

32 位微控制器

HC32F4A0 系列的基于 SDIO 的 FatFS 文件系统移植

适用对象

F 系列	HC32F4A0
------	----------

目 录

1	摘要	3
2	FatFS 文件系统	3
2.1	FatFS 目录结构.....	4
2.2	FatFS 源码.....	5
3	SDIO 控制器	7
3.1	SDIO 总线	7
4	FatFS 移植	8
4.1	FatFS 系统组织.....	8
4.2	FatFS 移植.....	9
4.3	FatFS 底层 API 实现	11
4.3.1	创建 sd_diskio 驱动文件并添加到工程	12
4.3.2	编辑 sd_diskio.h 文件	13
4.3.3	编辑 sd_diskio.c 文件.....	14
4.3.4	修改 diskio.c 文件.....	20
4.4	FatFS 功能配置.....	23
4.5	FatFS 测试验证.....	24
4.5.1	修改 main.c 文件	24
4.5.2	验证功能	28
5	版本信息 & 联系方式	29

1 摘要

本篇应用笔记主要介绍基于 HC32F4A0 系列 SDIO 控制器模块移植 FatFs 文件系统，并演示 FatFs 文件系统的文件读写功能。

2 FatFS 文件系统

FatFs 是用于小型嵌入式系统的通用 FAT/exFAT 文件系统。代码编写符合 ANSI C (C89)，并且完全独立于底层的 I/O 介质。因此可以快速地移植到其他的处理器中，如 8051，PIC，AVR，ARM，Z80，RX 等。FatFs 支持 FAT12、FAT16、FAT32 等格式。

FatFs 文件系统的源码可以从 fatfs 官网下载：

http://elm-chan.org/fsw/ff/00index_e.html

2.1 FatFS 目录结构

在移植 FatFs 文件系统前，先到 FatFs 的官网获取源码，样例使用的版本为 R0.14，官网对 FatFs 做了详细的介绍，下载并解压后可看到里面有 documents 和 source 这两个文件夹，见图 1。

