GigaDevice Semiconductor Inc.

基于 GD32 MCU 的 FATFS 文件系统移植

应用笔记

AN065

1.0 版本

(2023 年 8 月)

目录

图索引

表索引

1. FATFS 简介

文件系统是用于在存储介质上存储和管理数据的组织结构，其包括系统引导区、目录和文件。在存储介质上建立文件系统前，需要先对存储介质进行格式化擦除原有的数据，然后新建文件分配表和目录，从而便于记录和管理数据存放的物理地址和剩余空间，如同图书管理分类系统一样。

文件系统庞大而复杂，需要根据应用的文件系统格式而编写，并且一般与驱动层分离开来，方便移植，因此工程应用中通常移植现成的文件系统源码。FATFS 是面向小型嵌入式系统的一种通用的 FAT 文件系统。FATFS 基于 ANSIC 语言编写，并且完全独立于底层的 I/O 介质。因此 FATFS 可以很方便地移植到多种处理器之中。

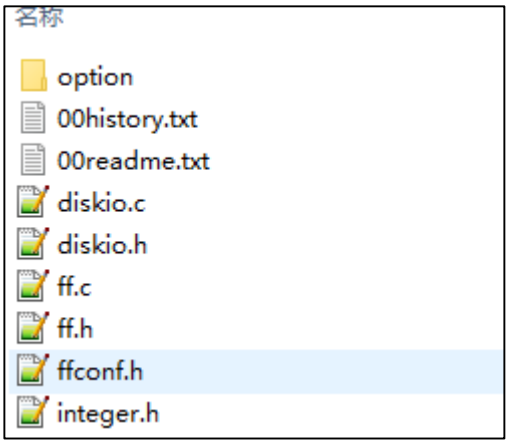
2. FATFS 移植

2.1. FATFS 移植平台

本文基于 GD32103C EVAL 开发板移植 FATFS，通过 FATFS 实现对 SPI-flash(GD25Q16)、片上 flash 以及 SD 卡进行存储数据的文件管理。FATFS 移植的 IDE 平台为 keil4.

FATFS 文件系统包可以从网址 ***http://elm-chan.org/fsw/ff/00index_e.html*** 获取，其包含的文件有：1、与底层接口硬件驱动移植相关的文件：diskio.c、diskio.h；2、与 FATFS 模块相关，用于实现 FAT 文件读/写协议的文件：ff.c、ff.h；3、与 FATFS 模块配置相关的文件：ffconf.h；4、与数据类型定相关的文件：interger.h；5、FATFS 提供支持不同语言的外部功能有关的文件夹：option 文件夹，如*图 2-1. FATFS 包文件结构*所示。本次移植 FATFS 文件系统的版本为 R0.11a.

