STM32Cube高效开发教程（高级篇）

第14章 用FatFS管理SD卡文件系统

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（高级篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2022年2月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc. edu. cn

14.1 SD卡文件系统概述

14.2 阻塞式访问SD卡示例

14.1 SD卡文件系统概述

- SD卡容量一般是16GB、32GB或更大，在实际使用中一般是在SD卡上建立文件系统，用FatFS管理SD卡上的文件
- 如果将SD卡看做一个User-defined存储介质，可以用第12章介绍的方法进行FatFS的移植
- 在使用CubeMX进行SD卡的FatFS文件系统管理开发时，无需自己实现FatFS底层Disk IO函数的移植，CubeMX生成的代码中已经针对SD卡做好了FatFS的移植
- SD卡数据读写可以采用轮询方式或DMA方式。在FreeRTOS中使用FatFS管理SD卡的文件系统时，SDIO只能使用DMA方式

14.2 阻塞式访问SD卡示例

14.2.1 示例功能与CubeMX项目设置

14.2.2 项目文件组成和初始代码分析

14.2.3 SD卡的Disk IO函数实现

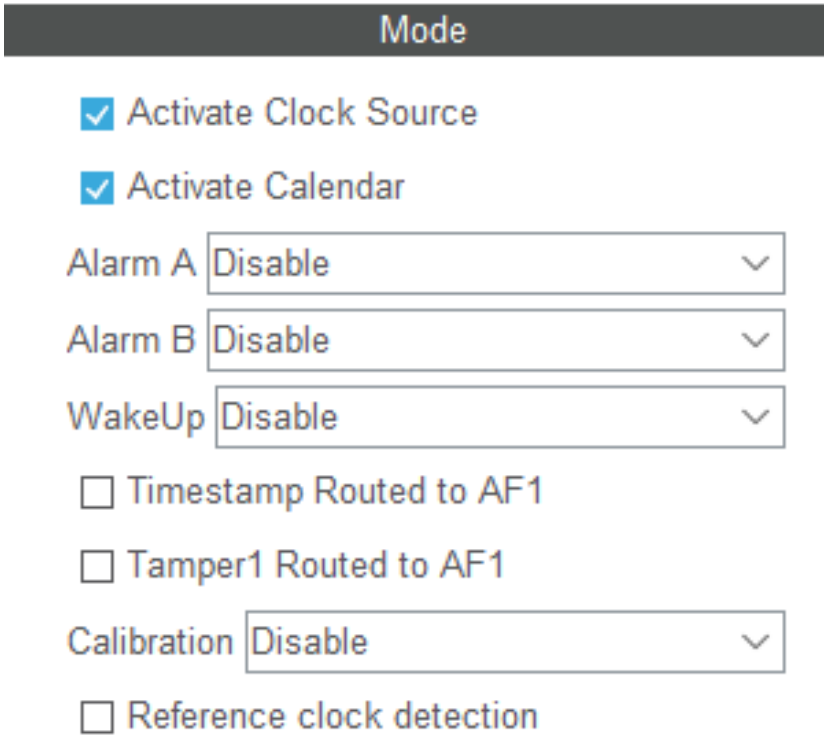
14.2.4 SD卡文件管理功能的实现

14.2.1 示例功能与CubeMX项目设置

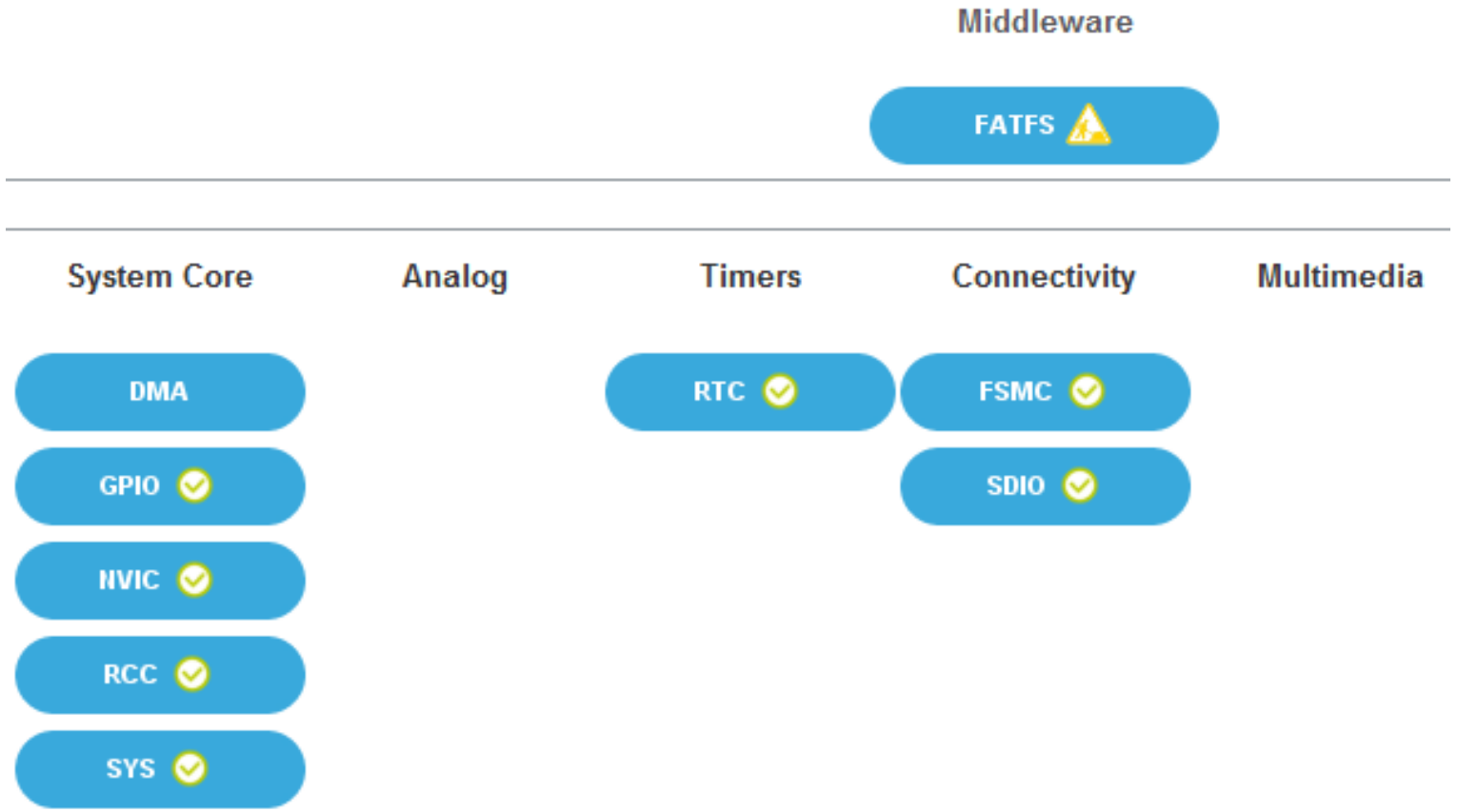
示例Demo14_1SD_FAT使用FatFS在SD卡上创建文件系统，并测试文件读写功能。

