基于 ARM9 嵌入式系统的 U 盘文件加解密系统

摘　要

针对目前 USB 移动存储设备在数据安全中存在的漏洞，本文提出了一种新的 USB 存储设备数据安全解决方案。本系统通过 USB 接口连接于计算机与 USB 存储设备之间，对计算机和 USB 设备之间传输的数据进行解析，利用指纹模块提供的稳定密钥加密并转发 USB 数据。实现文件加密、文件名不加密的效果，使得 U 盘中普通文件与加密文件可以共存，多个用户可以共同使用一块 U 盘而无需担心数据泄露。设备具有相对计算机、USB 存储设备独立的特点，无需对两者做任何改动，保证了对普通 U 盘的支持。

本文选用 AT91SAM9260 作为实现平台，针对嵌入式核心板设计了外接电路，提供了 USB 的主从接口、RS232 串口、面向指纹模块的 SPI 接口。不使用操作系统，移植了 USB 海量存储协议的主机和从机驱动。上电启动时，使用 bootstrap 直接加载应用程序，并跳转执行。主从驱动协同工作，并添加对 FAT32 文件系统的识别，使系统对文件名透明传输的同时对文件数据加密。加密算法采用 AES 标准、CBC 分组模式，采用 32 位快速 AES 算法。另外，核心板空余的 32M 内存被用来缓存 USB 数据，对同一数据的再次访问可以直接从内存获取。最后通过 SPI 接口同指纹模块连接。在 USB 全速设备的基础上，通过优化软件，读写分别可以达到 550KB/s 和 230KB/s 的传输速度。

关键词：USB 协议　FAT32 文件系统　文件加密　AES-CBC　AT91SAM9260

A File Encryption System for USB Memory Device Based on Embedded System of ARM9

Abstract

As the threat exists in the storage protection of the present USB Mass Storage Device, this article proposes a new method to provide them with security services. All of the packets will be processed when they go through this system, which connects PC and USB Mass Storage Device. Before forwarding them to the USB device, these packets will be chosen to be encrypted using the key provided by the fingerprint module. As a result, when be directly connected, the disk will be found normal with all the directories correctly shown while the contents of certain file are obfuscated. Thus it becomes possible that in such disk coexist the plaintext and the ciphertext and that multi people share one disk without worrying about the privacy. This system is not only relatively independent to the computer and USB Device, but compatible with various USB Memory Devices without any update to PC or USB Device.

On the platform of AT91SAM9260, this article designed the extension circuit equipping the core-board with various ports, such as USB HOST&DEVICE port, RS232 and the SPI to fingerprint module. After startup, the system runs directly the Bootstrap which would load the application mainly consisting of the two drivers, USB MSD Host Driver and Device Driver, without any operating systems. With the two drivers cooperating, the system should transmit the directory related data transparently and encrypt the contents of files depending on the knowledge of FAT32 system. And a 32-bit optimized algorithm supports AES and CBC block cipher mode well with a fast run. What's more, sparing 32M memory in SDRAM and using it as a Cache helps to reduce the procedure to fetch it from USB Device when the in Cache data were requested again. At last, the SPI provides the interface to communicate with the fingerprint module. The final data rate can reach 230KB/s while writing and 550KB/s while reading respectively on the full-speed USB port.

Keyword: USB Protocol; FAT32 File System; File Encryption; AES-CBC; AT91SAM9260

目　录

第一章　绪论

1.1 引言

在信息技术突飞猛进的今天，人们在享受技术进步的同时，也要遭受越来越多的数字危机。U 盘，由于其拥有容量大、体积小、价格便宜等众多优点，已成为人们移动存储的首选设备。因此 U 盘的数据安全问题也日益露出水面。世界各地关于 U 盘数据泄露的事件层出不穷，既有企业私人的，也有政府军方的。这些事件的发生给个人隐私和集体利益带来了巨大的损失。

可移动存储设备的安全现状：

目前市场上所见到的 U 盘大多不带有任何安全功能，计算机的文件数据直接经过 USB 端口存储到 U 盘中的 Flash 芯片中，这类 U 盘我们称之为普通 U 盘。在很多场合，人们并没有意识到存储于 U 盘中数据的安全性，唯一可以做到的就是尽量保证 U 盘不会丢失。而一般情况下，丢失之后的 U 盘中存储的数据在他人眼中变得一览无余。同时 U 盘互借、互拷也会给攻击者带来窃取数据的机会，随着 USB 的数据传输速率的不断提高，上 G 字节的数据瞬间就可以被攻击者偷偷拷走。

现有安全方案概述：

1、计算机上的安全措施[6]

一种方式是通过在操作系统内核中嵌入对 USB 接口转存数据的安全控制来实现写到磁盘上的数据为密文形式，当需要从磁盘上读出数据时做相应的解密工作。这种安全机制相当于在磁盘前段加入一块加解密装置，在一定范围内可以保证数据的流通，通用性强。如果使用硬件电路的方式实现加解密对文件读取速度上的影响也较小。但是这种方案需要对操作系统的内核做修改，或者是对主板的结构进行修改，工作量大、标准得不到统一，一旦到了大的范围内，各种操作系统的计算机之间就无法进行文件的共享，产生了无谓的麻烦。而目前国内使用最多的操作系统 Windows 仍然是一个商业运作的操作系统，普通用户无法对其内核进行修改，使用该种方案对数据保护的用户只能局限在 Linux 几种开放的操作系统下，故至今未得到广泛的应用。而对主板进行修改费用高、难度大，很难成为普通用户及小公司的首选。

于是在这种情况下，很多渴望数据安全的用户将目光投到了一些加密软件上，使用折中的办法满足自己的需求。通过授权 PC 机上的程序，使其工作在系统层上，对出入 U 盘的数据进行加解密来提高 U 盘的安全性能。这种加密方式有很多不同的形式，不论是系统自身安装的软件还是通过移动设备上的 autorun 自动加载的应用程序，都使用了这种方式。这种加密方式经济代价低、硬件上无需做变动。但是却缺乏通用性，一般来说各个厂家提供的软件都各不相同。不仅占用系统资源，而且还容易受到木马、病毒的威胁。通常来说，这种加密方式需要手动输入密码，所以长度有限的字符、数字组合很容易被后台运行的木马劫持。

2、可移动磁盘上的改动

此种方案来自与各大 U 盘制造厂家，它们的共同点就是加密模块集成在 U 盘之上，使 U 盘自身带有保密功能。这类方法类似与在计算机中的磁盘前段加上加解密模块一样，只不过将该模块放到了可移动磁盘内部 Flash 的前面。同样，这类产品也有很多种不同的模式，而且大多数 U 盘第一次使用时会要求先安装驱动程序。之后每次插入该 U 盘之后，系统自动调用驱动程序，开始文件传输。但是也有的产品是不需要驱动的，但此类也采用 autorun 自动加载应用程序。因为病毒的泛滥，通常 autorun 会被杀毒软件认为是可疑对象而隔离。使用起来带来不必要的麻烦。同样也会受到病毒的侵扰……

这些产品将加密过程转由硬件实现，但是具体使用的加密标准却不以公开，无法对其展开理论分析，安全强度未知。

3、独立的加解密设备[7]

此种类型加解密设备的典型的例子就是华中科大的“基于 CH375 的嵌入式 USB 文件加解密系统”，这个系统完全与计算机脱离，工作时首先将 U 盘中的数据全部取出来，加密之后再写入 U 盘，然后将明文删除。总的来说这样的系统有几个缺点，首先不是实时的加解密，每次使用 U 盘的前后都必须使用该设备然后将其加解密之后再使用；其次是我们知道在删除数据时很难在磁盘上不留下痕迹，因为在 FAT 文件系统下，文件的删除是通过标定目录已删除来实现的，真实的文件数据并未得到损毁。

这种对 U 盘文件的处理过程实际上就等价于在计算机平台上一个可以将 U 盘中数据取出、加密、再写入的软件。该方案可以使用对计算机和 U 盘两者都独立的设备提供数据安全服务，在思想上有一定的新意，但是并没有对 U 盘数据安全方面的保护起到实际意义上的贡献。

近年来，U 盘市场上又出现了基于指纹识别身份认证技术的产品，硬件上增加了指纹识别模块，但仍会依赖一段在 PC 上执行的程序，这段程序或预先安装在 PC 上，或来自 U 盘上的某块存储区域，国内代表作是亚略特系列 U 盘。除了解决了无需手动输入密码外，软硬结合方案的其他缺点依旧没有解决。

1.2 关于本课题的研究

本课题的主要目标就是整合现有的解决方案，提炼出他们的优点，将其集中到我们所要完成的 USB 文件加解密系统。在 1.1 节中对计算机的主板甚至是操作系统所做的改进我们可以拿到 USB 接口外面来完成；可以吸取指纹 U 盘上使用指纹作为身份认证的高度安全的方案；同样把本系统做成一个脱离于计算机与普通 U 盘的独立设备，这样便可以集多家之长，完成预定的 USB 文件加解密系统。

1.2.1 本文的研究目标

本着一种高度的灵活性与独立性的设计理念，本课题所设计的 USB 文件加解密系统必然是

独立于计算机与普通U盘而单独存在的。它通过一主、一从两个USB端口连接于计算机与U盘之间，实现透明的数据传输（即任何一方都不会感觉到中间这个加解密设备的存在）。同时必须做到文件级别的数据加密，这样就可以在同一磁盘中存放多个不同用户的文件和没有加密的文件而相互之间不出现干扰，做到高度的灵活性。

而另一方面实现指纹的无模板存储的特殊处理，直接由采集到的指纹图像生成加解密所需要的密钥。而这一过程则需要尽可能的保证这串密钥的稳定性，否则很可能发生加密后的文件自己也无法打开的尴尬情况。

在以上描述的基础上，可以得出结论：该系统可以做到只有在U盘（密文）、手指（密钥）、加解密装置三者同时存在的情况下才可以在计算机上打开用户存入该 U 盘的文件；而普通用户也只能浏览U盘内的文件名和不同的目录而已。

1.2.2 主要工作内容以其创新点

由于本课题工作量大，需要大量的理论创新和实践工作，所以分由三位同学共同完成。本文主要负责ARM平台上的USB协议的解析、文件级的数据加密以及和负责指纹图像处理的DSP平台之间的接口设计。通过对设计目标的进一步分析，可以将其主要的工作内容概括为一下几个步骤：

(1). 掌握并完成硬件电路板的制作技术和调试

(2). USB 主机、设备的工作原理，以及 USB 主从设备间的通信协议

(3). 支持对普通U盘文件的加解密，文件名称可见，内容不可见，方便文件管理

(4). 基于嵌入式的加密算法实现

(5). 连接指纹模块

(6). 保证对计算机与U盘的透明，并支持市场上大多数的通用U盘

主要创新点：

(1). 采用了独立于 U 盘与计算机的设计思想，采用过滤的技术实现对原始操作无干扰的数据加密

(2). 对 U 盘中的每一个文件进行加解密，而不是对整个 U 盘加解密，使用和管理都更为方便。

第二章　总体方案概述

为了给 USB 设备的使用带来最小的影响，同时又要具备数据加密的功能，因此将加密设备设计成一个数据过滤型的设备成为首选。所谓数据过滤就是将本设备连接于 USB 存储设备和计算机之间，数据的存储、读取都必然要通过此设备，因此也就有可能在使用过程中，实时的提供数据加密功能。借助于 ARM 平台强大处理能力和丰富的外设，这部分功能可以在 ARM 平台上实现，具体芯片型号选为 AT91SAM9260.

另一方面通过对指纹信息的读取，采用先进的图像处理和特征提取的算法，从指纹图像中提取出用户的身份信息，并将其作为密钥提供给数据加密模块。该部分的处理涉及到大量的浮点运算，所以考虑在 DSP 平台上实现，DSP 与 ARM 之间可以使用 SPI 接口实现简单的通信协议，完成密钥的传输。实现结构如图 1 所示。

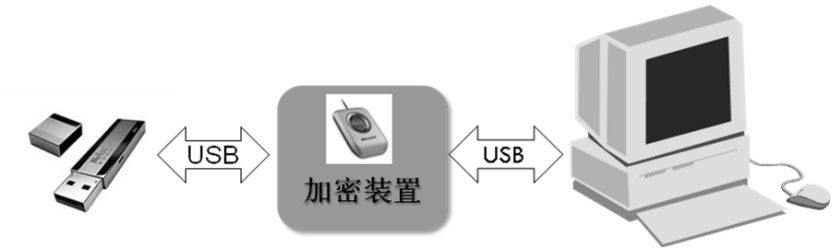

图 1 系统方案图

本文的重点工作放在ARM平台上的USB协议解析和文件数据的加密。因此可以将此部分的工作分为以下三个主要部分：

1、USB 协议的解析[2]

要在同一系统中配备一主、一从两个 USB 接口以及对应的驱动程序，对计算机伪装成普通的 U 盘，响应所有的计算机请求；对 U 盘伪装成“计算机”，根据计算机发来的请求稍加改动再次转发给 U 盘。

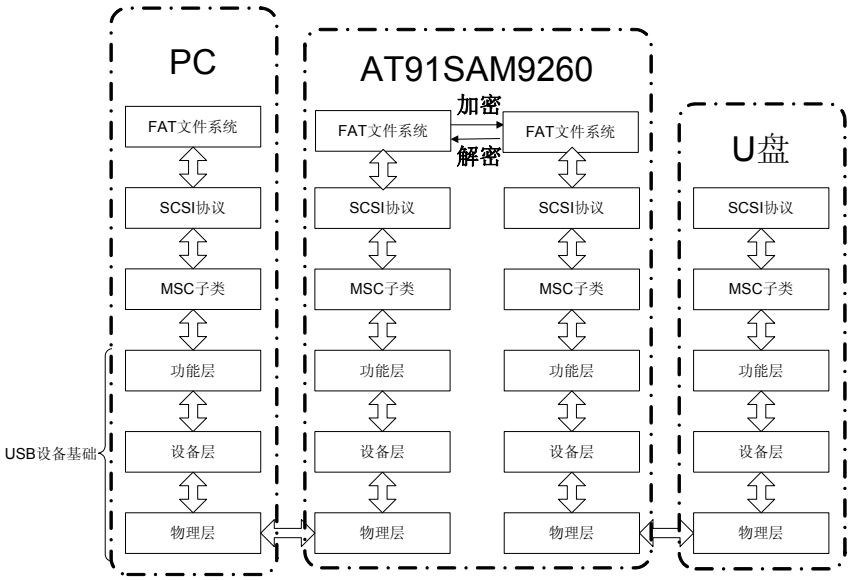

图 2 系统加解密协议栈

类似于 TCP/IP 协议层，USB 自身也含有类似的协议，下层为上层提供服务，逻辑层之间互相通信，真正的数据流都必须通过物理信道。图 2 描述的是整个系统对 USB 数据的处理过程。ARM 通过其 USB 接口连接到 PC 的 USB 接口获得底层的差分数据信号，使其进行转化为可识别的数据包。CPU 进行后续解析，在 SCSI 层[19]解析出的数据包是包括写入到 Flash 中的数据块、地址及其数据长度等信息，并进一步判断当前数据是否需要加解密。调用 AES 子程序，对数据加密，之后把它按照加密前的方式转发，在物理层通过 USB 的主驱动程序发送给 U 盘。

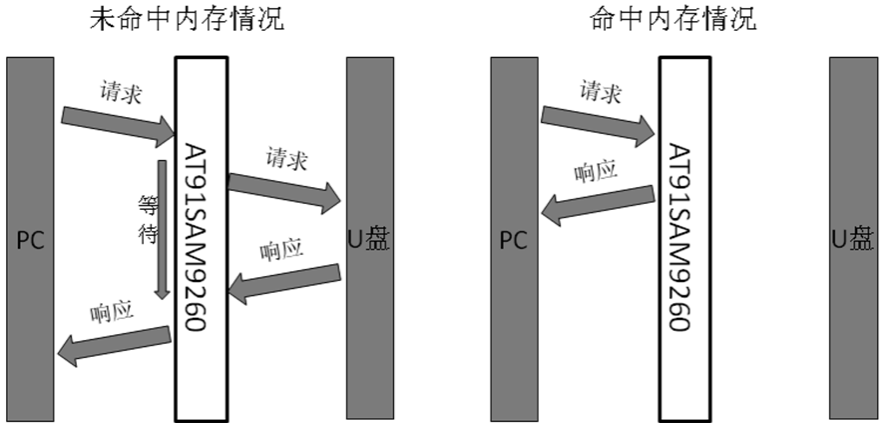

图 3 透明传输示意图

另外 USB 是一种基于主从结构的总线形式，总线上的数据传输都必须由主机来发起，只有通过请求才会产生相应的响应。从图 3 的左图可以看出在数据进行转发过程中会在时间等待上产生一定的代价，但是在目前的嵌入式系统中还存在这一大块没有经过开发利用的内存，如果将这部分内存开发出来，存储一些已经通过本设备的数据，那么下次有相同的请求时，就没有必要重新到 U 盘那边请求数据，直接可以从内存中取出，减小时间开销。