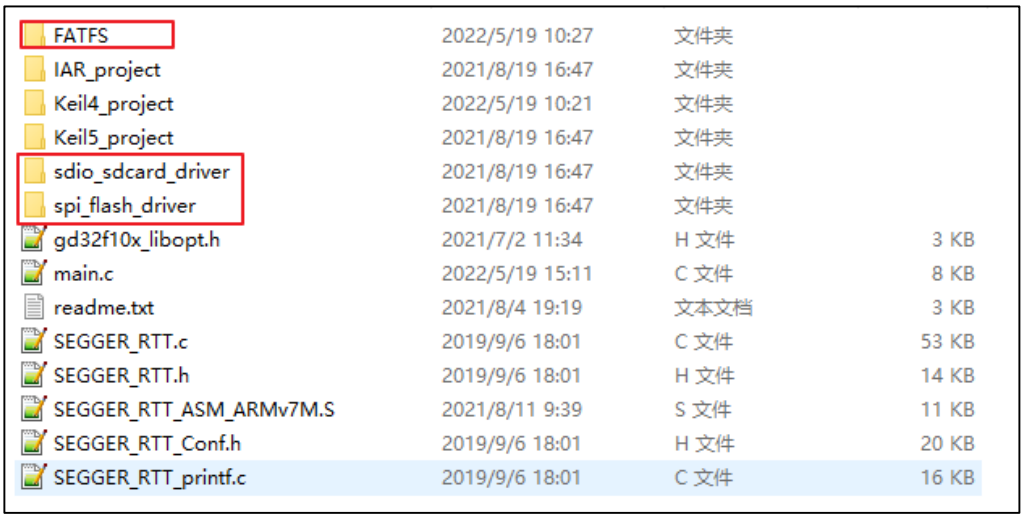
图 2-1. FATFS 包文件结构



2.2. 添加 FATFS 源代码

将工程文件夹命名为 SDIO_FATFS,并在该文件夹下添加已下载解压完成的 FATFS 包，并添加 SD 卡驱动文件和 SPI_flash 驱动文件，为了方便移植可以基于 Demo_Suites 中 16_SDIO_SDCardTest 例程进行修改。工项目添加上述文件之后，文件夹内容如*图 2-2. 项目文件*所示。

图 2-2. 项目文件



2.3. 修改 ffconf.h 文件

打开 FATFS 文件夹下的 ffconf.h 文件，在此需要修改以下部分：

1、 使能文件系统挂载函数 f_mkfs()，将编译宏_USE_MKFS 的值定义为 1；

2、 修改支持硬件驱动的数量 3，本次移植支持三种存储介质（SD 卡、SPI_Flash、片上 Flash），将编译宏_VOLUMES 的值定义为 3；

3、 定义存储介质块大小的范围，根据本次移植所用到的存储介质，将编译宏_MIN_SS 的值定义为 512，将编译宏_MAX_SS 的值定义为 4096；

2.4. 修改 diskio.c 文件

同样打开 diskio.c 文件，进行底层驱动编写，其中 disk_initialize 用于存储介质的初始化；disk_status 用于获取存储介质的工作状态；disk_read 用于存储介质的读操作；disk_write 用于存储介质的写操作；disk_ioctl 用于获取存储介质的块大小和块数量；get_fattime 用于获取文件系统时间戳，此次移植未用到此功能，不做修改。分别将 SD 卡、SPI_Flash 和片上 Flash 的驱动接口代码整合到 diskio.c 中，代码修改之后如*表 2-1. diskio.c 代码*所示

