Running FAT16 File Systems and DOS Commands on SHARC® Processors

Contributed by Mitesh Moonat, Jagadeesh Rayala and Aseem Vasudev

Rev 1 – September 28, 2007

Introduction

File Allocation Table 16 (FAT16) is a widely used file format implemented on mid-size memory devices (up to 2GB) such as NAND flash, Multi-Media Card (MMC), and Secure Digital (SD) cards. These memory devices are used mainly to store multimedia content and data. Therefore, it is necessary for media processing software applications to interpret the FAT16 file format in order to access and/or manipulate the content stored on such memory devices.

This EE-Note discusses the implementation of various commands similar to Disk Operating System (DOS) commands to access and modify the FAT16 file system on an MMC interfaced to an ADSP-21369 SHARC® processor. This document includes details about dealing with the files stored on MMC. The source code for this interface has been developed and validated on an ADSP-21369 EZ-KIT Lite® evaluation board and is provided with this application note.

Although, this document refers to the ADSP-21369 SHARC processor, the same concepts presented here apply to ADSP-21367, ADSP-21368, and ADSP-2137x SHARC processors.

The implementation of basic DOS commands that list files and folders in current directory, create a directory, copy a file, read a file, search a file, navigate through sub-folders in current directory, or delete a file or directory are discussed in detail. The code supports long file names (up to 255 bytes). Other file systems such as FAT12 and FAT32 are briefly discussed and are compared to FAT16 file system.

FAT Overview

FAT is a partially patented file system developed by Microsoft® for MS-DOS®. It involves the use of a table that centralizes the information about the areas belonging to files. This information helps to identify which areas are free or usable, and where each file is stored on the disk. It also identifies the areas that are unusable. To reduce the management complexity, disk space is allocated to files in contiguous groups of hardware sectors called as *clusters*. The maximum number of clusters possible that can be addressed by a file system depends on the number of bits being used to identify a cluster in FAT.

Currently, there are three types of FAT file systems: FAT12, FAT16, and FAT32. The basic difference between them is the number of bits in the FAT entries on a disk. FAT12 uses 12-bit entries, FAT16 uses 16-bit entries, and FAT32 uses 28-bit entries. The FAT standard has also evolved in several ways, preserving backward compatibility with existing software. Table 1 shows basic differences between the different FAT file systems.

FAT is still a normal file system for removable media (with the exception of CDs and DVDs), with FAT12 used on floppies and FAT16 used on most other removable media (such as flash memory cards for digital cameras, mobile phones, and USB flash drives). Most removable media are not yet large enough to benefit from FAT32, although some larger flash drives such as Secure Digital High-Capacity (SDHC) cards use it. FAT16 is used on these drives for reasons of compatibility and size overhead.

	FAT12	**FAT16**	**FAT32**
Uses	Floppies and small hard disk volumes	Small to large hard disk volumes	Medium to very large hard disk volumes
Size of each FAT entry	12 bits	16 bits	28 bits
Volume size (maximum)	16 Mbytes	2 Gbytes	about 2^41

Table 1. Comparison between various FAT versions

FAT16 on a Disk

Figure 1 shows the order of structures on a disk or a FAT partition.

Boot sector	More reserved sectors (optional)	File Allocation Table 1	File Allocation Table 2	Root Directory	Data Region (for files and directories) (To end of partition or disk)

Figure 1. Sections of a disk or partition

Reserved Sectors

The beginning sectors are reserved sectors. The first sector (`sector 0`), which is called the `Boot Sector` (or `Partition Boot Record`), is mandatory; the others are optional. The boot sector includes an area called the `BIOS` parameter block that includes the basic file system information (in particular, the FAT type and pointers to the location of the other sections). The total count of reserved sectors is indicated by a field inside the `Boot Sector`.

FAT Region

The FAT region contains two copies of FAT. Though rarely used, the second copy is helpful when the default copy gets corrupted. This region contains the mapping details of the data, region indicating the clusters are used and the clusters that are not used.

Root Directory Region

The root directory region contains a `Directory Table` that stores information about the files and directories located in the root directory region. This region has a fixed maximum size that is pre-allocated at the time that the volume is formatted. This region is only used for FAT12 and FAT16; FAT32 stores the root directory entries in the data region just as any other sub-directories.

Data Region

The data region is the actual region where all files and directories are stored. This region occupies the majority of the partition space. Files and directories stored in this region can be made arbitrarily long simply by adding more clusters in the clusters chain of the FAT.

Details of FAT16 Structure

The following sections describe the entire FAT16 structure in detail.

Boot Sector

Table 2 lists the fields in the boot sector.

Offset	Description	Size (bytes)
0x00	Jump code + NOP	3
0x03	OEM name	8
0x0B	Bytes per sector	2
0x0D	Sectors per cluster	1
0x0E	Reserved sectors	2
0x10	Number of copies of FAT	1
0x11	Maximum root directory entries	2
0x13	Number of sectors in partition smaller than 32 Mbytes	2
0x15	Media descriptor (F8H for hard disks)	1
0x16	Sectors per FAT	2
0x18	Sectors per track	2
0x1A	Number of heads	2
0x1C	Number of hidden sectors in partition	4
0x20	Number of sectors in partition	4
0x24	Logical drive number of partition	2
0x26	Extended signature (29H)	1
0x27	Serial number of partition	4
0x2B	Volume name of partition	11
0x36	FAT name (FAT16)	8
0x3E	Executable code	448
0xFE	Executable marker (0x55 0xAA)	2
l	TOTAL	512

Table 2. Boot sector fields

Based on information from the boot sector, the offset locations for all the other regions are calculated as shown in Table 3.

Offset	Description
Start of partition	Boot sector
Start + number of reserved sectors	Fat tables
Start + number of reserved sectors+ (number of sectors per FAT * 2)	Root directory
Start + number of reserved sectors+ (number of sectors per FAT * 2) + ((maximum root directory entries * 32) / bytes per sector)	Data area (starts with cluster number 2)

Table 3. Offset calculation for various regions of the disk

File Allocation Table

A partition is divided into identically sized clusters, small blocks of contiguous space. Cluster size may vary, depending upon the size of the partition. Typically, cluster size is between 2 Kbytes and 32 Kbytes. Each file may occupy one or more of these clusters, depending on its size. A file is represented by a chain of these clusters. The clusters in these chains are not necessarily adjacent to one another on the disk's surface and may be fragmented throughout the data region.

FAT is a list of entries that map to each cluster on the partition. Each entry records one of the following information listed in Table 4.

Sector Number	FAT entry	Description
1	0x0002 - 0xFFEF	Address of the next cluster in a chain.
2	0xFFF8 - 0xFFFF	Last cluster of the file (*EOF*).
3	0xFFF7	Bad cluster
4	0x0001	Reserved cluster
5	0x0000	Unused/Free cluster

Table 4. FAT entries

Figure 2 shows the mapping of three files using FAT entries.

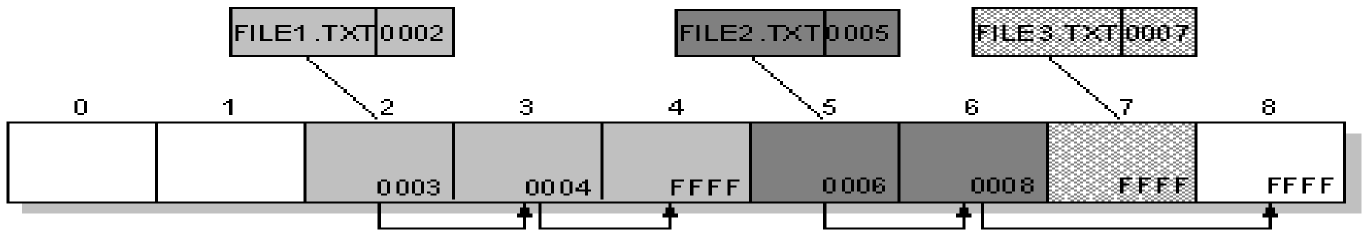


Figure 2. File allocation table

Directory Table

A directory table is a special type of file that represents a directory (commonly called a *folder*). The parameters related to any file or sub-directory are represented by a 32-byte entry in the directory table, corresponding to the parent directory. Each entry records the name, extension, attributes, date and time of creation, address of the first cluster, and the size of the file or directory. The directory table for the root directory is stored in the root directory region location; the directory tables for directories other than the root directory are stored in the data region. Entries with the sub-directory flag set should have a size of 0. The number of entries for any directory of data region can be increased by allocating a new free cluster and updating the FAT accordingly. Table 5 shows various entries of a directory table.

Byte Offset	**Length in Bytes**	**Description**
0x00	8	DOS file name (padded with spaces) The first byte can have the following special values:

Valu e	**Description**
0x00	Entry is available, and no subsequent entry is in use
0x05	Initial character is actually 0xE5
0x2E	'Dot' entry; either '`.`' or '`..`'
0xE5	Entry has been previously erased and is not available

Byte Offset	**Length in Bytes**	**Description**
0x08	3	DOS file extension (padded with spaces)
0x0B	1	File attributes The first byte can have the following special values:

Bit	**Mask**	**Description**
0	0x01	Read only
1	0x02	Hidden
2	0x04	System
3	0x08	Volume label
4	0x10	Sub-directory
5	0x20	Archive
6	0x40	Device (internal use only, never found on disk)
7	0x80	Unused

An attribute value of 0x0F is used to designate a long file name entry.

Byte Offset	**Length in Bytes**	**Description**
0x0C	1	Reserved
0x0D	1	Creation time, fine resolution: 10 ms units, values from 0 to 199.

0x0E	2	Creation time. The hour, minute, and second are encoded according to the following bitmap: **Bits** | **Description** 15-11 | Hours (0-23) 10-5 | Minutes (0-59) 4-0 | Seconds/2 (0-29) Note: Seconds are recorded only to a 2-second resolution. Finer resolution for file creation is found at offset 0x0D.
0x10	2	Creation date. The year, month, and day are encoded according to the following bitmap: **Bits** | **Description** 15-9 | Year (0 = 1980, 127 = 2107) 8-5 | Month (1 = January, 12 = December) 4-0 | Day (1 - 31)
0x12	2	Last access date; see offset 0x10 for description.
0x14	2	EA-index (used by OS/2 and NT)
0x16	2	Last modified time; see offset 0x0E for description.
0x18	2	Last modified date; see offset 0x10 for description.
0x1A	2	First cluster
0x1C	4	File size

Table 5. Various fields of a 32-byte entry in the directory table

Long File Name (LFN) Entries

FAT file systems use a trick to store long file names so that they are compatible with the older operating systems that do not support long file names. It is done by using entries of a special attribute, as follows:

```
ATTR_READ_ONLY|ATTR_HIDDEN |ATTR_SYSTEM |ATTR_VOLUME_ID  =  0x0F
```

Thus, the entries are marked with a volume label attribute, which is impossible for a regular file. Because of this, they are ignored by most old MS-DOS programs. A checksum also allows verification of a long file name. A mismatch can occur if a file was deleted and re-created using DOS in the same directory position. The checksum is calculated using the code shown in Listing 1. Long file names are limited to 255 characters, not including the trailing `NULL` character. The format of an `LFN` entry is shown in Table 6.

```
unsigned char checksum(const unsigned char *shortname)
{
        int i;
        unsigned char sum=0;
                for (i=11; i; i--)
                sum = ((sum & 1) ? 0x80 : 0) + (sum >> 1) + *shortname++;
        return sum;
}
```

Listing 1. Checksum calculation for long file entries

`LFN` entries use the format shown in Table 6.

