USB Mass Storage 学习笔记－STM32＋FLASH 实现 U 盘

一、 内容概述

采用 STM32 内部自带 USB 控制器外加大页 NAND FLASH K9F1G08U0A 实现一个 128M 的 U 盘。

1、STM32 的 USB 控制器

STM32F103的MCU自带USB从控制器，符合USB规范的通信连接；PC主机和微控制器之间的数据传输是通过共享一专用的数据缓冲区来完成的，该数据缓冲区能被USB外设直接访问。这块专用数据缓冲区的大小由所使用的端点数目和每个端点最大的数据分组大小所决定，每个端点最大可使用512字节缓冲区，最多可用于16个单向或8个双向端点。USB模块同PC主机通信，根据USB规范实现令牌分组的检测，数据发送/接收的处理，和握手分组的处理。整个传输的格式由硬件完成，其中包括CRC的生成和校验。

每个端点都有一个缓冲区描述块，描述该端点使用的缓冲区地址、大小和需要传输的字节数。

当USB模块识别出一个有效的功能/端点的令牌分组时，(如果需要传输数据并且端点已配置)随之发生相关的数据传输。USB模块通过一个内部的16位寄存器实现端口与专用缓冲区的数据交换。在所有的数据传输完成后，如果需要，则根据传输的方向，发送或接收适当的握手分组。

在数据传输结束时，USB模块将触发与端点相关的中断，通过读状态寄存器和/或者利用不同的中断来处理。

USB的中断映射单元：将可能产生中断的USB事件映射到三个不同的NVIC请求线上：

（1）USB低优先级中断(通道20)：可由所有USB事件触发(正确传输，USB复位等)。固件在处理中断前应当首先确定中断源。

（2）USB高优先级中断(通道19)：仅能由同步和双缓冲批量传输的正确传输事件触发，目的是保证最大的传输速率。

（3）USB唤醒中断(通道42)：由USB挂起模式的唤醒事件触发。

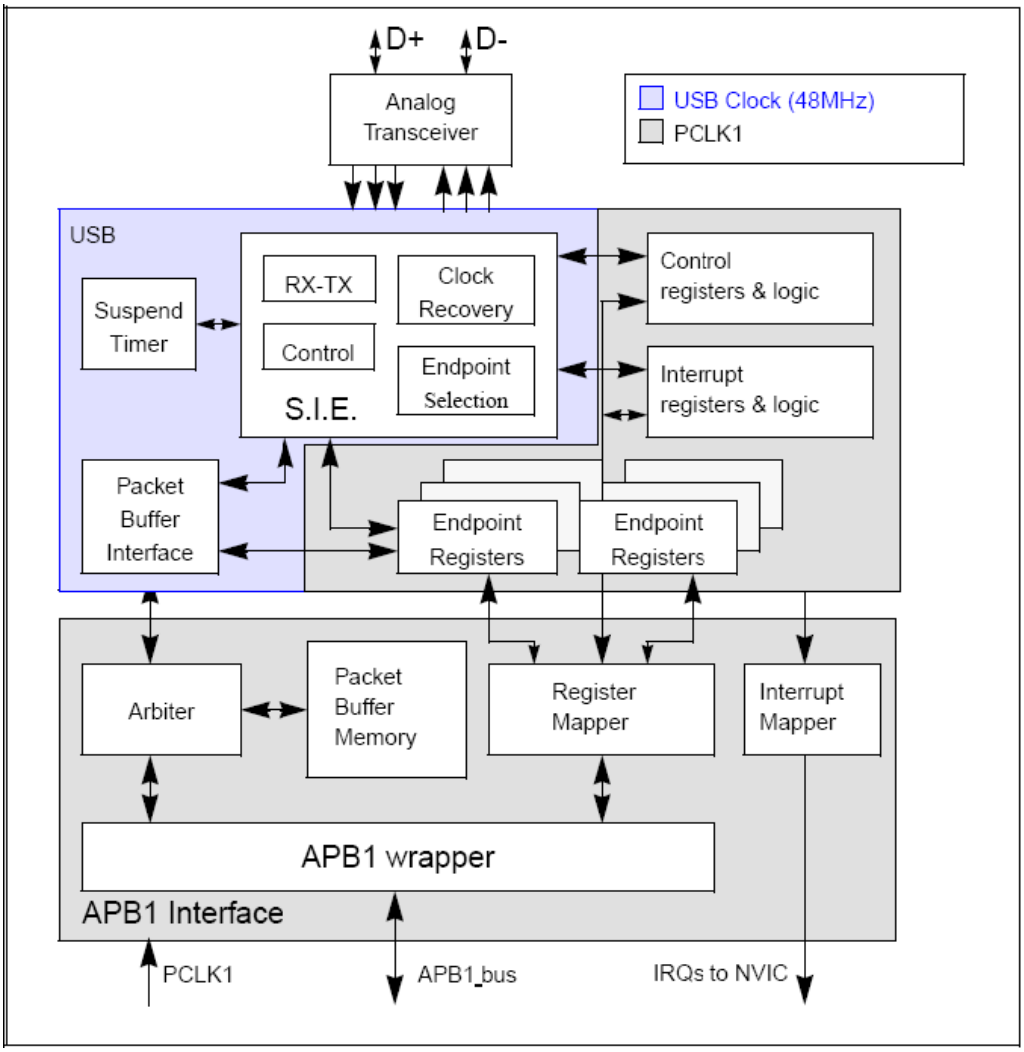

图 1、USB设备框图

2、大页NAND K9F1G08

Nand flash 以页为单位读写数据，而以块为单位擦除数据。根据NAND的容量等级又将NANDFLASH分为大页NAND和小页NAND；K9F1G08就是大页NAND，它的页大小为(2K+64)Byte,块大小为（128K＋4K）Byte。K9F1208U0M为小页NAND，它的页大小为(512+16)Byte,块大小为（16K＋512）Byte。