2、文件数据的识别

在 USB 上传输的数据是非常简单的，主机只通知 U 盘该数据要存放在 Flash 的哪块地址，这段数据的长度。所有的文件系统的结构在操作系统中得到了封装，不论用户的操作是读写、删除还是格式化，所有的操作在 USB 总线上的看到的只会有两种数据的读和写。因此要通过被封装了的数据还原出用户的原始操作几乎变得不太可能。

但是换一种思维，实际上的保密只要做到磁盘上的文件数据保密便可达到目标，那么该问题的求解转化成“在 USB 上传输的数据中文件数据的检测”问题，进一步化简为“USB 数据流中目录簇的识别”。既然文件系统的组织必须遵循一系列规则[5][8]，那么这两种数据形式必然有其数字特征，相比文件数据，目录簇的这种特征显然要更加突出，易于发现。

3、数据加密算法

加密算法多种多样，在本系统中要选择一种快速、且安全性高的数据加密算法必然会想到 AES 算法。在密码学界，AES（Advanced Encryption Standard）加密算法是美国政府所采用的一种加密标准，是世界上最流行的对称加密算法之一，它的安全性可以得到保证。另外正因为它

的广泛应用，代码的重新利用可以大大减小项目的开发周期。

除此之外，数据加密靠的不仅仅是算法，同时也需要加密模式的参与。本文选用 CBC 分组密码模式。

最后，为了更加快速的使系统启动，考虑不使用操作系统，直接加载底层的应用程序，速度快、效率高。

第三章　硬件电路设计

基于 AT91SAM9260 的 ARM 开发平台已经有着比较成熟的应用方案，可以利用实验室现有的硬件资源来构造本文所需的硬件平台。考虑到 ARM 核心板部分在各种应用情况下的变化一般不是很大的情况下，开发人员通常将其做成单独的一个电路模块，然后根据不同的应用需求，通过配备不同的底板来扩展 IO 端口和电源模块。核心板，顾名思义，就是 ARM 硬件电路的核心部分，它包括了内存、NAND Flash、模式选择开关以及必要的电源管理模块。相对于底板，核心板的设计复杂但相对固定，通常采用 Atmel 公司推荐的配置方式[20]，因此本毕设并未涉及到核心板电路上的设计工作[21]，只是在此对它做简要的列表说明：

>> CPU：AT91SAM9260

>> SDRAM：32M×2，两片 HY57V561620 构成 32bit

>> NAND FLASH：128MB，一片 K9F1G08

>> RTC：PCF8563T

>> JTAG：JTAG 调试接口，标准 20PIN

>> 电源：3.3V 电源输入和 3V RTC 后备电源输入

>> 扩展接口：数据、地址总线接口和剩余 GPIO 口通过 2 个 80PIN 双排插阵引出

根据本毕设的需求和对核心板硬件资源的分析，在底板上必须配备一个电源模块、USB 的主从接口、用来作为调试工具的串口、必要的 LED 状态显示以及对外的接口。

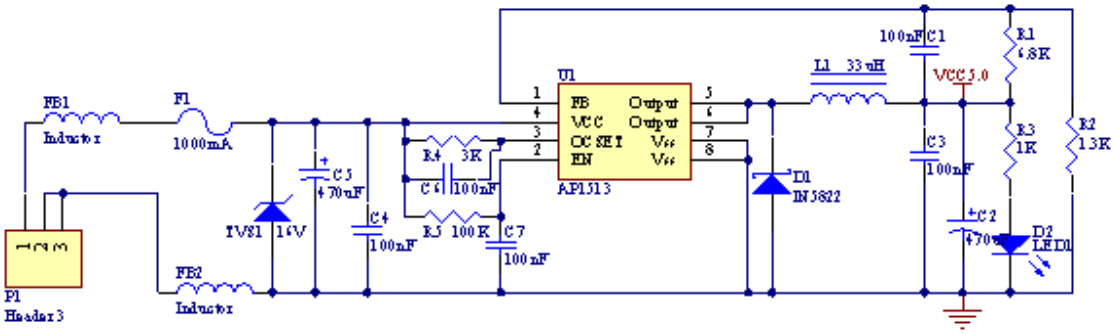

图 4　电源模块原理图

电源管理使用 AP1513 芯片产生更加稳定的 5V 电源，另外还可以从 USB 电源中取 5V 电源，该电源通过二极管连到上述电源管理芯片的输出，保证在外部电源供电时，USB 工作也不会受到影响。

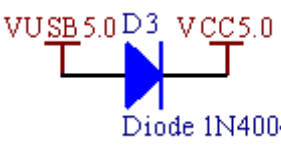

图 5 USB 供电在系统中的应用

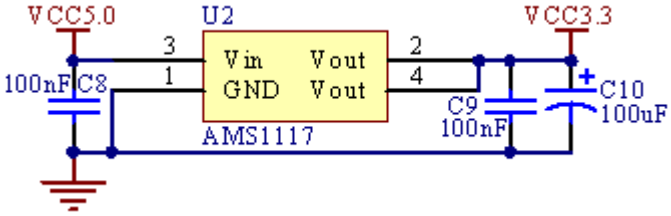

图 6 5V 到 3.3V 的电压转化电路

如图 6 所示，使用 5V 到 3.3V 电源转化，向核心板供电。

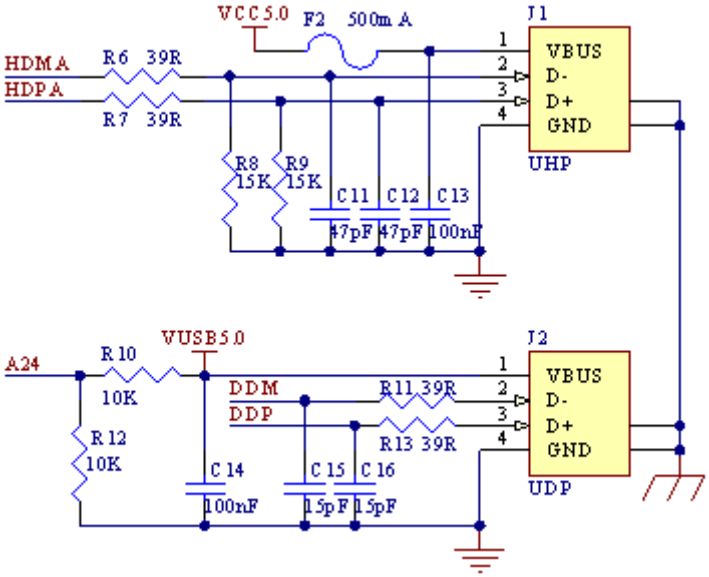

图 7 USB 接口电路

上图为 USB 主设备接口，5V 电源通过 500mA 的保险丝介入，以保护 U 盘以及不正确时候时电路的安全。下图为 USB 从设备电路，其中 VUSB 通过 10K 电阻接到 ARM 的 A24 引脚，该引脚可以用于检测本设备上端是否有主机供电。不论是从口还是主口，D+和 D-两根数据线必须根据主芯片的要求接入必要的电阻以及旁路电容，以提高电路的抗干扰性能。

完整的原理图见附录的图 28，双层 PCB 版图的正面效果见图 8。

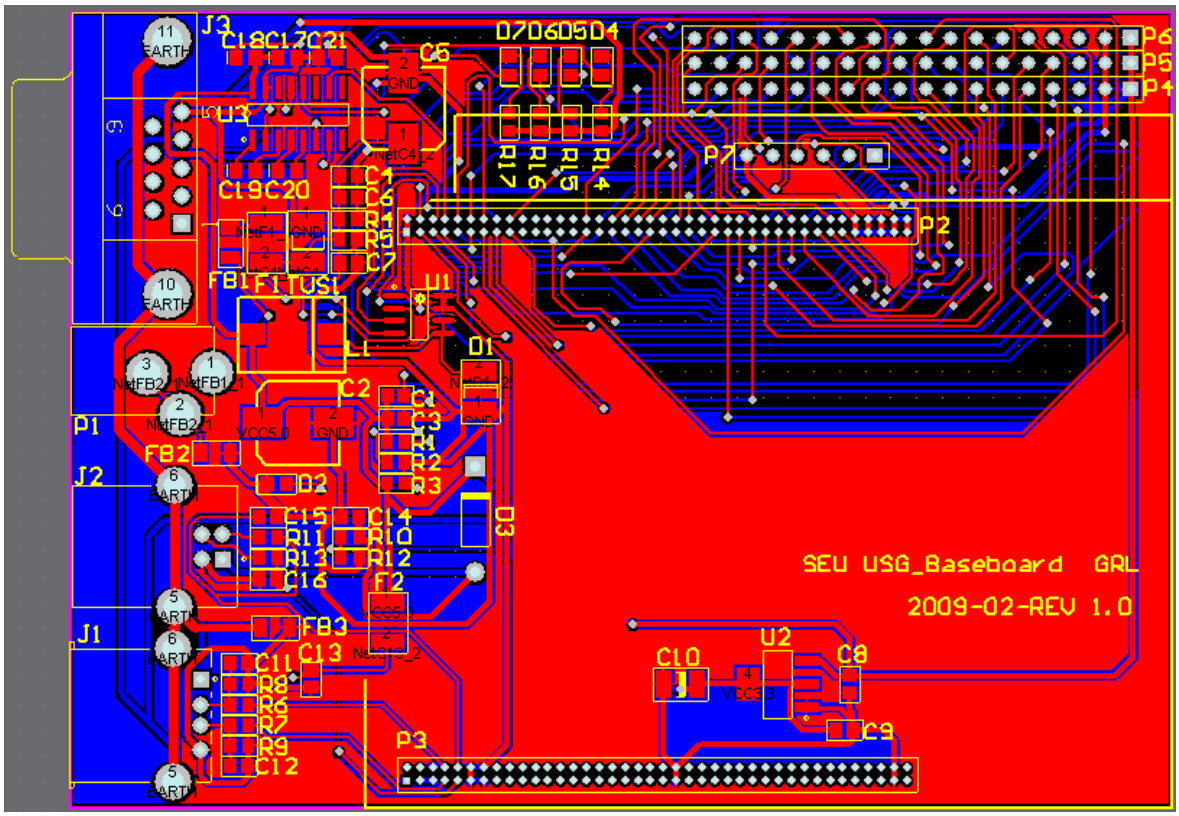

图 8 母板的 PCB 版图

如图 9 所示，为制作完成的硬件电路，ARM 核心板通过两列细口双插槽与母板相连接。上面分别分布了 SPI 接口（连接 DSP 指纹预处理模块）、USB 设备端口（作为 USB 设备连接到计算机）、USB 主机端口（作为 USB 控制器连接 USB 存储设备）。

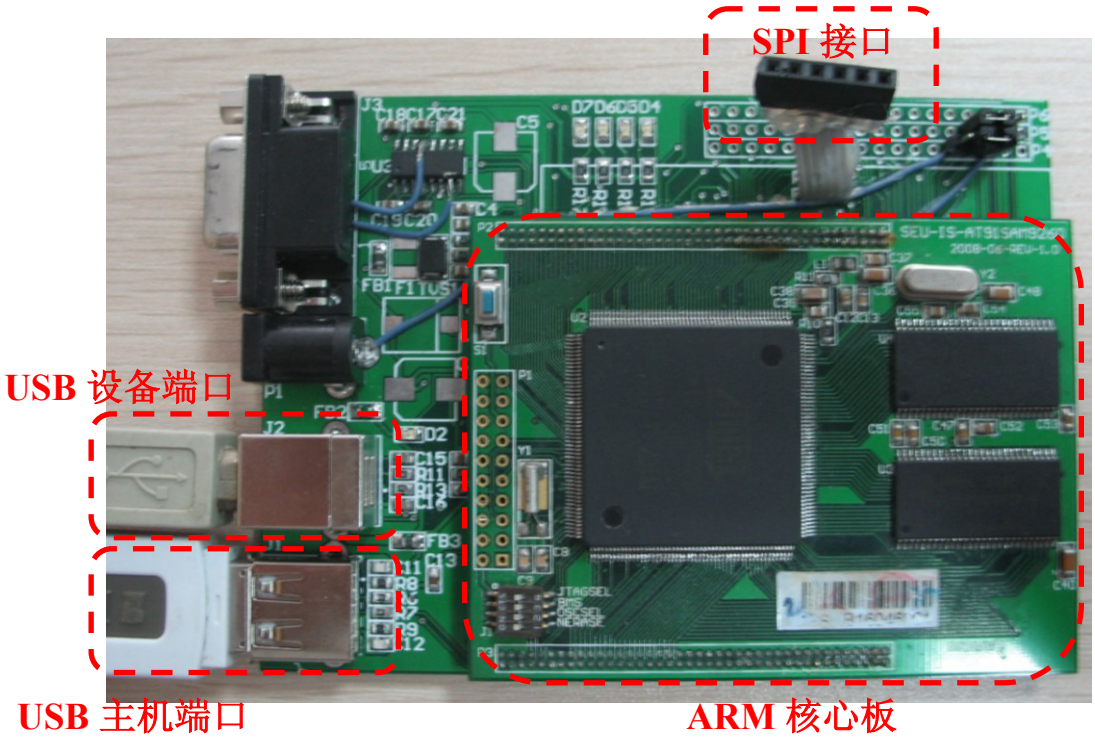

图 9 电路（母板+核心板）实物图

第四章　开发环境的建立

一个完整的开发环境是开发 ARM 应用程序的必要前提，只有当第一个简单的 Hello World 成功的出现在电脑上的时候，意味着从代码的编写、下载、程序运行一直到终端显示都能够正常工作了。任何复杂的系统都是构建在这个简单的基础上的，并且要尽量使这个过程尽量的简单，为以后的调试工作节省下大量的时间。

本毕设的开发环境是建立在计算机内部搭建的虚拟局域网上的，在 Windows XP 上进行代码的编辑和二进制可执行文件的下载，在 Fedora 8 下对工程文件进行编译、连接。所有的工程文件存储在 Linux 操作系统中，通过虚拟局域网，使用 Fedora 自带的 Samba 服务功能共享工作空间[21]，并在 Windows 下将共享文件夹映射成网络驱动器，从而实现不同的操作系统和不同的文件系统之间的文件共享功能。

图 10 开发环境示意图

首先介绍两款 Windows 下的必备软件：Source Insight 和 SAM-BA。Source Insight 是一个面向项目开发的程序编辑器和代码浏览器，它拥有内置的对 C/C++, C#和 Java 等程序的分析。Source Insight 能分析源代码并在工作的同时动态维护它自己的符号数据库，并自动为用户显示有用的上下文信息。Source Insight 不仅仅是一个强大的程序编辑器，它还能显示 reference trees，class inheritance diagrams 和 call trees。而 SAM Boot Assistant (SAM-BA™)则一种提供了在 Windows 下对 Atmel AT91 ARM 器件编程的操作十分简单的可视化软件。它可以选择通过 USB 端口或者更为常用的 RS232 端口对目标设备进行编程。它可以定义可执行文件下载的具体地址，还可以对内存空间的任意位置进行查看。

考虑到开发过程中会不得不对大量的现有的 USB 驱动程序源代码的阅读，为了更好的了解各种变量的定义和大量的函数间相互调用的复杂关系，提高工作效率，SourceInsight 可以说是一款十分符合要求的编辑器。

SourceInsight 编辑好的源程序通过 Windows 和 Linux 之间的共享通道，存储到虚拟机上，然后在 Linux 在执行编译命令。在 Makefile 中编写需要编译的源程序和所使用的编译器[22]，本文所使用的交叉编译工具 Gnuarm3.4.3 为 ATMEL 官方推荐使用的编译器版本。编译成功的二进制文件再通过共享通道由 Windows 下的 SAM-BA[23]下载到目标板上。在下一章，我将详细介绍可执行的二进制文件下载的地址以及 ARM 启动的详细过程。

