SMT387-File System

User Manual

Sundance Digital Signal Processing Inc, 4790 Caughlin Parkway #233, Reno, NV 89509-0907, USA.

Revision History

	Changes Made	Issue	Initials
02/04/05	**Edition**	**1.1**	

List of Abbreviations

Abbreviation	Explanation
FAT	File Allocation Tables
FS	File System
SMT	Sundance Multiprocessor Technology
SATA	Serial ATA Disk
TIM	Texas Instruments Module

Table Of Contents

Introduction

This document describes the file system software package provided for use with the Sundance SMT387 module. It includes descriptions of the functions available in the package as well as details of the procedures to be used when setting up a new hard disk drive for use with the SMT387.

Overview

The SMT387 File System Package allows an application running on the SMT387 to read and write files in FAT32 file systems on attached SATA hard disk drives. Attaching the same hard disk drive to a personal computer running Windows XP or Windows 2000 allows the same FAT32 file system to be read and written by applications on that computer.

The library should only be used in a *single* user task. Use in multiple tasks will result in data corruption. If you need to access a FAT32 file system from more than one user task, you must use inter-task communication to forward requests from one task to another.

Within the task that is using the File System Package, multiple threads can all access files at the same time, subject to available memory. The SMT387 File System Package synchronizes access to the files, and manages all file system metadata such as File Allocation Tables (FATs) and directory records. The package is compatible with the Diamond stand-alone library for use in fully embedded applications.

Performance features included in the SMT387 File System Package include a data buffering facility to decouple user threads from activity on the disk drive, and a preallocation facility designed to reduce the time taken when files are extended.

Requirements

The following requirements must be met for a system to be able to use the SMT387 File System Package:

- SMT387 module with V2 DSP silicon.
- One or two attached SATA disk drives.
- Diamond version 2.5.2 or later.

Packaging

The software is provided as the following files:

- *Smt387fs.h* contains the API definitions as described in the *User Guide,* and must be #included into user applications.
- *Smt387fs.lib* is the library containing the code implementing the file system functionality, and must be linked into user applications.
- *Blockman.mod* is a required kernel extension module. It must be visible to the Diamond configurer, either by being placed in the same directory as your user application or in the Diamond modules directory,\3L\Diamond\C6000\modules.
- *Smt387show.app* is an application you can run using the Diamond Windows Server. When run, it first displays each drive and each FAT partition. After this it attempts to mount all partitions in the same way an application using the library would do, so that any problems in this process can be diagnosed. Finally, the files in the root directory of each volume are listed.

In addition, a demonstration program is supplied as demo_create.c. You can build an application from this file using the supplied make_demo.bat command file. This demonstration program creates two files on the first detected FAT32 file system; one file is empty, the second is roughly 172MB in size containing a repeated pattern.

Hard Disk Drive Preparation

In order to be used with the SMT387 File System Package, a hard disk drive must first be prepared through a process of *partitioning* and *formatting.*

Partitioning

An initial step in the preparation of any hard disk drive is the creation of a standard *partition table* on the drive. This is most easily achieved by connecting the drive to a Windows XP computer, which will offer to write a standard disk label on any drive that does not already have one. A system reboot is normally required after this operation.

The standard partition table written to a drive can support up to four *primary* partitions. In addition, one of these partitions can be an *extended* partition capable of holding additional *logical* partitions.

The SMT387 File System Package supports up to four FAT32 file systems on each of two physical drives. When partitioning the drive to create these FAT32 file systems, you must bear the following constraints in mind:

- Partitions greater than around 32GB can be created, read and written under Windows XP, but not formatted as a file system. In order to use a FAT32 partition larger than 32GB, you can create an unformatted partition under Windows XP and format it using the smt387fmt utility provided with the SMT387 File System Package. This utility runs on the SMT387 module, not under Windows XP.
- The SMT387 File System Package supports only primary partitions, not extended partitions or logical partitions. This limits the system to four partitions per drive.

If you have no other requirements, the simplest configuration for use with the SMT387 File System Package is to use Windows XP to create a single unformatted partition on the drive, starting at the beginning of the drive and extending to use all of the drive's space. Then, use smt387fmt to format the partition with an appropriate FAT32 file system.

Formatting

The second stage of drive preparation is to format the partition as a FAT32 file system.

When you create a partition under Windows XP, you will be offered the option of formatting the partition at the same time. If the partition is less than around 32GB, you can select the following options to create a valid file system for use with the SMT387 File System Package:

- File system type: FAT32
- Cluster size: 32KB

If the partition is larger than about 32GB, you will *not* be given the option of a FAT32 file system, but only an NTFS file system. An NTFS file system can *not* be accessed by the SMT387 File System Package, and in this case you must use the smt387fmt application to format the partition.

If you format the partition under Windows XP, it is also important to note that you must select a 32KB cluster size, as this is not the default. For performance reasons, the SMT387 File System Package only processes FAT32 file systems with a 32KB cluster size.