由于写数据至FLASH时，只能将指定的位变为0，而不能将指定的位变位1。因此在写一个页的数据前，必须先擦除（将所有的位全部置1），否则写数据会失败。

在编制FLASH的读写程序时，需要传递三个参数，要操作的地址，要操作的数据缓存，要操作的数据长度；在写操作时，还要有擦写和坏块管理。

3、USB Mass storage Bulk Only

Mass Storage 类支持两个传输协议：

1、Bulk-Only 传输（BOT）

2、Control/Bulk/Interrupt 传输（CBI）

Mass Storage 类规范定义了两个类规定的请求：Get_Max_LUN 和 Mass Storage Reset，所有的 Mass Storage 类设备都必须支持这两个请求。

Bulk-Only Mass Storage Reset（bmRequestType=00100001b and bRequest= 11111111b）用来复位 Mass Storage 设备及其相关接口。

Get_Max_LUN（bmRequestType= 10100001b and bRequest= 11111110b）用来确认设备支持的逻辑单元数。Max LUN 的值必须是 0~15。注意：LUN 是从 0 开始的。主机不能向不存在的 LUN 发送 CBW。

支持 BOT 传输的 Mass Storage 设备接口描述符要求如下：

接口类代码 bInterfaceClass=08h，表示为 Mass Storage 设备；

接口类子代码 bInterfaceSubClass=06h，表示设备支持 SCSI Primary Command-2（SPC-2）；

协议代码 bInterfaceProtocol 有 3 种：0x00、0x01、0x50，前两种需要使用中断传输，最后一种仅使用批量传输（BOT）。

支持 BOT 的设备必须支持最少 3 个 endpoint：Control, Bulk-In 和 Bulk-Out。USB2.0 的规范定义了控制端点 0。Bulk-In 端点用来从设备向主机传送数据。Bulk-Out 端点用来从主机向设备传送数据。

Bulk-Only 传输（BOT）

像控制传输一样，BOT 也是由 Command 阶段，可选的数据阶段和状态阶段组成。所有的 command 请求都可能有或没有 Data 阶段。下图说明了 BOT 的 Command 传输，Data-In，Data-Out 传输及 Status 传输。

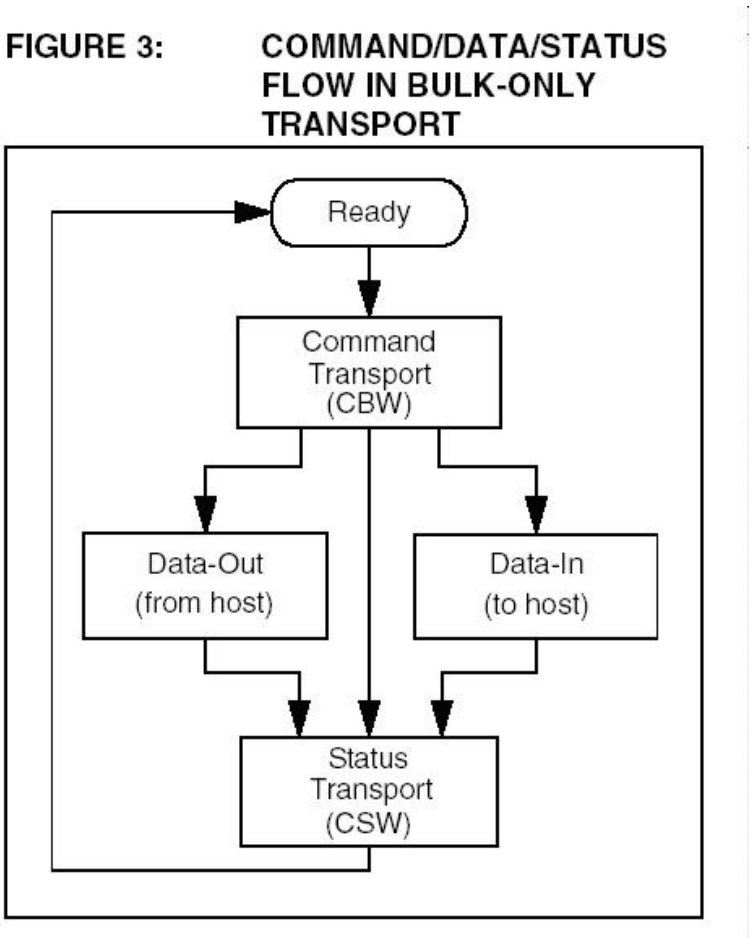

图 2、Bulk-Only 传输示意图

CBW 是由 31 个字节组成的短包。CBW 和后续的数据以及 CSW 都是从新封包开始的。要注意的是所有 CBW 传输都是 little-endian 模式。

在 CBW 中，dCBWSignature 必须是“43425355h”，表示是 CBW 封包。dCBWTag 是 CB 标签，会通过对应的 CSW 的标签反馈回来。

在 CSW 中，dCSWSignature 必须是“53425355h”,表示是 CSW 包。

二、 系统的初始化

1、初始化系统时钟，设置 USB 时钟为 48MHz；

2、USB 中断配制，

选择通道、设置优先级、使能中断。

3、USB 初始化：连接 USB、USB 硬件复位、配制 CNTR 寄存器使能和屏蔽中断、清零中断状态寄存器。

4、FLASH 初始化。

三、 USB 的枚举

当 USB 连接时，进入 USB 低优先级中断。首先获取中断状态（读 ISTR

寄存器），在 MASS STORAGE 中有 USB 复位中断、USB 挂起中断和正确的数据传输中断。

注：在 usb_istr.c 的 void USB_Istr(void)函数中。

1、USB 总线复位：

设置分组缓冲区描述表起始地址；

初始化端点：端点 0 为控制端点、端点 1、2 为批量端点；设置发送和接收状态，设置发送和接收缓冲区地址。

设置 CBW 签名， CBW.dSignature=0x43425355;

