AN10916

NXP LPC1700 FAT 库 EFSL 和 FatFs 移植

Rev. 2-6/7/2010 应用手册

文档信息

信息	内容
关键字	LPC1700，文件系统，EFSL，FatFs，SDC/MMC
摘要	EFSL 和 FatFs 是开发小型嵌入式系统的两个常用 FAT 库。该应用手册描述了如何在 NXP Cortex-M3 LPC1700 上移值这些 FAT 库。使用一个外部 SDC/MMC，连接到 LPC1700 SPI/SSP0，作为物理磁盘。同时，提供一套易用的 SPI 和 SDC/MMC API 函数。

1. 简介

EFSL 和 FatFs 是开发小型嵌入式系统的两个常用 FAT 库。该应用手册描述了如何在 NXP Cortex-M3 LPC1700 上移植这些 FAT 库。

提供一套易用的 SPI 和 SDC/MMC API 函数来访问 SDC/MMC，方便快捷。

该应用手册包括：

（1） 易用的 SPI 和 SDC/MMC API，通过 LPC1700 SPI 访问 SDC/MMC。

（2） 一步步地教你如何在 LPC1700 上移植 EFSL 和 FatFs。

示例软件代码，使用 2G 的金士顿微 SD 卡在 Keil MCB1700 评估板上进行测试。

2. 通过 LPC1700 SPI 访问 SDC/MMC。

2．1 SDC/MMC 介绍

安全数码卡（以下简称 SDC）/多媒体卡（以下简称 MMC）是基于 FLASH 的存储卡，专为满足下一代移动电话、消费电子设备的安全性、容量、性能和使用环境而设计。

SDC 通信是基于先进的 9 个引脚接口（时钟、命令、4*数据、3*电源线），为低电压操作而设计。SDC 主（host）接口支持常规的 MMC 操作。也有一些精简版本，像 RS-MMC、miniSD、microSD，功能相同。

SDC 和 MMC 主要区别在于初始化过程不同。

2．2 SDC/MMC 接口

SDC/MMC 接口可以很容易地集成到任何设计中，无论用的是什么微控制器。为了与现有的控制器兼容，SDC/MMC 提供了，不同于 SDC/MMC 接口的、基于标准 SPI 的通信协议。SDC/MMC 引脚分配如表 1 所示。

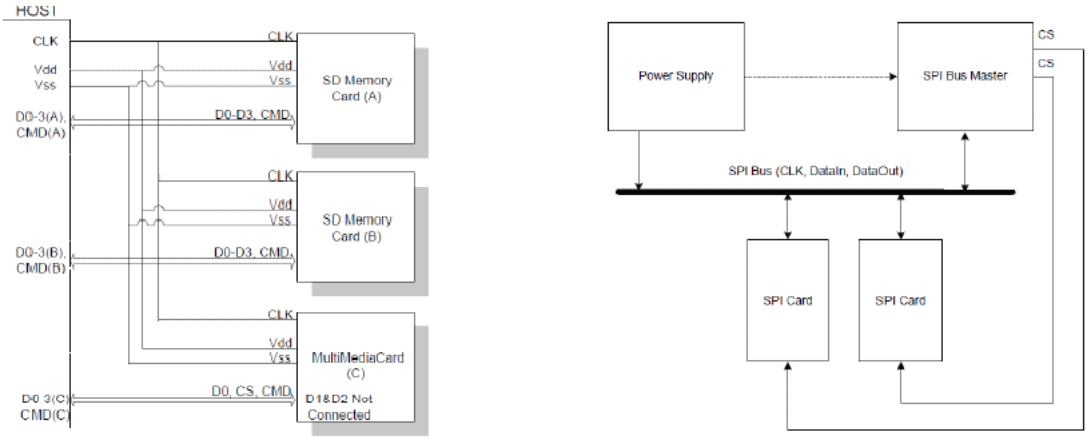
表 1 SDC/MMC 引脚分配

深圳市伟博创科技有限公司

Pin No.	Name	Type	Description
SDC/MMC Bus Mode[1]			
1	CD/DAT3	I/O, PP	Card detect/Data line [Bit 3]
2	CMD	I/O, PP	Command/Response
3	Vss1	S	Supply voltage ground
4	Vdd	S	Supply voltage
5	CLK	I	Clock
6	Vss2	S	Supply voltage ground
7	DAT0	I/O, PP	Data line [Bit 0]
8	DAT1	I/O, PP	Data line [Bit 1]
9	DAT2	I/O, PP	Data line [Bit 2]
SPI Bus Mode			
1	CS	I	Chip Select (active low)
2	DataIn	I	Host-to-card Commands and Data
3	Vss1	S	Supply voltage ground
4	Vdd	S	Supply voltage
5	CLK	I	Clock
6	Vss2	S	Supply voltage ground
7	DataOut	O	Card-to-host Data and Status
8	RSV	---	Reserved
9	RSV	---	Reserved

[1] For MMC, only one data line, DAT0, is used.

SDC/MMC 总线两种模式的拓扑结构，如图 1 所示。



a.SDC/MMC 总线模式　　b.SPI 总线模式

图 1　SDC/MMC 系统总线拓扑

因为 LPC1700 没有 SDC/MMC 本地主接口，所有只能通过 SPI 接口访问 SDC/MMC。在本部分中，只讨论 SPI 模式。

2．3　SPI 模式

2．3．1　SPI 总线拓扑

SPI 模式是 SDC/MMC 第二通信协议，这个模式是 SDC/MMC 协议的子集，用 SPI 通道进行通信，常见于 NXP 和其他厂商的微控制器中。

通过 SPI 接口，SDC/MMC 可以和大多数的微控制器配合使用。因为 SPI 模式适用于低成本的嵌入式应用，所以没有必要选择本地模式，低成本的微控制器没有本地 SDC/MMC 接口。

