引起设备的 stall 状态。
03h	Set_Feature	清除设备的使能特性。比如消除设备的 stall 状态等。
05h	Set_Address	主机指定以后与设备通讯的地址
06h	Get_Descriptor	获得设备描述符，有设备、配置、接口、终端和字符串 5 种类型的描述符，可获得设备使用的协议以及分类等基本信息。
07h	Set_Descriptor	更改设备的描述符。
08h	Get_Configuration	获得设备目前用的配置
09h	Set_Configuration	引导设备所使用的配置值
0Ah	Get_Interface	对于支持多接口的设备，主机请求当前的接口号
0Bh	Set_Interface	对于支持多接口的设备，主机请求某一个接口的设置
0Ch	Synch_Frame	设备设置和报告一个终端的同步帧

4.1.3 其它控制请求

除了标准请求外，USB 允许类的控制传输和用户自定义控制传输。只要使用规定的格式，然后在主机端和外设端给出相应的软硬件设定即可。

下图是标准请求SET_ADDRESS的数据组成图，图中可以清楚看出包（packet），传输事务（transaction），基本传输（tranfer）中控制传输的组成关系。

图 4-1 Set_Address 控制传输结构图

4.2 USB 设备的基本状态与列举设备

4.2.1 基本状态

USB 设备共有 6 种基本状态：上电、默认、地址、配置、连接以及挂起。

USB 对设备的不同状态做了一些物理参数和软件响应条件的规定，定义了相应的能力和行为。上面 11 种标准请求就必须在各自限定的状态下才可以执行。比如 Set_Configuration 只能在地址和配置状态下执行。如果在地址状态下执行，该命令执行成功后，设备进入配置状态。

这里面连接状态和挂起状态可以出现在任何时间。

如果 VBUS 不提供电源，设备就会处于连接状态。它可能出现在过载保护或者 USB 软件移去电源的情况下。

如果设备在总线上没有活动至少有 3ms，设备进入挂起状态。此时，设备必须消耗最少的总线电能。配置和未配置都必须支持挂起状态。

4.2.2 列举 USB 设备

列举是所有 USB 设备驱动的第一步，可以简单的看作是两个过程：第一、确认系统已经找到新插入的 USB 设备；第二、发送标准请求，让待驱动的设备处于 USB 配置状态，打开全部访问权限，等待相应子协议的进一步操作。

列举设备从设备状态上看，将经历状态中的：上电、默认、地址和配置。USB 设备插入计算机，自动进入上电状态；上电完成，进入默认状态；发送 Set_Address 命令，进入地址状态；最后通过 Set_Configuration 进入配置状态。只有配置状态，才可能打开 USB 设备的全部机能，接口才可以完全被使用。

上面只是列举设备的简单概述，一个完全的列举不是这样的，它要考虑支持许多复杂的 USB 设备，具体内容可以参考 USB 协议。

4.3 USB 基本协议层的设计

利用四种基本传输中的控制传输，就可以完成 USB Host 的基本驱动。这层主要是设计 11 个标准请求函数并在此基础上列举设备。

4.3.1 设计标准请求

外部设备不会使用全部标准请求，但是 Host 设备可能要驱动多种设备。11 个标准请求的程序量不大，所以最好一次性的完成 11 个标准请求的函数。

控制传输的程序在 3.6.2 节已经给出了，为了阅读方便，重新写出来。实现 11 个标准请求实际上就是要正确填写下面程序中的 stage1 和 stage2 数组以及 length 参数。这是一个控制写操作，Stage2 数据是从 host 传输到 slave。

```
//SETUP stage
 new_make_ptd(cbuf,SETUP,0,64,0,addr,active_port,8);//SETUP 阶段的 PTD
 array_app(cbuf+4,stage1,4);//SETUP 阶段要发送的数据
 new_send_ptd(cbuf,rbuf,8,0);//通知寄存器，产生该传输事务
//DATA stage
```

```
if(length)
{
 new_make_ptd(cbuf,OUT,0,64,1, addr,active_port,length);//传输数据长度为 length Byte
 array_app(cbuf+4,stage2,length/2);
 new_send_ptd(cbuf,rbuf,4+length/2,0);
}
//STATUS stage
 new_make_ptd(cbuf,IN,0,64,1, addr,active_port,0);
   new_send_ptd(cbuf,rbuf,4,0);
```

以实现标准请求中的 Set_configuration 命令为例（这个函数正好需要使用一个控制写传输）。

我们可以在 USB 规范中查到 Set_configuration 对应的二进制值，如下所示。

bmRequestType	Brequest	wValue	wIndex	wLength	Data
00h	09h	Config 值	0000h	0000h	None

依表所示，给无符号 16 位数组赋值：

```
U16 Stage1[4]={0x0900,0x0000,0x0000,0x0000};
Stage[1]= config;
```

传输中的 SETUP 阶段就是发送 Stage1 这个数组；表中的 wLenght 段为 0，说明 length=0，DATA 阶段不存在。Config 是配置设备的值，可以由 Get_Descriptor 命令读取设备描述符获得。

这只是 11 个标准请求中比较简单的一个命令，很多命令的复杂程度远远超过它。只要按照协议来，还是可以很快完成 11 个函数的编写。

4.3.2 列举设备

制作好标准请求，接着就是列举设备。列举可以简单的看作是两个过程：第一、确认系统已经找到新插入的 USB 设备；第二、发送标准请求，让待驱动的设备处于 USB 配置状态，等待相应子协议的进一步操作。

第一步，可以有两种方法来查找设备。通过查询 ISP1362 的 HcRhPortStatus1/2 寄存器来检测是否有设备插入，这种方法软件上与 ISP1362 硬件相关。如果用一个循环，不断的向外设发送标准请求，也可以查到是否有设备插入，这样的好处是做到软件与 ISP1362 无关。我们系统采用第一种方法。

第二步最少需要使用以下三个标准请求：首先通过 Get_Descriptor 命令获得设备描述符，通过查表获取设备基本信息，如配置设备的配置值，设备终端分配情况，设备所属的 USB 类等；然后用 Set_Address 命令给设备分配一个地址，让设备由默认状态进入地址状态；最后发送 Set_Configuration 命令给设备一个配置值，使设备进入配置状态。

在 USB 协议中，一个标准的 USB 列举共包含了 11 步，主要是针对比较复杂的设备来定义的。我们的系统主要是支持 USB 海量存储类的设备，这几步已经完全足够了，可以流畅而快速的列举并启动设备。

4.4 USB 设备的分类

USB 最大的特点之一就是可以支持各种各样的设备。由于设备的多样性，它们数据传输的方式，对应的传输协议必然也是各不相同。在嵌入式系统中，没有必要也不可能支持所有的 USB Slave 设备，我们只需要支持系统所要求支持的设备就可以。

嵌入式 USB Host 系统的控制固件都是针对特定的设备类型来编写。进行软件开发的第一步是识别我们要支持的 USB 设备的类型，设备类型初步决定了设备所使用的传输协议和控制命令集。[27]

USB 设备的类代码可以通过发送标准请求 Get_Descriptor 命令读取设备上的 USB 标准接口描述符（Standard Interface Descriptor）获得。描述符一般比较长，我们最关心的是的 bInterfaceClass、bInterfaceSubClass、bInterfaceProtocol 这三个字节的含意。

（1）bInterfaceClass：

这一个字节反映 USB 设备类型。由 USB IF 制订的 USB 设备常见分类以及类代码如下：

01h：音频类。主要是音频控制，音频流产生播放，MIDI 流产生播放等设备。

02h：通讯设备类。如 Modem，电话，以太网控制设备，ATM 控制设备等。

03h：人机交互类。如键盘，鼠标等[28]。

06h：图象类。如扫描仪器，数码相机等。

07h：打印机类。各种各样的单向和双向打印机。

08h：海量存储类。带存储功能的设备都可分到这一类，如数码相机也可归到这一类。

09h：USB Hub 类。

0Ah：数据接口（Data Interface）类。典型的设备是网络摄像头。（并没有规范化，

正在研究中。）

0Bh：Chip card/Smart card 类。

0FFh：用户自定义类。常见的有 USB 与 RS232 接口转换设备。

我们要支持的数码相机和 MO 在这一个字节上填写的值为 8，都是属于海量存储类。

（2）bInterfaceSubClass

这个字节反映的设备进一步分类的情况。对于海量存储设备，这里实际上反映了设备采用的工业标准命令协议，而不是设备的真正类别（如 USB 光驱、USB 硬盘等）。规定值如下[29]：

