《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6章

创新功能，思维拓展：多元化功能特性与数据重定向的实现

到这里，所有的基本功能均已实现，包括文件的打开、数据的读取、文件与目录的创建、数据的写入等，而且我们还提出了一些原创性的核心机制和算法。现在，znFAT 的功能与运行效率已经不可小觑。不过，这些也只是一个文件系统方案本应包含的最基本的功能。实际的应用需求是多样而复杂的，这就要求我们要继续实现一些创新性、拓展性的功能。从某种意义上来说，这些功能或许更具有亮点，这不光表现在实现的技巧上，更多的是在设计思想上。你会发现，这些创新功能在代码实现上也许非常简单，但重点在于我们是否能想得到。在长期的应用与项目实践的过程中，在与广大使用者、爱好者的交流切磋中，振南总结了一些常用的扩展功能，如多文件、多设备、数据重定向等，本章就进行一一介绍。

6.1 多元化文件操作

多元化文件操作就是可以对一个存储设备或多个不同存储设备上的多个文件同时进行操作的属性或者功能。也许这样说有些抽象，来看看下面的具体内容。

6.1.1 多文件

关于多文件这一概念，其实前面就已经提过，并且还在很多实验中应用过。回忆一下，在上册的音乐数码相框实验中，我们是如何同时读取 MP3 文件与图片文件，最终完成音频播放与图像显示的？在后面的汉字电子书实验中，文本数据和汉字字模数据分别位于 TXT 文件与 HZK16 文件中，我们又是如何实现它们的同时读取的呢？像这种同时操作多个文件的应用其实已经多次出现在我们的实验中了。这种应用的实现就得益于 znFAT 对多文件的支持，如图 6.1 所示。

在 znFAT 的应用程序中可以定义多个文件信息体，它们相对独立地记录着各自对应文件的相关信息。调用文件操作函数时，我们向它传入谁的文件信息体，函数所操作的文件就是谁。所以，多个文件可以同时进行操作，而且互不影响。znFAT 中的多文件特性并不需要我们专门编程来实现，它是伴随着文件信息体的提出应运而

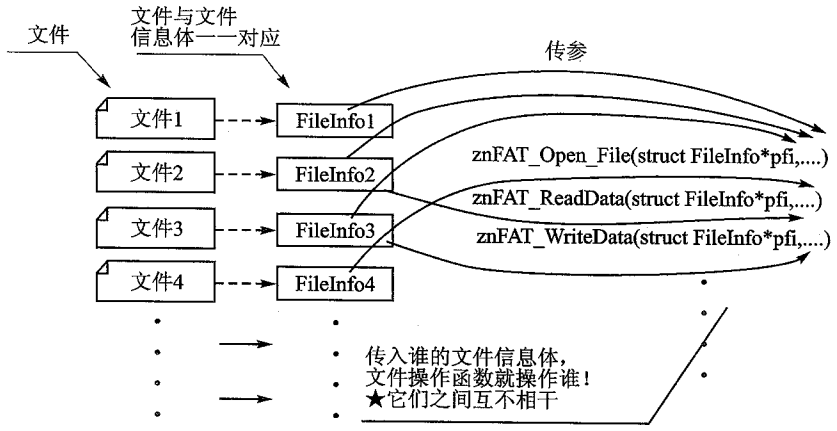

图 6.1　znFAT 中多文件操作示意图

生的。文件信息体(Struct FileInfo)对文件参数进行了封装,实现了对其集中化、独立化的管理,这一点是多文件的重要基础。

6.1.2　多设备

在实际的应用中,振南发现很多人都在用 znFAT 的多文件特性做一件事情——文件的复制。也就是把一个文件的数据全部或部分地转存到另一个文件中,这应该算是多文件的典型应用了。有些人提出了更深层的功能需求:"能不能在多个存储设备之间传输文件数据?比如将 SD 卡上的文件复制到另一张 SD 卡上。"如果要对多个存储设备上的文件同时进行操作的话,就必须让 znFAT 具有管理和调用多种存储设备扇区读写驱动的能力。能够让多套驱动程序并存,而且还要能够适时地、准确地在它们之间进行切换。要实现这一点,我们就要对 znFAT 的底层抽象驱动接口进行改进。代码如下(deviceio.c):

```
UINT8 Dev_No; //设备号
UINT8 znFAT_Device_Read_Sector(UINT32 addr,UINT8 * buffer)
{
 UINT8 res = 0;
 switch(Dev_No) //依设备号不同,调用相应驱动函数
 {
  case 0:
        res = Device0_Read_Sector(addr,buffer);
        break;
  case 1:
        res = Device1_Read_Sector(addr,buffer));
        break;
  case 2:
        res = Device2_Read_Sector(addr,buffer);
```

```
        break;
    case 3:
        res = Device3_Read_Sector(addr,buffer));
        break;
    ....
}
  return res;
}
//注:znFAT_Device_Write_Sector、znFAT_Device_Read_nSector、
    znFAT_Device_Write_nSector 与上面同理
```