第五章　AT91SAM9260 的启动

首先介绍一下系统的存储的地址映射

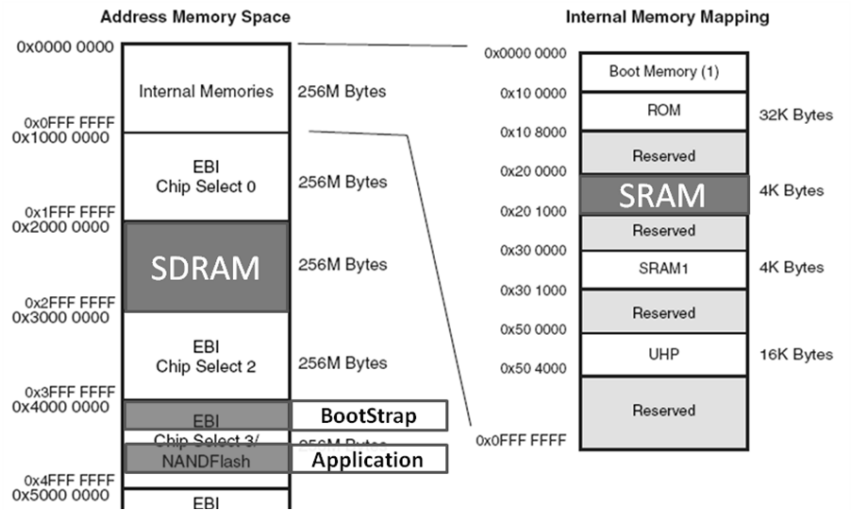

图 11 AT91SAM9260 地址映射简图

图 11 只是列出了前面一部分的存储器地址映射的信息，从中我们可以看到 CPU 内部已经配备了一个 32K 的 ROM 和两个 SRAM，CPU 对它们的存储都可以在 1 个时钟周期内完成。另外在核心板上还有片外的 SDRAM 与 NAND Flash，SDRAM 是由两片 16-bit 的 32M 内存通过位扩展而构成的 32-bit 的 64M 内存，而 NAND Flash 单片大小为 128M。

系统具体从以上那么多的存储器中的哪一个开始启动还要取决于 BMS（Boot Mode Select）引脚的电平，具体配置见下表。CPU 启动之后总是从 0 地址开始执行指令的，所以在 REMAP 之前会根据 BMS 的情况载入相应的数据到 Boot Memory 中运行。一旦系统启动完成，重新进行地址映射之后从内部的 SRAM 启动。

表 1 内部存储映射

Address	REMAP = 0		REMAP = 1
	BMS = 1	BMS = 0	
0x0000 0000	ROM	EBI_NCS0	SRAM0 4K

因为核心板 EBI_NCS0 上没有挂载任何存储设备，所以启动模式 BMS 选择置为高电平，从固化 ROM 启动。ROM 内部的启动程序要完成的工作以列表的方式罗列如下[1]：

(1). 初始化调试单元的串口（DBGU）和 USB 设备端口（用于进入 SAMBA 模式）。

(2). 加载 ROM 中的程序，运行。它会从 NAND Flash 中 0 地址处开始检查是否有 8 个有效的中断向量存在。所有的这些中断向量必须是 B 跳转或是 LDR 加载寄存器指令。与其他存储器不同的是 NAND Flash 的第六个中断向量无需指定加载程序镜像的长度，因为在 NAND Flash 中每次加载都会拷贝 4K 程序到片内 SRAM 中。加载完成后重定向地

[1] 在 DBGU 初始化之后实际上还会有关于 DataFlash 的检查工作，具体流程同 NAND Flash 相同，因为核心板没有配备 DataFlash，故在次隐去。

址，跳转。

(3). 如果没有合法的中断向量，SAM-BA 启动程序被执行，并等待计算机端通过串口或 USB 接口发起 SAM-BA 会话。

以上工作的具体流程见下图：

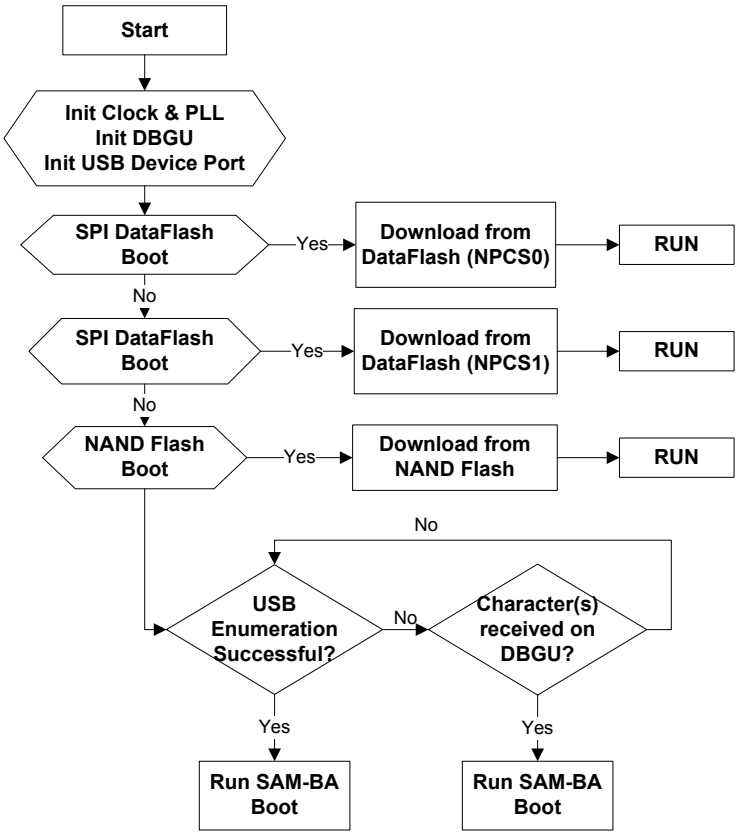

图 12 启动程序算法流程

从启动的结果来看，实际上只有两种效果：1、将程序拷贝到内部 4KSRAM，REMAP 并开始运行；2、进入 SAM-BA 模式。后者和我们之前介绍的 Windows 平台上的程序 SAM-BA 有很大关系，SAM-BA 模式通过计算机上运行的一段主机控制程序和另一段在嵌入式平台上运行的从机程序的相互配合通信，可以将计算机上的可执行代码下载到嵌入式指定的地址上。具体的通道也分为 USB 方式和串口方式，因为串口要留作调试信息的输出，因此选用 USB 电缆完成程序的下载工作。

Boot loader：

从上面的流程可以看到，在系统 REMAP 之后程序会跳转到 4K 大小的 SRAM 中运行，虽然 SRAM 有着很高的存储速度，指令能够更快的被 CPU 执行。但是谁也不能保证自己的程序能够被限制在 4K 的大小之内，而且一个 4K 的程序也不能指望它能够完成多少复杂的系统功能。那么就需要在这 4K 大小的空间中再次完成一个跳转工作，将程序指针 CP 指向到可用范围更加广的内存中去（当然这之前需要完成一些必要的系统配置）。这个工作就是由 Boot Loader 来完成的。

与操作系统的 Boot Loader 相比[17][18]，这里只需要使用第一级的 Loader——Bootstrap 就能完

成任务。因为不需要启动操作系统，Bootstrap 只要能将自己的指针指到应用程序的入口即可。

首先该 Boostrap 要能被启动程序认为是一段能够正确运行的程序才会被拷贝到片内内存，所以它必须是以 8 个中断向量为开头的。因为 Bootstrap 功能比较简单，用不着处理中断，8 个中断的存在只是为了被启动程序发现为可用二进制代码，所以实际上这些中断向量只是指向一个死循环而已。它的八个向量分别如下所示，虽然在这边用不到，但是在应用程序中却是中断处理的必由之路。

```
_exception_vectors:
    b    reset_vector        /* reset */
    b    undef_vector        /* Undefined Instruction */
    b    swi_vector          /* Software Interrupt */
    b    pabt_vector         /* Prefetch Abort */
    b    dabt_vector         /* Data Abort */
    b    rsvd_vector         /* reserved */
    b    irq_vector          /* IRQ : read the AIC */
    b    fiq_vector          /* FIQ */
```

然后它要依次进行：

(1). 堆栈初始化

(2). 时钟配置

(3). 初始化 bss 字段

(4). 跳转到 Bootstrap 的 main 函数

(5). 从 main 函数返回并跳转到目标地址

注：bss 字段是指那些被分配的初始值为 0 静态变量所对应的地址，在 main 开始之前必须将其声明的未赋初值的静态变量置为 0；main 函数完成对 NAND Flash 的初始化，并将相应的应用程序从中复制到片外内存中的 JUMP_ADDR 地址，并将该地址返回以供 main 函数之前的这段 crt0_gnu 的汇编代码跳转用。

对程序下载方法所做的改进

开发板最初所争对的应用主要是对基于 Linux 操作系统来开发的，所以在那种情况下 Bootstrap 并不直接引导应用程序，而是从 NAND Flash 复制一段被称为 U-Boot 的程序（U-Boot 通常和 Bootstrap 共同构成我们所熟知的二级和一级启动代码 Bootloader）到片外 SDRAM 中的最后一段地址中，然后跳转到 U-Boot 执行二级启动代码，U-Boot 实际上是一段相对比较复杂的程序，它能够简单的支持一些命令行。Linux 内核也就是借助 U-Boot 才可以被复制到 SDRAM 的 0 地址才能够被正确的运行。

那么当第一段的 Bootstrap 烧入到 NAND Flash 的首地址之后，怎样才能再次令嵌入式系统进入到 SAM-BA 模式呢？原来 Bootstrap 除了加在应用程序还有一项功能：每次运行自动扫描 PC11 引脚[2]，如果该引脚低电平，则程序写入一段 0 到 NAND Flash 的 0 地址，这样就可以阻止下次复

[2] 在 Bootstrap 从 Flash 的指定地址载入应用程序或 U-Boot 前对 NAND Flash 的初始化函数 nandflash_hw_init 中调用 nand_recovery 实现此功能。

位时电路开始运行程序，而可以再次进入到 SAM-BA 模式，在这个模式下可以重新烧入更新后的程序。但这样做的一个缺点就是每次烧入程序代码之后都必须重新写入 Bootstrap，给程序的调试工作带来了很大的不便。

表 2 合法中断向量表

地址	机器码		汇编语言
00000	ea000006	B	0x20
000004	eafffffe	B	0x04
000008	ea00002f	B	_main
00000c	eafffffe	B	0x0c
000010	eafffffe	B	0x10
000014	eafffffe	B	0x14
000018	eafffffe	B	0x18
00001c	eafffffe	B	0x1c

此外，由于Flash的烧写次数是有限的，所以在以上基础上要尽量减少对Flash的写入次数，并最大程度的使调试过程中程序的烧写变得简单。首先，每次写入新的程序时都必须进行一次 Bootstrap 的擦除和写入，而 Bootstrap 在调试过程中通常是不会被改变的，所以这个过程确实是一个多余的过程，它所达到的目标就是进入SAM-BA模式。而多次的对NAND Flash的0地址反复做无意义的擦出、写入很大程度上缩短了系统的使用寿命。

根据 CPU 进入 SAM-BA 模式的流程，可以看到在检测 NAND Flash 中是否有正确的执行码取决于对前 8 个中断向量的检测（如图 13 所示），可以发现如果在复位的同时人为的对上面读到的数据进行干扰就可以使 CPU 认为 Flash 中的数据不为可用代码，最终达到误导 CPU 进入SAM-BA模式而无需累赘的多次重复的对Bootstrap的烧写。再看到NAND Flash的引脚，发现它只是用了一个 8 位的数据总线[24]，因此可以在该总线上下功夫来欺骗 CPU。具体的做法是：在复位的同时短接D7与地线。这样做可以方便的解决启动是选择SAM-BA还是正常运行，而且每次仅需对改动过的代码进行写入就可以了。

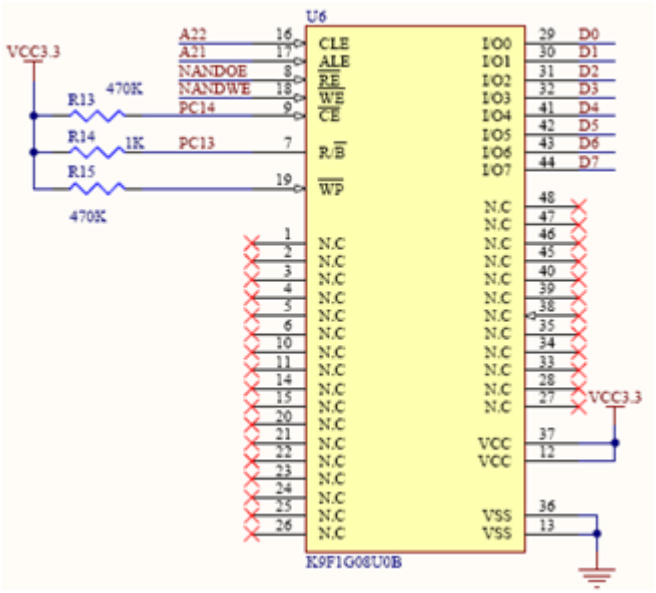

图 13 核心板中 NAND Flash 电路原理图

第六章　USB 透明传输

为了分析出入 USB 存储设备的数据流就必须截取所有的通过 USB 端口的数据，因此介于计算机与可移动存储设备之间的加解密装置必须同时具备 USB 海量存储设备类所具有的全部功能，于此同时还要有主动向 USB 从设备读写数据的能力。本章先从理论上简述 USB 以及它的 MSC 子类的基本原理和最基本的命令、数据格式，然后再从实现角度分别讨论主机驱动和从机驱动的实现，最后论述如何利用在加解密设备中主机驱动和从机驱动之间的通信实现文件的透明传输功能的。

6.1 USB 协议规范

USB 全名为 Universal Serial Bus，即通用串行总线。它是一种基于主从方式的总线结构，在本加解密设备中就同时实现了 USB 的主、从功能，主、从功能分别面向移动存储设备和计算机。数据信息通过串行接口引擎在串行数据和并行数据之间转化，在物理层上通过一对基于差分曼彻斯特码的差分电流信号传递数据。主从设备分别配备了复杂的 USB 硬件电路以更好、更快的实现 USB 的数据传输，同时为上层驱动和应用程序的编写提供最大的便利。

6.1.1　USB 基础简介

USB 有着自己的一套理论体系，从物理层一直到上层应用，从严格规整到灵活多变，每一个层次都有一定的规范。就 USB 自身而言十分复杂，很难在篇幅有限的地方叙述清楚，在下面的几个小的部分中主要做一些简略的介绍，并将文章的主体部分留给如何实现 USB 数据的透明传输用。有关 USB 详细的内容可以参考相关网站[3]。

6.1.1.1USB 的拓扑结构

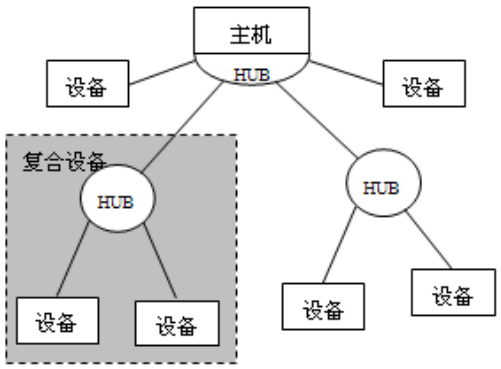

图 14 USB 物理总线的拓扑

USB 系统在硬件上主要分为 USB 主控制器和 USB 设备。USB 的主控制器和设备以星型方式排布，主控制器永远都处于中心节点的位置，而我们在日常生活中所用的 U 盘、USB 鼠标和键

[3] http://www.usb.org

盘、摄像头等等都是属于 USB 设备。USB 的主控芯片就是通过 Hub 的概念将众多的设备连接成一个星型网络，每个不同的 USB 设备在初始化阶段都会被分配一个特定的地址以供主机区别。既然 Hub 可以扩展出众多的 USB 端口，那么作为设备的 Hub 可以将这个星型网络向下进行扩展。USB 的拓扑结构如图 14 示。

6.1.1.2USB 的传输速率和传输类型

USB 的传输速率分为低速（Low speed）、全速（Full speed）和高速（High speed）三种模式，他们的波特率分别为 1Mbit/s、12Mbit/s 和 480Mbit/s。在 USB2.0 的规范中同时规定了这三种速度模式，并要求在主从设备的速度不同时，高速设备应该向低速设备靠拢。通常一个 USB 端口的速度上限是由硬件来决定的，通过软件的方式通过对程序的优化减少不必要的等待来使得真正的传输的字节数据速率达到最优。AT91SAM9260 所配备的两个 USB 端口都是属于全速端口，因此实现的传输波特率为 12Mbit/s.

