UM1721
用户手册

在 STM32Cube 上开发 FatFs 相关应用

前言

STMCube™ 计划源自意法半导体，旨在通过减少开发的工作量、时间与成本，使开发者受益。 STM32Cube 涵盖 STM32 产品组合。

STM32Cube 1.x 版包括：

- 图形软件配置工具 STM32CubeMX，可通过图形向导生成 IAR/KEIL 工程。
- 针对每个系列提供综合的嵌入式软件平台 （即 STM32CubeF4 用于 STM32F4 系列）
  - STM32 抽象层嵌入式软件 STM32Cube HAL，确保在 STM32 各个产品之间实现最大限度的可移植性
  - 一套一致的中间件，比如 RTOS、 USB、 TCP/IP、图形
  - 所有嵌入式软件实用工具均配备一套完整的示例。

文件系统是指文件命名的方式，以及文件的逻辑存储位置，以便进行保存和检索。它的主要目标是管理文件数据的访问，以及管理包含这些文件的设备的可用空间。文件系统可帮助用户确保可靠性并以高效的方式来组织数据。

本用户手册的目标读者为在 STM32 微控制器上使用 STM32Cube 固件的开发者。它完整描述了如何使用具有通用 FAT 文件系统 （FatFs）的 STM32Cube 固件组件；本用户手册还提供了一组示例说，它们基于通用 FatFs 所提供的 API。

关于所使用的 FatFs 固件组件版本，请参考 STM32Cube 固件包的发布文档。

本文档适用于所有 STM32 器件；然而为了简洁起见，以 STM32F4xx 器件和 STM32CubeF4 作为参考平台。如需了解更多关于 STM32 器件所支持的物理介质盘和实现示例，请参考相关 STM32Cube 固件包中所提供的 readme 文件。

目录

表格索引

图片索引

缩略语与定义

表 1. 缩略语与定义

缩略语	定义
ANSI	美国国家标准协会
API	应用编程接口
BPB	BIOS 参数块
BSP	板级支持包
CPU	中央处理器
CMSIS	Cortex™ 微控制器软件接口标准
DBCS	双字节字符串
DOS	磁盘操作系统
EFI	可扩展固件接口
FAT	文件分配表
HAL	硬件抽象层
LFN	长文件名
MBR	主引导记录
MSD	微型安全数字
OEM	原始设备制造商
RAM	随机存取存储器
RTC	实时时钟
RTOS	实时操作系统
SD	安全数字
SDRAM	同步动态随机存取存储器
SFN	短文件名
SRAM	静态随机存取存储器
USB	通用串行总线

1 FAT 文件系统概述

1.1 FAT 概述

文件分配表 （FAT）文件系统是由比尔盖茨与麦克唐纳所开发。它是一种格式，某种程度上也算是软件，它用于在存储设备 （比如磁盘驱动或内存）上保存和组织文件。它用于方便文件与目录的访问。

FAT 文件系统提供一种途径来记录文件被创建或更改时的时间标记，并且提供了识别文件大小的方法。这套系统提供了保存文件其它属性的一套机制，比如文件是否只读，是否应在目录显示中隐藏，或者是否应在下一次磁盘备份中归档。

FAT 文件系统特别适合消费电子产品中的移动闪存介质，比如数码相机、媒体播放器和闪存盘等。

FAT 文件系统可以在以下场合中带来帮助：

- 由于 FAT 文件系统具备向后兼容性，用户可以利用记忆棒或软盘在消费电子设备和采用过时操作系统的计算机之间传输文件。
- FAT 文件系统让用户能够快速删除电子设备上的文件，就像在专业广播媒介中那样。
- FAT16 或 FAT32 的文件系统版本均适用于硬盘卷。

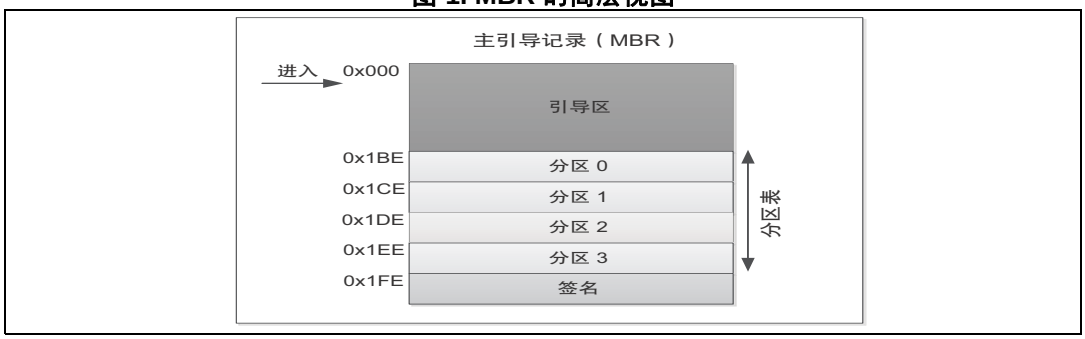
另外，如果用户想要通过软盘访问硬盘卷上的数据 （往往指系统恢复工具）来引导计算机的话，这些版本也很有用处。

