SPI FLASH 及 FATFS 文件系统调试

2013 年 9 月 14 日

修改记录

制定日期	生效日期	制定 / 修订 内容摘要	页数	版本	拟稿	审查	批准
2013.09.14		初稿	14	0.01	朱正晶		

目 录

1 本文档组成部分

主要由以下几个方面组成：

① STM32 SPI 模块介绍

② AT45DB321D SPI FLASH 驱动实现

③ FATFS 文件系统移植及调试

④ 系统稳定性测试

2 STM32 SPI

STM32F207 系列有 3 个 SPI 接口，SPI1 挂接在 APB2 上，最大速度为 30Mbits/s（60MHz/2）。SPI2 和 SPI3 挂接在 APB1 上，最大速度为 15Mbit/s（30MHz/2）。

AT45DB321D FLASH 最大操作速度为 66MHz，因此在这里我们选择 SPI1 来和 SPI FLASH 通信。

图 2-1 为 SPI 框图。SPI 有四条线，MISO 主机接收数据从机发送数据，MOSI 主机发送数据从机接收数据，SCK 由 SPI 主机提供的时钟信号，NSS，片选。

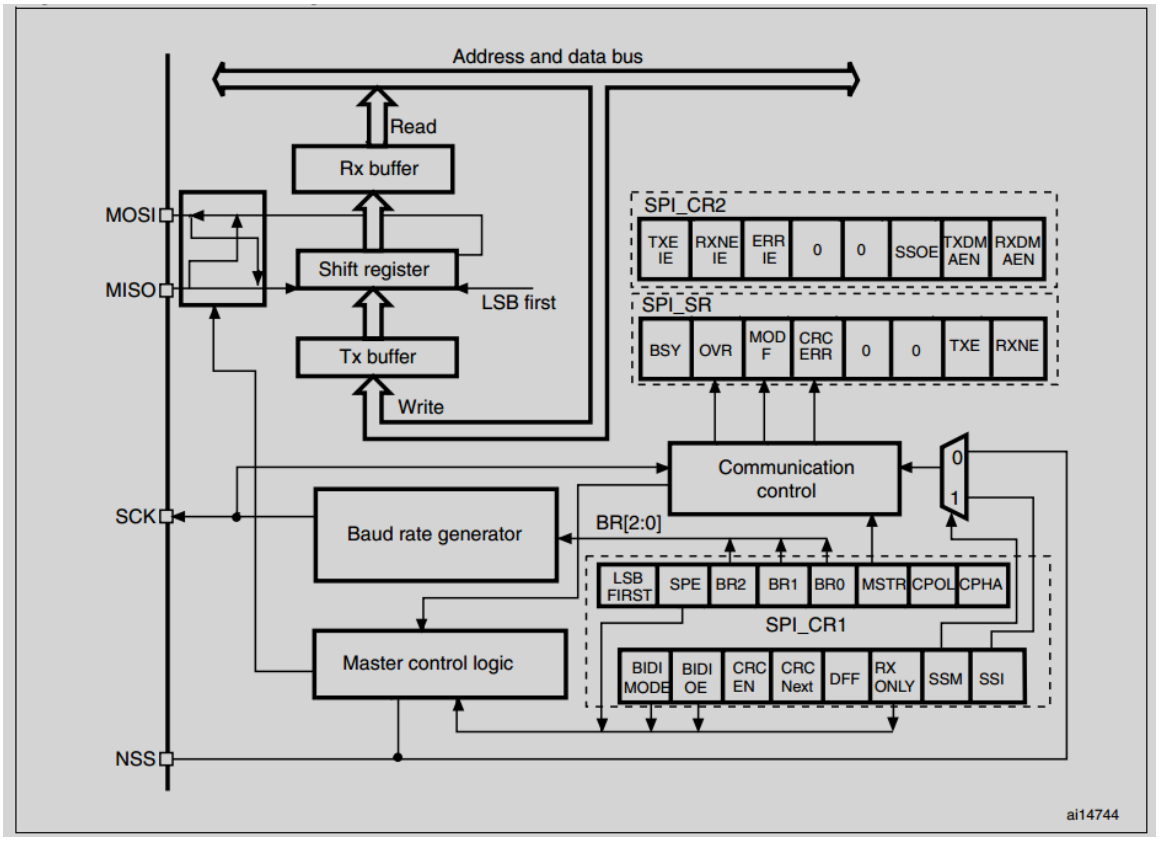

图 2-1 SPI 框图

普通的 SPI 有四种操作方式，如图 2-2 所示。

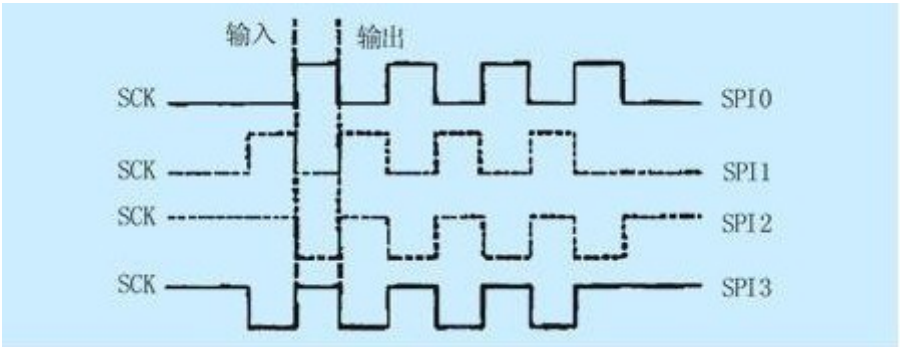

图 2-2 SPI 四种操作方式

图 2-3 为 STM32 SPI 分别对应上面四种方式。

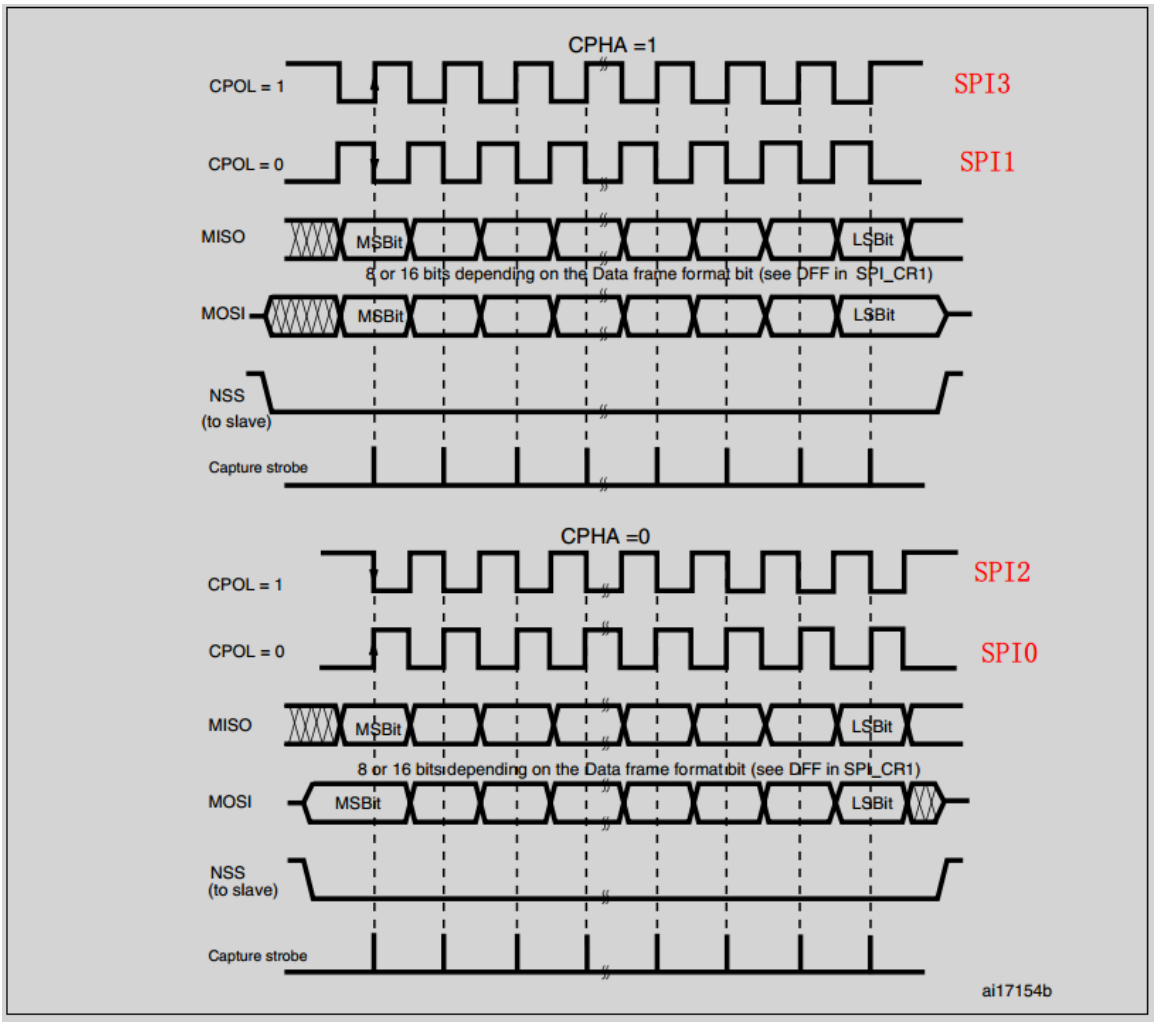

图 2-3 STM32 SPI 时序图

AT45DB321D SPI FLASH 支持 SPI0 和 SPI3 模式，这里我们选择 SPI0 操作方式。STM32 SPI 进行如下配置就行：

