《振南 **znFAT--**嵌入式 **FAT32** 文件系统设计与实现》一书

【上下册】已正式出版发行

全国各渠道全面发售

（在当当、京东、亚马逊、淘宝等网络平台上搜索

关键字"**znFAT**"即可购买，各地实体书店也有售）

此书是市面上 唯一 一套详细全面而深入讲解嵌入式存储技术、**FAT32** 文件系统、**SD** 卡驱动与应用方面的专著。全套书一共 **25** 章，近 **70** 万字。从基础、提高、实践、剖析、创新、应用等很多方面进行阐述，力求通俗，振南用十年磨一剑的精神编著此书，希望对广大工程师与爱好者产生参考与积极意义。

此书在各大电子技术论坛均有**长期的**「抢楼送书活动」，如 21IC、elecfans 等等。

振南的【ZN-X 开发板】是市面上唯一全模块化、多元化的开发板，可支持 51、AVR、STM32(M0/M3/M4)

详情请关注 **www. znmcu. cn** （振南个人主页!!）

第11章

文件匹配，目录扩展：文件名匹配与目录的簇链结构

前面引出了PC平台并完成了汉字字形显示实验，但是似乎一直遗留着一个问题，大家有没有发现！提示一下：在前面的实验中，我们向Analyze_FDI函数传入的文件目录项的首地址是从哪来的？对！是从首目录扇区的固定位置上获取的，而这个位置是通过WinHex软件看到的。但是，实际复制的文件所对应的文件目录项又怎么会只放在固定的位置上呢？应该是被文件系统分配到某个适当的位置上去了。进一步来说，我们应该根据给定的文件名来确定与之匹配的文件目录项的位置，进而完成文件信息解析及数据读取等操作。这样，我们将初步完成本书前面所说的“文件名与数据的映射”（第3章）。其实，在FAT32中只要与文件名一沾边，那么涉及的内容就多了。本章还是先来处理一些比较简单的情况，至于那些复杂的内容我们会在后面和下册的章节再来详细讲解。

11.1 文件的匹配

在文件的匹配过程中，我们将完成两项工作：文件目录项的搜索与文件名的匹配（这里仅处理8·3格式的短文件名，至于长文件名请参见下册相关章节）。

11.1.1 文件目录项的搜索

一个目录的簇中包含了很多的文件目录项（比如前面实验中TEST.MP3、HZK16.DAT的文件目录项就在首目录簇中）。对文件目录项的搜索其实就是对它们进行遍历的过程，可以通过图11.1来说明。具体的代码实现也很简单，只是几个循环而已，程序如下（znFAT.c）：

```
sec_temp = SOC(Cur_Clust);                                    //当前簇首扇区
for(iSec = 0;iSec<(Init_Args.SectorsPerClust);iSec ++ ) //读取簇内各扇区
{
 znFAT_Device_Read_Sector(sec_temp + iSec,znFAT_Buffer);
 pitems = (struct FDIesInSEC * )znFAT_Buffer;
 for(i = 0;i<16;i ++ )                                        //访问扇区中各文件目录项
```

```
{
 pitem = &(pitems->FDIes[iFDI]);                    //指向文件目录项数据
 //匹配操作
}
}
```

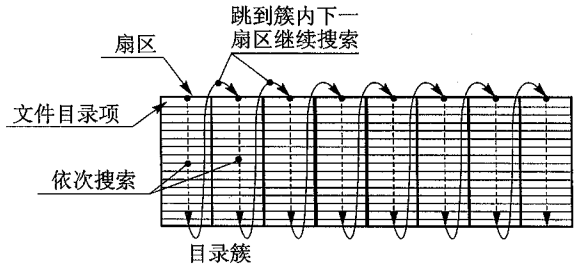

图 11.1　对目录簇中的文件目录项进行搜索

11.1.2　8·3 短文件名(SFN)

有人说,文件名的匹配只不过是字符串比较而已,有什么难的?其实不然,单单是 SFN 都要经过一些变换,最终才能进行比较。如何变换?我们先来了解一下什么是 8·3 格式的短文件名。

"8·3"的意思是说文件名中的主文件名占 8 个字节,扩展名占 3 个字节。所以,普通的文件目录项只能表达短文件名(言外之意就是说还有非普通的文件目录项,后文会说到)。其实 8·3 格式的短文件名最初是在 DOS 操作系统上使用的。我们应该有过这样的经历,在使用一些较早期版本的 DOS 时,一些较长的文件名会自动截断为 AABBCC~1.TXT 这样的形式,这就是因为它只支持短文件名,而无法处理长文件名(LFN)的缘故。