USB 使用 4 种数据传输方式分别为控制传输（control）、中断传输（interrupt）、批量传输（bulk）和同步传输（synchronous）。**控制传输**是每个 USB 设备与主机通信中必须使用的一种传输方式，也是默认方式，主机通过控制传输来对设备枚举、配置、分配地址并协商下面所使用的传输方式，因此控制传输在整个对话过程中必须保证不会出错，否则显出混乱的系统便无法通过主机的绝对控制来恢复正常的通信秩序；**中断传输**用于小批量、非连续和不可预测何时会产生数据的场合，它只能单向的向主机发送数据，所以通常用在键盘、鼠标等 USB 设备中；**批量传输**则是成批量的传输大量数据，对时间上的要求并不是太高，通常用在打印机和扫描仪等设备中，而通常所说的 USB 移动磁盘也是使用该中传输方式进行文件数据传输；同步传输要求数据实时并且连续，它对数据的传输时间要求很高，主要应用于对数据实时性要求很高的语音业务传输类型。

6.1.1.3USB 的传输事务和数据包

一次 USB 传输是有一个或多个事务组成的，USB 主机和设备之间通过这些事务的里三角湖来进行通信。一个事务大多由主机发起，主机发送令牌包（token）后，视情况而定的传输数据包（data），该数据包的方向大小通常都是在令牌包中事先说明的，最后完成数据传输之后要由接受方是不同的情况发送握手信号（handshake）。以上的这些包都是符合 USB 传输数据的定义的，通过包头的 PID 来识别他们具体的类型。而通过将多个复杂的事务组成在一起又可以在 USB 上建立起新的传输协议或者说是子类，海量存储类（Mass Storage Class）就是其中的一种。

6.1.1.4USB 的端点概念

从主机发来的数据包根据它们不同的类型发往不同的 USB 端点。USB 端点是 USB 设备特有的一种硬件的接收发送单元。USB 主机与设备之间都是通过端点来传输数据的。端点是桥梁和纽带，不同的端点其传输数据的能力不同，适于不同的应用场合。在 Bulk Only Transportation 的海量存储实现规范中，使用了三个端点，他们分别是控制端点、数据输出端点和数据输入端

点。控制端点可以用于控制命令的输出，配置信息的输入和输出，它通常表示为 0 端点，这个端点的流向是双向的，不会出现 STALL 的状态，它负责设备的配置交互，最大信息包 64 字节。输入输出端点负责数据的输入、输出，他们都是单向的只能负责输出或者输入，最大信息包 64 字节(USB 全速设备)。

表 3 BOT 端点的属性

端点	传输类型	端点类型	传输方向	最大传输包
0	控制输入 控制输出	默认	输入 输出	64Byte
1	数据输出	普通	输出	64Byte
2	数据输入	普通	输入	64Byte

6.1.2 USB 设备的枚举过程

枚举过程定义为：当设备插入系统时，主机对设备进行配置，获取 USB 设备的各种描述(包括设备描述、配置描述、Bulk Only 数据接口描述、Bulk-In 端点描述、Bulk-Out 端点描述和字符描述，这些描述相互之间的关系如图 15 所示），这也就是 USB 设备为什么可以热插拔的原因。USB 设备可分为以下几类：显示器、通讯设备、音频设备、人机输入、海量存储……特定类的设备又可划分成子类。枚举使得主机可以精确定位 USB 设备类型，搜索相应的驱动程序。

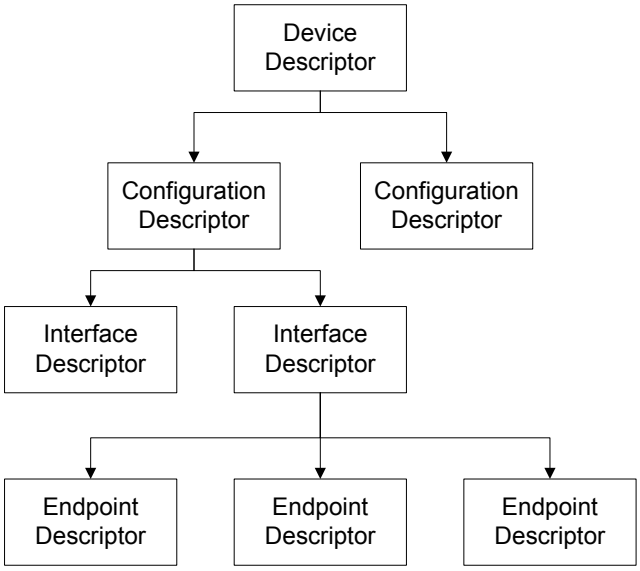

图 15 USB 描述的层次结构

为了更加清晰的演示在 USB 设备插入主机时的枚举过程，这里简要的引用 BUSHOUND 软件截取的一段 U 盘的枚举示例，并加以解释。

```
Device  Phase  Data                                                Description
------  -----  --------------------------------------------------  ----------------
  18.0  CTL    80 06 00 01  00 00 12 00                            GET DESCRIPTOR
  18.0  DI     12 01 00 02  00 00 00 40  fe 13 00 1f  10 01 01 02  .......@........
               03 01                                               ..
```

Device18 即代表该 U 盘在本地计算机上获得的 USB 通信的地址为 18，而后面的跟的 0 代表现在他们正在使用 0 端点进行通信。上面一行是计算机给 U 盘发送**设备描述（device description）**请求（CTL 代表控制传输方式），并规定设备应该返回 0x12 字节的数据。然后 U 盘回复了自己

的设备描述，其中带有信息包括USB版本、最大包大小、USB-IF分配的制造商ID、厂商分配的产品ID、设备发行时的设备号等等。

```
18.0  CTL    80 06 00 02  00 00 09 00                          GET DESCRIPTOR
18.0  DI     09 02 20 00  01 01 00 80  64                      .. .....d
```

主机请求**配置描述（configuration description）**，长度要求为9Bytes，设备返回的信息包括真正的配置描述长度、接口数、配置值、设备的最大消耗电流等。

```
18.0  CTL    80 06 00 02  00 00 20 00                          GET DESCRIPTOR
18.0  DI     09 02 20 00  01 01 00 80  64 09 04 00  00 02 08 06  .. .....d.......
             50 00 07 05  81 02 00 02  00 07 05 02  02 00 02 00  P...............
```

这次根据上次返回的真正的配置描述长度为0x20重新请求32字节的配置描述，这个配置描述实际上还包括了**接口描述（Interface Description）**、和输入数据端点、输出数据**端点描述（Endpoint Description）**。这些描述中间就规定了该设备是一个海量存储设备（Mass Storage Device）、使用的数据传输的方式（在移动存储设备中采用批量传输）。

```
18.0  CTL    80 06 00 03  00 00 02 00                          GET DESCRIPTOR
18.0  DI     04 03                                            ..
18.0  CTL    80 06 00 03  00 00 04 00                          GET DESCRIPTOR
18.0  DI     04 03 09 04                                      ....
18.0  CTL    80 06 03 03  09 04 02 00                          GET DESCRIPTOR
18.0  DI     1a 03                                            ..
18.0  CTL    80 06 03 03  09 04 1a 00                          GET DESCRIPTOR
18.0  DI     1a 03 35 00  42 00 38 00  33 00 30 00  35 00 30 00  ..5.B.8.3.0.5.0.
             30 00 30 00  30 00 45 00  45 00                  0.0.0.E.E.
```

这段主机开始询问设备的**字符串描述（String Description）**信息，其总往来了好几个回合，分别在问询交换的语言（一般都采用国际通用的英文）、设备的序列号5B8305000EE（由Unicode码的形式给出）。

```
18.0  CTL    00 09 01 00  00 00 00 00                          SET CONFIG
18.0  CTL    01 0b 00 00  00 00 00 00                          SET INTERFACE
18.0  CTL    a1 fe 00 00  00 00 01 00                          GET MAX LUN
18.0  DI     00                                               .
```

从右边的信息可以看到此时计算机大概已经知道这个具体是什么设备，用什么驱动来进行交互了，所以开始对设备进行配置：设置配置（Set Configuration）、设置接口（Set Interface）、获取最大逻辑单元数（Get Max LUN）。

至此通过0端点所传输的控制的控制过程，即枚举过程，都已经结束了。这里只是简要介绍了枚举过程中设备和计算机之间到底发生了什么事情，具体的这些描述的严格的格式可以参见附录和相关网站。

6.1.3 CBW命令包格式[1][4]

在完成U盘的枚举过程之后，识别出为Bulk-Only的Mass Storage设备，然后即进入Bulk-Only传输方式。在此方式下，计算机与U盘之间所有数据均通过Bulk-In和Bulk-Out来进行传输，不再通过控制端点（0端点）传输任何数据。

在这种传输方式下，有三种类型的数据在USB和设备之间传送，CBW、CSW和普通数据。CBW(Command Block Wrapper)是从USB Host发送到设备的命令，命令格式遵从接口中的bInterfaceSubClass所指定的命令块，这里为SCSI传输命令集。USB设备需要将SCSI命令从

CBW 中提取出来，执行相应的命令，完成以后，向 Host 发出反映当前命令执行状态的CSW(Command Status Wrapper)，Host 根据 CSW 来决定是否继续发送下一个 CBW 或是数据。Host 要求 USB 设备执行的命令可能为发送数据，则此时需要将特定数据传送出去，完毕后发出CSW，以使 Host 进行下一步的操作。USB 设备所执行的操作可用图 16 描述。

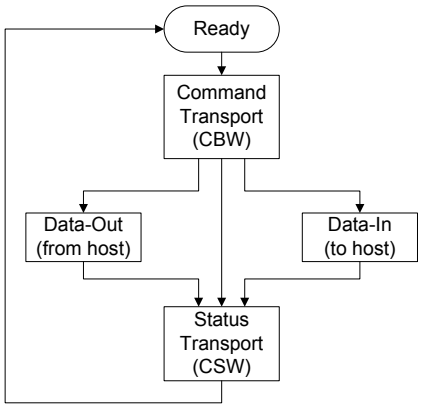

图 16 命令、数据、状态流程图

在流程图中包括了三种形式的数据包，USB 的设备层统一将他们看成数据，不加区分的进行传输，而在功能层上才开始区分这三种不同形式、不同功能的数据包。

表 4 CBW 格式[9][4]

字节	值
0-3	0x43425355
4-7	dCBTag
8-11	要传输的数据长度
12	bmCBWFlags
13	bCBWLUN
14	命令字的长度
15-30	CBWCB

如表 4 CBW 命令包格式，其中 0x43425355 表示当前发送的是一个 CBW；dCBWTag 的内容需要原样作为 dCSWTag 再发送给 Host；bmCBWFlags 反映数据传输方向；bCBWLUN 为零；bCBWCBLength 为本次命令字的长度；CBWCB 即为真正的传输命令集的命令。得到一个 CBW 后，解释出 CBWCB 中所代表的命令，并向 USB Hosting 发送 CSW 来反映命令执行的状态。

dCSWSignature 的内容为 0x53425355，dCSWTag 即为 dCBWTag 的内容，dCSWDataResidue 还需要传送的数据，此数据根据 dCBWDataTransferLength 减去本次已经传送的数据得到。Host 端根据此值决定下一次 CBW 的内容，如果没有完成则继续；如果命令正确执行，bCSWStatus 返回 0 即可。按这个规则组装好 CSW 后，通过 Bulk－In 端点将其发出即可。

6.1.4 SCSI 命令格式[19]

USB 是随机存取、基于扇区存储的设备，接口遵循 SCSI-2 标准的直接存取存储设备协议。在 6.1.3 节中介绍的 BOT（Bulk-Only Transportation）协议的 CBW 数据包中有一个长度为 6 到 12 不等的字段 CBWCB，其传输的就是 SCSI 命令。

SCSI 命令集是一套能够完成对一般外接存储系统设备的存取、配置、检查等任务的完备指令集。SCSI 拥有的这套指令集有很多指令，不同的版本分别用在光驱、硬盘、软驱等设备的控制中，表 5 列出的 7 条指令已经可以完全实现 USB 存储设备的操作了，这一特性大大减小了设备设计的难度与复杂度。

表 5 MSC 基本 SCSI 命令集

命令	描述
INQUIRY	查询设备的详细信息，如生产厂家、序列号等
PREVENT-ALLOW MEDIUM REMOVAL	禁止或允许取走 U 盘
READ CAPACITY(10)	读取 Flash 容量，扇区大小以及总扇区数
READ(10)	***输入要读取的数据，从 Flash 读出***
REQUEST SENSE	请求判断，用于发生错误时系统自我纠正
TEST UNIT READY	测试单元准备，电脑通过此命令保持 U 盘一直处于激活状态
WRITE(10)	***提供 Flash 地址，向 Flash 写数据***

在这 7 条指令中，READ10 和 WRITE10 是两个十分重要的指令，文件的读写等各种对磁盘的操作基本上全部由这两条指令来完成的。故在此要简要的介绍一下 READ10 命令的详细格式，因为后面的内容会讨论到这两个命令。

表 6 READ10 Command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (28h)							
1	RDPROTECT			DPO	FUA	Reserved	FUA_NV	Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
5								(LSB)
6	Reserved			GROUP NUMBER				
7	(MSB)	TRANSFER LENGTH						
8								(LSB)
9	CONTROL							

在该命令中 0 字节 0x28 是 READ10 的编码，1、6、9 三个字节是用来对数据传输做一些具体的控制，但是一般不会使用。2~5 这 4 个字节作为一个 32bit 长整型数据指定了逻辑块地址，它的以扇区为单位，所以在寻址能力上，USB 的存储器可以支持大到 2TG 的容量空间（扇区的大小规定为 512Byte）。7~8 的 2 个字节作为一个 16bit 数据指定传输数据的大小，同样这个值的单位也是扇区，虽然该值最大可以达到 255，但是在现实应用中，通常不会大于 0x0008，即最多一次传输 4KB 的数据。

表 7 WRITE10 Command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (2Ah)							
1	WRPROTECT			DPO	FUA	Reserved	FUA_NV	Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
5								(LSB)
6	Reserved			GROUP NUMBER				
7	(MSB)	TRANSFER LENGTH						
8								(LSB)
9	CONTROL							

如果要实现 WRITE10 实际上在 READ10 的基础上只要将传输方向加以改变，而上述的逻辑

块地址和传输长度的定义仍然是我们最关心的两个参数。

通过分析，可以看到在操作系统对一个磁盘做读取操作时，底层硬件上看到的仅仅是系统所读数据的地址和大小，而对数据的内容是什么不加以关心。即使是一般的删除操作，操作系统所做的工作也就是向要被删除数据的区域写入一串零而已。这就给后面文件系统中目录数据的识别带来了难度。这部分的内容在第八章有详细的介绍。

最后给出 BOT 传输开始阶段的示例，紧接在 USB 枚举后面数据传输的过程。

表 8 CBW 数据包传输示例

Device	Phase	Data	Description
18.2	DO	55 53 42 43 d0 27 0c 82 24 00 00 00 80 00 06 12	USBC.'..$.....
		00 00 00 24 00 00 00 00 00 00 00 00 00 00 00 00	...$..........
18.1	DI	00 80 00 01 1f 00 00 00 4b 69 6e 67 73 74 6f 6e	........Kingston
		44 61 74 61 54 72 61 76 65 6c 65 72 20 32 2e 30	DataTraveler 2.0
		50 4d 41 50	PMAP
18.1	DI	55 53 42 53 d0 27 0c 82 00 00 00 00 00	USBS.'.....
18.2	DO	55 53 42 43 d0 27 0c 82 08 00 00 00 80 00 0a 25	USBC.'.....%
		00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
18.1	DI	00 1d cf ff 00 00 02 00	.....
18.1	DI	55 53 42 53 d0 27 0c 82 00 00 00 00 00	USBS.'.....
18.2	DO	55 53 42 43 d0 27 0c 82 00 02 00 00 80 00 0a 28	USBC.'.....(
		00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00	.....
18.1	DI	fa 33 c0 8e d0 bc 00 7c 8b f4 50 07 50 1f fb fc	.3.....P.P...
		bf 00 06 b9 00 01 f2 a5 ea 1d 06 00 00 be be 07	.....
		b3 04 80 3c 80 74 0e 80 3c 00 75 1c 83 c6 10 fe	...<.t..<.u....
		cb 75 ef cd 18 8b 14 8b 4c 02 8b ee 83 c6 10 fe	.u......L.....
		cb 74 1a 80 3c 00 74 f4 be 8b 06 ac 3c 00 74 0b	.t..<.t.....<.t.
		56 bb 07 00 b4 0e cd 10 5e eb f0 eb fe bf 05 00	V.......^.....
		bb 00 7c b8 01 02 57 cd 13 5f 73 0c 33 c0 cd 13	......W.. s.3...