documents 文件夹：帮助文档

source 文件夹：FatFs 文件系统的源码

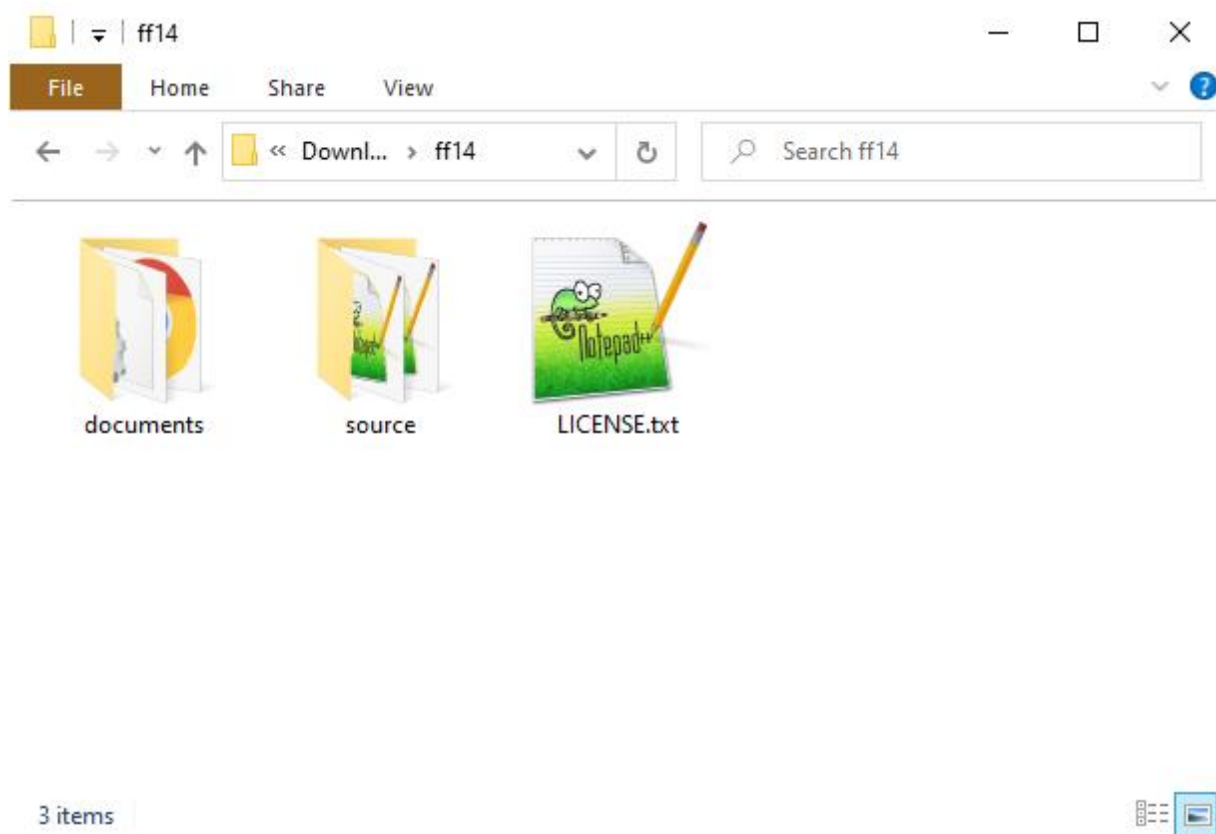


图 1 FatFs 文件目录

2.2 FatFS 源码

打开 source 文件夹，可看到如下图的文件目录：

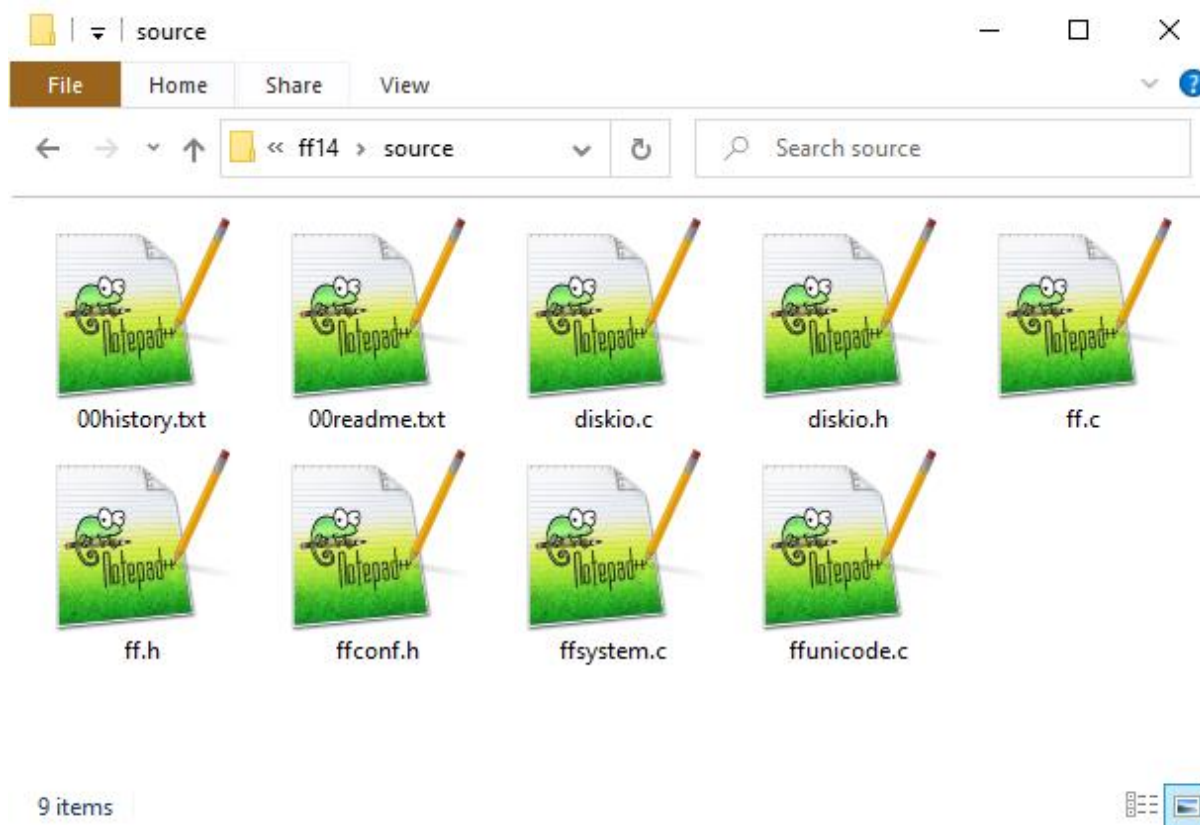


图 2 source 文件夹的文件目录

其中，diskio.c 文件是 FatFs 移植最关键的文件，它为文件系统提供了最底层的访问 I/O 介质（如：SD 卡、MMC 卡等）的方法，FatFs 有且仅有它需要用到 I/O 介质相关的函数。

diskio.h 定义了 FatFs 用到的宏，以及 diskio.c 文件内与底层硬件接口相关的函数声明。

source 文件夹下的文件功能简介如下：

- 00readme.txt: 介绍 source 目录下每个文件的功能。
- 00history.txt: 介绍 FatFs 的版本更新信息。
- ff.c: FatFs 模块的核心文件，文件管理的实现方法。
- ffconf.h: FatFs 模块的配置文件，通过修改文件中的宏定义就可以裁剪 FatFs 的功能。
- ff.h: FatFs 模块应用的头文件，包含 FatFs 的数据结构、宏定义和函数接口。

- **diskio.h:** FatFs 模块 I/O 介质的头文件，包含底层硬件接口的函数声明及相关宏定义。
- **diskio.c:** FatFs 模块 I/O 介质的操作函数，需要用户根据 I/O 介质在这些函数中添加相应的底层驱动程序。
- **ffunicode.c:** 可选的 Unicode 实用函数，如果配置 ffconf.h 中的宏定义 FF_USE_LFN 大于 0，则需要把此文件添加到工程中。
- **ffsystem.c:** 可选的 O/S 相关的函数，如果需要支持操作系统，则需要把此文件添加到工程，并根据 O/S 重新实现文件中的函数接口。

3 SDIO 控制器

SDIO 全称是安全数字输入/输出接口，SDIO 控制器提供了一个 SD 主机接口和一个 MMC 主机接口，用于和支持 SD2.0 协议的 SD 卡，SDIO 设备以及支持 eMMC4.2 协议的 MMC 设备进行通信。

