《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 1 章

数据记录，偷梁换柱：使用变通方法实现文件数据存储

上册一直都是围绕读取操作来进行的，也就是说只用到了扇区读取函数（znFAT_Device_Read_Sector），并没有涉及与扇区写入相关的内容。其实 znFAT 早期的开发工作到这里就基本完成了，并没有实现写入操作相关的功能，仅仅是为了完成上册开篇所说的 MP3 数码相框实验而已，但是后来有很多人看到振南在研究 FAT32，于是经常问有没有实现文件创建和数据写入的功能，想通过 znFAT 来满足他们数据记录的需求，于是决定把 FAT32 继续做下去，最终形成一个完备的方案，实现文件操作的各种常用功能。

虽然当时 znFAT 中还没有成型的数据写入功能，但振南还是使用一些变通，甚至是“偷梁换柱”的方法帮助一些人巧妙地实现了数据记录的功能。振南觉得有必要将这些方法介绍给读者，如果可以满足需求，那也许就可以无须或者尽量少地去牵扯 FAT32 了，因为 FAT32 毕竟还是比较复杂的。当然，这些“变通方法”必然是有一定局限性的，只能针对于个别的、要求不高的应用场合。要想实现真正意义上的、通用的文件创建、数据写入等功能，那么还是要继续对 FAT32 进行研究，一步一步地去实现我们想要的功能。

1.1 把 SD 卡用作一个大容量的 ROM

1.1.1 大 ROM 思想的提出

2010 年，振南完成的一个项目的主要功能就是实现一个塔吊黑匣子，具体的描述请看图 1.1。放置于驾驶室中的一个装置，一方面接收从塔吊总控制器发送过来的吊钩位置与报警信息（通过 CAN 总线进行通信），另一方面把数据存入到 SD 卡中，并向总控制器发送回执。

起初，这个公司因为没有数据存储和文件系统方面的开发经验才找到了振南。项目经理说：“只要能将数据存入 SD 卡，并能方便地导入到计算机中就行了。”于是，振南向他介绍了“大 ROM”思想。其实很简单，就是把 SD 卡直接当成一个大容量的

