AN10916

FAT library EFSL and FatFs port on NXP LPC1700

Rev. 3 — 1 May 2011 Application note

Document information

Info	Content
Keywords	LPC1700, Cortex-M3, File system, EFSL, FatFs, SDHC
Abstract	EFSL and FatFs are two widely used FAT libraries for developing small embedded systems. This application note describes how to port these two FAT libraries to NXP Cortex-M3 LPC1700 devices. External SDC/MMC connected to LPC1700 SPI/SSP0 will be used as physical disk. SDHC is also supported.

Revision history

Rev	Date	Description
3	20110501	• Removed section “Access SDC/MMC via SPI on LPC1700” since it is described in AN11070. • Modified some test results since FatFs was updated from 0.07e to 0.08a and added support for SDHC.
2	20100706	• Added text “and applicable licenses and/or copyrights” to sentence regarding URLs for FAT, EFSL, and FatFs.
1	20100304	• Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

EFSL and FatFs are two widely used FAT libraries for developing small embedded system nowadays. This application note describes how to port these two FAT libraries to NXP Cortex-M3 LPC1700 devices.

A set of easy-to-use SPI and SDC/MMC API functions is also provided to access SDC /MMC conveniently as a physical disk.

This application note describes how to port EFSL and FatFs to LPC1700 in a step by step manner.

The sample software is tested on Keil's MCB1700 evaluation board (LPC1768) with 2/4/8 GB SanDisk Micro SDC/SDHC cards.

2. EFSL and FatFs introduction

2.1 About FAT

The FAT (File Allocation Table, also known as FAT12, FAT16 and FAT32) file system was developed by Bill Gates and Marc McDonald. It is the primary file system architecture now widely used on most operating systems and memory cards.

FAT was created for managing disks efficiently. The name originates from the usage of a table which centralizes the information about which areas belong to files, are free or possibly unusable, and where each file is stored on the disk. To limit the size of the table, disk space is allocated to files in contiguous groups of hardware sectors called **clusters**. As disk drives have evolved, the maximum number of clusters has dramatically increased, and so the number of bits used to identify each cluster has grown. The successive major versions of the FAT format are named after the number of table element bits: 12, 16, and 32. The FAT standard has also been expanded in other ways while preserving backward compatibility with existing software.

For more information about FAT and applicable licenses and/or copyrights, please go to:

http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx

2.2 About EFSL

The Embedded File Systems Library (EFSL) project aims to create a library for file systems, to be used on various embedded systems. Currently EFSL supports the Microsoft FAT file system family. It is EFSL's intent to create pure ANSI C code that compiles on anything that bears the name 'C compiler'.

Adding code for specific hardware is straightforward, just add code that fetches or writes a 512 byte sector, and the library will do the rest.. For example, it supports secure digital cards in SPI mode.

This project is released under the regular Public License with an exception clause. This clause states that users are allowed to statically link against the library without having to license proprietary code as GPL as well.

For more information about EFSL please refer to

http://efsl.be/

2.3 About FatFs

FatFs is a generic FAT file system module for small embedded systems. The FatFs is written in compliance with ANSI C and completely separated from the disk I/O layer. Therefore it is independent of hardware architecture. It can be incorporated into low cost microcontrollers, such as AVR, 8051, PIC, ARM, Z80, etc., without any change.

The FatFs has the following features:

- Windows compatible FAT12/16/32 file system.
- Platform independent. Easy to port.
- Very small footprint for code and work area.
- Various configuration options:
  - Multiple volumes (physical drives and partitions).
  - Multiple OEM code pages including DBCS.
  - Long file name (LFN) support in OEM code or Unicode.
  - RTOS support.
  - Multiple sector size support.
  - Read-only, minimized API, I/O buffer and etc.

The FatFs module is free software opened for education, research and development. It is ok to modify and/or redistribute it for personal, non-profit use or commercial products without any restriction.