SDC/MMC 识别和寻址方法被硬件 CS 信号所取代。根据 CS 信号（低有效）选择卡（从）。参考图 1（b）。

在 SPI 传送期间（命令、响应和数据），CS 信号必须处于激活状态。

单向的数据输入（dataIn）和数据输出（dataOut）取代双向的 CMD 和 DAT 线。这个可以避免在读取或写数据的时候执行命令，防止多次的读/写操作。

2．3．2　SPI 总线协议

SPI 标准只是定义了物理连接，而不是完整的数据传输协议。在 SPI 模式，SDC/MMC 使用 SDC/MMC 协议、命令集的子集。

与 SDC/MMC 总线协议类似，SPI 信息由命令、响应和数据块标识符组成。主机控制主机和卡的所有通信。根据 CS 信号（低），主机开启所有总线的传输。

在以下 3 种情况下，SPI 模式的响应行为不同于 SDC/MMC 模式：

（1）　选中的卡保持对命令的响应。

（2） 使用 8 位或 16 位响应结构。

（3） 在 SDC/MMC 总线模式，当发生数据检索问题，卡将作出“错误”的响应（正常情况时，响应为数据块），而不是超时。

2．3．3 模式选择

在 SDC/MMC 模式，SDC/MMC 唤醒。如果复位命令生效（CMD0），且 CS 信号有效（负），卡将进入 SPI 模式。如果卡需要的是 SDC/MMC 模式，则它不会对命令作出应答，维持在 SDC/MMC 模式。如果需要的是 SPI 模式，卡将选择 SPI 模式。

2．4 LPC1700 SPI 接口

LPC1700 器件有一个 SPI 控制器，同步、串行、全双工通讯以及数据长度可编程。

注意：SSP0 是作为 SPI 接口的，也作为剩余的外设。在任一时间，只有一个外设可用。

SSP 可以产生比 SPI 更快的数据位。SPI 数据位最大速率达输入时钟速率的八分之一，最大的 SSP 速率（主模式）为 pclk/2。在从模式，主机提供的 SSP 时钟速率不得超过 SSP 外设时钟（在外设时钟选择寄存器中进行选择）的 1/12。

例如，如果 PCLK 设为 100MHz，SPI 最大速率为 12.5Mbit/sec（100MHz/8）。在主模式中，SSP 最大速率为 50Mbit/sec（100MHz/2），从模式时，8Mbit/sec（100MHz/12）。

在 Keil MCB1700 板上，LPC1700 SSP0 与外部 Micro SD 卡的引脚连接如图 2 所示。

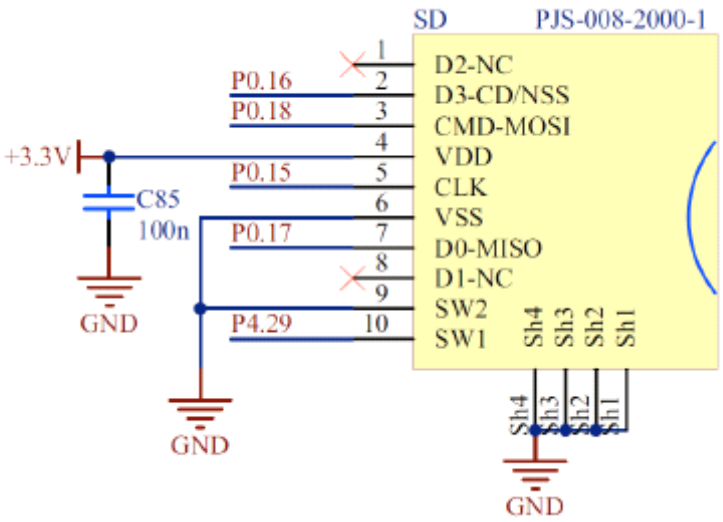


图 2 Keil MCB1700 板 SDC/MMC 的原理图

表 2 Micro SD 卡引脚分配

Pin No.	Name	Type	Description
SDC Bus Mode			
1	DAT2	I/O, PP	Data line [Bit 2]
2	CD/DAT3	I/O, PP	Card detect/Data line [Bit 3]
3	CMD	I/O, PP	Command/Response
4	Vdd	S	Supply voltage (2.7v / 3.6v)
5	CLK	I	Clock
6	Vss	S	Supply voltage ground
7	DAT0	I/O, PP	Data line [Bit 0]
8	DAT1	I/O, PP	Data line [Bit 1]
SPI Bus Mode			
1	RSV	---	Reserved
2	CS	I	Chip Select (active low)
3	DataIn	I	Host-to-card Commands and Data
4	Vdd	S	Supply voltage
5	CLK	I	Clock
6	Vss	S	Supply voltage ground
7	DataOut	O	Card-to-host Data and Status
8	RSV	---	Reserved

2．5 SPI 驱动和 APIs

lpc17xx_spi.c 提供了 SPI 通信 8 个 API 函数：

（1） void LPC17xx_SPI_Init (void);

这个 API 用来初始化 SPI 接口（通过在 LPC1700 上配置 SSP0，PCONP，GPIO，和控制寄存器）。

（2） void LPC17xx_SPI_DeInit (void) ;

这个 API 用来清除 SSP0 寄存器的初始状态，并停用 SSP0。

（3） void LPC17xx_SPI_Select (void);

这个 API 用来确认 CS（低）。

主机确认 CS 信号为低时启动总线传输，且 CS 信号必须在传输过程中检测。如果在数据传输过程的任何时间，CS 信号转为高，该传输将作废。这个信号不是由主机直接驱动。它由软件控制下的 GPIO 驱动。

（4） void LPC17xx_SPI_DeSelect (void);

