嵌入式 Flash 文件系统的设计与实现①

陈桂生　李志刚 (健雄职业技术学院 计算机工程系 江苏 太仓 215411)

摘　要： Flash 存储器是一种在嵌入式系统中日益普及的存储介质，它提供了高密度且成本相对较低的固态存储。使用 Flash 存储器需要很多技巧来确保数据可靠性并延长 Flash 器件的使用寿命。根据 Flash 存储器的特点提出了一种文件系统模型，可以合理且有效地利用 Flash 存储器，同时提供日志特性以增强使用该文件系统的嵌入式系统的可靠性。

关键词： flash 存储器；文件系统；设计；STR710F；K9F2808U0C

Design and Implementation of Embedded Flash File System

CHEN Gui-Sheng, LI Zhi-Gang

(Department of Computer Engineering, Chien-shiung Institute of Technology, Taicang 215411, China)

Abstract: Flash memory is an increasingly common storage medium in embedded devices. It provides solid state storage with high density at a relatively low cost. The usage of flash memory needs many techniques to provide reliability of data and to prolong the lifetime of flash devices. Based on the characteristics of flash memory, a file system model is put forward. This model can take advantage of flash memory reasonably and efficiently. It can also provide journalling features to enhance the reliability of embedded systems who use the file system.

Keywords: flash memory; file system; design; STR710F; K9F2808U0C

1　引言

随着嵌入式计算机技术的快速发展，各种应用对存储设备的要求也越来越高。嵌入式计算机系统由于受成本、功耗、体积等因素的影响而普遍采用大容量半导体存储介质作为其非易失性存储器。就非易失性半导体存储器而言，从原来的ROM、EPROM、EEPROM到现在的 Flash(又称闪存)，总的发展趋势是容量越来越大、可编程能力越来越强且单位成本越来越低。尤其是 Flash，其大容量及可编程的特性使其在大多数应用场合下完全可以代替磁盘作为嵌入式系统中的主要非易失存储设备，用于在断电情况下保存全部程序软件及数据，并在硬件复位时进行系统引导。

因此如何利用好 Flash 存储器关系到整个系统的功能、效率、可靠度甚至使用寿命。虽然目前已经有一些成熟且开源的 Flash 文件系统如 JFFS、YAFFS 等，但它们大多依附于操作系统环境(如 Linux)且功能复杂，需要的系统资源也比较多，不适合简单的嵌入式系统。本文针对这些问题提出了一种适合 Flash 存储器的文件系统模型，并在 STR710F+K9F2808U0C 的平台上实现示例系统，以及一个简化了的属性记录系统。

2　Flash存储器特点

严格地说，Flash 存储器是 EEPROM 的一种，分为 NOR Flash 和 NAND Flash 两大类。传统的 NOR Flash 可以直接通过通用总线访问，一般容量较小，单位成本较高；新的 NAND Flash 存储密度更高，容量普遍较大，单位成本较低，但只能通过一个 8 位总线和一些专有的控制线路及协议访问。其主要特点如下：

① Flash 存储器可以像其它 ROM 那样直接读取数据，但写入数据的操作与传统的 EEPROM 不同。

① 收稿时间:2009-09-01;收到修改稿时间:2009-10-30

Flash 的写操作只能有选择地把一些位从逻辑“1”置为逻辑“0”，而无法将逻辑“0”置为逻辑“1”。由于这个特性，如果向同一个单元写入多次则该单元最终的值将是曾经写入的所有值逻辑与的结果，而非最后一次写入的值。

② 为了向已经执行过写操作的单元写入新的内容但不会因原来的内容而改变，必须事先对单元执行擦除操作。Flash 的擦除操作是将指定区域内的全部逻辑“0”复位到逻辑“1”，擦除以区块(Block)为单位，必须整块擦除。由于擦除过的区块中所有位均被置为逻辑“1”，而任何值和逻辑全“1”的值相与都不会改变，因此接下来写入的内容就会被忠实地记录下来。

③ 一般 NOR Flash 区块在一次擦除后可以进行任意次随机写入，且写入操作以字为单位(通常为 16 位或 32 位)。而 NAND Flash 区块在一次擦除后只能进行有限几次写入，写入操作以页面(Page)为单位(通常为 512 字节或 2048 字节)，可以一次只写入页面的一部分。NAND Flash 每页都有额外的带外空间(OOB，512 字节的页面配有 16 字节的 OOB，2048 字节的页面配有 64 字节的 OOB)。