For more information about FatFs please refer to

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

3. EFSL port on LPC1700

3.1 EFSL structure

The EFSL internal structure is shown below:

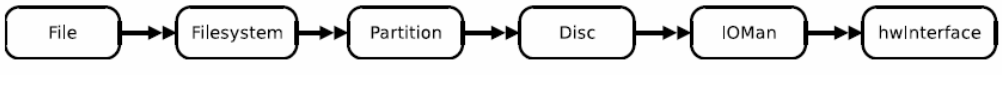

Fig 1. EFSL structure

EFSL has created a linear object model that is quite simple. The ***Filesystem*** object deals with handling the file system specific stuff. The ***Partition*** object is responsible for translating partition relative addressing into disc-based LBA addressing. The ***Disc*** object holds the partition table, and has a direct link to a cache manager, IOMan. In ***IOMan***, all requests for disc sectors come together. IOMan will perform checks to see if sectors have to be read from disc (or from memory), or written back to disc. In the latter case (reading or writing to disc), a request is made to the ***hardware*** layer.

The hardware interface has three responsibilities:

1. Initialize the hardware
2. Read sectors from disc
3. Write sectors to disc

All requests are sector-based. A sector is a 512 byte piece from the disc, which is aligned to a 512 byte boundary.

EFSL port on LPC1700 is rather straightforward, just adding code that fetches or writes a 512 byte sector, and the library will do the rest.

The rest of this section will describe step by step how to port EFSL (revision 0.2.8) to LPC1700.

3.2 Setup basic framework

3.2.1 Define a name for the endpoint

The endpoint name is needed to create the required defines in the source code. In this project, the name is **HW_ENDPOINT_LPC17xx_SD** which is defined in config.h:

```
* 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
```

Fig 2. Name definition for LPC17xx

3.2.2 Define the sizes of integer types

Open inc/types.h and create a new entry.

```
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;
```

Fig 3. Integer types definitions for EFSL

3.2.3 Add an endpoint to interface.h

Add the new entry in inc/interface.h.

```
#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
```

Fig 4. Add endpoint in interface.h

3.2.4 Configure EFSL

The configuration file (\efsl\conf\config.h) defines the behavior of the library. In the configuration files there are many settings, most of which default to safe or standard compliant settings.

The configurations used in this project are listed below.

Table 1. Configurations of EFSL in this project

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	6x512byte (3kB) 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 returned from directory listing requests
Debugging	/* #define DEBUG */	Disable debugging behavior

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

3.2.5 Create source files

Create header files in inc/interfaces and source files in src/interfaces. In this project, the fileslpc17xx_spi.h, lpc17xx_sd.h, lpc17xx_spi.c and lpc17xx_sd.c are used.

Lpc17xx_spi.c(h) includes APIs to communicate via SSP0 on the LPC1700.

Lpc17xx_sd.c(h) includes APIs to access SDC/MMC via SSP0 on the LPC1700.

3.3 Implement low level functions

3.3.1 hwInterface

This structure represents the underlying hardware. There are some fields that are required to be present (because EFSL uses them). As always in embedded design it is recommended to keep this structure as small as possible.

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

Fig 5. Definition of structure hwInterface

3.3.2 If_initInterface

This function will be called one time, when the hardware object is initialized by efs_init(). This code should bring the hardware in a ready to use state.

Optionally (but recommended) the sectorCount filed in the structure hwInterface should be filled in.

```
esint8 if_initInterface(hwInterface* file, eint8* opts)
{
    if (SD_Init() == SD_FALSE)
        return (-1);
    if  (SD_ReadConfiguration() == SD_FALSE)
        return (-2);

    file->sectorCount = CardConfig.sectorcnt;

    return 0;
}
```

Fig 6. Implementation of if_initInterface

3.3.3 If_readBuf

This function is used to read a sector from the disc and store it in a user supplied buffer.

Be very careful to respect the boundaries of the buffers, since it will usually be IOMan calling this function. If there is a buffer overflow, corruption of the cache of the next buffer may occur, which in turn may produce extremely rare and impossible to retrace behavior.

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

Fig 7. Implementation of if_readBuf

The address is a LBA address, relative to the beginning of the disc. If accessing an old hard disc, or a device which uses some other form of addressing to the address, it will have to be recalculated based on the chosen addressing scheme. Please note that there is no support for sectors that are not 512 bytes in size.

3.3.4 If_writeBuf

The function works exactly the same as its reading variant.