RTC设置

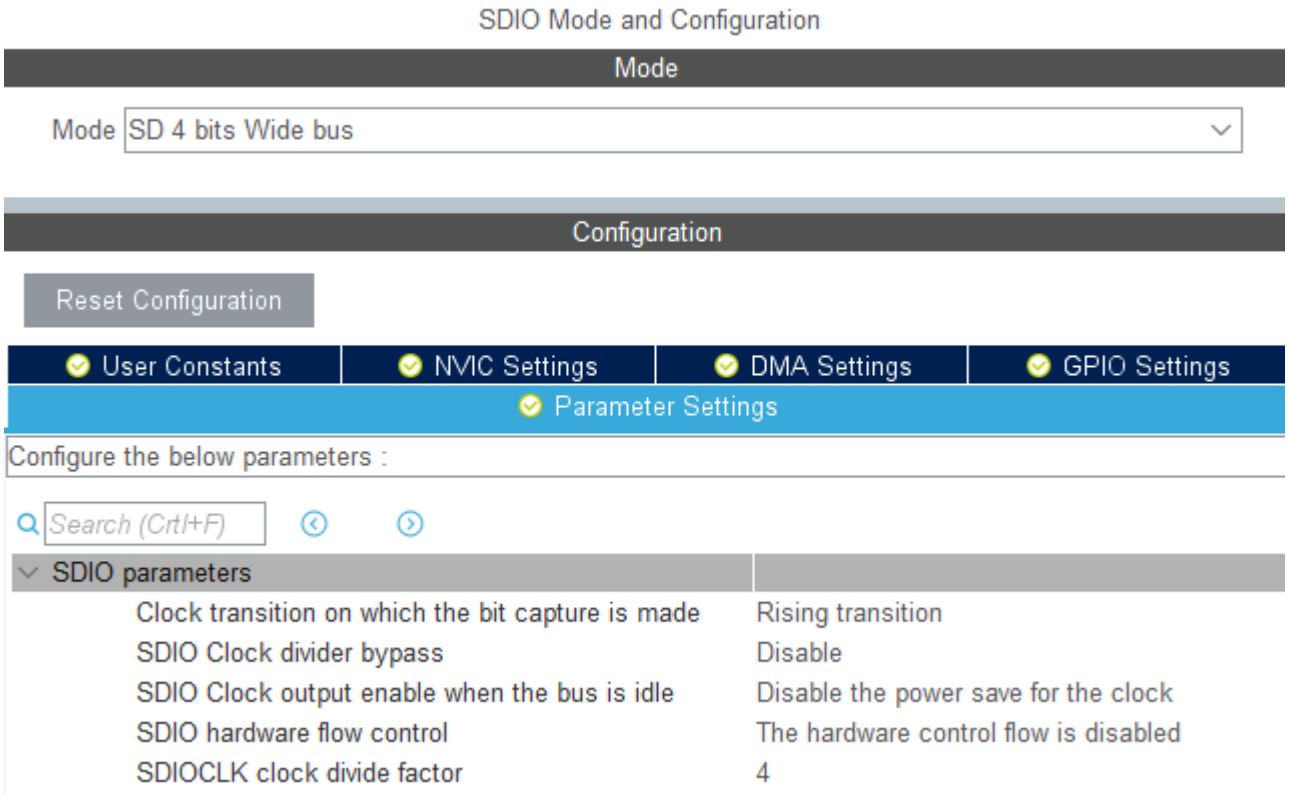
使用RTC为FatFS提供时间戳。在RTC的模式设置中，只需启用时钟源和日历，示例中只是要读取RTC的当前日期和时间。



本示例要用到的资源


Middleware
FATFS
System Core
Analog
Timers
Connectivity
Multimedia
DMA
GPIO
NVIC
RCC
SYS
RTC
FSMC
SDIO

1. SDIO设置

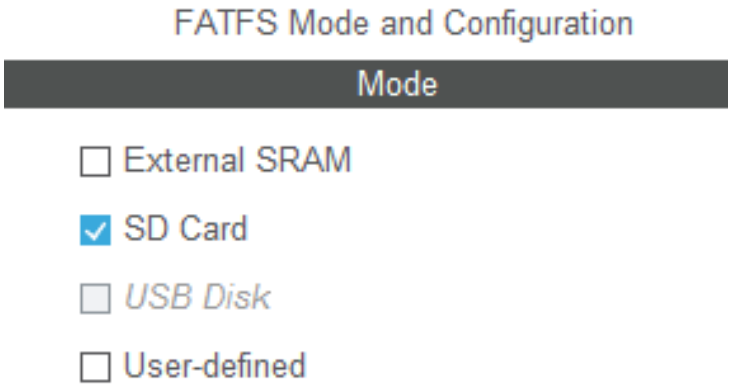


- 分频系数（SDIOCLK clock divide factor）决定了给SD卡的时钟信号SDIO_CK的频率，这里的系数设置为4，SDIO_CK的频率就是8 MHz。

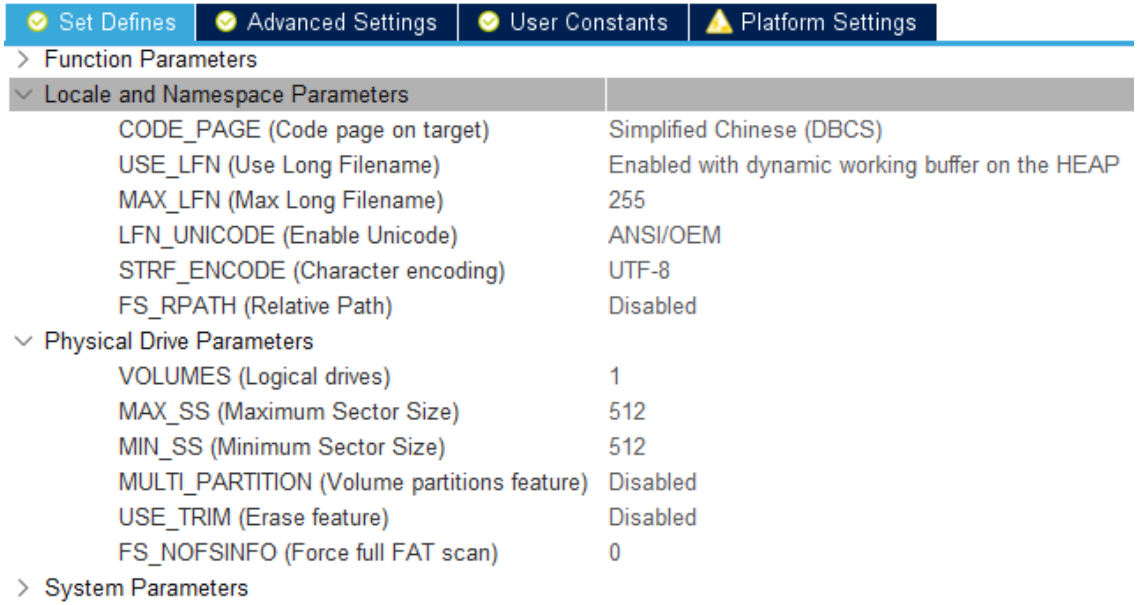
$$f_{\mathrm{SDIO\_CK}} = \frac{f_{SDIOCLK}}{2+N}$$

