SmartFusion cSoC: Implementation of FatFs on Serial Flash

Table of Contents

Introduction

The SmartFusion® customizable system-on-chip (cSoC) device contains a hard embedded microcontroller subsystem (MSS), programmable analog circuitry, and FPGA fabric consisting of logic tiles, static random access memory (SRAM), and phase-locked loops (PLLs). The MSS consists of a 100 MHz ARM® Cortex™-M3 processor, advanced high-performance bus (AHB) matrix, system registers, Ethernet MAC, DMA engine, real-time counter (RTC), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), fabric interface controller (FIC), the Philips Inter-Integrated Circuit (I^2C), serial peripheral interface (SPI), and external memory controller (EMC).

This application note describes:

- How to port and use the FatFs file system on the serial flash connected to the SPI interface of the SmartFusion Evaluation Kit Board and the SmartFusion Development Kit Board
- Configuration options of the FatFs
- How to use the SmartFusion RTC for the timestamping requirements of the FatFs

A basic understanding of the SmartFusion design flow is assumed. Refer to the *Using UART with a SmartFusion cSoC - Libero SoC and SoftConsole Flow tutorial* to understand the SmartFusion design flow.

Introduction to FatFs File System

FatFs is a software module used to organize a storage medium. It abstracts the physical layer interfaces to the storage medium and provides the functionalities to manage user data more efficiently with files and directories. Embedded file system library (EFSL) and FatFs are two popular and freely available FAT libraries for developing small embedded systems.

FatFs is a generic FAT file system module for small embedded systems that can be ported to any underlying hardware. It has been architected with separate layers for hardware accesses.

Figure 1 shows the structure of FatFs.

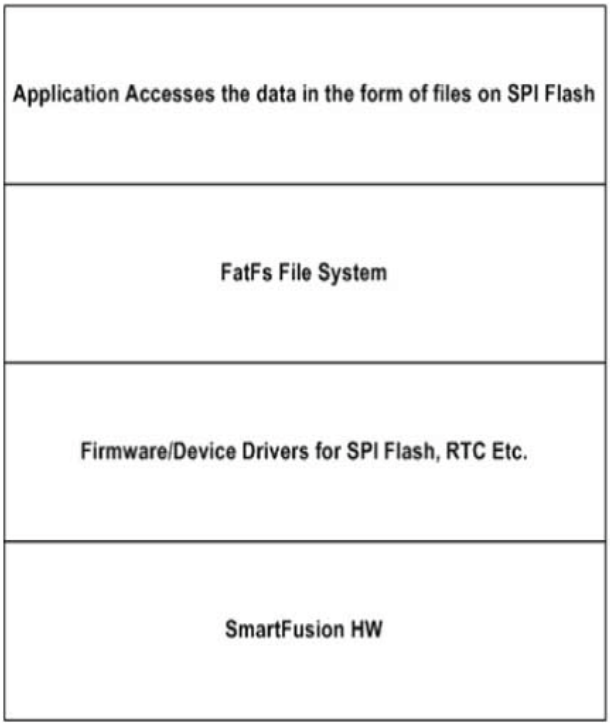

***Figure 1* • Structure of FatFs**

The FatFs has the following features:

1. Windows compatible FAT12/16/32 file system
2. Platform independence and easy to port
3. Very small footprint for code and work area
4. Various configuration options:
   - Multiple volumes (physical drives and partitions)
   - Multiple original equipment manufacturer (OEM) code pages, including double byte character set (DBCS)
   - Long file name (LFN) support in OEM code or Unicode
   - Real-time operating system (RTOS) support
   - Multiple sector size support
   - Read-only, minimized API, I/O buffer, etc.

This application note aims at porting and using the FatFs generic file system module on SPI flash memory available on the SmartFusion Evaluation Kit Board and the SmartFusion Development Kit Board as a physical medium.

Refer to the *Accessing Serial Flash Memory Using SPI Interface* application note for understanding SPI flash Read/Write operations on the SmartFusion Evaluation Kit Board and the SmartFusion Development Kit Board.

The FatFs module is free software available for education, research, and development. For more information about FatFs, refer to http://elm-chan.org/fsw/ff/00index_e.html.

Porting FatFs Requirements

You must provide the following functions to FatFs to give control for read and write to the physical medium.