初始化 BOT 状态机。

注：在 usb_prop.c 的 void MASS_Reset()函数中。

2、USB 总线挂起：Xms 总线上无数据传输，USB 总线挂起，进入低功耗模式。

注：在 usb_pwr.c 的 void Suspend()函数中。

3、正确的数据传输中断（usb_int.c CTR_LP();）

清除中断标志；

获取端点标识符；

控制端点处理：读端点寄存器，用来判断是数据输入、输出还是建立包。

非控制端点处理：下一节介绍。

详见软件流程图。

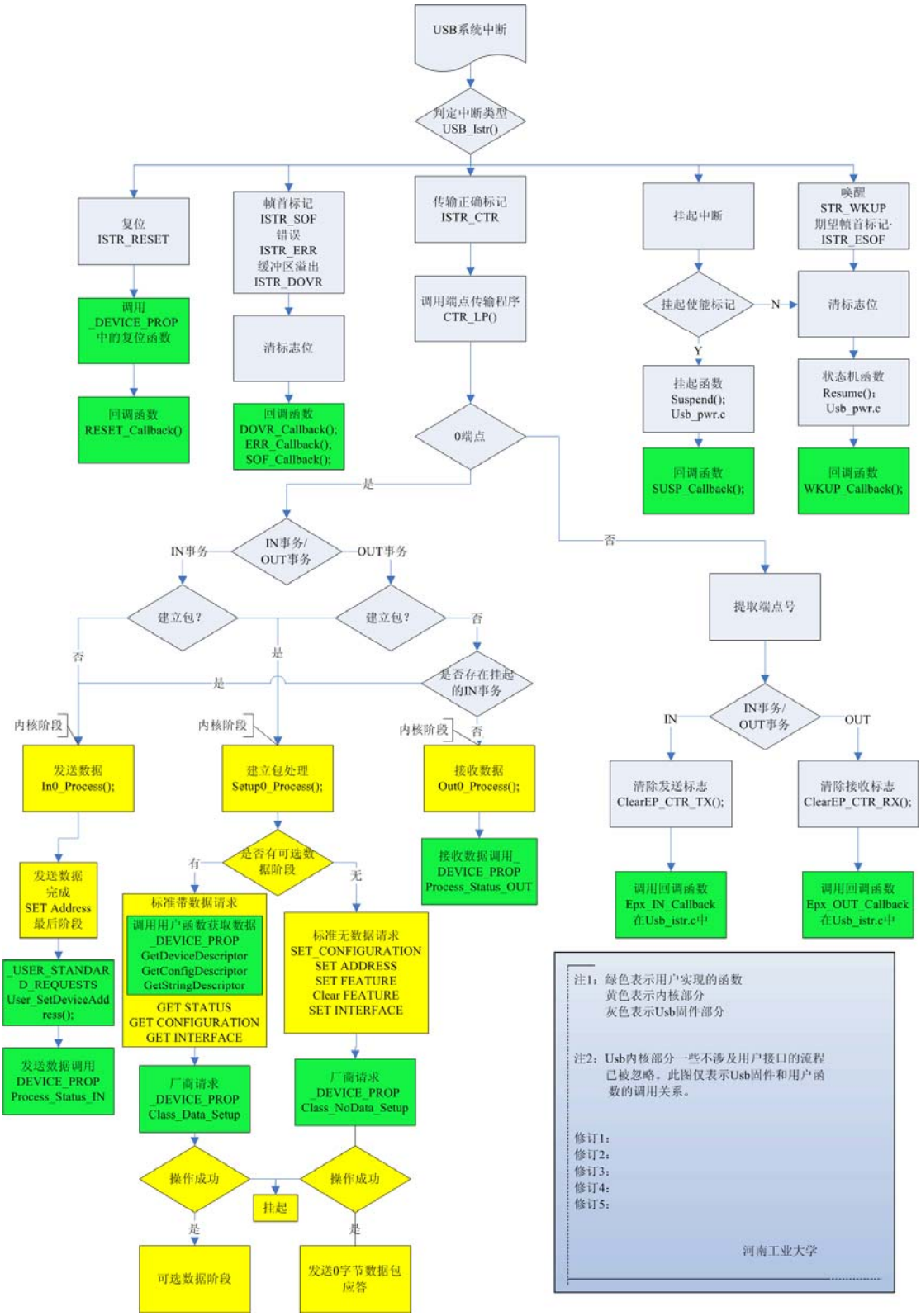

图 3、USB 枚举软件流程图

四、 非控制端点处理（usb_endp.c －> usb.bot.c）

端点 2 输出中断：usb 主机传数据或命令包至 mcu。

端点 1 输入中断：mcu 传数据或描述符至 usb 主机。

1、端点 2 输出中断

（1） 将主机传过来的数据从 USB 端点缓存区 copy 至 MCU 内存；

（2） 判断 BOT 状态，根据 BOT 状态做出相应的处理：当 BOT 状态位为 0 时，CBW 包解析，并处理 SCSI 命令；当 BOT 状态为 1 时，表示数据输出，执行 WRITE10 命令处理。

2、端点 1 输入中断

判断 BOT 状态，如果 BOT 状态为 2，表示数据输入，执行 READ10 命令处理；如果 BOT 状态为 4，则表示数据输入完成，则返回 CSW，进行到命令状态。