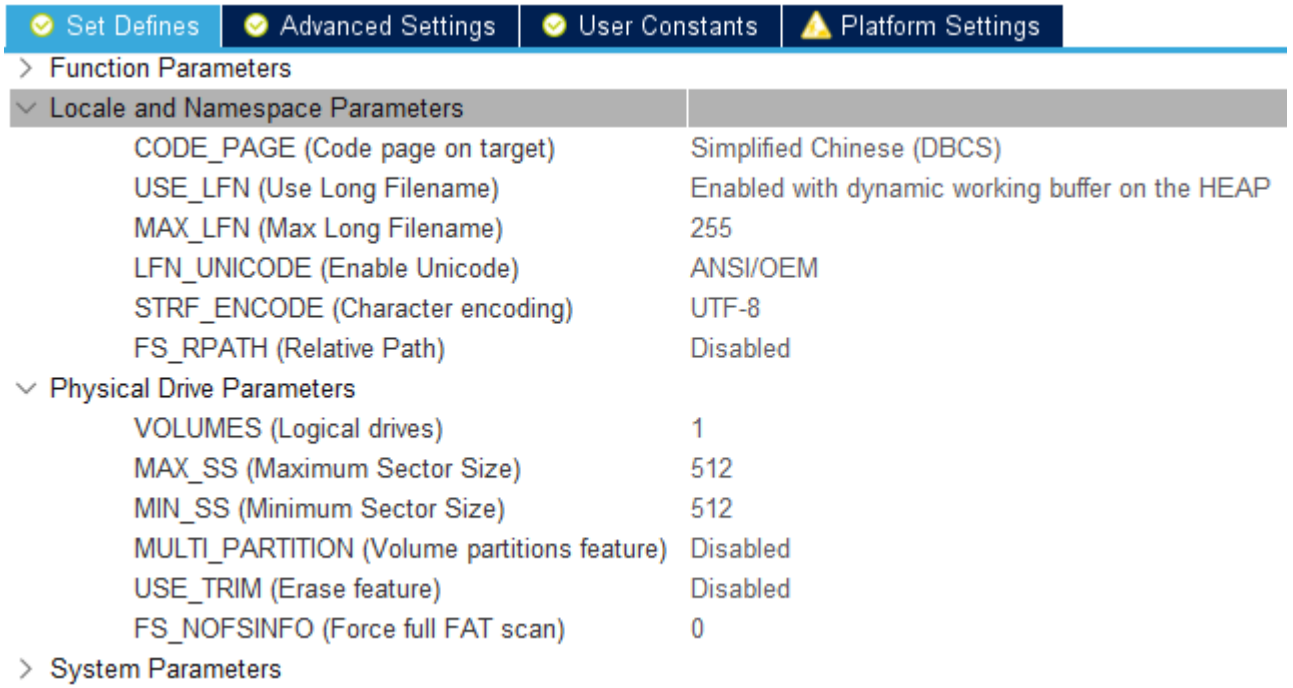
2. FatFS设置

- 模式设置中选择SD Card
- 参数设置部分有4个页面，相对于使用User-defined模式时多了“Advanced Settings”和“Platform Settings”

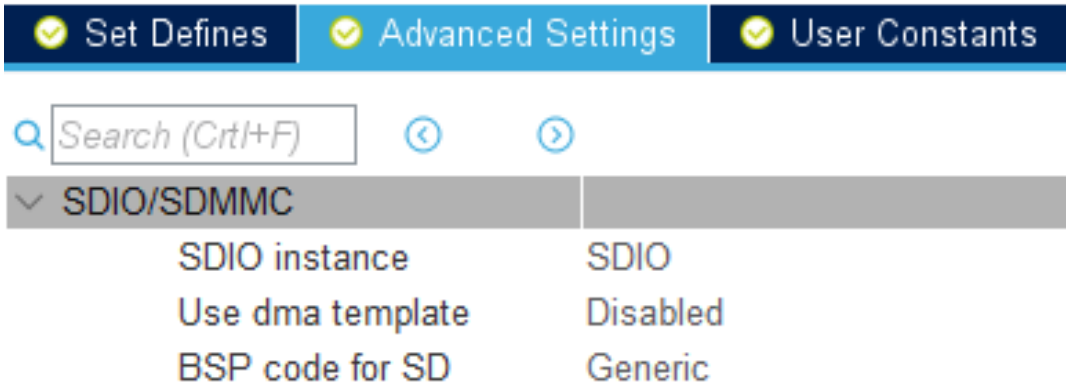


模式设置

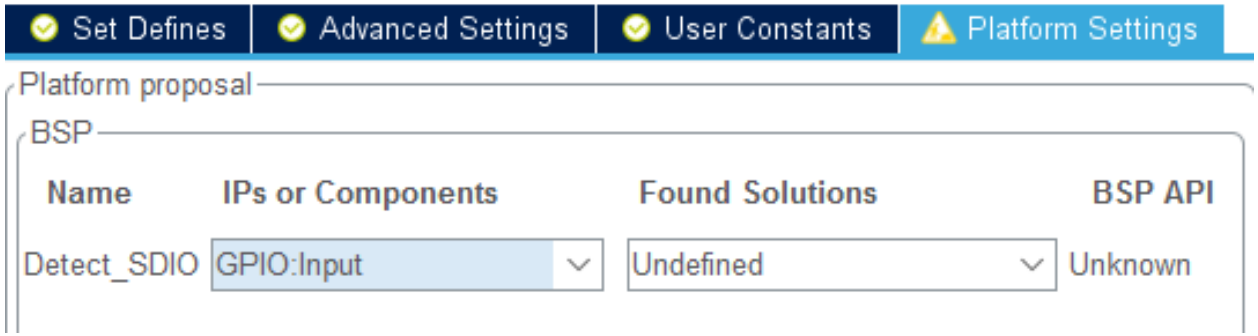




- CODE_PAGE（代码页）设置为Simplified Chinese（简体中文）
- USE_LFN（使用长文件名）设置由FatFS在其堆空间自动为LFN分配存储空间
- MAX_SS（最大扇区大小）和MIN_SS（最小扇区大小）都自动设置为512，SD卡的块大小为512字节，FatFS的扇区就对应于SD卡的块