Table 1 • **FatFs Porting Layer APIs**

Function	Use	When Required
Disk_initialization	Required for initialization of the physical medium	Always
Disk_status	Required to get the status of the physical medium for read and write	Always
Disk_read	Required to read the data from the physical medium	Always
Disk_write	Required to write the data to the physical medium	If file system is configured in both read and write mode
Disk_ioctl (CTRL_SYNC)	Required to get the write operations to be synchronized	If the file system is configured in both read and write mode
Disk_ioctl (GET_SECTOR_COUNT)	Required to send the number of sectors on the physical medium	When creating the file system on the medium
Disk_ioctl (GET_SECTOR_SIZE)	Required to send the sector size	If sector size is > 1024
Disk_ioctl (GET_BLOCK_SIZE)	Required to send the block size of the physical medium	When creating the file system on the medium
Disk_ioctl (CTRL_ERASE_SECTOR)	Required when file system required to erase the sectors	When erase is required

Disk I/O Interface

Since the FatFs module is completely separated from the disk I/O layer, it requires the following functions to access the physical storage media. The low level disk I/O module is not a part of the FatFs module and hence the following APIs are provided to access the SPI Flash on the SmartFusion Evaluation Kit Board and the SmartFusion Development Kit Board:

- disk_initialize – Initialize disk drive
- disk_status – Get disk status
- disk_read – Read sector(s)
- disk_write – Write sector(s)
- disk_ioctl – Control device dependent features
- get_fattime – Get current time

The following sections provide more details on API requirements.

Disk Initialization

API in C code implementation:

```
DSTATUS disk_initialize (
BYTE drv    /* Physical drive  umber (0..) */
)
{
     spi_flash_init();
     spi_flash_control_hw(SPI_FLASH_GLOBAL_UNPROTECT,0,NULL);
     return 0;
}
```

Description: The above API performs the SPI Flash initialization and provides read and write access variables.

Disk Read

API in C code implementation:

```
#define ERASE_BLOCK_SIZE 4096
DRESULT disk_read (
   BYTE drv,/* Physical drive  umber (0..) */
   BYTE *buff,/* Data buffer to store read data */
   DWORD sector,/* Sector address (LBA) */
   BYTE count/* Number of sectors to read (1..255) */
)
{
   return (spi_flash_read(sector* ERASE_BLOCK_SIZE, buff, count* ERASE_BLOCK_SIZE));
}
```

Description: This API implements read functionality for the SPI Flash based on the sector address and number of sectors to be read.

Disk Write

API in C code implementation:

```
DRESULT disk_write (
   BYTE drv,/* Physical drive  umber (0..) */
   const BYTE *buff,/* Data to be written */
   DWORD sector,/* Sector address (LBA) */
   BYTE count/* Number of sectors to write (1..255) */
)
{
    int i;
    for (i =1; i<=count; i++)
{

spi_flash_control_hw(SPI_FLASH_4KBLOCK_ERASE,(sector*i* ERASE_BLOCK_SIZE),NULL);
    }
    return (
      spi_flash_write(sector* ERASE_BLOCK_SIZE, (uint8_t *) buff, count*
ERASE_BLOCK_SIZE));
}
```

Description: This API implements write functionality for the SPI Flash based on the sector address and number of sectors to written.

Disk I/O Controls

API in C code implementation:

```
#define NO_OF_SECTORS 2048
DRESULT disk_ioctl (
   BYTE drv,/* Physical drive  umber (0..) */
   BYTE ctrl,/* Control code */
   void *buff/* Buffer to send/receive control data */
)
{
    UINT *result = (UINT *)buff;
    switch (ctrl)
    {
        case GET_SECTOR_COUNT:
           *result = NO_OF_SECTORS;
     break;

        case GET_SECTOR_SIZE:
       *result = ERASE_BLOCK_SIZE;
     break;
        case GET_BLOCK_SIZE:
           *result = 1;/*in no.of Sectors */
        break;
        default:
        break;
    }
    return 0;
}
```

Description: This API implements the GET_SECTOR_COUNT, GET_SECTOR_SIZE, and GET_BLOCK_SIZE functionalities. SECTOR_COUNT is the number of physical sectors available on SPI Flash, SECTOR_SIZE is the minimum erase block size available for the SPI Flash, and GET_BLOCK_SIZE is the number of sectors for each block, it being one in this case.