CPOL 为 LOW

CPHA 为第一个脉冲

3 AT45DB321D SPI FLASH 驱动

AT45DB321D 为 Atmel 公司生产的 32Mbit SPI FLASH，最大操作速度为 66MHz。SPI 操作方式支持 SPI0 和 SPI3 模式。每页大小为 512 bytes。图 3-1 为 SPI FLASH 的封装引脚。

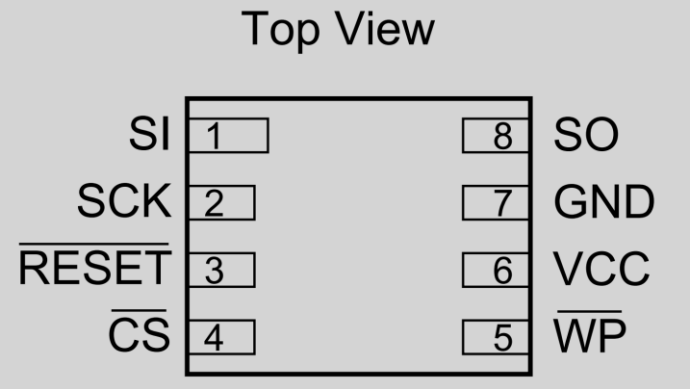

图 3-1 封装引脚

表 3-1 为各个引脚说明

引脚	说明
SI	SPI FLASH 数据输入
SO	SPI FLASH 数据输出
SCK	时钟信号输入
GND	电源地
VCC	电源，3.3V
CS	片选信号，低有效
RESET	复位，低有效
WP	写保护，低有效。低电平时无法更改 FLASH 中的内容