这个 API 用来撤消 CS（高）以释放 SPI 总线。

（5） void LPC17xx_SPI_Release (void);
这个 API 用来释放 SPI 总线。
（6） void LPC17xx_SPI_SetSpeed (uint8_t speed);
这个 API 用来配置 SPI 数据位速率。
在 SDC/MMC 初始化期间，当数据发送时，速率通常设置为 400kMz；也可以设置为更高的速率（MMC 可达 20MHz，SDC 可达 25MHz）。

（7） void LPC17xx_SPI_SendByte (uint8_t data);
这个 API 用来发送一字节的数据（通过 SPI 总线）。

（8） uint8_t LPC17xx_SPI_RecvByte (void);
这个 API 用来接收一个字节的数据（通过 SPI 总线）。

2．6 SDC/MMC 驱动和 APIs

Lpc17xx_sd.c 提供了 4 个 API 函数用来访问 SDC/MMC。
（1） bool LPC17xx_SD_Init (void);
这个 API 用来初始化 SDC/MMC。
（2） bool LPC17xx_SD_ReadCfg (SDCFG *cfg);
这个 API 用来读取 SDC/MMC 的设置，包括寄存器 OCR、CID、CSD 和一些计算参数，像扇区数、扇区大小等等。
（3） bool LPC17xx_SD_ReadSector (uint32_t sector, uint8_t *buff, uint32_t count);
这个 API 用来在 SDC/MMC 中写入指定的数据的扇区号。

3． EFSL 和 FatFs 介绍

3．1 关于 FAT

FAT（文件配置表）文件系统（常见的 FAT12、FAT16、FAT32），是由比尔盖茨和 Marc McDonald 发明。FAT 是最主要的文件系统结构，广泛应用于大多数操作系统和存储卡。

FAT 用来管理磁盘。这个名称源于这个表的用途，这个表汇集磁盘的相关信息，包括哪些区域属于文件，哪些是空闲的，哪些不可用，以及每个文件在磁盘的存储位置。为了限制表的大小，磁盘空间分配为几个“文件”——几个连续的硬件扇区组，称为簇。磁盘驱动升级，簇的数目将增加，包括位的宽度。FAT 格式的几个版本都是根据表的位的宽度来命名：12、16 和 32。FAT 标准也有了扩展，保证与现有软件兼容的前提下。
更多 FAT 以及许可、版板信息，请参考 http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx

深圳市伟博创科技有限公司

3．2 关于 EFSL

嵌入式文件系统库（EFSL）用于各种嵌入式系统。当前 EFSL 支持微软 FAT 文件系统。EFSL 就是为了能在所有 C 编译器上编译纯 ANSI C 代码。在硬件上增加代码是最直接的；只要加入代码（获取或写入一个 512 字节的扇区），剩下的就由库来完成。可以使用现有的代码，只有当你的硬件没有目标存在时，才使用你自己的代码。例如，它支持 SPI 模式下的安全数码卡。

许可、免责条款、版权等，更多信息请参考 http://efsl.be/

3．3 关于 FatFs

FatFs 是用于小型嵌入式系统的 FAT 文件系统模块。FatFs 遵循 ANSI C 规范而编写，与磁盘 I/O 层完全独立；它是独立的硬件架构。它可以整合到低成本的微控制器上，不需要任何改动。

FatFs 有以下特点：

（1） 兼容 Windows FAT12/16/32 文件系统
（2） 与平台无关；易于移植
（3） 代码小，占用空间小
（4） 配置选项：
A． 多卷（物理驱动和分区）
B． 多 OEM 字符集包括 DBCS（Multiple OEM code pages including DBCS）
C． 长文件名，支持 OEM 代码和 Unicode（Long File Name (LFN) support in OEM code or Unicode）
D． 支持 RTOS
E． 支持多扇区大小
F． 只读、最小化 API、I/0 缓冲，等等

FatFs 模块是用于教育、科研、开发的免费软件。你可以使用、修改和/或重新分配，只限于个人、非营利使用。

更多关于 FatFs、许可、版权，请参考
http://elm-chan.org/fsw/ff/00index_e.html

深圳市伟博创科技有限公司

4. LPC1700 EFSL 移植

4.1 EFSL 结构

EFSL 内部结构如图 3 所示：

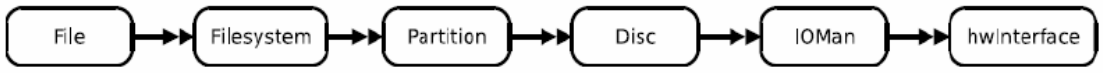


图 3 EFSL 结构

EFSL 创造了简单的目标模型。文件系统对象（Filesystem object）解决了文件系统特定的问题。分区对象（Partition object）则把分区相关寻址转为磁盘 LBA 寻址。磁盘对象保存分区表，且有一个直接的链接连到缓存管理、IOMan。在 IOMan，汇集所有的磁盘扇区的请求。IOMan 将检查扇区是否须要从磁盘读取（或从存储器），或回写磁盘。在后一种情况（读或写磁盘），硬件层产生一个请求。

硬件接口有三个功能：
（1） 初始化硬件
（2） 从磁盘读扇区
（3） 磁盘写扇区

所有的请求都是基于扇区。扇区是磁盘的一块 512 字节的区域，与另一 512 字节边界对齐。

LPC1700 EFSL 移值是相当简单的，增加获取或写 512 字节的扇区的代码，剩下的就由库来完成。

本部分剩下内容将描述一步一步移植 EFSL（0.2.8 版本）到 LPC1700。

4.2 设置基础框架

4.2.1 定义终端的名字

你需要在源代码中用这个名字建立一个相应的定义。在这个工程中，这个名字为：HW_ENDPOINT_LPC17xx_SD，定义在 config.h 文件中。

深圳市伟博创科技有限公司

```
 * Here you will define for what hardware-endpoint EFSL should be compiled.
 * Look in interfaces.h to see what systems are supported, and add your own
 * there if you need to write your own driver. Then, define the name you
 * selected for your hardware there here. Make sure that you only select one
 * device!
*/
    /*#define HW_ENDPOINT_LINUX*/
    /*#define HW_ENDPOINT_ATMEGA128_SD*/
    //#define HW_ENDPOINT_LPC2000_SD
    /* defines the interface for LPC213x (0=SPI0 1=SPI1) */
    // #define HW_ENDPOINT_LPC2000_SPINUM  (0)
    //#define HW_ENDPOINT_LPC2000_SPINUM  |1|
    /*#define HW_ENDPOINT_DSP_TI6713_SD*/
    /* define the interface for LPC17xx SSP0 */
    #define HW_ENDPOINT_LPC17xx_SD
```