FatFs Configurations

The ffconfig.h file, available in the SoftConsole project provided in the design files, defines all the possible configuration for the FatFs. Table 2 on page 6 describes the configurations that are enabled in the software.

Table 2 • **FatFs Configuration Selection**

Modules	Configuration	Description
Function and Buffering Configurations	#define _USE_MKFS 1	Enabling support for making the file systems command f_mkfs. After setting _USE_MKFS to 1, set _FS_READONLY to 0.
	#define _FS_TINY 0	When _FS_TINY is set to 1, FatFs uses the sector buffer in the file system object instead of the sector buffer in the individual file object for file data transfer. This reduces memory consumption by 512 bytes in each file object.
	#define _FS_READONLY 0	Setting _FS_READONLY to 1 defines read only configuration. This removes writing functions, f_write, f_sync, f_unlink, f_mkdir, f_chmod, f_rename, f_truncate, and f_getfree.
	#define _FS_MINIMIZE 0	The _FS_MINIMIZE option defines minimization level to remove some functions. 0: Full function. 1: f_stat, f_getfree, f_unlink, f_mkdir, f_chmod, f_truncate, and f_rename are removed. 2: f_opendir and f_readdir are removed in addition to level 1. 3: f_lseek is removed in addition to level 2.
	#define _USE_STRFUNC1	To enable string functions, set _USE_STRFUNC to 1 or 2.
	#define _USE_FORWARD 0	To enable f_forward function
	#define _USE_FASTSEEK 0	To enable the fast seek function

Table 2 • **FatFs Configuration Selection (continued)**

Modules	Configuration	Description
Locale and Namespace Configurations	#define _CODE_PAGE 437	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. 932 - Japanese Shift-JIS (DBCS, OEM, Windows) 936 - Simplified Chinese GBK (DBCS, OEM, Windows) 949 - Korean (DBCS, OEM, Windows) 950 - Traditional Chinese Big5 (DBCS, OEM, Windows) 1250 - Central Europe (Windows) 1251 - Cyrillic (Windows) 1252 - Latin 1 (Windows) 1253 - Greek (Windows) 1254 - Turkish (Windows) 1255 - Hebrew (Windows) 1256 - Arabic (Windows) 1257 - Baltic (Windows) 1258 - Vietnam (OEM, Windows) 437 - U.S. (OEM) 720 - Arabic (OEM) 737 - Greek (OEM) 775 - Baltic (OEM) 850 - Multilingual Latin 1 (OEM) 858 - Multilingual Latin 1 + Euro (OEM) 852 - Latin 2 (OEM) 855 - Cyrillic (OEM) 866 - Russian (OEM) 857 - Turkish (OEM) 862 - Hebrew (OEM) 874 - Thai (OEM, Windows) 1 - ASCII only (Valid for non LFN cfg.)

Table 2 • **FatFs Configuration Selection (continued)**

Modules	Configuration	Description
	#define _USE_LFN 1	To enable long file names. These can show 0-3 options. The _USE_LFN option switches the LFN support. 0: Disable LFN. _MAX_LFN and _LFN_UNICODE have no effect. 1: Enable LFN with static working buffer on the bss. NOT REENTRANT. 2: Enable LFN with dynamic working buffer on the STACK. 3: Enable LFN with dynamic working buffer on the HEAP. The LFN working buffer occupies (_MAX_LFN + 1) X 2 bytes. After enabling LFN, Unicode handling functions ff_convert() and ff_wtoupper() must be added to the project. When enabled to use heap, memory control functions ff_memalloc() and ff_memfree() must be added to the project.
	#define _MAX_LFN 255	To set the maximum file name length
	#define _LFN_UNICODE 0	0: ANSI/OEM or 1:Unicode To switch the character code set on FatFs API to Unicode, enable LFN feature, and set _LFN_UNICODE to 1.
	#define _FS_RPATH 1	To enable the relative path. When _FS_RPATH is set to 1, the relative path feature is enabled and f_chdir, f_chdrive functions are available.
Physical Drive Configurations	#define _DRIVES 1	Number of volumes (logical drives) to be used
	#define _MAX_SS 4096	This is the minimum erase sector size of the SPI Flash.
	#define _MULTI_PARTITION 0	When _MULTI_PARTITION is set to 0, each volume is bound to the same physical drive number and can mount only the first primary partition. When it is set to 1, each volume is tied to the partitions listed in Drives.
System Configurations	#define _WORD_ACCESS 0	Set to 0 first and it is always compatible with all platforms. The _WORD_ACCESS option defines which access method is used to the word data on the FAT volume. 0: Byte-by-byte access. 1: Word access. Do not choose this unless following condition is met: When the byte order on the memory is big-endian or address misaligned word access results incorrect behavior, the _WORD_ACCESS must be set to 0. If it is not the case, the value can also be set to 1 to improve the performance and code size.
	#define _FS_REENTRANT 0	This option is needed in a multi-threaded environment, such as when RTOS/OS is used.
	#define _FS_TIMEOUT 1000	Timeout period in unit of time ticks
	#define _FS_SHARE 0	To enable the file sharing feature, set _FS_SHARE to ≥1 and also user provided memory handlers. The ff_memalloc and ff_memfree functions must be added to the project. The value defines the number of files that can be opened per volume.

