ARC

FatFs

支持	www.armrunc.com http://armrunc.taobao.com

Change History

版本号	更改描述	作者	修改时间
1.0.0	第一版	armrunc	2012-03-07

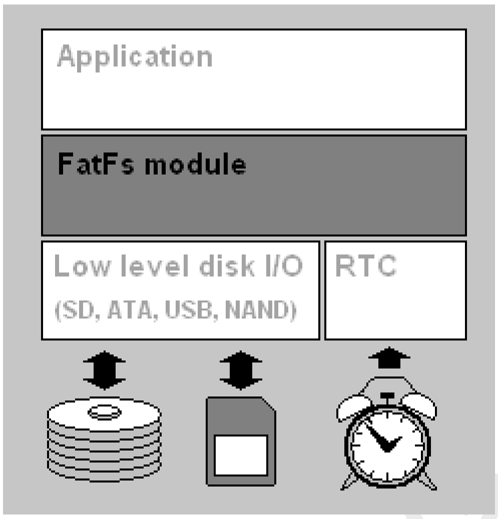
目录

第20章 FatFs

20.1 FatFs 简介

FatFs 是一个通用的文件系统模块，用于在小型嵌入式系统中实现 FAT 文件系统。 FatFs 的编写遵循 ANSI C，因此不依赖于硬件平台。它可以嵌入到便宜的微控制器中，不需要做任何修改。

FatFs 的软件结构如下：



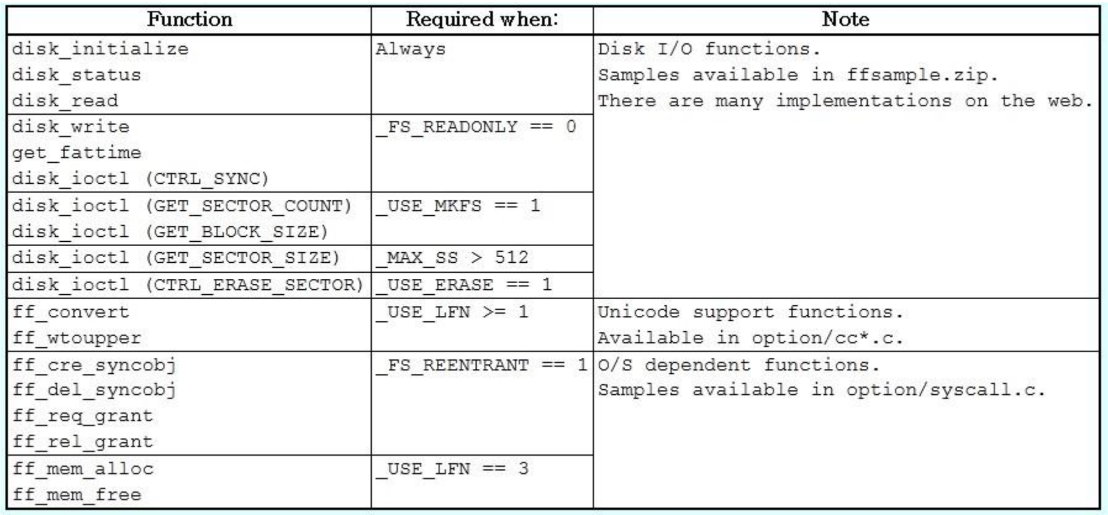
20.2 FatFs 移植

20.2.1 注意事项

- FatFs 模块全部用 ANSI C 编写，所以你需要一个兼容 ANSI C 的编译器；
- FatFs 假定 char/short/long 的大小为 8/16/32 位，int 可以为 16 或者 32 位，如果你的编译器和这个不兼容，请修改 integer.h

20.2.2 需要重写的函数

你仅需要提供底层的磁盘读写函数，下面的表格列出了你需要实现的函数。



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	

本实例中我们配置了上面表格的前三行。具体代码参考软件设计一章。

20.3 FatFs 应用实例 ----- SD 卡上创建读取文件

20.3.1 实例描述

本实例需要 SD 卡，移植了 FatFs，通过串口提示操作，可以 格式化 SD 卡，创建删除一个文件，列出文件列表，得到 SD 卡信息，创建文件夹，读取和编辑文件，你也可以放到电脑上读取该 SD 卡测试以上操作。

20.3.2 硬件设计

请参考 第 SD 14 章。

20.3.3 软件设计

我们在底层实现了磁盘读写程序，如下：
disk_initialize
disk_status
disk_read
disk_write
get_fattime
disk_ioctl
在应用层，我们通过串口提示操作，调用 FatFs API，实现一些功能。具体代码如下：
文件 Fatfs_main.c

```
/**
  * @brief   Main program, SD card SPI interface read example.
  * @param   None
  * @retval None
  */
int main(void)
{
    ARC_SysTick_Init();
    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);

    ARC_SD_SPI_Init();

    SPI_Cmd(SPI1, ENABLE); /*!< SD_SPI enable */

    ARC_DMA1_RCC_Init();

    while (1)
    {
        ARC_fat_menu_init();
        ARC_SysTick_Delay(1000);
    }
}
```