上面三个框圈住的分别执行了 INQUIRY、READ CAPACITY10 和 READ10：

INQUIRY 命令	返回 Kingston DataTraveler 2.0PMAP；
READ CAPACITY10	返回磁盘的大小：0x1DCFFF 个扇区，扇区大小：0x0200Byte
READ10	读取逻辑地址为 0，长度为 1 个扇区的数据（数据太多，图中仅显示开头的部分数据）。之后，操作系统会逐个读取扇区，包括了 MBR、FAT1、FAT2 和根文件目录。

6.2 ARM9 中 USB 的具体实现

本小节详细介绍了系统如何实现 USB 协议，包括了主机驱动和设备驱动两个部分，这两个部分有着很大的工作量，代码的实现工作也很大程度上依赖于开源项目。芯片制造商 ATMEL 公司在硬件的软件化包装上也给编程实现带来了很大的便利。他所提供的 at9lib_softpack_1.5 很好的将对硬件的操作封装成函数的调用，在开发文档给予了很详细的说明，在一定程度简化开发过程，提高嵌入式应用程序的可重用性。

如图 17 所示，在现有的基础上进行开发，只有在保证主机程序和设备驱动可以分别正确的运行才有成功的将二者结合起来的可能。

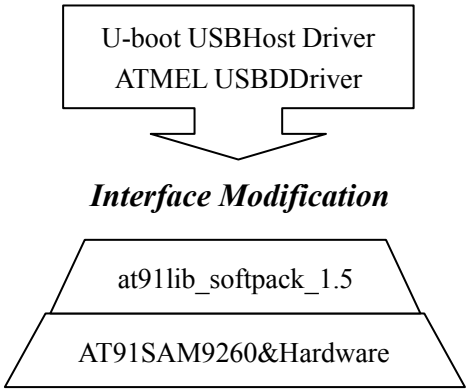

图 17 软硬件层次架构图

6.2.1 MSDDriver 的工作原理

本模块原型来自于 ATMEL 公司网站，作为该款芯片的一个示例程序，它可以将核心板上的 64M 内存开发出来，做成一个大小不超过 60M 的移动磁盘。当然作为一个磁盘，必须以非易失性存储介质存储数据，而这里使用 SDRAM 只是为了一个演示效果。但正是因为这里内存的时候，可以考虑在连接主从驱动时，将内存作为一个简单的 cache 来使用，以减少真正从 U 盘中读取数据的机会。

本部分要做的内容即将以上所述的原理予以实现，所以同样分成两个部分考虑，分别是枚举和数据传输。

1、USB 设备驱动的枚举过程

当设备插入计算机的一瞬间，便需要进入这个状态。所以应用程序启动之后的操作步骤如下：

(1). 对 USB 进行初始化；

(2). 拉高 D+数据线，通知计算机有 USB 设备接入，可以开始枚举；

(3). 等待 SET CONFIGURATION 请求，该请求标志着枚举的成功，设备和计算机之间的交互可以进入下一个阶段。

第 3 个等待的步骤是由 While 循环完成，不断的检查是否已经得到 SET CONFIGURATION 请求。在 While 循环过程中，通过中断处理 USB 总线上的其他控制请求。

2、MSDDriver 状态机实现

枚举任务完成之后，对 USB 来说系统便进入了纯数据的传输阶段，但是这些数据又遵循了 MSD 协议，满足 CBW、CSW 等格式的要求。所以在实现普通 U 盘功能时需要借助状态机实现图 16 所示的命令、数据和状态的流程图。

如图 18 所示，将状态改变及相关操作描述如下：

(1). 初始化后的默认状态为 READ_CBW，此状态一直要保持到 USB 总线上出现数据传输；

(2). 在 WAIT_CBW 状态，顾名思义为等待 CBW 传输完毕后转入下一状态；

(3). 在 PROCESS_CBW 状态，需要检查接收到的 CBW 数据是否满足上文中 CBW 的标准，即以 0x43425355 开头、长度为 31 字节。然后进一步检查 SCSI 命令的合法性，最后根

据具体的命令调用不同的函数执行。该状态一直要保持到 CBW 请求的数据发送完毕或接受完毕才可以转到 SEND_CSW。

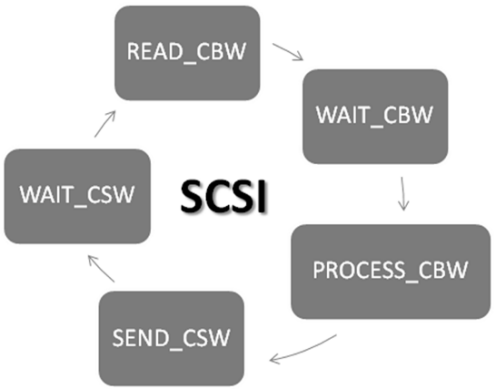

图 18 MSDDriver 的数据传输阶段状态图

(4). SEND_CSW，在此状态检查 PROCESS_CBW 的执行结果，根据执行情况选择相应的状态发送给主机；

(5). WAIT_CSW 等待 CSW 状态传输完毕，转到下一个会话。

6.2.2 USBHostDriver 的工作原理

ARM 芯片中的 USB 主口服从 OHCI（Open Host Control Interface）的标准，OHCI 主要面对的对象为 USB2.0 的全速设备和低速设备。用于 USB 的 OHCI 规范是 USB 主机控制器的寄存器描述，该标准的使用使得硬件制造商在满足 USB 协议的同时，面对用户也有了一个统一的通过寄存器描述的接口，只要改动 OHCI 驱动寄存器的偏移地址便可以使 USB 主机驱动成功的移值。OHCI.C 提供了非常方便的外部接口以供调用。

USB 的主机驱动分成三个层次，由上到下分别为：Usb_storage.c、Usb.c、Usb.ohci.c

其中 Usb.c 提供了两个非常重要的接口函数：

int submit_bulk_msg(*dev, pipe, *buffer, transfer_len);

int submit_control_msg(*dev, pipe, *buffer, transfer_len, devrequest *setup);

它们分别实现批量传输和控制传输，有了这两个关键的函数才使得 Usb_storage.c 中各种复杂的功能得以实现。

主机对 U 盘的访问和 U 盘响应主机的步骤完全一致，只不过一个处于主动的地位而另一个被动。首先同样是对 USB 进行初始化，然后扫描设备，这里的 USB 主控器默认是一个 Hub，所有的 USB 设备如果连接到 USB 接口，那么它们以树形的方式结合起来。驱动中含有大量的结构体定义，以对设备的属性中包含的各种参数进行存储。

6.3 文件透明传输的实现（主从驱动互联）

在简要的介绍完主机驱动和从机驱动的实现，需要对如何将两者进行合并，在同一个系统中系统工作，对 USB 数据流进行分析、转发等工作做一些详细的介绍。因为不管是主设备还是从设备，整个通信过程必须分为两个部分，首先是枚举，然后是数据传输，所以在联合主从驱

动时也必须对这两部分分别给予考虑。

1、枚举阶段

枚举阶段主要是对设备的各种属性、参数的问询，此过程因为实时性不是很强，故可以分开进行，即先对 U 盘进行枚举，将枚举完成所获得的结果赋给加解密设备的 USB 从机驱动的相关变量，使得不论是什么 U 盘，通过加解密设备之后在计算机上看到的能和直接连接计算机的效果相同。

```
typedef struct block_dev_desc {
    int         if_type;        /* type of the interface */
    int         dev;            /* device number */
    unsigned char part_type;    /* partition type */
    unsigned char target;       /* target SCSI ID */
    unsigned char lun;          /* target LUN */
    unsigned char type;         /* device type */
    unsigned char removable;    /* removable device */

    unsigned int   lba;         /* number of blocks */
    unsigned int   blksz;       /* block size */
    unsigned char vendor [40+1];    /* IDE model, SCSI Vendor */
    unsigned char product[20+1];    /* IDE Serial no, SCSI product */
    unsigned char revision[8+1];    /* firmware revision */
    unsigned int   (*block_read)(int dev,
                        unsigned int start,
                        unsigned int  blkcnt,
                        unsigned int *buffer);
}block_dev_desc_t;
```

图 19 主机驱动中设备的信息描述结构体定义

如图 19 所示，为加解密设备对 U 盘完成枚举后能够得到的设备信息，这些信息就是上文中所说必须转移到 USB 从机驱动中的，使计算机同样能看到加解密设备上的 USB 存储器的参数。

2、数据传输阶段

如表 5 所示，其中的两个命令 READ10、WRITE10 负责了全部的数据传输的任务，所以在数据传输阶段要做的工作便将是这些命令传递到 USB 存储器上。USBDDrive 的状态机在读到 READ10 命令时会调用 SBC_Read10 函数，在其处理 READ10 时也用到了状态机的形式，因此在对 SBC_Read10 改写时，最大可能的遵循了原来的结构，保证最小的改动，它的实现如图 20 所示：

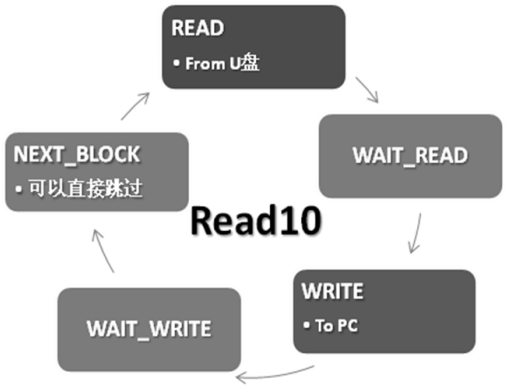

图 20 透明传输 READ10 状态图

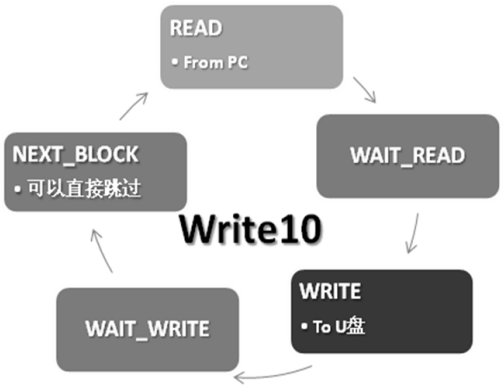

图 21 透明传输 WRITE10 状态图

(1). READ：调用 submit_bulk_msg 函数，想 U 盘下达读取计算机要求的数据命令。因为

USB 主机驱动的函数会阻塞至传输完成，所以在完成传输之后，程序会立即判断该数据是否需要加密，如果需要执行加密子程序。

(2). WAIT_READ：等待U盘返回数据传输完成，实际上这个状态一跳而过，因为上一个状态 READ 调用的读取函数一直会阻塞到数据传输结束。

(3). WRITE：将内存中的数据发送给计算机，该过程并不阻塞。调用完相关函数立即转到 WAIT_WRITE 状态。

(4). WAIT_WRITE：等待 USB 数据传输完毕。

(5). NEXT_BLOCK：因为如果一次性计算机要求的数据量太大，而超过了USB的一次性传输的能力，则数据会分为多个 BLOCK 分别进行传输。通常情况该状态不会使用到，而是由上一状态直接开始下一个 READ10 的命令周期。

WRITE10 命令的处理过程与 READ10 类似，此不再赘述，状态如图 21 所示。

第七章　内存的分配、映射及其管理

对一个存储设备来说，传输速率是个至关重要的性能指标，提高数据传输的速率的手段通常有两个：一是通过不断的优化程序的方式来提高速度，这种方法在程序架构和算法确定之后基本上没有多少提升的空间；二是通过减少对存储器的访问的机会。本章要讨论的问题就是怎样通过开辟内存的方式来在现有的基础上提高数据传输的速度。

众所周之，在计算机系统中 CPU 速度最快，而计算机的整体性能并不仅仅取决于 CPU 的处理能力，还在于访问存储器的平均时间。如果一个快速的 CPU 配上了一个很慢的硬盘，那么大量的可以用在计算上的时间都在 CPU 等待硬盘相应数据中度过了。同样的对于嵌入式系统而言，在这里的传输速度很大程度的取决于对 U 盘的访问次数。例如，计算机要从 U 盘中读取数据，加解密设备收到命令之后要对 U 盘进行访问，从 U 盘那里先取得这串数据，然后再转发给计算机。在这之间如果不算控制和命令解析的处理时间，整体的传输速度是计算机直接访问 U 盘的时间还要加上命令从计算机到加解密设备和数据从加解密设备传回计算机的两个时间。因此真实的传输速率大概会成为直接传输速度的一半左右。但是如果计算机下次再次对该数据进行访问，那么从 U 盘重新读取该数据还是直接从内存中获取上次驻留在内存中的那个部分呢？显然，如果从内存中直接读取，速度会快很多。与直接从 U 盘读取相比，采用内存的方式不仅省去读 Flash 的时间，而且将读 Flash 的时间变成了读 SDRAM 的时间，从理论上来讲，执行速度甚至会快于直接读取 U 盘。

基于这样的思想，怎样在核心板上 64M 内存空间中合适的地方开辟合适大小的内存以供中转；以什么样的方法将这个内存映射到各种不同的 USB 存储设备；在什么时候将该内存中的数据写入到 U 盘中等等都成为本章要解决的问题。

7.1 内存的分配

在 64M 的内存中存放的有当前运行的程序代码、各种形式的变量、变量的初始化值和堆栈。那么这四个元素到底分布在内存的什么部位呢？首先可以肯定的是在连接完之后，这些内存空间的使用是已经可以确定的。只有弄清楚他们的位置，才能确定哪里可以腾出多少空间供数据缓存使用。

表 9 程序资源消耗表

数据类型	起始地址	末地址	大小
.text	0x2000 0000	0x2001 05d3	0x105d4
.rodata	0x2001 05d4	0x2001 597f	0x53ac
.data	0x2001 5980	0x2001 e3ff	0x8a80
.bss	0x2001 e400	0x2004 b7f7	0x2d3f8
COMMON	0x2004 b7f8	0x2025 5104	0x20990d
stack	*0x2400 0000*		

因此，必须对工程文件夹下每次 Make 都会生成的 toto.map 文件[27]进行分析，得到的结果如

表 9 所示。表中地址都是从片外的 16M 内存 SDRAM 的首地址 0x2000 0000 开始的。其中.text 内容表示为程序的代码段，共 65K 左右；.rodata (read only data)为程序中所申明的常量，大小 1K 左右；.data 段为所有的带有初值的局部变量，共 2K 左右；.bss 段用于存放没有初始化值的静态变量，该部分在程序加载时会自动填零，共 180K 左右；COMMON 段存放全局变量，和一些大数组，大小约为 2M。所以从 SDRAM 的 0 地址开始的大 2M 多的空间需要腾出来作为程序的使用范围，具体的范围为：0x2000 0000~0x2025 5104.