Byte Offset	Lengt h	Description
0x00	1	Sequence number (if masked with 0x40, indicates that the entry is the last long directory entry in a set of long directory entries. All valid sets of long directory entries must begin with an entry with this mask.
0x01	10	Name characters (five `UTF-16` characters)
0x0B	1	Attributes (always 0x0F)
0x0C	1	Reserved (always 0x00)
0x0D	1	Checksum of DOS file name
0x0E	12	Name characters (six `UTF-16` characters)
0x1A	2	First cluster (always 0x0000)
0x1C	4	Name characters (two `UTF-16` characters)

Table 6. Fields of a long file entry

DOS Command Implementation for FAT16

The code for FAT16 uses drivers supplied with *Interfacing MultiMediaCard with ADSP-2126x SHARC Processors (EE-264)*[1] to perform read or write accesses to MMC. First, the software reads the first 512 bytes (`boot sector` of the MMC) to gather the basic information about the file system. This information, which is copied to internal memory, includes bytes per sector, sectors per cluster, number of reserved sectors, number of FAT tables, number of root entries, total number of sectors on a disk, number of sectors occupied by each FAT table, and so on. These entries are collectively called the `BIOS` parameter block.

Using this information, offsets such as the starting sector of FAT table, the starting sector for root the directory table, the starting sector of first data cluster (cluster number 2), and the total clusters available in the data region are calculated. An error in calculating these offsets may lead to file system crash.

Table 7 lists various commands that are implemented; these commands are described in the sections that follow. Figure 3 shows the interface between MMC and the ADSP-21369 SHARC processor. For details about this interface and the implementation of low level drivers for an MMC card, refer to *EE-264*[1].

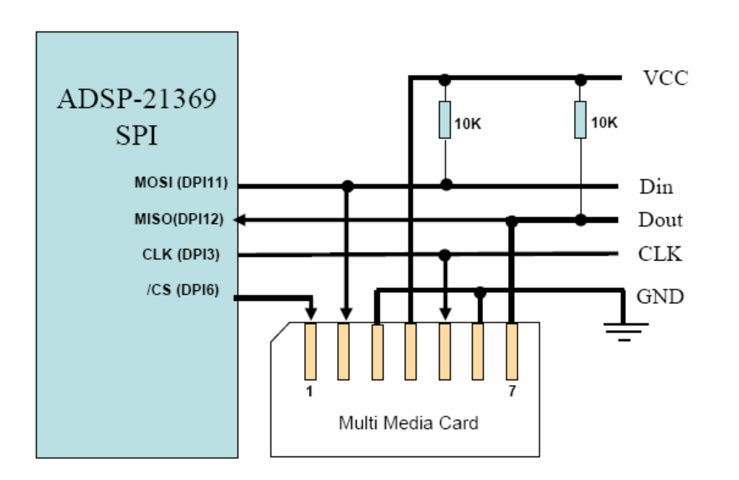


Figure 3. SHARC processor to MMC interface

Command	**Description**
`dir`	Lists the folders and files of the current directory
`cd` *dir_name*	Changes the current directory to the *dir_name* sub-directory of the current directory
`cd..`	Changes the current directory to the parent directory
`root`	Changes the current directory to the root directory
`pwd`	Prints the current working directory path
`find` *file_name*	Searches for all the instances of files named *file_name* and prints all the paths of those files
`mkdir` *dir_name*	Creates a new directory named *dir_name* in the current working directory
`copy` *source_file destination_file*	Copies *destination_file* from the PC and places it in the current working directory of the MMC naming it as *source_file*
`read` *source_file destination_file*	Reads *source_file* on the MMC and copies it to *destination_file* on the hard disk of the PC
`del` *file_name* `del` *dir_name*	Deletes file/directory with name *file_name*/*dir_name* from the current working directory
`exit`	Exits the command prompt

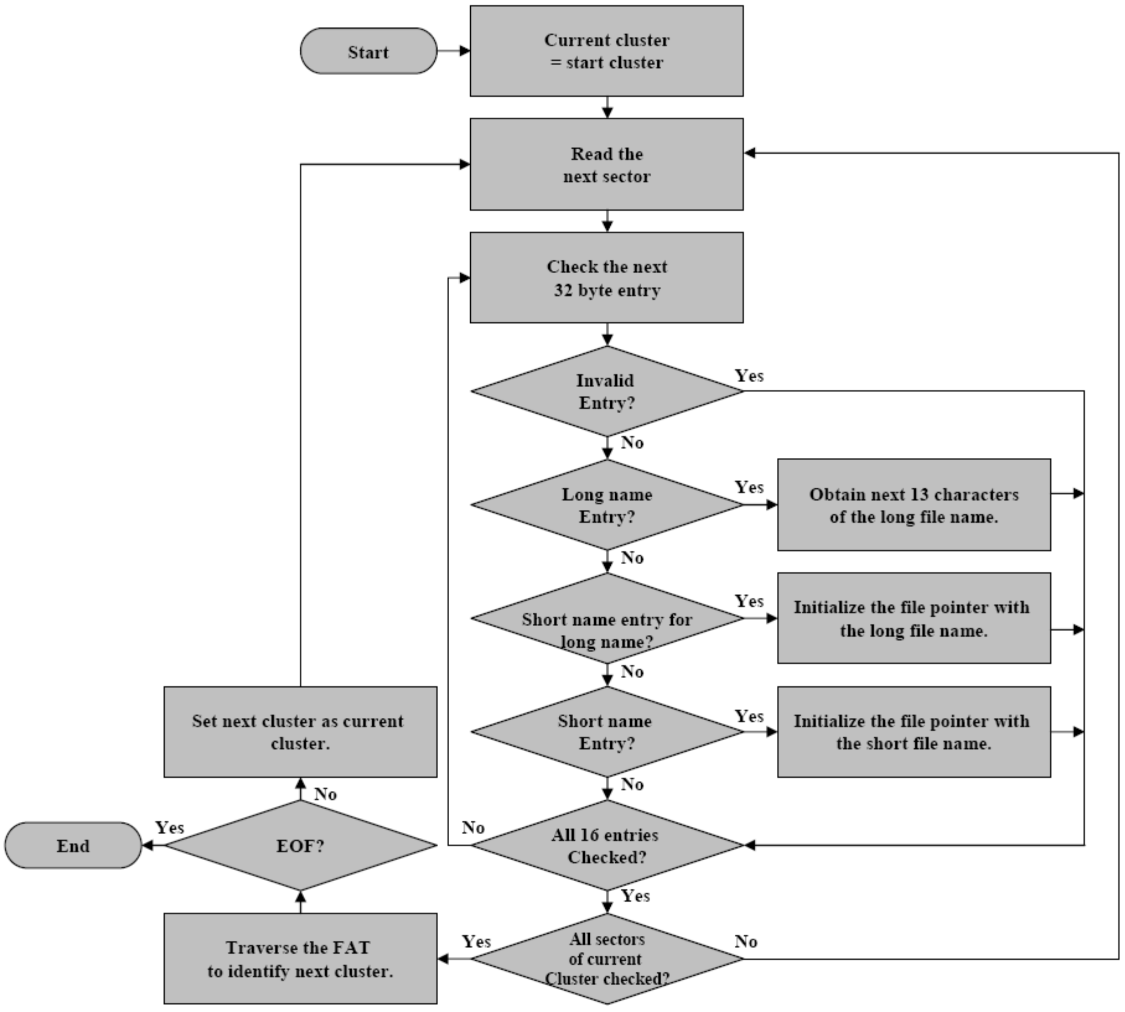
Table 7. List of commands

Listing Files and Directories of the Current Directory – the "dir" Command

A list of file pointers must be initialized to list the files and folders in the current directory. Each file pointer holds the information about a file or folder in the current directory. Figure 4 shows a flowchart for implementing this command. The "detect directory" function (Figure 5) detects the number of files and folders in the current directory. Once the list is initialized, it can be displayed.

The `dir` command uses the `fnListDirectory(WORD startcluster)` function. The start cluster number for the current directory is passed as an argument to this function.

For the root directory, the argument passed is `0`. Note that all the flowcharts are generalized for root directory and directories in the data region. The root directory does not come under the data region, and number of root directory entries is fixed; therefore, it must be treated differently than the directories in the data region. A variable keeps track of the start cluster for the current directory. It is updated each time the current directory is changed. Initially, the current cluster is set to the start cluster. All the sectors of the current cluster are read one by one. Sequential 32-byte entries are read from each sector. The attribute field of each entry is checked to determine the length of file name entry. The entry can be either a short file name entry or a long file name entry. The short file name entry can either correspond to a short file name or a long file name. The current file pointer of the list is updated with the corresponding short file name (for short file name entries) or long file name (for long file name entries) and other attributes such as size, creation time, creation date. Once all the sectors of the current cluster have been read, the FAT table is traversed to identify the next cluster. If there are more unread clusters, the current cluster is set to the next unread cluster and the entire process is repeated. The process ends when an `EOF` (end of file) is identified.



Start
Current cluster = start cluster
Read the next sector
Check the next 32 byte entry
Invalid Entry?
Yes
No
Long name Entry?
Yes
Obtain next 13 characters of the long file name.
No
Short name entry for long name?
Yes
Initialize the file pointer with the long file name.
No
Short name Entry?
Yes
Initialize the file pointer with the short file name.
No
All 16 entries Checked?
No
Yes
All sectors of current Cluster checked?
Yes
No
Traverse the FAT to identify next cluster.
EOF?
Yes
No
End
Set next cluster as current cluster.