文件 ARC_Fatfs.c

```
/** @defgroup ARC_FATFS_Private_Functions
  * @{
  */

void ARC_fat_menu_init(void)
{
    uint8_t key = 0;
    while (1)
    {
        printf("\nplease choose...\n");
        printf("format-------------------------------- f\n");
        printf("create file ----------------------------c\n");
        printf("delete file ----------------------------d\n");
        printf("list files -----------------------------l\n");
        printf("reboot     ------------------------------ s\n");
        printf("disk info ------------------------------i\n");
        printf("create folder --------------------------t\n");
        printf("edit file ------------------------------e\n");
        printf("read file ------------------------------r\n");
        key = getchar();
        printf("\n");
```

```
                switch (key)
                {
                        case 'f':                      //Format
                                ARC_format_disk();
                                break;

                        case 'c':                      //Creat File
                                ARC_creat_file();
                                break;

                        case 'd':                      //Delete File
                                ARC_delete_file();
                                break;

                        case 'l':                      //list Files
                                ARC_list_file();
                                break;

                        case 'i':                      //Disk info
                                ARC_get_disk_info();
                                break;

                        case 't':                      //Creat Dir
                                ARC_creat_dir();
                                break;

                        case 'e':                      //Edit File
                                ARC_edit_file();
                                break;

                        case 'r':                      //Read File
                                ARC_read_file();
                                break;

                        case 's':                      //soft reset
                                ARC_Sys_Soft_Reset();
                                break;

                        default:
                                printf("invalid input, try again\n");
                                break;
                }
        }
}
```

```
void ARC_edit_file(void)
{
    FATFS fs;
    FIL file;
    FRESULT res;
    DIR dirs;
    FILINFO finfo;
    char key = 0;
    char path[20];

    uint32_t index = 0x00;
    uint32_t reindex = 0x00;
    uint8_t file_buff[512] = {0};

    uint32_t files_num = 0;
    res = f_mount(0,&fs);
    if (res != FR_OK)
    {
        printf("mont file system error, error code: %u\n",res);
        return;
    }
    res = f_opendir(&dirs,"/");
    printf("file list\n");
    if (res == FR_OK)
    {
        while ((f_readdir(&dirs, &finfo) == FR_OK) && finfo.fname[0])
        {
            if (finfo.fattrib & AM_DIR)
            {
                continue;
            }
            else
            {
                files_num++;
                printf("/%12s%7ld KB\n", &finfo.fname[0],(finfo.fsize + 512) /
1024);
            }
        }
        if( files_num == 0 )
        {
            printf("no file\n!");
        }
    }
```

```
else
{
    printf("failed to open root directory, error code: %u\n", res);
}
printf("input the full name of the file, terminated with enter\n");
scanf("%[^\n]",path);
res = f_open(&file,path,FA_READ | FA_WRITE);
if (res == FR_OK)
{

    printf("file: %s opened successfully\n",path);
    printf("input the text!");
    printf("terminated with ESC or Ctrl+C\n");

    while(1)
    {
        key = getchar();
        if ((key == 0x1B) || (key == 0x03))     //key ESC or Ctrl + C
        {
            printf("saving data...\n");
            res = f_write(&file,file_buff,index,&reindex);
            if ((res == FR_OK) && (reindex == index))
            {
                printf("data saved\n");
                f_close(&file);
                index = 0x00;
                reindex = 0x00;
            }
            else
            {
                printf("fail to save data, error code: %u", res);
            }
            break;
        }
        else
        {
            file_buff[index++] = key;
            if (index > 512)
            {
                index = 0x00;
            }
        }
    }
}
```

```
    else
    {
        printf("fail to open the file, error code: %u\n",res);
    }
}

void ARC_read_file(void)
{
    FATFS fs;
    FIL file;
    FRESULT res;
    DIR dirs;
    FILINFO finfo;
    char path[20];
    char buffer[512] = {0};
    uint32_t i;
    uint32_t re,files_num = 0;
    res = f_mount(0,&fs);
    if (res != FR_OK)
    {
        printf("mont file system error, error code: %u\n",res);
        return;
    }
    res = f_opendir(&dirs,"/");
    if (res == FR_OK)
    {
        printf("file list\n");
        while ((f_readdir(&dirs, &finfo) == FR_OK) && finfo.fname[0])
        {
            if (finfo.fattrib & AM_DIR)
            {
                continue;
            }
            else
            {
                files_num++;
                printf("/%12s%7ld KB\n",   &finfo.fname[0],(finfo.fsize + 512)
/ 1024);
            }
        }
        if( files_num == 0 )
        {
            printf("no files\n");
            return;
        }
    }
    else
    {
        printf("failed to open root directory, error code: %u\n", res);
    }
    printf("input the full name of the file, terminated with enter\n");