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

Fig 8. Implementation of if_writeBuf

3.4 Demo

Create a Keil uVision4 project and add all related source files.

Main.c is the test file. It will list all files in the root directory, write and read specified length of data from/to a file. The R/W speed will also be calculated.

This demo is tested on the KEIL MCB1700 evaluation board. For more information about MCB1700, please refer to: http://www.keil.com/mcb1700/.

Tera Term (or a similar tool) is used for serial communication between PC terminal and MCB1700 and configured at 115200 baud, 8-bits, no parity, 1 stop bit, XON/XOFF.

2/4/8 GB SanDisk Micro SD/SDHC cards are used for the test.

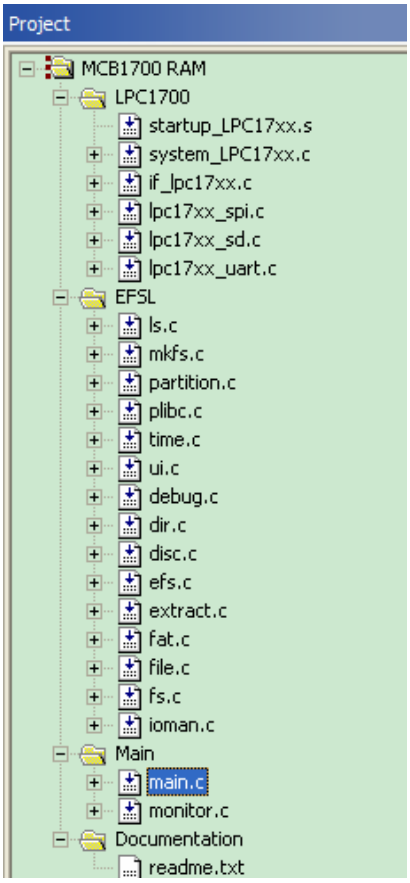

Fig 9. Source files of the project

Below is the file structure in root directory of a 4 GB Micro SDHC card.

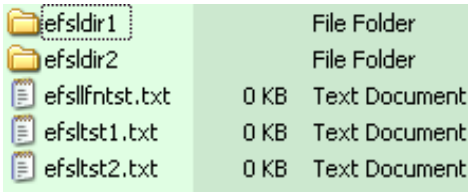

Fig 10. Files on root directory

Below is the COM output of the demo with the 4 GB Micro SDHC card.