3、BOT 状态机软件流程图

（1）端点 2 输出中断流程图（usb_bot.c -> Mass_Storage_Out()）

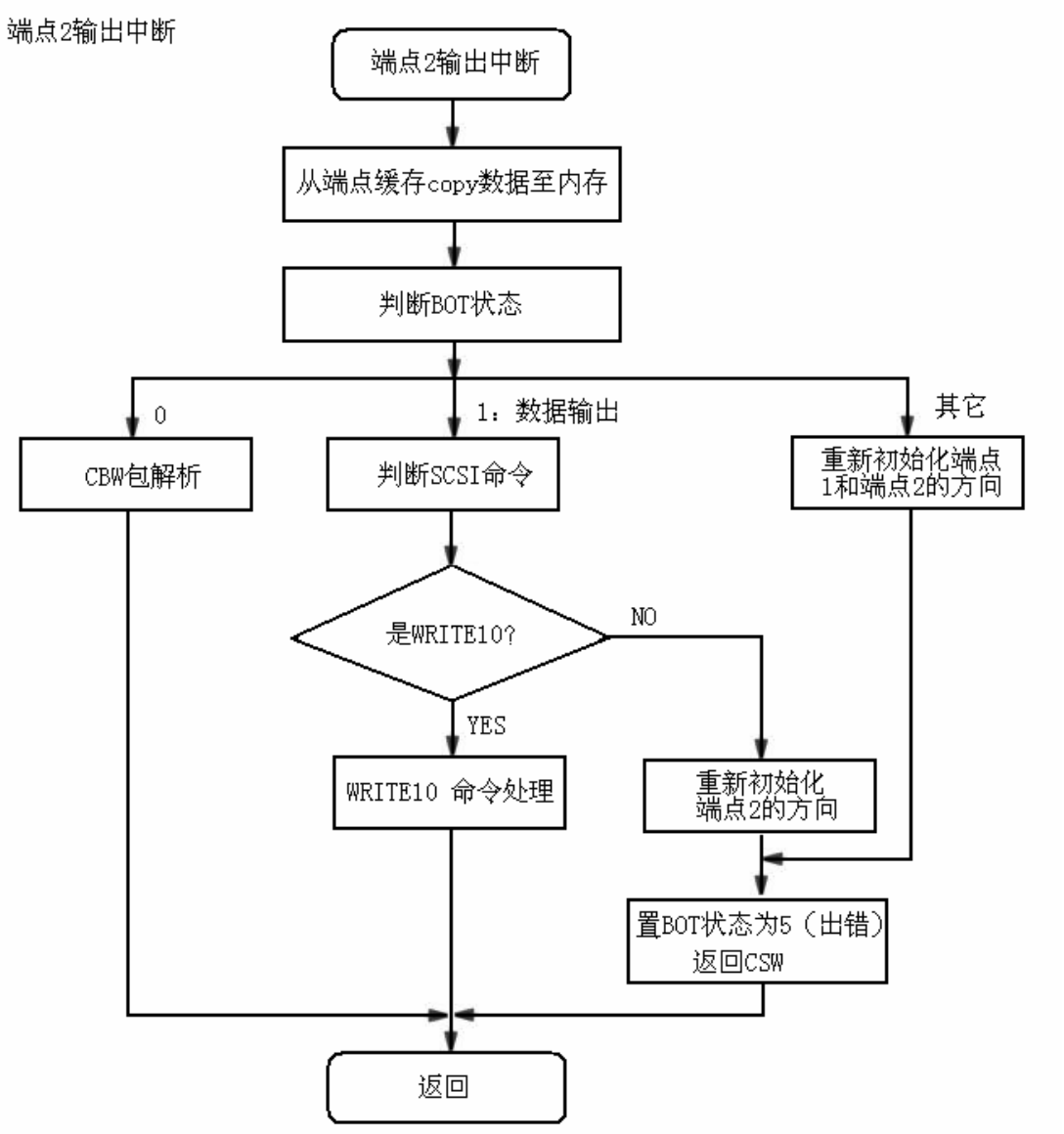

图 4、端点 2 输出中断

（2）CBW 包解析软件流程图（usb_bot.c -> CBW_Decode()）

CBW包解析

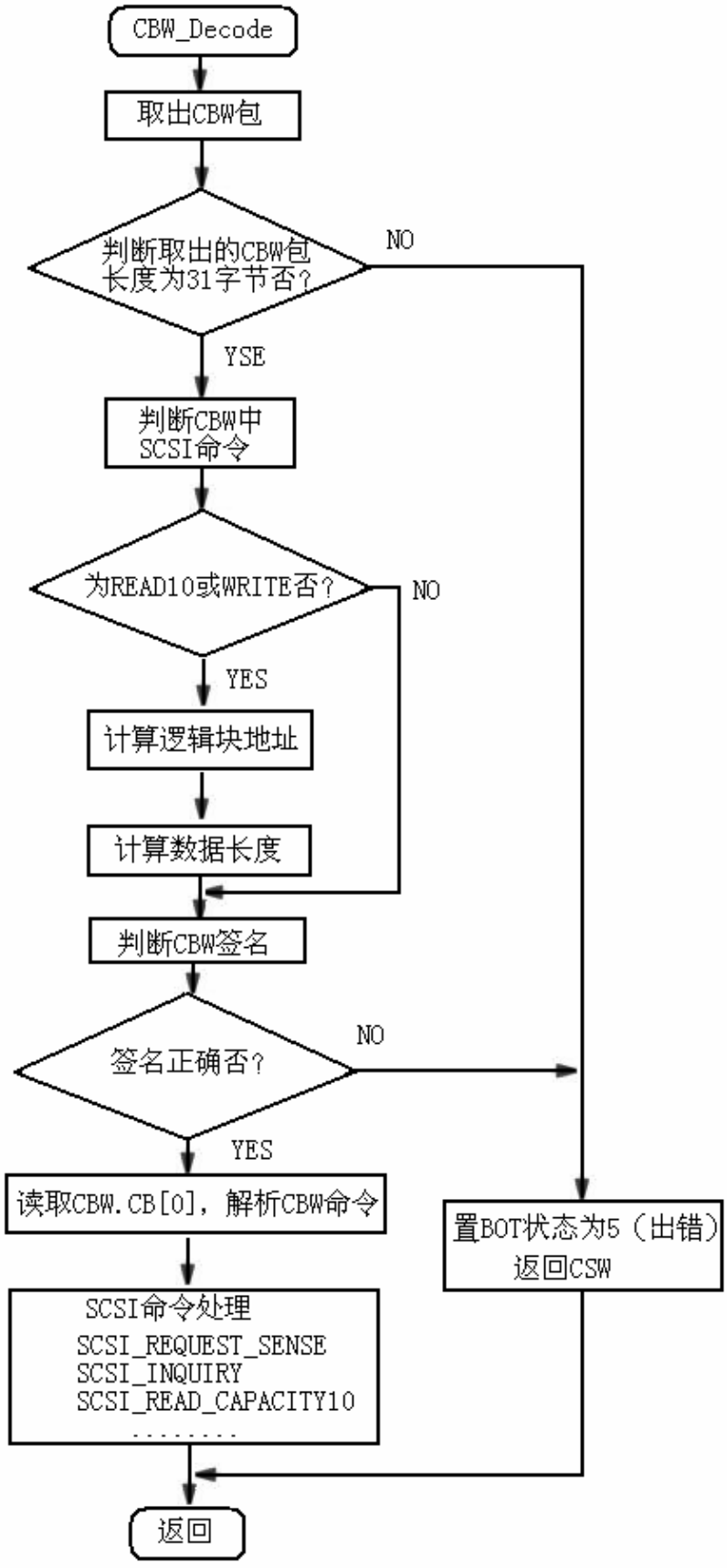
CBW_Decode
取出CBW包
判断取出的CBW包
长度为31字节否?
NO
YSE
判断CBW中
SCSI命令
为READ10或WRITE否?
NO
YES
计算逻辑块地址
计算数据长度
判断CBW签名
签名正确否?
NO
YES
读取CBW.CB[0], 解析CBW命令
置BOT状态为5（出错）
返回CSW
SCSI命令处理
SCSI_REQUEST_SENSE
SCSI_INQUIRY
SCSI_READ_CAPACITY10
.....
返回