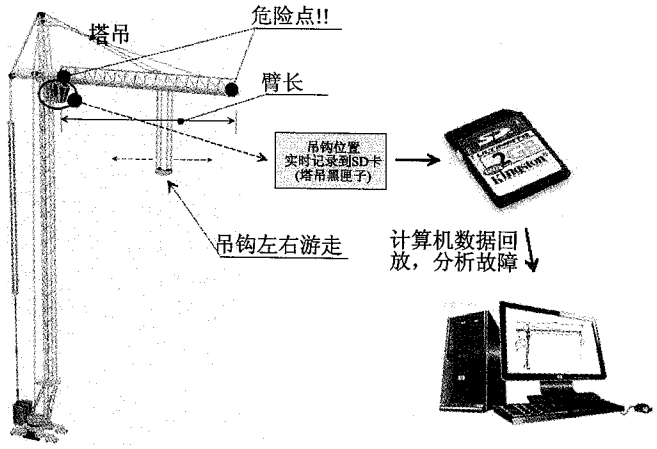

图 1.1　塔吊黑匣子项目功能示意图

ROM 存储器来使用，如图 1.2 所示。

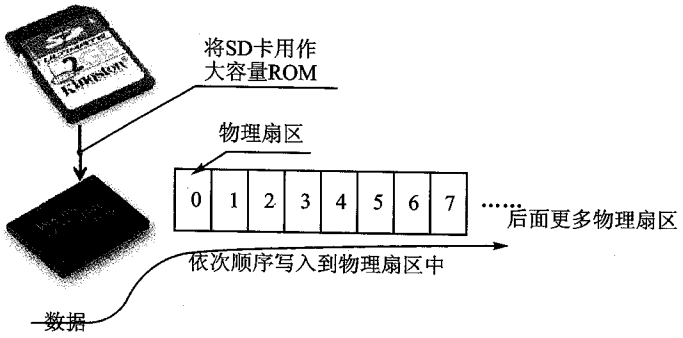

图 1.2　将数据直接顺序写入到物理扇区

此时，项目经理立即反驳："这样不行，SD 卡放到计算机上识别不出来的。"确实是！根本原因就是因为我们写入数据的方式并没有遵循 FAT32 的标准。数据非但不能识别，而且还会把现有的文件系统毁掉，计算机会提示我们"格式化磁盘"（原本用于存储 MBR 或 DBR 的物理扇区被数据覆盖，从而导致整个文件系统的崩溃）。"解铃还须系铃人"，若用直接写物理扇区的方式来进行数据存储，那在计算机上也要以物理扇区方式来获取数据。还记得 WinHex 软件的物理模式吗？

1.1.2　思想的验证：数据采集与记录实验

项目经理迷惑地问道："似乎可行，你能不能先在开发板上实现一下，做一个演示？"于是就有了数据采集与记录实验，如图 1.3 所示。这个实验基于 ZN－X 开发板（配合基础实验资源模块），如图 1.4 所示。