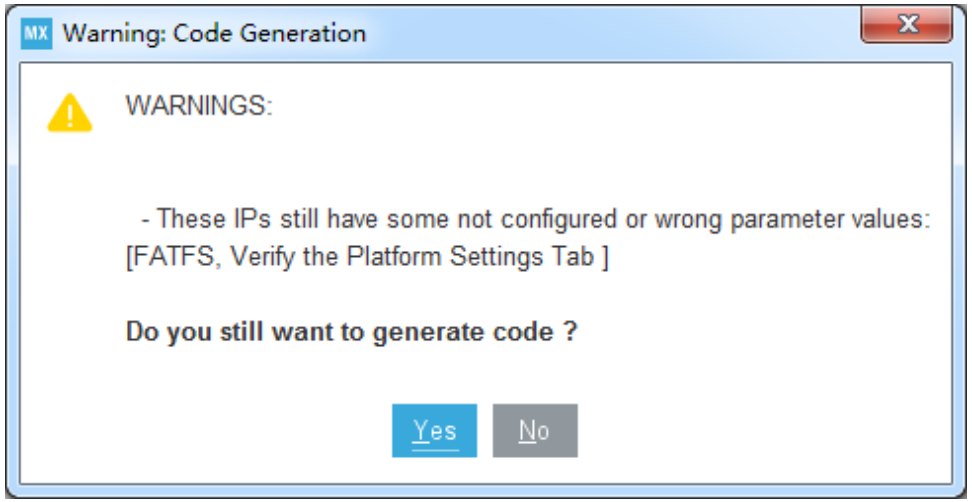


- SDIO instance，SDIO接口方式。只能设置为SDIO，STM32F4的SDIO接口不支持SPI模式。
- Use dma template，是否使用DMA模板。如果要使用DMA方式进行SD卡数据读写，就需要设置为Enabled。
- BSP code for SD，SD卡的BSP（bord support package）代码，这个参数固定为Generic，也就是只能使用CubeMX自动生成的SD卡BSP代码。



“Platform Settings”页面用于设置SD卡的卡检测信号引脚CD。开发板上microSD卡槽没有CD信号，所以在第二个下拉列表框中选择Undefinded即可。

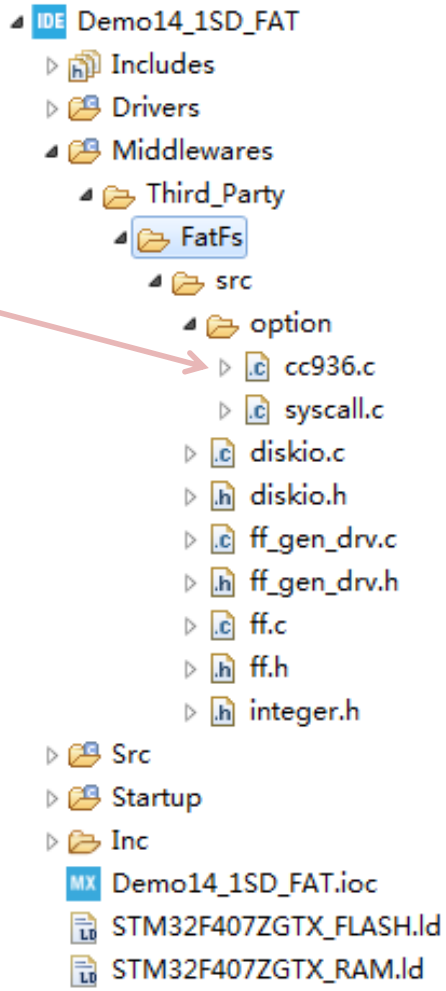
生成代码时会出现如图所示的警告对话框，提示FatFS的“Platform Settings”页的参数没有设置。点击Yes按钮，仍然生成代码即可