1.1.1 主引导记录

主引导记录 （MBR）位于设备物理起始位置上的一个或多个扇区。 MBR 的引导区包含 DOS 引导加载程序代码，该代码会在设备格式化后被写入 （否则不会被动态 C FAT 文件系统所使用）。引导区之后是分区表。分区表中含有四个 16 字节的条目，允许设备划分多达四个分区。

分区表条目中含有一些关键信息：分区类型 （动态 C FAT 可识别 FAT12 和 FAT16 的分区类型）以及分区的起始与结束扇区号。另外还有一个字段指明分区中的扇区总数。如果该数字为零，则对应的分区是空的可用分区。

图 1. MBR 的高层视图

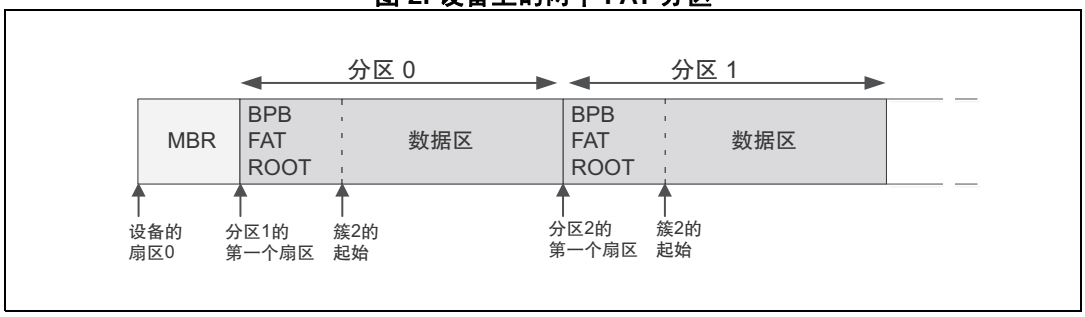


某些设备经格式化后没写入 MBR，因此便也没有分区表。目前在动态 C FAT 文件系统中并不支持这种配置。

1.1.2 FAT 分区

有效 FAT 文件系统分区的第一个扇区中包含 BIOS 参数块（BPB），之后是文件分配表（FAT），再之后是根目录。下图显示了具备两个 FAT 分区的设备。

图 2. 设备上的两个 FAT 分区



BIOS 参数块

BPB 中的字段包含该分区的描述信息：

- 每个扇区的字节数；
- 每个簇的扇区数；
- 该分区的扇区总数；
- 根目录下的条目数。

FAT 分配表

文件分配表是一种结构，也是 FAT 文件系统的命名由来。 FAT 中保存了关于簇的分配信息。一个簇既可以分配给一个文件，也可以使用，还可以标记为坏簇。 FAT 分配表的副本会紧跟着第一个 FAT 分配表存放。

根目录

根目录具有预定义的位置及大小。根目录有 512 个条目，每个条目 32 字节。根目录中的条目既可以是空条目，也可以包含文件或子目录名称（以 8.3 格式）、文件大小、上一次修改的日期时间以及文件或子目录的起始簇号。

数据区

数据区占据了分区中的大部分空间。其中包含文件数据与子目录。请注意，按照惯例，分区的数据区必须从第 2 簇开始。

欲了解更多信息，请参考 Microsoft® EFI FAT32 文件系统规范。

1.1.3 FAT 许可

Microsoft 可扩展固件计划 FAT32 文件系统规范，1.03 版， 2000 年 12 月 6 日 , Office Word 文档格式 （268 KB）。

下载许可协议仅允许结合可扩展固件计划规范 （v1.0）中的固件实现来使用 Microsoft EFI FAT32 文件系统规范。1.0. 如果计划将 FAT32 文件系统规范用于其它用途，必须从 Microsoft 获取额外的许可。

比如，必须获取额外的许可，才能创建用于读取的文件系统；在数码相机中读取和写入FAT32闪存介质；在计算机操作系统中读取和写入内部/外部硬盘或闪存介质；又或者读取机顶盒中FA 格式的介质。

欲了解更多关于 FAT 和适用许可证和 / 或版权的信息，请参考 Microsoft 网站。

2 FatFs 文件系统

2.1 FatFs 概述

FatFs 是适用于小型嵌入式系统的 FAT 文件系统模块。FatFs 是按照 ANSI C 的标准来指定，且与磁盘 I/O 层完全分隔开。因此， FatFs 与硬件架构完全无关，具有以下特点：

- 兼容 Windows 的 FAT 文件系统。
- 极小的代码量和工作区
- 丰富的配置选项：
  - 多卷 （物理驱动与分区）。
  - 多个 ANSI/OEM 代码页，包括 DBCS。
  - 以 ANSI/OEM 或 Unicode 支持长文件名。
  - 支持 RTOS。
  - 支持多种扇区大小。
  - 只读、最小化的 API、 I/O 缓冲等等 .....
  - FAT 子类型：FAT12、 FAT16 和 FAT32。
  - 打开的文件数量：无限制，取决于可用的内存。
  - 卷的数量：多达 10 个。
  - 文件大小：取决于 FAT 规范。（多达 4G-1 字节）
  - 卷的大小：取决于 FAT 规范。（512 字节 / 扇区情况下，支持多达 2T 字节）
  - 簇的大小：取决于 FAT 规范。（512 字节 / 扇区情况下，支持多达 64K 字节）
  - 扇区的大小：取决于 FAT 规范。（多达 4K 字节）