从 TLC549、DS18B20、PCF8563 分别读取模拟信号、温度数据和时间信息，将它

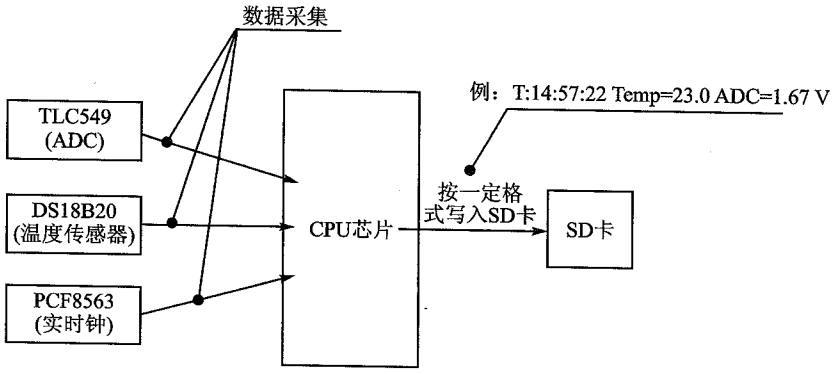

图 1.3　数据采集与记录实验示意图

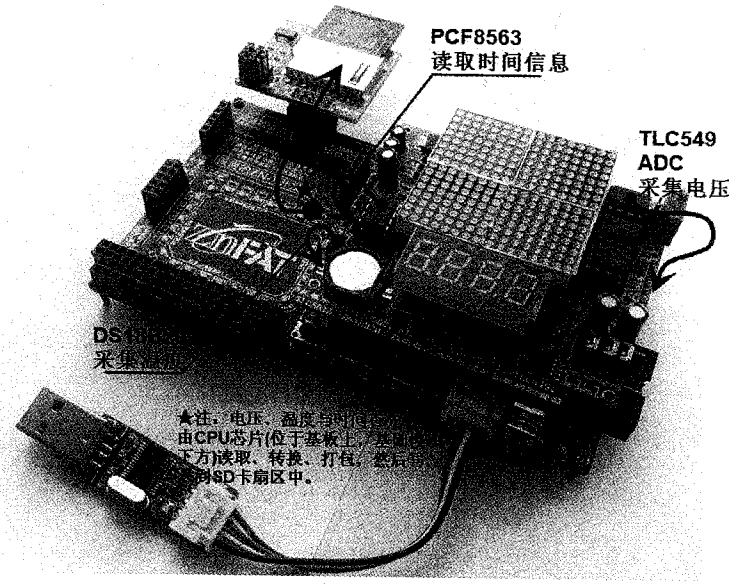

图 1.4　ZN－X 开发板配合基础实验资源模块效果图

们打包为固定 32 字节的数据格式，形如“T:14:57:22 Temp=23.0 ADC=1.67V\r\n”，我们称之为一个数据帧。每采集到 16 个数据帧，即数据量达到 512 字节，就将其写入到物理扇区。具体实现代码(_main.c)如下：

```
unsigned char buf[512];
struct Time time; //用于装载时间数据的结构体型变量 time
void main(void)
{
 unsigned int counter = 0;
 unsigned long addr = 0;
 SD_Init(); //SD 卡初始化
```

```
while(1)
{
 P8563_Read_Time(); //读取时间信息
 Format_Dat(&time,DS18B20_ReadTemperature(),TLC549_GetValue(),buf + counter);
 counter + = 32;
 if(counter == 512) //数据量达到 512 字节
 {
  SD_Write_Sector(addr ++ ,buf); //将数据写入到 SD 卡物理扇区
  counter = 0;
 }
}
 while(1);
}
```

程序运行后可以使用 WinHex 软件打开 SD 卡的物理扇区查看数据，如图 1.5 所示。可以看到，包含了时间信息、温度与电压的数据帧整齐地排列在扇区中。接下来可以使用 WinHex 软件中的“克隆磁盘”功能，将扇区数据转存为文件，具体方法如图 1.6 所示。

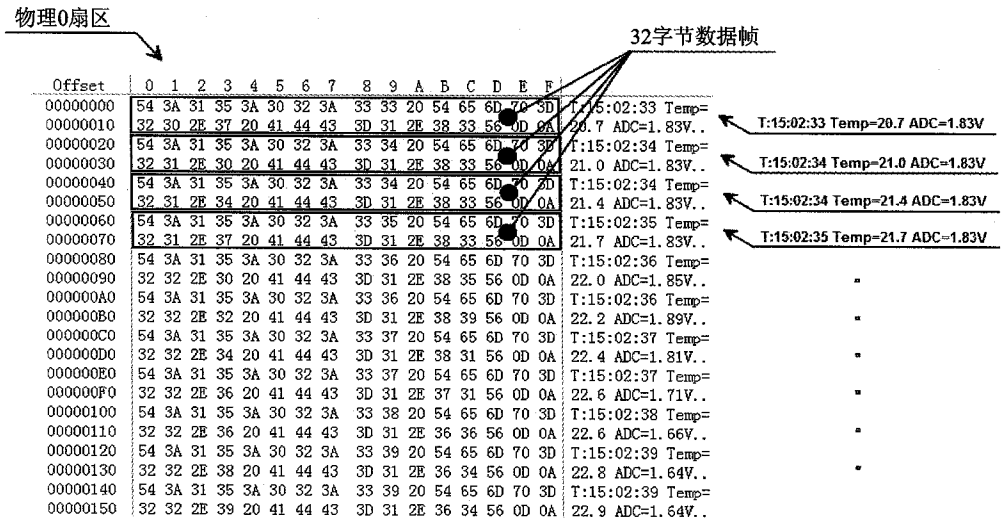

图 1.5　物理 0 扇区中的数据帧

其实，克隆磁盘功能可以将从某一扇区开始的任意多个扇区中的数据转存为文件，甚至是整个磁盘。转存得到的文件有一个专门的名字来称呼它——“镜像”。

上面的实验其实是存在一个小问题的：“数据向扇区顺序写入，那怎么知道最后它停在哪个扇区上呢？应该用 WinHex 复制多少个扇区呢？”其实很简单，我们从 1 扇区开始写数据，停止的时候将当前扇区地址写入到 0 扇区中，也就是说，留出 0 扇区专门用于记录结束扇区。