Figure 4. Listing a directory

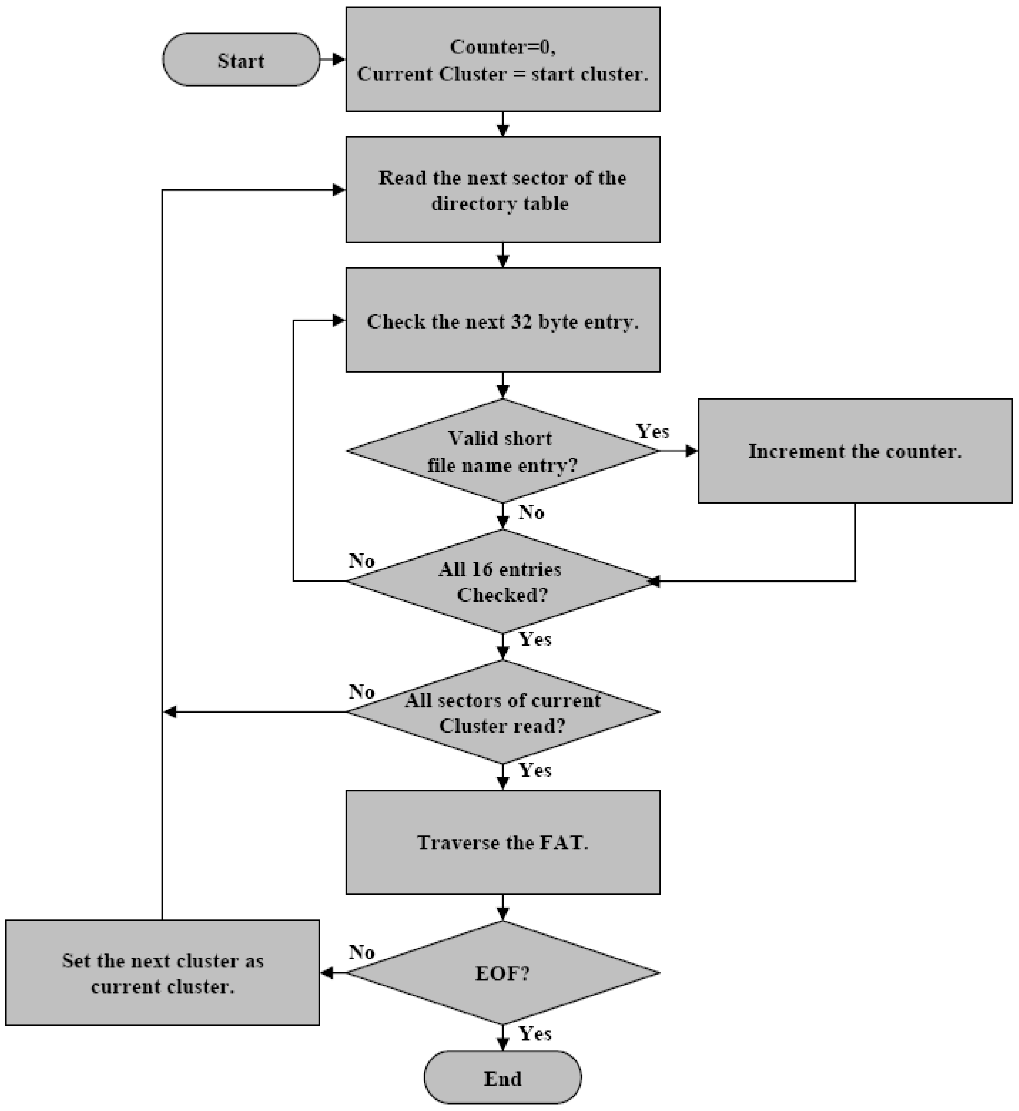

Start
Counter=0,
Current Cluster = start cluster.
Read the next sector of the directory table
Check the next 32 byte entry.
Valid short file name entry?
Yes
Increment the counter.
No
No
All 16 entries Checked?
Yes
No
All sectors of current Cluster read?
Yes
Traverse the FAT.
Set the next cluster as current cluster.
No
EOF?
Yes
End

Figure 5. Detecting the number of files (or directories) in a directory

Searching a File– the "find" Command

File searching is performed recursively. An array of strings stores the path (starting from the root directory) and is updated each time the current directory changes. This is performed for a deeper level search. The recursive function gets the list of the files and folders in the current directory. The attribute of each file pointer of the list is then checked. If it is a directory, the search function is called recursively to perform a deeper level search. If it is a file, its name is compared against the name of the file to be searched. If a match occurs, the current path is printed. Recursion starts with the current directory (as the root directory) and stops when all the file pointers of the current directory have been checked. The flowchart is shown in Figure 6.

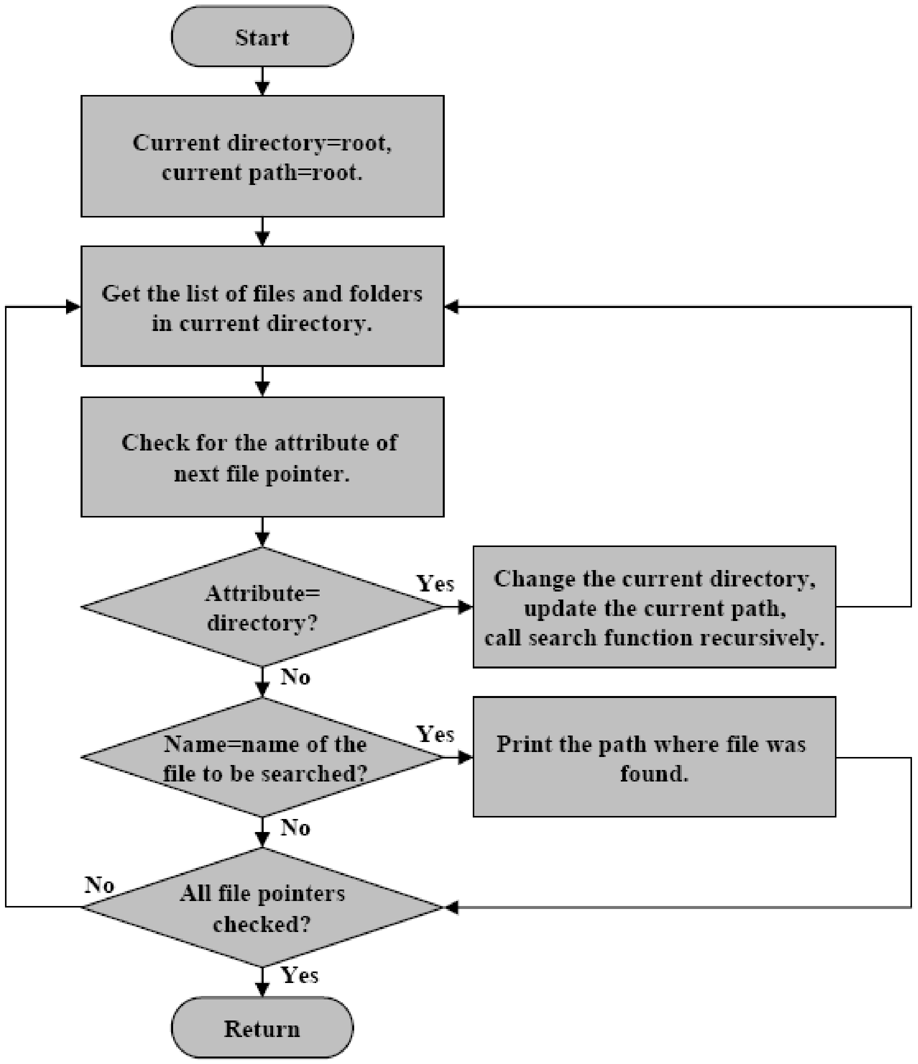

Start
Current directory=root, current path=root.
Get the list of files and folders in current directory.
Check for the attribute of next file pointer.
Attribute= directory?
Yes
Change the current directory, update the current path, call search function recursively.
No
Name=name of the file to be searched?
Yes
Print the path where file was found.
No
No
All file pointers checked?
Yes
Return

Figure 6. Searching a file in the MMC

Creating a Directory – the "mkdir" command

To create a sub-directory in the current directory, first the FAT is traversed to identify the next available cluster. It is necessary to clear this cluster to make all the 32-byte entries as available entries and also to remove any invalid entries due to undefined or junk data. The next step is to find one or more (in case of long file name entries) empty labels to accommodate the information about the newly created directory. Allocate a new cluster to the current directory if sufficient labels are not available and make the corresponding entry in the FAT. Initialize the first two 32-byte entries of a newly created directory with the corresponding "`.`" and "`..`" entries. Figure 7 shows the flowchart for the `mkdir` command.

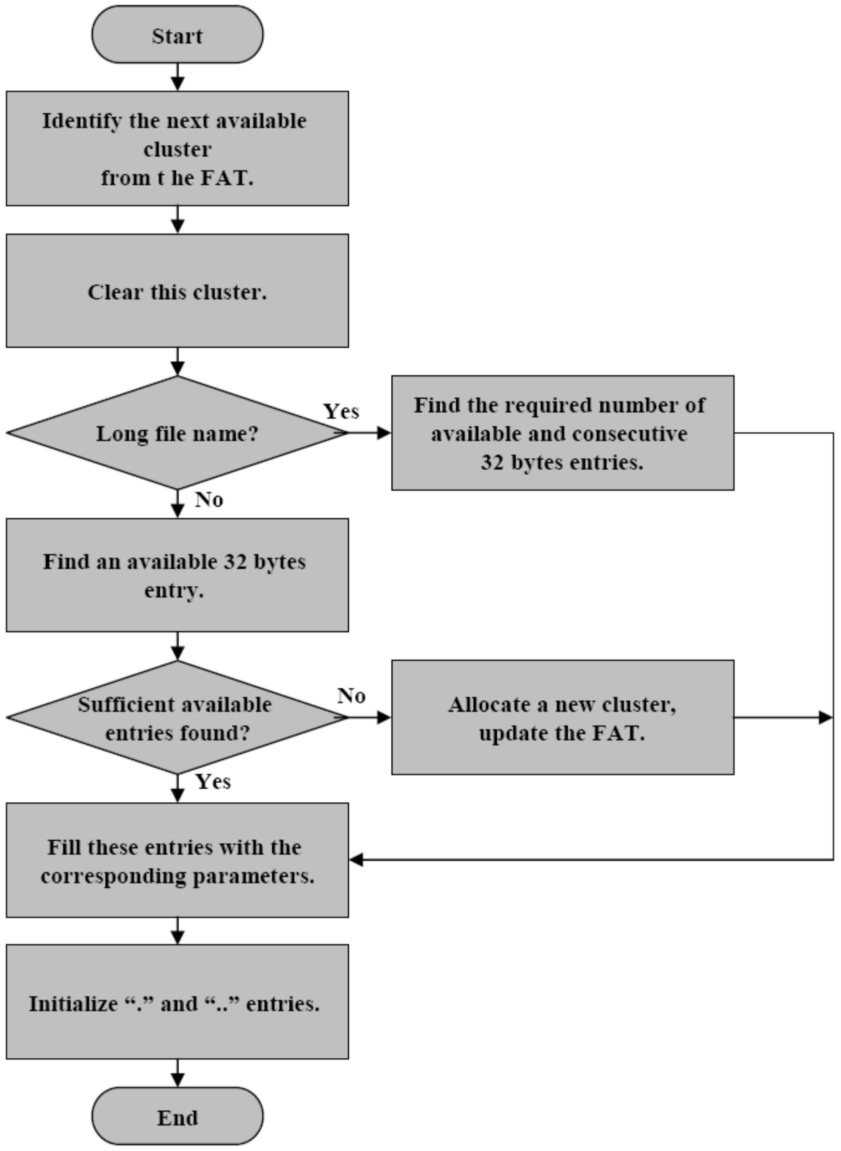

Start
Identify the next available cluster from t he FAT.
Clear this cluster.
Long file name?
Yes
Find the required number of available and consecutive 32 bytes entries.
No
Find an available 32 bytes entry.
Sufficient available entries found?
No
Allocate a new cluster, update the FAT.
Yes
Fill these entries with the corresponding parameters.
Initialize "." and ".." entries.
End

Figure 7. Creating a sub-directory in the current directory

Copying a File to MMC – the "copy" Command

Figure 8 shows the flowchart for copying a file from the PC to the current directory of the MMC. The basic functions used include creating a file in the current directory of MMC and appending a file with a buffer of known length (in bytes). File creation uses the same sequence as creating a directory, except that the cluster allocated need not be cleared and there are no "`.`" and "`..`" entries.

It is necessary to check whether the destination file name already exists in the current directory of the MMC. The source file on the PC is read into a buffer using the file I/O function until the `EOF` is reached. Each time the buffer contains a specified number of bytes, the content of the buffer is appended to the destination file.