    scanf("%[^\n]",path);
    res = f_open(&file,path,FA_READ);
    printf("file opened\n");

    if (res == FR_OK)
    {
        while (1)
        {
            for(i = 0;i < 512;i++)
            {
                buffer[i] = 0x00;
            }
            res = f_read(&file,buffer,512,&re);
            printf("%s\n",buffer);

            if (res || re == 0)
            {
                printf("finish reading file, about to close the file!\n");
                f_close(&file);
                break;
            }
        }
    }
    f_mount(0,NULL);
}

void ARC_creat_dir(void)
{
    FATFS fs;
    FRESULT res;
    char path[20];
    res = f_mount(0,&fs);
    if (res != FR_OK)
    {
        printf("mont file system error, error code: %u\n",res);
        return;
    }
    printf("please input the directory name terminated by enter\n");

    scanf("%[^\n]",path);

    res = f_mkdir(path);
    if (res == FR_OK)
    {
        printf("directory created successfully\n");
    }
    else
    {
        printf("failed to create directory, error code: %u", res);
    }
    f_mount(0,NULL);
}

void ARC_format_disk(void)
{
    FATFS fs;
    uint8_t res;
    res = f_mount(0,&fs);
    if (res != FR_OK)
    {
        printf("mont file system error, error code: %u\n",res);
        return;
    }
    printf("formating, my need minutes, please wait...\n");
    res = f_mkfs(0,1,4096);
    if (res == FR_OK)
    {
        printf("successful...\n");
    }
    else
    {
        printf("failed to format disk, error code: %u\n", res);
    }
    f_mount(0,NULL);
}

void ARC_creat_file(void)
{
    FIL file;
    FIL *pf = &file;
    FATFS fs;
```

```
        uint8_t res;
        uint8_t name[16] = {0};
        printf("please input file name, format: 8 + 3...\n");
        printf("for example:123.txt\n");
        scanf("%[^\n]",name);
        printf("name %s", name);
        res = f_mount(0, &fs);
        if (res != FR_OK)
        {
                printf("mont file system error, error code: %u\n",res);
                return;
        }
        res = f_open(pf,(const TCHAR *)name,FA_READ | FA_WRITE |
FA_CREATE_NEW);
        if (res == FR_OK)
        {
                printf("file created successfully\n");
                res = f_close(pf);
                if (res != FR_OK)
                {
                        printf("file created successfully, but failed to close\n");
                        printf("error code: %u",res);
                }
        }
        else
        {
                printf("failed to create file, error code: %u", res);
        }
        f_mount(0,NULL);
}

void ARC_delete_file(void)
{
        FATFS fs;
        FRESULT res;
        uint8_t name[16] = {0};
        printf("please input the file name that you want to delete\n");
        scanf("%[^\n]",name);
        res = f_mount(0,&fs);
        if (res != FR_OK)
        {
                printf("mont file system error, error code: %u\n",res);
                return;
        }
```

```
    res = f_unlink((TCHAR *)name);

    if (res == FR_OK)
    {
        printf("file deleted successfully!\n");
    }
    else if (res == FR_NO_FILE)
    {
        printf("no such file or directory\n");
    }
    else if (res == FR_NO_PATH)
    {
        printf("no such path\n");
    }
    else
    {
        printf("error code: %u\n",res);
    }
    f_mount(0,NULL);
}

void ARC_list_file(void)
{
    FATFS fs;
    FILINFO finfo;
    FRESULT res;
    DIR dirs;
    int files_num = 0;
    res = f_mount(0, &fs);
    if (res != FR_OK)
    {
        printf("mont file system error, error code: %u\n",res);
        return;
    }
    res = f_opendir(&dirs, "/");
    if (res == FR_OK)
    {
        printf("------------ file list ------------\n");
        while ((f_readdir(&dirs, &finfo) == FR_OK) && finfo.fname[0])
        {
            if (finfo.fattrib & AM_DIR)//if Directory
            {
                files_num++;
                printf("/%s\n", &finfo.fname[0]);
            }
            else
            {
                continue;
            }
        }
    }
    else
    {
        printf("failed to open root directory, error code: %u\n", res);
    }
    res = f_opendir(&dirs, "/");
    if (res == FR_OK)
    {
        while ((f_readdir(&dirs, &finfo) == FR_OK) && finfo.fname[0])
        {
            if (finfo.fattrib & AM_DIR)
            {
                continue;
            }
            else
            {
                files_num++;
                printf("/.%12s%7ld KB \n",   &finfo.fname[0],(finfo.fsize +
512) / 1024);
            }
        }
        if( files_num == 0 )//no file
        {
            printf("no file!\n");
        }
    }
    else
    {
        printf("failed to open root directory, error code: %u\n", res);
    }
    f_mount(0,NULL);
}

void ARC_get_disk_info(void)
{
    FATFS fs;
    FATFS *fls = &fs;
    FRESULT res;
```

```
    DWORD fre_clust,tot_sect,fre_sect;

    res = f_mount(0,&fs);
    if (res != FR_OK)
    {
        printf("mont file system error, error code: %u\n",res);
        return;
    }
    res = f_getfree("/",&fre_clust,&fls);
    if (res == FR_OK)
    {
        tot_sect = (fls->n_fatent - 2) * fls->csize;
        fre_sect = fre_clust * fls->csize;


        /* Print free space in unit of KB (assuming 512 bytes/sector) */
        printf("%lu KB total drive space.\n"
                "%lu KB available.\n",
                fre_sect / 2, tot_sect / 2);
    }
    else
    {
        printf("failed to get disk info, error code: %u\n", res);
    }
    f_mount(0,NULL);
}

void ARC_Sys_Soft_Reset(void)
{
    NVIC_SystemReset();
}
```