01h：Reduce Block Commands（RBC）

02h：SFF8020i

03h：QIC－157

04h：UFI

05h：SFF8070i

06h：SCSI

数码相机和 MO 分别为 05h 和 06h。

（3）bInterfaceProtocol

这个字节通常反映的是设备采用的协议，但是在海量存储类中，这里反映的是海量存储设备的传输方式[29]。共有 3 个值：

00h：CBI 传输（控制传输有状态阶段）。

01h：CBI 传输（控制传输没有状态阶段）。

50h：Bulk Only 传输方式。

试验用的数码相机是 01h，MO 为 50h。

4.5 海量存储类协议层的设计

市面上带有存储功能的 USB 数码设备，基本上都是采用 USB 海量存储类协议来设计的，如 U 盘、移动硬盘等。与这类设备相关的协议与规范主要有：规定传输方法的 Bulk Only（BO） 传输规范和 Control/Bulk/Interrupt(CBI) 传输规范；规定传输命令的 SCSI－II、SFF8070i 和 UFI 等命令协议。

我们系统中要控制的数码相机和 MO 都是使用的海量存储类的规范来设计。数码相机使用 CBI 传输规范和 SFF8070i 协议，MO 采用 BO 传输方式和 SCSI－II 协议，

正好覆盖 USB 海量存储类的大部分情况。

4.5.1 SCSI－II 命令协议与 SFF8070i 命令协议

这两个协议上都是工业标准规定的传输协议，它们针对接口透明，有时称为透明命令集（Transparent Command Set）。它们是为了控制各种计算机设备，而制订的一套二进制代码。不同的二进制组合，代表不同的控制命令。计算机的硬盘管理就是使用 SCSI 命令协议，只不过传输接口通常是 IDE 或 SCSI 接口。

从功能上看，这些透明命令集主要针对不同设备，比如 SCSI 主要针对大容量的存储设备，UFI 主要针对软驱一类的设备，而 SFF8070i 主要是针对即插即用的存储设备。不过也不是绝对，比如 SCSI 实际上也可以适用于即插即用设备，软驱也常常采用 SFF8070i，系统应该依照实际情况具体分析。

从历史上看，SFF8070i 和 UFI 都是在 SCSI 基础上发展而来，有时候可以看作是 SCSI 命令协议的一个子集。

我只看过 UFI、SCSI 和 SFF8070i 这 3 个协议[30] [31] [32]，它们的格式都差不多。包含的命令，如果是同一个命令，就会使用的同样的操作代码（Operation Code，位于数据包的第一个数据段），比如读数据块的 READ12 指令，操作代码统一为 0A8h；测试设备状态的 Test Ready 指令，操作代码统一为 00h。

这些协议的区别在于：第一、这些协议包含的指令数量不同。第二、如果是同一个指令，指令的长度、格式相同、响应方式虽然差不多，但是在某一个协议中，该指令中的保留位，可能在其它协议中是有定义的。保留位通常为 0，在其它协议中，如果该位有定义，那么 0 值会是该定义的默认情况。

我们系统主要采用 SCSI－II 命令，数码设备常用的指令见表 4-2。

我对表中命令的保留位统统设为 0。最后运行中测试，可以控制市面上大部分的 USB 移动存储设备。

表 4-2　常见 SCSI－II 控制命令表

命令名称（操作代码）	功能
TEST READY（00h）	检测设备是否准备就绪
INQUIRY（12h）	获取设备基本信息
PREVENT–ALLOW MEDIUM REMOVE (1Eh)	控制设备是否从系统中移除
READ10 (28h)	读取数据块，可读取 0－2^{16} 个数据块
READ12 (0A8h)	读取数据块，可读取 0－2^{32} 个数据块
WRITE10 (2Ah)	写入数据块，可写入 0－2^{16} 个数据块
WRITE12 (0AAh)	写入数据块，可写入 0－2^{32} 个数据块
READ CAPACITY (25h)	获取设备存储容量以及每扇区字节数
VERIFY (2Fh)	检测设备间数据传输是否正常
START-STOP UNIT (1Bh)	启动某些可以软件锁定的设备
REQUEST SENSE (03h)	获取设备 SENSE 代码，ASC，ASCQ，通过查表，可以获得设备现在的状态信息。
MODE SENSE (5Ah)	获得设备可选用的 SENSE 代码
MODE SELECT (55h)	选择 SENSE 代码

4.5.2　海量存储类的两种传输方式

USB 海量存储类设备选择一套透明命令集以后（以 SCSI－II 协议为例），接着就是使用 USB 特有的传输方式把命令送出。USB 海量存储类规定了两种传输方式，我们系统中正好两种都用到了。为了便于比较，这里一起给出这两种传输方式和前面提到的控制传输的流程图，图的每一步都是一个 USB 传输事务。

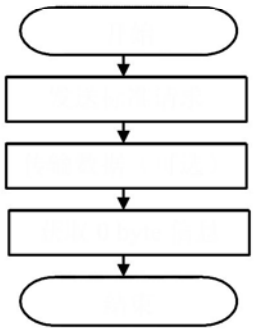


图 4-2 控制传输流程

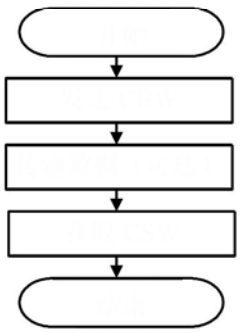


图 4-3　Bulk Only 传输流程

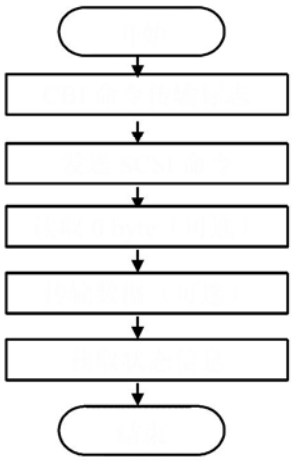


图 4-4　CBI 传输流程

(1) BO 传输方式

海量存储类用得最多的传输方式是 Bulk Only 传输，市面上的 U 盘、移动硬盘就是采用这种传输。图 4－3 是 Bulk Only 传输框图，它看起来与控制传输的框图差不多，也分成 3 个阶段。但是控制命令、数据和状态信息都是使用批量传输传送。[33]

因为各种类型的数据都是在批量终端中传输，为了区分批量终端中传输的数据类型，Bulk Only 协议规定控制命令和状态信息都必须进行打包处理。SCSI 等命令打包后为 CBW（Command Block Wrap），状态信息打包后为 CSW（Command Status Wrap）。具体的打包方法如下：

第一阶段，主机端发送 CBW，送出控制命令。CBW 打包的格式如图 4-5 所示。

	7	6	5	4	3	2	1	0
0－3	dCBWSignature(43425355)							
4－7	dCBWTag							
8－11	dCBWDataTransferLength							
12	bmCBWFlags(00/80)							
13	Reserved(0)				bCBWLUN			
14	Reserved(0)			bCBWCBLength				
15－31	CBWCB							

图 4-5 CBW 结构图

dCBWSignature: CBW 包的标志，值为 43425355h

dCBWTag：主机可以发送任何值，外设必须在 CSW 阶段把它完整无误的送回主机。

dCBWDataTransferLength：外设返回数据的长度。

bmCBWFlags：输入数据为 80,输出数据为 00h。

bCBWLUN：设备的逻辑编号。

bCBWCBLength: CBWCB 命令的长度。如果是 SCSI 命令，SCSI 命令长度有 6 byte、10byte 和 12 byte 三种。

CBWCB：需要外设执行的 CBWCB 命令，比如我们要使用的 SCSI 透明命令。

第二阶段，执行 SCSI 命令。命令执行完成后，就是设备报告数据传送状态的第三阶段 CSW，如图 4－6。

	7	6	5	4	3	2	1	0
0－3	dCSWSignature(53425355)							
4－7	dCSWTag							
8－11	dCSWDataResidue							
12	bCSWStatus							