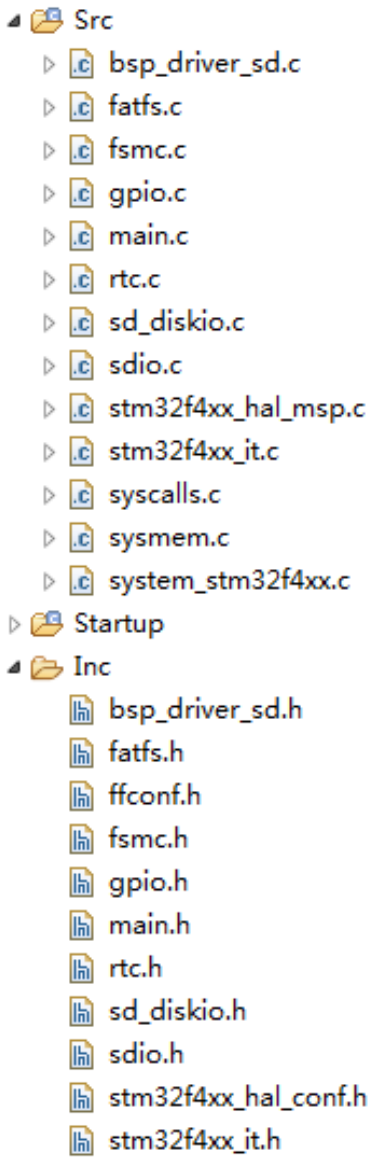
14.2.2 项目文件组成和初始代码分析

1. 项目文件组成

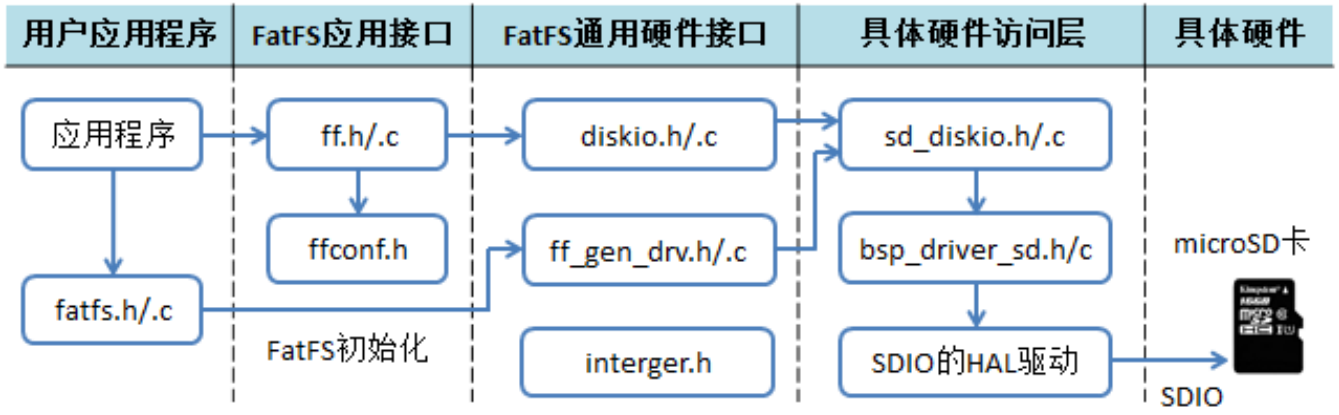
中文编码相关文件



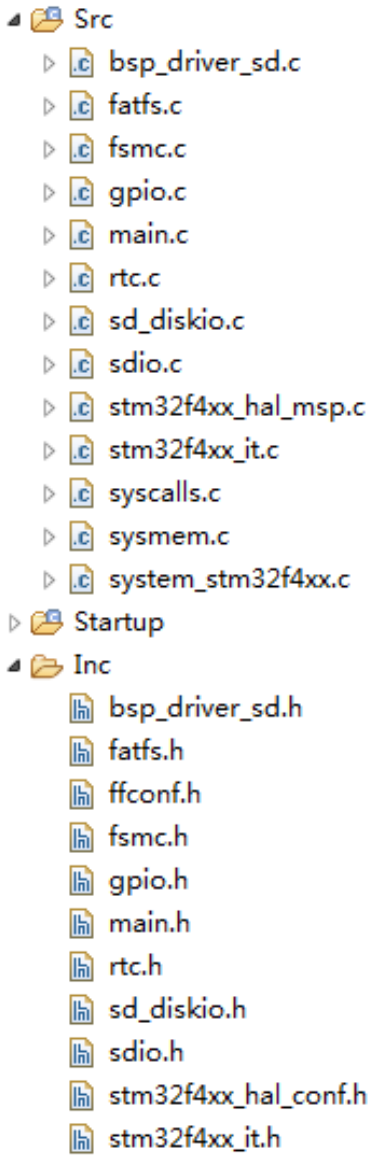
FatFS文件



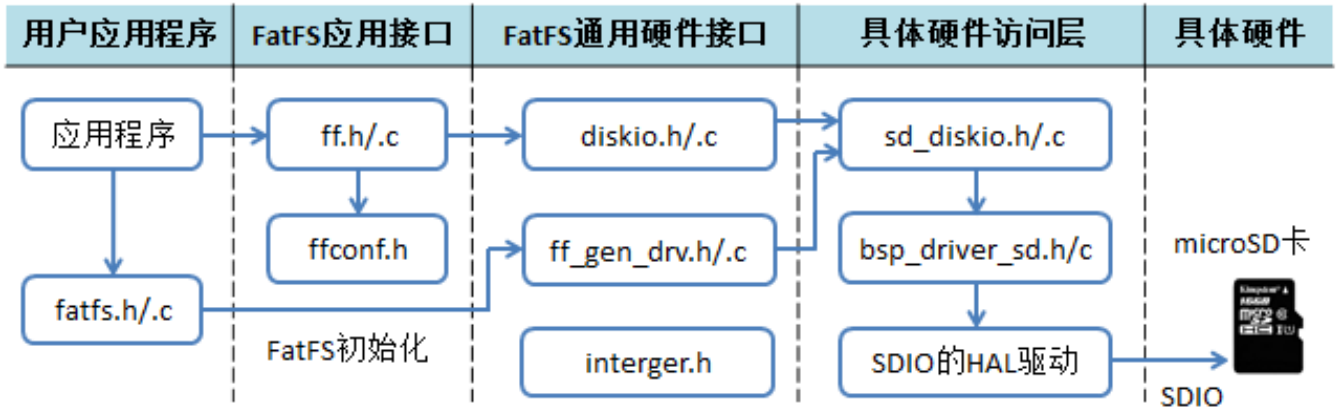
用户程序文件



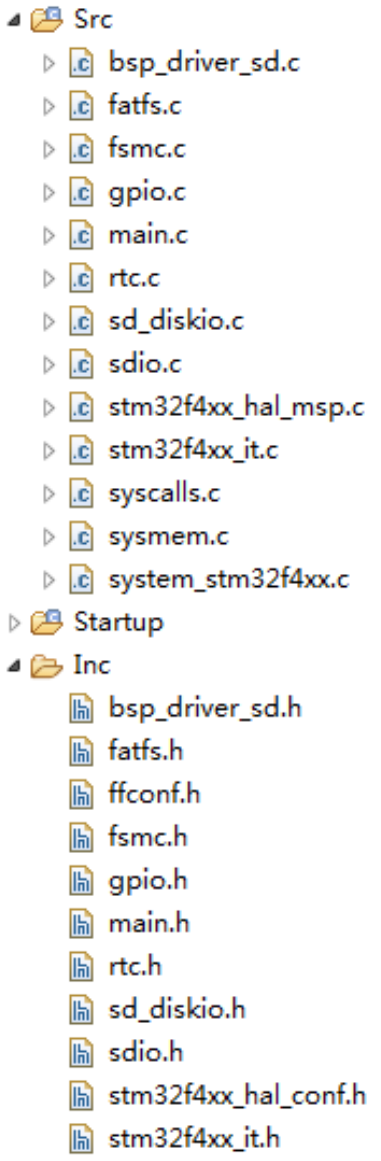
- sdio.h和sdio.c，是SDIO接口的外设初始化程序文件，包含函数MX_SDIO_SD_Init()
- ffconf.h，是FatFS的配置文件，包含很多的宏定义，与CubeMX里的FatFS设置对应
- fatfs.h和ftafs.c，这是用户的FatFS初始化程序文件，包括FatFS初始化函数MX_FATFS_Init()
- sd_diskio.h和sd_diskio.c，是实现了SD卡的Disk IO函数的程序文件，例如SD_read()，SD_write()。无需用户再编写任何代码



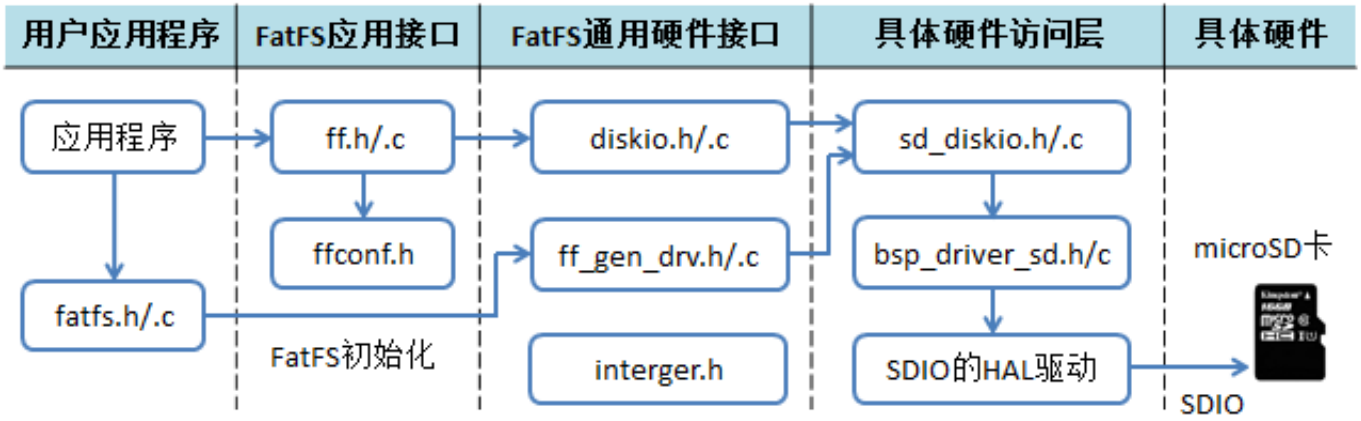
用户程序文件



- bsp_driver_sd.h和bsp_driver_sd.c，这是SD卡各种具体操作的BSP函数的程序文件，如读取SD卡数据块的函数BSP_SD_ReadBlocks()，而这个函数则是调用HAL驱动函数HAL_SD_ReadBlocks()实现的。