图 4 LPC17XX 名字的定义

4．2．2 定义整型的大小

打开 inc/types.h，创建一个新的入口。如果设置跟你的一样，你可以直接复制、粘贴。

```
typedef char eint8;
typedef signed char esint8;
typedef unsigned char euint8;
typedef short eint16;
typedef signed short esint16;
typedef unsigned short euint16;
typedef int eint32;
typedef signed int esint32;
typedef unsigned int euint32;
```

图 5 EFSL 整型的定义

4．2．3 在 interface.h 中增加你的终端

在 inc/interface.h 中增加新的入口。

```
#if defined(HW_ENDPOINT_LINUX) || defined(HW_ENDPOINT_LINUX64)
    #include "interfaces/linuxfile.h"
#elif defined(HW_ENDPOINT_ATMEGA128_SD)
    #include "interfaces/atmega128.h"
#elif defined(HW_ENDPOINT_DSP_TI6713_SD)
    #include "interfaces/dsp67xx.h"
#elif defined(HW_ENDPOINT_LPC2000_SD)
    #include "interfaces/lpc2000_spi.h"
#elif defined(HW_ENDPOINT_LPC17xx_SD)
    #include "interfaces/if_lpc17xx.h"
#else
    #error "NO INTERFACE DEFINED - see interface.h"
#endif
```

图 6 在 interface.h 中增加终端

4．2．4 配置 EFSL

配置文件（\efsl\conf\config.h）定义了库的作用。在配置文件中，大部分设置默认为安全或者标准的设置。

该工程使用的配置文件如表 3 所示。

表 3 EFSL 配置

Item	Configuration	Description
Hardware target	#define HW_ENDPOINT_LPC17xx_SD	Access SDC/MMC via LPC17xx SSP0
Memory	/* #define BYTE_ALIGNMENT */[1]	Specify that the MCU can not access memory byte oriented
Cache	#define IOMAN_NUMBUFFER 6 #define IOMAN_NUMITERATIONS 3 #define IOMAN_DO_MEMALLOC	6x512 byte (3 KB) RAM used for cache
Cluster pre-allocation	#define CLUSTER_PREALLOC_FILE 2 #define CLUSTER_PREALLOC_DIRECTORY 0	The number of clusters pre-allocated when writing files.
Endianess	#define LITTLE_ENDIAN	All FAT structures are stored in Intel little endian order
Date and Time support	/*#define DATE_TIME_SUPPORT*/	Disable date and time support
Error reporting support	#define FULL_ERROR_SUPPORT	Enable error recording for all object
List options	#define LIST_MAXLENFILENAME 12	Configure what kind of data you will get from directory listing requests
Debugging	/* #define DEBUG */	Disable debugging behavior

[1] Being commented out means the macro is not defined.

4．2．5 建立源文件

在 inc/interfaces 中建立头文件，src/interfaces 中建立源文件。在这个工程中，我们使用 lpc17xx_spi.h, lpc17xx_sd.h, lpc17xx_spi.c and lpc17xx_sd.c。
Lpc17xx_spi.c(h)包含 API，通过 LPC1700 上的 SSP0 进行通信。
Lpc17xx_sd.c(h)包含 API，通过 LPC1700 上的 SSP0 访问 SDC/MMC。

4．3 实现低级功能

4．3．1 硬件接口（hwInterface）

这个结构代表了底层硬件。有一些区域需要呈现出来（EFSL 有用到），但你可以根据你的驱动需要访问硬件。。

```
/*************************************************************\
              hwInterface
               ----------
* long      sectorCount     Number of sectors on the file.
\*************************************************************/
struct  hwInterface{
    euint32     sectorCount;
};
typedef struct hwInterface hwInterface;
```

图 7 硬件接口结构

4．3．2 If_initInterface

当 efs_init()初始化硬件，这个函数将调用一次。这个代码将让硬件进入待使用状态。

建议，但不是必须，在结构中填充扇区数区域。

```
esint8 if_initInterface(hwInterface* file, eint8* opts)
{
    SDCFG SDCfg;

    if (LPC17xx_SD_Init() == false)
        return (-1);
    if  (LPC17xx_SD_ReadCfg(&SDCfg) == false)
        return (-2);

    file->sectorCount = SDCfg.sectorcnt;

    return 0;
}
```

图 8 执行 if_initInterface

4．3．3 If_readBuf

这个函数从磁盘读取扇区，并保存在缓存中。务必小心缓存的边界问题，调用这个函数

时它通常是 IOMan。如果缓存溢出，将会影响到下一级的缓存，有可能会引起回塑。

```
/*
    read a sector from the disc and store it in a user supplied buffer.

    note that there is no support for sectors that are not 512 bytes large
*/

esint8 if_readBuf(hwInterface* file,euint32 address,euint8* buf)
{
    if (LPC17xx_SD_ReadSector (address, buf, 1) == true)
        return 0;
    else
        return (-1);
}
```

图 9 执行 if_readBuf

这是一个 LBA 地址，与磁盘的起始部分相关联。当访问一个旧的磁盘或使用其他寻址方式的设备，你必须重新计算地址。请注意，不支持非 512 字节的扇区。

4．3．4 If_writeBuf

这个函数类似于读函数。

```
/*
    write a sector.

    note that there is no support for sectors that are not 512 bytes large.
*/

esint8 if_writeBuf(hwInterface* file,euint32 address,euint8* buf)
{
    if ( LPC17xx_SD_WriteSector(address, buf, 1) == true)
        return 0;
    else
        return (-1);
}
```