在程序运行中所遇要的不仅仅是代码和各种变量，还包括调用函数时所遇要用到的堆栈，堆栈的位置放在SDRAM的最后，与其他段不同的是，它的地址的增长方向是相反的，从高地址一直到低地址。

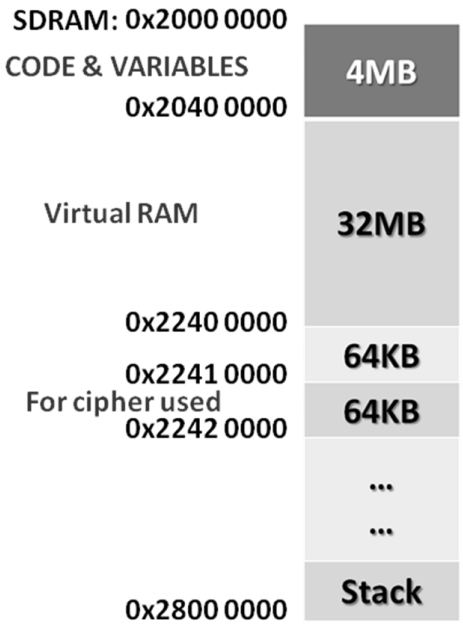

图 22 内存分配示意图

综上所述，考虑以后程序的改动和各种可能出现的变更，如图 22 所示将前面的 4M 和最后 28M 空间腾出来，剩下的 32M 内存作为数据的中转存储地使用。

7.2 内存的映射及管理

由于系统对文件传输的速度影响越小越好，所以在内存管理上采用最简单的机制，即类似于 Cache 地址映射中的 Direct mapping 的概念[25]——主存储器被分为一页一页的，每页的大小与 Cache 的大小相同，具体的实现描述如下：

32M 内存在以 512 大小的扇区为单位寻址时的地址空间可以使用 16bit 表示，而在 USB 传输中的 SCSI 命令中的地址空间为 32bit，所以对每一个内存中的扇区都必须使用一个额外的 16bit 的变量存储该扇区的原始地址的高 16 位，这样才能准确无误的将扇区地址和 USB 存储器的地址一一对应。

因此开辟数组 table 记录各个扇区的高 16 位地址，table 的下标使用扇区在内存中的偏移地址表示。初始化时一定要将其全部置为 0xffff，以区分当前该扇区还没有和存储器映射过。

```
/*
 * virtual ram
 */
#define SIZE_OF_TABLE   0x10000
static unsigned short table[SIZE_OF_TABLE];
```

图 23 table 数据申明的代码段

假设当前有数据要存入内存，它的存储器地址为 usb_addr，所以与之对应的内存映射地址应该为：

ram_addr=(usb_addr & 0xffff) + SDRAM_OFFSET+CODE_SIZE

但是也就产生了一个问题：USB 传输中可以一次传输 128 个扇区（64KB），那么如果这段数据的首地址映射到 32M 的末尾，而首地址则会对应到 32M 的开头，这样就给底层调用 USB 数据的接受函数造成了一定的麻烦，因为一旦对 USB 模块的底层代码进行改动就意味着程序的通用性变得很差，一旦换了硬件平台就很难再对系统进行移植。因此在 32M 内存后面再分配 64K 的缓冲区域，以提高兼容性。

此外，内存是为明文准备的。当计算机读数据时，可以如下进行操作：

(1). 首先从磁盘中将密文读入内存；

(2). 然后在当前内存地址解密数据；

(3). 解密之后再将数据发送给计算机。

这样可以保证内存中存储的仍然为明文，但是当计算机写数据时问题便出现了：

(1). 首先将明文写到内存中；

(2). 然后在当前内存地址加密数据；

(3). 将加密后的数据写到磁盘中。

从上面的步骤看，似乎没有错，但是所有的操作结束之后，内存中存储的竟然是密文，这样就将内存中的数据搞混了，也没有遵循“内存是为明文准备的”这条规定，下次读取该数据时便会出现错误。因此在上面开辟的 64K 用于跨区的内存后再开辟一块用于加密数据的 64K 内存来解决这个问题，从而将计算机写数据的步骤更改为：

(1). 首先将明文写到内存中；

(2). 然后将该数据复制到加密专用的地址；

(3). 在当前内存地址加密数据；

(4). 将加密后的数据写到次盘中。

多了一个 copy 的步骤。

为了降低实际应用中多次拷贝数据给系统带来的时间开销，真正使用的内存复制函数并不是 C 标准库的 memcpy 函数，而是快速的 fast_memcpy 函数：

```
void * fast_memcpy(void * dest,const void *src,unsigned int count)
{
    unsigned int *tmp = (unsigned int *) dest, *s = (unsigned int *) src;
    count=count>>2;
    while (count--)
        *tmp++ = *s++;

    return dest;
}
```

该 fast_memcpy 利用了 32 位处理器每次处理 32 位数的特点，每次复制的数据大小为 4-byte 而非 C 标准库中提供的 1 个字节的长度。从理论上可以将同样长度的数据复制时间变为原来的 1/4。

但是此处 fast_memcpy 的时候有一个限制：src 和 dest 两个指针的值必须满足 4 的倍数，因为在 AT91SAM9260 硬件中规定 4-byte 寻址必须向 4-byte 对齐，否则会出现读到的数据不是想要的数据的结果。例如：

0	1	2	3	4	5	6	7
03	53	49	3F	D1	15	B8	3B
		A		B			

一个 4-byte 的长整型的指针 p 的位置在 A 处，那么*p 的值并不是所想像的 0x15D13F49，通过实验可以发现实际上*p 的值却是 0x53033F49，这就是因为 p 的地址没有对齐 4-byte；而如果 p 指向的是 B，那么*p 得到的值理所当然为 0x3BB815D1。

不过幸运的是，在 USB 进行数据读取时所有的数据都是以 512-byte 为单位的，即数据的地址为 512 的倍数，显然更是 4 的倍数了。

第八章　文件目录簇的识别

目前在大容量 U 盘中普遍采用的文件系统格式为 FAT32，而 FAT16 及 FAT12 也因为 U 盘容量的不断扩展早早的退出了历史的舞台。因此本文选取 FAT32 作为研究对象，实现该文件系统下的文件级别的数据加密。但是就 USB 的存储设备而言并不存在文件系统的概念，它所看到的只能是地位相同的 RAW DATA，因此这也给基于 USB 的文件级别的加密带来了挑战。

8.1 FAT32 文件系统简介[5][8]

对于 FAT32 文件系统，硬盘上的数据按照不同的特点和作用大致可分为 5 部分：MBR 区(主引导记录区)、DBR 区(DOS 引导记录区)、FAT 区(文件分配表区)和 DATA 区(数据区)。其中，MBR 区由分区软件创建，而 DBR 区、FAT 区和 DATA 区由高级格式化程序创建。这 4 个区域共同作用才有利于整个硬盘的管理。

图 24 FAT 文件系统结构示意图

8.1.1　主引导记录区（MBR）

MBR 在磁盘中的地位是首要的，它的缺失对磁盘中存储的数据而言可能是灾难性的。一个 MBR 的大小为 512 字节，以 0x55AA 作为结束的标志，这一标志通常也被用于判定该磁盘所使用的是否是 DOS 文件系统。MBR 中存储的主要就是启动代码和磁盘的分区表。

在计算机启动时 BIOS 会选择从哪个盘启动，一但 BIOS 将控制权交给磁盘之后，第一件事就是执行位于磁盘 0 地址的前 446 个字节的启动代码，这段代码对本问的研究没有任何影响，故不做详细的介绍。

然后从偏移为 0x1BE 的地址开始就是 16 个字节的分区表，分区表共有四个共 64 字节。在 Windows 操作系统下的可移动磁盘是无法分区的，即使通过强行的方法进行分区，Windows 也只对第一个分区做识别，后面的分区无法访问。

表 10 MBR 分区表

Partition Table Entry #1		
446	80 = active partition	80
447	Start head	1
448	Start sector	1
448	Start cylinder	0
450	Partition type indicator (hex)	0B
451	End head	121
452	End sector	32
452	End cylinder	502
454	Sectors preceding partition 1	32
458	Sectors in partition 1	1953760

表 10 所示便是一块 U 盘的主分区表，在这里规定了磁盘的开始和结束的磁头、扇区和柱

面，这些参数实际上没有真实的作用，只是习惯上沿用以前盘片式磁盘的习惯。有用的参数有：Partition type indicator（指明了该分区中使用的操作系统），本文简单的检查是否为我们所需要的 FAT32 的代码 0x0B；Sectors preceding partition 1（指明真正分区的 DBR 开始的逻辑块地址），本文通过该数值找到 DBR；Sectors in partition 1（指明该分区的大小，以扇区为单位），在这里也可以不用。

值得一提的是，正是因为 Windows 下的 U 盘只有一个分区，所以很多时候 MBR 可以不存在的，磁盘的 0 扇区直接存储 DBR 也可以正常工作，所以在本设计中也加入了对这一特殊情况的考虑。

8.1.2 DOS 引导记录区（DBR）

与 MBR 不同，DBR 中存储着大量的在识别目录簇时可以使用到的数据。DBR 区（DOS BOOT RECORD）即操作系统引导记录区的意思，通常占用分区的第 0 扇区共 512 个字节（特殊情况也要占用其它保留扇区，我们只讨论第 0 扇）。在这 512 个字节中，其实又是由跳转指令，厂商标志和操作系统版本号，BPB(BIOS Parameter Block)，扩展 BPB，OS 引导程序，结束标志几部分组成，如表 11 所示。

表 11 FAT32 分区上 DBR 中各部分的位置划分

字节位移	字段长度	字段名
0x00	3 个字节	跳转指令
0x03	8 个字节	厂商标志和 os 版本号
0x0B	53 个字节	BPB
0x40	26 个字节	扩展 BPB
0x5A	420 个字节	引导程序代码
0x01FE	2 个字节	有效结束标志

这里详细介绍如何通过读取该扇区获得簇大小和 FAT1、FAT2、跟目录的地址。

表 12 部分 BPB 值（BIOS Parameter Block）

OFFSET	TITLE	VALUE
B	Bytes per sector	512
D	Sectors per cluster	8
E	Reserved sectors	38
10	Number of FATs	2
FAT32 Section		
24	Sectors per FAT	1905

如表 12 所示，它是从 Winhex 软件中截取出来的截屏，显示的是一只 U 盘的几个属性，其中偏移地址为 0xD 的说明了簇的大小，这里为 8 个扇区，即 4KB；一个重要信息为保留的扇区数，这个保留的意思就是指 FAT1 之前的扇区的个数（从 DBR 的开始算起）；最后的偏移为 0x24 的 FAT 的扇区数可以利用了确定 FAT2 以及跟目录的位置。

FAT1_addr=Reserved_sectors+DBR_addr
FAT2_addr=FAT1_addr+Sectors_per_FAT;
root_directory_addr=FAT2_addr+ Sectors_per_FAT;

8.1.3 文件分配表区（FAT）

FAT 用于存储磁盘中大于一簇的文件的跨簇存储的信息，采用链表的方式实现了文件的连续和不连续的存储，同时 FAT 会存有相同的两份，以供其中一个出现错误时恢复用。

8.1.4 数据区（DATA）

FAT32 文件系统的目录结构其实是一颗有向的从根到叶的树，这里提到的有向是指对于 FAT32 分区内的任一文件（包括文件夹），均需从根目录寻址来找到。可以这样认为：目录存储结构的入口就是根目录。

FAT32 文件系统根据根目录来寻址其他文件（包括文件夹），故而根目录的位置必须在磁盘存取数据之前得以确定，这一工作已经在前面对 DBR 分析的过程中完成。

FAT32 文件系统的一个重要思想是把目录（包括跟目录）当作一个特殊的文件来处理。FAT32 分区中所有的目录，实际上可以看作是一个存放其他文件（目录）入口参数的数据表。所以目录的占用空间的大小并不等同于其下所有数据的大小，但也不等同于 0。通常是占很小的空间的，可以看作目录文件是一个简单的二维表文件。其具体存储原理是：

不管目录文件所占空间为多少簇，一簇为多少字节。系统都会以 32 个字节为单位进行目录文件所占簇的分配。这 32 个字节以确定的偏移来定义本目录下的一个文件(或文件夹)的属性，实际上是一个简单的二维表。

这 32 个字节的各字节偏移定义如

表 13 FAT32 短文件目录项 32 个字节的表示定义

字节偏移(Hex)	字节数	定义	
0x00~0x07	8	文件名	
0x08~0x0A	3	扩展名	
0x0B*	1	属性字节	00000000(读写)
			00000001(只读)
			00000010(隐藏)
			00000100(系统)
			00001000(卷标)
			00010000(子目录)
			00100000(归档)
0x0C	1	系统保留	
0x0D	1	创建时间的 10 毫秒位	
0x0E~0x0F	2	文件创建时间	
0x10~0x11	2	文件创建日期	
0x12~0x13	2	文件最后访问日期	
0x14~0x15	2	文件起始簇号的高 16 位	
0x16~0x17	2	文件的最近修改时间	
0x18~0x19	2	文件的最近修改日期	
0x1A~0x1B	2	文件起始簇号的低 16 位	
0x1C~0x1F	4	表示文件的长度	

对表 13 中的一些取值进行说明：

(1). 对于短文件名，系统将文件名分成两部分进行存储，即主文件名+扩展名。0x0~0x7 字节记录文件的主文件名，0x8~0xA 记录文件的扩展名，取文件名中的ASCII码值。不记录主文件名与扩展名之间的“.”，主文件名不足 8 个字符以空白符（20H）填充，扩展名不足 3 个字符同样以空白符（20H）填充。0x0 偏移处的取值若为 00H，表明目录项为空；若为 E5H，表明目录项曾被使用，但对应的文件或文件夹已被删除（这也是误删除后恢复的理论依据）。***文件名中的第一个字符若为“.”或“..”表示这个簇记录的是一个子目录的目录项。“.”代表当前目录；“..”代表上级目录（和我们在 dos 或 windows 中的使用意思是一样的，如果磁盘数据被破坏，就可以通过这两个目录项的具体参数推算磁盘的数据区的起始位置，猜测簇的大小等等，故而是比较重要的）***

(2). 0xB 的属性字段：可以看作系统将 0xB 的一个字节分成 8 位，用其中的一位代表某种属性的有或无。这样，一个字节中的 8 位每位取不同的值就能反映各个属性的不同取值了。如 00000101 就表示这是个文件，属性是只读、系统。该字段的用途在长文件名的表示中还会用到。

(3). 0xE~0xF 和 0x16~0x17 中的时间=小时*2048+分钟*32+秒/2。得出的结果换算成 16 进制填入即可。也就是：0x16 字节的 0~4 位是以 2 秒为单位的量值；0x16 字节的 5~7 位和 0x17 字节的 0~2 位是分钟；0x17 字节的 3~7 位是小时。

(4). 0x10~0x11 和 0x18~0x19 中的日期=(年份-1980)*512+月份*32+日。得出的结果换算成 16 进制填入即可。也就是：0x18 字节 0~4 位是日期数；0x18 字节 5~7 位和 0x19 字节 0 位是月份；0x19 字节的 1~7 位为年号，原定义中 0~119 分别代表 1980~2099，目前高版本的 Windows 允许取 0~127，即年号最大可以到 2107 年。

(5). 由于 FAT32 可寻址的簇号为 32bit。所以系统在记录文件（文件夹）开始簇地址的时候也需要 32 位来记录，FAT32 启用目录项偏移 0x12~0x13 来表示起始簇号的高 16 位。偏移 0x1A~0x1B 表示起始簇的低 16 位。

以上介绍的仅仅是对短文件名的支持，最多只支持 8 个字符，因此在 FAT32 中规定长文件名依然记录在目录项中。为了低版本的 OS 或程序能正确读取长文件名文件，系统自动为所有长文件名文件创建了一个对应的短文件名，使对应数据既可以用长文件名寻址，也可以用短文件名寻址。不支持长文件名的 OS 或程序会忽略它认为不合法的长文件名字段，而支持长文件名的 OS 或程序则会以长文件名为显式项来记录和编辑，并隐藏起短文件名。当创建一个长文件名文件时，系统会自动加上对应的短文件名，其一般有的原则：

(1). 取长文件名的前 6 个字符加上"~1"形成短文件名，扩展名不变。

(2). 如果已存在这个文件名，则符号"~"后的数字递增，直到 5。

(3). 如果文件名中"~"后面的数字达到 5，则短文件名只使用长文件名的前两个字母。通过数学操纵长文件名的剩余字母生成短文件名的后四个字母，然后加后缀"~1"直到最后。

长文件名的实现有赖于目录项偏移为 0xB 的属性字节，当此字节的属性为：只读、隐藏、系统、卷标，即其值为 0FH 时，操作系统会认为该目录项正在为一个长文件名服务。系统将长

文件名以 13 个字符为单位进行切割，每一组占据一个目录项。所以可能一个文件需要多个目录项，这时长文件名的各个目录项按倒序排列在目录表中，以防与其他文件名混淆。长文件名中的字符采用 unicode 形式编码，每个字符占据 2 字节的空间。其目录项定义如表 15。

表 14 FAT32 长文件目录项 32 个字节的表示定义

字节偏移(Hex)	字节数	定义		
0x00	1	属性字节位意义	7	保留未用
			6	1 表示长文件最后一个目录项
			5	保留未用
			4	顺序号数值
			3	
			2	
			1	
			0	
0x01~0x0A	10	长文件名 unicode 码①		
0x0B	1	长文件名目录项标志，取值 0FH		
0x0C	1	系统保留		
0x0D	1	校验值(根据短文件名计算得出)		
0x0E~0x19	12	长文件名 unicode 码②		
0x1A~0x1B	2	文件起始簇号(目前常置 0)		
0x1C~0x1F	4	长文件名 unicode 码③		

系统在存储长文件名时，总是先按倒序填充长文件名目录项，然后紧跟其对应的短文件名。从表 14 可以看出，长文件名中并不存储对应文件的文件开始簇、文件大小、各种时间和日期属性。因此长文件名和短文件名之间的联系变得非常敏感，光靠他们之间的位置关系维系显然远远不够。其实，长文件名的 0xD 字节的校验和起很重要的作用，此校验和是用短文件名的 11 个字符通过一种运算方式来得到的。

8.2 数据区中目录簇的识别[26]