表 2-1. diskio.c 代码

```
#include "diskio.h"        /* FatFs lower layer API */
#include "spi_flash.h"
#include "sdcard.h"

#define   SD_CARD          0
#define   SPI_FLASH        1
#define   INTER_FLASH      2
#define   FLASH_SECTOR_COUNT   512        /*SPI_Flash SECTOR number*/
```

```
#define   FLASH_SECTOR_SIZE     4096        /*SPI_Flash SECTOR size*/
#define   FLASH_BLOCK_SIZE      1           /*smallest unit of erased sector*/
#define   SD_CARD_BLOCK_SIZE    1
#define   FMC_WRITE_START_ADDR      ((uint32_t)0x08000000U)
extern   sd_card_info_struct   sd_cardinfo;
/*-----------------------------------------------------------------------*/
/* Get Drive Status                                                      */
/*-----------------------------------------------------------------------*/

DSTATUS disk_status(
    BYTE pdrv           /* Physical drive nmuber to identify the drive */
)
{
    DSTATUS stat;

    switch(pdrv) {
    case SD_CARD :
        return 0;
    case SPI_FLASH :
        if(spi_flash_ID_read() == 0xC84015) {
            stat =   0; //initialization normal
        } else {
            stat = STA_NOINIT; //initialize not normal
        }
        return stat;
    case INTER_FLASH:
        stat =   0;
        return stat;
    }
    return STA_NOINIT;
}

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

DSTATUS disk_initialize(
    BYTE pdrv/* Physical drive nmuber to identify the drive */
)
{
    DSTATUS stat;

    switch(pdrv) {
```

```
    case SD_CARD:
        stat &= ~STA_NOINIT;
        return 0;
    case SPI_FLASH :
        spi_flash_config();
        return   disk_status(SPI_FLASH);
    case INTER_FLASH:
        stat =   0;
        return stat;
    }
    return STA_NOINIT;
}

/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/

DRESULT disk_read(
    BYTE pdrv,          /* Physical drive nmuber to identify the drive */
    BYTE *buff,         /* Data buffer to store read data */
    DWORD sector,    /* Sector address in LBA */
    UINT count          /* Number of sectors to read */
)
{
    uint32_t *ptrd, *btrd;
    DRESULT res;
    sd_error_enum   SD_stat = SD_OK;
    switch(pdrv) {
    case SD_CARD :
        if(count > 1) {
            SD_stat = sd_multiblocks_read((uint32_t *)buff, sector * sd_cardinfo.card_blocksize,
sd_cardinfo.card_blocksize, count);
        } else {
            SD_stat = sd_block_read((uint32_t *)buff, sector * sd_cardinfo.card_blocksize,
sd_cardinfo.card_blocksize);
        }
        if(SD_stat == SD_OK) {
            res = RES_OK ;
        } else {
            res = RES_ERROR ;
        }
        return res;
```

```
    case SPI_FLASH :
        spi_flash_buffer_read((uint8_t *)buff, sector * FLASH_SECTOR_SIZE, count * FLASH_SECTOR_SIZE);
        res = RES_OK;

        return res;
    case INTER_FLASH:
        btrd = (uint32_t *)buff;
        for(ptrd = (uint32_t *)(FMC_WRITE_START_ADDR + (sector + 47) * 2048);ptrd < (uint32_t *)(FMC_WRITE_START_ADDR + ((sector + 47) * 2048) + (count * 2048)); ptrd++) {
            *btrd = *ptrd;
            btrd++;
        }
        res = RES_OK;
        return res;
    }
    return RES_PARERR;
}

/*-----------------------------------------------------------------------*/
/* Write Sector(s)                                                       */
/*-----------------------------------------------------------------------*/

#if _USE_WRITE
DRESULT disk_write(
    BYTE pdrv,              /* Physical drive nmuber to identify the drive */
    const BYTE *buff,       /* Data to be written */
    DWORD sector,           /* Sector address in LBA */
    UINT count              /* Number of sectors to write */
)
{
    DRESULT res;
    sd_error_enum    SD_stat = SD_OK;
    uint32_t address;
    uint32_t erase_counter;

    switch(pdrv) {
    case SD_CARD :
        if(count > 1) {
            SD_stat = sd_multiblocks_write((uint32_t *)buff, sector * sd_cardinfo.card_blocksize, sd_cardinfo.card_blocksize, count);
        } else {
            SD_stat = sd_block_write((uint32_t *)buff, sector * sd_cardinfo.card_blocksize,
```

```
sd_cardinfo.card_blocksize);
            }
            if(SD_stat == SD_OK) {
                res = RES_OK ;
            } else {
                res = RES_ERROR ;
            }
            return res;

        case SPI_FLASH:
            /*first erase then write*/
            spi_flash_sector_erase(sector * FLASH_SECTOR_SIZE);
            spi_flash_buffer_write((uint8_t *)buff, sector * FLASH_SECTOR_SIZE, count * FLASH_SECTOR_SIZE);
            res = RES_OK;
            return res;
        case INTER_FLASH:
            fmc_unlock();
            fmc_flag_clear(FMC_FLAG_BANK0_END);
            fmc_flag_clear(FMC_FLAG_BANK0_WPERR);
            fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
            /* erase the flash pages */
            for(erase_counter = 0; erase_counter < count; erase_counter++) {
                fmc_page_erase(FMC_WRITE_START_ADDR + ((sector + 47) * 2048) + (2048 * erase_counter));
                fmc_flag_clear(FMC_FLAG_BANK0_END);
                fmc_flag_clear(FMC_FLAG_BANK0_WPERR);
                fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
            }
            address = (sector + 47) * 2048 + FMC_WRITE_START_ADDR;
            while(address < (((sector + 47) * 2048 + FMC_WRITE_START_ADDR) + count * 2048)) {
                fmc_word_program(address, *(uint32_t *)buff);
                address += 4;
                buff += 4;
                fmc_flag_clear(FMC_FLAG_BANK0_END);
                fmc_flag_clear(FMC_FLAG_BANK0_WPERR);
                fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
            }
            fmc_lock();
            res = RES_OK;
            return res;
        }
        return RES_PARERR;
```

```
}
#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;

    switch(pdrv) {
    case SD_CARD :
        switch(cmd) {
        /*return sector number*/
        case GET_SECTOR_COUNT:
            *(DWORD *)buff = sd_cardinfo.card_capacity / (sd_cardinfo.card_blocksize);
            break;
        /*return each sector size*/
        case GET_SECTOR_SIZE:
            *(WORD *)buff = sd_cardinfo.card_blocksize;
            break;
        /*Returns the smallest unit of erased sector (unit 1)*/
        case GET_BLOCK_SIZE:
            *(DWORD *)buff = SD_CARD_BLOCK_SIZE;
            break;
        }
        res = RES_OK;
        return res;

    case SPI_FLASH :
        switch(cmd) {
        /*return sector number*/
        case GET_SECTOR_COUNT:
            *(DWORD *)buff = FLASH_SECTOR_COUNT;
            break;
        /*return each sector size*/
```