图 5、CBW 包解析

（3）READ10 命令软件流程图（usb_scsi.c -> SCSI_Read10_Cmd(lun , LBA , BlockNbr)）

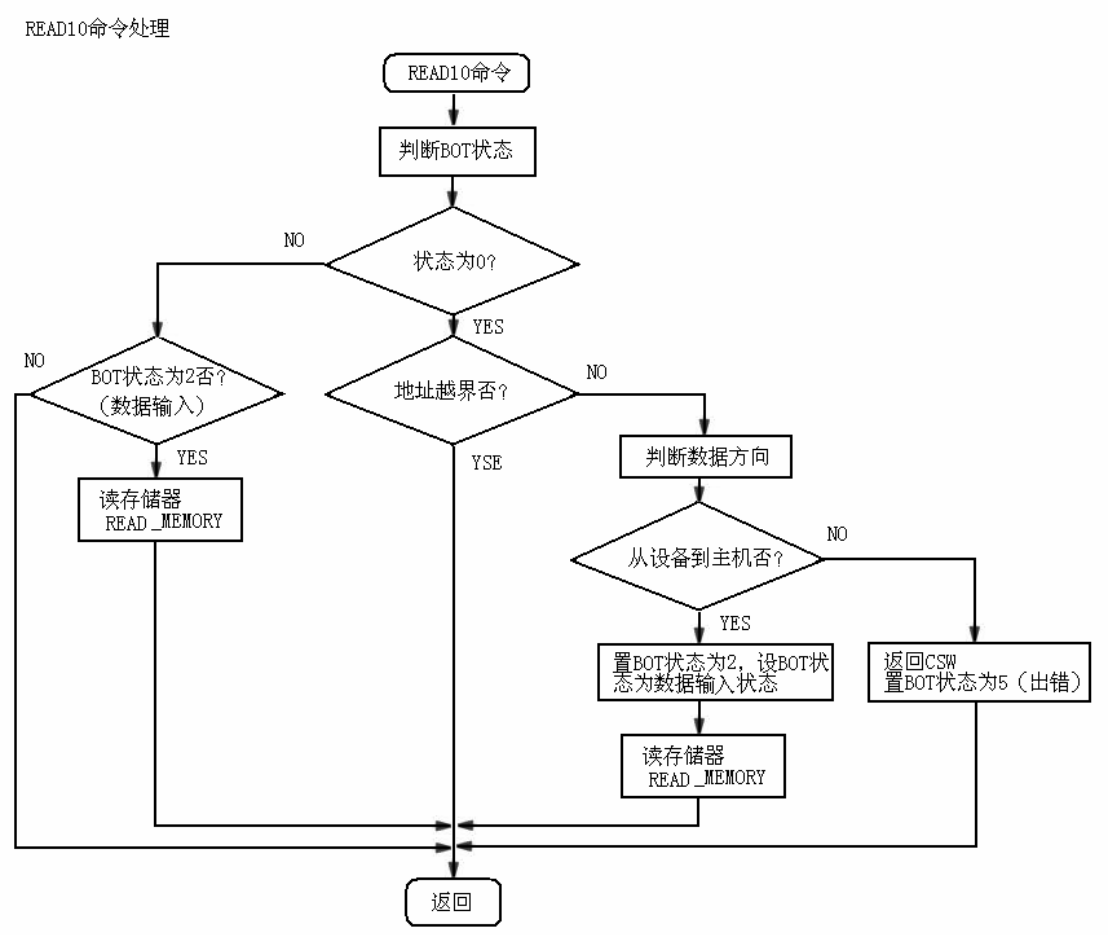

图 6、READ10 命令

（4）WRITE10 命令软件流程图（usb_scsi.c -> SCSI_Write10_Cmd(lun , LBA , BlockNbr)）

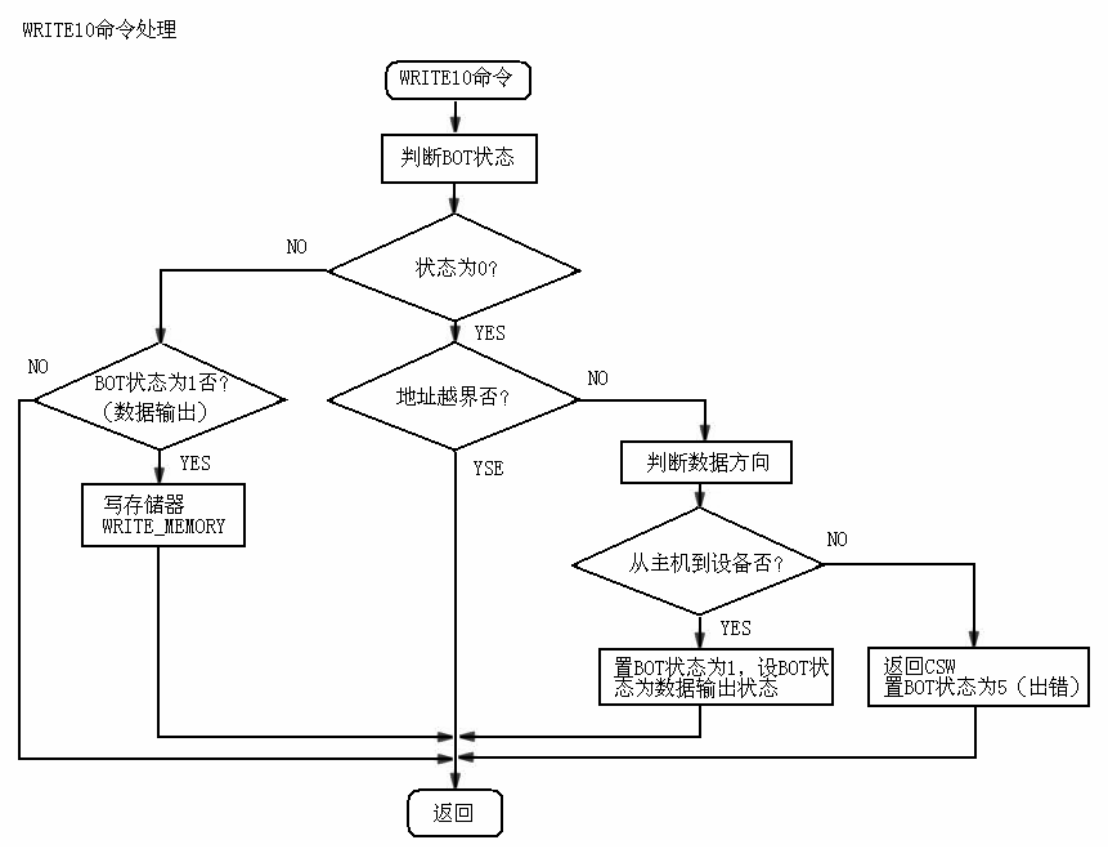

图 7、WRITE10 命令

（5）端点 1 输入中断流程图（usb_bot.c -> Mass_Storage_In()）

