Optimization of CF Host Operation

Application Note

Version 1.1

Document No. 80-36-00233

February 2005

SanDisk Corporation

Corporate Headquarters • 140 Caspian Court • Sunnyvale, CA 94089

Phone (408) 542-0500 • Fax (408) 542-0503

www.sandisk.com

SanDisk® Corporation general policy does not recommend the use of its products in life support applications where in a failure or malfunction of the product may directly threaten life or injury. Per SanDisk Terms and Conditions of Sale, the user of SanDisk products in life support applications assumes all risk of such use and indemnifies SanDisk against all damages.

The information in this document is subject to change without notice.
SanDisk Corporation shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Lit. No. 80-36-00233 Rev. 1.1, Edited on 2/28/05

Printed in U.S.A.

Revision History
- *Revision 0.1—First document draft*
- *Revision 0.2—Second review*
- *Revision 1.0—Initial release*
- *Revision 1.1—additions of PIO4-6, DMA 3,4*

Introduction

This application note provides host application developers with insight into SanDisk memory card operations. Developers who use this information will be able to better integrate SanDisk memory cards into their applications for optimum overall application performance.

Developers will be able to better understand their impact on a particular host's behavior by reviewing the architectural and operational details presented in this document. Details begin with a review of each host behavior point of optimization and end with examples of host behavior one should avoid.

Updates will be made to this document when significant changes in memory or card system architecture occur.

Memory Architecture

SanDisk CompactFlash® memory cards are based on NAND flash-memory technology. **NAND flash memory** is composed of groups of memory cells that can be read, programmed, and erased by a memory card controller. Host data is programmed to, or read from, units of sectors (512 bytes) or multiple numbers of sectors. Sectors are erased in large groups commonly referred to as **erase blocks**.

Understanding the detailed implementation of the memory architecture is not critical to understanding the recommendations in this document. However, this information can serve as a basis for showing why our recommendations are necessary.

Data-throughput Efficiency

The highest performance during write and read operations can be achieved when a memory card performs operations on multiple sectors simultaneously. Operation efficiency is dependent on both the data transfer length for the overall operation[1] and the starting and ending boundaries of the operation.[2]

The various reasons behind this efficiency depend on the internal memory card architecture. For example—a memory card may operate simultaneously on multiple physical locations of memory. However, to perform these simultaneous operations optimally, certain logical-address boundary conditions in the memory must be met.

Optimizing memory card operation for performance is accomplished by performing operations that are minimally the size of maximum sector operation concurrency and fall

[1] Sector count.
[2] Usually logical block addresses or LBAs.

on optimal logical address boundaries. More detail for the optimization process is provided in the sections that follow.

Erase Block Architecture

As previously mentioned, NAND flash-memory is divided into erasable units referred to as erase blocks. Each erase block contains a number of host-addressable elements called **sectors,** which are used to store host data. A host sector contains 512 bytes of data. Currently available technologies typically use NAND architectures that contain 32 or 64 sectors in each erase block.

Multiple blocks can be erased concurrently to improve the memory card's maximum write performance. In such cases sector addresses from logically contiguous groups are assigned to multiple, parallel blocks. This increases the effective size of the erase block by a degree equal to the degree of concurrency. For example, Table 1 outlines several SanDisk products and their most effective erase-block sizes.

Table 1. Memory Effective Erase Block Examples

Product	Effective Erase Block
8 MB Standard	8 kB
16 MB Standard	16 kB
64 MB Standard	128 kB
128MB Ultra	128 kB
Future	256 kB or greater

The large difference in size between the effective erase block and the basic sector unit of host access sometimes requires the card to perform internal maintenance operations. Such operations copy valid data sectors from one erase block to another in order to keep some number of blocks available for subsequent write operations. As a result, average system performance suffers when these copy operations occur. The larger the data group copied, the greater the negative impact on average performance.

Although most SanDisk products are optimized to handle write operations that are not aligned to block boundaries, it is more efficient when the first sector of a write command aligns to the first page in an erase block. In addition, writing as large of a portion of data as possible adds to the efficiency. More details for these recommendations can be found in sections that follow.

Host Behavior Recommendations

Large Sequential Write Commands

The only way to sustain the memory card's maximum transfer rate to the host interface occurs when the host uses the minimum number of commands with the maximum number of sectors, per command, to write data to the card to sequential addresses. For example—in the SanDisk CompactFlash® specification, up to 256 sectors can be written with a single command; by writing the maximum number of sectors allowed, the card is able to exercise its maximum level of flash-operation concurrency.