3.1 SDIO 总线

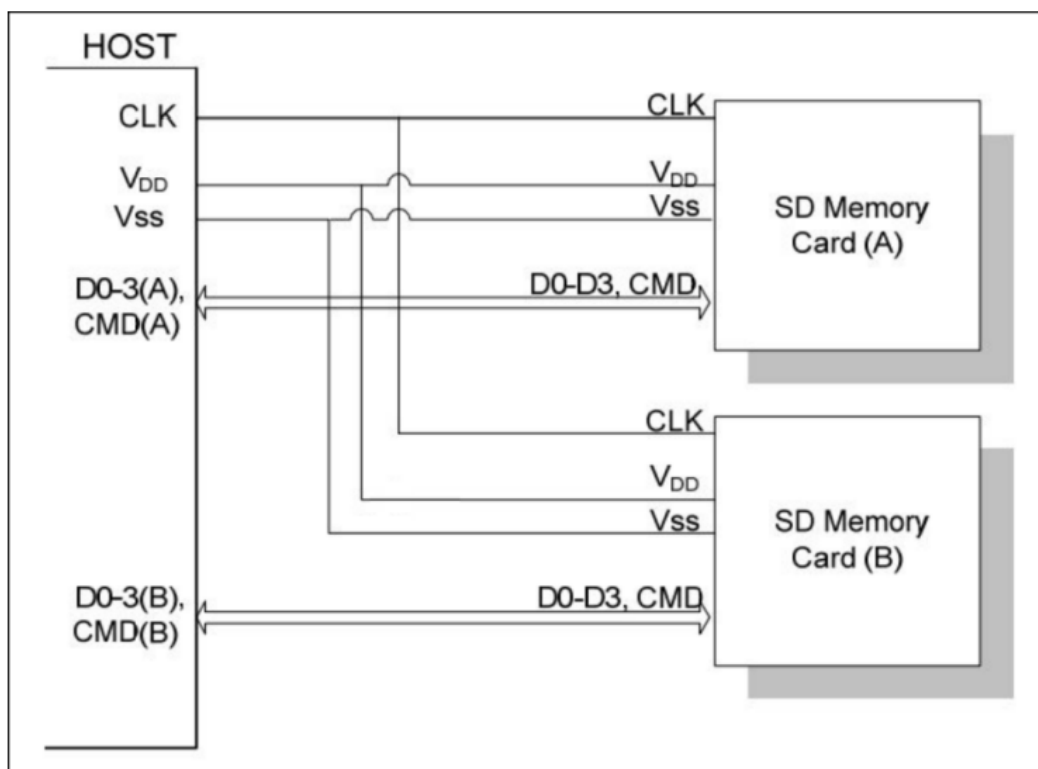


图 3 SD 卡总线拓扑参考图

SD 卡使用 9-pin 接口通信，其中 3 根电源线、1 根时钟线、1 根命令线和 4 根数据线，具体说明如下：

- CLK：时钟线，由 SDIO 主机产生，即由 SDIO 控制器输出
- CMD：命令线，SDIO 主机通过该线发送命令控制 SD 卡，如果命令要求 SD 卡提供应答，SD 卡也通过该线传输应答信息
- D0-3：数据线，传输读写数据；SD 卡可将 D0 拉低表示忙状态
- VDD、VSS1、VSS2：电源和地信号

SD 卡操作过程会使用两种不同频率的时钟同步数据，一个是识别卡阶段时钟频率 FOD，最高为 400kHz，另外一个为数据传输模式下时钟频率 FPP，默认最高为 25MHz。

4 FatFS 移植

4.1 FatFS 系统组织

移植 FatFs 之前先通过 FatFs 的典型配置了解 FatFs 在程序中的关系网络，见下图。

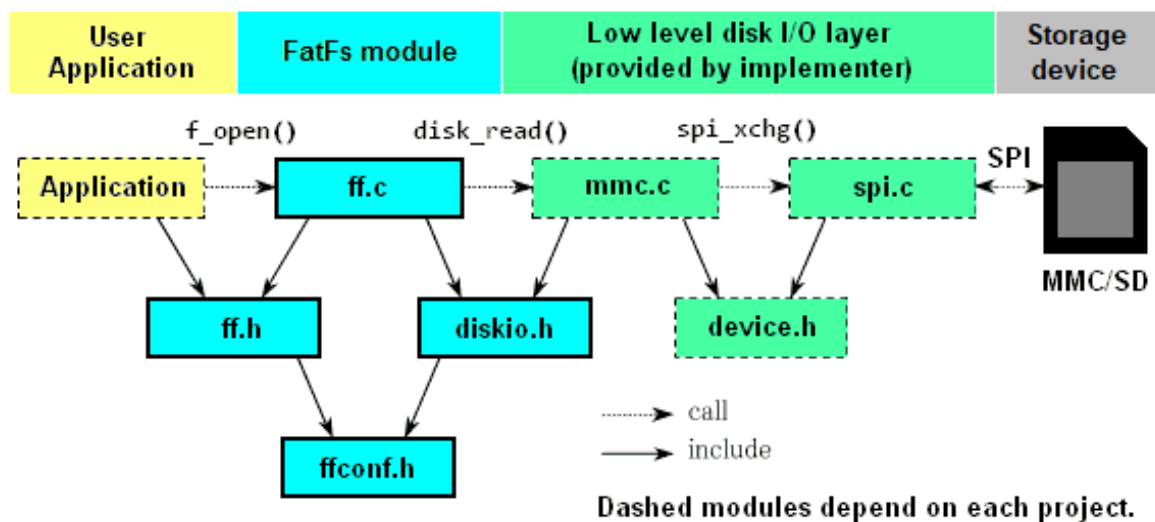


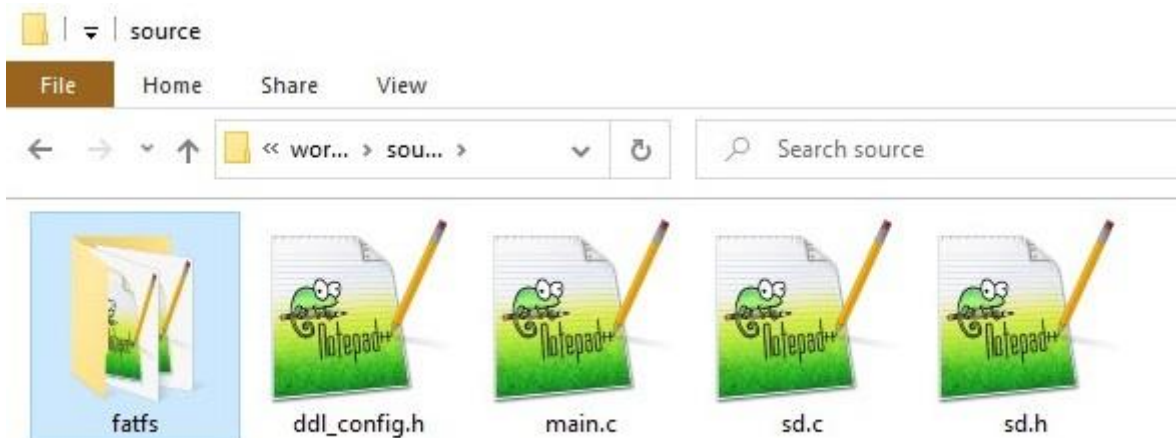
图 4 典型配置

FatFs module 是 FatFs 的主体，文件都在 source 文件夹中，其中 ff.c、ff.h 和 diskio.h 三个文件不需要改动，只需要修改 ffconf.h 和 diskio.c 两个文件。