图 10 执行 if_writeBuf

4．4 Demo

建立一个 Keil uVision4 工程，并添加所有相关源文件。
Main.c 是测试文件，列出了根目录所有文件，打开一个文件，然后在文件的末尾增加一行。这个 Demo 在 KEIL MCB1700 评估板上测试过。更多关于 MCB1700 的信息，请参考 http://www.keil.com/mcb1700/。

PC 终端和 MCB1700 的串口通信，采用 Tera term（或其他工具），相关参数为：波特率 115200，8 位，无奇偶校验，1 个停止位，XON/XOFF。

测试中使用 2GB 金士顿 微 SD 卡。

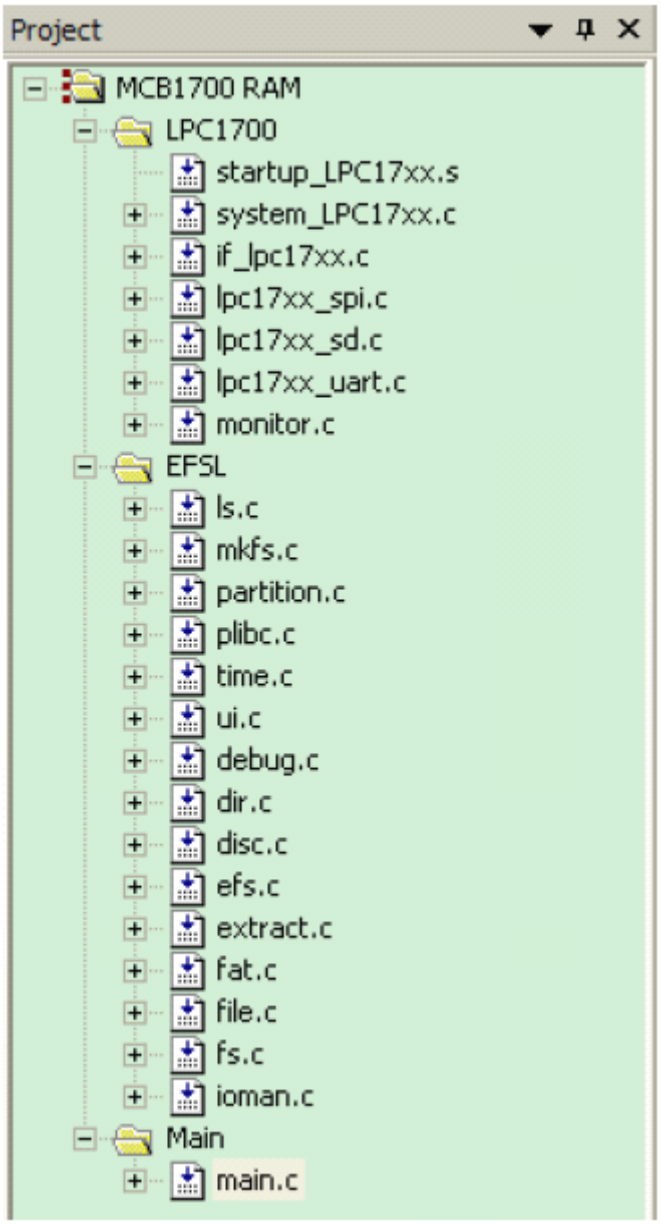
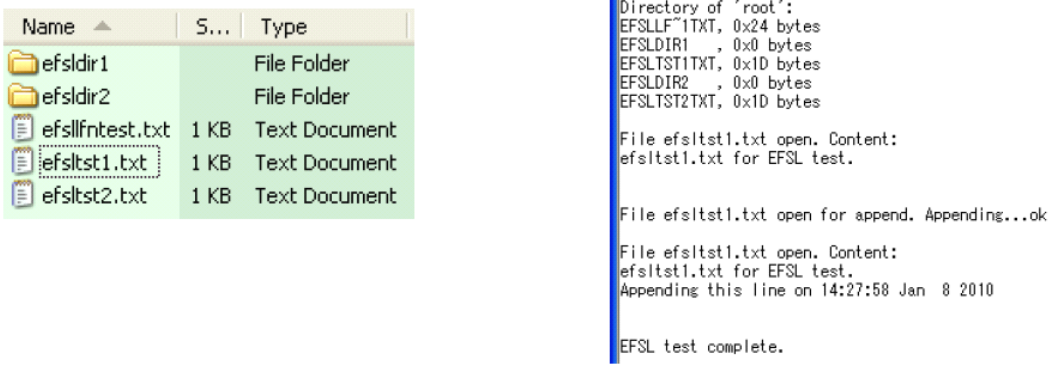


图 11 工程源文件

SDC/MMC 根目录和 COM 输出的文件结构在后面说明。

注意：因为 EFSL 不支持长文件名（LFN），文件“efsllfntest.txt”显示为“efsllf~1.txt”。



a. Files on root directory　　b. COM output

图 12 Demo 结果

5. LPC1700 FatFs 移植

5.1 FatFs 结构

FatFs 结构如图 13 所示。

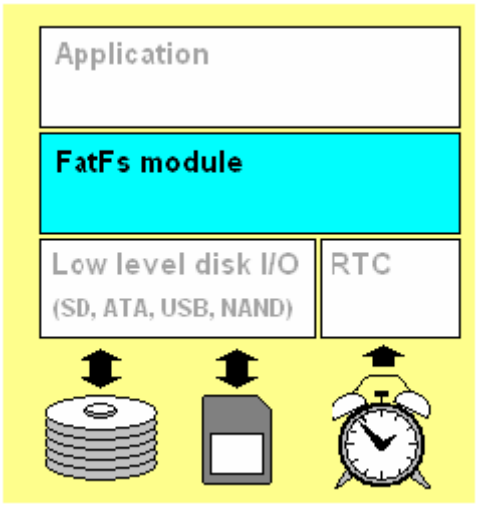


图 13 FatFs 结构

FatFs 模块是一个中间件，提供了多个函数访问 FAT 卷，像 as f_open, f_close, f_read, f_write, etc (参考 ff.c)。无需依赖于模块的平台，只要编译器符合 ANSI C。