You may also partition and in some cases format a drive for use with the SMT387 File System Package under Windows 2000. Note, however, that a bug in Windows 2000 means that you may be presented with the option of formatting a FAT32 file system larger than 32GB. Although this option is presented, accepting it does *not* result in a correctly formatted file system, and in most cases the format operation will fail after a delay of up to an hour.

For a file system larger than 32GB, you must request an unformatted partition under Windows XP and format the partition later using smt387fmt. You may also follow this course for smaller partitions, and we suggest that smt387fmt is used for all partitions for use with the SMT387 File System Package.

To format a partition using smt387fmt, attach the drive to either connector on the SMT387 module and run the smt387fmt application using the Diamond Windows Server.

Once loaded, the application will describe the drives attached to the SMT387 along with their partitions, and indicate which partitions may be formatted. To ensure that other user data on the drive is not damaged by this operation, it will only permit formatting on an unformatted partition, or on a partition already formatted as FAT32.

The latter option is useful to clear out a damaged FAT32 partition, or to reformat one that was formatted with the wrong cluster size. Note that all data in the selected partition will be deleted during a reformat operation of this type.

Application Programming Interface (API)

API Overview

User code accesses the SMT387 File System Package through definitions contained in the header file smt387fs.h, as follows:

```
#include <smt387fs.h>
```

The task making use of the SMT387 File System Package must make exactly one call to the smt387fs_init() function before using any other API functions, passing through details of a block of on-chip memory for use by the package. The memory passed through is used to provide buffering and caching of file system metadata such as File Allocation Table (FAT) contents. The more space is provided for these purposes, the better the performance of the SMT387 File System Package will be.

Once the SMT387 File System Package has been initialized and space donated to it, applications will normally open files for read or create files for write, read and write data to open files, and finally close those files. Each open file is represented within the user code as an smt387_stream_handle object; this type is defined within smt387fs.h. It is an abstract type: all manipulation of smt387_stream_handle objects should be performed through the API.

API functions are also provided to provide file-level operations such as file rename and delete, as well as file system-level functions such as synchronizing the state of the disk drives.

When all application operations have been completed, applications must call smt387fs_deinit() to signify that they no longer require use of the SMT387 File System Package.

File Names

The SMT387 File System Package uses a DOS-like file naming convention with an optional “volume” prefix followed by a traditional “8+3” file name.

To distinguish SMT387 volumes from those on the host system, the SMT387 File System Package uses numeric volume prefixes instead of alphabetic ones. These volume prefixes range from 0: to 7: and are assigned in the order that the volumes are mounted. For example, if the first SATA drive contained two mountable volumes, they would be referred to as volumes 0: and 1:. The first volume on the second SATA drive would then be referred to as volume 2:.

Thus, a complete file name might look like 2:xyz.txt. File names without file system numbers, such as xyz.txt, are assumed to be on file system 0, the first recognized file system.

For example, the following are valid file names for use with the SMT387 File System:

- X.TXT
- 0:X.TXT
- 3:X

API Details

Each function provided in the API is now described on a separate page. The function's synopsis is followed by a description of its function, along with details of its synchronization characteristics and common conditions under which it may fail. The error conditions listed against each should not be regarded as exhaustive; for example, internal errors and disk errors may be reported from most API functions. User applications should always test all API return values, even when no error conditions are listed here.

Function smt387fs_init

Synopsis

int smt387fs_init(size_t size, void *space);

Definition

The user task making use of the SMT387 File System Package must call this function exactly once before calling any other API functions.

Donates the *size* bytes of memory at *space* for use by the SMT387 File System Package. This will normally be space allocated from the calling task's heap but may have been acquired in any way.

The memory provided should be word aligned, and in on-chip memory. To get a rough idea of how much memory is required in a particular situation, add together:

- 0.5KB, plus
- 1KB for each volume mounted, plus
- 1KB for every 32GB of disk space or part thereof, plus
- 32.75KB for each open file

At least 50KB of memory should therefore be provided for basic functionality, but any increase above this will provide performance benefits.

Synchronization

Must be called only once, before all other API calls.

Error Conditions

SMT387FS_MEM_NULL Null pointer passed.

SMT37FS_MEM_ALLOC Not enough memory provided.

SMT387FS_MEM_ALIGN Memory is not word aligned.

SMT387FS_MEM_OFFCHIP Memory is not on-chip.

Function smt387fs_deinit

Synopsis

```
int smt387fs_deinit ();
```

Definition

The user task making use of the SMT387 File System Package may call this function when it knows that it no longer requires use of the file system facility. This may be used to tidy resources associated with the task. It is an error to call any API functions (including smt387fs_init ()) after a call to smt387fs_deinit ().

Synchronization

Thread safe.

Error Conditions

Always returns FS_OK.