图 4-6　CSW 结构图

dCSWSignature: CSW 包的标志，固定值为 53425355h

dCSWTag：主机可以发送的 dCBWTag 值，外设在 CSW 阶段把它完整无误的送回主机。

dCSWDataResidue: 用来比较要求传输长度和实际传输长度的差值

bCSWStatus：用来表示传输是否成功，如果成功，这个值为 00h

图 4-7 是用 BusHound 软件截取的 PC 机与实验用 MO 之间的 BO 传输数据图。

```
          (4) USB Root Hub
          (5) USB Root Hub
          (6) USB Root Hub
          (13) USB Storage Adapter V2 (TPP) [154KB/Sec]
          (14) FUJITSU MCF3064UA [ROM=0013] [154KB/Sec]
Phase:    Phase Type:
          CDB    Command descriptor block
          DI     Data in
          DO     Data out
          SNS    SCSI sense data
          SSTS   SCSI request block status
Data:     Hex dump of the data transferred
Info:     Description of the phase
Time:     Elapsed time leading up to the Phase
Cmd:      Unique identifier of the line
```

```
Device  Phase  Data                                                  Description
------  -----  ----------------------------------------------------  ----------------
  14    CDB    00 00 00 00 - 00 00                                   TEST UNIT RDY
  13.1  DO     55 53 42 43 - e4 7b 69 c1 - 00 00 00 00 - 00 00 06 00 USBC..i.........
               00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00    ................
  13.2  DI     55 53 42 53 - e4 7b 69 c1 - 00 00 00 00 - 01          USBS..i......
  13.1  DO     55 53 42 43 - e4 7b 69 c1 - 0e 00 00 00 - 80 00 0c 03 USBC..i.........
               00 00 00 0e - 00 00 00 00 - 00 00 00 00 - 00 00 00    ................
  13.2  DI     70 00 06 00 - 00 00 00 18 - 00 00 00 00 - 28 00       p...........(.
  13.2  DI     55 53 42 53 - e4 7b 69 c1 - 00 00 00 00 - 00          USBS..i......
  14    SNS    70 00 06 00 - 00 00 00 18 - 00 00 00 00 - 28 00       p...........(.
  14    SSTS   84                                                    check condition
  14    CDB    25 00 00 00 - 00 00 00 00 - 00 00                     READ CAPACITY
  13.1  DO     55 53 42 43 - e4 7b 69 c1 - 08 00 00 00 - 80 00 0a 25 USBC..i........%
               00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00    ................
  13.2  DI     00 04 bc 4f - 00 00 08 00                             ...O....
  13.2  DI     55 53 42 53 - e4 7b 69 c1 - 00 00 00 00 - 00          USBS..i......
  14    DI     00 04 bc 4f - 00 00 08 00                             ...O....
  14    SSTS   01                                                    ok
  14    CDB    28 00 00 00 - 00 00 00 00 - 01 00                     READ10
  13.1  DO     55 53 42 43 - e4 7b 69 c1 - 00 08 00 00 - 80 00 0a 28 USBC..i........(
               00 00 00 00 - 00 00 00 01 - 00 00 00 00 - 00 00 00    ................
  13.2  DI     eb 3e 90 2b - 5b 43 57 72 - 49 48 43 00 - 08 08 01 00 .>.+[CWrIHC.....
               02 00 02 00 - 00 f8 26 00 - 20 00 40 00 - 00 00 00 00 ......&. .@.....
               50 bc 04 00 - 80 00 29 68 - 35 e4 17 4e - 4f 20 4e 41 P.....)h5..NO NA
```

图 4-7　USB 接口 BO 传输数据图

(2) CBI 传输方式

另一种传输方法是 CBI 传输。控制命令由控制传输传送，数据由批量传输传送，状态信息使用中断传输传送[34]。由于使用了不同的传输方式，SCSI 命令不需要打包。

CBI 传输流程 4 如图 4－4 所示，从功能上看，也可以分成 3 个阶段。第一阶段（命令阶段，流程图的前三步）发送 SCSI 命令到设备控制终端，它实际上是一个类定义的控制传输，传输的数据是 SCSI 命令。仿照上面的 USB 标准请求表，CBI 传输的第一阶段如下所示：

bmRequestType	bRequest	wValue	wIndex	wLength	Data
21h	00h	0000h	0000h	000Ch	SCSI 命令

第二阶段，在批量传输通道上执行相关命令。

第三阶段（状态信息获取），通过中断传输返回两个字节的信息。第一字节值固定为 00h，表示中断传输完成。第二字节必须在第一字节为 00 的情况下，才有意义，含意如下：

Bit7－Bit4：用户自定义。

Bit3－Bit2：保留。

Bit1－Bit0：00b 成功，01b 失败，10b 阶段出错，11b 连续出错。

如果系统使用的是 UFI 透明命令集，中断传输的这两个字节含意与上面不同，它的第一个字节是 ASC，第二字节是 ASCQ。使用场景代码（Sense Code）报错。

图 4-8 是用 BusHound 软件截取的 PC 机与实验用数码相机之间的 CBI 传输数据图。

```
Bus Hound 3.02 capture. Complements of www.perisoft.net

plaste2

  Dev   - Device ID
  Time  - Elapsed time since the start of the previous Phase
  Phase - ADDR= 1394 transfer address  LOCK= 1394 lock transaction
          CDB = Command block          NSTS= NT status
          CTL = USB control packet     RSET= bus reset
          DI  = Data In                RSTS= I/O Request Status
          DO  = Data Out               SNS = SCSI Sense Data
          IDE = IDE task file command  SSTS= SCSI Request Block Status
          ISOC= Isochronous Transfer   USTS= USB status

  (0) Intel(R) 82801DB/DBM USB Universal Host Controller - 24C2
  (1) Intel(R) 82801DB/DBM USB Universal Host Controller - 24C4
  (2) Intel(R) 82801DB/DBM USB Universal Host Controller - 24C7
  (3) Intel USB 2.0 Enhanced Host Controller
  (4) USB Root Hub
  (5) USB Root Hub
  (6) USB Root Hub
  (11) FinePix Digital Camera 020715 [412KB/Sec]

Dev  Phase  Data         Info          Time   Cmd.Phase.Ofs
---  -----  -----------  ------------  -----  -----------------
 11  CTL    21 00 00 00  CLASS           0us         1.1.0
             00 00 0c 00                              1.1.4
 11  DO     00 00 00 00  ....          4.7ms         1.2.0    //test ready
             00 00 00 00  ....                        1.2.4
             00 00 00 00  ....                        1.2.8
 11  CTL    21 00 00 00  CLASS          20us         2.1.0
             00 00 0c 00                              2.1.4
 11  DO     03 00 00 00  ....          4.9ms         2.2.0    //request sense
             12 00 00 00  ....                        2.2.4
             00 00 00 00  ....                        2.2.8
 11  DI     70 00 00 00  p...          1.9ms         3.1.0
             00 00 00 0a  ....                        3.1.4
             00 00 00 00  ....                        3.1.8
             00 00 00 00  ....                        3.1.12
 11  CTL    21 00 00 00  CLASS         1.4sc         4.1.0
             00 00 0c 00                              4.1.4
 11  DO     00 00 00 00  ....          4.8ms         4.2.0
```

图 4-8　USB 接口 CBI 传输数据图

4.6　系统纠错功能的设计

ISP1362 固件层的纠错目的是保证 PTD 参数与数据传输的一致性，主要是防止意外错误后，USB 传输中的 toggle 不匹配。由于大部分的错误都是偶然发生，一般重新发送一次命令，错误就可以解决了。在 USB 海量存储类中，命令的发出、数据的响应、系统状态报告都是在海量存储规范中规定的，所以命令重发来进行系统纠错应该

集中在 USB 海量存储类协议层，而不是在上面的 ISP1362 固件层。

我们系统中，处理错误的方法由下到上主要为：类的 reset 处理、recover 处理、sense code 处理以及 USB 传输重启。

（1） 类的 Reset

上面两种传输规范中都规定了该传输的类 reset 命令，主要作用是让 Slave 设备停止现在的传输，放弃当前的命令，清除异常报警状态。

bulk only 传输中 reset 命令如下，是一个类定义的控制传输命令。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
21h	FFh	0000h	Interface	0000h	None

CBI 控制传输的数据阶段一般是 SCSI 命令，也有可能是类的 Reset 命令。CBI 类的 Reset 命令为“1Dh 04h FFh FFh FFh FFh”。

类的 Reset 命令没有对通道 Stall 以及 toggle 的变化做要求，所以纠错中用得最多的是下面的 Recover 纠错。

（2） Recovery 纠错

类 Reset 处理后，紧接着发送标准请求 Clear Feature 命令，清除批量终端的挂起状态，并让设备的 toggle 置零，这一过程在规范中称为 recovery 处理。

当 Slave 设备接收到一个错误的 SCSI 命令后，无论是批量 IN 还是批量 OUT，都无法主动的向 Host 报错。它的处理办法就是挂起批量终端，阻止数据传输的进行。如果 Host 发现与批量终端进行通讯的时候，总是返回一个 STALL，就需要进行 recovery 处理。

一个 recovery（类 reset + Clear Feature）操作，可以强制启动 MO 和数码相机等 Slave 设备软件系统中的纠错功能，并且让系统的 toggle 重置，可以说是条件非常强的纠错命令。

（3） Sense Code 处理

这层实际上是属于 SCSI、UFI 等透明命令集中的纠错系统，已经脱离 USB 协议的范围。

利用 SCSI 协议中的 request sense 命令可以获得获得设备的 ASC（Additional Sense Code）代码和 ASCQ（Additional Sense Code Qualifier）代码，通过查表可以获得设备的具体状态和错误的具体原因，然后调用相应的 Mode Sense、Select Sense 等 Sense 处理命令处理错误。ASCQ 和 ASC 大小都是一个字节，二者联合可以报出 65536 种