2.2 FatFs 架构

FatFs 模块是一个中间件，提供许多用于访问 FAT 卷的函数，比如 f_open()、 f_close()、f_read()、 f_write() 等等 （参考 ff.c）。

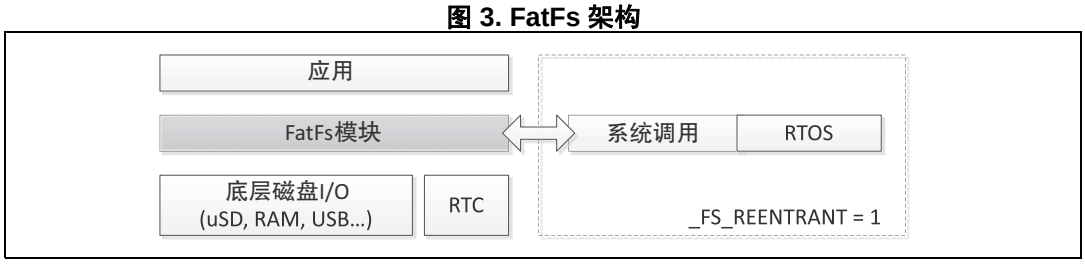
该模块没有平台依赖性，只要编译器符合 ANSI C 即可。

使用底层磁盘 I/O 模块来读取 / 写入物理驱动。

采用 RTC 模块来获取当前时间。

底层磁盘 I/O 和 RTC 模块均与 FatFs 模块完全分离。它们必须由用户提供，这是将 FatFs 模块与其它平台相连的主要工作。

图 3. FatFs 架构



2.3 FatFs 许可

图 4 是包含在源代码中的 FatFs 许可文档的副本。

图 4. FatFs 许可

```
/*----------------------------------------------------------------------------/
/  FatFs - FAT file system module  R0.10                      (C)ChaN, 2013
/-----------------------------------------------------------------------------/
/ FatFs module is a generic FAT file system module for small embedded systems.
/ This is a free software that opened for education, research and commercial
/ developments under license policy of following trems.
/
/  Copyright (C) 2013, ChaN, all right reserved.
/
/ * The FatFs module is a free software and there is NO WARRANTY.
/ * No restriction on use. You can use, modify and redistribute it for
/   personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
/ * Redistributions of source code must retain the above copyright notice.
/
----------------------------------------------------------------------------/
```

因此，FatFs 许可是其中一种 BSD 型许可，但存在显著差异。因为 FatFs 适用于嵌入式项目，为提高其可用性，未规定其二进制形式的重新分发条件，如嵌入式代码、十六进制文件以及二进制库等。分发文档不必包含关于 FatFs 及其许可的文档，但也可包含在内。当然，FatFs 与 GNU GPL 下的项目兼容。进行任意修改再重新分发时，该许可也可以改为 GNU GPL 或 BSD 型的许可。

2.4 FatFs 特性

2.4.1 复制文件访问

FatFs 模块默认并不支持对复制文件的共享控制。仅当文件的打开方式为只读模式时，才允许共享控制。禁止以写入模式对文件进行复制打开，而且打开的文件不允许被重命名和删除，否则会破坏卷的 FAT 结构。

当将 _FS_LOCK 设为 1 或更大的数值时，也可使用文件共享控制。 该数值设定了同时管理的文件数目。在这种情况下，如果尝试 “ 打开 ”、“ 重命名 ” 或 “ 删除 ” 等违反上述文件共享规则的操作，则文件功能失败，提示 FR_LOCKED。如果打开的文件数量大于 _FS_LOCK，那么 f_open() 函数也会执行失败，提示 FR_TOO_MANY_OPEN_FILES。

2.4.2 可重入性

对不同卷的文件操作是可重入的，并且可以同时工作。对同一个卷的文件操作是不可重入的，但可利用 _FS_REENTRANT option 选项配置为线程安全。在这种情况下，必须将依赖于操作系统的同步对象控制函数ff_cre_syncobj()、ff_del_syncobj()、ff_req_grant()和ff_rel_grant()添加到项目中。

当卷被其它任务使用时，如果文件函数被调用，则这个文件函数会保持挂起，直至该任务结束。如果等待时间超过 _TIMEOUT 所设定的时间，那么文件函数会以 FR_TIMEOUT 退出。某些 RTOS 并不支持这种超时功能。

f_mount() 和 f_mkfs() 函数是例外。这些函数对于同一个卷并不具备可重入性。当使用这些函数时，所有其它任务必须关闭该卷上的对应文件，以避免对卷的访问。

