By 棉花

目标：spi-flash 9M 后开一个 fatfs

将需要的文件复制到 fatfs 里面

在 makefile 中把头文件包含进去

```
69 INCDIRS      +=  include                         \
70                  arch/$(ARCH)/include            \
71                  arch/$(ARCH)/$(MACH)/include    \
72                  kernel/fs/fatfs
```

填写 diskio.c 文件

```
/* Low level disk I/O module skeleton for FatFs     (C)ChaN, 2014          */

#include "diskio.h"     /* FatFs lower layer API */
#include "ff.h"
#include <xboot.h>

/* Definitions of physical drive number for each drive */
#define ATA         0   /* Example: Map ATA harddisk to physical drive 0 */
#define SPI_FLASH 1     /* Example: Map MMC/SD card to physical drive 1 */

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

    switch (pdrv) {
    case ATA :
        //result = ATA_disk_status();

        // translate the reslut code here

        return stat;

    case SPI_FLASH :
        //result = MMC_disk_status();
```

```
        // translate the reslut code here

        return 0;//直接返回 OK
    }
    return STA_NOINIT;
}



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

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

    switch (pdrv) {
    case ATA :
        //result = ATA_disk_initialize();

        // translate the reslut code here

        return stat;

    case SPI_FLASH :
        //result = MMC_disk_initialize();

        // translate the reslut code here
        stat=0;
        return stat;//驱动的时候已经初始化过了，这里就直接返回 ok
    }
    return STA_NOINIT;
}



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

```
DRESULT disk_read_fat (
	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 */
)
{
	DRESULT res;
	u64_t result;//int result;
	struct block_t * spiblock= NULL;

	switch (pdrv) {
	case ATA :
		// translate the arguments here

		//result = ATA_disk_read(buff, sector, count);

		// translate the reslut code here

		return res;

	case SPI_FLASH :
		// translate the arguments here

		//result = MMC_disk_read(buff, sector, count);

		// translate the reslut code here
		sector+=2304;//start 9M
		spiblock = search_block("spi-flash.0");
		result=block_read(spiblock, buff, sector<<12, count<<12);
		if(result==count<<12)
		{
			res=RES_OK;
		}else
		{
			res=1;
			printf("block_read fail\r\n");
		}
		return res;

	}

	return RES_PARERR;
}
```

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

#if _USE_WRITE
DRESULT disk_write_fat (
	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;
	u64_t result;//int result;
	struct block_t * spiblock= NULL;

	switch (pdrv) {
	case ATA :
		// translate the arguments here

		//result = ATA_disk_write(buff, sector, count);

		// translate the reslut code here

		return res;

	case SPI_FLASH :
		// translate the arguments here

		//result = MMC_disk_write(buff, sector, count);

		// translate the reslut code here
		sector+=2304;
		spiblock = search_block("spi-flash.0");
		result=block_write(spiblock, (u8_t*)buff, sector<<12, count<<12);
		if (result == count << 12) {
			res = RES_OK;
		} else
		{
			printf("block_write fail\r\n");
			res = 1;
```

```
        }
        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 ATA :

        // Process of the command for the ATA drive

        return res;

    case SPI_FLASH :

        // Process of the command for the MMC/SD card
        switch (cmd) {
        /* 扇区数量：2560*4096/1024/1024=10(MB) */
        case GET_SECTOR_COUNT:
            *(DWORD *) buff = 1792;//7M
            break;
            /* 扇区大小  */
        case GET_SECTOR_SIZE:
            *(WORD *) buff = 4096;
            break;
            /* 同时擦除扇区个数 */
        case GET_BLOCK_SIZE:
```

```
                *(DWORD *) buff = 1;
                break;
            }
            res=RES_OK;
            return res;
        }

        return RES_PARERR;
}
#endif

DWORD get_fattime(void) {
        /* 返回当前时间戳 理论上应该从 RCT 获取*/
        return      ((DWORD)(2015 - 1980) << 25)    /* Year 2015 */
                | ((DWORD)1 << 21)                  /* Month 1 */
                | ((DWORD)1 << 16)                  /* Mday 1 */
                | ((DWORD)0 << 11)                  /* Hour 0 */
                | ((DWORD)0 << 5)                     /* Min 0 */
                | ((DWORD)0 >> 1);                  /* Sec 0 */
}
```

配置 ffconf.h

```
#define  _USE_MKFS          1
#define _CODE_PAGE  1
#define _VOLUMES     2
#define  _MAX_SS      4096
```