这段程序中引入了一个变量 Dev_No(设备号)。同时将抽象驱动接口函数的实现由直接调用一个驱动改为使用 switch...case...分支结构来间接地、选择性地调用多个驱动，而选择的依据就是 Dev_No。如果把原来的抽象驱动接口函数比喻为“路标”，那此时它就变成了一个“罗盘”。此为何意？请看图 6.2。

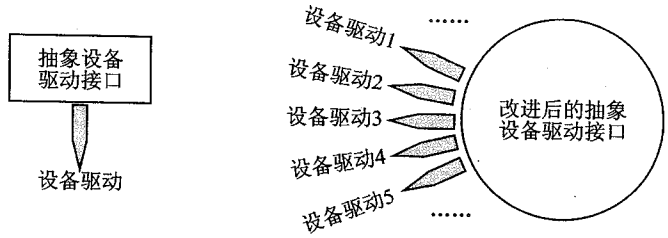

图 6.2 抽象设备驱动接口由路标变成了罗盘

适时更改 Dev_No 的值就可以在多种设备驱动之间随意切换，但是要实现多设备文件操作，光有这一点还不够。还记得文件系统初始化参数集合(Init_Args)吗？它是用于记录文件系统相关参数的载体，对一个存储设备上的文件进行操作的过程中使用到的所有重要参数均源自于此。现在，我们既然加入了多设备的支持，那文件系统初始化参数集合也必然要有多个来与各个存储设备对应。而且，还要保证在切换到某一设备时所使用的初始化参数集合必须是与之配套的，否则就可能产生极为严重的错误，请看图 6.3。

我们将 znFAT 中的结构体变量 Init_Args 改为指针 pInit_Args，以便指向不同的初始化参数集合，通过 pInit_Args→...的方式来间接地对其参数进行访问。所以设备的切换其实包含了两部分：设备号的更改(它决定了使用哪一套驱动)，将 pInit_Args 指向相应的初始参数集合。我们使用 znFAT_Select_Device 函数来完成这一操作，这个函数就是“拨动罗盘的金手指”，代码如下(znFAT.c)：

```
UINT8 znFAT_Select_Device(UINT8 devno,struct znFAT_Init_Args * pinitargs)
                                                    //对设备进行选择
```

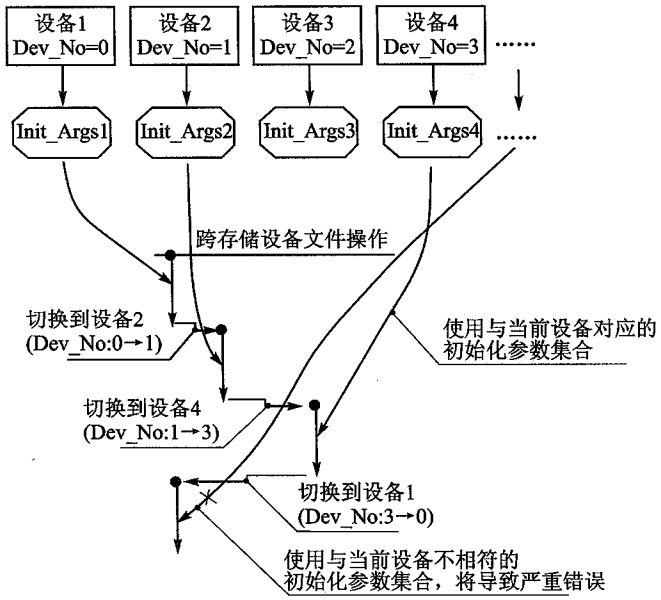

图 6.3　多设备之间的相互切换

```
{
 pInit_Args = pinitargs; //将初始化参数集合指针指向设备的初始化参数集合
 Dev_No = devno; //修改设备号
 return 0;
}
```