请注意，这一节描述的是 FatFs 模块本身的可重入性，但底层磁盘 I/O 层也必须是可重入的。

2.4.3 长文件名

FatFs 模块从版本 0.07 开始便支持长文件名 （LFN）。除了 f_readdir() 函数以外，一个文件的 SFN 和 LFN 这两种不同的文件名对于文件函数来说是透明的。如需使能 LFN 功能，请将 _USE_LFN 设为 1、2 或 3，然后在项目中添加 Unicode 代码转换函数 ff_convert() 和 ff_wtoupper()。LFN 功能需要附加一个特定的工作缓冲。缓冲的大小可根据可用内存大小，利用 _MAX_LFN 来进行配置。长文件名可达 255 个字符，所以 _MAX_LFN 应设为 255，以实现完全的 LFN 功能。如果工作缓冲的大小不足以存放给定的文件名，则文件函数执行失败并提示 FR_INVALID_NAME。当以可重入功能使能 LFN 特性时，_USE_LFN 必须设为 2 或 3。这种情况下，文件函数会在栈或堆里分配工作缓冲。工作缓冲会占据 (_MAX_LFN + 1) * 2 个字节。

当使能 LFN 功能时，模块大小会根据所选择的代码页而有所增大。右表显示了当以某些代码页使能 LFN 功能时，所增加的字节数。

2.5 FatFs API

FatFs 的 API 层用于执行文件系统 API。它采用磁盘 I/O 接口与适当的物理驱动通信。这些 API 可划分为四组：

- 操作逻辑卷或分区的 API 分组。
- 操作目录的 API 分组。
- 操作文件的 API 分组。
- 操作文件和目录的 API 分组。

以下列出了 FatFs 访问 FAT 卷时所能执行的操作：

- f_mount()：挂载 / 卸载逻辑磁盘
- f_open()：打开 / 创建文件
- f_close()：关闭文件
- f_read()：读取文件
- f_write()：写入文件
- f_lseek()：移动读 / 写指针，扩大文件的大小
- f_truncate()：截取文件到当前已读 / 已写指针位置
- f_sync()：同步内存中的数据到磁盘
- f_opendir()：打开一个目录
- f_readdir()：读取目录条目
- f_getfree()：获取空闲的簇的数量
- f_stat()：检查对象是否存在，并获取其状态
- f_mkdir()：创建目录
- f_unlink()：删除文件或目录

- f_chmod()：更改属性
- f_utime()：更改时间戳
- f_rename()：重命名 / 移动文件或目录
- f_chdir()：更改当前目录
- f_chdrive()：更改当前驱动
- f_getcwd()：检索当前目录
- f_getlabel()：获取卷标签
- f_setlabel()：设置卷标签
- f_forward()：将文件数据直接转发至流
- f_mkfs()：在驱动上创建文件系统
- f_fdisk()：划分物理驱动
- f_gets()：读取字符串
- f_putc()：写入字符
- f_puts()：写入字符串
- f_printf()：写入格式化的字符串
- f_tell()：获取当前的读 / 写指针
- f_eof()：检验是否达到文件末尾
- f_size()：获取文件大小
- f_error()：检验文件是否出错

2.6 FatFs 底层 API

由于 FatFs 模块与磁盘 I/O 和 RTC 模块完全分离，所以需要一些底层功能来操作物理驱动：读 / 写和获取当前时间。因为底层磁盘 I/O 功能和 RTC 模块并非 FatFs 模块的组成部分，所以必须由用户提供。

FatFs 中间件解决方案为某些支持的磁盘驱动 （RAMDisk、 uSD、 USBDisk）提供底层磁盘 I/O 驱动。

已添加额外接口层 diskio.c，用于为 FatFs 模块动态添加 / 删除 （链接）物理介质，提供如下所述的底层磁盘 I/O 函数：

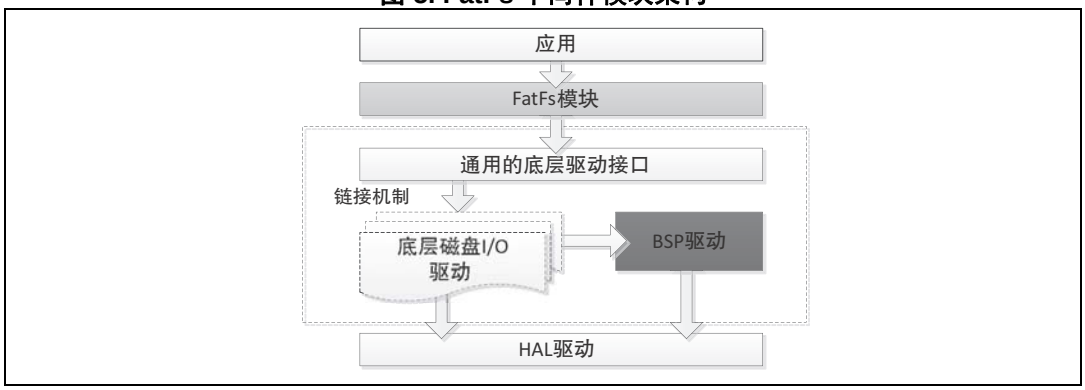
- disk_initialize()：初始化物理磁盘驱动
- disk_status()：返回所选物理驱动的状态
- disk_read()：读取磁盘中的扇区
- disk_write()：将扇区写入磁盘
- disk_ioctl()：控制设备的专用功能
- get_fattime()：返回当前时间

应用程序禁止调用这些函数，这些函数仅可由 FatFs 文件系统函数调用，比如 f_mount()、f_read()、 f_write()...