前面提到过,文件目录项中的文件名字段只能记录大写字符(这是 FAT32 的约定),难道 8·3 的短文件名就只能是大写吗?比如 AAA.TXT 或 ABC.MP3。振南刚开始也是这样认为的,但后来发现有些文件目录项竟然能够表达 aaa.txt 或 AAA.txt,于是纳闷:"它到底是怎么表达小写的呢?"将 AAA.TXT、aaa.TXT、aaa.txt 及 AAA.txt 的文件目录项进行了对比,才恍然大悟,请看图 11.2。显然,这些文件目录项中唯独不同的就只有第 12 个字节,它到底代表什么意义?来回顾一下在第 6 章中介绍的文件目录项的结构定义:

```
struct FDI
{
    UINT8 Name[8];          //文件名,不足部分以空格补充
    UINT8 Extension[3];     //扩展名,不足部分以空格补充
    UINT8 Attributes;       //文件属性
```

```
    UINT8 LowerCase;          //指示主文件名与扩展名是否全为小写
    ......
};
```

aaa.txt

41	41	41	20	20	20	20	20	54	58	54	20	18	3C	AA	9D
2F	40	2F	40	00	00	A1	9D	2F	40	00	00	00	00	00	00

AAA.TXT

41	41	41	20	20	20	20	20	54	58	54	20	00	3C	AA	9D
2F	40	2F	40	00	00	A1	9D	2F	40	00	00	00	00	00	00

AAA.txt

41	41	41	20	20	20	20	20	54	58	54	20	10	3C	AA	9D
2F	40	2F	40	00	00	A1	9D	2F	40	00	00	00	00	00	00

aaa.TXT

41	41	41	20	20	20	20	20	54	58	54	20	08	3C	AA	9D
2F	40	2F	40	00	00	A1	9D	2F	40	00	00	00	00	00	00

图 11.2　4 种小写 8·3 短文件名的文件目录项比较

我们当时忽略了这个字节，关于这个字节在很多资料中也都没有提及。如图 11.2 中所看到的，它的第 3 和第 4 位分别用于指示主文件名和扩展名是否全为小写。注意是“全为小写”，如果是 aAa.Txt 这种大小写混编的文件名它仍然无法表达（它属于长文件名）。

另外，在 8·3 短文件名中不允许使用“\/：*?"<>|”这些非法字符，因为它们在文件名解析的时候是有特殊意义的。

现在可以来总结一下，怎样的文件名才算是合法有效的 8·3 短文件名？它要满足下面这 3 个条件：

- 主文件名长度 0 ＜m≤8，扩展名长度 0≤n≤3；
- 没有非法字符；
- 主文件名或扩展名均为大写或小写。

依照这 3 个条件可以实现对 SFN 合法性的校验。对于一个文件名，首先要保证它是合法的，否则后面的工作将是徒劳的。可以使用下面的这 3 个函数共同完成对 SFN 合法性的校验，它们与上面的 3 个条件一一对应（函数的具体实现请参见 zn-FAT 源代码）：

```
UINT8  Check_SFN_Illegal_Length(INT8 * pfn)      //检查 SFN 的长度
UINT8  Check_SFN_Illegal_Char(INT8  * pfn)       //检查 SFN 有没有非法字符
UINT8  Check_SFN_Illegal_Lower(INT8  * pfn)      //检查 SFN 的大小写合法性
```

11.1.3　SFN 的匹配

要完成短文件名的匹配，首先要将文件目录项中的文件名字段转换成标准 8·3 格式的短文件名，然后再将它与目标文件名进行比对，看是否相等（文件名均使用大写）。为了更好地说明这一过程，我们来看下面的例子，如图 11.3 所示。

图中给定了一个文件名 abc.txt，要找到它所对应的文件目录项，首先将文件名转为了大写，即 ABC.TXT。为什么要转为大写？如果 abc.TXT、ABC.txt 都要转