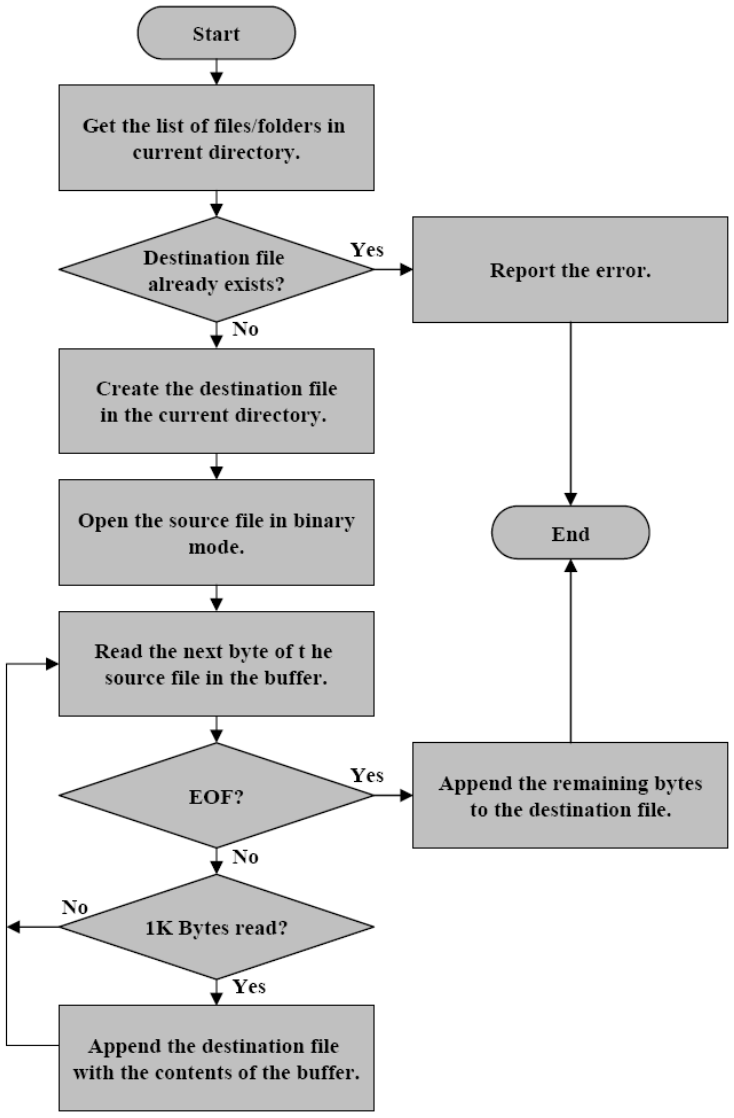

Start
Get the list of files/folders in current directory.
Destination file already exists?
Yes
Report the error.
No
Create the destination file in the current directory.
Open the source file in binary mode.
End
Read the next byte of t he source file in the buffer.
EOF?
Yes
Append the remaining bytes to the destination file.
No
No
1K Bytes read?
Yes
Append the destination file with the contents of the buffer.

Figure 8. Copying a file "source" from a PC to a file "destination" on the MMC

Figure 9 shows the flowchart for appending a known length buffer `"x"` to a destination file. The sector offset of the file and the byte offset of the sector from where the write has to be started is calculated. This can be obtained from the current file size attribute of the file pointer. The file size attribute of that file pointer must be updated with a new size. The current sector is then read into an internal buffer `"y"`. A variable `"remain_write"` tracks the number of bytes left to be written and is initially set to the total number of bytes to be written. If the number of bytes left to be written is more than the number of available bytes in the current sector, a variable `"length"` is set to the number of bytes available in the current sector. If not, `"length"` is set to `"remain_write"` and `"remain_ write"` is set to 0 as there are no bytes left to be written. The number of bytes indicated by `"length"` is copied from buffer `"x"` to buffer `"y"`. Buffer `"y"` is then written back to the MMC at an appropriate sector location. The process is repeated for the next available sectors of the current cluster until all the bytes have been written. If all the sectors of the current cluster are written, the next free cluster is allocated to the file and the FAT is updated accordingly.

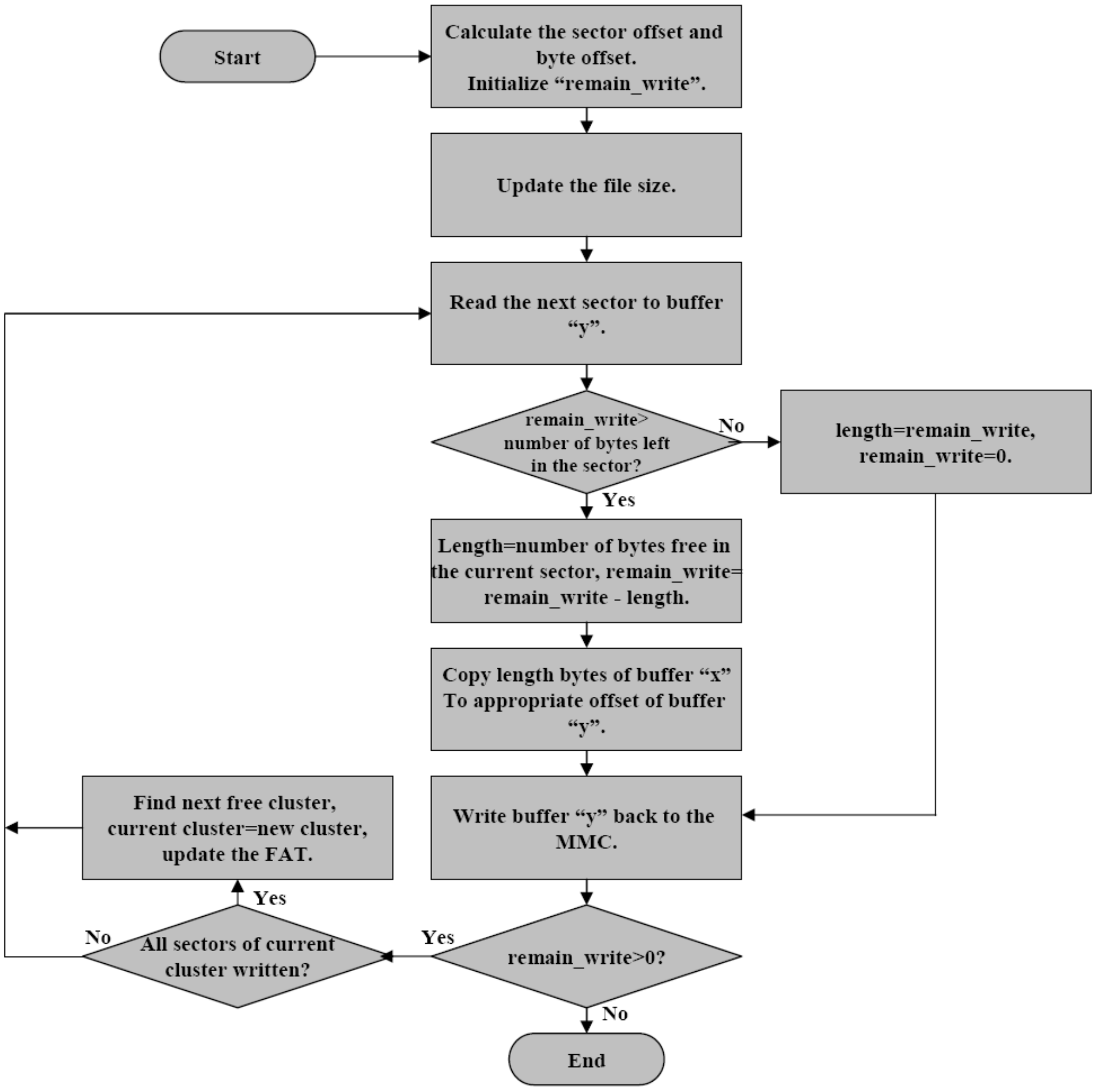

Start
Calculate the sector offset and byte offset. Initialize "remain_write".
Update the file size.
Read the next sector to buffer "y".
remain_write> number of bytes left in the sector?
No
Yes
length=remain_write, remain_write=0.
Length=number of bytes free in the current sector, remain_write= remain_write - length.
Copy length bytes of buffer "x" To appropriate offset of buffer "y".
Find next free cluster, current cluster=new cluster, update the FAT.
Write buffer "y" back to the MMC.
Yes
No
All sectors of current cluster written?
Yes
remain_write>0?
No
End

Figure 9. Writing/appending a file on the MMC

Reading a File from an MMC – the "read" Command

Figure 10 shows the flowchart for reading a source file from the current directory of the MMC and copying it to a destination file on the PC. The main function used in the process is to read a fixed number of bytes from a particular location of a file on the MMC (Figure 6). The list of files and folders in the current directory is initialized using the list directory function. A source file pointer is used to read 1 Kbytes each time from the file on the MMC and to copy them to the destination file.

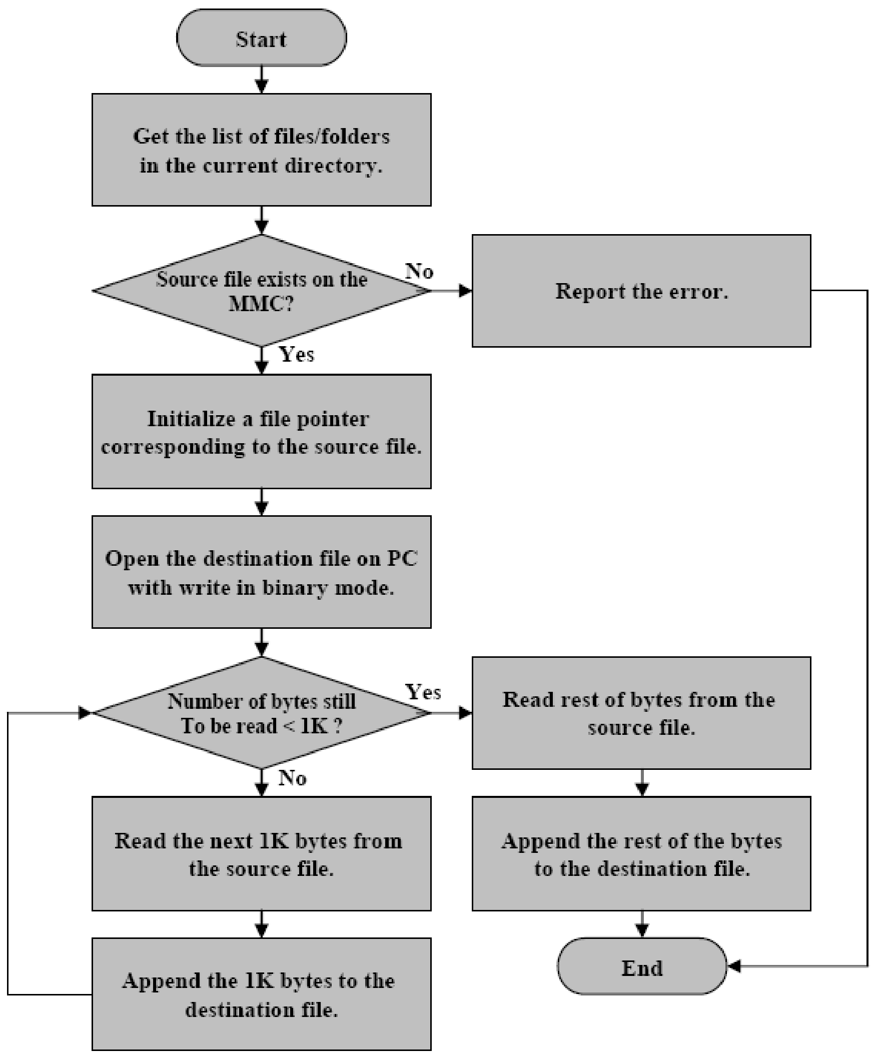

Start
Get the list of files/folders in the current directory.
Source file exists on the MMC?
No
Report the error.
Yes
Initialize a file pointer corresponding to the source file.
Open the destination file on PC with write in binary mode.
Number of bytes still To be read < 1K ?
Yes
Read rest of bytes from the source file.
No
Read the next 1K bytes from the source file.
Append the rest of the bytes to the destination file.
Append the 1K bytes to the destination file.
End

Figure 10. Reading a source file from the MMC and copying it to the destination file

Figure 11 shows the sequence for reading a known number of bytes into a buffer "`x`" from a file on the MMC. The sector offset and byte offset for that sector from where the read of the file has to be started is calculated. A variable "`remain_read`" is initialized with the number of bytes to be read. It tracks the number of remaining bytes. The current sector is read into buffer "`y`". If the total number of bytes remaining to be read is greater than the number of bytes left to be read in the current sector, a variable "`length`" is initialized to the number of bytes left to be read from the current sector. "`remain_read`" is reduced by an equal amount. If not, "`length`" is set to the total number of bytes to be read. The bytes indicated by "`length`" are copied from the appropriate offset of buffer "`y`" to buffer "`x`". If the number of remaining bytes is zero, the process stops. Otherwise, the process continues for the next sector of the current cluster. When all the sectors of the current cluster have been read, the current cluster is set to the next cluster.