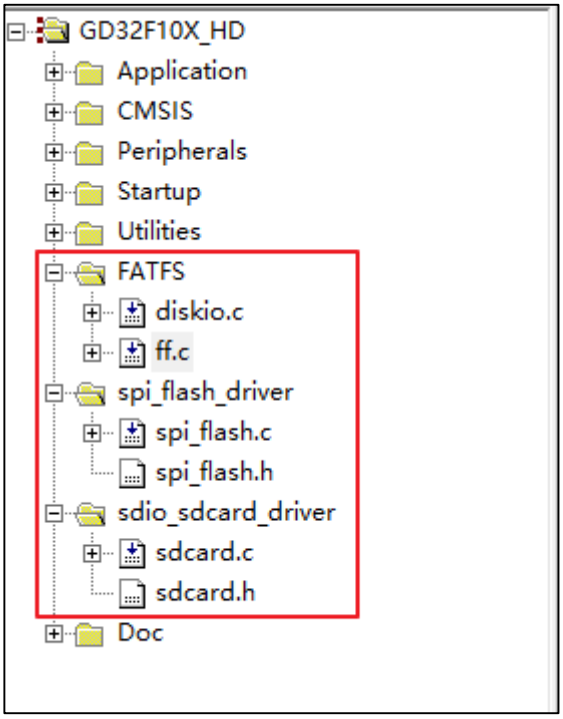
```
            case GET_SECTOR_SIZE:
                *(WORD *)buff = FLASH_SECTOR_SIZE;
                break;
            /*Returns the smallest unit of erased sector (unit 1)*/
            case GET_BLOCK_SIZE:
                *(DWORD *)buff = FLASH_BLOCK_SIZE;
                break;
            }
            res = RES_OK;
            return res;

        case INTER_FLASH:
            switch(cmd) {
            /*return sector number*/
            case GET_SECTOR_COUNT:
                *(DWORD *)buff = 128;
                break;
            /*return each sector size*/
            case GET_SECTOR_SIZE:
                *(WORD *)buff = 2048;
                break;
            /*Returns the smallest unit of erased sector (unit 1)*/
            case GET_BLOCK_SIZE:
                *(DWORD *)buff = 1;
                break;
            }
            res = RES_OK;
            return res;

        }

        return RES_PARERR;
}
#endif

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

2.5. 添加工程代码

在 keil4 中添加各存储介质的驱动程序，并添加 FATFS 代码，添加完成后工程目录如*图 2-3.*

*工程目录*所示。

图 2-3. 工程目录



3. FATFS 文件系统测试

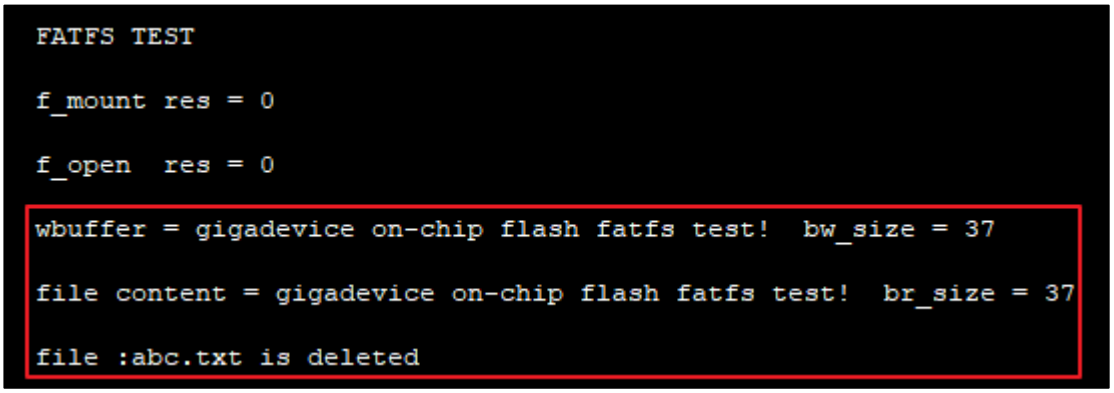
1、 利用 FATFS 文件系统对片上 Flash 进行文件的增删读写测试，并利用 J-Link RTT Viewer 打印结果，测试代码如下：