使用一个低级磁盘 I/O 模块读/写物理磁盘。
RTC 模块用来获取实时时间（current time）。

低级磁盘 I/O 和 RTC 模块完全独立于 FatFs 模块。它们由用户提供，这是移植 FatFs 到其他平台的主要任务。

剩下的部分将一步一步地说明如何移植 FatFs（0.07e 版）到 LPC1700。

5．2 定义整型大小

FatFs 模块设定 char/short/long 的大小分别为 8/16/32 位，int 为 16 或 32 位。这些定义在 integer.h 文件中。在大部分编译器中这些都不会有问题。任何与现有的定义相抵触的情况都要小心处理。

```
/* These types must be 16-bit, 32-bit or larger integer */
typedef int             INT;
typedef unsigned int    UINT;

/* These types must be 8-bit integer */
typedef signed char     CHAR;
typedef unsigned char   UCHAR;
typedef unsigned char   BYTE;

/* These types must be 16-bit integer */
typedef short           SHORT;
typedef unsigned short  USHORT;
typedef unsigned short  WORD;
typedef unsigned short  WCHAR;

/* These types must be 32-bit integer */
typedef long            LONG;
typedef unsigned long   ULONG;
typedef unsigned long   DWORD;
```

图 14 FatFs 模块整型定义

5．3 FatFs 模块配置

所有的配置和细节都可以在 ffconf.h 中查看（FatFs 0.07e 版）。

该工程的配置所表 4 所示。

深圳市伟博

Table 4. Configurations of FatFs module in this project

Item	Configuration	Description
Function and Buffer Configurations	#define _FS_TINY 0	Use the sector buffer in the individual file data transfer.
	#define _FS_READONLY 0	Enable both read and write functions.
	#define _FS_MINIMIZE 0	Enable full function.
	#define _USE_STRFUNC 0	Disable string functions.
	#define _USE_MKFS 1	Enable f_mkfs function
	#define _USE_FORWARD 0	Disable f_forward function
Locale and Namespace Configurations	#define _CODE_PAGE 850	OEM code page "Multilingual Latin 1" will be used on the target system.
	#define _USE_LFN 1	Enable LFN
	#define _MAX_LFN 255	Maximum LFN length to handle
	#define _LFN_UNICODE 0	Disable Unicode.
	#define _FS_RPATH 1	Enable the relative path feature and f_chdir and f_chdrive function are available.
Physical Drive Configurations	#define _DRIVES 1	Only 1 physical driver is allowed.
	#define _MAX_SS 512	Maximum sector size to be handled

Item	Configuration	Description
	#define _MULTI_PARTITION 0	Each volume is bound to the same physical drive number and can mount only first primary partition.
System Configurations	#define _WORD_ACCESS 0	Enable the Byte-by-byte access
	#define _FS_REENTRANT 0	Disable reentrancy.

5．3．1 _USE_LFN

FatFs 模块 0.07e 版本支持长文件名（LFN）。两种不同文件名（SFN 和 LFN）在函数中是一样的，除了 f_readdir 函数。要使用 LFN，设置_USE_LFN 为 1 或 2，并增加 Unicode 码转换函数 ff_convert 和 ff_wtoupper 到工程中。这个函数在 option\cc*.c。

注意：FAT 文件系统的 LFN 是微软的专利。商业应用须要得到微软的许可。

5．3．2 _CODE_PAGE

_CODE_PAGE 指定用于目标系统的 OEM 字符集（OEM code page）。

当使用 LFN，模块的大小取决于所选择的字符集（code page）。表 5 所示为不同字符集的 LFN 模块大小的不同之外。中国和韩国语言有上万个字符，需要很大的 OEM-Unicode 双向转换表；因此，模块大小急剧增加，如表 5 所示。因此，受限于 ROM 的大小，带 LFN 的 FatFs 在一些微控制器下不可用。

Table 5. ROM size increase with different code pages on Cortex-M3

Code page	ROM size increase (byte)
SBSC	2796
CP932 (Japanese Shift-JIS)	61656
CP936 (Simplified Chinese GBK)	176856
CP949 (Korean)	138912
CP950 (Traditional Chinese Big5)	110544

[1] Compiler: armcc V4.0.0 Optimization: O3

5．4 低级函数

因为 FatFs 模块完全独立于 disk I/O 和 RTC 模块，它需要下面这个函数来读/写物理磁盘，并获得实时时间（current time）。因为，低级 disk I/O 和 RTC 模块不属于 FatFs 模块，需要由用户自己提供。

5．4．1 disk_initialize

disk_initialize 函数初始化物理驱动器。在 FatFs 模块 volume mount 处理过程中调用这个函数来管理媒体的变更（media change）。当 FatFs 激活时，应用程序不要调用这个函数；FAT 结构可能会失效。重新初始化文件系统，使用 f_mount 这个函数。

```
/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/
DSTATUS disk_initialize (
    BYTE drv        /* Physical drive number (0) */
)
{
    if (drv) return STA_NOINIT;          /* Supports only single drive */
//  if (Stat & STA_NODISK) return Stat; /* No card in the socket */

    if (LPC17xx_SD_Init() && LPC17xx_SD_ReadCfg(&SDCfg))
        Stat &= ~STA_NOINIT;

    return Stat;
}
```