文件 ARC_SD.c

```
static SD_Card_Info SDCardInfo;

/**
  * @brief   get the pointer to SD card information struct.
  * @param    None
  * @retval pointer to the struct.
  */
SD_Card_Info * ARC_SD_SPI_GetCardInof(void)
{
    return &SDCardInfo;
}
```

```
/**
   * @brief   Wait for the SD SPI bus ready.
   * @param   None
   * @retval None
   */
__inline uint8_t ARC_SD_CSReady(void)
{
      uint8_t i = 0;
      uint16_t tmp = 0;
      do
      {
            tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);

      }while(tmp != 0xFF && tmp != 0x00 && ++i < 0xFFFE);
      if(i >= 0xFFFE)
            return 0;
      return 1;
}

/**
   * @brief   Wait for card is not busy.
   * @param   None
   * @retval None
   */
__inline void ARC_SD_CardBusy(void)
{
      while(ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE) == 0);
}

/**
   * @brief   Send 6 bytes command to the SD card.
   * @param   Cmd: The command send to SD card.
   * @param   argument: The command argument.
   * @param   response_type: the SPI command response type.
   * @param   *response: the SPI response returned.
   * @retval The SD Response.
   */
uint8_t ARC_sd_send_command(uint8_t cmd, uint32_t argument,
                                        SD_Response response_type, uint8_t
*response)
{
      int32_t i = 0;
      uint8_t crc = 0x01;
```

```
int8_t response_length = 0;
uint8_t tmp;
uint8_t Frame[6];

if (cmd & 0x80)    /* Send a CMD55 prior to ACMD<n> */
{
    cmd &= 0x7F;
    ARC_sd_send_command(SD_CMD_APP_CMD, 0, R1, response);
    if (response[0] > 0x01)
    {
        ARC_SD_CS_HIGH();
        ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        return response[0];
    }
}
ARC_SD_CS_LOW();
while(!ARC_SD_CSReady());
if(cmd == SD_CMD_GO_IDLE_STATE)
    crc = 0x95;

if(cmd == SD_CMD_SEND_IF_COND)
    crc = 0x87;

/* All data is sent MSB first, and MSb first */
/* cmd Format:
cmd[7:6] : 01
cmd[5:0] : command */

Frame[0] = ((cmd & 0x3F) | 0x40); /*!< Construct byte 1 */

Frame[1] = (uint8_t)(argument >> 24); /*!< Construct byte 2 */

Frame[2] = (uint8_t)(argument >> 16); /*!< Construct byte 3 */

Frame[3] = (uint8_t)(argument >> 8); /*!< Construct byte 4 */

Frame[4] = (uint8_t)(argument); /*!< Construct byte 5 */

Frame[5] = (uint8_t)(crc); /*!< Construct CRC: byte 6 */

for (i = 0; i < 6; i++)
{
    ARC_SPI_SendByte(SPI1, Frame[i]); /*!< Send the Cmd bytes */
}
```

```
        switch (response_type)
        {
                case R1:
                case R1B:
                        response_length = 1;
                        break;
                case R2:
                        response_length = 2;
                        break;
                case R3:
                case R7:
                        response_length = 5;
                        break;
                default:
                        break;
        }

        /* Wait for a response. A response can be recognized by the start bit (a
zero) */
        i = 0xFF;
        do
        {
                tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        }while ((tmp & 0x80) && --i);

        response[0] = tmp;

        /* Just bail if we never got a response */
        if ((i > 0) && ((response[0] & SD_ILLEGAL_COMMAND) !=
SD_ILLEGAL_COMMAND))
        {
                i = 1;
                while(i < response_length)
                {
                        tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
                        response[i] = tmp;
                        i++;
                }

                /* If the response is a "busy" type (R1B), then there's some
                  * special handling that needs to be done. The card will
                  * output a continuous stream of zeros, so the end of the BUSY
                  * state is signaled by any nonzero response. The bus idles
```

```
                 * high.
                 */
               if (response_type == R1B)
               {
                      do
                      {
                             tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);;
                      }while (tmp != 0xFF);

                      ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