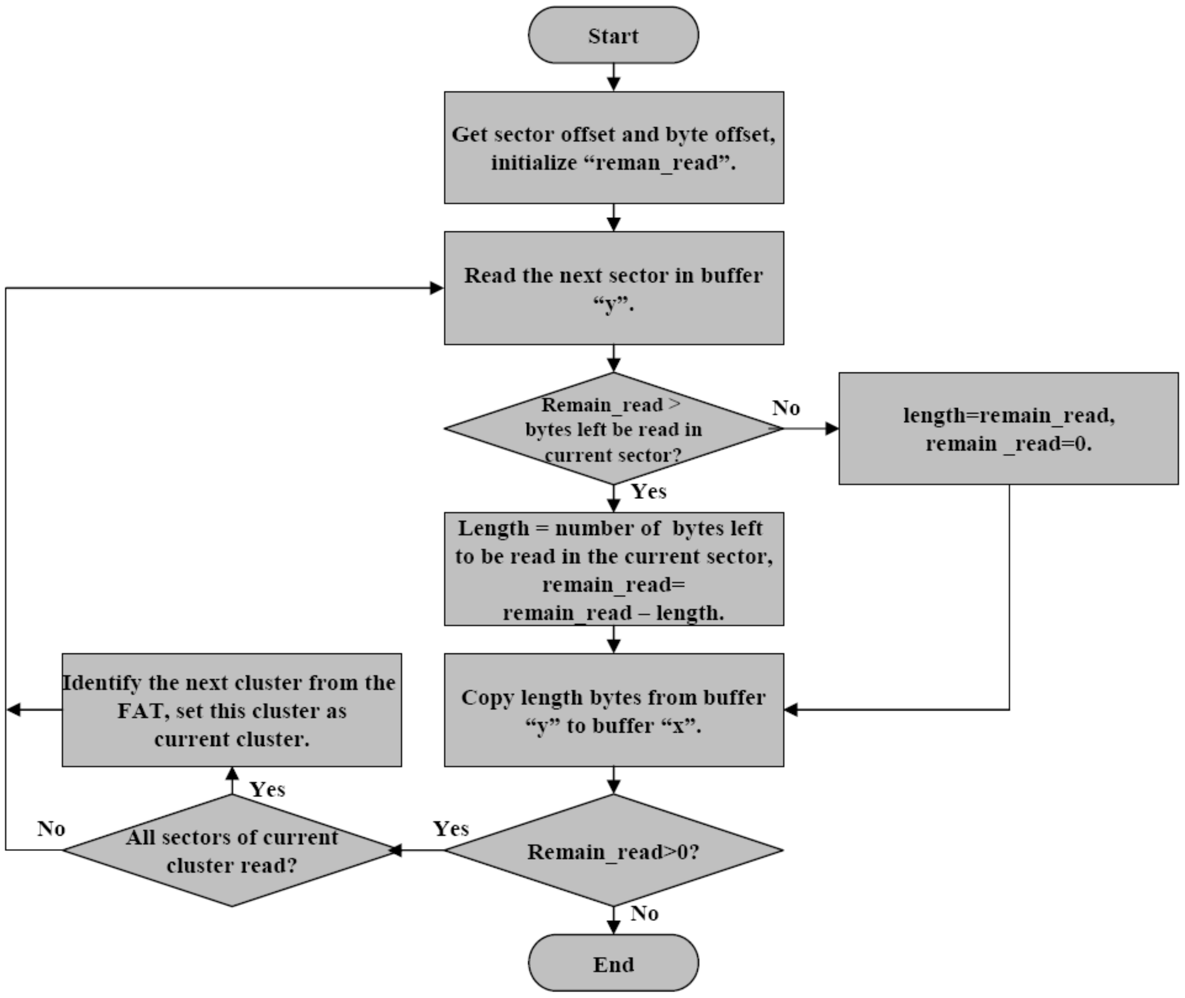
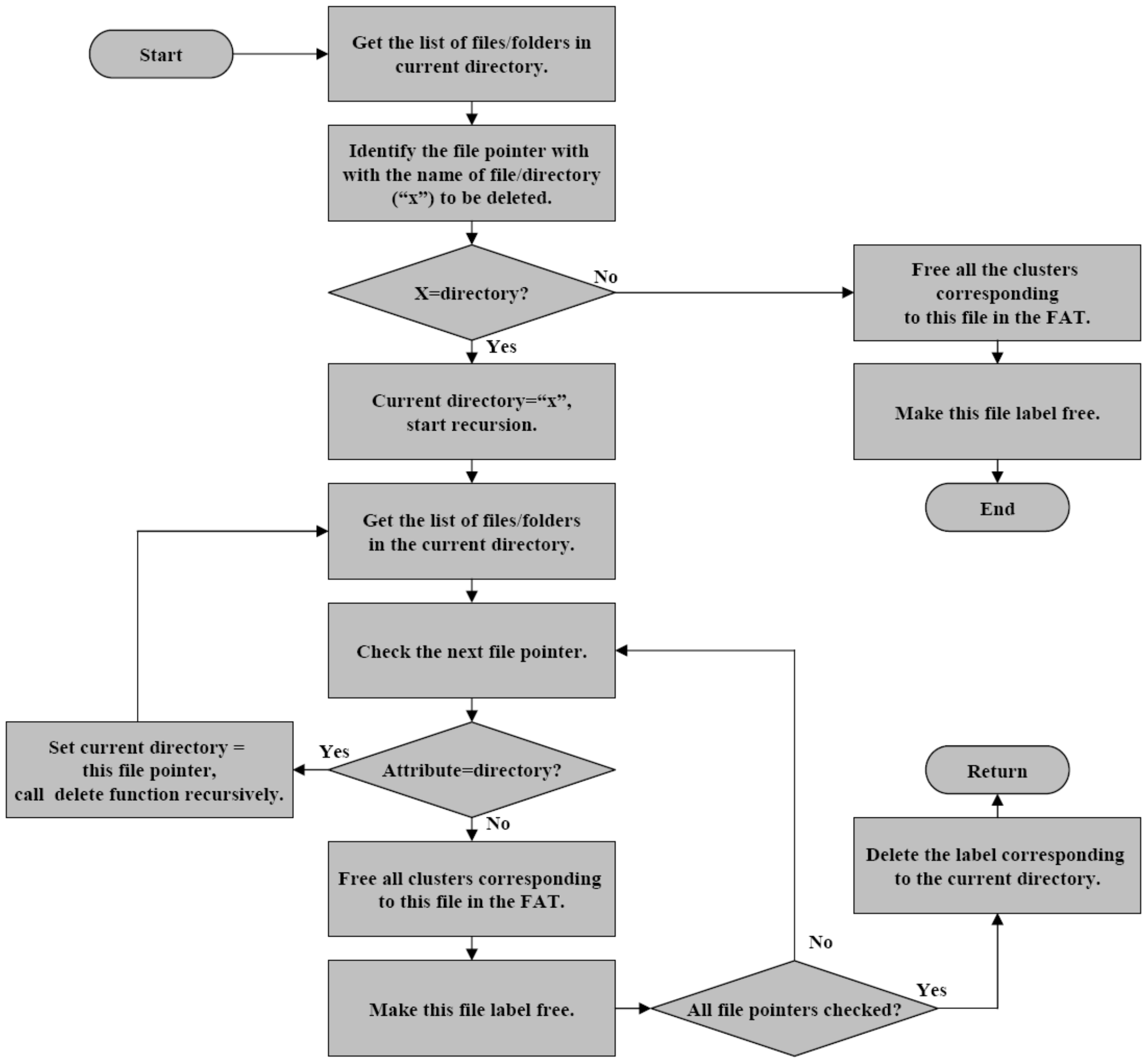

Start
Get sector offset and byte offset, initialize "reman_read".
Read the next sector in buffer "y".
Remain_read > bytes left be read in current sector?
No
length=remain_read, remain _read=0.
Yes
Length = number of bytes left to be read in the current sector, remain_read= remain_read – length.
Identify the next cluster from the FAT, set this cluster as current cluster.
Copy length bytes from buffer "y" to buffer "x".
Yes
No
All sectors of current cluster read?
Yes
Remain_read>0?
No
End

Figure 11. Reading a file from an MMC

Erasing a File or Directory – the "del" Command

Figure 12 shows the sequence to erase a file or directory from the current directory. A list of file pointers is initialized with the files and folders of the current directory. Deleting a file requires freeing up all the clusters allocated to the file by updating the corresponding entries in the FAT. The label corresponding to the file in the directory table of the current directory is made available to be used for a new entry. Deleting a directory is a recursive process as the delete function has to be called for each file or folder of that directory. The recursion is started setting the current directory as the directory to be deleted. The list is initialized with the files/folders in the current directory. Each file pointer is checked for its attribute. If it is a file, it is deleted by the method discussed above. If it is a directory, the same function is called recursively. The recursion stops when all the files/folders of the current directory are deleted and the label corresponding to the current directory is updated to "available".

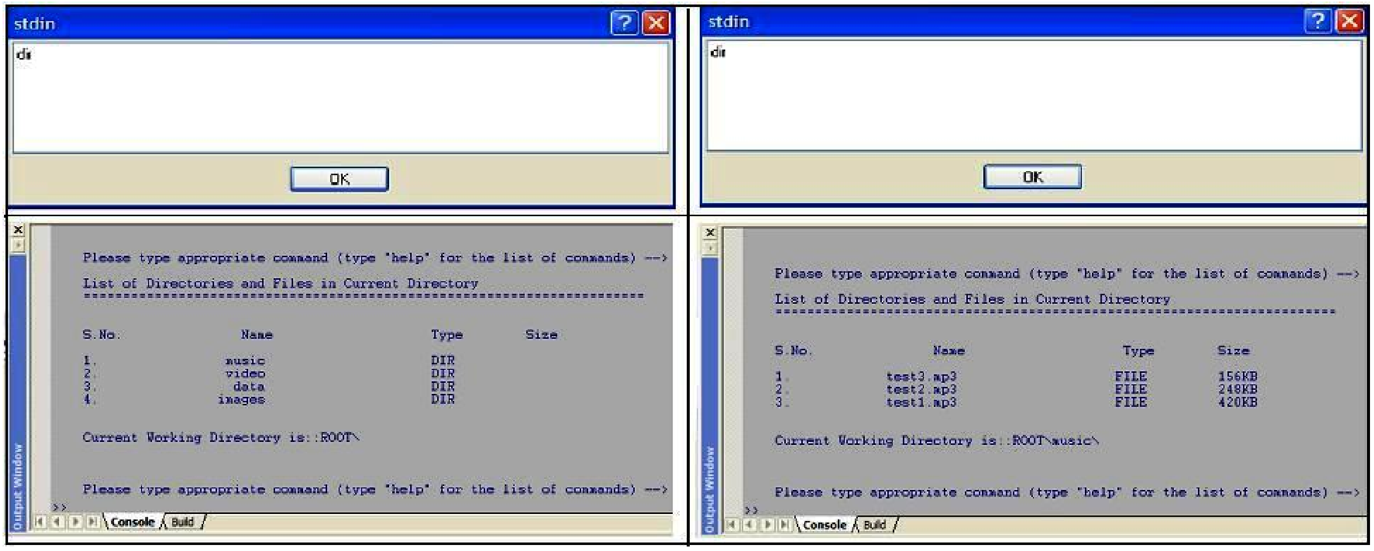


Start
Get the list of files/folders in current directory.
Identify the file pointer with with the name of file/directory (“x”) to be deleted.
X=directory?
No
Yes
Free all the clusters corresponding to this file in the FAT.
Make this file label free.
End
Current directory=“x”, start recursion.
Get the list of files/folders in the current directory.
Check the next file pointer.
Set current directory = this file pointer, call delete function recursively.
Yes
Attribute=directory?
No
Return
Free all clusters corresponding to this file in the FAT.
Delete the label corresponding to the current directory.
No
Make this file label free.
Yes
All file pointers checked?