Table 3 shows which API function is removed by configuration options for the module size reduction.

Table 3 • **FatFs APIs with Configuration**

Function	_FS_MINIMIZE			_FS_READONLY	_USE_STRFUNC	_FR_RPATH FUNCTION		_USE_MKFS	_USE_FORWARD
	1	2	3	1	0	0	1	0	0
f_mount									
f_open									
f_close									
f_read									
f_write				X					
f_sync				X					
f_lseek			X						
f_opendir		X	X						
f_readdir		X	X						
f_stat	X	X	X						
f_fgetfree	X	X	X	X					
f_truncate	X	X	X	X					
f_unlink	X	X	X	X					
f_mkdir	X	X	X	X					
f_chmod	X	X	X	X					
f_utime	X	X	X	X					
f_rename	X	X	X	X					
f_chdir						X			
f_chdrive						X			
f_getcvd						X	X		
f_mkfs				X				X	
f_forward									X
f_putc				X	X				
f_puts				X	X				
f_printf				X	X				
f_gets					X				

Table 4 • **Some of the FatFs API Usage and Description**

API	Usage	Description
f_mount	f_mount(isked, &fatfs);	This API is used to mount the FatFs file system on the physical medium.
f_open	f_open(&file1,(char*)file_name,FLAGS);	This API is used to open the file with different flags, such as read, write, create etc.
f_close	f_closes(&file1);	This API is used to close the opened file.
f_read	f_read(&file1, Buff, sizeof(Buff), &s1);	This API is used to read an open file.
f_write	f_write(&file1 Buff, sizeof(Buff), &s1);	This API is used to write to an open file.
f_lseek	f_lseek(&file1, size);	This API is used to move the file pointer to the required location.

You must include the ff.h header file in the application to use the FatFs APIs. The entire file system configuration has to be selected in the ffconfig.h header file. These two files are available in the design files provided with this application note.

Example Application Software

This demo example shows how to create, mount, and use the FatFs file system on SPI flash using the SmartFusion cSoC devices. Use HyperTerminal help once the design files are loaded in the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board.

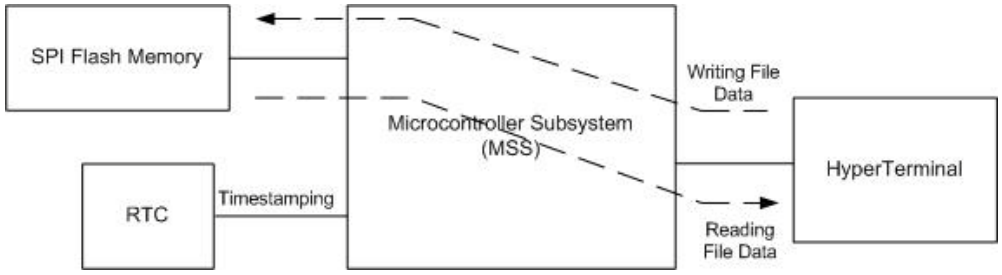

Figure 2 • **Data Flow Diagram for the FatFs**

Running the Design

Board Settings

The design example works on the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board with default board settings. Refer to the following user's guides for default board settings:

- *SmartFusion Development Kit User's Guide*
- *SmartFusion Evaluation Kit User's Guide*

Program the Design

Program the SmartFusion Evaluation Kit Board or the SmartFusion Development Kit Board with the generated/provided *.STP file (refer to "Appendix A – Design Files" on page 12) using FlashPro, and then power cycle the board.