Function smt387fs_delete

Synopsis

int smt387fs_delete (char *filename);

Definition

Deletes the named file.

Synchronization

Thread safe.

Error Conditions

FS_BAD_NAME if the file name provided is not a valid name.

SMT387FS_FILE_NOEXIST if the file does not exist.

SMT387FS_FILE_RDONLY if the file’s attributes state that it is a read-only file.

SMT387FS_FILE_OPEN if the file is currently open for reading or writing.

Otherwise FS_OK.

Function smt387fs_rename

Synopsis

int smt387fs_rename (char *from, char *to);

Definition

Changes the name of file *from* to *to*.

Synchronization

Thread safe.

Error Conditions

FS_BAD_NAME if one of the file names provided is not a valid name.

SMT387FS_FILE_NOEXIST if the file does not exist.

SMT387FS_FILE_OPEN if the file is currently open for reading or writing.

SMT387FS_FILE_EXISTS if the to file already exists.

SMT387FS_FILE_RDONLY if the file’s attributes state that it is a read-only file.

SMT387FS_VOL_DIFF if from and to are on different volumes

Otherwise FS_OK.

Function smt387fs_create_write

Synopsis

int smt387fs_create_write (char *filename, smt387fs_stream_handle *handle);

Definition

Creates a new file and opens it for writing. If successful, the object referred to by *handle* is made a handle for the open file. The library supports a maximum of 10 files open at any time; trying to open more than this will result in the ??? error. However, this value may be further restricted if insufficient memory has been made available to the library in the initial call to smt387fs_init; in this case, attempting to open too many files may cause the library to pause waiting for additional resources, which will never arrive. If this occurs, try increasing the amount of memory passed to smt387fs_init.

Synchronization

Thread safe.

Error Conditions

FS_BAD_NAME if the file name provided is not a valid name.

SMT387FS _FILE_EXISTS if the named file already exists.

FS_NO_SPACE if the file system is full and the directory needs to be extended in order to create this file.

??? Too many open files.

Otherwise FS_OK.

Function smt387fs_open_read

Synopsis

int smt387fs_open_read (char *filename, smt387fs_stream_handle *handle);

Definition

Opens the named file for read. If successful, the object referred to by *handle* is made a handle for the open file. The library supports a maximum of 10 files open at any time; trying to open more than this will result in the ??? error. However, this value may be further restricted if insufficient memory has been made available to the library in the initial call to smt387fs_init; in this case, attempting to open too many files may cause the library to pause waiting for additional resources, which will never arrive. If this occurs, try increasing the amount of memory passed to smt387fs_init.

Synchronization

Thread safe.

Error Conditions

FS_BAD_NAME if the file name provided is not a valid name.

SMT387FS_FILE_NOEXIST if the file does not exist.

SMT387FS_FILE_OPEN if the file is currently open for reading or writing.

??? Too many open files.

Otherwise FS_OK.

Function smt387fs_close

Synopsis

int smt387fs_close (smt387fs_stream_handle *handle);

Definition

Closes the open file represented by *handle*, and overwrites *handle* so that it can no longer be used.

Synchronization

Thread safe.

Error Conditions

Always returns FS_OK.

Function smt387fs_read

Synopsis

int smt387fs_read (smt387fs_stream_handle *handle, size_t size, void *buffer, size_t *actual);

Definition

Reads up to *size* bytes from the open file represented by *handle* into the buffer at *buffer*. Returns the number of bytes actually read in *actual;* if the end of the file is reached before *size* bytes are read, *actual* will be less than *size.*

Synchronization

For performance reasons, this function is *not thread safe,* and must not be called from more than one thread at any given time.

Error Conditions

FS_OPEN_MODE if the file is not open for reading.

Otherwise, returns FS_OK.

Function smt387fs_write

Synopsis

int smt387fs_write (smt387fs_stream_handle *handle, size_t size, void *buffer);

Definition

Reads up to *size* bytes from the open file represented by *handle* into the buffer at *buffer*. Returns the number of bytes actually read in *actual;* if the end of the file is reached before *size* bytes are read, *actual* will be less than *size.*

Synchronization

For performance reasons, this function is *not thread safe,* and must not be called from more than one thread at any given time.

Error Conditions

FS_OPEN_MODE if the file is not open for writing.

FS_TOO_BIG if the file has reached the 4GB limit for files on FAT32 file systems.

FS_NO_SPACE if the file system is full.

Otherwise, returns FS_OK.

Function smt387fs_sync

Synopsis

int smt387fs_sync ();

Definition

Synchronizes the state of all open files' data and metadata by flushing any buffered data for the file and re-writing the file's directory entry. In addition, halts all operations on all files until all modified data has been written to the hard disk. This function is implicitly called by smt387fs_deinit; therefore, most user applications will have no need to call smt387fs_sync directly.

Synchronization

Thread safe.

Error Conditions

Always returns FS_OK.