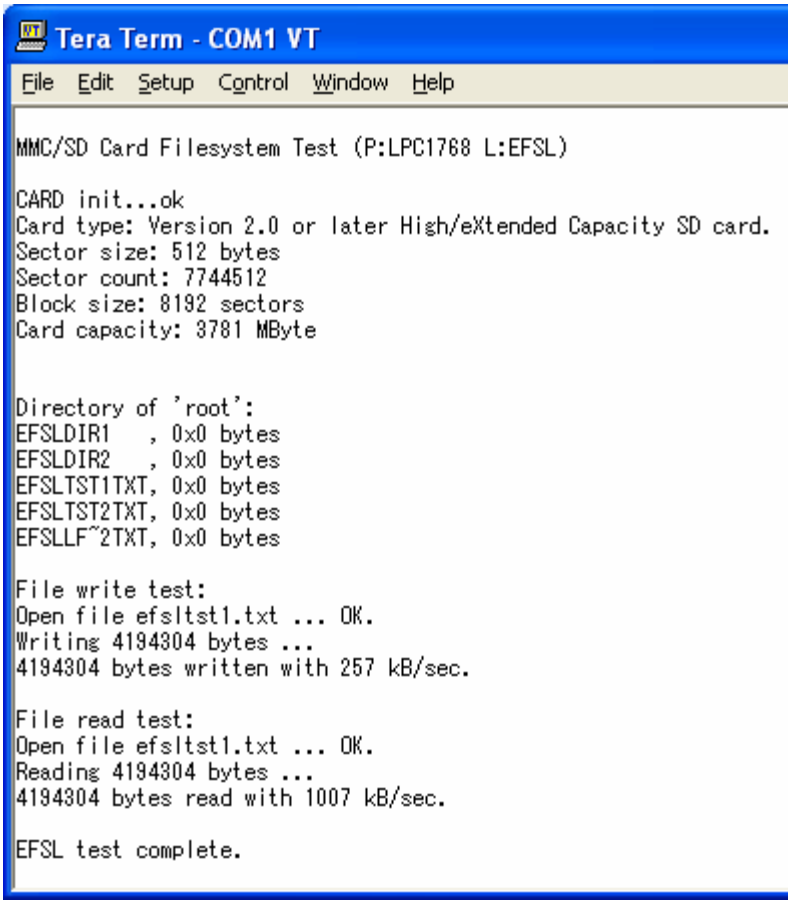

Fig 11. COM output

Remark: since EFSL does not support long file name (LFN), file “efsllfntst.txt” is displayed as “efsllf~2.txt”.

4. FatFs port on LPC1700

4.1 FatFs structure

The FatFs structure is shown below:

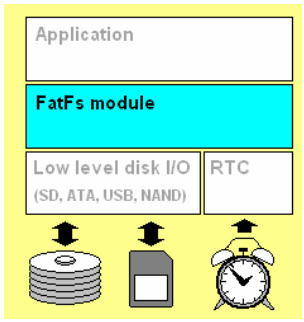

Fig 12. FatFs structure

FatFs module is a middleware which provides many functions to access the FAT volumes, such as f_open, f_close, f_read, f_write, etc (refer to ff.c). There is no platform dependence in this module, as long as the compiler is in compliance with ANSI C.

Low level disk I/O module is used to read/write the physical disk.

An RTC module is used to get the current time.

The Low level disk I/O and RTC module are completely separate from the FatFs module. They must be provided by the user, which is the main task of porting FatFs module to other platforms.

The rest of this section will describe step by step how to port FatFs (revision 0.08a) to LPC1700.

4.2 Define the size of integer types

The FatFs module assumes that the size of ***char/short/long*** are 8/16/32-bit and ***int*** is 16-bit or 32-bit. These correspondences are defined in integer.h. This will not be a problem on most compilers. Any conflict with existing definitions must be resolved carefully.

```
/* 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;
```

Fig 13. Integer types definitions for FatFs module

4.3 Configure the FatFs module

All of the configurations and detailed descriptions can be found in ffconf.h (for FatFs revision 0.08a).

The configurations used in this project are listed below.

Table 2. 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
	#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.

4.3.1 _USE_LFN

The FatFs module supports Long File Name (LFN) in revision 0.07e. The two different file names, SFN and LFN, of a file are transparent in the file functions except for f_readdir function. To enable LFN feature, set _USE_LFN to 1 or 2, and add a Unicode code conversion function ff_convert and ff_wtoupper to the project. This function is available in option\cc*.c.

Note that the LFN feature on the FAT file system is a patent of Microsoft Corporation. When enabled on commercial products, a license from Microsoft may be required depending on the final destination.

4.3.2 _CODE_PAGE

The _CODE_PAGE specifies the OEM code page to be used on the target system. Incorrect setting of the code page can cause a file open failure.

When the LFN feature is enabled, the module size will be increased depending on the selected code page. Table 3 shows the difference in module size when LFN is enabled with some code pages. The Chinese and Korean language have tens of thousands of characters and require a huge OEM-Unicode bidirectional conversion table; therefore, the module size will be drastically increased as shown in Table 3. As a result, the FatFs with LFN will not be able to be implemented in some microcontrollers with limited ROM size.

Table 3. ROM size increase for 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

4.4 Implement low level functions

Since the FatFs module is completely separated from disk I/O and RTC module, it requires the following functions to read/write the physical disk and to get the current time. Because the low level disk I/O and RTC module are not a part of the FatFs module, they must be provided by the user.

4.4.1 disk_initialize

The disk_initialize function initializes a physical drive.

This function is called from the volume mount process in the FatFs module to manage the media change. The application program must not call this function while the FatFs module is active, as this may cause the FAT structure on the volume to collapse. To re-initialize the file system, use f_mount function.

```
/*-----------------------------------------------------------------------*/
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 (SD_Init() && SD_ReadConfiguration())
        Stat &= ~STA_NOINIT;

    return Stat;
}
```

Fig 14. Implementation of disk_initialize

4.4.2 disk_status

The disk_status function returns the current disk status which is a combination of the following flags.