图 15

5．4．2 disk_status

disk_status 这个函数返回当前磁盘状态，由以下几个标志组成：

STA_NOINIT：指示磁盘驱动未初始化。

STA_NODISK：指示驱动没有媒介

STA_PROTECTED：指示媒介被写保护。因为 MCB1700 板没有提供卡的检测和写保护，我们忽视两个标志：STA_NODISK 和 STA_PROTECTED。

```
/*-----------------------------------------------------------------------*/
/* Get Disk Status                                                       */
/*-----------------------------------------------------------------------*/
DSTATUS disk_status (
    BYTE drv        /* Physical drive number (0) */
)
{
    if (drv) return STA_NOINIT;     /* Supports only single drive */

    return Stat;
}
```

图 16 disk_status

5．4．3 disk_read

disk_read 函数从磁盘驱动器读取一个或多个扇区。

```
/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/
DRESULT disk_read (
    BYTE drv,           /* Physical drive number (0) */
    BYTE *buff,         /* Pointer to the data buffer to store read data */
    DWORD sector,       /* Start sector number (LBA) */
    BYTE count          /* Sector count (1..255) */
)
{
    if (drv || !count) return RES_PARERR;
    if (Stat & STA_NOINIT) return RES_NOTRDY;

    if (LPC17xx_SD_ReadSector (sector, buff, count) == true)
        return RES_OK;
    else
        return RES_ERROR;
}
```

图 17 disk_read

5．4．4 disk_write

disk_write 函数在磁盘写一个或多个扇区。

在只读配置中不需要这个函数。

```
/*-----------------------------------------------------------------------*/
/* Write Sector(s)                                                       */
/*-----------------------------------------------------------------------*/
#if _READONLY == 0
DRESULT disk_write (
    BYTE drv,           /* Physical drive number (0) */
    const BYTE *buff,   /* Pointer to the data to be written */
    DWORD sector,       /* Start sector number (LBA) */
    BYTE count          /* Sector count (1..255) */
)
{
    if (drv || !count) return RES_PARERR;
    if (Stat & STA_NOINIT) return RES_NOTRDY;
//  if (Stat & STA_PROTECT) return RES_WRPRT;

    if ( LPC17xx_SD_WriteSector(sector, buff, count) == true)
        return RES_OK;
    else
        return  RES_ERROR;

}
#endif /*  READONLY == 0 */
```

图 18 disk_write

5．4．5 disk_ioctl

disk_ioctl 这个函数控制器件特定的特性和功能（非磁盘读/写）。

深圳市伟博创科技

Table 6. Supported commands in disk_ioctl functions

Command	Description
Device independent	
CTRL_SYNC	Ensures that the disk drive has finished pending write process. When the disk I/O module has a write back cache, flush the dirty sector immediately. This command is not required in read-only configuration
GET_SECTOR_SIZE	Returns sector size of the drive into the WORD variable pointed by Buffer. This command is not required in single sector size configuration, _MAX_SS is 512.
GET_SECTOR_COUNT	Returns total sectors on the drive into the DWORD variable pointed by Buffer. This command is used in only f_mkfs function.
GET_BLOCK_SIZE	Returns erase block size of the memory array in unit of sector into the DWORD variable pointed by Buffer. This command is used in only f_mkfs function.
Device dependent	
MMC_GET_TYPE	Get card type flags (1 byte)
MMC_GET_CSD	Receive CSD as a data block (16 bytes)
MMC_GET_CID	Receive CID as a data block (16 bytes)
MMC_GET_OCR	Receive OCR as an R3 response (4 bytes)
MMC_GET_SDSTAT	Receive SD status as a data block (64 bytes)

Please refer to the software example for the detailed implementation of these functions.

5. 4. 6 get_fattime

get_fattimep 这个函数获取实时时间，在只读配置中不需要这个函数。

```
/*-----------------------------------------------------------*/
/* User Provided RTC Function for FatFs module               */
/*-----------------------------------------------------------*/
/* This is a real time clock service to be called from       */
/* FatFs module. Any valid time must be returned even if     */
/* the system does not support an RTC.                       */
/* This function is not required in read-only cfg.           */
DWORD get_fattime ()
{
    RTCTime rtc;

    /* Get local time */
    rtc_gettime(&rtc);

    /* Pack date and time into a DWORD variable */
    return    ((DWORD)(rtc.RTC_Year - 1980) << 25)
            | ((DWORD)rtc.RTC_Mon << 21)
            | ((DWORD)rtc.RTC_Mday << 16)
            | ((DWORD)rtc.RTC_Hour << 11)
            | ((DWORD)rtc.RTC_Min << 5)
            | ((DWORD)rtc.RTC_Sec >> 1);
}
```