表 3-1 各个引脚说明

3.1 操作指令

3.1.1 读 FLASH ID

SPI 发送一个字节 0x9F，然后接收四个字节，四个字节即为 FLASH ID。

3.1.2 读 FLASH 状态

FLASH 在操作之前必须先读状态，如果 FALSH 不忙，即可进行下一步操作。否则必须等到。

SPI 发送一个字节 0xD7，然后读取一个字节，判断 bit7 即可获知 FLASH 状态，0 为忙，1 为空闲。

下面的编码列出了状态各个 bit 的意义。

```
/*****************************************************************************/
/* Status Register Format:                                                   */
/*  ---------------------------------------------------------------------    */
/*  | bit7  | bit6  | bit5  | bit4  | bit3  | bit2  | bit1  | bit0  |       */
/*  |-------|-------|-------|-------|-------|-------|-------|-------|       */
/*  |RDY/BUSY| COMP |   0   |   1   |   1   |   1   |   X   |   X   |       */
/*  ---------------------------------------------------------------------    */
/*  bit7 - 忙标记，0 为忙 1 为不忙。                                          */
/*         当 Status Register 的位 0 移出之后，接下来的时钟脉冲序列将使 SPI 器件继续  */
/*         将最新的状态字节送出。                                            */
/*  bit6 - 标记最近一次 Main Memory Page 和 Buffer 的比较结果，0 相同，1 不同。  */
/*  bit5                                                                     */
/*  bit4                                                                     */
/*  bit3                                                                     */
/*  bit2 - 这 4 位用来标记器件密度，对于 AT45DB041B，这 4 位应该是 0111，一共能标记  */
/*         16 种不同密度的器件。                                             */
/*  bit1                                                                     */
/*  bit0 - 这 2 位暂时无效                                                    */
/*****************************************************************************/
```

3.1.3 芯片整片擦除

发送命令序列 0xC7 0x94 0x80 0x9A，然后芯片会自动进入擦除状态，我们需要就是等待擦除完成。

3.1.4 擦除一页

发送命令 0x81，页地址，发送一个随意字节。等待完成。

3.1.5 读 FLASH

读 FLASH 有几种方式，这里选择了 CONTINUOUS_ARRAY_READ。

先发送 CONTINUOUS_ARRAY_READ 命令 0x0B，接下来送页地址和字节地址最后随便发送一个字节。之后就可以读取数据了。

3.1.6 写 FLASH

写 FLASH 分为两个步骤：

1）将数据写到 FLASH 内部 buffer

发送 0x84，发送页地址（随意），发送字节地址，发送要写的数据。

2）将 buffer 内容写到 flash 中

发送 0x83，发送页地址，发送一个字节（随意），等待完成。

3.2 shell 测试命令

为了测试 SPI FLASH，我在 shell 程序中加入了几条测试命令。下面分别介绍。

3.2.1 宏定义

首先为了系统的可移植性，在系统顶层头文件（stm32f2xx_conf.h）中加入了 **USE_SPI_FLASH_AT45DB_FTR**，打开此宏以下命令才可用。

3.2.2 命令说明

命令以 **flash** 开头，后面可以加参数。一共有 5 条命令。注意，必须先初始化 SPI FLASH，否则有可能引起系统卡死。

使用 **flash -h** 获取帮助信息。

```
"flash usage:"
"-I                 get FLASH id"
"-init              init SPI flash"
"-e [PA]            erase whole chip or spec PAGE"
"-w [PA] [BA byte]  write test data to spec PA or assign a byte to BA"
"-r  PA             read PAGE data"
"-h                 get help information"
```

例子：

1）初始化 SPI FLASH

```
VINY>flash -init
SPI FLASH init...done
```

2）得 SPI FLASH ID

```
VINY>flash -i
SPI FLASH ID[0x1f270100] -- AT45DB321D [page size 512]

VINY>
```

3）获取帮助信息

```
VINY>flash -h
flash usage:
-i                 get FLASH id
-init              init SPI flash
-e [PA]            erase whole chip or spec PAGE
-w [PA] [BA byte]  write test data to spec PA or assign a byte to BA
-r  PA             read PAGE data
-h                 get help information

VINY>
```

4）擦除整片芯片或擦除某一页

```
VINY>flash -e
erase whole chip...done

VINY>flash -e 1
erase FLASH PAGE[1]...done
```

5）读某一页数据，每页有 512 字节。

```
VINY>flash -r 1000
read FLASH PAGE[1000]...
5B 34 38 38 5D 48 65 6C 6C 6F 2C 20 74 68 69 73 [488]Hello, this
20 69 73 20 42 72 75 63 65 20 66 6F 72 20 74 65  is Bruce for te
73 74 20 66 69 6C 65 0A 00 00 00 00 00 00 00 00 st file.........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

VINY>
```