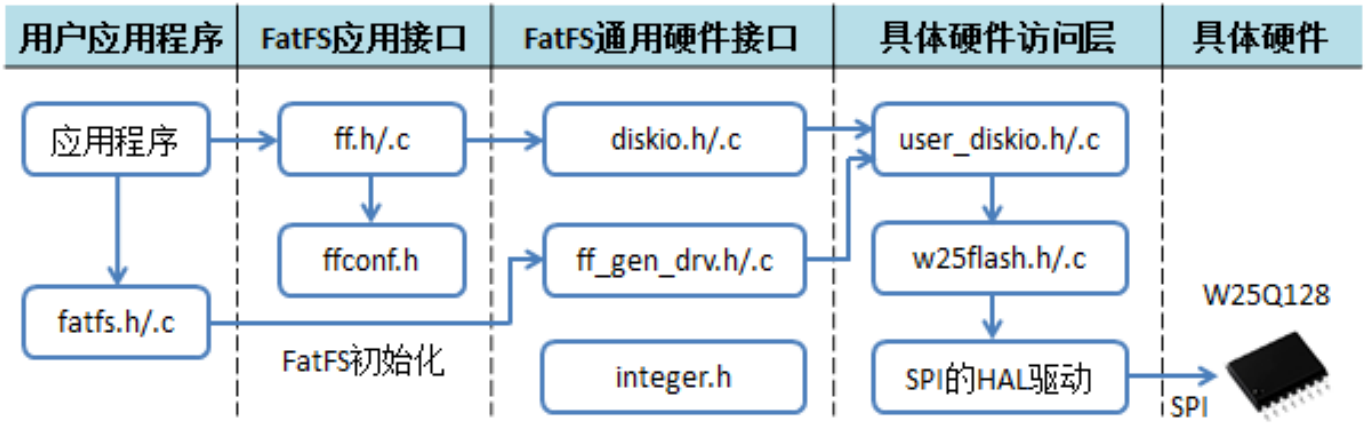


用户程序文件



SD卡的FatFS文件

对比：差异就在具体硬件访问层



Flash芯片的FatFS文件

2. 初始主程序

```
/* 文件： main.c  ------------------------------------------------------------*/
#include "main.h"
#include "fatfs.h"
#include "rtc.h"
#include "sdio.h"
#include "gpio.h"
#include "fsmc.h"

int main(void)
{
        HAL_Init();
        SystemClock_Config();
        /* Initialize all configured peripherals */
        MX_GPIO_Init();
        MX_FSMC_Init();
        MX_FATFS_Init();                    //FatFS初始化
        MX_RTC_Init();
        MX_SDIO_SD_Init();                  //SDIO初始化
        /* Infinite loop */
        while (1)
        {
        }
}
```

3. SDIO初始化

主程序中调用MX_SDIO_SD_Init()对SDIO接口进行初始化，但是没有完全初始化，没有调用函数HAL_SD_Init()进行SDIO接口的初始化

```
#include "sdio.h"
SD_HandleTypeDef hsd;                    //SDIO外设对象变量，也用于表示SD卡

/*  SDIO初始化，只设置hsd属性，并未调用HAL_SD_Init()进行SDIO初始化  */
void MX_SDIO_SD_Init(void)
{
    hsd.Instance = SDIO;      //寄存器基址
    hsd.Init.ClockEdge = SDIO_CLOCK_EDGE_RISING;
    hsd.Init.ClockBypass = SDIO_CLOCK_BYPASS_DISABLE;
    hsd.Init.ClockPowerSave = SDIO_CLOCK_POWER_SAVE_DISABLE;
    hsd.Init.BusWide = SDIO_BUS_WIDE_1B;
    hsd.Init.HardwareFlowControl = SDIO_HARDWARE_FLOW_CONTROL_DISABLE;
    hsd.Init.ClockDiv = 4;
}
```

4. FatFS的初始化

MX_FATFS_Init()是在文件fasfs.h和fastfs.c中定义和实现的FatFS初始化函数。文件fatfs.h的完整代码如下：

```
/* 文件：fatfs.h  ----------------------------------------------------------*/
#include "ff.h"
#include "ff_gen_drv.h"
#include "sd_diskio.h"                  //定义了SD_Driver 及其底层Disk IO函数

extern uint8_t retSD;                   //用于一般函数返回值
extern char SDPath[4];                  //SD 逻辑驱动器路径，即”0:/”
extern FATFS SDFatFS;                   //SD 逻辑驱动器文件系统对象
extern FIL SDFile;                                  //SD 卡上的文件对象

void MX_FATFS_Init(void);//FatFS初始化函数
```

文件fatfs.c的完整代码如下：

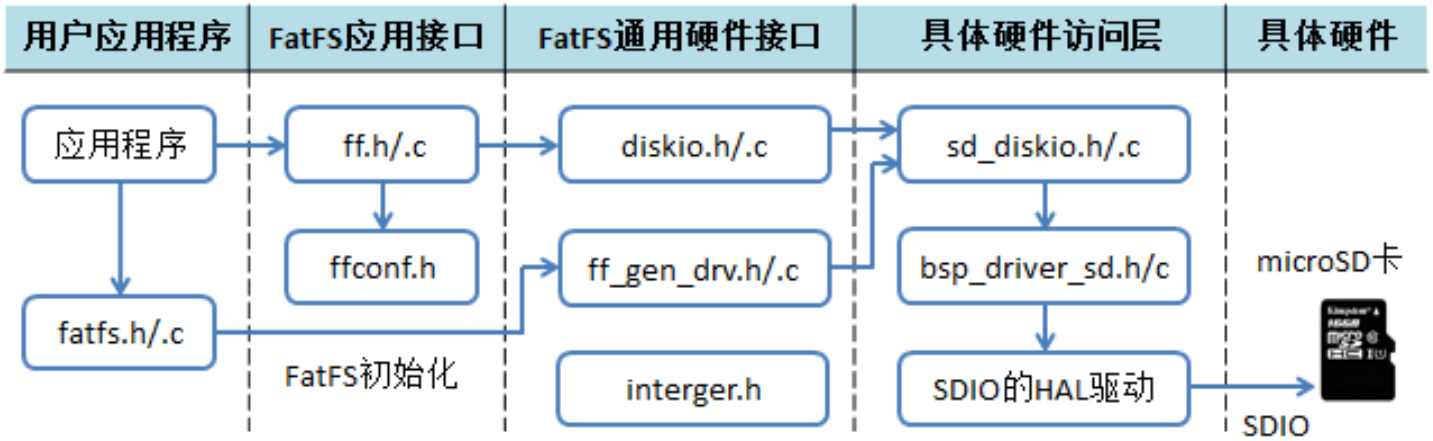
```
/* 文件：fatfs.c  -------------------------------------------------------------*/
#include "fatfs.h"

uint8_t retSD;                          //用于一般函数返回值
char SDPath[4];                         //SD 逻辑驱动器路径，即”0:/”
FATFS SDFatFS;                          //SD 逻辑驱动器文件系统对象
FIL SDFile;                             //SD 卡上的文件对象

/* USER CODE BEGIN Variables */
#include "file_opera.h"
/* USER CODE END Variables */

void MX_FATFS_Init(void)
{
        /*## FatFS: 连接 SD 驱动器 ###############################*/
        retSD = FATFS_LinkDriver(&SD_Driver, SDPath);
}

/*  获取RTC时间作为文件系统的时间戳数据  */
DWORD get_fattime(void)
{
        /* USER CODE BEGIN get_fattime */
        return fat_GetFatTimeFromRTC();
        /* USER CODE END get_fattime */
}
```