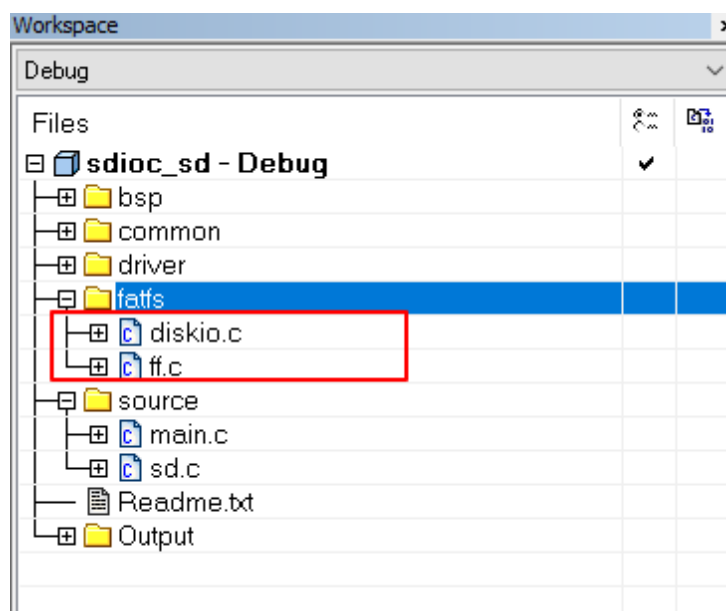
4.2 FatFS 移植

打开 DDL 中 “..\example\sdioc\sdioc_sd” 样例，在此工程基础上添加 FatFs 组件，并修改 main 函数的用户程序即可。

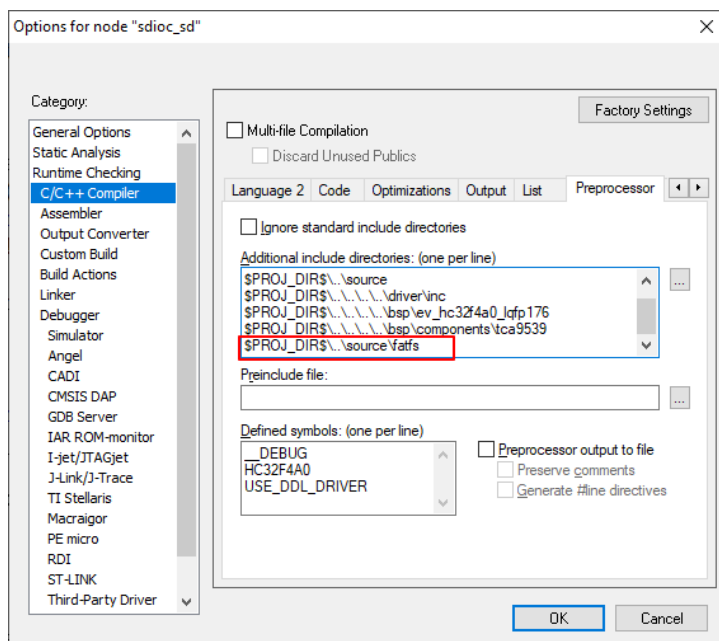
- 1) 将 FatFs 源码中的 source 文件夹整个文件夹复制到 “..\sdioc_sd\source” 文件夹下并修改文件夹名为 “fatfs”，如下图。



- 2) 使用 IAR 软件打开工程文件(.\sdioc_sd\EWARM\sdioc_sd.eww)，并将 FatFs 组件文件添加到工程中，需要添加有 ff.c 和 diskio.c 两个文件，如下图。



3) 添加 FatFs 文件夹到工程的 include 选项中，如下图。



4) 至此，移植 FatFs 模块的框架已经完成，接下来就需要修改 diskio.c 文件和 ffconf.h 文件。

4.3 FatFS 底层 API 实现

FatFs 文件系统与底层 I/O 介质的驱动是分离开的，对底层 I/O 介质的操作都交给用户去实现，它仅仅提供了一个函数接口。

下表是 FatFs 移植时用户必须支持的函数。

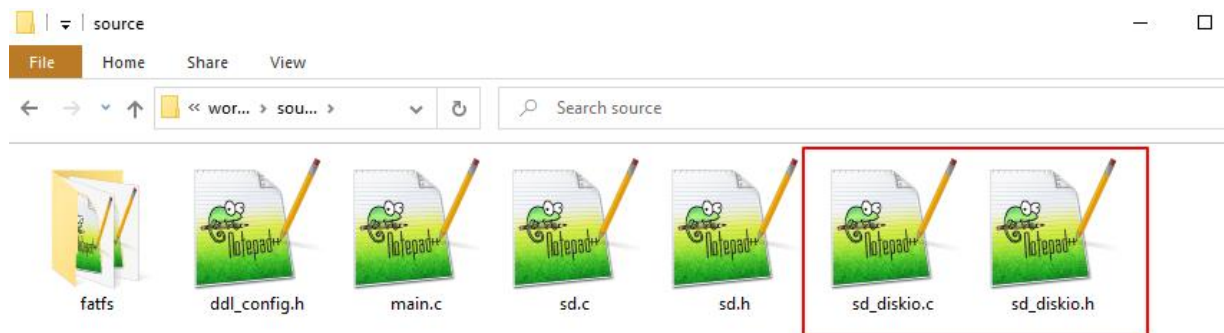
Function	Required when	Note
disk_status disk_initialize disk_read	Always	Disk I/O functions. Samples available in <code>ffsample.zip</code> . There are many implementations on the web.
disk_write get_fattime disk_ioctl (CTRL_SYNC)	FF_FS_READONLY == 0	
disk_ioctl (GET_SECTOR_COUNT) disk_ioctl (GET_BLOCK_SIZE)	FF_USE_MKFS == 1	
disk_ioctl (GET_SECTOR_SIZE)	FF_MAX_SS != FF_MIN_SS	
disk_ioctl (CTRL_TRIM)	FF_USE_TRIM == 1	
ff_uni2oem ff_oem2uni ff_wtoupper	FF_USE_LFN != 0	
ff_cre_syncobj ff_del_syncobj ff_req_grant ff_rel_grant	FF_FS_REENTRANT == 1	Unicode support functions. Add optional module <code>ffunicode.c</code> to the project.
ff_mem_alloc ff_mem_free	FF_USE_LFN == 3	
		O/S dependent functions. Sample code is available in <code>ffsystem.c</code> .