2.7 将 FatFs 整合至 STM32CubeF4

在 STM32CubeF4 解决方案中，已添加额外的接口层，用于动态地添加 / 删除 FatFs 模块的物理介质。如需以底层磁盘 I/O 驱动来连接 FatFs 模块，用户可以使用 FATFS_LinkDriver() 和 FATFS_UnLinkDriver() 动态地添加或者删除磁盘 I/O 驱动；应用程序可能需要知道当前连接的磁盘 I/O 驱动数量，这一点可通过 FATFS_GetAttachedDriversNbr() API 来实现。

图 5. FatFs 中间件模块架构



通用的底层驱动 ff_gen_drv.c/h 位于 FatFs 模块的根目录下。采用两个磁盘 I/O 驱动类型定义结构，协助动态管理 ff_gen_drv.h 文件下所连接的磁盘驱动，如下所述：

表 2. “Diskio_drv_TypeDef” 结构

字段	说明
`DSTATUS (*disk_initialize)(void);`	初始化磁盘驱动
`DSTATUS (*disk_status)(void);`	获取磁盘状态
`DRESULT (*disk_read)(BYTE*, DWORD, BYTE);`	读取扇区
`DRESULT (*disk_write)(const BYTE*, DWORD, BYTE);`	写入扇区 _USE_WRITE 应设为 0
`DRESULT (*disk_ioctl)(BYTE, void*);`	I/O 控制操作 _USE_IOCTL 应设为 1

表 3. “Disk_drv_TypeDef” 结构

字段	说明
`Diskio_drvTypeDef *drv[_VOLUMES];`	Diskio_drv_TypeDef 结构
`uint8_t nbr;`	所连接的驱动数量

如需将 FatFs 模块与底层磁盘 I/O 驱动相连接，用户可以使用以下 API：

- FATFS_LinkDriver()：用于动态添加磁盘 I/O 驱动。
- FATFS_UnLinkDriver()：用于动态删除磁盘 I/O 驱动。
- FATFS_GetAttachedDriversNbr()：了解当前所连接的磁盘 I/O 驱动数量

2.7.1 FATFS_LinkDriver()

该函数用于连接兼容的磁盘 I/O 驱动，并增加已激活连接的驱动的数量。如果成功则返回 0，失败则返回 1。

注： *因为 FatFs 方面的限制，所连接的磁盘最大数目（_VOLUMES）为 10 个。*

FATFS_LinkDriver 的实现：

```
uint8_t FATFS_LinkDriver(Diskio_drvTypeDef *drv, char *path)
{
  uint8_t ret = 1;
  uint8_t DiskNum = 0;
  if(disk.nbr <= _VOLUMES)
  {
    disk.drv[disk.nbr] = drv;
    DiskNum = disk.nbr++;
    path[0] = DiskNum + '0';
    path[1] = ':';
    path[2] = '/';
    path[3] = 0;
    ret = 0;
  }
  return ret;
}
```

2.7.2 FATFS_UnlinkDriver()

该函数用于解除与磁盘 I/O 驱动的连接，并减少已激活连接的驱动数量。如果成功则返回 0，失败则返回 1。

FATFS_UnLinkDriver 的实现：

```
uint8_t FATFS_UnLinkDriver(char *path)
{
  uint8_t DiskNum = 0;
  uint8_t ret = 1;

  if(disk.nbr >= 1)
  {
    DiskNum = path[0] - '0';
    if(DiskNum <= disk.nbr)0
    {
```

```
      disk.drv[disk.nbr--] = 0;
      ret = 0;
    }
  }

  return ret;
}
```

2.7.3 FATFS_GetAttachedDriverNbr()

该函数返回已连接至 FatFs 模块的驱动数量。

FATFS_GetAttachedDriversNbr 的实现：

```
uint8_t FATFS_GetAttachedDriversNbr(void)
{
  return disk.nbr;
}
```

2.8 将自己的磁盘连接至 FatFs

如果工作存储控制模块可用，应通过粘合函数连接至 FatFs，而不是进行修改。用户可以开发适当的磁盘 I/O 底层驱动来连接任意的新磁盘（mynewdisk_diskio.c/.h），并将这些驱动文件保存在：\Middlewares\Third_Party\FatFs\src\drivers。

值得注意的是，所提供的 FatFs 磁盘 I/O 底层驱动依赖于板级 BSP 驱动。如需移除这种 BSP 依赖性，用户可以将 “BSP_...” 的 API 调用替换为自己的代码，以确保实现正确的功能性。