```
void on-chip_flash_fatfs_test(void)
{
    FRESULT res;
    SEGGER_RTT_printf(0, "\r\n FATFS TEST \r\n");
    res = f_mount(&fsObject, "2:", 1);
    SEGGER_RTT_printf(0, "\r\n f_mount res = %d \r\n", res);
    if(res == FR_NO_FILESYSTEM) {
        /*creates an FAT volume on on- chip FLASH(format)*/
        res = f_mkfs("2:", 0, 0);
        SEGGER_RTT_printf(0, "\r\n f_mkfs res = %d \r\n", res);
        /*unmount file system*/
        res = f_mount(NULL, "2:", 1);
        /*mount file system*/
        res = f_mount(&fsObject, "2:", 1);
        SEGGER_RTT_printf(0, "\r\n f_mkfs 2 res = %d \r\n", res);
    }
    /*create a file enabile write and read*/
    res = f_open(&fp, "2:abc.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ);

    SEGGER_RTT_printf(0, "\r\n f_open   res = %d \r\n", res);
    if(res == FR_OK) {
        /*write data into a file*/
        res = f_write(&fp, wbuffer1, sizeof(wbuffer1), &bw_size);
        SEGGER_RTT_printf(0, "\r\n wbuffer = %s   bw_size = %d\r\n", wbuffer1, bw_size);
        if(res == FR_OK) {
            f_lseek(&fp, 0);
            /*read data from a file*/
            f_read(&fp, rbuffer, f_size(&fp), &br_size);
            if(res == FR_OK) {
                SEGGER_RTT_printf(0, "\r\n file content = %s   br_size = %d\r\n", rbuffer,
br_size);
            }
        }
        f_close(&fp);
        res = f_unlink("2:abc.txt");
        res = f_open(&fp, "2:abc.txt", FA_READ);
        if(res != FR_OK) {
            SEGGER_RTT_printf(0, "\r\n file :abc.txt is deleted \r\n");
```