通过表中的信息可以知道很多函数是在一定条件下才需要添加的，只有前三个函数是必须添加的。因此，用户完全可以根据实际需求选择实现用到的函数，一般只要实现前面六个函数就可以满足大部分功能。

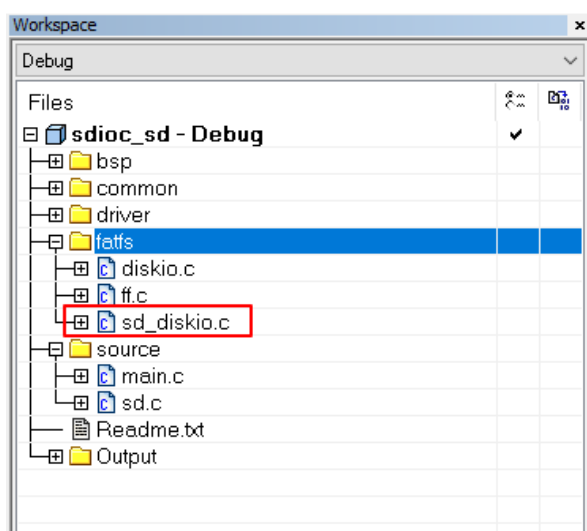
在 `diskio.c` 文件中已经把 `disk_initialize`、`disk_status`、`disk_read`、`disk_write`、`disk_ioctl` 这五个函数编写出来，只需要根据 I/O 介质在其中添加对应的驱动程序，并在文件中添加 `get_fattime` 函数，就可以让 FatFs 和 I/O 介质关联起来，让 I/O 介质受 FatFs 管理。

4.3.1 创建 sd_diskio 驱动文件并添加到工程

- 1) 因样例演示的是使用 SD 卡来作为 I/O 介质，所以需要在 “..\sdioc_sd\source” 文件夹下创建 sd_diskio.c 和 sd_diskio.h 文件，如下图；在 diskio.c 中调用 sd_diskio 中的底层驱动接口。



- 2) 添加 sd_diskio.c 到工程中，如下图：



4.3.2 编辑 sd_diskio.h 文件

文件中程序如下：

```
#ifndef __SD_DISKIO__
#define __SD_DISKIO__

/* C binding of definitions if building with C++ compiler */
#ifdef __cplusplus
extern "C"
{
#endif

/* Include files */
#include "hc32_common.h"
#include "ff.h"
#include "diskio.h"

/* Global function prototypes (definition in C source) */
DSTATUS SD_Status (BYTE);
DSTATUS SD_Initialize (BYTE);
DRESULT SD_Read (BYTE, BYTE*, DWORD, UINT);
DRESULT SD_Write (BYTE, const BYTE*, DWORD, UINT);
DRESULT SD_Ioctl (BYTE, BYTE, void*);

#ifdef __cplusplus
}
#endif

#endif /* __SD_DISKIO__ */
```

4.3.3 编辑 sd_diskio.c 文件

文件中程序如下：

```
#include "sd.h"
#include "sd_diskio.h"

/* Local pre-processor symbols/macros ('#define') */
#define SDIOC_SD_UINT          (M4_SDIOC1)
#define SDIOC_SD_CLK          (PWC_FCG1_SDIOC1)

/* CK = PC12 */
#define SDIOC_CK_PORT          (GPIO_PORT_C)
#define SDIOC_CK_PIN           (GPIO_PIN_12)
/* CMD = PD02 */
#define SDIOC_CMD_PORT         (GPIO_PORT_D)
#define SDIOC_CMD_PIN          (GPIO_PIN_02)
/* D0 = PB07 */
#define SDIOC_D0_PORT          (GPIO_PORT_B)
#define SDIOC_D0_PIN           (GPIO_PIN_07)
/* D1 = PA08 */
#define SDIOC_D1_PORT          (GPIO_PORT_A)
#define SDIOC_D1_PIN           (GPIO_PIN_08)
/* D2 = PC10 */
#define SDIOC_D2_PORT          (GPIO_PORT_C)
#define SDIOC_D2_PIN           (GPIO_PIN_10)
/* D3 = PB05 */
#define SDIOC_D3_PORT          (GPIO_PORT_B)
#define SDIOC_D3_PIN           (GPIO_PIN_05)

/* Block size is 512 bytes */
#define SD_DEFAULT_BLOCK_SIZE  (512U)
/* SD read/write timeout time */
#define SD_RW_TIMEOUT_TIME      (30000UL)

/* Local variable definitions ('static') */
static stc_sd_handle_t SdHandle;
static volatile DSTATUS SdStat = (DSTATUS)STA_NOINIT;

/**
 * @brief Initializes the SD GPIO.
 * @param None
 * @retval None
 */
static void SDCard_GpioInit(void)
{
    /* SDIO1_CD */
    BSP_IO_ConfigPortPin(EIO_PORT0, EIO_SDIC1_CD, EIO_DIR_IN);

    /* SDIOC pins configuration */
    GPIO_SetFunc(SDIOC_CK_PORT, SDIOC_CK_PIN, GPIO_FUNC_9_SDIO1_CK,
PIN_SUBFUNC_DISABLE);
```

```

    GPIO_SetFunc(SDIOC_CMD_PORT, SDIOC_CMD_PIN, GPIO_FUNC_9_SDIO1_CMD,
PIN_SUBFUNC_DISABLE);
    GPIO_SetFunc(SDIOC_D0_PORT, SDIOC_D0_PIN, GPIO_FUNC_9_SDIO1_DATA,
PIN_SUBFUNC_DISABLE);
    GPIO_SetFunc(SDIOC_D1_PORT, SDIOC_D1_PIN, GPIO_FUNC_9_SDIO1_DATA,
PIN_SUBFUNC_DISABLE);
    GPIO_SetFunc(SDIOC_D2_PORT, SDIOC_D2_PIN, GPIO_FUNC_9_SDIO1_DATA,
PIN_SUBFUNC_DISABLE);
    GPIO_SetFunc(SDIOC_D3_PORT, SDIOC_D3_PIN, GPIO_FUNC_9_SDIO1_DATA,
PIN_SUBFUNC_DISABLE);
}

/**
 * @brief Get SD card insert status.
 * @param None
 * @retval An en_flag_status_t enumeration value:
 *      - Set: SD card inserted
 *      - Reset: No SD card insert
 */
static en_flag_status_t SDCard_GetInsertStatus(void)
{
    en_flag_status_t enFlagSta = Set;

    /* Check SD card detect pin */
    if (0U != BSP_IO_ReadPortPin(EIO_PORT0, EIO_SDIC1_CD))
    {
        enFlagSta = Reset;
    }

    return enFlagSta;
}

/**
 * @brief SD card configuration.
 * @param None
 * @retval An en_result_t enumeration value:
 *      - Ok: Configuration success
 *      - Error: Configuration failed
 */
static en_result_t SDCard_Config(void)
{
    en_result_t enRet = Error;

    /* Init SD GPIO */
    SDCard_GpioInit();
    /* Enable SDIOC clock */
    PWC_Fcg1PeriphClockCmd(SDIOC_SD_CLK, Enable);

    /* Configure structure initialization */
    SdHandle.SDIOCx = SDIOC_SD_UINT;
    SdHandle.stcSdiocInit.u32Mode = SDIOC_MODE_SD;
    SdHandle.stcSdiocInit.u8CardDetectSelect = SDIOC_CARD_DETECT_CD_PIN_LEVEL;
    SdHandle.stcSdiocInit.u8SpeedMode = SDIOC_SPEED_MODE_HIGH;
    SdHandle.stcSdiocInit.u8BusWidth = SDIOC_BUS_WIDTH_4BIT;
    SdHandle.stcSdiocInit.u16ClockDiv = SDIOC_CLOCK_DIV_2;