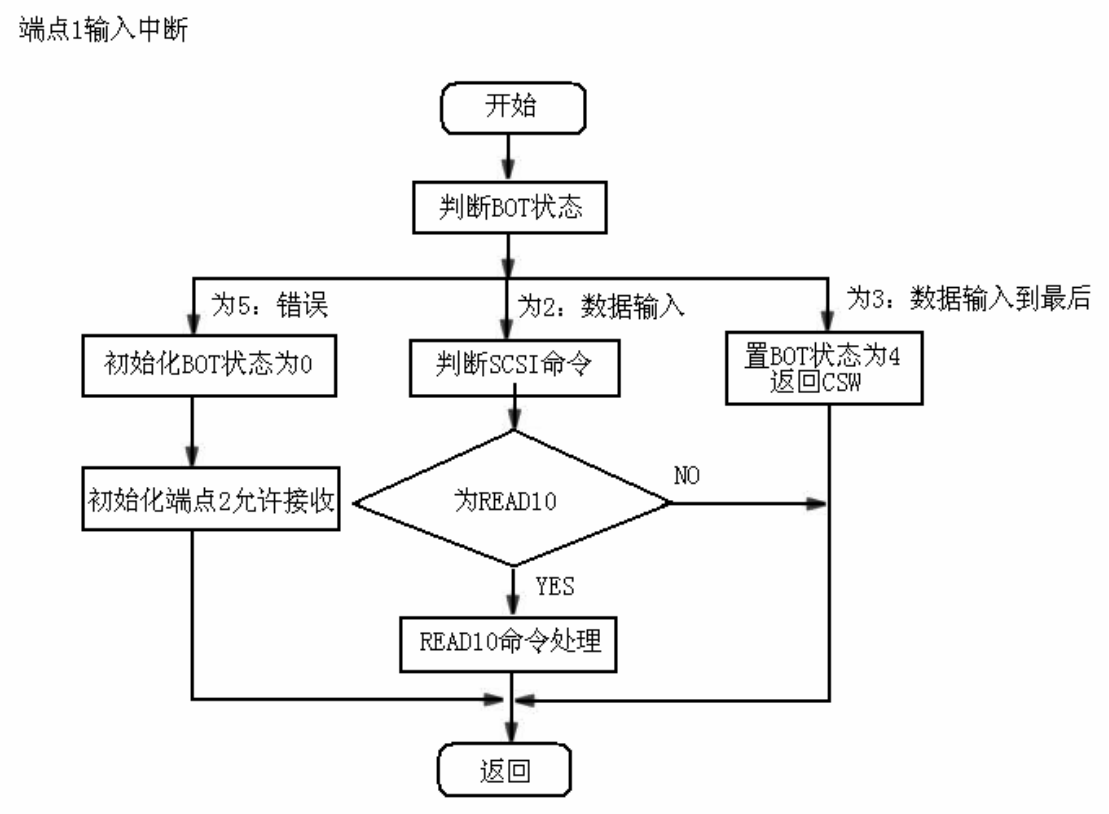

图 8、端点 1 输入中断

4、USB 对 Memory 的操作

在 MASS STORAGE 中，USB 对 MEMORY 的操作是以扇区（在 FAT 中，一个扇区为 512 字节）为单位的，而 USB 端点的最大包长为 64 字节，因此要发送或接收的数据会先放到内存，假设是 CPU 向端点 1 写数据，则首先从 FLASH 或 SD 卡中读取一扇区数据，再按最大包长分 8 次向 USB 端点发送。如果是端点 2 输出数据，则 CPU 将收到的数据先放至内存，并累加，当是 512 字节的整数倍时，再将数据写入 FLASH 或 SD，软件操作流程如下：