如需从头开始开发磁盘 I/O 底层驱动，用户可以从下述的粘合函数骨架开始，用已定义的 API 将现有的存储控制模块添加到 FatFs。

适用于 FatFs 的底层磁盘 I/O 模块骨架：

```
/*-----------------------------------------------------------------------*/
/* mynewdisk_diskio.c：适用于 FAT 的底层磁盘 I/O 模块框架 */
/*-----------------------------------------------------------------------*/

/* 包含的头文件 ---------------------------------------------------------*/
#include <string.h>
#include "ff_gen_drv.h"
/* 私有的宏定义 ---------------------------------------------------------*/
#define BLOCK_SIZE 512 /* Block Size in Bytes */

/* 私有变量 ------------------------------------- */
static volatile DSTATUS Stat = STA_NOINIT; /* 磁盘状态 */

/* 私有函数原型 ---------------------------------------------*/
DSTATUS mynewdisk_initialize (void);
```

```
DSTATUS mynewdisk_status (void);
DRESULT mynewdisk_read (BYTE*, DWORD, BYTE);
#if _USE_WRITE == 1
  DRESULT mynewdisk_write (const BYTE*, DWORD, BYTE);
#endif /* _USE_WRITE == 1 */
#if _USE_IOCTL == 1
  DRESULT mynewdisk_ioctl (BYTE, void*);
#endif  /* _USE_IOCTL == 1 */

Diskio_drvTypeDef  mynewdisk_Driver =
{
  mynewdisk_initialize,
  mynewdisk_status,
  mynewdisk_read,
#if  _USE_WRITE == 1
  mynewdisk_write,
#endif /* _USE_WRITE == 1 */

/*----------------------- 初始化驱动 ----------------------------*/
DSTATUS mynewdisk_initialize (void)
{
  Stat = STA_NOINIT;

  // 在此写入自己的代码，对驱动进行初始化

  Stat &= ~STA_NOINIT;
  return Stat;
}

/*------------------------- 获取磁盘状态 ------------------------------*/
DSTATUS mynewdisk_status (void)
{
  Stat = STA_NOINIT;

  // 在此写入自己的代码

  return Stat;
}

/*-------------------------- 读取扇区 -----------------------------*/
DRESULT mynewdisk_read (BYTE *buff, /* 保存读取数据的数据缓冲 */
                  DWORD sector, /* 扇区地址（LBA） */
                  BYTE count) /* 读取的扇区数（1..128）*/
{
  DRESULT res = RES_ERROR;

  // 在此写入自己的代码，读取驱动中的扇区

  return res;
}

/*--------------------------- 写入扇区 ----------------------------*/
#if _USE_WRITE == 1
DRESULT mynewdisk_write (const BYTE *buff, /* 需写入的数据缓存指针 */
```

```
                    DWORD sector, /* 扇区地址 （LBA） */
                    BYTE count) /* 需写入的扇区数 （1..128） */
{
  DRESULT res = RES_ERROR;

  // 在此写入自己的代码，向驱动写入扇区

  return res;
}
#endif /* _USE_WRITE == 1 */

/*----------------------- 其它函数 ----------------------*/
#if _USE_IOCTL == 1
DRESULT mynewdisk_ioctl (BYTE cmd, /* 控制代码 */
                    void *buff) /* 用于发送 / 接收控制数据的代码 */
{
  DRESULT res = RES_ERROR;

  // 在此写入自己的代码，控制驱动专用功能
  // CTRL_SYNC、 GET_SECTOR_SIZE、 GET_SECTOR_COUNT、 GET_BLOCK_SIZE
  // 如控制同步，获取扇区大小，获取扇区数量，获取块大小等

  return res;
}
#endif /* _USE_IOCTL == 1 */
```

底层磁盘 I/O 模块的头文件：

```
/*-------------------------------------------------------------------------*/
/* mynewdisk_diskio.h：底层磁盘 I/O 模块的头文件                  */
/*-------------------------------------------------------------------------*/

/* 避免递归包含的定义 ---------------------------------*/
#ifndef __MYNEWDISK_DISKIO_H
#define __MYNEWDISK_DISKIO_H

extern Diskio_drvTypeDef  mynewdisk_Driver;

#endif /* __MYNEWDISK_DISKIO_H */
```

3 FatFs 应用程序

STM32CubeF4 解决方案中提供了许多基于 FatFs 中间件的应用程序。下表显示了如何在不同例子中使用 FatFs 中间件组件，这些例子按照复杂度分类，并取决于所使用的物理驱动接口 （uSD、 RAMDisk、 USBDisk）：

表 4. FatFs 中间件使用示例

示例类别	示例	说明
起始	单一逻辑单元 （RAMDisk）上的 FatFs	将 FatFs 模块连接至 RAM 上的空磁盘 I/O 驱动，然后通过静态缓冲进行安装、打开、写入、读取和关闭操作。
特性	单一逻辑单元上的 FatFs	将 FatFs 模块连接至 uSD 磁盘 I/O 驱动，然后通过静态缓冲进行安装、打开、写入、读取和关闭操作。
	多个逻辑单元上的 FatFs	将 FatFs 模块连接至 uSD 和 RAM 磁盘 I/O 驱动，然后通过静态缓冲进行安装、打开、写入、读取和关闭操作。
集成	单一逻辑单元 （USB 盘）上的 FatFs	将 FatFs 模块连接至 USB 主机磁盘 I/O 驱动，然后通过静态缓冲进行安装、打开、写入、读取和关闭操作。