```

```
SdHandle.DMAx = NULL;

/* Reset SDIOC */
if (Ok != SDIOC_SoftwareReset(SdHandle.SDIOCx, SDIOC_SW_RESET_ALL))
{
    (void)printf("Reset SDIOC failed!\r\n");
}
else
{
    if (Set != SDCard_GetInsertStatus())
    {
        (void)printf("No SD card insertion!\r\n");
    }
    else
    {
        if (Ok != SD_Init(&SdHandle))
        {
            (void)printf("SD card initialize failed!\r\n");
        }
        else
        {
            enRet = Ok;
        }
    }
}

return enRet;
}

/**
 * @brief Get SD card state.
 * @param None
 * @retval An en_result_t enumeration value:
 *         - Ok: Data transfer is acting
 *         - Error: No data transfer is acting
 */
static en_result_t SDCard_GetCardTransState(void)
{
    en_result_t enRet;
    en_sd_card_state_t enCardState;

    enRet = SD_GetCardState(&SdHandle, &enCardState);
    if (Ok == enRet)
    {
        if (SDCardStateTransfer == enCardState)
        {
            enRet = Ok;
        }
    }

    return enRet;
}

/**
 * @brief Check the SD card status.
```



```

* @param lun: Not used
* @retval DSTATUS: Operation status
*/
static DSTATUS SD_CheckStatus(BYTE lun)
{
    SdStat = (DSTATUS)STA_NOINIT;
    en_result_t enRet;

    enRet = SDCard_GetCardTransState();
    if (Ok == enRet)
    {
        SdStat &= (DSTATUS)(~(DSTATUS)STA_NOINIT);
    }

    return SdStat;
}

/**
* @brief Gets Disk Status
* @param lun: Not used
* @retval DSTATUS: Operation status
*/
DSTATUS SD_Status(BYTE lun)
{
    return SD_CheckStatus(lun);
}

/**
* @brief Initializes a Drive
* @param lun: Not used
* @retval DSTATUS: Operation status
*/
DSTATUS SD_Initialize(BYTE lun)
{
    SdStat = (DSTATUS)STA_NOINIT;

    if (Ok == SDCard_Config())
    {
        SdStat = SD_CheckStatus(lun);
    }

    return SdStat;
}

/**
* @brief Reads Sector(s)
* @param lun: Not used
* @param buff: Pointer to data buffer used to store read data
* @param sector: Sector address (LBA)
* @param count: Number of sectors to read (1..128)
* @retval DRESULT: Operation result
*/
DRESULT SD_Read(BYTE lun, BYTE *buff, DWORD sector, UINT count)
{
    DRESULT res = RES_ERROR;

```

```

    if (Ok == SD_ReadBlocks(&SdHandle, (uint32_t)sector, (uint16_t)count, (uint8_t *)buff,
SD_RW_TIMEOUT_TIME))
    {
        /* Wait until the read operation is finished */
        while (Ok != SDCard_GetCardTransState())
        {
        }
        res = RES_OK;
    }

    return res;
}

/**
 * @brief Writes Sector(s)
 * @param lun: Not used
 * @param buff: Pointer to data to be written
 * @param sector: Sector address (LBA)
 * @param count: Number of sectors to write (1..128)
 * @retval DRESULT: Operation result
 */
DRESULT SD_Write(BYTE lun, const BYTE *buff, DWORD sector, UINT count)
{
    DRESULT res = RES_ERROR;

    if (Ok == SD_WriteBlocks(&SdHandle, (uint32_t)sector, (uint16_t)count, (uint8_t *)buff,
SD_RW_TIMEOUT_TIME))
    {
        /* Wait until the Write operation is finished */
        while (Ok != SDCard_GetCardTransState())
        {
        }
        res = RES_OK;
    }

    return res;
}

/**
 * @brief I/O control operation
 * @param lun: Not used
 * @param cmd: Control code
 * @param buff: Pointer to buffer used to send/receive data
 * @retval DRESULT: Operation result
 */
DRESULT SD_Ioctl(BYTE lun, BYTE cmd, void *buff)
{
    DRESULT res;
    stc_sd_card_info_t stcCardInfo;

    if (0U != (SdStat & (DSTATUS)STA_NOINIT))
    {
        res = RES_NOTRDY;
    }
}