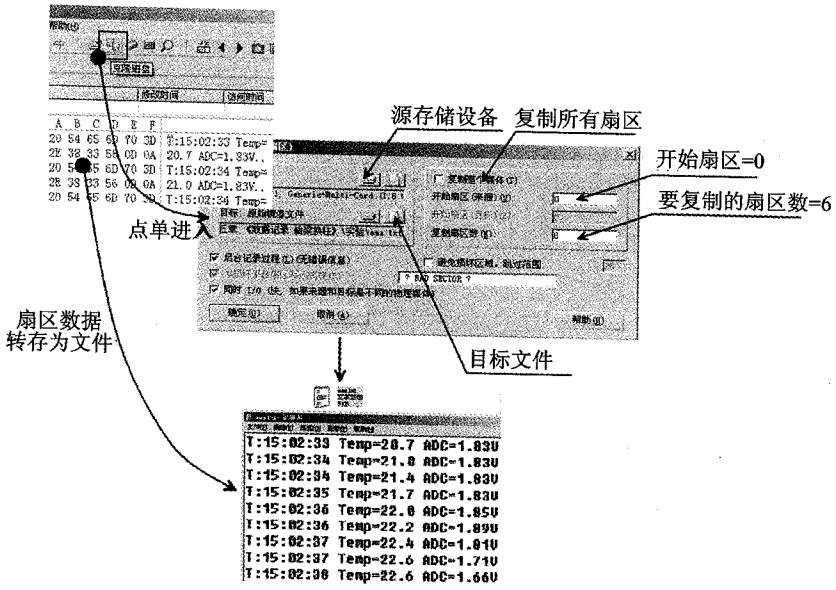

图 1.6　使用克隆磁盘功能将扇区数据转存为文件

可以看到，这种方法简单、实用、易于实现，在原理上基本不涉及文件系统的任何相关内容，但不足之处也很明显，它必须要借助于 WinHex 软件。那么有没有其他方法可以不依靠任何附加软件而实现数据的导入呢？答案是：有！

1.2　数据“偷梁换柱”——数据替换

下面要介绍的才是真正在塔吊黑匣子项目中使用的方法，它的基本思想就是“移花接木，偷梁换柱”。

1. 思想的提出

如果在一张刚刚被格式化的 SD 卡上创建一个文件，并向其写入数据，那么数据的存储将会是连续的，也就是说数据是分布在连续扇区上的。基于这种特点，我们提出了“偷梁换柱”的思想，其实质是数据替换，请看图 1.7。

首先使用计算机在 SD 卡上创建一个文件，并向其写入大量的数据（这些数据的具体内容无关紧要）。这个大文件其实就为我们构造了一个现成的 FAT32 文件框架，或者说是一个“数据池”。如果将新数据从这个文件的开始扇区依次向后写入（新写的数据将会覆盖原来的数据），当再一次打开这个文件的时候，那么将看到文件的内容已经变成写入的新数据了。既然是文件，自然是可以直接复制到计算机中，不再依靠任何其他软件。

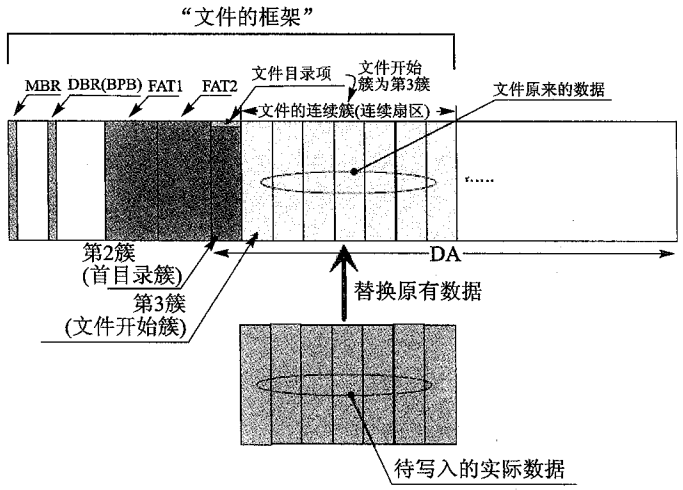

图 1.7　数据“偷梁换柱”思想的示意图

2. 思想的验证

上面的想法似乎是非常巧妙，但是到底是否可行呢？我们还需要通过实验来进行验证。要想实现它，则必须完成两件事情：大文件的创建与文件开始扇区的定位。

(1) 大文件的创建

针对于大文件的创建，振南专门写了一个小软件，名为 mbf(make big file)。使用之前首先要安装，如图 1.8 所示。然后在 Windows 命令行中输出如图 1.9 所示命令，回车。随即，在 SD 卡的首目录中出现一个大小约为 190 MB 的大文件，其内容均为 0XFF，如图 1.10 所示。