错误，但是一套协议中只规定了大约 40 多种错误。

可以看出，这套报错、纠错系统可以说是非常复杂，主要用在对稳定性要求比较高的设备中。移动数码设备基本不用它，只是偶尔使用 request sense 来启动设备和简单的报告状态。由于 USB 传输在有一套自己完善的纠错系统，所以 Sense 纠错主要是一个辅助作用。使用前面的 Recover 纠错，就可以纠正大部分错误了。

（4） 电气规则重起

如果上述方法都无法纠错，可以试试 USB 传输重启。向 ISP1362 的 HcReset 寄存器写入 0x00F6，会实现 USB 电气规则上的重启操作。接着就是重新列举设备，继续上次的传输。

这是软件纠错的最后防线，如果还无法纠错，就只能在硬件上进行系统重起。

4.7 小结

任何一个 USB 协议栈，从逻辑上都可分为两层。一层是 USB 基本协议层，它是针对所有 USB 设备的；另外一层是针对相应的 USB 类来设定的。

如果设计 Slave 端的协议栈，只需要按照 Slave 设备的类型来设计即可。设计 Host 端的协议栈，理论上是要支持所有的 USB 设备类。但是考虑到嵌入式系统资源比较有限，一般都是只使用要支持设备的类协议层，我们系统主要是支持 USB 海量存储类的设备。

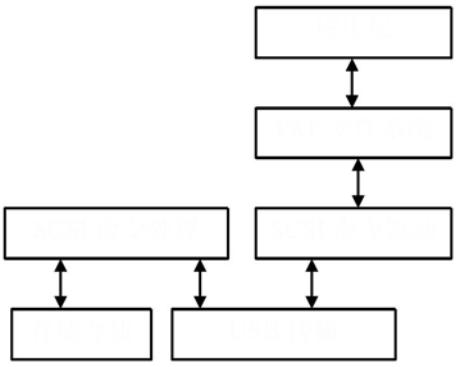
从程序设计上看，这两层应该是一个平行的关系，都是使用 USB 的 4 种基本传输完成，并没有上下级关系。但是 USB 基本协议层的列举是驱动的第一步，类驱动层偶尔也会用到 11 个标准请求，所以不妨认为类设备驱动层是在基本驱动层的基础上完成的驱动层。

USB Host 端协议栈的作用是：对各种要通过 USB 接口传输的数据进行正确的打包拆包。该协议栈的主要任务有：判断设备类型，判断设备使用透明命令集以及传输方式，然后调用相应的处理程序，用正确的传输方式把相应的命令传输出去。在下面的图 5－1 中可以清晰地看出 USB 海量存储类协议栈的在整个控制系统中作用。

5 简易 FAT 文件管理系统的设计

数据通过 USB Host 和 USB Slave 协议栈后，做到了 Host 设备和 Slave 设备的 SCSI 等透明命令集的透明。此时整个系统的框图如图 5－1 所示，如果把 USB 接口换成 IDE 接口，这实际上是一个计算机内硬盘的管理系统的框图。

要管理海量存储类设备，必然涉及文件管理的问题。目前数码设备一般使用 FAT 文件格式来组织存储空间，在我们系统中也就要开发一个简单的 FAT 文件管理系统来进行设备的存储空间管理。



5.1 存储设备的分区

从操作系统的观点上看，存储设备是块设备。SCSI 等协议中有许多块设备使用的命令，使用的存储基本单位是块（Block）。一个块设备上可以有多个文件系统，如果按 FAT 结构来组织存储设备，块大小等于扇区（Sector）。

FAT 存储设备的零扇区的最初 512 字节，通常为 MBR 区，它实际包含三部分，MBR 只是其中之一：

（1）MBR（Main Boot Recorder）

占用 446 字节，存放系统主引导程序和引导参数。其中的引导程序主要作用是：检查 DPT 表是不是正确；如果系统硬件自检通过后，激活标志分区上的操作系统，并把控制权限交给启动程序。MBR 可以由 Fdisk.com 等命令产生，不依赖于任何操作系统。磁盘引导程序是可以修改的，这样就可以实现多个操作系统以及文件系统的并

存。[35]

（2）DPT（Disk Partition Table）

占用 64 字节，记录磁盘分区基本信息。DPT 可以标识 4 个分区，每个分区的 16 字节信息如表 5－1。如果分区数大于 4，需要使用扩展分区的格式。

表 5-1　DPT 分区信息结构

偏移	长度	所表达的意义
0	字节	分区状态：0＝非活动分区，80＝活动分区
1	字节	该分区起始头
2	字	该分区起始扇区和起始柱面
4	字节	该分区类型：如 82h＝linux native 分区，83h＝linux swap 分区，0Bh＝FAT32 分区，07h＝NTFS,0Ch=主分区，0Fh＝扩展分区
5	字节	该分区终止头
6	字	该分区终止扇区和终止柱面
8	双字	该分区起始绝对扇区
C	双字	该分区占用的扇区数

（3）Boot Record ID

最后两字节，固定为 0x55AA，用来判断引导区是否合法。

如果使用扩展分区，那么 DPT 分区信息最多只能有两项，第一项是本分区的逻辑分区信息，第二项指出扩展分区所在位置和大小。两种标识磁盘结构方式如图 5－2 和 5－3 所示。

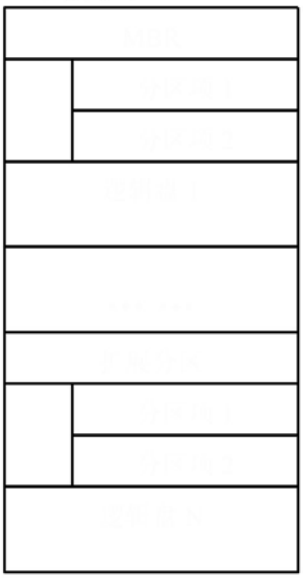


图 5-2 扩展分区标识磁

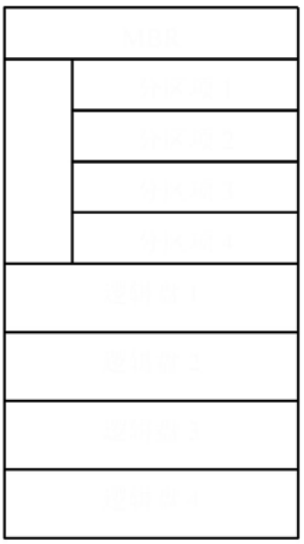


图 5-3 主分区标识磁盘

图 5－6 是实验中获取的 U 盘 MBR 区

分区格式的识别是由操作系统来完成。所以如果使用支持扩展分区的操作系统，那么就算是只有 2 个逻辑分区，同样可以使用扩展分区来组织存储设备。

```
00000000h: FA 33 C0 8E D0 BC 00 7C 8B F4 50 07 50 1F FB FC ; 3砍盾.|嫻P.P.
00000010h: BF 00 06 B9 00 01 F2 A5 EA 1D 06 00 00 BE BE 07 ; ?.?.顓?...揪.
00000020h: B3 04 80 3C 80 74 0E 80 3C 00 75 1C 83 C6 10 FE ; ?€<€t.€<.u.尷.?
00000030h: CB 75 EF CD 18 8B 14 8B 4C 02 8B EE 83 C6 10 FE ; 摹镜.?姨.媧尷.?
00000040h: CB 74 1A 80 3C 00 74 F4 BE 8B 06 AC 3C 00 74 0B ; 蔚.€<.t艚??.t.
00000050h: 56 BB 07 00 B4 0E CD 10 5E EB F0 EB FE BF 05 00 ; V?.??^腽膻?.
00000060h: BB 00 7C B8 01 02 57 CD 13 5F 73 0C 33 C0 CD 13 ; ?|?.W? s.3劳.
00000070h: 4F 75 ED BE A3 06 EB D3 BE C2 06 BF FE 7D 81 3D ; Ou阽?膝韭.傀}?
00000080h: 55 AA 75 C7 8B F5 EA 00 7C 00 00 49 6E 76 61 6C ; U濮蒜蹶.|..Inval
00000090h: 69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61 62 ; id partition tab
000000a0h: 6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67 ; le.Error loading
000000b0h: 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65 ;  operating syste
000000c0h: 6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61 74 ; m.Missing operat
000000d0h: 69 6E 67 20 73 79 73 74 65 6D 00 00 00 00 00 00 ; ing system......
000000e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000100h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000110h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000150h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000160h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 01 ; ..............€.
000001c0h: 01 00 06 11 E0 CE 20 00 00 00 A0 F3 01 00 00 00 ; ....嚣 ...昊....
000001d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA ; ..............U□
```

图 5-6　一个典型的 U 盘 MBR 区

5.2　FAT 文件系统

5.2.1　FAT 文件系统结构

不是所有的存储设备都必然有 MBR 区。如 MO 盘片的零扇区就直接是 FAT 文件系统的保留区。

FAT 文件系统的分区应该有这几个部分：保留区（Reserved Region），存放 FAT 的文件系统的重要参数和引导程序；FAT 表 （FAT Region），记录簇（Cluster）的使用情况；根目录区（Root Directory Region），记录根目录信息，FAT32 系统舍弃了这个区，根目录区可以任意指定一个簇；文件目录数据区（File and Directory Data