6）在某一页上特定的地址写数值，比如写十进制数 4 到 FLASH 第 1 页第 2 个字节。

```
VINY>flash -w 1 1 4
Write 04 to FLASH PAGE[1] BYTE[1]...done

VINY>flash -r 1
read FLASH PAGE[1]...
FF 04 FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
```

4 FATFS 文件系统移植及调试

4.1 简介

FatFs 是一个为小型嵌入式系统设计的通用 FAT(File Allocation Table)文件系统模块。FatFs 的编写遵循 ANSI C，并且完全与磁盘 I/O 层分开。因此，它独立(不依赖)于硬件架构。它可以被嵌入到低成本的微控制器中，如 AVR、8051、PIC、ARM、Z80、68K 等等，而不需要做任何修改。

图 4-1 为 FATFS 在系统中所处的位置。

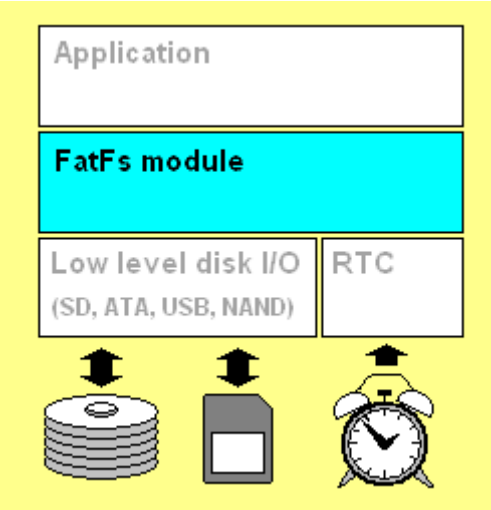

图 4-1 FATFS 文件系统

4.2 特点

1）Windows 兼容的 FAT 文件系统
2）不依赖于平台，易于移植
3）代码和工作区占用空间非常小
4）多种配置选项
- 多卷(物理驱动器和分区)
- 多 ANSI/OEM 代码页，包括 DBCS
- 在 ANSI/OEM 或 Unicode 中长文件名的支持
- RTOS 的支持
- 多扇区大小的支持
- 只读，最少 API，I/O 缓冲区等等

5）API 简单，使用方便

4.3 系统组织

下面的依赖图（图 4-2）给出了使用 FatFs 模块在嵌入式系统的典型配置。

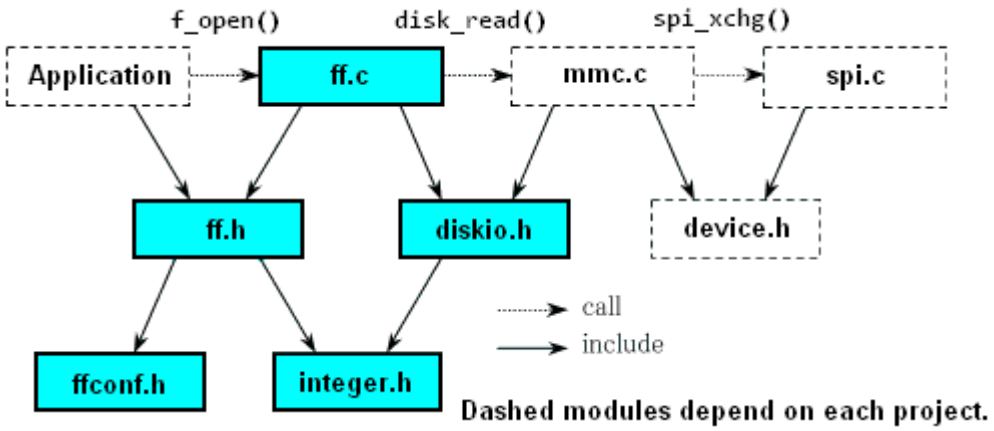

图 4-2 FATFS 依赖图

4.4 移植

只需要提供 FatFs 模块需要的底层的磁盘 I/O 函数即可。如果已经存在一个目标系统的工作磁盘模块，则只需要写聚合函数附加到 FatFs 模块即可。如果没有，则需要移植任何其他的磁盘模块或写一个。所有已经定义的函数不需要再写了。例如，磁盘写函数在只读配置方面就不需要。下表给出了根据配置选项需要哪些函数。