In addition, less overhead-command processing is required and amortized over more data transfers, lowering their impact on performance. Finally, by writing to sequential addresses with successive commands, the internal data structures of the card will need less processing which will further reduce overhead processing during commands.

An example of the sector count's effect on the memory card's maximum write-transfer rate is shown in Table 2. As shown, significant improvements in transfer rate can be achieved with larger sector counts.

Table 2. Sample Write Transfer Rate v. Command Size

Cylinder	Sct Cnt	Speed (KB/s)	Min. Tf Time (ms)	Max. Tf Time (ms)
0	1	205	2.97	3.2
0	2	363	3.3	3.45
0	3	530	3.37	3.58
0	4	683	3.46	3.63
0	5	795	3.7	3.9
0	6	926	3.8	3.94
0	7	1057	3.86	4.1
0	8	1169	3.96	4.1
0	16	1479	3.4	6.5
0	32	2421	6.9	9.9
0	64	3117	11	11.1
0	65	2103	11.15	11.4
0	128	3483	19.2	20.75
0	129	1289	21.7	55.4
0	255	3505	37.6	37.9

All the data for a single command does not need to be contained in a single RAM buffer. SanDisk CompactFlash® products will not timeout waiting for sector transfers from the host. For example—a transfer of 256 sectors may be processed in a single command but the data processed and transferred 16 sectors at a time, if a host has only 8 kB of buffer space.

Host Bus Bandwidth

One factor that affects the maximum sustained write-performance is the host-bus bandwidth. It is important to use the highest host-bus bandwidth possible.

Many host interface specifications for flash cards have various timing modes of operation (for the transfer of data and commands to and data from the card). SanDisk products are designed to operate at the highest timing modes, as specified in the various applicable specifications.

For example—SanDisk CompactFlash® memory cards are designed to support operation using *PIO Mode 4*, the fastest specified timing mode for the ATA interface. The host should take advantage of this capability whenever possible. The benefit is a reduction, in the data transfer to and from the card, as a contributor to the overall duration of a command. Using PIO Mode 4 also increases the memory card's performance. The maximum benefit, however, is achieved on cards with higher intrinsic write-performance.

Table 3. PIO Cycle Timing Summary

PIO Mode	Cycle Time	Transfer Rate (MB/s)
0	600	3.33
1	383	5.22
2	240	8.33
3	180	11.11
4	120	16.67
5	100	20.00
6	80	25.00

Table 4. Multi-word DMA Cycle Timing Summary

DMA Mode	Cycle Time	Transfer Rate (MB/s)
0	480	4.17
1	150	13.33
2	120	16.67
3	100	20.00
4	80	25.00

Higher performance occurs even when the host-interface timing mode is higher than the card-write rate. This is attributed to the overhead in the host file system and the interface drivers. Using the highest timing mode possible increases the overall efficiency. Another way to increase system efficiency is to use a DMA operation: doing so removes the data transfer burden from the host CPU.