STM32CubeF4 解决方案所提供的上述 FatFs 应用程序是一套有两种模式的固件：

- 独立模式
- RTOS 模式，使用 *FreeRTOS* 中间件组件

值得注意的是，当使用或者开发基于意法半导体磁盘 I/O 底层驱动的 FatFs 应用程序时，用户必须保证合适的堆栈值。

因此，当使用基于 USB 主机大容量存储类的 U 盘时，由于对齐的原因，栈的值必须根据最大扇区大小值 _MAX_SS 来增加。

当在 RTOS 模式中开发任何 FatFs 应用程序时，也必须使用基于 CMSIS-OS 包覆层通用 API 的 FreeRTOS 中间件组件对堆数值进行调整。

3.1 HAL 驱动配置

STM32CubeF4 解决方案中所提供的 FatFs 应用程序是一套用于与各种物理磁盘驱动（uSD、RAM 盘、 USB 盘）相连接的固件。用户需要某些运行 FatFs 应用程序所必需的 HAL 驱动。相应的 HAL 驱动可通过 HAL 配置文件 *stm32f4xx_hal_conf.h* 来使能，解除 HAL 驱动中所使用的适当模块的注释即可。

HAL 配置文件中，各个支持的磁盘驱动之间的主要差异在于与所使用的磁盘驱动相对应的正确 HAL 驱动的定义。根据各种驱动，以下宏定义必须可用：

- FatFs_uSD:
  - #define HAL_SD_MODULE_ENABLED
- FatFs_RAMDisk:
  - #define HAL_SDRAM_MODULE_ENABLED 或
  - #define HAL_SRAM_MODULE_ENABLED
- FatFs_USBDisk:
  - #define HAL_HCD_MODULE_ENABLED

3.2 FatFs 文件系统配置

FatFs 模块中包含各种配置选项。在这一层级，我们提供信息帮助用户根据所连接的物理磁盘驱动选择正确的选件，满足用户需求，达到最高性能。

3.2.1 可重入性

可重入性是独立和 RTOS 模式配置之间的主要差异，这一点可以在 FatFs 配置文件 *ffconf.h* 中进行设置。

- 独立模式中禁用可重入性：
  - #define _FS_REENTRANT 0
- RTOS 模式中使能可重入性：
  - #define _FS_REENTRANT 1

一旦使能后，用户必须提供依赖于 OS 的同步对象 （#define _SYNC_t osSemaphoreId）

RTOS 模式应用程序的项目需要包含 *syscall.c* 文件，以提供 OS 依赖函数，可在以下路径找到：*\Middlewares\Third_Party\FatFs\src\option*

3.2.2 长文件名

FatFs 模块支持长文件名 （LFN）以及 8.3 格式文件名 （SFN）。

请注意，FAT 文件系统上的 LFN 功能是微软公司的专利。虽然在 FAT32 上不是这样，但大多数 FAT32 驱动包含 LFN 功能。 FatFs 可以通过配置选项切换 LFN 功能。当在商业产品上使能 LFN 功能时，需要根据最终目标获取微软的许可。当使能 LFN 功能时，可以使用 LFN，这项功能可以在 FatFs 配置文件 *ffconf.h* 中设置：FatFs 配置文件 *ffconf.h* 中 (_USE_LFN > 0)

- 禁用 LFN 功能：
  - #define _USE_LFN 0
- 使能 LFN 功能，其中 3 ⩾ _USE_LFN > 0：

一旦在 *ffconf.h* 配置文件中使能之后，应用程序项目需要包含 *syscall.c/unicode.c* 文件，以提供内存管理功能，该文件可在以下路径中找到：*\Middlewares\Third_Party\FatFs\src\option*

用户在独立模式应用或 RTOS 模式应用中均可使能 LFN 功能。

3.3 FatFs 示例应用程序

如果用户已经连接了自己的磁盘，开发了适当的磁盘 I/O 底层驱动 (mynewdisk_diskio.c/.h)，请参考*第 2.8 章节：将自己的磁盘连接至 FatFs*，可按照以下方式将其驱动连接至 FatFs 模块以及使用逻辑磁盘：

```
/*-----------------------------------------------------------------------*/
/* main.c：主程序 */
/*-----------------------------------------------------------------------*/

/* 包括 ------------------------------------------------------------*/
#include "main.h"

/* 私有变量 --------------------------------------- */
FATFS mynewdiskFatFs;  /* 用户逻辑驱动的文件系统对象 */
FIL MyFile;            /* 文件对象 */
char mynewdiskPath[4]; /* 用户逻辑驱动路径 */

int main(void)
{
  uint32_t wbytes; /* 写入文件的字节计数 */
  uint8_t wtext[] = "text to write logical disk"; /* 写入文件的缓冲 */

if(FATFS_LinkDriver(&mynewdisk_Driver, mynewdiskPath) == 0)
  {
    if(f_mount(&mynewdiskFatFs, (TCHAR const*)mynewdiskPath, 0) == FR_OK)
    {
      if(f_open(&MyFile, "STM32.TXT", FA_CREATE_ALWAYS | FA_WRITE) == FR_OK)
      {
        if(f_write(&MyFile, wtext, sizeof(wtext), (void *)&wbytes) == FR_OK);
        {
          f_close(&MyFile);
        }
      }
    }
  }
  FATFS_UnLinkDriver(mynewdiskPath);
}
```