④ Flash 中每块的可重复擦除次数是有限的(通常 100,000 次)，当达到寿命后，对该区块的擦除可能会失效，有些位将无法从逻辑“0”复位到逻辑“1”。

STR710F 片内包含 256KB+16KB 的 NOR Flash，擦除区块大小不定长[1,2]。

K9F2808U0C 是总容量 16MB 的 NAND Flash 存储器，擦除区块大小为 16KB[3]。

3 Flash文件系统模型

鉴于上述 Flash 存储器的特点及嵌入式系统特性，要想设计一个针对 Flash 的稳定可靠的文件系统，必须考虑以下问题[4]：

① 写 Flash 前需要擦除，且不能擦除比区块更小的单位，因此对文件的修改不能简单地直接覆盖；

② 在对文件操作的整个过程中都有可能发生意外(例如嵌入式系统常见的断电)，文件系统需要有从意外中恢复的能力，并具有一定的日志(Journalling)特性以确保数据完整性；

③ 由于 Flash 区块可重复擦除次数有限，不适合采用通常文件系统中的集中式 FAT 结构，否则存放 FAT 的区块将会比其它数据区块先行损坏，而一旦 FAT 损坏则整个文件系统都将崩溃；

④ 基于同样的原因，需要进行磨损控制，尽量限制并平衡各区块的擦除次数，避免在文件系统使用量较小的情况下只利用较前端的区块并反复擦写；

⑤ 对于已经损坏的块，文件系统必须有能力检测并标记；

⑥ 需要对处理时间与占用内存空间这对矛盾进行权衡。

以下是在 K9F2808U0C 上实现示例系统的主要部分：

3.1 文件分配表(FAT)

FAT 每个项目被设计为 16 位，用于跟踪各簇的使用情况，每个 FAT 项目对应文件系统中的一个簇(Cluster)。

FAT 项目的低 15 位是一个索引，用于指向本簇所对应文件的下一个簇的编号，最高位用于标示该簇是文件的起始簇还是后续簇，另外几个特殊的值用于标示特殊状态的簇(如表 1 所示)。

表 1　FAT 项目含义描述

FAT项目值域	对应簇描述
0x0000 ~ 0x7FFC	本簇不是文件的起始簇，低 15 位指向下一簇
0x7FFD	本簇不是文件的起始簇，但是文件的结束簇
0x7FFF	本簇对应的区块是脏块，需要擦除才能继续使用
0x8000 ~ 0xFFFC	本簇是文件的起始簇，低 15 位指向下一簇
0xFFFD	本簇是文件的起始簇，也是文件的结束簇(该文件只占 1 簇)
0xFFFE	本簇对应的区块是坏块，不再使用
0xFFFF	本簇对应的区块是空闲块，已经擦除成功，可以直接使用

可以看到，理论上这种 FAT 设计可以支持最大 215 – 3 = 32765 个簇的文件系统。

对于 K9F2808U0C，定义簇大小为擦除块大小(16KB)，这样整个 16MB 空间就包含 1024 个簇。

FAT 只存在于内存中，不直接存放在 Flash 存储器上。系统初始化时将根据簇信息重新构建 FAT。

3.2 簇信息块(CIB)

簇信息块(CIB)用于记录每个簇的控制信息。对于文件起始簇，也保存了文件的相关信息。

K9F2808U0C 每个擦除区块都配有共 512 字节的 OOB[3]。为了避免集中保存控制信息的部分介质损坏而导致整个文件系统崩溃，该文件系统将 CIB 分布在各个簇相应的 OOB 内。

文件系统运行时只在内存中维护当前的 FAT，CIB 只在需要的时候才从相应的区块中读取且使用后不缓存，这样在本例中整个文件系统只需占用约 2KB 的 RAM 空间。

CIB 的结构定义如下：

```
typedef struct tagCIB {
    u8      file_info[256], ecc[96];
    u16     sn_file, sn_cluster;
    u16     ofs, size;
    u32     date, time;
    u16     chksum, sign;
} CIB;
```

file_info 用于保存簇所对应的文件信息。对于占用多簇的文件，只有起始簇保存文件信息，其它簇该域为空(逻辑全“1”)。文件系统本身并不关心该域保存的内容及结构，只在枚举文件时提供给用户，并在用户要求更改文件信息时进行更新。ecc 提供了对数据区的纠错码(Error Correcting Code)，每 512 字节数据有 3 字节 ECC。