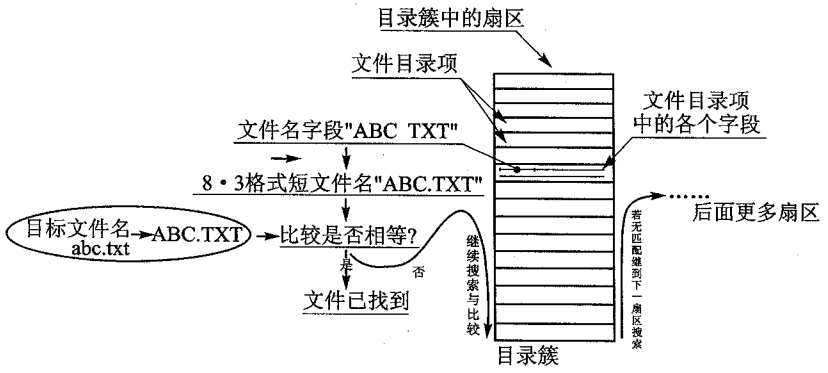

图 11.3 短文件名匹配的过程

为大写，那最终找到的文件目录项岂不都是同一个文件目录项吗？没错！在 FAT32 中，短文件名其实是不区分大小写的，它认为上面的这些文件名都是同名，因为这些文件名在文件目录项中的表达都是“ABC TXT”。

至于如何将文件目录项中的文件名字段转换为 8·3 格式的短文件名，其实很简单，就是一个字符串处理的过程(去掉多余的空格，再加个点儿就可以了)。我们编写了 To_File_Name 这个函数来完成这一功能，定义如下(具体实现请参见 znFAT 源代码)：

```
UINT8 To_File_Name(UINT8 * name_in_fdi,INT8 * pFileName)
```

其中，name_in_fdi 是指向文件目录项的文件名字段的指针；pFileName 是指向最终产生的 8·3 短文件名的指针。

最后，就是文件名的比较了，实质上就是字符串的比较，请看下面这个小函数(znFAT.c)：

```
UINT8 File_Name_Match(INT8 * fn1,INT8 * fn2)
{
 UINT8 fn1_len = 0,fn2_len = 0;          //用于记录两个字符串的长度
 fn1_len = StrLen(fn1);
 fn2_len = StrLen(fn2);
 if(fn1_len!= fn2_len)  return 0;        //如果长度不同，则必不相等，直接返回匹配失败
 while(fn2[fn2_len] == fn1[fn1_len]) //比较每个字符
 {
  if(0 == fn1_len)  return 1;            //返回匹配成功
  fn1_len -- ;
  fn2_len -- ;
 }
 return 0;
}
```

看了上面的代码，可能有人会比较纳闷：“C 语言里有现成的字符串比较函数

strcmp，为什么不用呢?”确实！（其实我们前面检验文件名中的非法字符时是有现成的函数 strchr 可用的）刚开始的时候，振南也直接调用了很多现成的函数。但后来发现，在不同的平台上 C 语言库函数也略有不同：可能有些函数是没有的，或者函数的定义不同。结果，总是牵制于一些库函数而使代码的移植性不好。因此，在 znFAT 中读者会发现，振南没有 include 任何 C 语言库的头文件，所有功能均亲自编写代码来进行实现。整套代码自成体系，绝不依赖于任何现成函数，使得 znFAT 可以不加或略加修改即可运行于很多不同的平台之上。

11.1.4　目录簇的拓展

有了文件名匹配函数，再加上前面的文件目录项搜索，我们就可以通过文件名来定位文件目录项的位置了，进而实现文件名与数据的映射。程序整合完善一下（znFAT.c）：

```
UINT8 Find_File_With_SFN(INT8 * filename)
{
UINT32 Cur_Clust = 2;                          //当前簇为首目录簇，即第 2 簇
 UINT32 sec_temp = 0;
 UINT8 iSec = 0,iFDI = 0;
 INT8 temp_filename[13];
 struct FDIesInSEC * pitems;                   //指向目录簇簇扇区的指针
 struct FDI * pitem;                           //指向文件目录项的指针
 //检查文件名合法性，若非法则直接返回，不再进行后面的处理
 if(Check_SFN_Illegal_Length(filename)) return 0;
 if(Check_SFN_Illegal_Char(filename)) return 0;
 if(Check_SFN_Illegal_Lower(filename)) return 0;
 sec_temp = SOC(Cur_Clust);                    //当前簇首扇区
 for(iSec = 0;iSec<(Init_Args.SectorsPerClust);iSec ++ )
 {
  znFAT_Device_Read_Sector(sec_temp + iSec,znFAT_Buffer);
  pitems = (struct FDIesInSEC * )znFAT_Buffer;
  for(iFDI = 0;iFDI<16;iFDI ++ )               //访问扇区中各文件目录项
  {
   pitem = &(pitems->FDIes[iFDI]);             //指向一个文件目录项数据
   if(CHK_ATTR_FILE(pitem->Attributes) && (0XE5!= pitem->Name[0]))
                                               //文件目录项属性为文件，并且没有被删除
   {
    To_File_Name(pitem->Name,temp_filename);//将 FDI 中的文件名字段
                                               //转为 8·3 格式短文件名
    Str2UpCase(filename);                      //将文件名所有字符转为大写
    if(File_Name_Match(filename,temp_filename))//文件名匹配
```

```
                                                    //filename 为目标文件名
        {
      //对匹配的文件目录项进行处理
        return 1;
        }
    }
   }
  }
```