多设备看似还比较简单，但实际上会牵扯很多其他东西。比如 CCCB 与 EXB 的回写，就要考虑向哪个设备进行回写；再比如在更新文件的大小与 FSINFO 扇区时，也要考虑它们是属于哪个设备的……其实在做一个系统方案或者是较为庞杂的程序的过程中通常会发现：改动一点而牵扯全身，加入一个新东西，往往会波及很多部分。所以这就要求我们不光要有较高的编程水平，还要有统筹全局的设计思想。

“说了这么多，到底如何实现跨设备的文件复制呢?”别急，振南下面就用实例来进行说明：将一张 SD 卡上的文件复制到另一张 SD 卡上。(这里就要用到 ZN - X 开发板上的第二个 SD 卡模块接口，大家也就知道为什么 ZN - X 开发板上要有两个 SD 卡模块接口了。)实际硬件平台如图 6.4 所示。代码的具体实现如下(deviceio. c)：

```
UINT8 znFAT_Device_Read_Sector(UINT32 addr,UINT8 * buffer)
{
 switch(Dev_No) //依设备号调用相应的驱动函数
 {
  case 0:
       SD1_Read_Sector(addr,buffer); //SD 卡 1 的读扇区函数
       break;
```

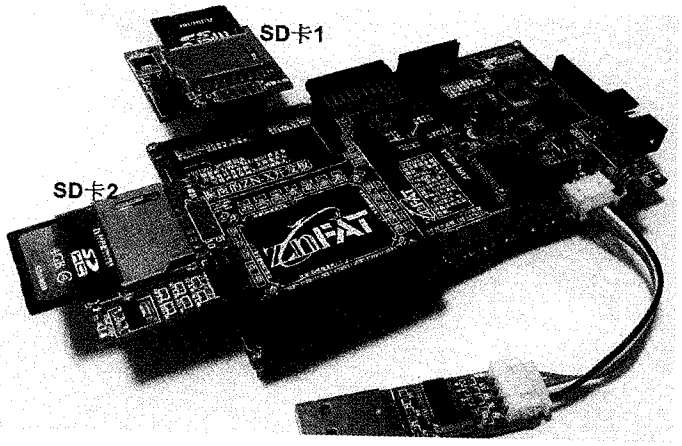

图 6.4　配备两个 SD 卡模块的 ZN－X 开发板

```
  case 1:
        SD2_Read_Sector(addr,buffer); //SD 卡 2 的读扇区函数
        break;
 }
 return 0;
}

//注:znFAT_Device_Write_Sector、znFAT_Device_Read_nSector、
    znFAT_Device_Write_nSector 与上面同理
```

_main.c 代码如下：

```
struct znFAT_Init_Args Init_Args1,Init_Args2; //初始化参数集合
struct FileInfo fileinfo1,fileinfo2; //文件信息集合
struct DateTime dt; //用于记录时间
unsigned char data_buf[1024]; //数据缓冲区
void main(void)
{
 int res = 0,len = 0;
 znFAT_Device_Init(); //存储设备初始化
 znFAT_Select_Device(0,&Init_Args1); //选择 SD 卡 1
 res = znFAT_Init(); //SD 卡 1 文件系统初始化
 if(!res) //文件系统初始化成功
 {
  //打印 SD 卡 1 文件系统相关参数
 }
 else //文件系统初始化失败
 {
  //打印错误信息
```

```
 while(1);
}
znFAT_Select_Device(1,&Init_Args2); //选择 SD 卡 2
res = znFAT_Init(); //SD 卡 2 文件系统初始化
if(!res) //文件系统初始化成功
{
 //打印 SD 卡 2 文件系统相关参数
}
else //文件系统初始化失败
{
 //打印错误信息
 while(1);
}
//以上代码用于完成两张 SD 卡的文件系统初始化
//将文件系统相关参数装入到 Init_Args1 与 Init_Args2 中
znFAT_Select_Device(0,&Init_Args1); //选择 SD 卡 1
res = znFAT_Open_File(&fileinfo1,"/source.txt",0,1); //打开源文件
if(!res) //如果打开文件成功
{
 //打印文件相关信息
}
else
{
 //打印错误信息
 while(1);
}
znFAT_Select_Device(1,&Init_Args2); //选择 SD 卡 2
//向 dt 装入时间信息
res = znFAT_Create_File(&fileinfo2,"/target.txt",&dt); //创建目标文件
if(!res) //创建文件成功
{
 //打印文件相关信息
}
else
{
 //打印错误信息
 while(1);
}
//以下开始进行数据的复制
while(1)
{
 znFAT_Select_Device(0,&Init_Args1); //选择 SD 卡 1
 len = znFAT_ReadData(&fileinfo1,fileinfo1.File_CurOffset,1024,data_buf);
                                          //读取 SD 卡 1 上的源文件数据
```

```
    if(len == 0) break; //如果数据已经读完，则跳出循环
    znFAT_Select_Device(1,&Init_Args2); //选择 SD 卡 2
    znFAT_WriteData(&fileinfo2,len,data_len); //将数据写入到 SD 卡 2 的目标文件中
  }
  znFAT_Select_Device(0,&Init_Args1); //选择 SD 卡 1
  znFAT_Close_File(&fileinfo1); //关闭文件
  znFAT_Select_Device(1,&Init_Args2); //选择 SD 卡 2
  znFAT_Close_File(&fileinfo2); //关闭文件
  znFAT_Flush_FS(); //刷新文件系统
  while(1);
}
```