```

```
else
{
    switch (cmd)
    {
        /* Make sure that no pending write process */
        case CTRL_SYNC :
            res = RES_OK;
            break;
        /* Get number of sectors on the disk (DWORD) */
        case GET_SECTOR_COUNT :
            (void)SD_GetCardInfo(&SdHandle, &stcCardInfo);
            *(DWORD*)buff = stcCardInfo.u32LogBlockNbr;
            res = RES_OK;
            break;
        /* Get R/W sector size (WORD) */
        case GET_SECTOR_SIZE :
            (void)SD_GetCardInfo(&SdHandle, &stcCardInfo);
            *(WORD*)buff = (uint16_t)stcCardInfo.u32LogBlockSize;
            res = RES_OK;
            break;
        /* Get erase block size in unit of sector (DWORD) */
        case GET_BLOCK_SIZE :
            (void)SD_GetCardInfo(&SdHandle, &stcCardInfo);
            *(DWORD*)buff = stcCardInfo.u32LogBlockSize / SD_DEFAULT_BLOCK_SIZE;
            res = RES_OK;
            break;
        default:
            res = RES_PARERR;
            break;
    }
}

return res;
}
```

4.3.4 修改 diskio.c 文件

- 1) 添加 I/O 介质驱动 “sd_diskio.h” 的头文件:

```
#include "ff.h" /* Obtains integer types */
#include "diskio.h" /* Declarations of disk functions */
#include "sd_diskio.h"
```

- 2) 修改 DSTATUS disk_status(BYTE pdrv)函数如下:

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

    stat = STA_NOINIT;
    switch (pdrv) {
        case DEV_RAM :
            // translate the reslut code here
            return stat;
        case DEV_MMC :
            stat = SD_Status(pdrv);
            return stat;
        case DEV_USB :
            // translate the reslut code here
            return stat;
    }
    return STA_NOINIT;
}
```

- 3) 修改 DSTATUS disk_initialize(BYTE pdrv)函数如下:

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

    stat = STA_NOINIT;
    switch (pdrv)
    {
        case DEV_RAM :
            // translate the reslut code here
            return stat;
        case DEV_MMC :
            stat = SD_Initialize(pdrv);
            return stat;
        case DEV_USB :
            // translate the reslut code here
            return stat;
    }
    return STA_NOINIT;
}
```

```
}
```

- 4) 修改 DRESULT disk_read(BYTE pdrv, BYTE* buff, LBA_t sector, UINT count)函数如下:

```
DRESULT disk_read (
    BYTE pdrv,      /* Physical drive nmuber to identify the drive */
    BYTE *buff,     /* Data buffer to store read data */
    LBA_t sector,   /* Start sector in LBA */
    UINT count      /* Number of sectors to read */
)
{
    DRESULT res;

    res = RES_PARERR;
    switch (pdrv)
    {
        case DEV_RAM :
            // translate the arguments here
            // translate the reslut code here
            return res;
        case DEV_MMC :
            res = SD_Read(pdrv, buff, sector, count);
            return res;
        case DEV_USB :
            // translate the arguments here
            // translate the reslut code here
            return res;
    }

    return RES_PARERR;
}
```

- 5) 修改 DRESULT disk_write(BYTE pdrv, const BYTE* buff, LBA_t sector, UINT count)函数如下:

```
DRESULT disk_write (
    BYTE pdrv,      /* Physical drive nmuber to identify the drive */
    const BYTE *buff, /* Data to be written */
    LBA_t sector,   /* Start sector in LBA */
    UINT count      /* Number of sectors to write */
)
{
    DRESULT res;

    res = RES_PARERR;
    switch (pdrv)
    {
        case DEV_RAM :
            // translate the arguments here
            // translate the reslut code here
            return res;
        case DEV_MMC :
            res = SD_Write(pdrv, buff, sector, count);
```

```
        return res;
    case DEV_USB :
        // translate the arguments here
        // translate the result code here
        return res;
    }

    return RES_PARERR;
}
```

6) 修改 DRESULT disk_ioctl(BYTE pdrv, BYTE cmd, void* buff)函数如下:

```
DRESULT disk_ioctl (
    BYTE pdrv,    /* Physical drive number (0..) */
    BYTE cmd,     /* Control code */
    void *buff    /* Buffer to send/receive control data */
)
{
    DRESULT res;

    res = RES_PARERR;
    switch (pdrv)
    {
        case DEV_RAM :
            // Process of the command for the RAM drive
            return res;
        case DEV_MMC :
            res = SD_Ioctl(pdrv, cmd, buff);
            return res;
        case DEV_USB :
            // Process of the command the USB drive
            return res;
    }

    return RES_PARERR;
}
```

7) 添加函数 DWORD get_fattime(void)如下:

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

4.4 FatFS 功能配置

ffconf.h 文件是 FatFs 功能配置文件，可以根据使用需求对文件内容进行修改。ffconf.h 对每个配置选项都做了详细的使用情况说明，下面只列出修改的配置，其他配置采用默认即可。

修改 ffconf.h 文件中的配置选项如下：

#define FF_USE_MKFS	1
#define FF_USE_FASTSEEK	1
#define FF_CODE_PAGE	936
#define FF_VOLUMES	3

4.5 FatFS 测试验证

到此，基于 SD 卡的 FatFs 移植已经完成，并把 FatFs 添加到工程中了，接下来就可以编写 mian.c 文件中的测试代码，来验证 FatFs 移植后工作是否正常。

4.5.1 修改 main.c 文件

```
#include "hc32_ddl.h"
#include "ff.h"

/* Local variable definitions ('static') */
static FATFS SDFatFs; /* File system object */
static FIL TestFile; /* File object */
/* A work buffer for the f_mkfs() */
static uint8_t u8WorkBuffer[FF_MAX_SS];

/**
 * @brief MCU Peripheral registers write unprotected.
 * @param None
 * @retval None
 * @note Comment/uncomment each API depending on APP requires.
 */
static void Peripheral_WE(void)
{
    /* Unlock GPIO register: PSPCR, PCCR, PINAER, PCRxy, PFSRxy */
    GPIO_Unlock();
    /* Unlock PWC register: FCG0 */
    PWC_FCG0_Unlock();
    /* Unlock PWC, CLK, PVD registers, @ref PWC_REG_Write_Unlock_Code for details */
    PWC_Unlock(PWC_UNLOCK_CODE_0 | PWC_UNLOCK_CODE_1);
    /* Unlock SRAM register: WTCR */
    SRAM_WTCR_Unlock();
    /* Unlock SRAM register: CKCR */
    // SRAM_CKCR_Unlock();
    /* Unlock all EFM registers */
    EFM_Unlock();
    /* Unlock EFM register: FWMC */
    // EFM_FWMC_Unlock();
    /* Unlock EFM OTP write protect registers */
    // EFM_OTP_WP_Unlock();
    /* Unlock all MPU registers */
    // MPU_Unlock();
}

/**
 * @brief MCU Peripheral registers write protected.
 * @param None
 * @retval None
 * @note Comment/uncomment each API depending on APP requires.
 */
static void Peripheral_WP(void)
{