               }
        }

        ARC_SD_CS_HIGH();
        ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        return response[0];
}

/**
    * @brief    send buffer to SPI via DMA.
    * @param    *buff, the memory address to be sent to the SPI device
    * @param    byteTransfer, the number to be sent to the SPI device
    * @retval None
    */
void ARC_SD_SPI_DMASend(const uint8_t *buff, uint32_t byteTransfer)
{
        ARC_DMA1_Ch3_Param_Init(buff, byteTransfer,
DMA_MemoryInc_Enable);

        /* Enable SPI_MASTER DMA Tx request */
        SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, ENABLE);

        /* Enable DMA channels */
        DMA_Cmd(DMA1_Channel3, ENABLE);

        /* Transfer complete */
        while(!DMA_GetFlagStatus(DMA1_FLAG_TC3));

        /* Enable DMA channels */
        DMA_Cmd(DMA1_Channel3, DISABLE);

        /* Disable SPI_MASTER DMA Tx request */
        SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, DISABLE);
}
```

```
/**
   * @brief    read buffer from SPI device via DMA.
   * @param    *buff, the memory address holding the content from SPI device
   * @param    byteTransfer, the number to be read from the SPI device
   * @retval None
   */
void ARC_SD_SPI_DMAReceive(uint8_t *buff, uint32_t byteTransfer)
{
        uint8_t dummyByte = SD_DUMMY_BYTE;
        ARC_DMA1_Ch2_Param_Init(buff, byteTransfer);
        ARC_DMA1_Ch3_Param_Init(&dummyByte, byteTransfer,
DMA_MemoryInc_Disable);

        /* Enable SPI_MASTER DMA Rx request */
        SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Rx, ENABLE);
        /* Enable SPI_MASTER DMA Tx request */
        SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, ENABLE);

        /* Enable DMA channels */
        DMA_Cmd(DMA1_Channel2, ENABLE);
        /* Enable DMA channels */
        DMA_Cmd(DMA1_Channel3, ENABLE);

        /* Transfer complete */
        while(!DMA_GetFlagStatus(DMA1_FLAG_TC2));

        /* Disable DMA channels */
        DMA_Cmd(DMA1_Channel2, DISABLE);
        /* Enable DMA channels */
        DMA_Cmd(DMA1_Channel3, DISABLE);

        /* Disable SPI_MASTER DMA Rx request */
        SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Rx, DISABLE);
        /* Disable SPI_MASTER DMA Tx request */
        SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, DISABLE);
}

/**
   * @brief    read a block from SPI device via DMA.
   * @param    *buff, the memory address holding the content from SPI device
   * @param    byteTransfer, the number to be read from the SPI device
   * @retval None
   */
```

```
uint8_t ARC_SD_SPI_ReadBlock(uint8_t *buff, uint32_t byteTransfer)
{
    uint16_t expire_count = 0xFFFF;
    uint8_t token, ret = 1;
    ARC_SD_CS_LOW();
    do
    {
        token = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
    }while(token != 0xFE && --expire_count);
    if(token != 0xFE)
    {
        ret = 0;
    }
    else
    {
        #if 1
        ARC_SD_SPI_DMAReceive(buff, byteTransfer);
        #else
        while(byteTransfer--)
        {
            *buff = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
            buff++;
        }
        #endif
        ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
    }
    ARC_SD_CS_HIGH();
    return ret;
}

/**
  * @brief   send a block to SPI via DMA.
  * @param   *buff, the memory address to be sent to the SPI device
  * @param   token
  * @retval successful or not
  */
uint8_t ARC_SD_SPI_WriteBlock(const uint8_t *buff, uint8_t token)
{
    uint8_t resp, ret = 1;
    #if 0
    uint16_t i = 0;
    #endif
    ARC_SD_CS_LOW();
    while(!ARC_SD_CSReady());
    ARC_SPI_SendByte(SPI1, token); /* transmit data token */
    if (token != 0xFD) /* Is data token */
    {

        #if 1
        ARC_SD_SPI_DMASend( buff, 512 );
        #else
        while(i < 512)
        {
            ARC_SPI_SendByte(SPI1, *buff);
            buff++;
            i++;
        }
        #endif

        ARC_SPI_SendByte(SPI1, 0xFF);                /* CRC (Dummy)
*/
        ARC_SPI_SendByte(SPI1, 0xFF);                /* CRC (Dummy)
*/
        resp = ARC_SPI_SendByte(SPI1, 0xFF);         /* Receive data
response */