Region），是各种文件数据的实际存放区域[36] [37]。图 5－4 和图 5－5 就是我们系统中数码相机和 MO 盘片的组织结构图。

5.2.2 保留区以及 BPB 参数

保留区的第一个扇区是 DBR（DOS Boot Recorder）区，FAT32 常常称这个区域为引导扇区（Boot Sector）。它是操作系统可以直接访问的第一个扇区，主要包括与引导操作系统有关的引导参数和引导程序，还有就是与 FAT 文件系统有关的 BPB 参数(Bios Parameter Block)。与引导操作系统有关的参数都用“BS_”作为前缀，主要是为微软的操作系统保留的，编写自己的文件管理系统，只需要关心 BPB 参数即可。BPB 参数都采用“BPB_”作为前缀，它主要记录着 FAT 各个部分的起始扇区以及占用扇区的数目、根目录大小、簇大小等重要信息。BPB 参数如表 5－2 所示：

表 5-2 BPB 表

名称	偏移	字节	所表达的意义
BPB_BytePerSec	11 (0Bh)	2	每个扇区的字节数，只能为：512，1024，2048，和 4096。
BPB_SecPerClus	13 (0Dh)		每一簇的扇区数。必须是 2 的整数次方，最大值为 128，但是每一簇的大小必须小于 32k 字节。
BPB_RsvdSecCnt	14 (0Eh)	2	保留区的保留扇区数，FAT12，FAT16 必须为 1，FAT32 为 32。
BPB_NumFATs	16 (10h)	1	FAT 结构表的份数，必须为 2。
BPB_RootEntCnt	17 (11h)	2	根目录的项数，。FAT12 和 FAT16 的根目录下一般可以保存 512 个文件，所以多数情况下为 512。FAT32 这里设为 0
BPB_TotSec16	19 (13h)	2	整个 FAT 分区的扇区总数。如果小于 0x10000，使用这个区域；如果大于这个值，设为零，使用 BPB_TotSec32 记录。FAT32，这个值必须为 0。
BPB_Media	21 (15h)	2	存储介质的类型：固定介质一般为 0xF8，移动介质一般使用 0xF0。与 FAT 表的 FAT[0]的低 8 位保存一致。
BPB_FATSz16	22 (16h)	1	FAT 表占用的扇区数
BPB_SecPerTrk	24 (18h)	2	每道（Track）的扇区数。
BPB_NumHeads	26 (1Ah)	2	磁头数。
BPB_HiddeSec	28 (1Ch)	2	FAT 分区表前面的隐藏扇区数。对于非分区的介质，可以为零，与操作系统有关。
BPB_TotSec32	32 (20h)	4	FAT 分区总的扇区数。

续表 5-2

以下是针对 FAT32 的信息			
BPB_FATSz32	36 (24h)	4	FAT32 的 FAT 表的大小（在 BPB_FATSz16 为 0 的时候，这 4 位才有效）。
BPB_ExtFlags	40 (28h)	2	扩展标志。
BPB_FSVer	42 (2Ah)	2	FAT32 系统的版本号。高 8 为表示主版本号，低 8 位表示本版本下修改的升级号
BPB_RootClus	44 (2Ch)	4	FAT32 的根目录所在的扇区号。
BPB_FSInfo	48 (30h)	2	保留区的文件信息区所占用的扇区号，通常为 1
BPB_BkBootSec	50 (32h)	2	保留区引导扇区备份所在的扇区号。通常为 6
BPB_Reserved	52 (34h)	12	保留为以后扩展 FAT 系统版本用，设为 0 即可。

图 5-7 是实验中 U 盘 DBR 区以及紧跟着的 FAT 表第一扇区部分（200h 以后）。

```
00000000h: EB 3E 90 2B 5B 5D 7A 3A 49 48 43 00 02 04 01 00 ; ?>?[]z:IHC.....
00000010h: 02 00 02 00 00 F8 7D 00 20 00 40 00 20 00 00 00 ; .....鴠. .@. ...
00000020h: A0 F3 01 00 80 00 29 6B 2E E3 10 4E 4F 20 4E 41 ; 昊..€.)k.?NO NA
00000030h: 4D 45 20 20 20 20 46 41 54 31 36 20 20 20 F1 7D ; ME    FAT16   駸
00000040h: FA 33 C9 8E D1 BC FC 7B 16 07 BD 78 00 C5 76 00 ; ?庵鸭龜..绎.舷.
00000050h: 1E 56 16 55 BF 22 05 89 7E 00 89 4E 02 B1 0B FC ; .V.U?.塘.墙.??
00000060h: F3 A4 06 1F BD 00 7C C6 45 FE 0F 8B 46 18 88 45 ; 蜊..?|鼬?婵.圇
00000070h: F9 FB 38 66 24 7C 04 CD 13 72 3C 8A 46 10 98 F7 ;   8f$|.?r<窅.橑
00000080h: 66 16 03 46 1C 13 56 1E 03 46 0E 13 D1 50 52 89 ; f..F..V..F..裉R?
00000090h: 46 FC 89 56 FE B8 20 00 8B 76 11 F7 E6 8B 5E 0B ; F鼓V   .燊.麈嫦.
000000a0h: 03 C3 48 F7 F3 01 46 FC 11 4E FE 5A 58 BB 00 07 ; .肏黧.F?N吶X?.
000000b0h: 8B FB B1 01 E8 94 00 72 47 38 2D 74 19 B1 0B 56 ; 嬷?獺.rG8-t.?V
000000c0h: 8B 76 3E F3 A6 5E 74 4A 4E 74 0B 03 F9 83 C7 15 ; 燊>螃^tJNt..鹟?
000000d0h: 3B FB 72 E5 EB D7 2B C9 B8 D8 7D 87 46 3E 3C D8 ; ;鹟咫?筛貊哮><?
000000e0h: 75 99 BE 80 7D AC 98 03 F0 AC 84 C0 74 17 3C FF ; u檐€)瑯.熘钭t.<
000000f0h: 74 09 B4 0E BB 07 00 CD 10 EB EE BE 83 7D EB E5 ; t.??.?脯縉)脲
00000100h: BE 81 7D EB E0 33 C0 CD 16 5E 1F 8F 04 8F 44 02 ; 缨)脉3劳.^.?麂.
00000110h: CD 19 BE 82 7D 8B 7D 0F 83 FF 02 72 C8 8B C7 48 ; ?緣)嵝.?.r荚茸
00000120h: 48 8A 4E 0D F7 E1 03 46 FC 13 56 FE BB 00 07 53 ; H翰.麽.F?V  ..S
00000130h: B1 04 E8 16 00 5B 72 C8 81 3F 4D 5A 75 A7 81 BF ; ??.[r商?MZu  ?
00000140h: 00 02 42 4A 75 9F EA 00 02 70 00 50 52 51 91 92 ; ..BJu熘..p.PRQ憭
00000150h: 33 D2 F7 76 18 91 F7 76 18 42 87 CA F7 76 1A 8A ; 3吟v.巴v.B嘱鼸.?
00000160h: F2 8A 56 24 8A E8 D0 CC D0 CC 0A CC B8 01 02 CD ; 驃V$婀刑刑.谈..?
00000170h: 13 59 5A 58 72 09 40 75 01 42 03 5E 0B E2 CC C3 ; .YZXr.@u.B.^.馓?
00000180h: 03 18 01 27 0D 0A 49 6E 76 61 6C 69 64 20 73 79 ; ...'..Invalid sy
00000190h: 73 74 65 6D 20 64 69 73 6B FF 0D 0A 44 69 73 6B ; stem disk  ..Disk
000001a0h: 20 49 2F 4F 20 65 72 72 6F 72 FF 0D 0A 52 65 70 ;  I/O error  ..Rep
000001b0h: 6C 61 63 65 20 74 68 65 20 64 69 73 6B 2C 20 61 ; lace the disk, a
000001c0h: 6E 64 20 74 68 65 6E 20 70 72 65 73 73 20 61 6E ; nd then press an
000001d0h: 79 20 6B 65 79 0D 0A 00 49 4F 20 20 20 20 20 20 ; y key...IO
000001e0h: 53 59 53 4D 53 44 4F 53 20 20 20 53 59 53 80 01 ; SYSMSDOS   SYS€.
000001f0h: 00 57 49 4E 42 4F 4F 54 20 53 59 53 00 00 55 AA ; .WINBOOT SYS..U?
00000200h: F8 FF FF FF FF FF FF FF FF 06 00 07 00 08 00 ; ?
00000210h: 09 00 0A 00 0B 00 0C 00 FF FF 00 00 00 00 00 00 ; ........    .....
00000220h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000230h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000240h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000250h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
```