```



```

/* Lock GPIO register: PSPCR, PCCR, PINAER, PCRxy, PFSRxy */
// GPIO_Lock();
/* Lock PWC register: FCG0 */
PWC_FCG0_Lock();
/* Lock PWC, CLK, PVD registers, @ref PWC_REG_Write_Unlock_Code for details */
// PWC_Lock(PWC_UNLOCK_CODE_0 | PWC_UNLOCK_CODE_1 |
PWC_UNLOCK_CODE_2);
/* Lock SRAM register: WTCR */
SRAM_WTCR_Lock();
/* Lock SRAM register: CKCR */
// SRAM_CKCR_Lock();
/* Lock EFM OTP write protect registers */
// EFM_OTP_WP_Lock();
/* Lock EFM register: FWMC */
// EFM_FWMC_Lock();
/* Lock all EFM registers */
EFM_Lock();
/* Lock all MPU registers */
// MPU_Lock();
}

/**
 * @brief SYS clock initialize.
 * @param None
 * @retval None
 */
static void SYS_CLK_Init(void)
{
    stc_clk_pllh_init_t stcPLLHInit;

    /* PCLK0, HCLK Max 200MHz */
    CLK_ClkDiv(CLK_CATE_ALL, (CLK_PCLK0_DIV1 | CLK_PCLK1_DIV2 |
    CLK_PCLK2_DIV4 | \
        CLK_PCLK3_DIV4 | CLK_PCLK4_DIV2 | CLK_EXCLK_DIV2 | \
        CLK_HCLK_DIV1));

    (void)CLK_PLLHStrucInit(&stcPLLHInit);
    /* VCO = (8/1)*100 = 800MHz*/
    stcPLLHInit.u8PLLState = CLK_PLLH_ON;
    stcPLLHInit.PLLCFGR = 0UL;
    stcPLLHInit.PLLCFGR_f.PLLM = 1UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLN = 100UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLP = 4UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLQ = 4UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLR = 4UL - 1UL;
    stcPLLHInit.PLLCFGR_f.PLLSRC = CLK_PLLSRC_XTAL;
    (void)CLK_PLLHInit(&stcPLLHInit);

    /* Highspeed SRAM set to 1 Read/Write wait cycle */
    SRAM_SetWaitCycle(SRAM_SRAMH, SRAM_WAIT_CYCLE_1,
    SRAM_WAIT_CYCLE_1);

    /* SRAM1_2_3_4_backup set to 2 Read/Write wait cycle */
    SRAM_SetWaitCycle((SRAM_SRAM123 | SRAM_SRAM4 | SRAM_SRAMB),
    SRAM_WAIT_CYCLE_2, SRAM_WAIT_CYCLE_2);

```

```

/* 0-wait @ 40MHz */
EFM_SetWaitCycle(EFM_WAIT_CYCLE_5);

/* 4 cycles for 200 ~ 250MHz */
GPIO_SetReadWaitCycle(GPIO_READ_WAIT_4);

CLK_SetSysClkSrc(CLK_SYSCLKSOURCE_PLLH);
}

/**
 * @brief Main function of SDIOC SD card.
 * @param None
 * @retval int32_t return value, if needed
 */
int32_t main(void)
{
    FRESULT fRet;
    uint32_t u32WBNbr, u32RBNbr;
    uint8_t u8WriteText[] = "This is a string used to test the FatFs";
    uint8_t u8ReadText[100];
    en_result_t enTestRet = Error;
    MKFS_PARM opt;
    char SDPath[] = "1:";

    /* Peripheral registers write unprotected */
    Peripheral_WE();
    /* Configure clock */
    SYS_CLK_Init();
    /* Configure BSP */
    BSP_IO_Init();
    BSP_LED_Init();
    /* Configure UART */
    (void)DDL_PrintfInit();

    /* Register the file system object to the FatFs module */
    if (FR_OK != f_mount(&SDFatFs, (TCHAR const*)SDPath, 0U))
    {
        (void)printf("FatFs Initialization Error!\r\n");
    }
    else
    {
        (void)memset(&opt, 0, sizeof(MKFS_PARM));
        opt.fmt = (BYTE)FM_ANY;
        /* Create a FAT file system (format) on the logical drive */
        if (FR_OK != f_mkfs((TCHAR const*)SDPath, &opt, u8WorkBuffer,
        sizeof(u8WorkBuffer)))
        {
            (void)printf("FatFs Format Error!\r\n");
        }
        else
        {
            /* Create and Open a new text file object with write access */
            if (FR_OK != f_open(&TestFile, "1:Test.txt", ((BYTE)FA_CREATE_ALWAYS |
            (BYTE)FA_WRITE)))

```



```
BSP_LED_On(LED_RED);
BSP_LED_Off(LED_BLUE);
}
else
{
    /* Test success */
    BSP_LED_On(LED_BLUE);
    BSP_LED_Off(LED_RED);
}
/* Peripheral registers write protected */
Peripheral_WP();

for (;;)
{
}
}
```

4.5.2 验证功能

- 1) 将目标板（EV_F4A0_LQ176_V10）上电，并将 TF 卡插到 TK 卡座（J46）上；
- 2) 重新编译 sdioc_sd 工程，下载程序代码并运行；
- 3) 观察 LED 状态，LED_B 亮则 FatFs 功能正常，LED_R 亮则 FatFs 功能错误。

5 版本信息 & 联系方式

日期	版本	修改记录
2021/7/27	Rev1.0	初版发布



如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email: mcu@hdsc.com.cn

网址: <http://www.hdsc.com.cn/mcu.htm>

通信地址: 上海市浦东新区中科路 1867 号 A 座 10 层

邮编: 201203