Function	Required when:	Note
disk_initialize disk_status disk_read	Always	Disk I/O functions. Samples available in ffsample.zip. There are many implementations on the web.
disk_write get_fattime disk_ioctl (CTRL_SYNC)	_FS_READONLY == 0	
disk_ioctl (GET_SECTOR_COUNT) disk_ioctl (GET_BLOCK_SIZE)	_USE_MKFS == 1	
disk_ioctl (GET_SECTOR_SIZE)	_MAX_SS > 512	
disk_ioctl (CTRL_ERASE_SECTOR)	_USE_ERASE == 1	
ff_convert ff_wtoupper	_USE_LFN >= 1	Unicode support functions. Available in option/cc*.c.
ff_cre_syncobj ff_del_syncobj ff_req_grant ff_rel_grant	_FS_REENTRANT == 1	O/S dependent functions. Samples available in option/syscall.c.
ff_mem_alloc ff_mem_free	_USE_LFN == 3	

其实移植工作分为两步：

1）配置文件 ffconf.h 中的一些选项配置

这里我们选择基本配置，没有加入中文支持（需要存字库，占用空间比较大）。具体的参考源代码。

2）只需要将 diskio.c 文件中几个函数填写完整就行。

下面的函数为移植 OK 的。具体的请参考代码。

```
/* Definitions of physical drive number for each media */
#define SPI_FLASH                0

/*-----------------------------------------------------------------------*/
/* Inidialize a Drive                                                    */
/*-----------------------------------------------------------------------*/
```

DSTATUS **disk_initialize** (

```
    BYTE pdrv            /* Physical drive nmuber (0..) */
)
{
    DSTATUS stat = STA_NOINIT;
    u32     flash_id;

    switch (pdrv)
    {
        case SPI_FLASH:
            {
                spi_flash_init();
                flash_id = read_flash_id();
                if (AT45DB321D == flash_id)
                {
                    stat = 0;
                }
            }
            break;

        default:
            break;
    }

    return stat;
}
/*-----------------------------------------------------------------------*/
/* Get Disk Status                                                       */
/*-----------------------------------------------------------------------*/
```

DSTATUS **disk_status** (

```
    BYTE pdrv     /* Physical drive nmuber (0..) */
)
{
    if (check_flash_status() == 1)
        return 0;
    else
```

```
        return STA_NOINIT;
}
/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/

DRESULT disk_read (
    BYTE pdrv,        /* Physical drive nmuber (0..) */
    BYTE *buff,       /* Data buffer to store read data */
    DWORD sector,/* Sector address (LBA) */
    BYTE count        /* Number of sectors to read (1..128) */
)
{
    DRESULT res = RES_PARERR;
    u32     i;

    switch (pdrv)
    {
        case SPI_FLASH:
            for (i = 0; i < count; i++)
            {
                continuous_array_read(sector,0,(u8*)buff, FLASH_PAGE_SIZE);
                sector++;
                buff += FLASH_PAGE_SIZE;
            }
            res = RES_OK;
            break;

        default:
            break;
    }
    return res;
}

/*-----------------------------------------------------------------------*/
/* Write Sector(s)                                                       */
/*-----------------------------------------------------------------------*/
#if _USE_WRITE

DRESULT disk_write (
    BYTE pdrv,            /* Physical drive nmuber (0..) */
    const BYTE *buff,     /* Data to be written */
    DWORD sector,     /* Sector address (LBA) */
    BYTE count            /* Number of sectors to write (1..128) */
```

```
)
{
    DRESULT res;
    u32     i;
    u8*     p;

    p = (u8*)buff;
    switch (pdrv)
    {
        case SPI_FLASH:
            for (i = 0; i < count; i++)
            {
                // first write data to FLASH internal buffer 1
                flash_buffer_write(1, 0, (u8*)p, FLASH_PAGE_SIZE);
                // second program buffer 1 to FLASH with erase
                program_buf_to_flash(1, sector + i);
                // increase page address
                p += FLASH_PAGE_SIZE;
            }
            res = RES_OK;
            break;
        default:
            break;
    }

    return res;
}
#endif

/*-----------------------------------------------------------------------*/
/* Miscellaneous Functions                                               */
/*-----------------------------------------------------------------------*/
#if _USE_IOCTL
DRESULT disk_ioctl (
    BYTE pdrv,      /* Physical drive nmuber (0..) */
    BYTE cmd,     /* Control code */
    void *buff      /* Buffer to send/receive control data */
)
{
    DRESULT res = RES_PARERR;

    switch (cmd)
    {
```

```
        case CTRL_SYNC:
            res = RES_OK;
            break;
        case GET_SECTOR_SIZE:
            *(DWORD*)buff = FLASH_PAGE_SIZE;
            res = RES_OK;
            break;
        case GET_BLOCK_SIZE:
            *(DWORD*)buff = 512;
            res = RES_OK;
            break;
        case GET_SECTOR_COUNT:
            *(DWORD*)buff = FLASH_SECTOR_COUNT;
            res = RES_OK;
            break;
        default:
            break;
    }

    return res;
}
#endif

/*-----------------------------------------------------------------------*/
/* get system time                                                       */
/*-----------------------------------------------------------------------*/

DWORD get_fattime (void)
{
    return 0;
}
```