图 5-7　U 盘 DBR 区以及部分 FAT 表

5.2.3 FAT 表

FAT（File Allocation Table）表的作用是记录磁盘上簇的分配情况，在条件允许的情况下，通常都会把 FAT 表全部读到内存中去。在 FAT 文件系统中，存储文件的最小单位是簇，它实际大小等于 BPB_BytePerSec 和 BPB_SecPerClus 的乘积。簇如果太大，存储小文件会浪费大量的存储空间，如果太小，会让 FAT 表变大，不方便管理。

所谓 FAT12，FAT16，FAT32 是指 FAT 表中标识每一簇占用的位数。FAT12 是 12 位表示一个簇，最多表示 4k 簇，FAT16 为 64k 簇，FAT32 为 4G 簇。

FAT 表各种值的常见意义如表 5－3。

表 5-3 FAT 表的常见意义

意义	FAT12 代码	FAT16 代码	FAT32 代码
空簇	0x000	0x0000	0x00000000
文件的下一簇	0x002－0xFEF	0x0002－0xFFEF	0x00000002－0xFFFFFF0E
文件最后一簇	0xFFF	0xFFFF	0xFFFFFF0F
坏簇	0xFF7	0xFFF7	0x00000001

一个文件往往需要占用很多个簇。同一个文件不一定会完整地存放在一个连续的存储空间内，而是分成若干段，像链子一样地存放。在文件名记录项中，指向文件存储的链头所在的 FAT 簇，而该簇的 FAT 表存放下一个链子的 FAT 簇值，如果是文件结尾，使用一个文件结束标志，表示到达链尾，这样就标识了文件的链式存储。

5.2.4 文件以及目录的表达方式

根目录区和数据区的子目录簇记录着文件名以及目录名。标准的文件名记录项如表 5－4 所示。

可以看出，目录其实也是一个文件，只不过文件内容都是文件记录项信息。目录簇的前两个文件记录项，必须是“.”和“..”文件（根目录除外），作用是分别记录当前目录所在簇和上一级目录所在簇。

上面是一个标准的 8.3 文件名记录方法，现在 Window 通常采用的是 VFAT（保护模式 FAT），最大特点是支持长文件名。长文件名的文件记录项生成法则如下：

字母前缀＋连接符号“~”+数字域“序号”＋符号“.”＋文件扩展名

截取长文件名的前 6 位做为字母前缀（如果是小写字母，要转换成大写字母），字母前缀相同文件名的个数加 1 作为数字域“序号”，如果序号大于 9，则取前 5 位作为文件名前缀。这样就保证了长文件名的文件记录项仍然是 8.3 格式。而实际的文件名保

存在该文件记录项的前面，该项使用 01，02 等序列号开头，用来区别真正的文件记录项。比如：abcdefgh00.txt 和 abcdefgh11.txt 在同一个目录下的记录，在 DOS 下则显示为 ABCDEF～1.txt 和 ABCDEF～2.txt。图 5－8 是实验中获取的 U 盘根目录区部分内容。

表 5-4　文件记录项

<table>
<tr><td>名称</td><td>偏移</td><td>字节</td><td colspan="8">所表达的意义</td></tr>
<tr><td>DIR_Name1</td><td>0</td><td>8</td><td colspan="8">文件的名称。首字母有几个特殊的定义如下：
00H：该目录项未使用。
05H：当文件名首字母为 E5H 时，由于 E5H 有特殊意义，所以 E5H 必须换成 05H
E5H：一个删除的文件名
2EH：本项为目录</td></tr>
<tr><td>DIR_Name2</td><td>8</td><td>3</td><td colspan="8">文件扩展名</td></tr>
<tr><td rowspan="3">DIR_Attr</td><td rowspan="3">11</td><td rowspan="3">1</td><td colspan="8">文件的属性。每一 bit 对应属性列表如下。</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>保留</td><td>保留</td><td>存档</td><td>目录</td><td>卷标</td><td>系统</td><td>隐藏</td><td>只读</td></tr>
<tr><td>DIR_NTRes</td><td>12</td><td>1</td><td colspan="8">FAT32 中使用长文件名长目录名时的校验和</td></tr>
<tr><td>DIR_CrtTimeTenth</td><td>13</td><td>1</td><td colspan="8">文件建立毫秒时间</td></tr>
<tr><td>DIR_CrtTime</td><td>14</td><td>2</td><td colspan="8">文件建立时间</td></tr>
<tr><td>DIR_CrtDate</td><td>16</td><td>2</td><td colspan="8">文件建立日前</td></tr>
<tr><td>DIR_LstAccData</td><td>18</td><td>2</td><td colspan="8">文件最后访问（读写）的日期。</td></tr>
<tr><td>DIR_FstClusHI</td><td>20</td><td>2</td><td colspan="8">起始簇的高 16 位（FAT32 才有效）</td></tr>
<tr><td>DIR_WrtTime</td><td>22</td><td>2</td><td colspan="8">文件最后的写时间。</td></tr>
<tr><td>DIR_WrtData</td><td>24</td><td>2</td><td colspan="8">文件最后的写日期。</td></tr>
<tr><td>DIR_FstClusLO</td><td>26</td><td>2</td><td colspan="8">起始簇的低 16 位</td></tr>
<tr><td>DIR_FileSize</td><td>28</td><td>4</td><td colspan="8">文件的大小。目录这一项设为 0</td></tr>
</table>

```
00000000h: E5 B0 65 FA 5E 87 65 F6 4E 39 59 0F 00 75 00 00 ; 瀬獠鹉嗼鲅9Y..u..
00000010h: FF FF FF FF FF FF FF FF FF FF 00 00 FF FF FF FF ;
00000020h: E5 C2 BD A8 CE C4 7E 31 20 20 20 10 00 75 31 52 ; 迓建文~1   ..u1R
00000030h: A4 32 A4 32 00 00 32 52 A4 32 03 00 00 00 00 00 ; ??..2R?......
00000040h: 41 42 43 30 30 31 20 20 20 20 20 10 08 75 31 52 ; ABC001      ..u1R
00000050h: A4 32 A4 32 00 00 32 52 A4 32 03 00 00 00 00 00 ; ??..2R?......
00000060h: E5 B0 65 FA 5E 87 65 F6 4E 39 59 0F 00 75 00 00 ; 瀬e鹉嗼鲅9Y..u..
00000070h: FF FF FF FF FF FF FF FF FF FF 00 00 FF FF FF FF ;
00000080h: E5 C2 BD A8 CE C4 7E 31 20 20 20 10 00 A2 36 52 ; 迓建文~1   ..?R
00000090h: A4 32 A4 32 00 00 37 52 A4 32 04 00 00 00 00 00 ; ??..7R?......
000000a0h: 41 41 41 41 41 20 20 20 20 20 20 10 08 A2 36 52 ; AAAAA       ..?R
000000b0h: A4 32 A4 32 00 00 37 52 A4 32 04 00 00 00 00 00 ; ??..7R?......
```

图 5-8　U 盘根目录区部分内容

5.3 SCSI 命令实现文件管理

我们系统初步采用一键传输的方式。当 MO 和数码相机插入样机后，点击传输按钮，系统会自动找到并驱动两个设备；然后读取数码相机上的全部有用数据，传输到在我们在 MO 中创建的目录下面。创建的目录放在 MO 的根目录下，编号为 PICXXX，“XXX”按照顺序自动排列。这里面只有两个简单的文件操作，创建目录和拷贝文件。

实现文件管理前首先要确定存储设备一个扇区的大小。这时候由于不知道 BPB 的位置，所以是发送 SCSI 命令中的 Read Capacity 命令来确定的。这条命令不但可以返回设备的扇区总数，而且附带有每个扇区占用多少字节。

SCSI 命令中与文件系统关系最大的是的 read10，read12，write10 和 write12 这 4 条数据块读写命令，它们的作用是在存储设备上读写规定地址的扇区。所有的文件操作都可以利用这 4 条命令来实现。