sn_file 和 sn_cluster 分别是两个序列号。文件系统依靠 sn_file 而非 file_info 区分不同的文件，每个文件都有唯一序列号，文件所占用的所有簇都要指明同一个序列号。而 sn_cluster 用于区分 Flash 存储器上相同文件(sn_file 相等)相同位置(ofs 相等)的簇，在文件写入及更新时用到，以实现日志功能。

ofs 是一个偏移量，表明该簇是所属文件的第几个簇。size 用于标示该簇中实际数据的大小，对于文件的结束簇，size 的值可能小于簇大小，否则 size 等于簇大小，而一个文件所有簇的 size 之和即文件大小。date 和 time 用于标示该簇内容写入的系统时间，一个文件所有簇写入时间的最大值即该文件的最后修改时间。

chksum 用于校验 CIB 中除 sign 之外的域，校验失败则认为此块为脏块。

sign 处于 CIB 的末尾，标示了簇所在区块的状态：0xFFFF 代表空闲块，0x7E7E 代表正在使用的块，0x0000 代表坏块，其它任何值(典型值为 0x3C3C)代表脏块(如图 1 所示)。

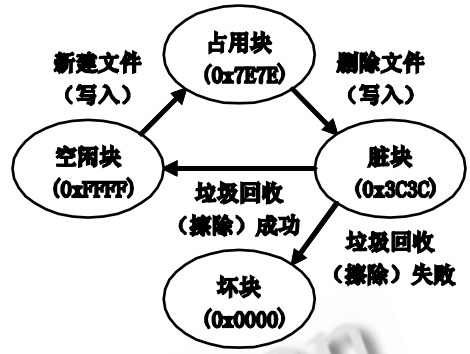

图 1　区块状态转换

空闲块由于刚经历过擦除操作，CIB 所有域都是逻辑全“1”，因此计算校验和是没有意义的。

3.3 初始化操作

文件系统初始化时会依次读取每个 Flash 区块上的 CIB，并在内存中建立 FAT。整个初始化过程需要几个临时缓冲区以加速 FAT 的建立，但初始化完成后，内存中只保留 FAT，其它缓冲区被释放。

CIB 扫描过程中，遇到空闲块和坏块则直接填写 FAT 项目，不再做进一步处理。否则对 CIB 进行校验，校验不通过则当作脏块填入 FAT。

对于校验通过的块，按照 sn_file 的值分类，再根据 ofs 排序并建成 FAT 中的簇链表。对于 sn_file 和 ofs 两个域值都相同的簇，通过比较 sn_cluster 的大小决定哪个簇是当前有效的，并在 FAT 中把已经失效的簇标为脏块。

3.4 写策略

由于一个文件的多个簇在 Flash 中并没有显式地形成链表结构，因此对文件的追加很容易实现，只需新建立一个簇，并在 ofs 域填入原来结束簇的 ofs + 1，并更新相应的 FAT 便可。

对于更新文件内容的操作(包括修改 file_info 域)，虽然可以通过修改 Flash 内容来完成(先擦除块，再写入新的内容)，但这样的策略是有隐患的。例如一旦擦除后写入前系统断电，不仅文件原来的内容丢失，还会破坏文件系统的完整性。另外该策略也不利于磨损控制，如果用户需要经常修改某个文件，反复擦写某些区块会缩短这些区块的使用寿命。

因此该文件系统执行更新操作时，并不修改 Flash 上原来的簇内容(包括 CIB)，而是另外建立一个新簇，该簇与原来的簇具有相同的 sn_file 和 ofs 值，但

sn_cluster 是原来的值加 1。文件系统规定 sn_cluster 值最大的簇为当前有效的,sn_cluster 值较小的簇被认为是过期的，并被当作脏块对待。

当然，sn_cluster 是一个 16 位值，因此对同一个簇更新 65536 次后，sn_cluster 会绕回到 0。解决这个问题的关键在于文件系统最多只能有 32765 个簇(对于 K9F2808U0C 示例系统,这个值是 1024)，因此即使在极限情况下 sn_cluster 也有至少 32771 个连续的序号空间来隔离 Flash 中 sn_file 和 ofs 相同的最新的簇和最老的簇，更老的簇已经被回收，不会在 Flash 存储器中继续存在。

写簇的操作分三步:首先将 CIB 中 sign 置成占用块状态，然后写入簇内容(文件数据)，最后写入 CIB 中的其余字段。对于删除或者截断文件的操作，文件系统直接将 sign 置成脏块状态。