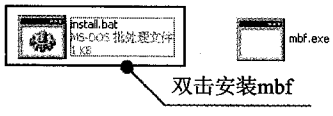

图 1.8　安装大文件制作软件 mbf

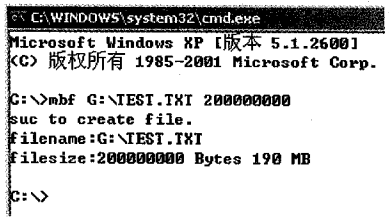

图 1.9　使用 mbf 软件创建大文件的具体方法

(2) 文件开始扇区的定位

文件的数据从哪个扇区开始，则可以使用前面实现的 znFAT_Open_File 函数的文件信息集合中的 File_CurSec 获知。但是其实有比这更简单的方法。我们可以想想，如果要向一张刚刚被格式化的、全空的 SD 卡中写入文件，那文件的文件目录项必定会被放入到第 2 簇中(首目录簇)，而文件的数据则会从第 3 簇开始存放，所以只需要计算出第 3 簇的开始扇区即可，代码如下：

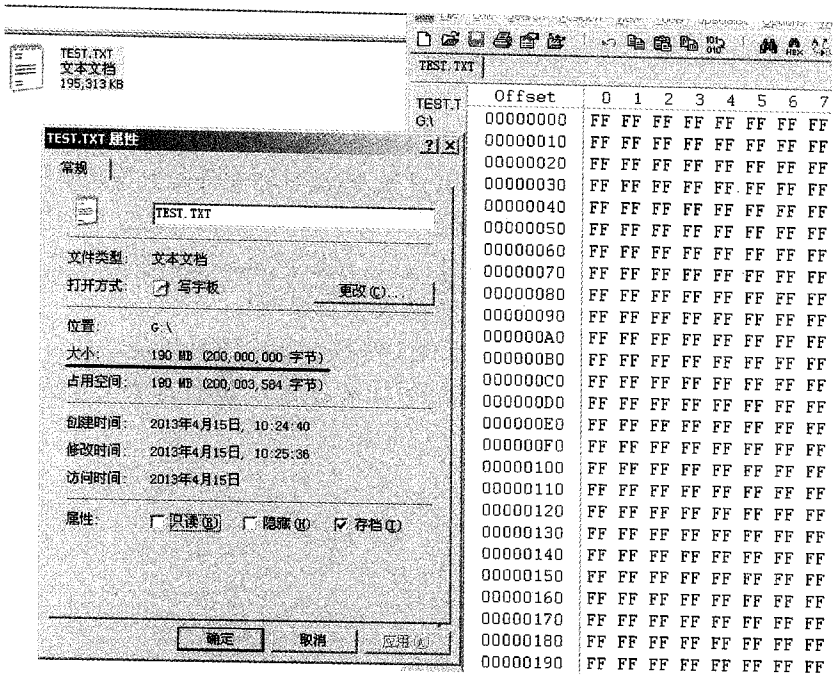

图 1.10　SD 卡首目录下的大文件 TEST.TXT

```
znFAT_Init(); //文件系统初始化,获取核心参数
StartClust = SOC(3); //计算第 3 簇开始扇区,则文件开始扇区
```