从这段代码中可以看到，多设备的文件操作其实也很简单，与单一存储设备上的文件操作的不同点在于操作文件之前要先选择设备。

6.2 数据重定向

6.2.1 数据重定向的提出

数据重定向(Data Redirection)是 znFAT 中引入的又一个新概念，目的是减少内存的使用量。我们想想，数据读取函数从文件中读到的数据放到了哪里？对，应用数据缓冲区。随后，再按照数据的用途将其从应用缓冲区送到相应的地方去。比如上面的文件复制实验中，数据先被读到 data_buf 中，然后再进一步写入到目标文件中；再比如上册中的 MP3 数码相框实验中，数据也是先被读到一个用户缓冲区中，然后再依数据的不同(音频或者图像数据)分别写入 MP3 解码器和 TFT 液晶中。所以，这就告诉我们要想读数据，则必须要有足够的内存来进行中转。对于一些内存资源比较匮乏的平台，要开辟出一个足够大的应用数据缓冲区其实并不容易，甚至是不可能的。难道内存不够，就不能完成数据读取的操作了吗？也不尽然。我们是否可以绕过应用数据缓冲区，而直接将数据送至其应用之处呢？答案是肯定的，请看图 6.5。

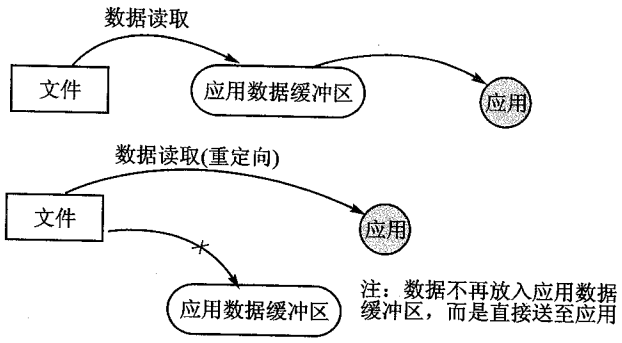

图 6.5 数据重定向功能示意图

6.2.2 数据重定向的实现

数据重定向是对数据读取函数的改造，它将数据装入应用数据缓冲区改为直接调用数据处理函数进行处理。这个数据处理函数其实就是图 6.5 中的“应用”，它由使用者自行定义及实现。为了便于与实际的数据处理函数相接驳，振南在 znFAT 中引入这样一个宏，代码如下：

```
#define Data_Redirect USER_FUNCTION //“数据重定向”中的“单字节”处理函数
```

Data_Redirect 函数用来对从文件中读取的每一个字节进行处理。比如我们从文件的某个扇区中读取了 512 字节的数据，按照常规的作法，实现代码是这样的(可以参见 znFAT_ReadData 函数的源代码)：

```
znFAT_Device_Read_Sector(pfi->File_CurSec,app_Buffer+have_read);
//读取文件当前扇区，将其数据拼入应用数据缓冲区
```

但如果是数据重定向，则会这样做：

```
znFAT_Device_Read_Sector(pfi->File_CurSec,znFAT_Buffer);
                                //将数据先读入 znFAT 的内部数据缓冲区
for(i=0;i<512;i++)
{
 Data_Redirect(znFAT_Buffer[i]);
                                //将数据直接使用单字节处理函数进行处理
}
```