File System Optimization

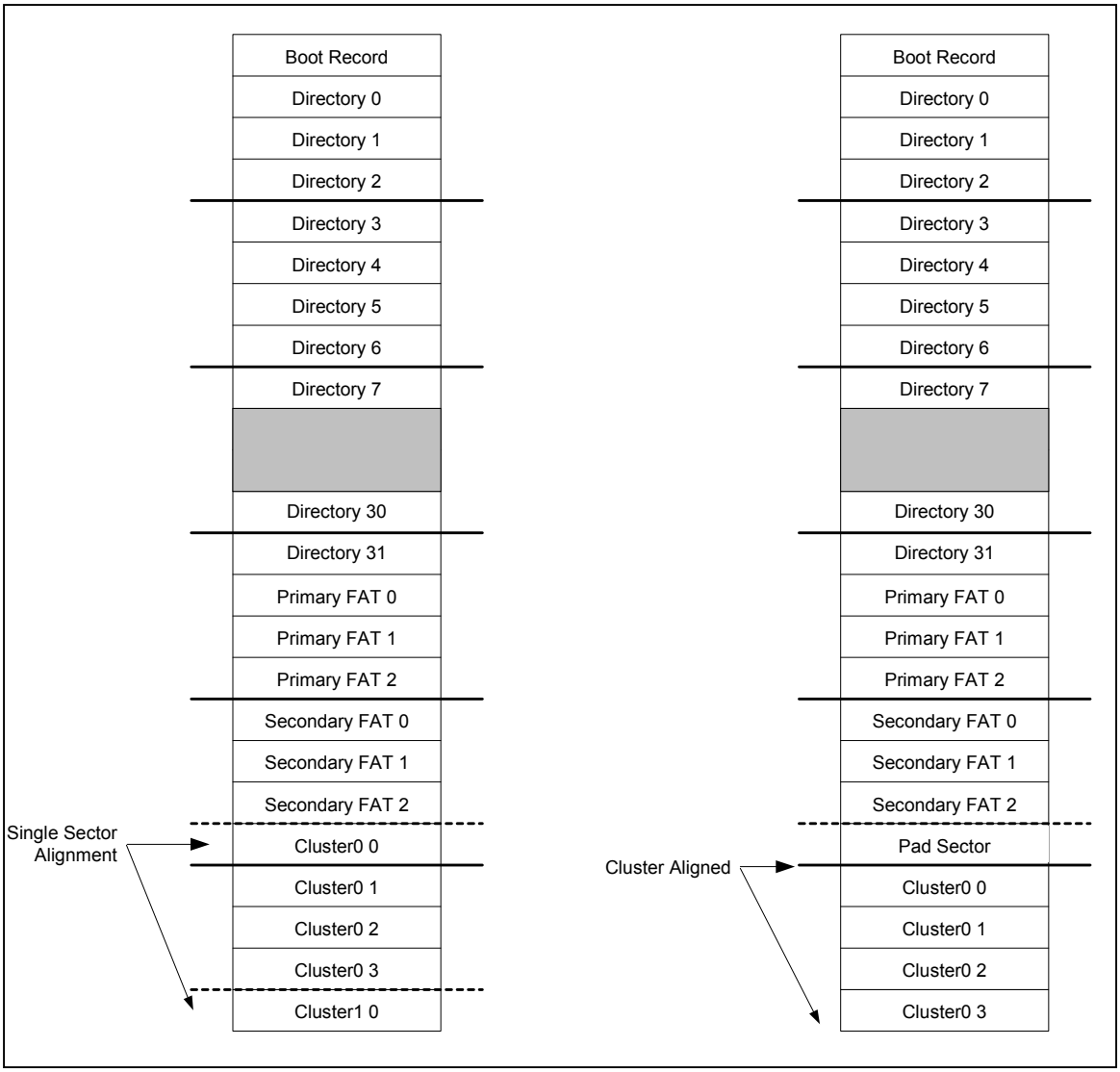
Cluster Boundary Alignment

Typically, write operations are performed on sector clusters defined by the operating system. These clusters occupy the logical address space from slightly beyond the directory and file allocation table (FAT) until the end of the usable storage space. Cluster boundaries should be aligned on a module boundary equal to the cluster size to increase the efficiency of parallel write operations. Misalignment can occur when the number of system area sectors for the boot record, directory, and FAT is not evenly divisible by the number of sectors in a cluster.

Misaligned clusters can cause parts of single clusters to reside in different physical erase block groups. When a cluster is written that straddles physical blocks, it might prohibit the card from writing the maximum data in parallel. It can also cause the memory card to perform internal-data copy operations, which further decreases overall performance.

Figure 1 illustrates two file-system configurations for cards having the same capacity and cluster size of four sectors. The diagram on the left shows data structures aligned in an optimized configuration with no wasted sectors. The configuration on the right shows how the insertion of a single sector can align the data clusters to a cluster-divisible boundary. Clusters aligned with even 256-sector boundaries are optimal. However, when optimal alignment is impossible, alignment to a cluster-divisible boundary should be pursued.

Figure 1 File System Examples of Cluster Alignment (4-sector cluster)