如果仅仅是使用一键传输，只需要调用相关的函数就可以了。如果要添加键盘，让用户可以自己创建目录，拷贝文件，删除文件等，一个简易文件管理系统基本流程如图 5－9。

第一步读取 MBR 区，主要是读 DPT 表，了解存储设备分区情况。比如一个 U 盘的 DPT 可能如下：

80 01 01 00 01 01 20 4E **20 00 00 00** DF 13 00 00（后面全部为 00）

对照上面的表 1，我们可以知道，这个 U 盘只有一个分区。分区起始绝对地址为 0x20 扇区，我们在这个地址上可以查到 BPB 信息。

第二步，读取 BPB 信息，确定 FAT 文件系统的各种基本参数。

第三步，显示当前目录位置，等待键盘等 HID 设备的输入。如果输入的是一个可被系统执行的命令 ，执行相应的操作即可。

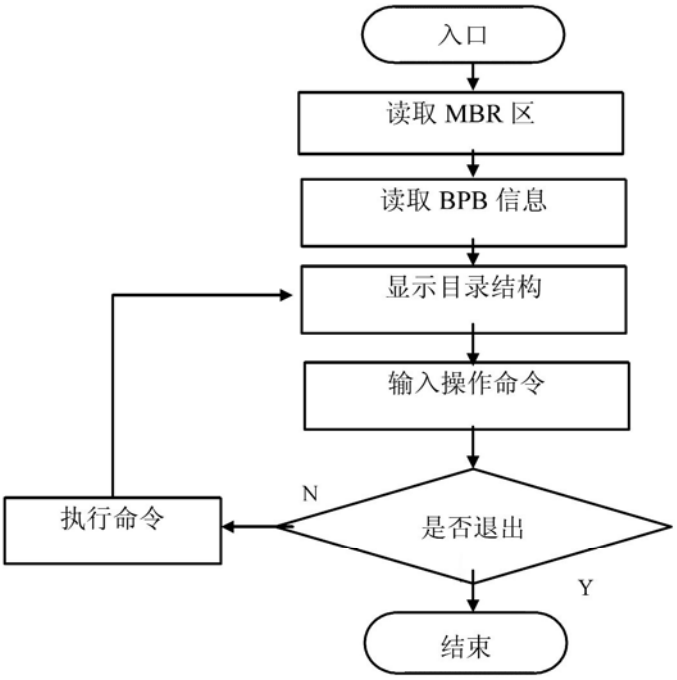


图 5-9　FAT 文件管理系统流程

总体来说，文件系统的实现，就是在熟悉 FAT 协议后，把对应的文件目录操作拆成相应的扇区读写命令。比如一个创建目录的操作实现过程如下：

扫描 FAT 表，找到一个空簇→在空簇对应的扇区生成“.”文件和“..”文件→在当前目录下生成该目录的文件记录项→修改 FAT 表，标识该簇为最后一簇。

5.4　管理多个存储设备

到这里，我们已经实现对海量存储设备的驱动以及存储管理。但是我们系统的基本要求是传输数码相机的相片到 MO 中去，需要同时驱动两个 USB 设备，并合理地建立二者之间的连接。

如果要管理多种类型的设备，要做到一个简易的通用的接口比较麻烦，是操作系统中设备管理的问题。我们这里只需要管理存储设备，使用的驱动程序相同，只需要记录两个 USB 存储设备的不同存储参数，在时间上轮换进行管理即可。

如果要在两个 USB 接口间传输数据，一个标准的数据传输流程如图 5－10：

图 5-10　USB 设备连接标准数据流程

为了提升传输速度，可以把数据传输流程变为图 5－11：

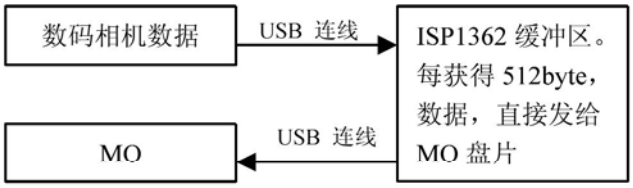


图 5-11 优化后的数据流程

这里省去了数据从 ISP1362 缓冲区到达系统 SDRAM 的阶段。使 MCU 的负担大大减小，传输速度有了明显的提高。但是，由于数据没有到达系统的 SDRAM，无法进行数据处理加工，不过如果仅仅要求数码相机的图片直接到达 MO，这种传输方式是个不错的选择。

要实现图 5－10 的数据流程，必须设计两个 PTD 的轮流切换，两种 USB 海量存储传输方式的轮流切换，以及两个 FAT 文件系统的轮流切换。这实际上已经横跨了系统的固件层，USB 协议栈层和 FAT 文件管理层。为了提高效率，我们专门设计了一个函数“CBItoBO（）”来实现该传输。

另外，ISP1362 还支持 Ping/Pong 传输（双缓冲）方式，这也可以提升传输速度。但是我们试验中，不论是在开发包上还是在我们自己设计的试验板上，ISP1362 的 ping/pong 传输都不是很稳定，估计是电路板在高频电路设计上存在缺陷[39]，这个问题只好留在以后解决。

5.5　小结

经过输数据包的打包拆包后，我们的 Host 系统就可以使用 SCSI 等命令协议来管理 USB 海量存储类设备。利用协议中的 read capacity，read10，read12，write10 和

write12 命令，在了解 FAT 文件系统协议的基础上，可以设计出适合自己系统的一个小型 FAT 文件管理系统。

由于试验中 MO 多数使用的是 FAT16，数码相机多数使用 FAT12 和 FAT16，我们系统中主要支持了 FAT12 和 FAT16。这两种格式除了使用的 FAT 表的簇标志位数不一样外，其它都是通用的。如果要升级到 FAT32，则需要在按照协议在许多细节上做相应的改动。

这个的文件系统实际上包含了操作系统中的设备管理系统和文件系统两个模块[40]，但是从功能上却是差远了。这个文件系统的唯一好处就是程序简单直接，拷贝文件的速度比较快。如果系统要进一步支持其它类的 USB 设备，程序的结构需要优化，设备管理模块也要独立出来[41]。

6 总结

6.1 技术特点和意义

从软件上看，系统的首先要实现一个脱离 Windows 平台的 USB Host 协议栈，然后升级到 USB OTG 协议栈的。目前已经实现了 USB Host 协议栈，该系统的技术特点主要有：

(1) 设计了一个比较合理的 USB Host 协议栈结构

在此结构上，可以实现一般 USB 设备的列举。可以按照自己系统的需求，任意添加和裁减 USB 设备类的驱动。

(2) 完成了 USB 海量存储类协议栈

可以支持市面上按照 Microsofts mass storage class device 标准设计的 USB 数码设备。这类设备的共同点就是在 Window XP 下可以不安装驱动程序直接使用。

(3) 完成了简易 FAT16 文件系统

系统目前只能支持使用 FAT16 和 FAT12 存储格式的设备。但是在此雏形上，支持 FAT32 只需要按照协议做相应的修改即可。

(4) 进行了简单的 USB Hub 管理

硬件上，系统最多只支持两个 USB 设备。然而，软件上实际上是按照管理多个 USB 设备来设计的，不过三个以上 USB 设备管理有待于测试。

(5) 对 USB 系统传输速度进行了一定的优化

初步分析了 USB 系统传输速度的瓶颈。在不使用 DMA 传输方式的基础上，仅仅通过软件上对传输流程的优化，让传输速度达到了一个可以接受的 150K byte/sec。

6.2 技术创新点

整个系统的软件雏形设计主要集中在 2003 年 4 月到 2003 年 8 月。当时国内的 USB OTG 和嵌入式 USB Host 技术才刚刚起步。ISP1362 以及开发包在国内完全没有代理，是从新加坡定购的。中文参考文献基本上没有，整个系统按照 USB 英文协议来设计。技术开发中遇到问题，主要是通过在网上论坛参加讨论加上多实践解决。如果从设计嵌入 USB Host 协议栈来说，勉强算得上是做了有一定独立性和创新性的工作。

现在市面上已经出现了一些带有主机端功能的产品，如：可以直接连接打印机的

OTG 数码相机，可以用来读取 U 盘数据的 OTG 移动硬盘等。这类产品的特点就是把一块 OTG 或者 Host 功能的芯片移植到该产品的内部，它的固件和硬件兼顾自身的功能和控制 Slave 设备的功能。

我们开发系统更接近于一个 USB 数码交换平台，系统管理各式各样的 USB 设备，所有 USB 设备都可以通过我们的平台建立连接。

对比其它嵌入式 USB Host 系统，我们系统的创新主要集中在“使 USB Host 协议栈开发独立”上。我们的系统和现在市面上的 OTG 产品都实现了 USB Host 协议栈，但是我们系统有自己的优点:

（1）系统开发和升级上的优点

数码设备固件不仅仅包含了 USB 协议栈更必须有自身硬件功能的控制程序，由于 USB Host 协议栈比较复杂，这样设计 OTG 产品必然使该数码设备的系统更加复杂。最初，Host 功能支持的数码设备都非常有限，如果多支持一种设备，就需要软件上的一次升级，这样初期软件的频繁升级是不可避免的。

我们的系统把 USB Host 的开发从传统的计算机外设开发中独立出来，所有的 USB 数码产品只需要做好本身的功能就可以了，与其他设备的连接问题可以交给 USB 数码交换平台。这样避免不同 USB 设备上的 Host 功能的反复开发，便于嵌入式 Host 技术的快速成熟；如果需要移植到特定 USB 数码设备上，形成 USB OTG 设备，只需要对 USB 数码交换平台的软件系统进行相应的裁减就可以。