```
        }
    }
}
```

测试结果如*图 3-1. FATFS 片上 Flash 文件增删读写结果*所示，说明 FATFS 文件系统成功实现在片上 Flash 增删读写 abc.txt 文件。

图 3-1. FATFS 片上 Flash 文件增删读写结果



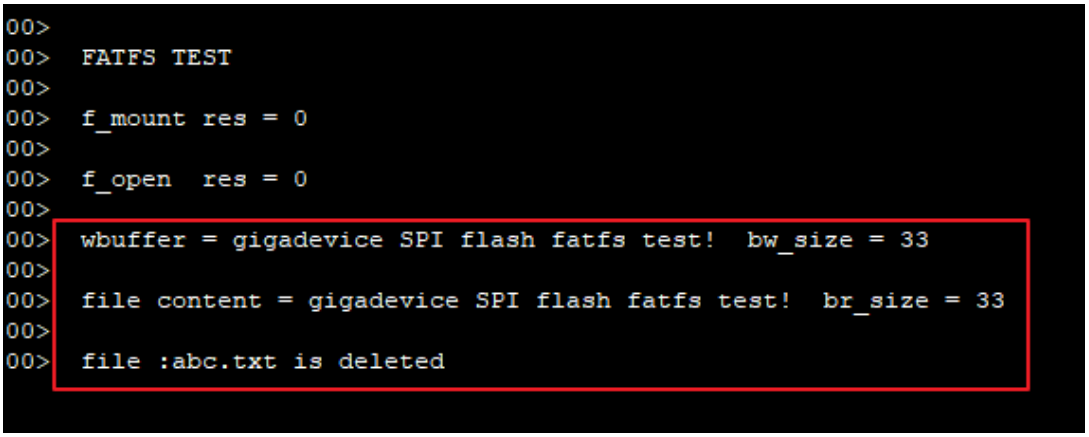
2、 利用 FATFS 文件系统对 SPI_Flash 进行文件的增删读写测试，并利用 J-Link RTT Viewer 打印结果，测试代码只需将片上 Flash 测试代码中的盘符修改为 SPI_Flash 即可测试，测试代码如下：

```
void SPI_Flash_fatfs_test(void)
{
    FRESULT res;
    SEGGER_RTT_printf(0, "\r\n FATFS TEST \r\n");
    res = f_mount(&fsObject, "1:", 1);
    SEGGER_RTT_printf(0, "\r\n f_mount res = %d \r\n", res);
    if(res == FR_NO_FILESYSTEM) {
        /*creates an FAT volume on SPI FLASH(format)*/
        res = f_mkfs("1:", 0, 0);
        SEGGER_RTT_printf(0, "\r\n f_mkfs res = %d \r\n", res);
        /*unmount file system*/
        res = f_mount(NULL, "1:", 1);
        /*mount file system*/
        res = f_mount(&fsObject, "1:", 1);
        SEGGER_RTT_printf(0, "\r\n f_mkfs 2 res = %d \r\n", res);
    }
    /*create a file enabile write and read*/
    res = f_open(&fp, "1:abc.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ);
    SEGGER_RTT_printf(0, "\r\n f_open    res = %d \r\n", res);
    if(res == FR_OK) {
        /*write data into a file*/
        res = f_write(&fp, wbuffer1, sizeof(wbuffer1), &bw_size);
        SEGGER_RTT_printf(0, "\r\n wbuffer = %s    bw_size = %d\r\n", wbuffer1, bw_size);
```