在 USB 的读写过程中，USB 控制器无需对文件系统有任何了解，在对文件的处理过程中所需要的操作全部由 PC 机上的操作系统完成，将需要读写的数据读出或写到 Flash 中。而 USB 所要做的就是不断的响应 PC 的请求，这些请求就是 6.1.4 节所述的 SCSI 的读写命令，完全不需要管上层的文件系统究竟在做什么工作。因此在这种条件下就需要以来与前一小节的分析来提供依据，用最快、最方便的方法将隐藏与大量数据中的目录簇识别出来。

从 8.1.1 与 8.1.2 可知通过对数据的过滤可以很容易的识别出 MBR、DBR、FAT1、FAT2 以及根目录，在地址超过跟目录的时候，如果该组数据是目录簇，那么不加密，而如果此次传输的数据为文件数据则需要加密。

FAT32 是以簇为单位进行存储的文件系统中，当前簇不是文件数据就是目录数据，所以只需要对每个簇的前面一部分的数据进行判断就可以识别出该簇的真正身份。文件数据的组织形式取决于该文件，而目录数据的组织形式却是由文件系统规定，因此既然有规定了目录数据的

格式、标准，它就必然存在不同于文件数据的数字特征。将文件数据看成是一种随机数，那么识别目录簇的工作便是从噪声中对有用信号的检测。通过 8.1.4 的分析可以发现，目录簇存在着这样几个特征：

(1). 在非跟目录的簇的开头，如果该簇为字母的首簇，那么前两个文件的文件名必定是“.”和“..”，他们分别代表当前目录和上级目录，因此同时还提供给我们的信息有：1）每个目录项的文件的长度一定为 0；2）“.”和“..”两者的创建时间的 10 毫秒位、创建时间和日期、最经修改时间和日期、最后访问日期都是一样的。

(2). 即使当前簇不是目录簇的首簇，它的短文件名所使用的值必定是可识别的 ASCII 码，那么就从一定程度上限制了相应字段的数值。

(3). 偏移为 0xB 的属性字节的最高两个 BIT 是 0

(4). 时间格式特征。表示小时的 5bit 的区间为[0,23]，表示分钟的 6 个 bit 的取值区间为[0,59]，同样表示“秒/2”的 5 个 bit 的范围为[0,29]。这些都没有将各自的空间使用完整。

(5). 日期格式特征。表示月份的 4bit 的取值区间为[1,12]，而表示日和年的 bit 利用价值不是很大。

(6). 长文件名的特征。1）偏移为 0xD 的字节的校验值应该同它的短文件名相对应；2）0x0 的属性字节的低 4bit 表示的顺序号数值的倒叙关系可以成为特征。

从以上分析可以看到目录簇存在很多特征，通过抓住这些特征，虽然不能 100%的命中目标，但至少可以保证在对合法的 FAT32 文件系统扫描时，不会出现漏检的现象。

但是在实现上，既要保证高的识别率，又要保证不影响数据传输的速度，最终采用上面的第一条特征，保证高速的识别。但是如果文件夹内部的文件项的个数超过 128 个，即子目录的出现了跨簇的现象时，就无法保证首簇后的文件的正确显示了。虽然在 U 盘中出现某个子文件夹下文件数过多的情况也不太可能，但是如果能够将上述的大多数特征都用上，对每个簇的前几项做一个合适的运算，是满足特征的预算结果都映射到一个区间，那么就可以很清晰的解决这一问题。

第九章　文件数据的加解密

在加密方式上，采用 AES 的加密方式，不只是因为它的安全性能，更因为 AES 是一种很好的分析加密算法。但是在加密模式上一直存在一定的问题。普通的 ECB 模式因为它的一一对应容易受到水印攻击(watermarking attack)，而 CBC(Cipher Block Chain)模式一直以来是众多模式中使用最为广泛的加密模式，尽管在某些方面还存在一些不足。综合考虑各种因素，CBC 仍然具有多方面的优越性。

9.1 AES 标准[16]

在密码学界，AES（Advanced Encryption Standard）加密算法是美国政府所采用的一种加密标准，是世界上最流行的对称加密算法之一。它具有三种不同的分组加密形式：AES-128、AES-192、AES-256，每种方法都使用 128-bit 分组长度，密钥长度可以分别为 128、192 和 256 比特。该加密算法的安全性能已经得到世界范围内的认可，并广泛应用。

简单的将 AES 算法表述如下：

1. 使用称之为 Rijndael's key schedule 的算法对密钥进行扩展
2. Initial Round
   a) 轮密钥加：利用单签分组和扩展密钥的一部分进行按位 XOR
3. Rounds （对于 128-bit 密钥需要进行 9 轮，192-bit 需要 11 轮，256-bit 密钥的 AES 需要 13 轮）
   a) 字节代换：用一个 S 盒完成分组中的按字节代换
   b) 行移位：行置换，状态表的每一行都要进行一定数量的移位
   c) 列混淆：一个利用在域 GF(28)上的算术特性的代换。
   d) 轮密钥加：利用单签分组和扩展密钥的一部分进行按位 XOR
4. Final Round (没有列混淆)
   a) 字节代换：用一个 S 盒完成分组中的按字节代换
   b) 行移位：行置换，状态表的每一行都要进行一定数量的移位
   c) 列混淆：一个利用在域 GF(28)上的算术特性的代换。

9.2 CBC 加密模式[16]

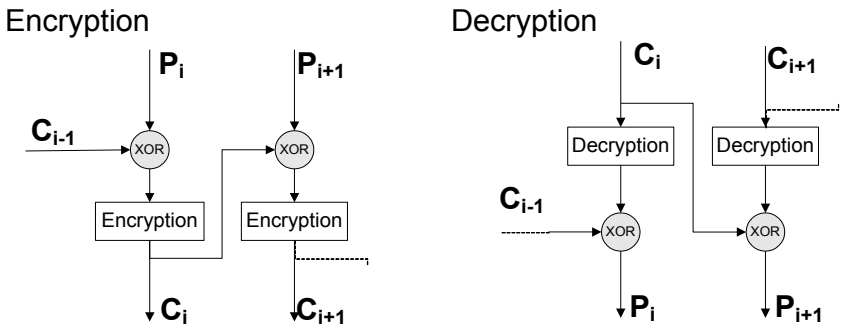

图 25 CBC 加解密模式

在一块可以扇区寻址的 Flash 或者硬盘上，通常一个扇区的大小为 512Byte，那么一个扇区

所需要的 AES 加密次数为 32 次。这 32 次加密本文使用 CBC 模式完成，IV(Initiation Vector)初始化向量由当前数据地址加密得到。

设扇区的地址为 n，该扇区中的第 i 个（i 从 0 开始计数）128 位明文记为 P_i^n ， P_i^n 经过 AES 加密之后的密文记为 C_i^n ，密钥为 k

$$C_i^n = E(k,[C_{i-1}^n \oplus P_i^n]); \qquad i \geq 0$$

$$C_{-1}^n = IV(n) = E(k,n)$$

由于 IV 的生成涉及到密码 k 的介入，所以 IV 的不可预测性很大程度增加了破解的难度。

第十章　测试与总结

10.1 总述

系统测试采用先局部后整体的方案，首先将各个模块（USB 数据传输模块、AES 加密模块）在分离的情况下单独测试，通过对它的各种操作来尽可能多的暴露问题，并一一解决。最后再将其整合起来进行总体测试。

对于软件测试完全依靠 RS232 串口输出程序中的数据、变量，然后通过计算机上的分析。除了串口工具，测试中还用到的主要工具有：

(1). Bus Hound 5.0：一款用于分析 PC 上 IO 数据软件，能够对多种协议的数据包进行截取、解析并保存。这些数据代表了在 PC 上看到的 USB 数据，与串口发回的 USB 数据对比可以找到程序的漏洞，并作出相应的改进。

(2). USB-IF Test Suite：USB 开发者论坛提供的用于 USB 标准测试的软件，能够查看当前设备是否符合国际标准。

(3). HD Tach v3.0.4.0：用来对 USB 的传输速率测试，它会对 U 盘提供大小不同的数据包，分别计算他们的传输速率。

(4). ATTO Disk Benchmark v2.34：同样是 USB 测速软件，使用不同的两款软件可以提高测试数据的可信度。

(5). WinHex 15.0 SR-2：用于观察 Flash 上每一个扇区的数据变化，有效地判断加密的正确性并评估效果。

10.2 AES 加密函数验证性测试

因为 AES 加解密函数来源于互联网，难免会有疏漏，质量不能得到保证。稍作修改后应当对齐正确性进行验证，通过权威机构发布的测试向量对齐进行验证具有权威性。测试向量是一组已经给出的输入明文、密钥和输出密文的集合。以下数据来源于 NIST 发布的联邦信息处理标准(FIPS PUBS) 197。例如，对于一个 128-bit 的密钥为

"000102030506070 80A0B0C0D0F101112"

并且输入明文为

"506812A45F08C889B97F5980038B8359"

128-bit 的 AES 加密过后的正确输出应当为

"D8F532538289EF7D06B506A4FD5BE9C9"

另一组测试值为：

明文：　"4EC137A426DABF8AA0BEB8BC0C2B89D6"

密钥：　"95A8EE8E89979B9EFDCBC6EB9797528D"

密文：　"D9B65D1232BA0199CDBD487B2A1FD646"

若使用上以测试向量的明文作为密文，密钥不变，则解密明文为：

"9570C34363565B393503A001C0E23B65"

通过编写专门测试程序，串口输出测试结果显示 AES 加解密函数准确无误。

10.3 传输性能测试

将加解密设备插到电脑 USB 端口，再将 U 盘连接到本设备。等待指纹输入完成，连接计算机，显示 U 盘盘符。可以开始数据传输，传输速率测试结果如下图：

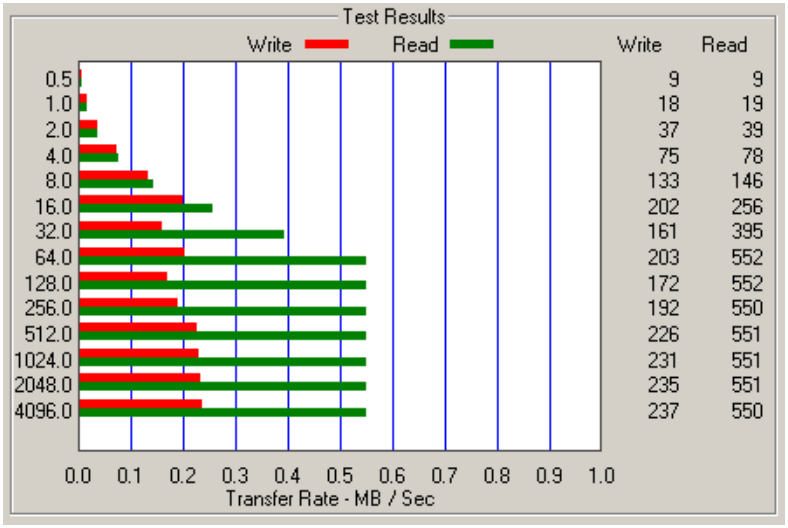

图 26 设备读写速度测试结果

如图 28 所示为一款金士顿 U 盘在本平台上，通过专业测试软件 ATTO Disk Benchmark 所得的 USB 传输速率。通过对测试结果的观察，可以发现随着传输数据的增加，读写速度都会有很大的提高，因为在传输过程中没有一簇的数据会自动补全一簇处理，在加上其余的传输控制使得在传输小的文件数据时所得的读写速度偏低。直到当文件达到 64K 左右时，读写速度相对开始稳定，写入速度稳定在 230KB/s 左右；而因为缓存的使用，使得同样情况的读速率达到了将近写的两倍，为 550KB/s。根据文献 28 的测试，USB 全速设备的传输速度理论上最高只能达到 1MB/s 左右，而文献 29 指出真正实现中能达到的速度为 600KB/s~700KB/s 左右，因此，相比而言系统所得传输效果令人满意。

再将加密后的文件重新拷贝到电脑上，打开后与原文件一样，本设备加解密运算在系统中运行正确。将加密后的 U 盘直接连到电脑上，电脑无法读写 U 盘中的数据。

通过以上测试，作品完全能够正确完成加解密运算，为 U 盘提供可靠的保护，完全实现了预期目标。

第十一章 结束语

11.1 论文总结

通过不断的调试与改进，系统已经能够正确运行，并且得到了让人满意的效果。通过该系统，在计算机上可以正确无误的看到文件目录，在读取文件数据时只有存入该数据时使用的指纹才能够正确的解密文件，否则看到的只能是无法解开的乱码。通过总结，本文完成了以下工作：

(1). 完成硬件电路板的制作技术和调试

(2). 移植 USB 主机、设备段驱动程序，实现 USB 主从设备间的数据透明传输

(3). 支持对普通 U 盘文件的加解密，文件名称可见，内容不可见，方便文件管理

(4). 与指纹模块相互通信，获取指纹图像

在与 DSP 指纹模块相互连接，完成系统整体功能的调试之后，加解密设备的实物图如图 27 所示。从上之下四种不同类型的虚线框分别是：指纹采集传感器、DSP 指纹预处理模块、SPI 接口、ARM 嵌入式平台。

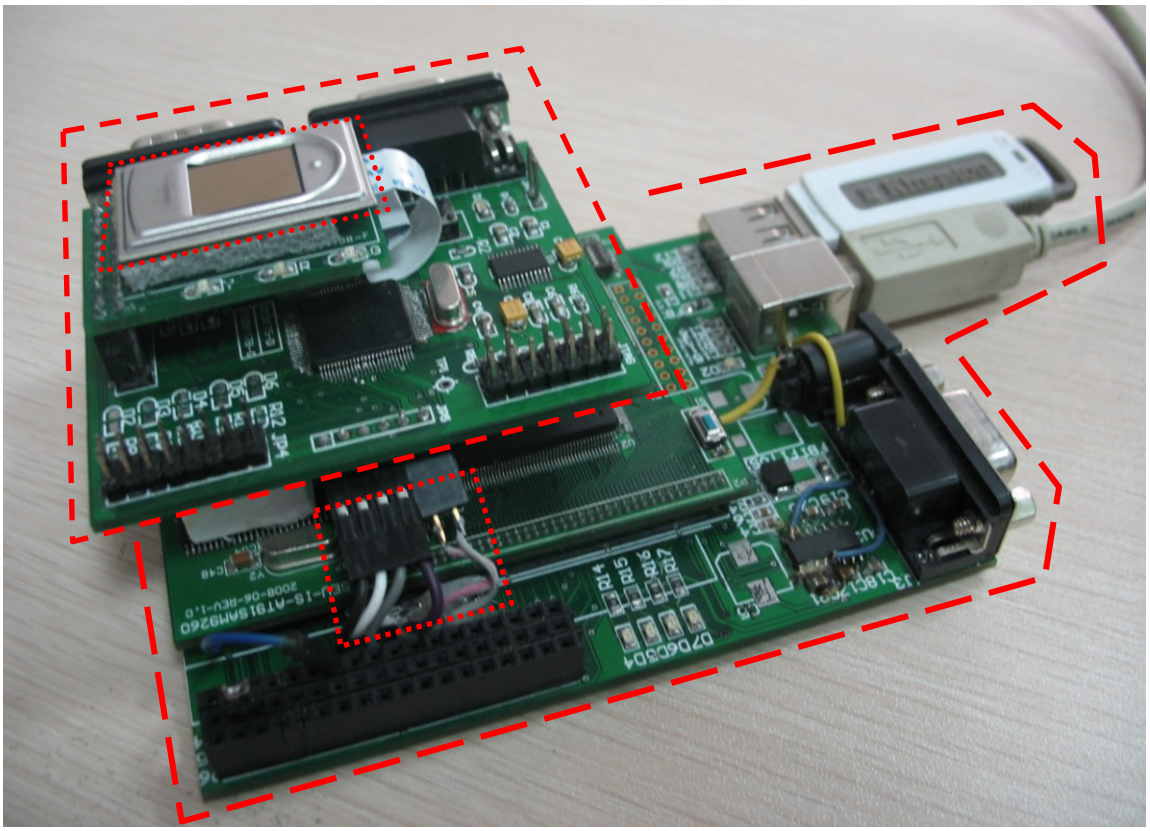

图 27 系统最终实物图

在完成这些工作的同时，努力提高 USB 方面的理论水平，学习了 OHCI 规范和 FAT32 文件系统的标准，收获颇深。通过代码的改进，提高了对 USB 存储器的兼容性。在实践过程中，有如下创新：