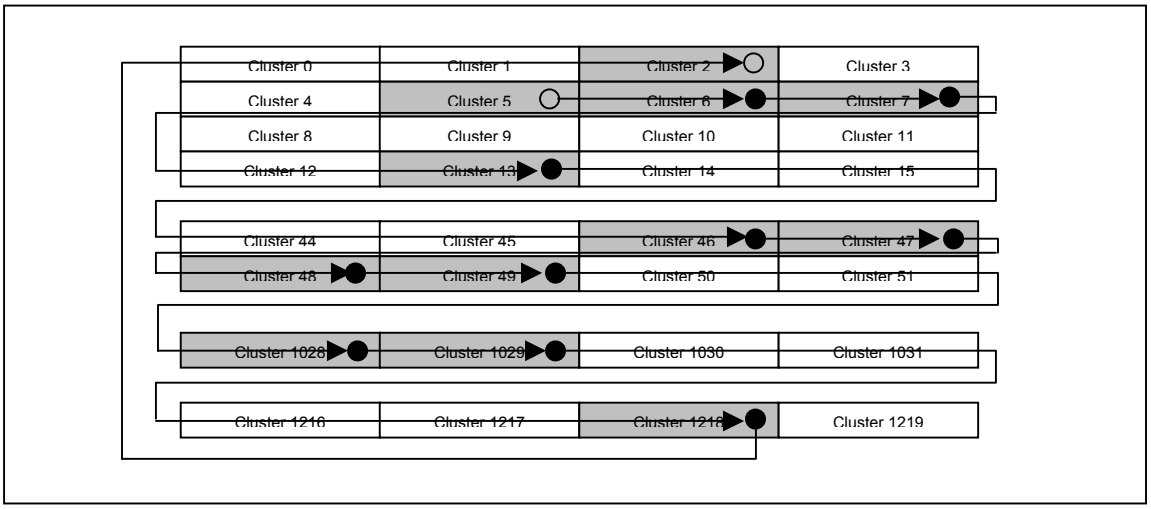
File Fragmentation

The most effective way to use a card is to sequentially write entire physical erase-blocks; doing so avoids file system fragmentation.

If a file system becomes fragmented, clusters from single files become scattered logically, and hence, physically, around a card. Subsequently, when the file is re-written, all physical blocks containing clusters from that file will be updated. The card then needs to perform additional maintenance operations on the data (from other files) contained in those physical blocks. Additional maintenance may involve copy operations that result in reduced overall performance.

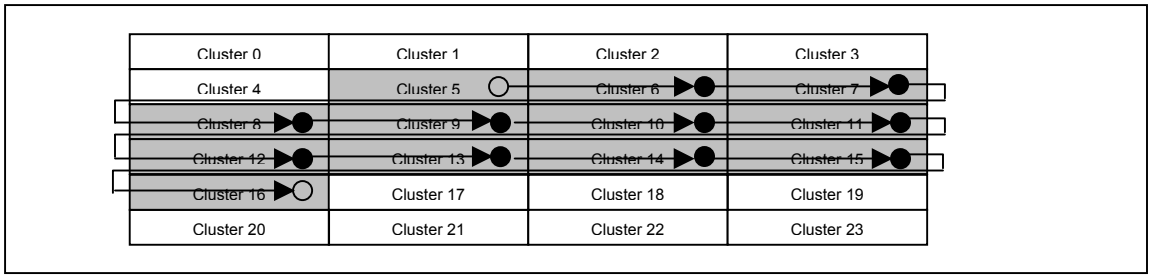
Figure 2 shows an example of a fragmented file in its cluster allocation. The particular file cluster chain illustrated is {5, 6, 7, 13, 46, 47, 48, 49, 1028, 1029, 1218, 2}. If the card had a physical erase-block containing 4 clusters each, potentially 16 clusters would have to be copied internally to the card. The file-write itself is only 12 clusters. That makes the overall operation efficiency less than 43% (worst case).

Figure 2 File Allocation with Fragmentation Example



The diagram in Figure 3 shows how a same-size file can be better allocated in terms of the number of fragmented blocks. In this case, the same 12-cluster file would potentially cause 4 additional clusters to be copied, which could result in an overall efficiency of 75% (worst case).

Figure 3 File Allocation without Fragmentation Example



In most cases, file-system fragmentation cannot be avoided. Periodic file-system de-fragmentation, during periods when such activity is acceptable to the overall system, is a recommended solution.

Minimal System-file Updates

File systems used on flash memory cards typically have system-file structures intended for directory and file allocation. These records are contained in the file system's directory and FAT system files. The records contained in these system files are typically very small and result in update operations occurring with only single sector writes. Some SanDisk products are optimized for such system-file writes that occur to frequently used, repeating addresses.

For severely fragmented file systems or systems containing many small files, such updates may not occur to repeating addresses. Single-sector write operations can result in internal card operations which can cause a maximum reduction in the overall card-write performance. Therefore, system file modifications should be consolidated into the fewest number of write operations possible and occupy the smallest possible range of addresses.

Larger cluster sizes will reduce the number of records required for a specific capacity and, as a result, reduce the overall size of the FAT. As mentioned earlier, using larger cluster sizes provides many other benefits.

Some designs incorporate an update to the FAT for every cluster of the data file written. These types of designs can slow write performance, because the FAT is constantly being erased and re-written. The best design approach would be to write all the file clusters and update the FAT once to avoid multiple erases and re-writes of all the blocks within the erase block.