做一个 cmd 测试

```
#include <command/command.h>
#include "ff.h"

FATFS fs; /* FatFs 文件系统对象 */
FIL fnew; /* 文件对象 */
FRESULT res_flash; /* 文件操作结果 */
UINT fnum; /* 文件成功读写数量 */
BYTE ReadBuffer[1024] = { 0 }; /* 读缓冲区 */
BYTE WriteBuffer[] = /* 写缓冲区*/
"hello3\r\n";

static void usage(void)
{
        printf("usage:\r\n");
        printf("testfatfs\r\n");
```

```
}

FRESULT scan_files (
    char* path        /* Start node to be scanned (also used as work area) */
)
{
    FRESULT res;
    FILINFO fno;
    DIR dir;
    int i;
    char *fn;   /* This function assumes non-Unicode configuration */
#if _USE_LFN
    static char lfn[_MAX_LFN + 1];   /* Buffer to store the LFN */
    fno.lfname = lfn;
    fno.lfsize = sizeof lfn;
#endif


    res = f_opendir(&dir, path);                       /* Open the directory */
    if (res == FR_OK) {
        i = strlen(path);
        for (;;) {
            res = f_readdir(&dir, &fno);                   /* Read a directory item */
            if (res != FR_OK || fno.fname[0] == 0) break;  /* Break on error or end of dir */
            if (fno.fname[0] == '.') continue;             /* Ignore dot entry */
#if _USE_LFN
            fn = *fno.lfname ? fno.lfname : fno.fname;
#else
            fn = fno.fname;
#endif
            if (fno.fattrib & AM_DIR) {                    /* It is a directory */
                sprintf(&path[i], "/%s", fn);
                res = scan_files(path);
                path[i] = 0;
                if (res != FR_OK) break;
            } else {                                       /* It is a file. */
                printf("%s/%s\r\n", path, fn);
            }
        }
        f_closedir(&dir);
    }

    return res;
}
```

```
static int do_tfatfs(int argc, char ** argv)
{
    printf("\r\n****** test_fatfs start ******\r\n");
    res_flash = f_mount(&fs,"1:",1);
    /*---------------------- 格式化测试 ---------------------------*/
#if 0
    printf("\r\n****** force to format ******\r\n");
    res_flash = f_mkfs("1:", 0, 0);
    printf("format res=%d",res_flash);
#endif
    /* 如果没有文件系统就格式化创建创建文件系统 */
    if (res_flash == FR_NO_FILESYSTEM) {
        printf("need to format...\r\n");
        /* 格式化 */
        res_flash = f_mkfs("1:", 0, 0);

        if (res_flash == FR_OK) {
            printf("fromat finish\r\n");
            /* 格式化后，先取消挂载 */
            res_flash = f_mount(NULL, "1:", 1);
            /* 重新挂载 */
            res_flash = f_mount(&fs, "1:", 1);
        } else {
            printf("format fail\r\n");
            return -1;
        }
    } else if (res_flash != FR_OK) {
        printf("fail to mount spi flash(%d)\r\n", res_flash);
        printf("maybe init fail\r\n");
        return -1;
    } else {
        printf("sucess mount\r\n");
    }

    /*---------------------- 文件系统测试：写测试 ----------------------------
*/
    /* 打开文件，如果文件不存在则创建它 */
    printf("\r\n****** going to test write ******\r\n");
    res_flash = f_open(&fnew, "1:mytest3.txt", FA_CREATE_ALWAYS | FA_WRITE);
    if (res_flash == FR_OK) {
        printf("open file sucess\r\n");
        /* 将指定存储区内容写入到文件内 */
        res_flash = f_write(&fnew, WriteBuffer, sizeof(WriteBuffer), &fnum);
```

```
        if (res_flash == FR_OK) {
            printf("write sucess:%u\r\n", fnum);
            printf("write:\r\n%s\r\n", WriteBuffer);
        } else {
            printf("write fail:(%d)\r\n", res_flash);
        }
        /* 不再读写，关闭文件 */
        f_close(&fnew);
    } else {
        printf("open file fail\r\n");
    }

    /*------------------- 文件系统测试：读测试 --------------------------------
---*/
    printf("****** going to test read******\r\n");
    res_flash = f_open(&fnew, "1:mytest3.txt", FA_OPEN_EXISTING | FA_READ);
    if (res_flash == FR_OK) {
        printf("open file sucess\r\n");
        res_flash = f_read(&fnew, ReadBuffer, sizeof(ReadBuffer), &fnum);
        if (res_flash == FR_OK) {
            printf("read sucess:%u\r\n", fnum);
            printf("read:\r\n%s \r\n", ReadBuffer);
        } else {
            printf("read fail：(%d)\r\n", res_flash);
        }
    } else {
        printf("open file fail\r\n");
    }
    /* 不再读写，关闭文件 */
    f_close(&fnew);

    //getfree
    FATFS *fs2;
    DWORD fre_clust, fre_sect, tot_sect;
    f_getfree("1:", &fre_clust, &fs2);
    tot_sect = (fs2->n_fatent - 2) * fs2->csize;
    fre_sect = fre_clust * fs2->csize;
    printf("%lu bytes total drive space.\r\n%lu KiB available.\r\n", tot_sect*4096 ,
fre_sect*4096);

    //list files
    scan_files("1:");
    /* 不再使用文件系统，取消挂载文件系统 */
    f_mount(NULL, "1:", 1);
```

```
	return 0;
}

static struct command_t cmd_tfatfs = {
	.name	= "tfatfs",
	.desc	= "test fatfs",
	.usage	= usage,
	.exec	= do_tfatfs,
};

static __init void tfatfs_cmd_init(void)
{
	register_command(&cmd_tfatfs);
}

static __exit void tfatfs_cmd_exit(void)
{
	unregister_command(&cmd_tfatfs);
}

command_initcall(tfatfs_cmd_init);
command_exitcall(tfatfs_cmd_exit);
```

测试结果

```
****** test_fatfs start ******
f_mount:vol=1
sucess mount

****** going to test write ******
open file sucess
write sucess:9
write:
hello3

****** going to test read******
open file sucess
read sucess:9
read:
hello3

7057408 bytes total drive space.
7045120 KiB available.
1:/MYTEST.TXT
1:/MYTEST2.TXT
1:/MYTEST3.TXT
f_mount:vol=1
xboot: /$
```