因为不再使用外部的应用数据缓冲区，而是用内部数据缓冲区来充当临时性的数据载体，因此数据重定向是无法使用多扇区读/写驱动的，这就注定了它的数据操作速度和效率并不会太高。

Data_Redirect 的实体其实是用户函数 USER_FUNCTION，既然它是一个单字节处理函数，那么它在形式上必定是这样的：

```
void USER_FUNCTION(unsigned char byte)
{
 //字节数据处理程序
}
```

比如 #define Data_Redirect UART_Send_Byte，就可以实现将文件数据直接从串口输出。

带有数据重定向功能的数据读取函数，我们给它起名为 znFAT_ReadDataX，函数定义如下：

```
UINT32 znFAT_ReadDataX(struct FileInfo *pfi,UINT32 offset,UINT32 len)
```

它的形参与 znFAT_ReadData 的不同之处就在于少了一个指向应用数据缓冲区的指针。

其实，起初振南在实现数据重定向的时候字节处理函数并不是以宏的形式来实现的，而是使用形如 void (*pfun)(UINT8)的函数指针。它作为 znFAT_ReadDataX 函数的形参，通过指向实际的字节处理函数来间接地对其进行调用，示例如下(znFAT.c)：

```
UINT32 znFAT_ReadDataX(struct FileInfo *pfi,UINT32 offset,
                                        UINT32 len,void (*pfun)(UINT8))
{
 //其他实现代码
 znFAT_Device_Read_Sector(pfi->File_CurSec,znFAT_Buffer);
                          //将数据先读入 znFAT 的内部数据缓冲区
 for(i=0;i<512;i++)
 {
  *pfun(znFAT_Buffer[i]); //使用函数指针指向的函数对字节进行处理
 }
 //....
}
```

其实这种方式就是传说中的回调(CallBack)，pfun 指向的就是回调函数。不过，很多人反映对函数指针不太熟悉，在理解上有一些困难。所以，振南才将其改为了函数名宏定义的方式。

6.2.3 数据重定向实现 MP3 播放

有了数据重定向之后，SD 卡 MP3 播放器实验就又多了一种实现方式，比上册讲过的多步式实现方式更省内存(这里跨度有点大，请大家好好回忆一下)，示意如图 6.6所示。具体实现代码如下(config.h)：

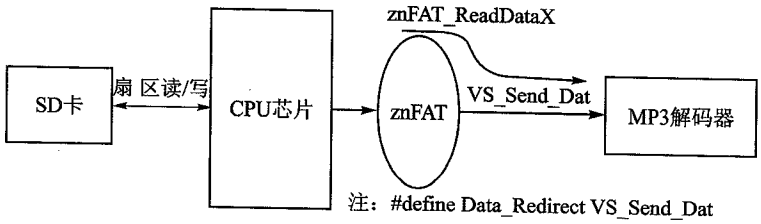

图 6.6 使用数据重定向实现 MP3 播放

```
#define Data_Redirect   VS_Send_Dat
```

_main.c 代码如下：

```
struct znFAT_Init_Args Init_Args;
```

```
struct FileInfo fileinfo;
void main(void)
{
 //相关器件初始化
 znFAT_Select_Device(0,&Init_Args);//选择设备
 znFAT_Init(); //文件系统初始化
 znFAT_Open_File(&fileinfo,"/test.mp3",0,1); //打开 MP3 文件
 SET_VS_XDCS(0); //使能 VS1003 芯片的数据片选
 znFAT_ReadDataX(&fileinfo,0,fileinfo.File_Size);
                          //读取文件全部数据直接送至 MP3 解码器
 SET_VS_XDCS(1);
 while(1);
}
```

在这个程序中 MP3 文件的数据被一次性全部读出，并直接送至 MP3 解码器，期间不再经过缓冲区中转，这就是数据重定向了。

好，上面就是本章的全部内容。我们介绍了 znFAT 的一些创新性的概念和功能，它们在实际开发过程中也确实得到了较为广泛的应用。这些内容是 znFAT 中的亮点，同样也是本书的精华。后面仍然有更多的精彩在等待着大家，敬请翻篇。

感谢对振南及 znFAT 的关注与支持，希望振南在嵌入式 FAT32 文件系统方面的研究对您有所帮助！

更多内容请关注 振南电子网站

www.znmcu.cn