        if ((resp & 0x1F) != 0x05)                   /* If not accepted, return
with error */
            ret = 0;
    }
    ARC_SD_CardBusy();
    ARC_SD_CS_HIGH();
    return ret;
}

#ifdef ARC_FATFS
/**
  * @brief Read Sector(s).
  */
DRESULT disk_read(uint8_t drv, uint8_t *buff, DWORD sector,uint8_t count)
{
    uint8_t sd_response[5];
    SD_Card_Info *sdCardInfo;
    sdCardInfo = ARC_SD_SPI_GetCardInof();

    if (drv || !count)
        return RES_PARERR;
    if (sdCardInfo->sd_stat & SD_Status_NotInit)
```

```
        return RES_NOTRDY;

    if (!(sdCardInfo->sd_ct & SD_SDHC))
        sector *= 512;    /* Convert to byte address if needed */

    if (count == 1)        /* Single block read */
    {
        if (ARC_sd_send_command(SD_CMD_READ_SINGLE_BLOCK,
sector, R1, sd_response) == 0) /* READ_SINGLE_BLOCK */
        {
            if (ARC_SD_SPI_ReadBlock(buff, 512))
            {
                count = 0;
            }
        }
    }
    else /* Multiple block read */
    {
        if (ARC_sd_send_command(SD_CMD_READ_MULT_BLOCK,
sector, R1, sd_response) == 0)    /* READ_MULTIPLE_BLOCK */
        {
            do
            {
                if (!ARC_SD_SPI_ReadBlock(buff, 512))
                {
                    break;
                }
                buff += 512;
            } while (--count);
            ARC_sd_send_command(SD_CMD_STOP_TRANSMISSION, 0,
R1B, sd_response); /* STOP_TRANSMISSION */
        }
    }
    return count ? RES_ERROR : RES_OK;
}

/**
  * @brief    Write Sector(s).
  */
DRESULT disk_write(uint8_t drv, const uint8_t *buff, DWORD sector, uint8_t
count)
{
    uint8_t sd_response[5];
    SD_Card_Info *sdCardInfo;
    sdCardInfo = ARC_SD_SPI_GetCardInof();

    if (drv || !count)
        return RES_PARERR;
    if (sdCardInfo->sd_stat & SD_Status_NotInit)
        return RES_NOTRDY;
    if (sdCardInfo->sd_stat & SD_Status_Protect)
        return RES_WRPRT;

    if (!(sdCardInfo->sd_ct & SD_SDHC))
        sector *= 512;    /* Convert to byte address if needed */

    if (count == 1) /* Single block write */
    {
        if ((ARC_sd_send_command(SD_CMD_WRITE_SINGLE_BLOCK,
sector, R1, sd_response) == 0) && /* WRITE_BLOCK */
             ARC_SD_SPI_WriteBlock(buff, 0xFE))
            count = 0;
    }
    else /* Multiple block write */
    {
        if (sdCardInfo->sd_ct & SD_SDC)

ARC_sd_send_command(SD_ACMD_APP_SET_WR_BLK_ERASE_COUNT,
count, R1, sd_response);
        if (ARC_sd_send_command(SD_CMD_WRITE_MULT_BLOCK,
sector, R1, sd_response) == 0)    /* WRITE_MULTIPLE_BLOCK */
        {
            do
            {
                if (!ARC_SD_SPI_WriteBlock(buff, 0xFC))
                    break;
                buff += 512;
            } while (--count);
            if (!ARC_SD_SPI_WriteBlock(0, 0xFD)) /* STOP_TRAN token */
                count = 1;
        }
    }
    return count ? RES_ERROR : RES_OK;
}

/**
  * @brief   Get Disk Status.
  */
```

```
DSTATUS disk_status (uint8_t drv)
{
    SD_Card_Info *sdCardInfo;

    if (drv)
        return STA_NOINIT;          /* Supports only single drive */

    sdCardInfo = ARC_SD_SPI_GetCardInof();
    return (DSTATUS) (sdCardInfo->sd_stat);
}

/**
  * @brief   get disk information.
  */
DRESULT disk_ioctl (uint8_t drv, uint8_t ctrl, void *buff)
{
    DRESULT res;
    uint8_t sd_response[5];
    uint8_t n, csd[16], *ptr = buff;
    uint32_t csize;
    SD_Card_Info *sdCardInfo;

    if (drv)
        return RES_PARERR;

    sdCardInfo = ARC_SD_SPI_GetCardInof();

    res = RES_ERROR;

    if (ctrl == CTRL_POWER)
    {
        /* not available */
        res = RES_OK;
    }
    else
    {
        if (sdCardInfo->sd_stat & SD_Status_NotInit)
            return RES_NOTRDY;

        switch (ctrl)
        {
            case CTRL_SYNC :   /* Make sure that no pending write process
*/
                /* Not available */
                res = RES_OK;
                break;

            case GET_SECTOR_COUNT :    /* Get number of sectors on the
disk    */
                if ((ARC_sd_send_command(SD_CMD_SEND_CSD, 0, R1,
sd_response) == 0) && ARC_SD_SPI_ReadBlock(csd, 16))
                {
                    if ((csd[0] >> 6) == 1)    /* SDC version 2.00 */
                    {