14.2.3 SD卡的Disk IO函数实现

1. 文件sd_diskio.h/.c概览

文件sd_diskio.h中只有一行有效语句，就是对外声明了变量SD_Driver，表示SD驱动器

```
/* 文件：sd_diskio.h    ----------------------------------------------------*/
#include "bsp_driver_sd.h"
extern const Diskio_drvTypeDef  SD_Driver;    //SD驱动器
```



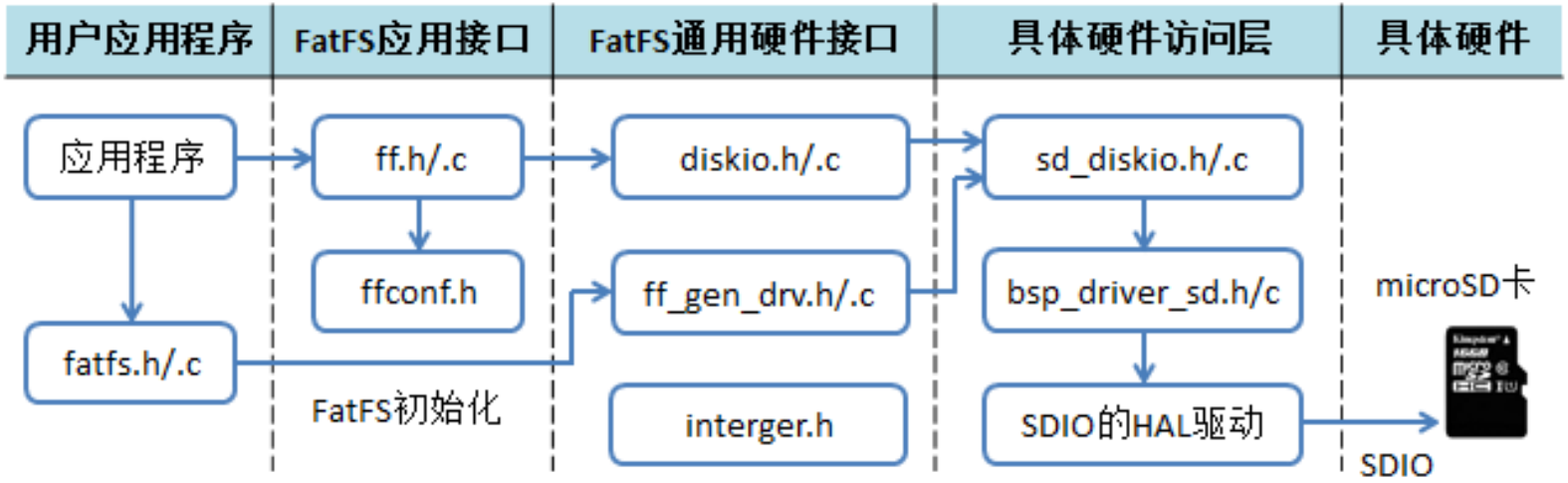
文件sd_diskio.c定义了5个Disk IO函数，并赋值给SD_Driver的函数指针

```
#include "ff_gen_drv.h"
#include "sd_diskio.h"

#define SD_TIMEOUT  SDMMC_DATATIMEOUT          //BSP驱动的timeout
#define SD_DEFAULT_BLOCK_SIZE 512              //SD卡默认Block大小，512字节
static volatile DSTATUS Stat = STA_NOINIT;     //Disk status

/* Private function prototypes -----------------------------------------------*/
static DSTATUS SD_CheckStatus(BYTE lun);       //检查SD卡状态
//以下是Disk IO访问关联的函数
DSTATUS SD_initialize(BYTE);     //SD卡初始化，关联函数disk_initialize()
DSTATUS SD_status(BYTE);                  //SD卡状态，关联函数disk_status()
DRESULT SD_read(BYTE, BYTE*, DWORD, UINT);                  //读取SD卡，关联函数disk_read()
DRESULT SD_write(BYTE, const BYTE*, DWORD, UINT);           //写入SD卡，关联函数disk_write()
DRESULT SD_ioctl(BYTE, BYTE, void*);           //SD卡IO控制，关联函数disk_ioctl()

/*下面的代码是对结构体变量SD_Driver的各成员变量赋值，结构体类型Diskio_drvTypeDef在文件
*   ff_gen_drv.h中定义，其成员变量是函数指针,赋值使其指向本文件中定义的USER_函数
*/
const Diskio_drvTypeDef  SD_Driver =
{
        SD_initialize,                 //函数指针disk_initialize 指向函数 SD_initialize()
        SD_status,                     //函数指针disk_status 指向 SD_status()
        SD_read,                       //函数指针disk_read 指向 SD_read()
        SD_write,                      //函数指针disk_write 指向SD_write()
        SD_ioctl,                      //函数指针disk_ioctl 指向SD_ioctl()
};
```

2. 文件bsp_driver_sd.h/.c概览

文件bsp_driver_sd.h/.c是SD卡的BSP驱动程序，也就是针对开发板上SD卡具体硬件电路实现的SD卡常用操作函数的程序文件。BSP驱动程序是比HAL驱动程序高一级的驱动程序。