图 19 get_fattime

5．5 Demo

这个 Demo 也是在 Keil’s MCB1700 评估版上进行测试，使用相同的 2GB 金士顿微 SD 卡和 COM 配置。

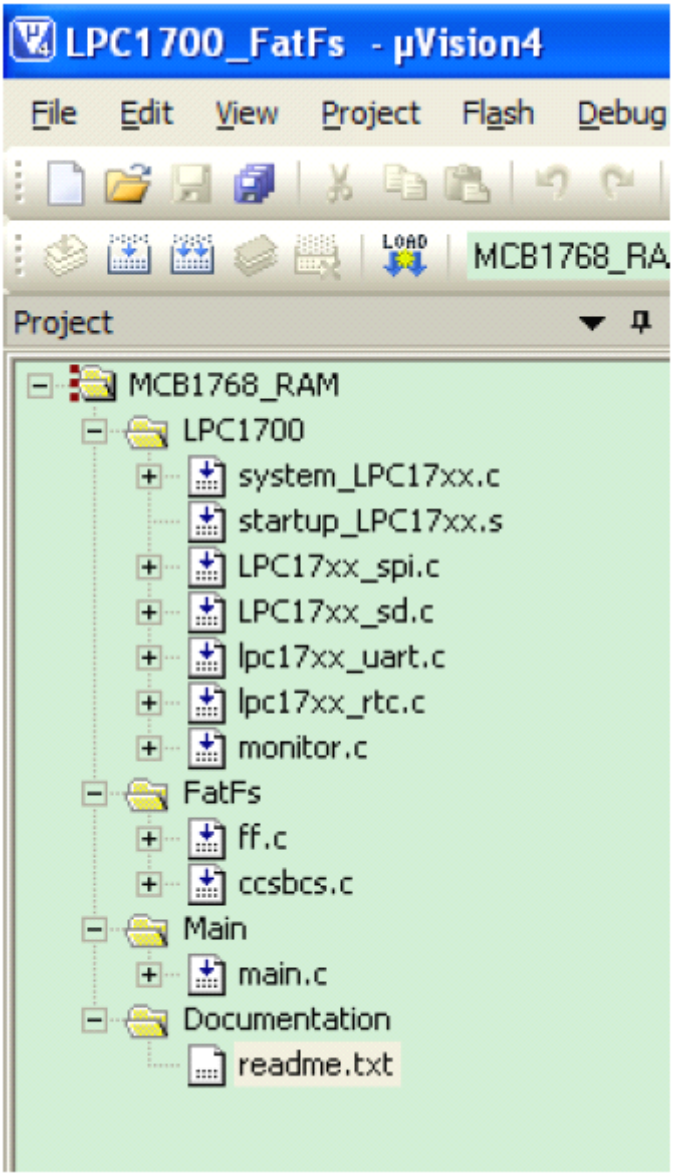
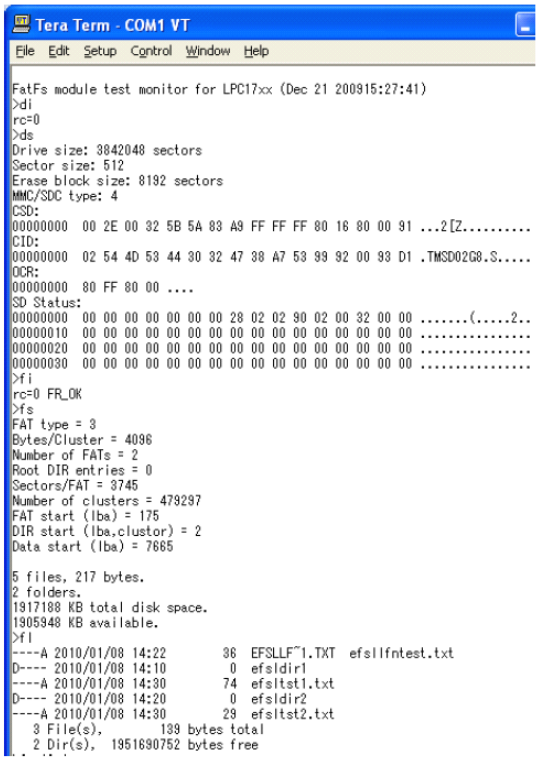


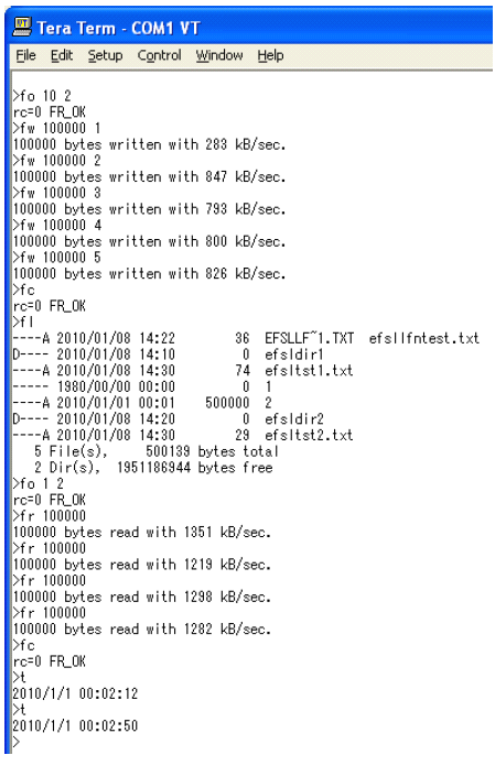
图 20 工程源文件

Table 7. Supported commands for FatFs

Command	Description
Disk functions	
Di	Initialize the disk
Ds	Show disk status
dd [<lba>]	Dump a specific sector
File functions	
fi	Force initialize the logical drive
fs	Show logical drive status
fl [<path>]	Directory listing
fo <mode> <file>	Open a file
fc	Close a file
fe	Seek file pointers
fd <len>	Read and dump file from current fp
fr <len>	Read file
fw <len> <val>	Write file
fn <old_name> <new_name>	Change file/dir name
fu <name>	Unlink a file or dir
fv	Truncate file
fk <name>	Create a directory
fa <atrr> <mask> <name>	Change file/dir attribute
ft <year> <month> <day> <hour> <min> <sec> <name>	Change timestamp
fx <src_name> <dst_name>	Copy file
fg <path>	Change current directory
fj <drive#>	Change current drive
fm <partition rule> <cluster size>	Create file system
fz [<rw size>]	Change R/W length for fr/fw/fx command
Time functions	
t [<year> <mon> <mday> <hour> <min> <sec>]	Get or set the current date and time



a. Card and file system information test



b. Card R/W and RTC test

图 21 Demo 输出

6. 参考文献

[1] NXP LPC17xx User Manual UM10360 (Rev. 00.07), NXP Semiconductors, July 31, 2009

[2] SanDisk SD Card Product Manual (Version 2.2), SanDisk Corporation. Nov, 2004

[3] The MultiMediaCard System Specification, Version 3.1, MMC Association Technical Committee, June 2001.