（2）系统资源的占用以及设备成本上的优点

嵌入式系统设计者如果要把传统的 USB Host 移植到数码设备中去，不但要占用嵌入式系统本身就有限的资源，还会造成应用固件的大量反复写入和优化；其次，嵌入式系统中一般要求对实时事件作出明确的相应，添加 OTG 功能会出现更多的中断和实时事件；再次，移植 OTG 功能需要开发人员在 USB 技术上有很高的专业技能，由于 USB 协议繁多，必然会大大增加开发时间；最后，这样开发 USB OTG 模块必然与系统紧密结合，移植性比较差，容易造成 OTG 系统的多次重复开发。

如果让 USB Host 和 OTG 的开发与数码设备的本身功能的开发脱离出来，避免了上面的问题，有利于 USB Host 和 OTG 技术的快速成型。而且这样的系统，成本低，可扩展性和移植性也比较强，可以任意的添加自己需要的功能，不但符合实验室的研究性开发要求，也适合公司的产品开发。

总之，要建立 USB 设备之间的连接，USB 数码交换平台是个不错的选择。如果准备在产品中移植 USB Host 功能，USB 数码交换平台也可以起到辅助设计的作用。

6.3 进一步研究方向

(1) 完善 USB 海量存储类设备管理

现在市面上的数码设备基本上都是使用 USB 海量存储类协议与 USB host 端建立连接。我们系统中海量存储类协议栈已经比较完善了，进一步研究应该集中在多种文件格式（如 NTFS、EXT2 等）的支持上。

(2) 支持多种 USB 设备类

尽量完善 USB Host 系统的设备类协议栈。除了海量存储类外，现在用的比较多的还有人机交互类（键盘、鼠标、手写板等）；打印机类；数据接口类（摄像头）等。

(3) 扩展多种接口与 USB 设备的连接

设计成数码交换平台的另一个优点就是可以方便的扩展接口，其它接口的设备可以与 USB 设备建立连接[42]。比如扩展一个 IDE 接口，就可以实现计算机硬盘与 USB 设备的连接。

致 谢

值此论文完成之际，谨向在我攻读硕士研究生的三年中，曾经关心、帮助、支持和鼓励过我的老师、朋友、亲人和同学致以最真挚的谢意！难忘的三年，这些支持都让我深深地感动，是我的一笔无价财富。

首先要感谢我的导师杨晓非教授。杨老师渊博的学识、严谨的治学态度、不断进取的精神和乐观向上的人生观时刻激励着我克服困难，奋勇向前。谨在此向杨老师表示最诚挚的谢意和最衷心的祝福！

感谢实验室的李佐宜教授、胡用时教授和林更琪老师。他们给我提供了良好的学习、科研环境，给了我许多在科研项目中锻炼自己的机会，使我在短短的三年中得到了许多实际科研和工作经验。

感谢李震、晋芳、程晓敏、程伟明、鄢俊兵博士以及已经毕业的王鲜然师兄，在我最需要帮助的时候，能得到他们技术和精神上的支持与鼓励，让我没齿难忘。特别是李震老师，更给我很多有价值的建议和思想。

感谢同课题组的廖俊卿以及王竹秋、程凯、胡松等人。我们一起度过了艰苦但快乐的时光。

感谢我的家人、室友以及网上一起讨论问题的朋友们，工作和生活中有很多美好的回忆。

邓 剑

二零零五年 四月

参考文献

[1] Jan Axelson 著, 陈逸等译. USB 大全. 北京: 中国电力出版社, 2001

[2] 周立功. USB 2.0 与 OTG 规范及开发指南. 北京: 北京航空航天大学出版社, 2004

[3] On-The-Go Supplement to the USB2.0 Specification, Reversion 1.1. USB Implementers' Forum, 2001

[4] 赵蕴龙, 杨孝宗, 崔刚. USB Host & Host Controller 剖析与实现. 小型微型计算机系统, 2003(1): 54～57

[5] 马伟. 嵌入式 USB 主机系统的研究与设计. 计算机测量与控制. 2003(11): 381～384

[6] S3C44B0X Data Sheet. Samsung Semiconductor.

[7] ARM Software Development Toolkit User Guide, Version 2.50, ARM Limited, 1998

[8] 马忠梅, 马广云, 徐英慧等. ARM 嵌入式处理器结构与应用基础. 北京: 北京航空航天大学出版社, 2002

[9] 满春涛, 吕宁, 李双全等. 基于 ISP1161 的单片机 USB 主机接口设计. 信息技术. 2003(10): 23～24

[10] 冯旭哲, 金光虎. 基于ISP1161的嵌入式USB-HOST技术研究. 电子器件. 2003(4): 393～395

[11] ISP1362 Data Sheet, Reversion 02. Philips Semiconductor, 2003

[12] Jeff Gao. 利用嵌入式 USB 主控进行设计. 电子产品世界, 2003(11): 52～53

[13] 朱勇, 陈艳, 虞鹤松. 数字机顶盒中 USB 大容量存储类主机的设计实现. 电视技术, 2004(1): 50～52

[14] Steve Kolokowsky, Design Considerations for USB Mass Storage. Cypress Semiconductor. 2002

[15] Gary Nutt 著, 孟祥山, 晏益慧译. 操作系统现代观点. 北京: 机械工业出版社, 2004

[16] 王载新, 曾大亮, 郭一平. 程序设计基础与 C 语言. 武汉: 华中科技大学出版社, 1999

[17] ISP1362 PCI/DOS X2 User's Guide (Host Controller), Reversion 1.0. Philips Semiconductor, 2002

[18] ISP1362 PCI Evalution Board User's Guide, Reversion 1.0. Philips Semiconductor,

2002

[19] Universal Serial Bus Specification, Reversion 1.0, USB Implementers' Forum, 1996

[20] Universal Serial Bus Specification, Revision 2.0, USB Implementers' Forum, 2000

[21] ISP1362 Embedded Programming Guide, Reversion 0.9. Philips Semiconductor, 2002

[22] 张弘. USB 接口设计. 西安: 西安电子科技大学出版社, 2002

[23] 边海龙, 贾少华. USB 2.0 设备的设计与开发. 北京: 人民邮电出版社, 2004

[24] 张念淮, 江浩. USB 总线接口开发指南. 北京: 国防工业出版社, 2001.3

[25] Andrew S. Tanenbaum, 陈向群译. 现代操作系统. 北京: 机械工业出版社, 1999

[26] Open Host Controller Interface Specification for USB, Reversion 1.0a. USB Implementers' Forum, September 14, 1999

[27] Universal Serial Bus Device Class Definition, Reversion 1.0 Final. USB Implementers' Forum, 1997

[28] Universal Serial Bus HID Usage Tables, Release candidate 1.0. USB Implementers' Forum, 1997

[29] Universal Serial Bus Mass Storage Class Specification Overview, Reversion 1.1. USB Implementers' Forum, 2000

[30] Information Technology--Some Computer System Interface-II, X3T9 Technical Committee, 1993

[31] Universal Serial Bus Mass Storage Class UFI Command Specification, Reversion 1.0. USB Implementers' Forum, 1998

[32] SFF-8070i Specification for ATAPI Removable Rewritable Media Device, Rev 1.2

[33] Universal Serial Bus Mass Storage Class Bulk Only Transport, Reversion 1.0. USB Implementers' Forum, 1999

[34] Universal Serial Bus Mass Storage Class Command/Bulk/Interrupt (CBI) Transport, Reversion 1.0. USB Implementers' Forum, 1998

[35] 北航 Frank. USB 项目技术报告. 网络资料（www.t10.org）, 2002.12

[36] 朱宁. 硬盘分区及 FAT32 文件结构. 通讯广播电视, 2002(4): 27～36

[37] Microsoft Extensible Firmware Initiative FAT32 File System Specification, Microsoft Corperation, Version 1.03, 2000

[38] Brian W.Kernighan, Dennis M. Ritchie. The C Programming Language 2nd Ed. 北京: 清华大学出版社, 1997

[39] Reunhold Ludwig, Pavel Bretchko 著, 王子宇, 张肇仪, 徐承和等译. 射频电路设

计——理论与应用. 北京: 电子工业出版社, 2002

[40] Andrew S. Tanenbaum Albert S. Woodhull 著, 王鹏 尤晋元 朱鹏 敖青云 译. 操作系统: 设计与实现（第二版）. 北京: 电子工业出版社, 1998

[41] 郑宗汉. 实时系统软件基础. 北京 : 清华大学出版社, 2003

[42] 刘乐善, 欧阳星明, 刘学清等. 微型计算机接口技术及应用. 武汉: 华中科技大学出版社, 2000

附录 1　攻读硕士学位期间发表的论文目录

[1] 邓剑, 杨晓非. FAT 文件系统原理及实现. 计算机与数字工程, 2005(9)（已录用）

[2] 杨晓非, 邓剑, 廖俊卿. 嵌入式系统中 USB Host 功能的开发. 华中科技大学学报, 2005(10) （已录用）