这种写策略使得写簇操作中的任何一步系统断电，复位初始化后文件系统都能识别出老的簇并认为是当前有效的，因为它并没有被修改，而新簇则由于写入过程没有完整执行，CIB 校验不会通过，被文件系统当成脏块。

由此可见，这种日志特性可以帮助实现写操作的原子性和文件系统的完整性。

3.5 垃圾回收

无论是新建文件、追加文件还是更新文件，文件系统都要从空闲块中申请新的簇空间，当没有可用的空闲块时，文件系统就会启动垃圾回收操作，将 Flash 上所有标为脏块的区块统一擦除,并在FAT中标示为空闲块。

垃圾回收必须统一进行，不能只回收一部分区块(除非回收过程中发生意外),目的是为了尽量减少并平衡各区块的擦写次数。

如果垃圾回收过后仍然没有足够的空闲块以执行写操作，则说明文件系统已满，写操作失败(即便该写操作只是更新文件，并没有增加文件大小)。

3.6 关于文件信息域

系统为每个文件保留了一个file_info域用于存放文件信息，用户可以自定义该域的内部结构和内容。

例如用户可以在此存放文件名、文件属性、访问权限、文件建立时间等基本信息，也可以利用该域实现复杂的树形目录功能。而对于一些数据采集系统，可能并不需要文件名，用户只需在 file_info 中保存一个简单的文件序号就够了，例如数码相机等。

自定义文件信息的特性使得文件系统的应用十分灵活。

3.7 小结

由上文所述可知，该文件系统模型有以下特点:

① FAT 在每次系统初始化时构建到内存中,因此不单独占用 Flash 存储器空间，且访问速度快，尽管这会增加系统初始化时间和一定的内存开销;

② 分布式的 CIB 存储使得文件系统不会因为某些区块的损坏而受到致命影响;

③ 特别设计的区块状态编码可使状态转换时无需对区块进行擦除(垃圾回收除外——它只需擦除操作)，这不仅使操作速度加快，且降低了对 Flash 存储器的损耗;

④ 具有日志性质的写策略可以最大限度地保证文件系统的完整与数据一致性;

统一的垃圾回收使得各区块的磨损得到较好的平衡。

4 Flash属性记录系统

很多嵌入式系统只想利用 Flash 的非易失性来保存一些简单的数据，例如配置信息、属性表等，它们不需要一个复杂的文件系统，只要保证数据能够随时安全有效地存取即可。针对这样的需求，我们可以设计一种简单的属性记录系统。

属性记录系统是一种简化了的文件系统，只不过访问单位是属性记录而非文件记录。

这里的属性记录是指(名称，类型，数据)三元组，类似 Windows 操作系统的注册表。名称(Name)用于索引不同的属性记录; 类型(Type)定义了该属性的数据该如何解释，例如整数、浮点数、字符串等; 数据(Data)保存了属性的值，其长度与类型有关，可以是定长的或者变长的。

示例系统利用 STR710F 内部 Flash 的 Bank1 建立属性记录缓冲区。这块 Flash 共有 16KB 大小，分为两个扇区(Sector)，每扇区 8KB，可单独擦除[2-5]。以下为设计要点:

4.1 属性缓存与属性描述表

属性缓存是所有属性在 RAM 中的集中映射，用户通过该缓存直接读取属性数据。其表现形式为一个结构体变量，其中每个成员变量对应一条属性记录。每个属性的类型都是预先由用户定义好的，运行过程

中无法改变。

用户需要对所有属性进行编号，并另外定义一张属性描述表，用于定义每个属性的类型以及与属性缓存成员变量的映射关系。

例如用户有三个属性需要记录，分别是本地 IP 地址(32 位整数)、服务器域名(字符串)和服务器端口号(16 位整数)，则属性缓存与属性描述表定义如下：

```
enum ATTR_NAME{
    AN_LOCAL_IP,    AN_SERVER_NAME,    AN_SERVER_PORT,    // 为三个属性编号(0、1、2)
    AN_COUNT            // 属性个数(3)
};
struct {
    u32        local_ip;
    char       *server_name;
    u16        server_port;
} AttrCache;            // 定义属性缓存
const struct ATTR_DESC {                    // 属性描述符结构
void       *ptr;      // 属性缓存位置
u8         len;       // 非 0 表示属性具体长度，此时 ptr 需指向缓存中 len 长度的成员变量
                      // 0 表示属性长度可变，此时 ptr 需指向缓存中用于访问属性数据的指针
} AttrDescTab[AN_COUNT] = {                 // 定义属性描述表
{ &AttrCache.local_ip,          4 },        // 4 字节属性，对应 32 位整型变量
{ &AttrCache.server_name,       0 },        // 可变长属性，对应指针变量
{ &AttrCache.server_port,       2 }         // 2 字节属性，对应 16 位整型变量
};
```