Small File Writes

Like system file structures, small file storage can reduce the memory card's overall write performance. This is especially relevant when those files are written to a fragmented file system that scatters the files over a wide range of random addresses. Small file types may include picture "thumbnails" from a digital camera or data-log records in an embedded computing application. Because these files are not handled as optimally as system files, they may cause severe latencies due to internal card maintenance operations. These latency events could reduce the average write performance to its lowest level.

In addition, accessing random addresses causes thrashing of internal card data structures used to perform the logical-to-physical address translation. The result is increased card-processing overhead and further reduced average write performance. This type of data-structure thrashing has a negative impact on read and performance.

We recommend, when possible, grouping and writing small files together using fewer write commands. Also, updates to such files should only be performed when absolutely necessary and avoided whenever possible.

Examples of Host-access Behavior to Avoid

The following examples demonstrate the types of host access behavior to avoid.

Example 1: Small Transfer of Misaligned Sectors

Some host systems access the card sequentially but do not use large sector counts. The example below shows an 8-sector sequential write transferring the majority of the data, and single-sector writes transferring some portion of the data.

Example 1 also shows multiple-sector write operations that are not aligned to the boundaries of parallelism. In this case the cluster size is 8 sectors, therefore, the write operations should be aligned to addresses on 8-sector boundaries.

```
Write ADDR:0000097 LEN:08
Write ADDR:000009F LEN:08
Write ADDR:00000A7 LEN:08
Write ADDR:00000AF LEN:08
Write ADDR:00000B7 LEN:08
Write ADDR:00000BF LEN:08
Write ADDR:00000C7 LEN:01
Write ADDR:00000C8 LEN:01
Write ADDR:00000C9 LEN:01
Write ADDR:00000CA LEN:01
Write ADDR:00000CB LEN:01
Write ADDR:00000CC LEN:01
Write ADDR:00000CD LEN:01
```

The net result is an average of 300 kB/s on a card that is capable of a speed greater than 3 MB/s.

Example 2: Interrupted Write Transfer or Misaligned Sectors

The operation in Example 2 takes better advantage of the card's performance capabilities by using large sector count operations (up to 252 sectors). However, due to an internal operational issue, the host performed a 4-sector read-modify-write operation between the large writes.

As in the first example, this application is also not aligning the write operation to even addresses.

```
Write ADDR:00064BB LEN:48
Read  ADDR:0006503 LEN:04
Write ADDR:0006503 LEN:04
Write ADDR:0006507 LEN:FC
Read  ADDR:0006603 LEN:04
Write ADDR:0006603 LEN:04
Write ADDR:0006607 LEN:FC
Read  ADDR:0006703 LEN:04
Write ADDR:0006703 LEN:04
Write ADDR:0006707 LEN:3C
```

The net result is an average of 1.7 MB/s on a card that is capable of speeds greater than 3 MB/s.

Example 3: Repeated Address Write

Example 3 is similar to Example 2 in that due to internal host operational issues, sectors are occasionally written more than once. Also, similar to the previous example, the write stream is fragmented. Again, the write operation sector counts are relatively low. And similar to both of the previous examples, the write operations do not start on even cluster boundaries.

```
Write ADDR:0002617 LEN:08
Read  ADDR:000002A LEN:01
Write ADDR:000002A LEN:01
Write ADDR:000011F LEN:01
Write ADDR:000261F LEN:08
Read  ADDR:000002A LEN:01
Write ADDR:000002A LEN:01
Write ADDR:000011F LEN:01
Write ADDR:0002627 LEN:08
Read  ADDR:000002A LEN:01
Write ADDR:000002A LEN:01
Write ADDR:000011F LEN:01
Write ADDR:000262F LEN:08
Read  ADDR:000002A LEN:01
Write ADDR:000002A LEN:01
```

The net result is an average 13 kB/s on a card that is capable of speeds greater than 1 MB/s.

Example 4: Broken Write Address Sequence

The following sequence shows a host application that skips some addresses and then later returns and backfills the missing sectors. This application, like the others, does not adequately align its write operations to even boundaries.

```
Write ADDR:00010EB LEN:04
Write ADDR:00010EF LEN:11
Write ADDR:0001100 LEN:09
Write ADDR:000110A LEN:3F
Write ADDR:000114A LEN:3F
Write ADDR:000118A LEN:3F
Write ADDR:00011CA LEN:3F
Write ADDR:0001109 LEN:01
Write ADDR:000120A LEN:3F
Write ADDR:0001149 LEN:01
Write ADDR:000124A LEN:3F
Write ADDR:0001189 LEN:01
Write ADDR:000128A LEN:3F
```

The net result is an average of 632 kB/s on a card that is capable of speeds greater than 3 MB/s.