用户必须包含通用驱动头文件 *ff_gen_drv.h* 以及磁盘 I/O 模块头文件 *mynewdisk_diskio.h*

```
/*-----------------------------------------------------------------------*/
/* main.h：main.c 模块的头文件                                           */
/*-----------------------------------------------------------------------*/

/* 包括 ------------------------------------------------------------*/
#include "ff_gen_drv.h"
#include "mynewdisk_diskio.h"
```

4 结论

本用户手册解释了如何将 FatFs 中间件组件整合到 STM32Cube HAL 驱动当中。

为帮助用户开发基于 STM32Cube 解决方案中 FatFs 文件系统的应用程序，本手册描述了一组示例。

5 FAQ

如何通过 FatFs 使用 LFN 功能？

FatFs 模块支持长文件名 （LFN）。欲了解更多关于如何通过 FatFs 使用 LFN 功能的信息，请参考*第 2.4.3 章节：长文件名*和*第 3.2.2 章节：长文件名*。

FatFs 多分区和多驱动应用之间有什么区别？

多分区应用可以使用绑定到指定物理驱动分区上的多个逻辑驱动，而多驱动应用可以同时使用不同的逻辑驱动（uSD、RAMDisk）。用户可以通过FatFs配置文件*ffconf.h*中的_VOLUMES定义来选择所需使用的逻辑驱动 （卷）数量。

用户可将任意的新磁盘连接至 FatFs 吗？

可以，用户可以将新的磁盘连接至 FatFs。有关详细信息，请参见*第 2.8 章节：将自己的磁盘连接至 FatFs*。

FatFs 支持多实例吗？

不支持。 STM32CubeF4 解决方案为 HAL 驱动提供多实例功能，但至于 FatFs 中间件组件，无法支持物理驱动的多实例。换句话说，用户无法运行使用多个逻辑驱动实例的应用程序。

FatFs 支持哪些 FAT 子类型？

FatFs 支持微软 FAT 子类型的全部三种主要变型：FAT12、 FAT16 和 FAT32。根据微软所发布的 FAT 规范，FAT 子类型仅仅由卷中的簇数量所决定。因此，选择哪一种 FAT 子类型取决于卷的大小以及所指定的簇大小。

6 修订历史

表 5. 文档修订历史

日期	修订	变更
2014 年 3 月 4 日	1	初始版本。
2014 年 6 月 23 日	2	封面更新： – 文件标题 – 参照 STM32Cube 中的 STM32CubeF4

请仔细阅读：

中文翻译仅为方便阅读之目的。该翻译也许不是对本文档最新版本的翻译，如有任何不同，以最新版本的英文原版文档为准。

本文档中信息的提供仅与 ST 产品有关。意法半导体公司及其子公司 （“ST”）保留随时对本文档及本文所述产品与服务进行变更、更正、修改或改进的权利，恕不另行通知。

所有 ST 产品均根据 ST 的销售条款出售。

买方自行负责对本文所述 ST 产品和服务的选择和使用，ST 概不承担与选择或使用本文所述 ST 产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为 ST 授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在 ST 的销售条款中另有说明，否则，ST 对 ST 产品的使用和 / 或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途 （及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

意法半导体的产品不得应用于武器。此外，意法半导体产品也不是为下列用途而设计并不得应用于下列用途：（A）对安全性有特别要求的应用，例如，生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）汽车应用或汽车环境，且 / 或 （D）航天应用或航天环境。如果意法半导体产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向意法半导体发出了书面通知，采购商仍将独自承担因此而导致的任何风险，意法半导体的产品设计规格明确指定的汽车、汽车安全或医疗工业领域专用产品除外。根据相关政府主管部门的规定，ESCC、QML 或 JAN 正式认证产品适用于航天应用。

经销的 ST 产品如有不同于本文档中提出的声明和 / 或技术特点的规定，将立即导致 ST 针对本文所述 ST 产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大 ST 的任何责任。

ST 和 ST 徽标是 ST 在各个国家或地区的商标或注册商标。

本文档中的信息取代之前提供的所有信息。

ST 徽标是意法半导体公司的注册商标。其他所有名称是其各自所有者的财产。

意法半导体集团公司

澳大利亚 - 比利时 - 巴西 - 加拿大 - 中国 - 捷克共和国 - 芬兰 - 法国 - 德国 - 中国香港 - 印度 - 以色列 - 意大利 - 日本 - 马来西亚 - 马耳他 - 摩洛哥 - 菲律宾 - 新加坡 - 西班牙 - 瑞典 - 瑞士 - 英国 - 美国

www.st.com