Running the Application

Invoke the SoftConsole IDE, double-click **Write Application Code** under **Develop Firmware** in the Libero® System-on-Chip (SoC) software design flow window
(refer to "Appendix A – Design Files" on page 12) and launch the debugger. Start a HyperTerminal session with the baud rate set to 57600, 8 data bits, 1 stop bit, no parity, and no flow control.

If your PC does not have the HyperTerminal program, use any free serial terminal emulation program such as PuTTY or Tera Term. Refer to the *Configuring Serial Terminal Emulation Programs* tutorial for configuring HyperTerminal, Tera Term, or PuTTY.

When you run the debugger in SoftConsole, the HyperTerminal window prompts the file system commands and you must select the command to do the File system operation. Figure 3 shows the main menu of the FatFs.

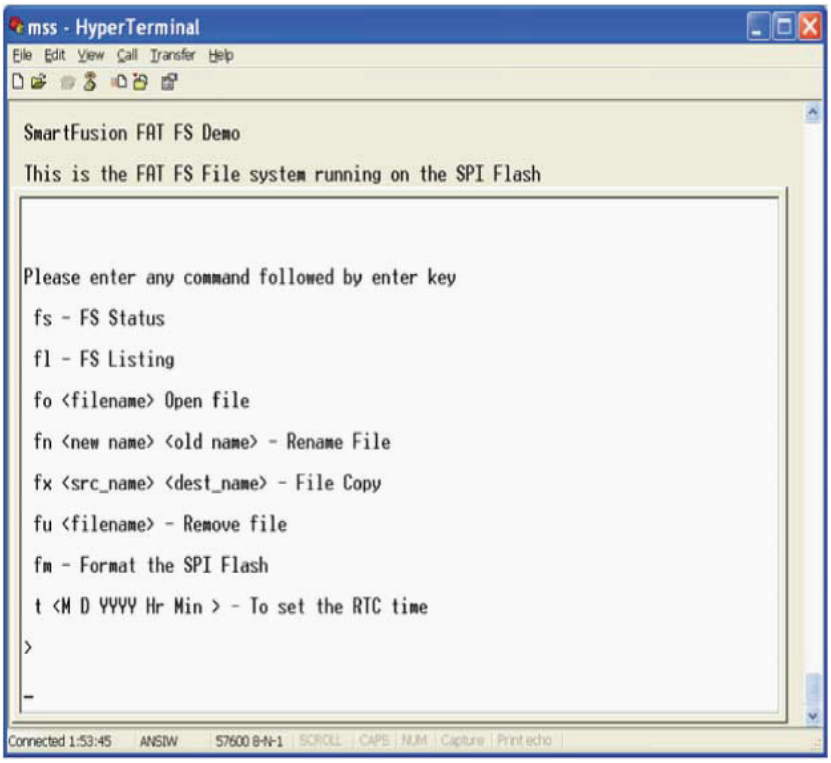

***Figure 3* • Menu of the FatFs Demo**

Release Mode

The release mode programming file (STAPL) is also provided. Refer to the Readme.txt file included in the programming file for more information.
Refer to the *SmartFusion cSoC: Building Executable Image in Release Mode and Loading into eNVM Tutorial* for more information on building an application in Release mode.

Appendix A – Design Files

You can download the design files from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC360_DF.
The design file consists of Libero SoC, Verilog project, SoftConsole software project, and programming files (*.stp). Refer to the Readme.txt file included in the design file for directory structure and description.

You can download the programming files (*.stp) in release mode from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC360_PF.
The programming file consists of STAPL programming file (*.stp) for A2F500-DEV-KIT and A2F-EVAL-KIT, and a Readme.txt file.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 4 (January 2013)	Added "Board Settings" section and Modified "Running the Design" section (SAR 43469).	10
Revision 3 (May 2012)	Modified "Appendix A – Design Files" (SAR 38435)	12
Revision 2 (February 2012)	Removed "zip" extension in the Design files link (SAR 36763).	12
Revision 1 (January 2012)	Modified the "Introduction" section (SAR 35869).	1
	Modified the "Running the Design" section (SAR 35869).	10
	Added section called "Release Mode" (SAR 35869).	11
	Modified the "Appendix A – Design Files" section (SAR 35869).	12

*Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

51900228-4/01.13