详见示例完整源程序



6. 函数SD_read()的实现【为例】

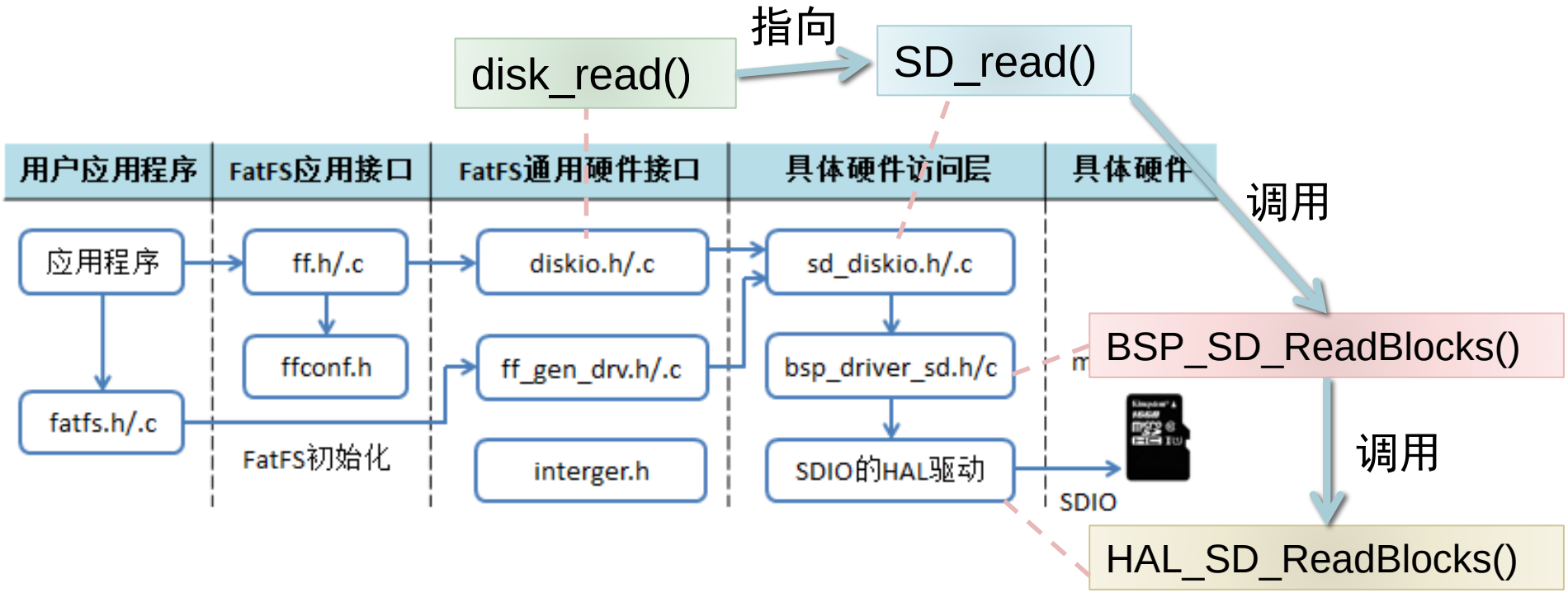
文件sd_diskio.c中的函数SD_read()关联通用Disk IO函数disk_read()，用于从SD卡读取数据。读取数据的最小单位是扇区，也就是SD卡的数据块，可以一次读取一个或多个扇区的数据。函数SD_read()的代码如下：

```
DRESULT SD_read(BYTE lun, BYTE *buff, DWORD sector, UINT count)
{
	DRESULT res = RES_ERROR;
	if (BSP_SD_ReadBlocks((uint32_t*)buff, (uint32_t) (sector),
				count, SD_TIMEOUT) == MSD_OK)
	{	/*  等待读取操作完成  */
		while(BSP_SD_GetCardState()!= MSD_OK) {
				}
		res = RES_OK;
	}
	return res;
}
```

BSP函数BSP_SD_ReadBlocks()在文件bsp_driver_sd.c中实现，就是调用SDIO的HAL驱动函数HAL_SD_ReadBlocks()读取SD卡的数据块。

```
__weak uint8_t BSP_SD_ReadBlocks(uint32_t *pData, uint32_t ReadAddr,
                                 uint32_t NumOfBlocks, uint32_t Timeout)
{
    uint8_t sd_state = MSD_OK;
    if (HAL_SD_ReadBlocks(&hsd, (uint8_t *)pData, ReadAddr,
                                 NumOfBlocks, Timeout) != HAL_OK)
        sd_state = MSD_ERROR;
    return sd_state;
}
```

disk_read()的实现示意图



14.2.4 SD卡文件管理功能的实现

1. 主程序功能

与第12章的示例Demo12_1FlashFAT类似，本示例在主程序中建立两级菜单，在SD卡上测试使用文件管理功能。

使用了长文件名。

[1]KeyUp =Format SD card
[2]KeyLeft =FAT disk info
[3]KeyRight=SD card info
[4]KeyDown =Next menu page

[5]KeyUp =Write files
[6]KeyLeft =Read a TXT file
[7]KeyRight=Read a BIN file
[8]KeyDown =Get a file info

2. SD卡格式化

响应菜单项“[1]KeyUp =Format SD card”，对SD卡格式化操作的代码如下

```
BYTE  workBuffer[4*BLOCKSIZE];              //工作缓存区
DWORD  clusterSize=0;    //cluster大小必须大于或等于1个扇区,0就是自动设置
LCD_ShowStr(10,LCD_CurY,(uint8_t*)"Formating (10secs)...");
FRESULT res =f_mkfs("0:", FM_FAT32, clusterSize,  workBuffer, 4*BLOCKSIZE);
```

簇（Cluster）的大小必须是扇区的整数倍，在调用函数f_mkfs()进行SD卡格式化时如果设置簇大小为0，就由FatFS自动确定簇的大小。由于测试中使用的SD卡容量是16GB，只能使用FAT32文件系统。

3. 获取FAT磁盘信息

菜单项“[2]KeyLeft =FAT disk info”的响应代码就是调用了测试函数fatTest_GetDiskInfo()。

在已经创建了4个文件和2个目录后，一个16GB的SD卡的磁盘信息在LCD上显示内容如下：

```
FAT type=3
[1=FAT12,2=FAT16,3=FAT32,4=exFAT]
Sector size(bytes)=512
Cluster size(sectors)=64
Total cluster count=475716
Total sector count=30445824
Total space(MB)=14866
Free cluster count=475709
Free sector count=30445376
Free space(MB)=14865
```

- 管理数据的最小单位是“簇”
- 1个簇 = 64个扇区（可设置）
- 4个文件和2个目录，用掉了7个簇（还有文件分配表），很浪费空间

4. 获取SD卡信息

函数SDCard_ShowInfo()显示的是SD卡的原始信息，即使SD卡没有被格式化，也可以返回这些信息。所以，使用同一张SD卡时，本示例显示的SD卡信息与示例Demo13_1SDRaw显示的内容是一样的。

```
Card Type= 1
Card Version= 1
Card Class= 1461
Relative Card Address= 7
Block Count= 30449664
Block Size(Bytes)= 512
LogiBlockCount= 30449664
LogiBlockSize(Bytes)= 512
SD Card Capacity(MB)= 14868
```

练习任务

1. 看教材，做本章示例