Figure 12. Erasing a file or directory

Using a Command Window

The following figures show the output of various commands listed in Table 7. An MMC formatted on a PC is interfaced with an ADSP-21369 SHARC processor. The “`stdin`” window shows the command entered; the VisualDSP++® Output window displayed below each “`stdin`” window shows the corresponding output.



```
di
```

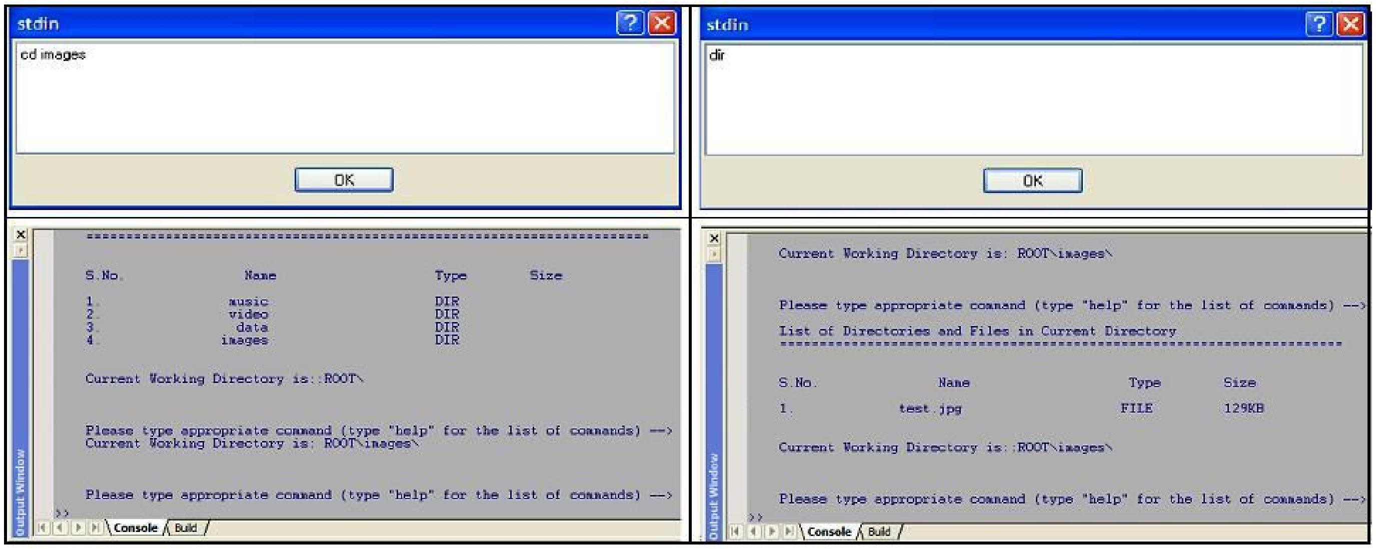
```
Please type appropriate command (type "help" for the list of commands) -->
List of Directories and Files in Current Directory
======================================================================

S.No.               Name                    Type        Size

1.                 music                    DIR
2.                 video                    DIR
3.                  data                    DIR
4.                images                    DIR

Current Working Directory is::ROOT\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
dir
```

```
Please type appropriate command (type "help" for the list of commands) -->
List of Directories and Files in Current Directory
======================================================================

S.No.               Name                    Type        Size

1.             test3.mp3                    FILE        156KB
2.             test2.mp3                    FILE        248KB
3.             test1.mp3                    FILE        420KB

Current Working Directory is::ROOT\music\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 13. “dir” command

The `cd` command in Figure 14 changes the current directory from `ROOT` to `ROOT\images`. The `dir` command also in Figure 14 displays a list of the files in the current directory (e.g., `images`).



```
cd images
```

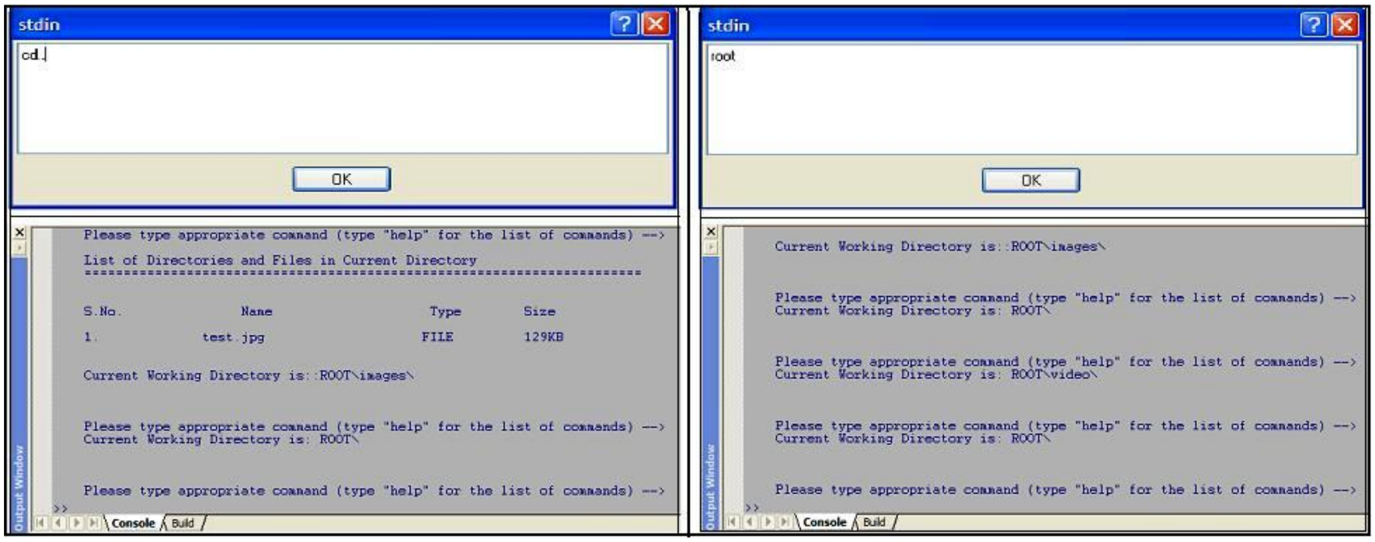
```
======================================================================

S.No.               Name                    Type        Size

1.                 music                    DIR
2.                 video                    DIR
3.                  data                    DIR
4.                images                    DIR

Current Working Directory is::ROOT\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
dir
```

```
Current Working Directory is: ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
List of Directories and Files in Current Directory
======================================================================

S.No.               Name                    Type        Size

1.              test.jpg                    FILE        129KB

Current Working Directory is::ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 14. “cd” command and directory list for current directory

The `cd..` command in Figure 15 changes the current directory from `ROOT\images` back to the parent directory (e.g., `ROOT`). Note that you can also change the current directory directly to `ROOT` by using the `root` command as shown in Figure 15.



```
cd..
```

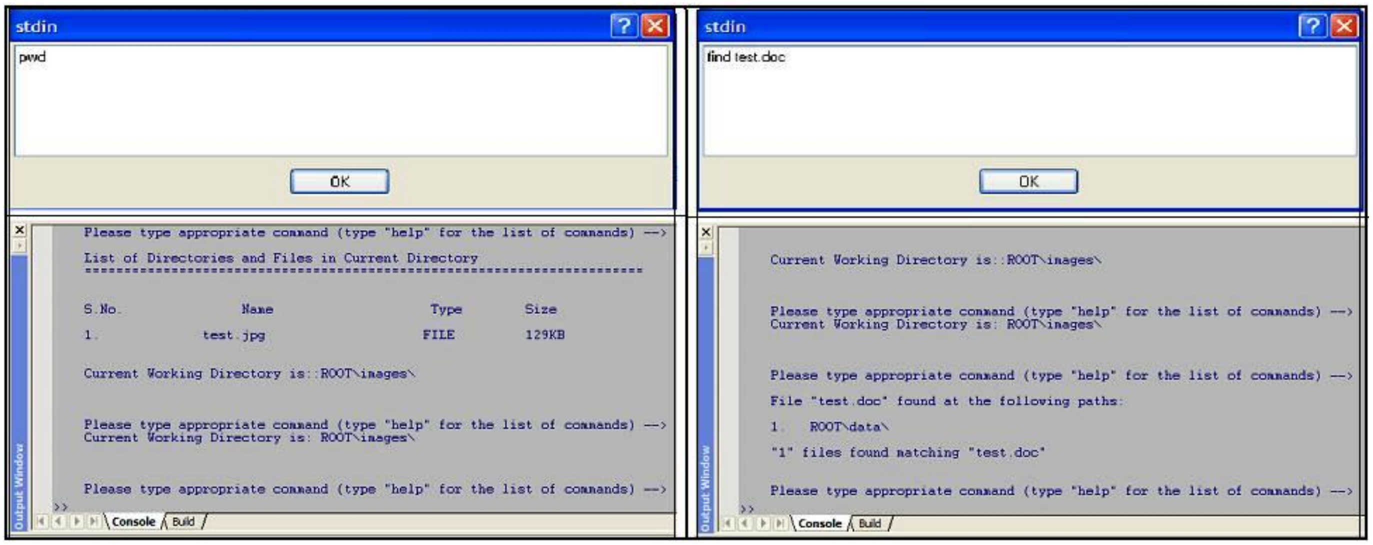
```
Please type appropriate command (type "help" for the list of commands) -->
List of Directories and Files in Current Directory
=======================================================================

S.No.              Name                     Type        Size

1.            test.jpg                      FILE        129KB

Current Working Directory is::ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
root
```

```
Current Working Directory is::ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\video\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 15. "cd.." command and "root" command

Figure 16 shows the `pwd` command, which displays the current working directory. The example in Figure 16 also shows how to use the `find` command to obtain the number of instances and the paths where the file `test.doc` exists on the MMC.