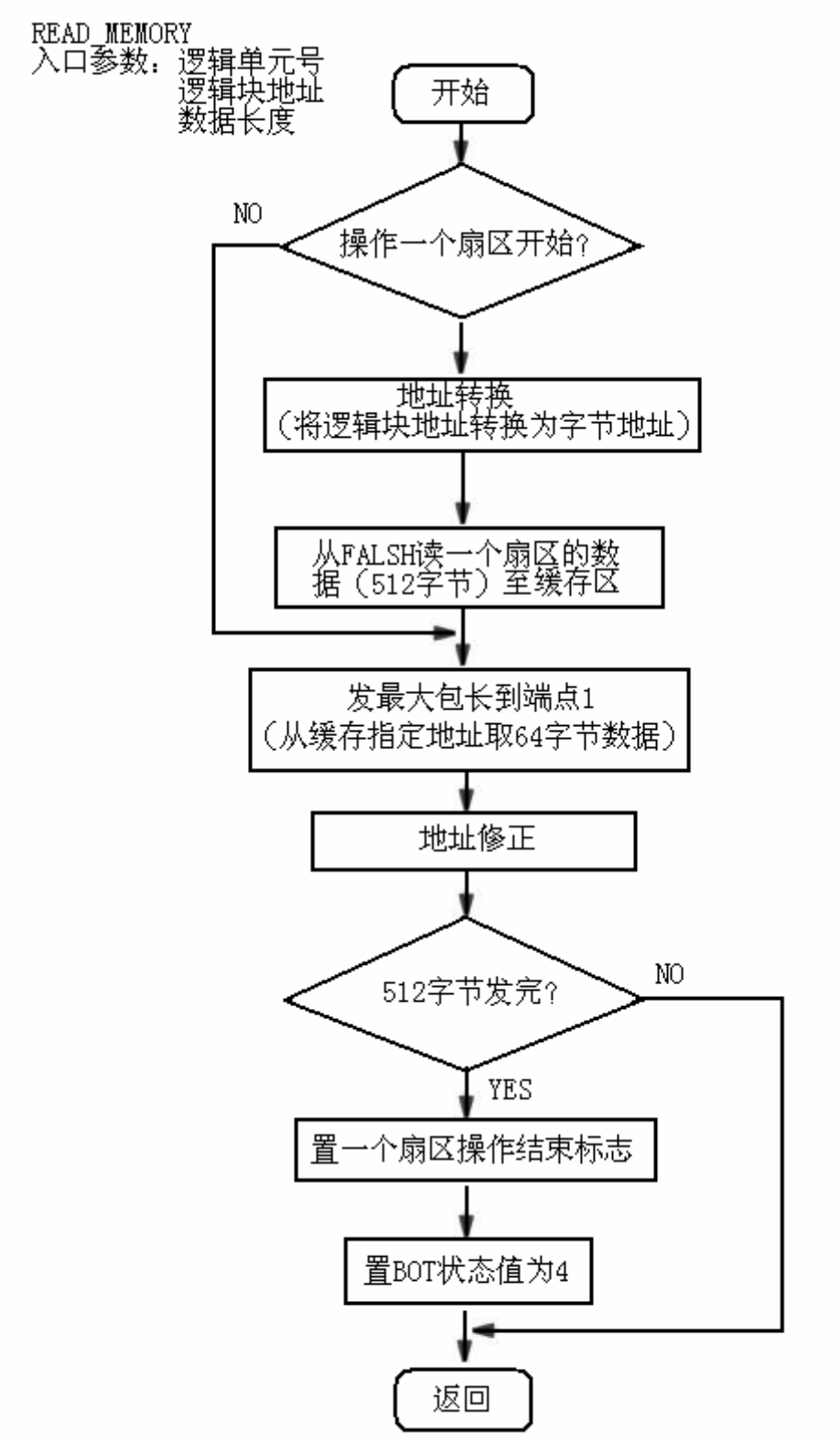

图 9、READ_MEMORY 流程图

在对 FLASH 或 SD 卡的读写操作中，该函数需要三个入口参数，第一个是逻辑单元号，用来告诉 CPU 主机是对哪个磁盘操作（假设有多个磁盘）；第二个是逻辑块地址，这里是指第几个扇区，而存储器的地址为字节地址，所以这个地址需要转换；第三个为数据长度，可以是多少个扇区，也可以是多少个字节，这个可以在 CBW 包中获得。

当数据发送完后，BOT 的状态值置为 4，进入到状态阶段。

五、 FLASH 读写

USB 对 FLASH 的操用是任意的，以扇区为单位，由于不是按连续地址写数据，所以 FLASH 的写函数中要有擦写管理。这里我是直接移植圈圈的 FLASH 读写函数，所以这里引用他的说明：

（以下文字出自圈圈的 BLOG）

由于 NAND FLASH 擦除时，只能按按块擦除，因此在写扇区时，首先要擦除一个块。在擦除块前，必须将块内其他数据复制出来，由于一个块比较大（128KB），无法在 MCU 内开辟如此大的缓冲区。只好借助该 NAND FLASH 内的页复制命令，将原来的块暂时复制到一个交换用的交换块中。但是如果仅用一个块作为交换的话，它就会被频繁擦写，因而寿命会大大降低。所以在该系统中，保留了 10 个块用来作为交换区，轮流使用。另外对扇区的连续写进行了优化，当连续写扇区时，就不必每次重复上面的操作，只有当地址跨块时，才需要重新擦除。连续写扇区的实现原理如下：当检测到扇区地址跨块时，就把原来的整块数据复制到交换块中，然后将该块内当前所写地址的前面部分页面复制到原来的块中，接着就从交换块中取出当前扇区地址所在页（使用页复制-随机写入命令），再把一个扇区的数据写入，并接着写入其他扇区，当扇区地址跨页时，就把该页写入到原来的块中，直到扇区被写完（当然如果写的过程中跨块了，还需要重复前面的块擦除以及复制过程）。接着把交换块中剩余的页再复制回原来的块中，这样一个连续写过程就完成了。