上面程序基本上包含了上面实现的各功能函数，并加入了文件属性的判断，从而保证其文件目录项是用以描述“文件”的(文件目录项并非只能用于描述文件，还可以描述卷标、目录或是长名，不过这都是后话了)。

另外，如果文件目录项中的文件名字段的第一个字节为 0XE5，则说明所对应的文件已经被删除(多数情况下，文件的删除只是在文件名字段上标一个 0XE5 作为记号，其实文件仍然是存在的，只是普通用户看不到而已，这是数据恢复的一个重要依据。同时，这也是为什么很多公司对废弃的磁盘进行彻底物理销毁的原因)，因此我们在程序中加入了对这种情况的判断，以确保文件未被删除。

给这个函数起名为 Find_File_With_SFN，即“用短文件名找文件”，目标文件名通过其形参 filename 传入。

进一步来想想，上面的程序在找到了匹配的文件目录项之后，随后就是对匹配的文件目录项进行处理。如果把第 6 章实现的 Analyse_FDI 函数放在这里，那么上面的程序就不光是找文件了，还有了解析文件的功能。这个时候把 Find_File_With_SFN 改为 znFAT_Open_File，并添加一个形参 struct FileInfo * pfi。这样就引出了振南的 znFAT 中的打开文件函数。具体的代码(znFAT.c)如下：

```
UINT8 znFAT_Open_File(struct FileInfo * pfi,INT8 * filename)
{
 //变量定义
 //检查文件名合法性
 //遍历目录簇中的文件目录项
 //检查文件目录项属性
 if(File_Name_Match(filename,temp_filename))    //文件名匹配
    {
  Analyse_FDI(pfi,pitem);                       //解析文件目录项
     return 0;
    }
 return 1;
}
```

只需要给它一个文件名，最终就会将匹配的文件信息装入 pfi 指向的文件信息集合中。有没有看出其中所存在的问题呢？程序在遍历目录簇中的各个文件目录项时

仅局限于目录的首簇。我们来想，一个簇能记录的文件目录项的数目终归是有限的。如果一个簇有 8 个扇区，那么最多也只能记录 8×512/32＝128 个文件目录项，那么在这个目录下也最多只能有 128 个文件。这显然与平时看到的是不相符的：一个目录下能存入的文件远比这个数字要多得多，甚至是没有限制的。这到底是怎么回事？原因就在于我们忽略了一个非常重要的因素：目录也有簇链！如图 11.4 所示。