```
pwd
```

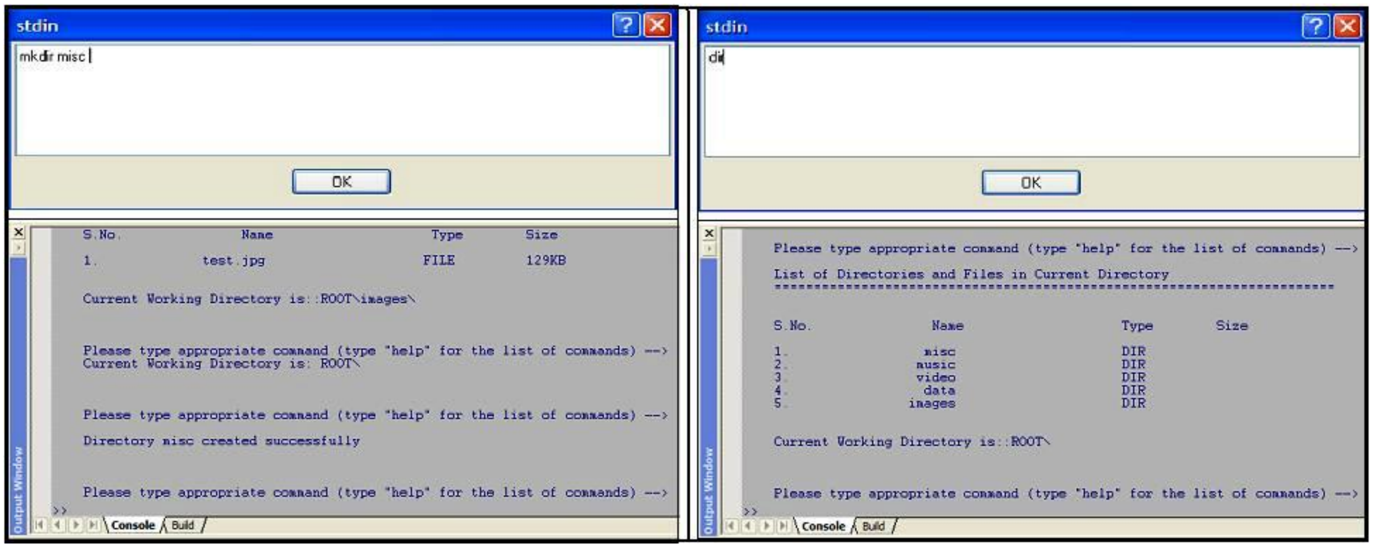
```
Please type appropriate command (type "help" for the list of commands) -->
List of Directories and Files in Current Directory
=======================================================================

S.No.              Name                     Type        Size

1.            test.jpg                      FILE        129KB

Current Working Directory is::ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
find test.doc
```

```
Current Working Directory is::ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
File "test.doc" found at the following paths:
1.   ROOT\data\
"1" files found matching "test.doc"

Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 16. Sample output for "pwd" command and sample output for "find" command

The example `mkdir` command in Figure 17 creates a new directory (named `misc`) in the root directory. Figure 17 also shows how the `dir` command lists the content of the `ROOT` directory, including the newly created `misc` directory.



```
mkdir misc
```

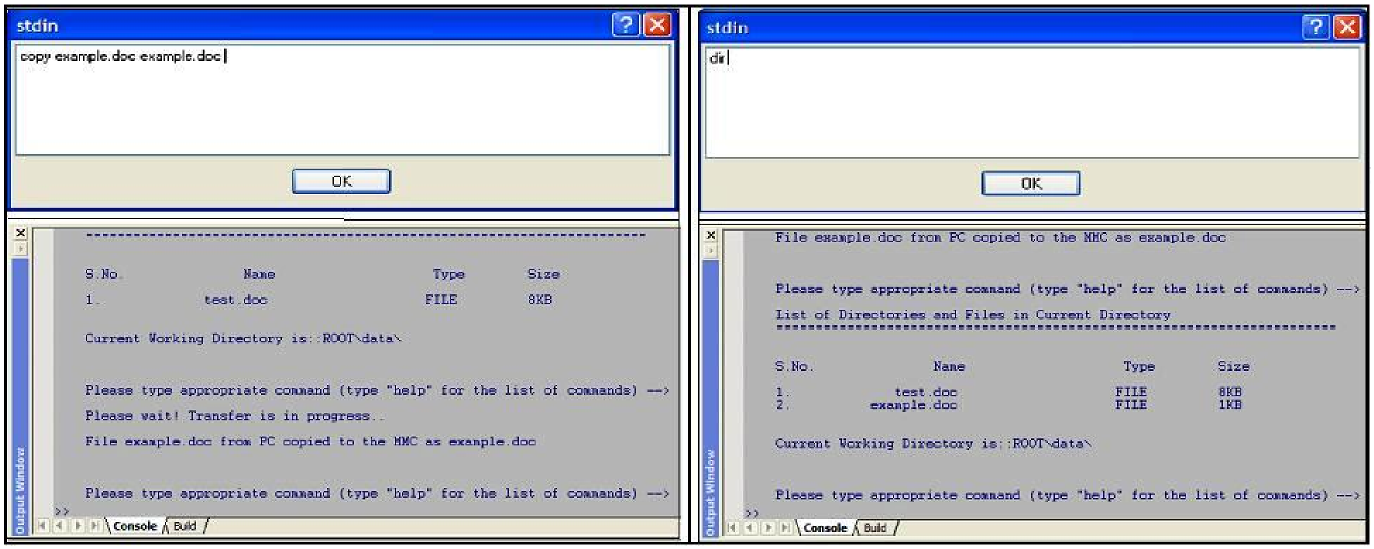
```
S.No.              Name                   Type        Size

1.            test.jpg                    FILE        129KB

Current Working Directory is::ROOT\images\


Please type appropriate command (type "help" for the list of commands) -->
Current Working Directory is: ROOT\


Please type appropriate command (type "help" for the list of commands) -->

Directory misc created successfully


Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
dir
```

```
Please type appropriate command (type "help" for the list of commands) -->

List of Directories and Files in Current Directory
========================================================================

S.No.              Name                   Type        Size

1.                 misc                   DIR
2.                music                   DIR
3.                video                   DIR
4.                 data                   DIR
5.               images                   DIR

Current Working Directory is::ROOT\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 17. “mkdir” command, and “dir” command showing ROOT directory with newly created directory

The example `copy` command in Figure 18 copies the `example.doc` file from a PC to a folder named `data` on the MMC. Note that the file has the same name. Figure 18 also lists the contents of the `data` directory.



```
copy example.doc example.doc
```

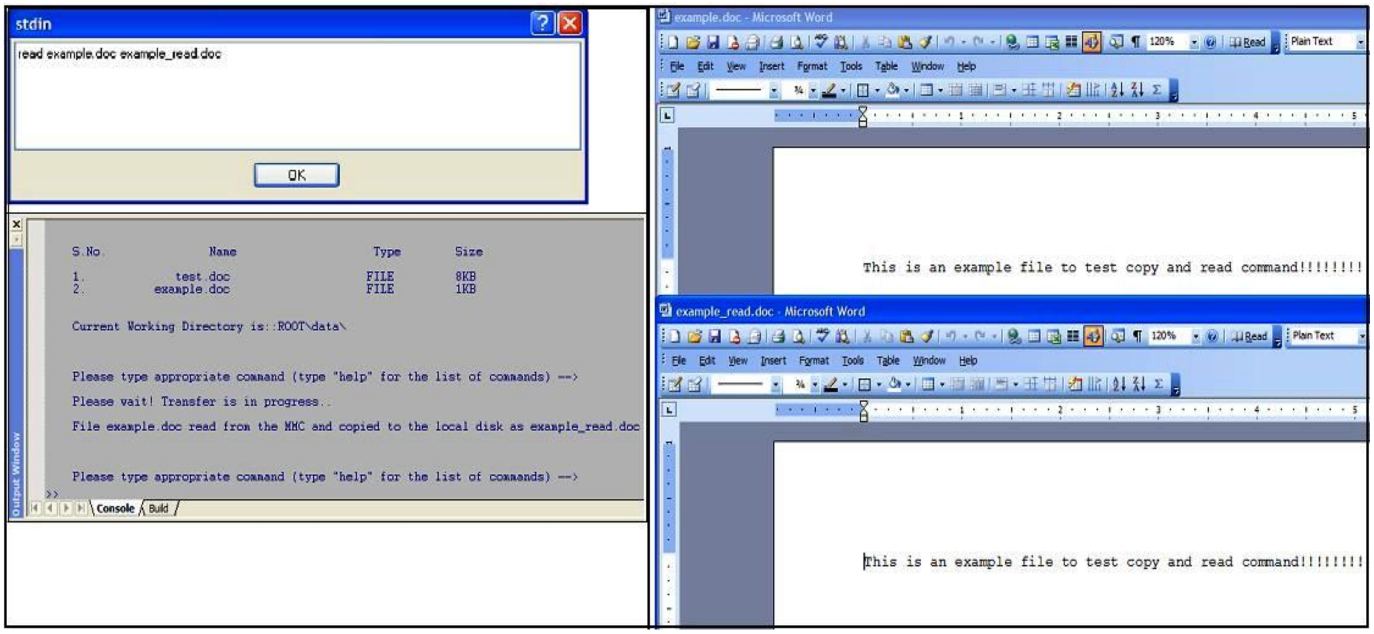
```
------------------------------------------------------------------------

S.No.              Name                   Type        Size

1.            test.doc                    FILE        8KB

Current Working Directory is::ROOT\data\


Please type appropriate command (type "help" for the list of commands) -->

Please wait! Transfer is in progress..

File example.doc from PC copied to the MMC as example.doc


Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
dir
```

```
File example.doc from PC copied to the MMC as example.doc


Please type appropriate command (type "help" for the list of commands) -->

List of Directories and Files in Current Directory
========================================================================

S.No.              Name                   Type        Size

1.             test.doc                   FILE        8KB
2.          example.doc                   FILE        1KB

Current Working Directory is::ROOT\data\


Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 18. “copy” command and current directory showing the copied file

Figure 19 demonstrates a read command. In this example, the example.doc file located in the data folder of the MMC is copied to the PC where it is named `example_read.doc`. The top-right portion of Figure 19 shows the contents of the original file `example.doc`. In this series of examples, this file was copied from the PC to the MMC and then copied back to the PC. The bottom-right left portion of Figure 19 shows the `example_read.doc` file, which was read back.


stdin
OK
example.doc - Microsoft Word
example_read.doc - Microsoft Word
Output Window
Console
Build

```
read example.doc example_read.doc
```

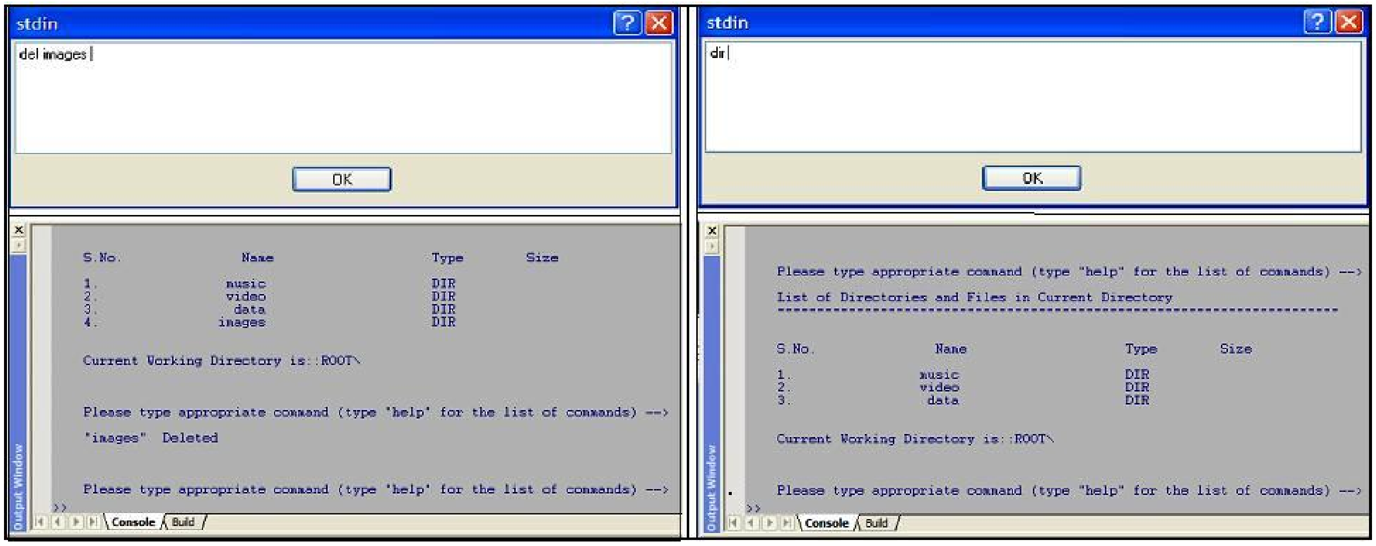
```
S.No.          Name                  Type       Size

1.           test.doc                FILE       8KB
2.         example.doc               FILE       1KB

Current Working Directory is::ROOT\data\

Please type appropriate command (type "help" for the list of commands) -->
Please wait! Transfer is in progress..
File example.doc read from the MMC and copied to the local disk as example_read.doc

Please type appropriate command (type "help" for the list of commands) -->
>>
```

```
This is an example file to test copy and read command!!!!!!!!
```

```
This is an example file to test copy and read command!!!!!!!!
```