这样，最终的实现代码中只会涉及简单的文件系统初始化操作，而不会牵扯其他与 FAT32 文件系统相关的内容。

至于“偷梁换柱”思想的具体实现其实与前面大 ROM 思想的实现过程基本一样，唯一不同的就是 addr 的初值是文件的开始扇区，而不再是 0 了，代码(_main.c)如下：

```
void main(void)
{
 unsigned int counter = 0;
 unsigned long addr = 0;
 SD_Init(); //SD 卡初始化
 znFAT_Init(); //文件系统初始化
 addr = SOC(3); //计算文件开始扇区
 while(1)
 {
  //采集数据,打包成固定格式
  //将数据写入物理扇区
  addr ++ ;
```

```
 }
 while(1);
}
```

此时遇到了跟前面一样的问题，即数据最终停止于哪里？有读者可能会说："这很好解决，把整个文件全部复制到计算机里来，然后看看文件的数据从哪里开始出现大量的 0XFF，哪里就是实际写入的数据的结尾。"没错，但是未免效率太低。我们希望文件复制到计算机后其中存储的就直接是有效数据，后面那些无用的、冗长的 0XFF 能够自动去掉。这如何实现呢？修改文件目录项！

上册在讲解文件目录项的时候曾经对文件目录项中的时间信息进行过修改，当时我们称为"文件信息的篡改"。现在要对文件大小进行修改，请看如下代码(_main.c)：

```
void main(void)
{
 unsigned int counter = 0;
 unsigned long addr = 0,temp = 0;
 SD_Init(); //SD卡初始化
 znFAT_Init(); //文件系统初始化
 addr = SOC(3); //计算文件开始扇区
 while(1)
 {
  //采集数据，打包成固定格式
  //将数据写入物理扇区
  if(KEY) //按键停止数据采集与记录
  {
   temp = (addr - SOC(3) + 1) * 512; //计算写入的数据量
   SD_Read_Sector(Init_Args.FirstDirSector,znFAT_Buffer); //读取首目录簇开始扇区
   //修改文件目录项中的文件大小字段(文件目录项在扇区中的位置相对固定)
   znFAT_Buffer[60] = temp;
   znFAT_Buffer[61] = (temp>>8);
   znFAT_Buffer[62] = (temp>>16);
   znFAT_Buffer[63] = (temp>>24);
   SD_Write_Sector(Init_Args.FirstDirSector,znFAT_Buffer); //回写首目录簇开始扇区
   break;
  }
  addr++;
 }
 while(1);
}
```

有人又会有疑问："光修改文件的大小能行吗？那 FAT 簇链还是那么长啊！是

不是还要把 FAT 簇链根据实际数据量从中间掐断?"他所描述的问题具体如图 1.11 所示。答案是，FAT 簇链无须截断，保留现状即可。在 Windows 中，文件的复制是依据文件大小描述的数据量来进行的。

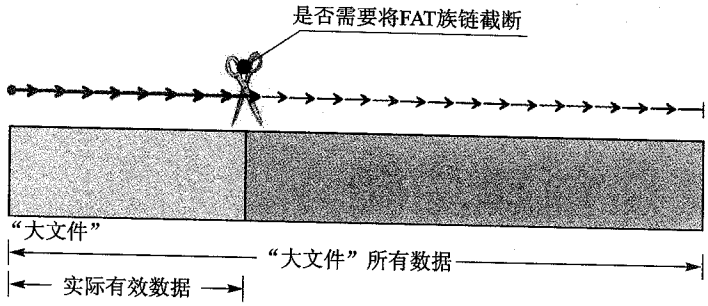

图 1.11　数据文件复制出来时是否需要截断 FAT 簇链

好，到这里本章就告一段落了。希望本章介绍的方法和思想，能够对读者有所启发；哪怕对文件系统不熟，也照样能把文件数据存储功能用起来。但是这里所讲的内容毕竟不是真正意义上的文件操作，而是"另辟蹊径"，下一章就开始对真正的文件创建、数据写入等功能的实现进行研究，回到 FAT32 的"正路"上来。

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