因为 NAND FLASH 在生产和使用过程中，会产生坏块，所以必须增加坏块管理。在该系统中，保留了 50 个块用做坏块管理。当上述的擦、写过程中，如果发现坏块，那么就把该块的地址重新影射到一个保留的块中。以后每次对该坏块地址操作时，都被重新定位到新的块地址。使用一个二维数组来保存该影射关系。二维数组的前半部分为坏块地址，后半部分为重新影射过后的块地址。每当发现坏块时，就需要重建这张表（主要是增加新的影射并排序，方便地址重新影射的二分查表法），并将其三份一样的写入到专门为此而保留的三个块中。之所以使用三个备份保存，是考虑到这些数据的重要性，因为一旦这个影射关系被破坏，后果将会是灾难性的。在保存这个表格时，做了特殊处理，首先标志他们准备擦除，然后才依次擦除并写入数据，这样即使在操作过程中突然断电，也至少有两个块的备份数据是可以用的，在系统开机初始化时，可以将这些数据恢复。数据使用了特殊标志（0x0055AAFF）以及累加和校验来判断是否有效。对于新出厂的 FALSH，在加载坏块表时，就会校验失败，程序就会假设没有坏块并初始化该数组后保存。每个备用块还由另外一个数组用来标志其状态（未用、已损坏、已用等）。当需要使用新的备用块时，就从该数组中查找未用的，并标志为已用。如果备用块本身是坏的，那么就标志它已损坏。该数组与坏块重影射表一并保存。此外还保存了当前坏块的数量。

地址的重新影射过程：当对一个地址进行读写操作时，首先要对其进行重影射。首先判断是否有坏块，当坏块数量为 0 时，就直接返回原来的地址即可。当坏块数量不为 0 时，先判断最后一次访问和本次访问的地址是否属

于同一页，如果属于，那么就直接影射到上一次影射过的块地址。如果不属于，那么就需要去查坏块影射表了。如果只有一个坏块，只要直接比较即可，不用查表。如果坏块数量大于 2，那么就需要查表。由于表中地址是按从小到大的顺序排列的，所以可以先和第一个和最后一个判断，如果不在该范围内，那么也不用重新影射，返回原来的地址即可。如果在该范围内，就使用二
分查表法查表，搜索它是否在坏块表中。如果是的话，就重新影射地址，并将这个地址保存，以备下一次重影射时地址未跨块直接使用。最大支持 50 个坏块，在最坏的情况下，该二分查表法需要判断 6 次。

六、 文件说明

1、USB 固件库文件

usb_conf.h USB 库文件配制;
usb_type.h USB 库文件类型声明，使 USB 库文件具有独立性;
usb_def.h USB 库文件公用的宏定义;
usb_regs.h USB 控制器寄存器描述;
usb_regs.c USB 控制器寄存器底层操作函数;
usb_init.c USB 控制器初始化;
usb_int.c USB 高优先级中断和低优先级中断处理函数，在本例中没有用到高优先级中级，所以去掉了;
usb_mem.c 这个函数用于将 USB 端点的数据传送给主机和主机的数据传送到 USB 端点;
usb_croe.c USB2.0 协议处理;

以上文件具有很强的独立性，除特殊情况，不需要用户修改，直接调用内部的函数即可.

2、USB Mass Storage Bulk Only 实现

以下文件也是出自 STM32 官方，但要根据实际的应用作修改.

usb_pwr.c USB 控制器的电源管理函数;
usb_istr.c USB 低优先级中断入;
usb_endp.c 非控制端点处理（大容量数据存储输入和输出函数）;
usb_prop.c Mass Storage 相关属性：mass 初始化、复位等等;
usb_bot.c BOT 状态机，CBW 解析和调用 SCSI 处理（批量数据输入输出的状态转换通过 BOT 状态机实现），
这个程序负责接收主机的 CBW 包，并解析，调用 SCSI 命令处理函数，返回 CSW;
usb_scsi.c SCSI 命令处理;
usb_desc.c USB 描述符;

七、 结语

第一次用 32 位的 ARM 单片机，第一次玩 USB，能够花两个多星期的时间实现这个 U 盘，心情还是蛮激动的，当然我参考了很多网友的和官方的代码；在英蓓特的 STM32 开发板中就有 MASS STORAGE 的例程，是对 SD 进行读写，对 NAND 不能操作，每次在 PC 端点 NAND 磁盘时，都提示要格式化磁盘，但总是格式化不成功。后来发现在写 NAND 的程序中，没有擦写管理，导致数据每次写入都失败，后来我移植了圈圈的 NAND 擦写管理函数，换了大页的 NAND FLASH，终于调试成功。

在调试的过程中，了解了 STM32 的 USB 控制器，了解了 USB MASS STORAGE 批量传输协议，学会使用 NAND FLASH，现将自已两个多星期来的学习成果进行总结，希望能与大家交流。

在这两个多星期来，得到了很多网友的支持，特别要感谢圈圈，帮助我解答了很多疑难问题，纠正了我很多错误的概念，使我能顺利的理清思路；还要感谢 21ICBBS 的网友们，还有香水城，给我解答了很多关于 STM32 的技术问题，还有很多不知名的网友，感谢你们无私贡献的资料。

由于初学，以上难免会有不对之处，还请大侠们批评指正。

Liu_xf
Eeliu88@gmail.com
2009-6-2