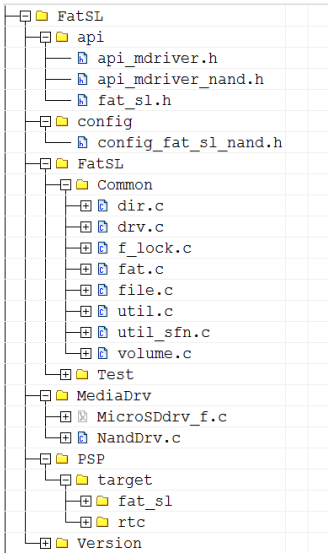
FatSL 移植笔记

一. FatSL 介绍

FatSL 是嵌入式软件供应商HCC 开发的一种FAT 类型的文件系统。FatSL 具有以下特性：

- 极低的RAM 和ROM 需求 <1K bytes RAM, 大约4K bytes ROM
- 高性能，针对FAT 风险的安全特性设计
- 支持FAT12，FAT16，FAT32 类型

FatSL 的结构如下图所示(FatSL 的软件结构设置的非常清晰，赞)



api: 介质驱动(api_mdriver.h)及文件操作(fat_sl.h)的头文件

config: FatSL 的配置信息

FatSL/Common：dir.c: 目录操作

drv.c: 调用底层驱动的接口

fat.c: 簇操作

file.c: 文件操作

volume.c：磁盘操作

util.c & util_sfn.c：内部调用的函数

FatSL/Test：一套标准的文件系统测试软件

FatSL/MediaDrv：存储介质的驱动（主要的移植内容）

PSP：应用级的移植文件（显示，RTC）

Version：版本控制，文件系统和驱动的版本

二. FatSL 的移植

如果了解 FAT 的系统结构，并理解了 FatSL 的软件结构，那么文件系统移植并不困难（其实文件 系统移植都是大同小异的）。本文给出的例子是基于STEVAL-CCM007V1 硬件平台，通过NFTL 层（NAND Flash Translation Layer）在NAND Flash 上建立该文件系统的情况。

1. 数据结构

F_PHY 主要是对存储介质的描述

```
 typedef struct
{
unsigned short number_of_cylinders; // 磁盘的参数，现在已经很少用到
unsigned short sector_per_track;
unsigned short number_of_heads;
unsigned long number_of_sectors; // 扇区数
unsigned char media_descriptor; // 介质是否支持拔插
unsigned short bytes_per_sector; // 扇区大小 512bytes
```

```
} F_PHY;
```

_F_DRIVER 就是文件系统对存储介质的驱动接口了，通过user_ptr 指针（例如指向用户自定义的存储介质的数据结构）可以为驱动实现更好的设计。

```
typedef struct F_DRIVER
{
unsigned long user_data; /* 用户自定义数据 */
void * user_ptr; /* 用户定义指针 */
/* 驱动函数 */
F_WRITESECTOR writesector;
F_READSECTOR readsector;
F_GETPHY getphy;
F_GETSTATUS getstatus;
F_RELEASE release;
} _F_DRIVER;
```

2. 驱动函数的编写

F_DRIVER *(*F_DRIVERINIT)(unsigned long ulDriverParameter);

初始化文件系统和存储介质驱动的接口，为 F_DRIVER 结构体添加内容

```
F_DRIVER *Nand_initfunc(unsigned long driver_param)
{
t_nandDrv *p;
p = nandDrv;
if (p->use)
{
return 0;
}
(void)psp_memset( p->driver, 0, sizeof( F_DRIVER ) );
p->driver->readsector = nand_readsector;
p->driver->writesector = nand_writesector;
p->driver->getphy = nand_getphy;
p->driver->getstatus = nand_getstatus;
p->driver->release = nand_release;
p->driver->user_ptr = p;
p->use = 1;
r eturn p->driver;
}
```

typedef int (*F_GETPHY)(F_DRIVER * pxDriver, F_PHY * pxPhy);

将存储介质的信息添加到 F_PHY 结构体中

```
static int nand_getphy(F_DRIVER *driver, F_PHY *phy)
{
phy->number_of_sectors = MassBlockCount;
phy->media_descriptor = F_MEDIADESC_FIX;
phy->bytes_per_sector = 512u;
return 0;
```

```
}
```

typedef int (*F_READSECTOR)(F_DRIVER * pxDriver, void * pvBuffer, unsigned long ulSector);

读扇区函数，从逻辑扇区 ulSector 中将数据读到 pvBuffer 所指向的缓存中

```
static int nand_readsector(F_DRIVER *driver, void *data, unsigned long sector)
{
uint16_t res;
t_nandDrv *p = (t_nandDrv *)(driver->user_ptr);
if(sector >= p->maxsector)
{
return 2;
}
res = NAND_Read(sector, data, 512);
if(res != NAND_OK)
return 1;
return 0;
}
```

typedef int (*F_WRITESECTOR)(F_DRIVER * pxDriver, void * pvBuffer, unsigned long ulSector);

写扇区函数，把 pvBuffer 指向的数据写入逻辑扇区 ulSector 中

```
static int nand_writesector(F_DRIVER *driver, void *data, unsigned long sector)
{
uint16_t res;
t_nandDrv *p = (t_nandDrv *)(driver->user_ptr);
if(sector >= p->maxsector)
{
return 2;
}
res = NAND_Write(sector, data, 512);
if(res != NAND_OK)
return 1;
return 0;
}
```

typedef long (*F_GETSTATUS)(F_DRIVER * pxDriver);

typedef void (*F_RELEASE)(F_DRIVER * pxDriver);

这两个函数是检测设备拔插的。

三. 经验杂谈

关于文件系统的移植，推荐大家阅读一些 FAT 结构的资料，至少要做到对 FAT 结构心中有数，对 调试会有很大的帮助的（工欲善其事，必先利其器）。

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。