```
        if(res == FR_OK) {
            f_lseek(&fp, 0);
            /*read data from a file*/
            f_read(&fp, rbuffer, f_size(&fp), &br_size);
            if(res == FR_OK) {
                SEGGER_RTT_printf(0, "\r\n file content = %s   br_size = %d\r\n", rbuffer,
br_size);
            }
        }
        f_close(&fp);
        res = f_unlink("1:abc.txt");
        res = f_open(&fp, "1:abc.txt", FA_READ);
        if(res != FR_OK) {
            SEGGER_RTT_printf(0, "\r\n file :abc.txt is deleted \r\n");
        }
    }
}
```

测试结果如**图 3-2.** ***FATFS SPI_Flash 文件增删读写结果***所示，文件增删读写成功。

图 3-2. FATFS SPI_Flash 文件增删读写结果



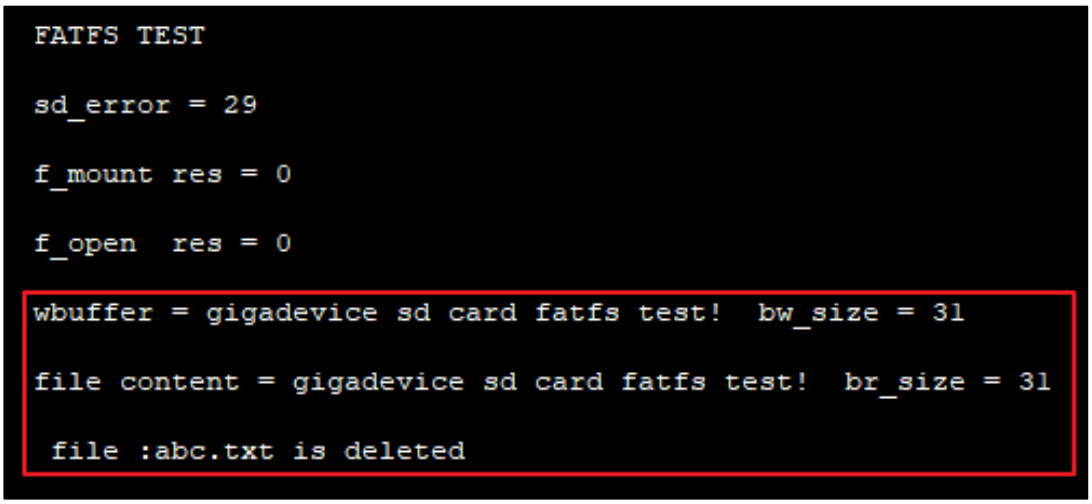
3、 利用 FATFS 文件系统对 SPI_Flash 进行文件的增删读写测试，并利用 J-Link RTT Viewer 打印结果,测试代码如下：

```
void sd_card_fatfs_test(void)
{
    sd_error_enum sd_error;
    uint16_t i = 5;
    FRESULT res;
    SEGGER_RTT_printf(0, "\r\n FATFS TEST \r\n");
    /* initialize SD card*/
    do {
        sd_error = sd_io_init();
    } while((SD_OK != sd_error) && (--i));
```

```
    if(sd_error == SD_OK) {
        SEGGER_RTT_printf(0, "\r\n sd_error = %d\r\n", sd_error);
    }
    /* registers/unregisters file system object to the FatFs module*/
    res = f_mount(&fsObject, "0:", 1);
    SEGGER_RTT_printf(0, "\r\n f_mount res = %d \r\n", res);
    if(res == FR_NO_FILESYSTEM) {
        /*creates an FAT volume on SD card(format)*/
        res = f_mkfs("0:", 0, 512);
        SEGGER_RTT_printf(0, "\r\n f_mkfs res = %d \r\n", res);
        /*unmount file system*/
        res = f_mount(NULL, "0:", 1);
        /*mount file system*/
        res = f_mount(&fsObject, "0:", 1);
        SEGGER_RTT_printf(0, "\r\n f_mkfs 2 res = %d \r\n", res);
    }
    /*create a file enabile write and read*/
    res = f_open(&fp, "0:abc.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ);
    SEGGER_RTT_printf(0, "\r\n f_open    res = %d \r\n", res);
    if(res == FR_OK) {
        /*write data into a file*/
        res = f_write(&fp, wbuffer, sizeof(wbuffer), &bw_size);
        SEGGER_RTT_printf(0, "\r\n wbuffer = %s    bw_size = %d\r\n", wbuffer, bw_size);
        if(res == FR_OK) {
            f_lseek(&fp, 0);
            /*read data from a file*/
            f_read(&fp, rbuffer, f_size(&fp), &br_size);
            if(res == FR_OK) {
                SEGGER_RTT_printf(0, "\r\n file content = %s    br_size = %d\r\n", rbuffer,
br_size);
            }
        }
        f_close(&fp);
        res = f_unlink("0:abc.txt");
        if(res == FR_OK) {
            SEGGER_RTT_printf(0, "\r\n    file :abc.txt is deleted \r\n");
        }
    }
}
```

测试结果如*图 3-3. FATFS SD 卡文件增删读写结果*所示，文件增删读写成功。

图 3-3. FATFS SD 卡文件增删读写结果



4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2023 年 8 月 11 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.