Figure 19. "read" command, and comparing original file to file on the MMC

Figure 20 illustrates the use of `del` command to delete the images directory. The second part of Figure 20, which displays the list of folders in the root directory, no longer contains the images directory.



```
del images
```

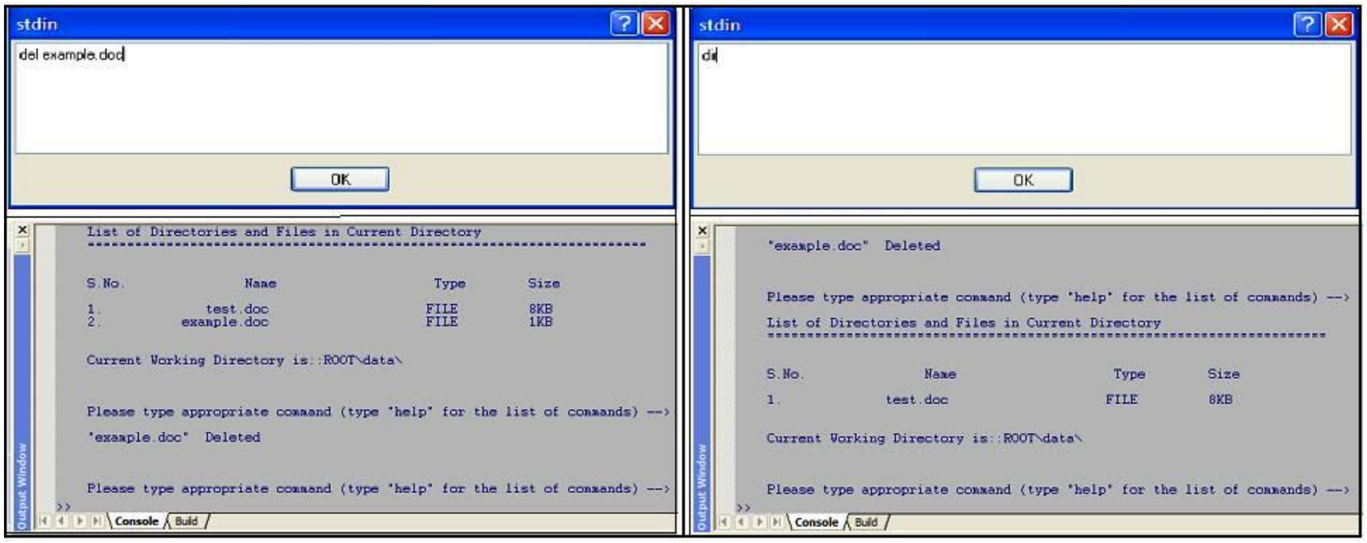
```
S.No.              Name                  Type        Size

1.                 music                 DIR
2.                 video                 DIR
3.                  data                 DIR
4.                 images                DIR

Current Working Directory is::ROOT\

Please type appropriate command (type 'help' for the list of commands) -->
"images"  Deleted

Please type appropriate command (type 'help' for the list of commands) -->
>>
```

```
dir
```

```
Please type appropriate command (type "help" for the list of commands) -->
List of Directories and Files in Current Directory
-----------------------------------------------------------------------

S.No.              Name                  Type        Size

1.                 music                 DIR
2.                 video                 DIR
3.                  data                 DIR

Current Working Directory is::ROOT\

Please type appropriate command (type "help" for the list of commands) -->
>>
```

Figure 20. "del" command and the ROOT directory after deleting images directory

Figure 21 shows the use of `del` command to delete the `example.doc` file from the `data` folder. Figure 21 also shows the list of files in the `data` folder after deleting `example.doc`.



```
del example.doc
```

```
List of Directories and Files in Current Directory
==================================================================

S.No.          Name                    Type        Size

1.          test.doc                   FILE        8KB
2.        example.doc                  FILE        1KB

Current Working Directory is::ROOT\data\

Please type appropriate command (type 'help' for the list of commands) -->
'example.doc'  Deleted

Please type appropriate command (type 'help' for the list of commands) -->
>>
```

```
dir
```

```
'example.doc'  Deleted

Please type appropriate command (type 'help' for the list of commands) -->
List of Directories and Files in Current Directory
==================================================================

S.No.             Name                   Type        Size
1.              test.doc                 FILE        8KB

Current Working Directory is::ROOT\data\

Please type appropriate command (type 'help' for the list of commands) -->
>>
```

Figure 21. "del" command and the resulting data directory after deleting example.doc

Using the FAT16 Plug-In

This section explains how to use the FAT16 VisualDSP++ plug-in. The FAT16 plug-in provides a graphical user interface (GUI) that you use to input commands. The FAT16 plug-in associated with this EE-Note can be used with VisualDSP++ 4.5 (or higher).

To install the FAT16 plug-in in the VisualDSP++ environment:

1. Extract the `FAT16.dll` file from the associated `.ZIP` file (`EE329v01.zip`) and place it in the VisualDSP++ `System` directory. If VisualDSP++ 4.5 is installed on your C drive, copy the attached file into the following directory:

   `C:\Program Files\Analog Devices\VisualDSP 4.5\System`

2. Register the `FAT16.dll` file by typing the following command line:

   `regsvr32.exe FAT16.dll`

   Run `regvr32.exe` from the <*install_path*>`\System` directory, not from the root directory.

The FAT16 plug-in now appears on the `Plugins` page of the `Preferences` dialog box (`Settings -> Preferences`). You can access the FAT16 plug-in from the Tools menu (`Tools -> Plugins`). Figure 22 shows the default state of the FAT16 window.

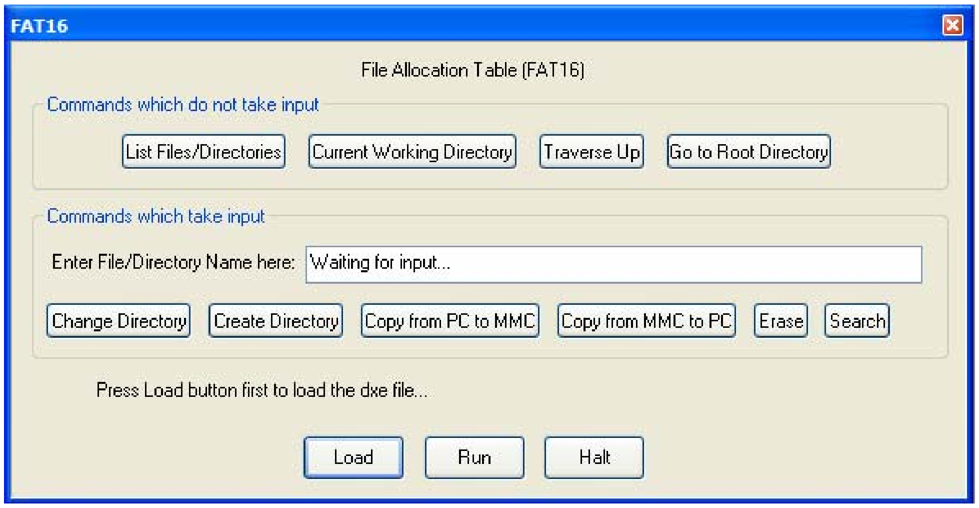

FAT16
File Allocation Table (FAT16)
Commands which do not take input
List Files/Directories
Current Working Directory
Traverse Up
Go to Root Directory
Commands which take input
Enter File/Directory Name here:
Waiting for input...
Change Directory
Create Directory
Copy from PC to MMC
Copy from MMC to PC
Erase
Search
Press Load button first to load the dxe file...
Load
Run
Halt

Figure 22. FAT16 GUI

Commands are grouped into two categories: commands that do not take an input, and commands that require one input (i.e., a file name or directory name, based on the command). Before clicking any of the command buttons, `FAT16_GUI.dxe` must be loaded using the `Load` button. Clicking `Load` opens a file browser window so you can navigate to the folder where `FAT16_GUI.dxe` is copied and can select it. After loading the `.dxe` file, execute the FAT16 application code by clicking the `Run` button. After clicking `Run`, the processor will wait for you to press any of the command buttons. The results of all the command buttons display on the `Console` page for the VisualDSP++ `Output` window. Table 8 briefly describes each command button.

Command Button	**Description**
List Files/Directories	Displays the list of files and directories present in the current working directory
Current Working Directory	Displays the path of the current working directory
Traverse Up	Changes the current working directory to the parent directory of the current working directory. This command has no effect when the current working directory is ROOT.
Go to Root Directory	Changes the current working directory to ROOT. This command has no effect when the current working directory is ROOT.
Change Directory	Changes the current working directory to the user-specified directory. An error message displays when the directory is not found within the current working directory.
Create Directory	Creates a directory with the user-specified name. An error message displays when the directory already exists within the current working directory.
Copy from PC to MMC	Creates the destination file in the MMC (with the user-specified name) using the content of the source file on the PC. Clicking this button opens a file browser window allowing you to browse through the files on the PC and to select a source file. An error message displays when the destination file name already exists within the current working directory.
Copy from MMC to PC	Copies the user-specified file from the MMC to the PC. Clicking this button opens a file browser window allowing you to select the path and the name of the destination file. Windows will prompt you for further action should the destination file already exists on the PC. An error message displays when the source file is not found within the current working directory.

Erase	Erases the user-specified file or directory. An error message displays when the file/directory is not found within the current working directory.
Search	Searches the entire MMC for the exact file name specified by the user and display the paths for all the matching entries found

Table 8. Command descriptions

Summary

FAT16 is a commonly used file system on various memory devices that store multimedia content. Software applications should be able to manage this file system. This EE-Note presented an overview of FAT16 and the implementation of various DOS commands using an ADSP-21369 SHARC processor. The provided source code was tested for an MMC interfaced to an ADSP-21369 SHARC processor. The source code can be used with any other FAT16-formatted memory device connected to the ADSP-21369 processor by using drivers to access it.

References

[1] *Interfacing MultiMediaCard™ with ADSP-2126x SHARC Processors (EE-264)*. Rev 1, March 11, 2005. Analog Devices, Inc.

[2] *Implementing FAT32 File Systems on ADSP-BF533 Blackfin Processors (EE-289)*. Rev 1, February 24, 2006. Analog Devices, Inc.

[3] *Microsoft Extensible Firmware Initiative FAT32 File System Specification*, Version 1.03, December 2000. Microsoft Corporation.

[4] *http://en.wikipedia.org/wiki/Fat*

Document History

Revision	**Description**
Rev 1 – September 28, 2007 by Mitesh Moonat, Jagadeesh Rayala and Aseem Vasudev	Initial Release