4.5 测试

和 flash 测试一样，首先编写 shell 测试命令。

命令以 **fatfs** 开头，后面可以加参数。一共有 9 条命令。使用 **fatfs -h** 获取帮助信息。

```
"fatfs usage:"
"-c                 Create FATFS"
"-w file name       Open file for write"
"-writetest         write 512 files to test FATFS"
"-autotest num      test num of times for writetest"
"-i drive num       get total and free drive space"
"-m drive num       mount drive to FATFS"
"-um drive num      unmount FATFS"
"-s path            scan files"
"-h                 get help information"
```

简单介绍一下：

1）创建文件系统

```
VINY>fatfs -c
Create FATFS...done

VINY>
```

2）创建一个文件并进行写入，红色中的内容为键盘输入。

```
VINY>fatfs -w hello.txt
write file[hello.txt], pleae input data...
Hello, this is Bruce for test file!
Complete, write to file[35]...done

VINY>fatfs -s /
scan files...
//HELLO.TXT size: 35bytes

VINY>
```

3）-writetest 和-autotest num 都是测试命令，用来测试 FATFS 文件系统的稳定性。下面的截图为运行过程中的一些 log。

```
VINY>fatfs -writetest
write file[0]...done
write file[1]...done
write file[2]...done
write file[3]...done
write file[4]...done
write file[5]...done
write file[6]...done
write file[7]...done
write file[7]...done
```

4）当前磁盘的使用情况

```
VINY>fatfs -i 0
Drive Number: 0
Total       : 3840KB
Available   : 3584KB
Used        : 7%

VINY>
```

5）扫描目录下的文件

```
VINY>fatfs -s /
scan files...
//HELLO.TXT size: 35bytes
//BRUCE0.TXT size: 38bytes
//BRUCE1.TXT size: 38bytes
//BRUCE2.TXT size: 38bytes
//BRUCE3.TXT size: 38bytes
//BRUCE4.TXT size: 38bytes
//BRUCE5.TXT size: 38bytes
//BRUCE6.TXT size: 38bytes
//BRUCE7.TXT size: 38bytes
//BRUCE8.TXT size: 38bytes
//BRUCE9.TXT size: 38bytes
//BRUCE10.TXT size: 39bytes
//BRUCE11.TXT size: 39bytes
//BRUCE12.TXT size: 39bytes
//BRUCE13.TXT size: 39bytes
//BRUCE14.TXT size: 39bytes
//BRUCE15.TXT size: 39bytes
//BRUCE16.TXT size: 39bytes
//BRUCE17.TXT size: 39bytes
```

经过几天的测试，文件系统正常。有一点需要注意，在进行文件写的过程中千万不要断电，否则整个 FATFS 文件系统可能会损坏。