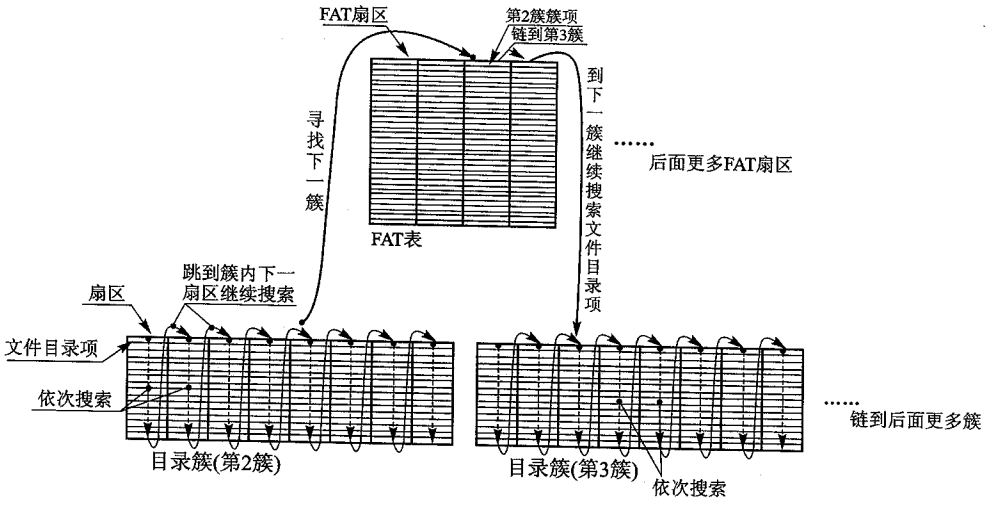

图 11.4　目录簇通过 FAT 簇链得到拓展

一开始介绍 FAT 表的时候，振南说过 FAT 簇链机制会渗透到 FAT32 的各个功能部分、各种文件操作中。此时，你有没有深切的感受呢？这正是 FAT32 在设计上的巧妙之处。同时，有没有感觉目录就是一种特殊的文件，它也有自己的首簇，也可以通过 FAT 簇链延伸到后续的更多簇。簇中存储了目录的数据，这些数据就是一个一个的文件目录项。

“目录是特殊的文件”，其实这是 FAT32 中的一个较为创新的概念，实现了目录与文件在存储机制上的统一化，尤其是根目录（FAT32 中称为首目录或第一个目录）。也许有人有过这样的经历：一个 U 盘或 SD 卡格式化为 FAT16（FAT32 的上一较早版本，现在仍在使用），如果在它的根目录下创建太多的文件或子目录，那么会提示无法创建，如图 11.5 所示。

为什么会这样？根本原因就在于 FAT16 的设计架构。它的根目录并不像 FAT32 那样可以通过 FAT 簇链进行拓展，而只是一块有着 32 个扇区的连续存储空间（最多存储 32×512/32＝512 个文件目录项，根目录下的文件或子目录总数不可能超过 512）。FAT16 文件系统的整体结构如图 11.6 所示。

但是，除了根目录以外，FAT16 其他目录中的文件和子目录的数目是没有限制的。显然，FAT16 没有把根目录和其他目录“一视同仁”，而是给了它一个单独的存储空间，没有纳入到数据区“簇”的范畴中来，自然也就不能使用簇链机制进行拓展。

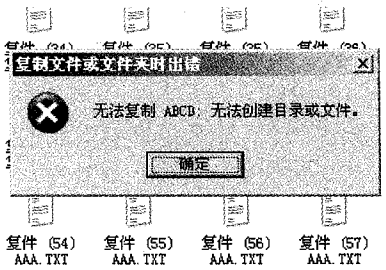

图 11.5　FAT16 文件系统不允许在根目录下创建过多文件或子目录

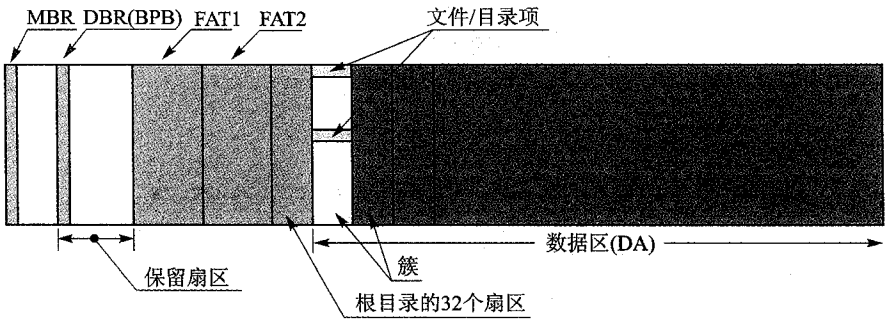

图 11.6　FAT16 文件系统的整体结构

而 FAT32 中取消了根目录的 32 个扇区，而直接使用簇来记录文件目录项，这样使得根目录与其它目录再无差别。因此，我们一直说："FAT32 没有根目录，而只有'首目录'!"(首目录是所有目录的魁首，是所有子目录和文件的最初入口)。

至于目录，我们在后面将会有专门的章节进行介绍，这里主要是为了告诉读者，在查找文件的时候，不光只限于目录首簇，还要拓展到后续簇。来看下面的代码(znFAT.c)：

```
……
do
{
 //文件目录项遍历搜索
 Cur_Clust = znFAT_GetNextCluter(Cur_Clust);    //获取下一簇
}while(! IS_END_CLU(Cur_Clust));                //如果不是最后一个簇，则继续循环
……
```

这个程序就将对文件目录项的搜索拓展到了后继簇上。最终，我们得到了 znFAT 中功能较为完善、逻辑较为严谨的文件打开函数(znFAT_Open_File)，它实际表现到底怎么样？有必要进行一个测试。

11.1.5　对文件打开函数的测试

对于功能函数的测试，振南一直认为要用一些极限而"变态"的测试方法，这样才