                        csize = csd[9] + ((WORD)csd[8] << 8) + 1;
                        *(DWORD*)buff = (DWORD)csize << 10;
                    }
                    else /* SDC version 1.XX or MMC*/
                    {
                        n = (csd[5] & 15) + ((csd[10] & 128) >> 7) + ((csd[9]
& 3) << 1) + 2;
                        csize = (csd[8] >> 6) + ((WORD)csd[7] << 2) +
((WORD)(csd[6] & 3) << 10) + 1;
                        *(DWORD*)buff = (DWORD)csize << (n - 9);
                    }
                    res = RES_OK;
                }
                break;

            case GET_SECTOR_SIZE :          /* Get R/W sector size
(WORD) */
                *(DWORD*)buff = 512;
                res = RES_OK;
                break;

            case GET_BLOCK_SIZE : /* Get erase block size in unit of
sector (DWORD) */
            if (sdCardInfo->sd_ct & SD_Ver2)    /* SDC version 2.00 */
            {
                if
((ARC_sd_send_command(SD_ACMD_APP_SD_STATUS, 0, R2,
sd_response) == 0) &&/* Read SD status */
                    ARC_SD_SPI_ReadBlock(csd, 16))
                {
                    for (n = 64 - 16; n; n--)
                        ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
/* Purge trailing data */
                    *(DWORD*)buff = 16UL << (csd[10] >> 4);
                    res = RES_OK;
                }
            }
            else /* SDC version 1.XX or MMC */
            {
                if ((ARC_sd_send_command(SD_CMD_SEND_CSD, 0, R1,
sd_response) == 0) &&
                     ARC_SD_SPI_ReadBlock(csd, 16))     /* Read CSD
*/
                {
                    if (sdCardInfo->sd_ct & SD_Ver1) /* SDC version 1.XX
*/
                    {
                        *(DWORD*)buff = (((csd[10] & 63) << 1) +
((WORD)(csd[11] & 128) >> 7) + 1) << ((csd[13] >> 6) - 1);
                    }
                    else /* MMC */
                    {
                        *(DWORD*)buff = ((WORD)((csd[10] & 124) >> 2)
+ 1) * (((csd[11] & 3) << 3) + ((csd[11] & 224) >> 5) + 1);
                    }
                    res = RES_OK;
                }
            }
            break;

            case MMC_GET_TYPE :       /* Get card type flags (1 byte) */
                *(DWORD*)buff = sdCardInfo->sd_ct;
                res = RES_OK;
                break;

            case MMC_GET_CSD :        /* Receive CSD as a data block
(16 bytes) */
                if ((ARC_sd_send_command(SD_CMD_SEND_CSD, 0, R1,
sd_response) == 0)   && /* READ_CSD */
                    ARC_SD_SPI_ReadBlock(ptr, 16))
                    res = RES_OK;
                break;

            case MMC_GET_CID :        /* Receive CID as a data block (16
bytes) */
                if (ARC_sd_send_command(SD_CMD_SEND_CID, 0, R1,
sd_response) == 0 &&     /* READ_CID */
                    ARC_SD_SPI_ReadBlock(ptr, 16))
                    res = RES_OK;
                break;

            case MMC_GET_OCR :        /* Receive OCR as an R3 resp (4
bytes) */
                if (ARC_sd_send_command(SD_CMD_READ_OCR, 0, R3,
sd_response) == 0)   /* READ_OCR */
                {
                    ptr = &(sd_response[1]);
                    res = RES_OK;
                }
                break;

            case MMC_GET_SDSTAT:            /* Receive SD status as a
data block (64 bytes) */
                if
((ARC_sd_send_command(SD_ACMD_APP_SD_STATUS, 0, R2,
sd_response) == 0) &&   /* SD_STATUS */
                    ARC_SD_SPI_ReadBlock(ptr, 64))
                {
                    res = RES_OK;
                }
                break;

            default:
            res = RES_PARERR;
        }
    }

    return res;
}

/**
  * @brief   get current time.
  */
DWORD get_fattime (void)
{
    uint32_t res;
    RTC_t rtc;

    ARC_RTC_gettime( &rtc );

    res =   (((DWORD)rtc.year - 1980) << 25)
    | ((DWORD)rtc.month << 21)
```

```
        | ((DWORD)rtc.mday << 16)
        | (WORD)(rtc.hour << 11)
        | (WORD)(rtc.min << 5)
        | (WORD)(rtc.sec >> 1);

        return res;
}

DSTATUS disk_initialize (BYTE drv)
{
        if (drv)
                return STA_NOINIT;                  /* Supports only single drive */
        else
        {