4.2 属性日志

同 Flash 文件系统类似，属性在 Flash 中以日志条目的形式依次存放。一个日志条目包含两部分：属性编号，用于标示该条目属于哪个属性，占 1 字节；属性数据，存放属性的具体值，长度由属性描述表的 len 域确定。对于可变长属性，数据的第一字节标示了后面数据的实际长度，因此可变长属性最大长度为 255。

有效日志条目的属性编号最高位须为 0，低 7 位为实际编号，因此该系统最多支持 128 个不同属性。

属性记录系统在同一时刻只使用 STR710F Flash Bank1 两个扇区的其中之一来存放日志，另一扇区用作备份。日志的开头用一个字节来标示日志的序列号及其反码(各占 4 位，其中反码用于校验日志的有效性)，因此日志的总量不能超过扇区大小 8KB － 1B = 8191 字节。

4.3 初始化操作

属性记录系统初始化时，首先根据序列号判断哪个扇区的日志是当前有效日志。方法是：

根据两个扇区的序列号高低 4 位是否互为反码来判断其有效性。如果两个扇区都无效，则任选一个扇区建立新的日志；如果只有一个扇区有效，则认为该扇区中的日志为当前有效日志；如果两个扇区都有效，则认为序列号较大的扇区中的日志为当前有效日志。

如果当前有效日志存在，系统将依次读取日志条目，根据属性编号及描述表将数据填入属性缓存的相应成员。如果读到多个相同属性的条目，数据以最后一个条目的记录为准。

4.4 写策略

属性记录系统的写策略类似 Flash 文件系统中更新文件的操作，修改属性并不会直接更改 Flash 中原有的值，而是在日志末尾追加包含新值的条目，同时更新属性缓存中的值。

写新条目时，先写入最高位为 1 的属性编号及属性数据，然后验证写操作的结果，在确认写入正确后，将属性编号的最高位清 0 以示有效。

如果日志所在的扇区没有足够的空间，则将日志记录转向另一个扇区：首先将另一扇区擦除，然后将属性缓存中的所有属性数据以日志条目的形式依次写入新扇区以建立新的日志。在确认所有写操作均正确后，在新日志的头部填入新的序列号及其补码以示有效。新序列号的值为老序列号加 1，这里同样存在序列号计数到 16 会绕回 0 的问题，解决方法类似 Flash 文件系统中的 sn_cluster。

无论写新条目还是建立新日志，这种类似 Flash 文件系统的写策略同样保证了写操作的原子性和属性记录系统的完整性。

4.5 小结

尽管属性记录系统的结构设计与实现较文件系统

(下转第 62 页)

(上接第 40 页)

要简单得多，但它依然具备文件系统最重要的优点：安全可靠、原子写入操作、对 Flash 的磨损平衡。

5 结语

本文提出的 Flash 文件系统只需在 RAM 中驻留 FAT，而属性记录系统只需在 RAM 中维持属性缓存，因此 RAM 开销都比较小，非常适合简单的、RAM 资源紧张的嵌入式系统应用。

由于篇幅所限，本文只介绍了 Flash 文件系统模型及其衍生的属性记录系统的基本框架实现，读者可以根据这个基本框架实现更能满足具体需求的 Flash 文件系统。

目前本文所实现的文件系统和属性记录系统均应用于一个图像监控系统中，经过大量测试证明其稳定可靠。

参考文献

1 STMicroelectronics. RM0002 STR71xF Microcontroller Family Reference Manual Rev 2[2008-06-08]. http://www.st.com/stonline/products/literature/rm/ 13691.pdf

2 STMicroelectronics. UM0116 STR7 Family Flash Programming User Manual Rev 4[2006-09-20]. http://www.st com/stonline/books/pdf/docs/ 11130.pdf

3 SAMSUNG Electronics. K9F2808U0C 16M x 8 Bit NAND Flash Memory Rev 2.5[2003-10-10]. http://www.datasheetarchive.com/pdf-datasheets/Datasheets-29/DSA-568169.pdf

4 Woodhouse D. JFFS: The Journalling Flash File System[2001-10-10]. http://sources.redhat.com/jffs2/jffs2.pdf

5 沈建华. STR71x 系列 ARM 微控制器原理与实践.北京:北京航空航天大学出版社, 2006.