(1). 采用了独立于 U 盘与计算机的设计思想，采用过滤 USB 数据流的方法实现文件加密。

(2). 对 U 盘中的每一个文件进行加解密，而不是对整个 U 盘加解密，使用和管理都更为方便。

a) 提出了 FAT32 文件系统中目录簇的识别方法及未来的扩展空间

b) 构造了数据缓存区，使文件传输更快。

(3). 提出了一种更快、更有利于保护硬件的程序下载方式。

11.2 前景与展望

作品拥有广阔的应用前景，能为私人、企业、政府等机构提供便捷、可靠的 U 盘安全解决方案。加密器与 U 盘相分离，减小指纹信息泄露的威胁，在需要保密的部门更便于安全上的管理，因此将大受欢迎。

致 谢

首先衷心感谢我的导师胡爱群教授！胡老师在我的本科四年最后的很长的一段时间里给我提供了良好的学习条件和宽松的研究氛围，在实验室里同样可以感觉到家的温暖。在跟随胡老师的这段时间里，我的学习、研究和动手能力有了很大的进步，不仅眼界得到了开阔，而且看问题的角度也变得更加全面、科学。胡老师渊博的学识、治学严谨的态度和永不停息的求学精神，给我很大的启迪。

感谢宋宇波老师和秦中元老师，在课题研究最困难的时候给予了我继续的信心，他们不厌其烦的和我讨论、交流，让我重新看到了希望的曙光，并且找到解决目录与文件数据分离的手段。他们对我的研究成果的肯定在精神上深深的激励我更加努力的学习和奋斗。

感谢蒋睿老师在项目前期对我的指导，他让我明白思维只有经过激烈的碰撞才能长生绚烂的光彩。感谢 Lakeview Research 的 Janet Louise Axelson 通过 E-mail 对我的前期工作的指导，和他的技术网站上提供的丰富经验以及参考代码。感谢 OpenSSL、Linux 开源社区的程序爱好者们。

感谢实验室的金制和周青学长，是他们在把我从对嵌入式的一知半解带入这道门槛，在学习资料与实践经验上给予了我很大帮助，祝他们在以后的工作中一帆风顺。

感谢同课题的范鹏飞、刘慧慧同学，感谢陈开志博士和郭晓乔师兄，和他们在一起的半年里学习和生活都非常愉快，尤其是讨论问题、共享学习所得是我在学习之余最大的乐趣。感谢顾群老师还有实验室的所有师兄师姐，大家一起为信息安全研究中心营造了良好的学习研究环境。

感谢吴健雄学院的钟辉书记近些年来学习和生活上对我无微不至的关怀，给了我很多宝贵的机会，使我视野更开阔，对自己信心更加坚定。

最后，感谢父母、姐姐和任艳的关心和支持，他们一如既往的支持和信任是我永远的动力和信心。

此外，在本科四年期间，还有很多在学业和生活上给我鼓励、帮助的老师和同学，虽然不能一一列举出他们的名字，但正是他们默默的帮助和支持给了我很多美好的回忆。

参考文献（Reference）

[1] AxelsonJan. USB Mass Storage:Designing and programming Devices and Embedded Hosts[M]. Lakeview Research LLC. 2006

[2] Compaq, Hewlett-Packard, Intel... Universal Serial Bus Specification[EB/OL]. www.usb.org, 2000-4-27

[3] Harlan Carvey, Cory Altheide. Tracking USB storage: Analysis of windows artifacts generated by USB storage devices[J]. Digital Investigation, 2005(2) 94-100

[4] USB-IF. Universal Serial Bus Mass Storage Class-Bulk Only Transport[EB/OL]. USB Implementers Forum, 1999-09-31

[5] 匿名. FAT 文件系统原理[EB/OL]. 数据恢复: www.sjhf.net

[6] 牛玲，李骞. 信息安全防护与 USB 安全控制技术[J]. 周口师范学院学报, 2006,23(2) 109-110.

[7] 王殊，程卓. 基于 CH375 的嵌入式 USB 文件加解密系统的设计. 电子技术应用, 2007(8) 157-160

[8] Microsoft Corporation. Microsoft Extensible Firmware Initiative FAT32 File System Specification[EB/OL]. http://www.microsoft.com, 2000-11-06

[9] Curtis E Stevens, Mark Gianopulos. USB Communication Device Class Definition for Mass Storage Devices[EB/OL]. USB Implementers Forum. 1996-02-02

[10] 王梅,林君,刘俊华. 一种基于嵌入式 USB-Host 的大容量数据存储方案及应用[J]. 计算机测量与控制. 2004. 12(4): 384-389

[11] Stephen D Wolthusen. Windows device interface security[J]. Information Security Technical Report, 2006(11) 160-165

[12] 张井刚，刘德活. 在嵌入式系统中提高 U 盘访问兼容性的几种措施[EB/OL]. www.eepw.com.cn/article/21990.htm, 2007-02-04

[13] frank_wang. 对 USB 协议层的深层分析[EB/OL]. http://www.embedusb.51.net, 2004-04-17

[14] frank_wang. USB 项目技术报告[EB/OL]. http://www.embedusb.51.net, 2004-04-17

[15] ATMEL Corporation. AT91Bootstrap[EB/OL]. http://www.atmel.com, 2005-12-06

[16] William Stallings. Cryptography and Network Security: Principles and Practices [M]. Forth Edition. New Jersey: Pearson Education, 2006. 99-115

[17] 赵鹏，蒋烈辉. AT91 系列微处理器启动过程的分析与实现[J]. 电子设计应用，2005.3 116-120

[18] 詹荣开. 嵌入式系统 Boot Loader 技术内幕[EB/OL]. Linux 爱好者, 2003-12-01

[19] INCITS. SCSI Primary Commands-3[EB/OL]. American National Standards Institute. 2002.11.10

[20] ATMEL Corporation. AT91 ARM Thumb Microcontrollers: AT91SAM9260 Preliminary[EB/OL]. http://www.atmel.com, 2008-01-31

[21] 深圳无线互联技术. CA9260 核心板用户手册[EB/OL]. http://www.szwinet.com. *2008.04.01*

[22] Richard M. Stallman, Roland McGrath, Paul Smith. GNU Make[M]. Version 3.8. MA: Free Software Foundation, 2002

[23] ATMEL Corporation. SAM Boot Assistant(SAM-BA) User Guide[EB/OL]. http://www.atmel.com, 2006-10-09

[24] SAMSUNG Electronics. 128M x 8 Bit NAND Flash Memory[EB/OL]. http://www.samsung.com, 2006.09.27

[25] William Stallings. Computer organization and architecture[M]. Fifth Edition . New Jersey: Pearson Education, 2006. 99-115

[26] 黄步根. 数据恢复与计算机取证[J]. 计算机安全, 2006.06 79-80

[27] Richard M. Stallman. Using and Porting the GNU Compiler Collection[M]. MA: Free Software Foundation, 2001

[28] 旧日足迹. 高价不高速,走出 USB2.0 接口速度规格误区[EB/OL]. http://blog.zol.com.cn, 2006.11.13

[29] shuiyan. USB 传输速率[EB/OL]. http://topic.csdn.net, 2008.05.29

附录

1、电路图：

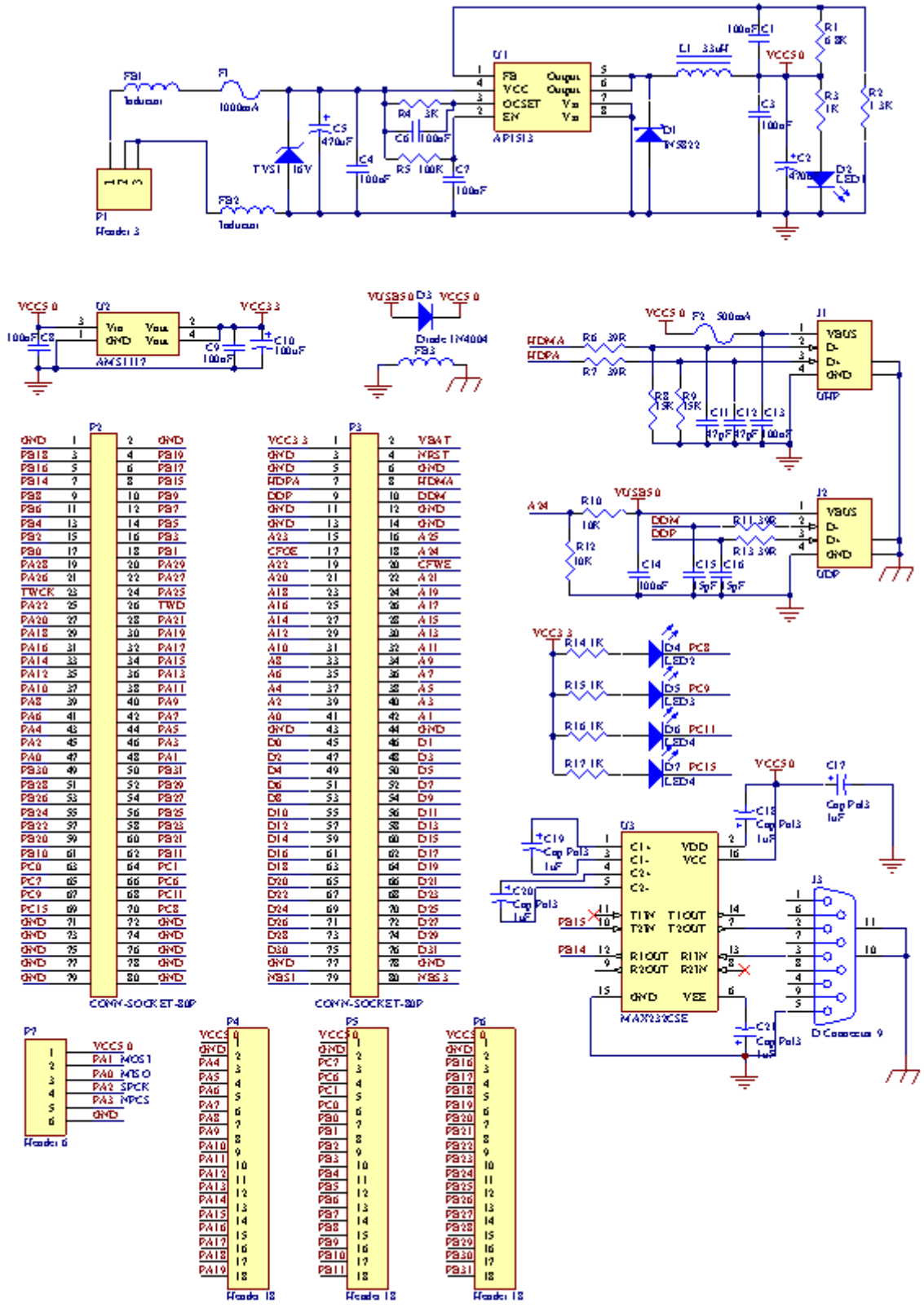

图 28 AT91SAM9260 底板电路原理图

2、标准 USB 描述符

表 15　　设备描述表(18Byte)

Byte	Field	Description
0	bLength	Descriptor size in bytes (12h)
1	bDescriptorType	The constant DEVICE (01h)
2	bcdUSB	USB specification release number (BCD). For USB 2.0, byte 2 = 00h and byte 3 = 02h.
4	bDeviceClass	Class code. For mass storage, set to 00h (the class is speci-fied in the interface descriptor).
5	bDeviceSubclass	Subclass code. For mass storage, set to 00h.
6	bDeviceProtocol	Protocol Code. For mass storage, set to 00h.
7	bMaxPacketSize0	Maximum packet size for endpoint zero.
8	idVendor	Vendor ID. Obtained from USB-IF.
10	idProduct	Product ID. Assigned by the product vendor.
12	bcdDevice	Device release number (BCD). Assigned by the product vendor.
14	iManufacturer	Index of string descriptor for the manufacturer. Set to 00h if there is no string descriptor.
15	iProduct	Index of string descriptor for the product. Set to 00h if there is no string descriptor.
16	iSerialNumber	Index of string descriptor containing the serial number. Must be > 00h for mass-storage devices.
17	bNumConfigurations	Number of possible configurations. Typically 01h.

表 16　　配置描述表

Byte	Field	Description
0	bLength	Descriptor size in bytes. Always 09h.
1	bDescriptorType	The constant CONFIGURATION (02h).
2	wTotalLength	The number of bytes in the configuration descriptor and all of its subordinate descriptors.
4	bNumInterfaces	The number of interfaces in the configuration.
5	bConfigurationValue	Identifier for Set Configuration and Get Configuration requests. Use 01h for the first configuration.
6	iConfiguration	Index of string descriptor for the configuration. Set to 00h if there is no string descriptor.
7	bmAttributes	Self/bus power and remote wakeup settings.
8	bMaxPower	The amount of bus power the device requires, expressed as (maximum milliamperes / 2).

表 17　　接口描述（海量存储功能描述）

Byte	Field	Description
0	bLength	Descriptor size in bytes (09h).
1	bDescriptorType	The constant INTERFACE (04h).
2	bInterfaceNumber	Number identifying this interface.
3	bAlternateSetting	Set to 00h for the default setting.
4	bNumEndpoints	Number of endpoints supported, not counting endpoint zero. Set to 02h for a bulk-only mass-storage device.
5	bInterfaceClass	Class code. Mass storage = 08h.
6	bInterfaceSubclass	Subclass code. Mass-storage values: 01h: Reduced Block Commands (RBC). 02h: SFF-8020i, MMC-2 (ATAPI) (CD/DVD drives) 03h: QIC-157 (tape drives). 04h: USB Floppy Interface (UFI). 05h: SFF-8070i (ATAPI removable rewritable media devices). 06h: SCSI transparent command set. Use the SCSI INQUIRY command to determine the peripheral device type. Recommended value for most devices.
7	bInterfaceProtocol	Protocol code. Mass storage values: 00h: CBI with command completion interrupt transfers 01h: CBI without command completion interrupt transfer 50h: bulk only. Recommended value for most devices.
8	iInterface	Index of string descriptor for the interface.

表 18　　批量传输协议的端点描述

Byte	Field	Description
0	bLength	Descriptor size in bytes (07h).
1	bDescriptorType	The constant Endpoint (05h).
2	bEndpointAddress	Endpoint number and direction.
3	bmAttributes	Transfer type supported. Bulk = 02h.
4	wMaxPacketSize	Maximum packet size supported.
6	bInterval	Maximum NAK rate for high-speed bulk OUT endpoints. Otherwise ignored for bulk endpoints.

表 19　　字符串描述

Byte	Field	Description
0	bLength	Descriptor size in bytes
1	bDescriptorType	The constant String (03h)
2	bSTRING or wLANGID	For string descriptor zero, an array of 1 or more Language Identifier codes. For other string descriptors, a Unicode string.

3、BOT 传输模式 CBW、CSW 定义

表 20　　USB 主机向设备发送 CBW 的格式

Name	Bits	Description
dCBWSignature	32	The value 43425355h, which identifies the structure as a CBW. The LSB (55h) transmits first on the bus.
dCBWTag	32	A value that associates this CBW with the CSW the device will send in response.
dCBWData Transfer Length	32	If bit 7 of bmCBWFlags = 0, the number of bytes the host will send in the data-transport phase. If bit 7 of bmCBWFlags = 1, the number of bytes the host expects to receive in the data-transport phase.
bmCBWFlags	8	Specifies the direction of the data-transport phase. Bit 7 = 0 for an OUT (host-to-device) transfer. Bit 7 = 1 for an IN (device-to-host) transfer. If there is no data-transport phase, bit 7 is ignored. All other bits are zero.
Reserved	4	Zero.
bCBWLUN	4	For devices with multiple LUNs, specifies the LUN the command block is directed to. Otherwise the value is zero.
Reserved	3	Zero.
bCBWCBLength	5	The length of the command descriptor block in the CBWCB field in bytes. Valid values are 1–16. Currently defined command descriptor blocks are all at least 6 bytes.
CBWCB	128	The command block for the device to execute.

表 21　　USB 设备发回主机的状态 CSW

Name	Bits	Description
dCSWSignature	32	The value 53425355h, which identifies the structure as a CSW. The LSB (55h) transmits first on the bus.
dCSWTag	32	The value of the dCBWTag in a CBW received from the host.
dCSWData Residue	32	For transfers where the host sends data to the device in the data-transport phase, the difference between dCBWDataTransferLength and the number of bytes the device processed. For transfers where the device sends data to the host in the data-transport phase, the difference between dCBWDataTransferLength and the number of valid bytes the device has sent, excluding any pad bytes.
bCSWStatus	8	00h = command passed. 01h = command failed. 02h = phase error. Host should perform a reset recovery.