                SD_Card_Info *sdCardInfo;
                sdCardInfo = ARC_SD_SPI_GetCardInof();
                ARC_SD_SPI_Start();
                return (DSTATUS)sdCardInfo->sd_stat;
        }
}

#endif

/**
    * @brief    start sd card.
    * @param    None
    * @retval The SD card type.
    */
SD_Card_Type ARC_SD_SPI_Start(void)
{
        uint8_t i = 0;
        uint16_t retry_times = 0;
        uint8_t sd_response[5];
        SD_Card_Info *sdCardInfo;
        sdCardInfo = ARC_SD_SPI_GetCardInof();

        if (sdCardInfo->sd_stat & SD_Status_NoDisk)
                return SD_Unknown;

        SPI_BaudRateConfig(SPI1, ARC_SPI_MIN_SPEED);

        /*!< SD chip select high */
        ARC_SD_CS_HIGH();
        ARC_SysTick_Delay(100);

        /*!< Send dummy byte 0xFF, 10 times with CS high */
        /*!< Rise CS and MOSI for 80 clocks cycles */
        for (i = 0; i < 10; i++)
        {
                /*!< Send dummy byte 0xFF */
                ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        }
        i = 20;
        do
        {
                ARC_sd_send_command(SD_CMD_GO_IDLE_STATE, 0, R1,
sd_response);
        }while(sd_response[0] != SD_IN_IDLE_STATE && --i);

        if(sd_response[0] == SD_IN_IDLE_STATE)
        {
                ARC_sd_send_command(SD_CMD_SEND_IF_COND, 0x1AA, R7,
sd_response);
                if(sd_response[0] == SD_IN_IDLE_STATE)/* SDv2? */
                {
                    if(sd_response[3] == 0x01 && sd_response[4] == 0xAA)
                    {
                        retry_times = 0xFFF;
                        do
                        {

ARC_sd_send_command(SD_ACMD_APP_SEND_OP_COND, 1UL << 30,
R1, sd_response);
                        }while(sd_response[0] && --retry_times);

                        if(retry_times > 0)
                        {
                            ARC_sd_send_command(SD_CMD_READ_OCR, 0x0,
R3, sd_response);
                            if(sd_response[1] & 0x80)
                            {
                                sdCardInfo->sd_stat &= ~SD_Status_NotInit;
                                sdCardInfo->sd_ct = (sd_response[1] & 0x40) ?
SD_SDHC : SD_SDSC;    /* SDv2 */
                            }
                            else
                            {
                                //printf("SD in power down status\n");
                            }
                        }
                    }
                }
                else /* SDv1 or MMCv3 */
                {
                    ARC_sd_send_command(SD_ACMD_APP_SEND_OP_COND,
0x0, R1, sd_response);
                    if(sd_response[0] <= 1)
                    {
                        sdCardInfo->sd_stat &= ~SD_Status_NotInit;
                        sdCardInfo->sd_ct = SD_Ver1;
                        retry_times = 0xFFF;
                        do
                        {

ARC_sd_send_command(SD_ACMD_APP_SEND_OP_COND, 0x0, R1,
sd_response);
                        }while(sd_response[0] && --retry_times);
                    }
                    else
                    {
                        retry_times = 0xFFF;
                        do
                        {

ARC_sd_send_command(SD_CMD_SEND_OP_COND, 0x0, R1,
sd_response);
                        }while(sd_response[0] && --retry_times);
                    }
                    if (retry_times > 0)
                    {
                        sdCardInfo->sd_stat &= ~SD_Status_NotInit;
                        sdCardInfo->sd_ct = SD_MMC_Ver3;
                        ARC_sd_send_command(SD_CMD_SET_BLOCKLEN, 512,
R1, sd_response);
                    }

                }
        }

        ARC_SD_CS_HIGH();
        SPI_BaudRateConfig(SPI1, ARC_SPI_DEFAULT_SPEED);
```

```
	return sdCardInfo->sd_ct;
}

/**
   * @brief   Initialize SD parameters.
   * @param   None
   * @retval None:
   */
void ARC_SD_SPI_Param_Init(void)
{
	SD_Card_Info *sdCardInfo;
	sdCardInfo = ARC_SD_SPI_GetCardInof();
	sdCardInfo->sd_ct = SD_Unknown;
	sdCardInfo->sd_stat = SD_Status_NotInit;
}

/**
   * @brief   Initialize SD card.
   * @param   None
   * @retval None:
   */
void ARC_SD_SPI_Init(void)
{
	ARC_SPI_Init();
	ARC_SD_SPI_Param_Init();
}
```