- STA_NOINIT: Indicates that the disk drive has not been initialized.
- STA_NODISK: Indicates that no medium is in the drive.
- STA_PROTECTED: Indicates that the medium is write protected.

Since the MCB1700 board does not provide card detection and write protection, we will neglect these two flags: STA_NODISK and 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;
}
```

Fig 15. Implementation of disk_status

4.4.3 disk_read

The disk_read function reads one or more sectors from the disk drive.

```
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 (SD_ReadSector (sector, buff, count) == SD_TRUE)
        return RES_OK;
    else
        return RES_ERROR;
}
```

Fig 16. Implementation of disk_read

4.4.4 disk_write

The disk_write writes one or more sectors to the disk.

This function is not required in read only configuration.

```
#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 ( SD_WriteSector(sector, buff, count) == SD_TRUE)
        return RES_OK;
    else
        return  RES_ERROR;

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

Fig 17. Implementation of disk_write

4.4.5 disk_ioctl

The disk_ioctl function controls device specified features and miscellaneous functions other than disk read/write.

Table 4. 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.

4.4.6 get_fattime

The get_fattime function gets current time which is not required in read only configuration.

```
/*-----------------------------------------------------------*/
/* 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);
}
```

Fig 18. Implementation of get_fattime

4.5 Demo

This demo was also tested on Keil's MCB1700 evaluation board with the same 2/4/8 GB SanDisk Micro SD/SDHC cards and COM configuration.

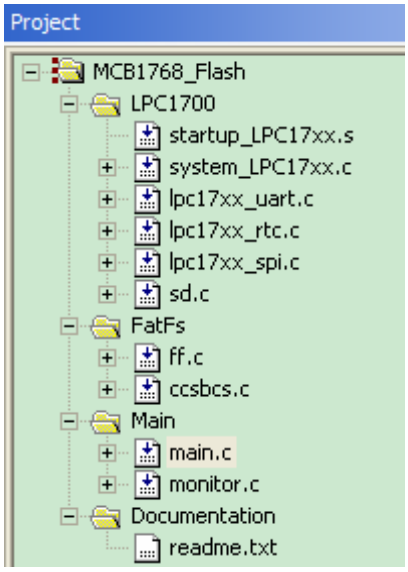

Fig 19. Source files of the project

Below is the COM output:

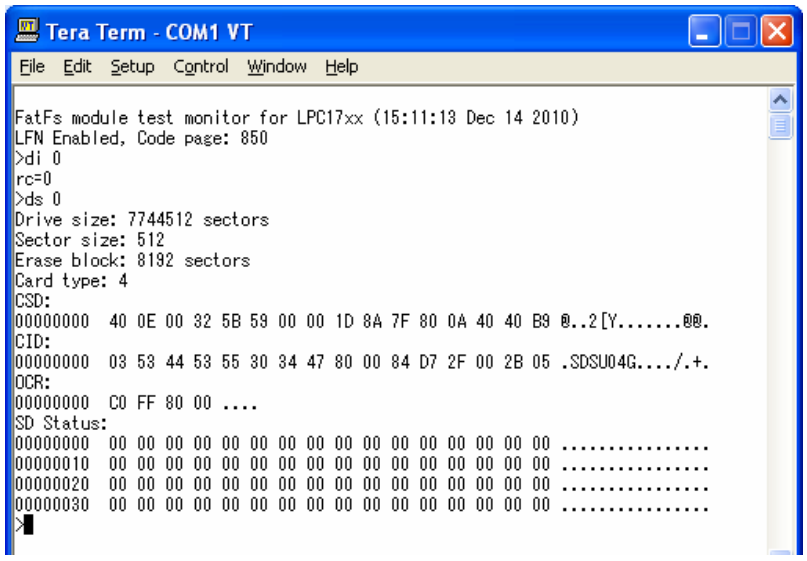

Fig 20. Disk commands (di/ds) test output

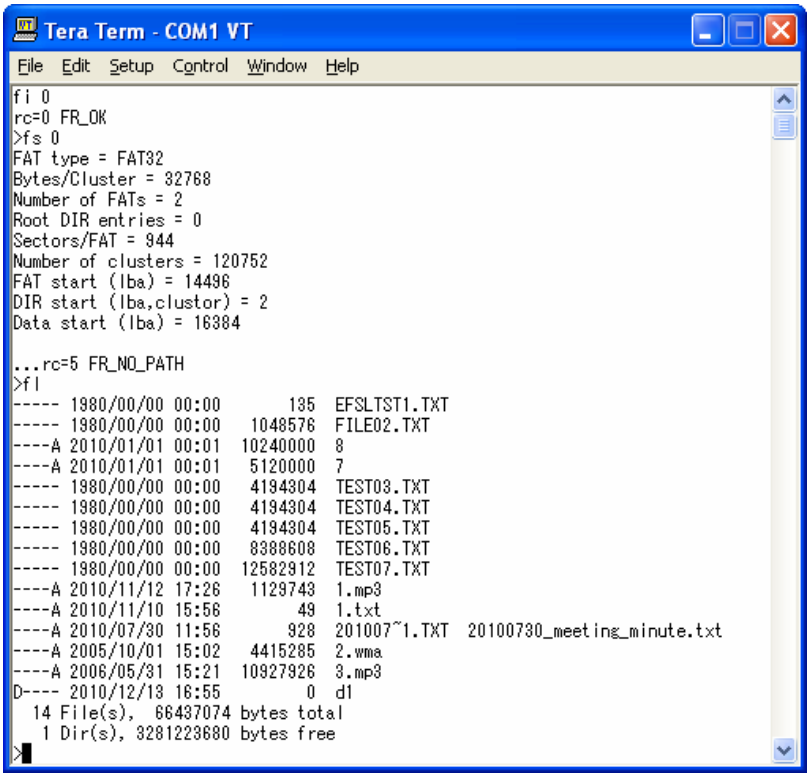

Fig 21. File commands (fi/fs/fl) test output

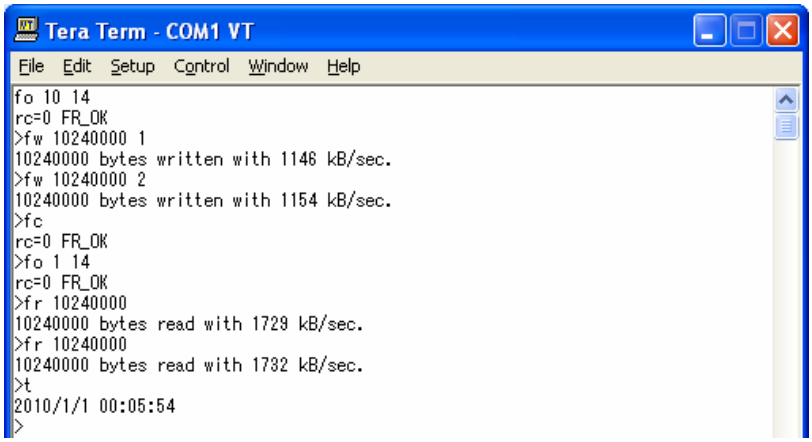

Fig 22. File read and write speed test output

5. References

[1] NXP LPC17xx User Manual UM10360 (Rev. 2), NXP Semiconductors, Aug. 18, 2010

[2] Embedded Filesystems Library (EFSL), Lennart Yseboodt & Michael De Nil, http://efsl.be/

[3] FatFs Generic FAT File System, Chan, http://elm-chan.org/fsw/ff/00index_e.html

[4] Embedded Filesystem Library for ARM controllers with interfaces for LPC2000(SPI) and AT91SAM7S(SPI), Martin THOMAS, http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/arm_memcards/index.html#efsl_arm

[5] ChaN's FAT-Module with LPC17xx SPI/SSP and USB-MSD, Martin THOMAS, http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/arm_memcards/index.html#chanfat_lpc_cm3

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

7. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an please send an email to: salesaddresses@nxp.com

Date of release: 1 May 2011
Document identifier: AN10916