Design of a solid-state file using flash EEPROM

by H. Niijima

This paper presents dynamic sector allocation, clustered sector allocation, and background garbage collection—mechanisms that are key algorithms in solid-state files (SSFs) using flash EEPROMs. Dynamic sector allocation resembles a log-structured file system, which sequentially writes all data modifications to the SSF. Clustered sector allocation is a technique for using the dynamic sector allocation mechanism in an SSF that incorporates a NAND flash EEPROM architecture. Dynamic sector allocation inevitably accumulates obsolete data on the SSF. This "garbage" must be erased in order to secure free space on the SSF. The garbage collection mechanism is a free-space-management method performed as a background process in the SSF. These three mechanisms are closely related and work collaboratively to enable flash EEPROMs to perform well in spite of the serious inherent limitations of the devices. We simulated the behavior of several SSFs and observed that 30% of the storage area of the SSFs must be used as a work area in order to ensure an acceptably low rate of memory-erase operations. We also demonstrated that the lifetime of the SSF is long enough for use in most personal computers. Finally, we have developed SSFs using the NAND type of flash EEPROMs that incorporate the above mechanisms.

Introduction

It has recently been predicted that flash EEPROM (electrically erasable and programmable read-only memory) technology will be a mainstay of the next generation of semiconductor devices [1, 2]. This technology is also expected to be employed to replace secondary storage systems such as hard disk drives (HDDs), rather than simply replacing ROMs and EPROMs (erasable programmable ROMs) [1, 3].

Current flash-memory devices involve some problems that must be solved if they are to be used to realize SSFs (*solid-state files*, i.e., secondary storage systems that use semiconductor memory devices instead of other storage media, such as magnetic storage devices). The most serious problem is that flash devices currently on the market have write/erase endurance (described in the section of flash EEPROM characteristics below) of only 10^6 cycles at most [4]. Consequently, increasing the lifetime of SSFs under this inherent endurance limitation is the most important issue for realizing practical SSFs with flash-memory devices.

In this paper, we deal with SSFs for personal computers (the issues are different when dealing with systems used in business and industry), and our primary target is to develop SSFs that are fully compatible with HDDs for PCs. Typical PC operating systems, such as PC DOS

0018-8646/95/$3.00 © 1995 IBM

and OS/2®, create system areas on HDDs that tend to become write hot-spots; i.e., the areas are modified very frequently. If SSFs using flash EEPROMs should emulate the HDDs as currently designed, write hot-spots would reduce the lifetimes of the SSFs severely. In HDDs, logical sectors are directly and statically mapped to physical sectors on the storage media; for example, the first logical sector is written on the first physical sector of the HDD. To allow flash EEPROMs to be used for SSFs, we propose the *dynamic sector allocation* mechanism, in which logical sectors are mapped dynamically to physical sectors on the SSFs in such a way as to avoid write-hot-spot sectors. With the dynamic sector allocation mechanism, the lifetime of the SSF system can be increased to a practical level.

This mechanism resembles a *log-structured file* system (LFS) [5–10] that writes modifications to disk sequentially in a log or journal structure. Since each successive write operation (log) is performed on the next sequential location (physically), each write causes the overwritten data to become invalid. Over time, all space will be used up, and a "copy-and-compact" operation will be necessary for acquiring free space. Consequently, for LFSs (and also for the dynamic sector allocation mechanism), retrieving information from the log and managing the free space for writing new data are the key issues that must be resolved. The main purposes of LFSs are a) improving write performance by replacing a number of small, random write operations with a single, large sequential write, and b) improving the performance of recovery from a "crash" by using the characteristics of the log structure. The main purposes of the dynamic sector allocation mechanism, on the other hand, are preventing the creation of write hot-spots on the SSFs, and improving write performance by eliminating the erase operation as a direct part of the write operation in flash devices. In the dynamic sector allocation mechanism, the log structure is concealed from the host and maintained by the SSF itself. To alleviate the write/erase restrictions of flash EEPROMs, the log structure is intentionally segmented and scrambled among the devices. This makes it impossible to distinguish new data from old data according to the physical location of the data on the SSF. To allow this mechanism to be adopted for SSFs using NAND flash EEPROMs, we also propose the *clustered sector allocation* mechanism.

Another important key for practical realization of SSFs with flash memory devices is concealing the long erasure time of the devices. For this purpose, we propose a *background garbage collection* mechanism by which the system automatically prepares blank sectors in the background, so that, under ideal conditions, the erasure time can be completely hidden. The background garbage collection mechanism that we implemented in the SSF system has another feature: It evens out the number of erasure operations over all the sectors.

To analyze the behavior of SSFs using the above mechanisms, we wrote a PC DOS file-system simulator and an SSF simulator. For the simulation, we created a technique called the *FAT monitor*. This is not actually part of the control mechanism of the SSFs; it is implemented as a device driver on an operating system. However, it greatly improves the performance of the SSFs by notifying them of the occurrence of erase-file operations on the host.

The paper is organized as follows. The following sections briefly review secondary storage systems using semiconductor devices, and the characteristics of flash EEPROMs currently on the market. This is followed by a discussion of the problems to be solved regarding SSFs using flash EEPROMs and a presentation of the control algorithms of the SSFs. In the next part, we give the results of simulating SSFs using our control algorithms and discuss their implications. Finally, we estimate the lifetime of SSFs using flash EEPROMs.

Secondary storage systems using semiconductor memory devices

Secondary storage systems based on semiconductor memory technology are divided into two categories: those with volatile memory devices, such as SRAMs and DRAMs, and those with nonvolatile memory devices, such as flash devices. Those using volatile memory devices are further classified into two types, depending on whether they are protected against power loss by batteries. Because of the random accessibility for both read and write operations of storage systems based on volatile memory devices, such storage systems are typically used for extended main memory. They can also be used as so-called *RAM disks*. Such storage systems have very simple structures, being organized as flat memory; however, they require special treatment because batteries are needed, except when the secondary storage systems are used solely as temporary memories. Furthermore, the combination of DRAMs and batteries involves a problem of battery durability due to the power consumption of DRAM refresh cycles, while the combination of SRAMs and batteries is very expensive.

Storage systems based on nonvolatile memory technology are suitable for replacing HDDs, since they need not be protected by batteries. Flash memories are the most promising of the nonvolatile semiconductor devices, although they have many practical drawbacks. As described in the following section, they have significant lifetime limitations in comparison with DRAMs and SRAMs; however, with novel control algorithms that overcome the limitations of flash memories, SSFs can enjoy the advantages of flash devices—nonvolatility and lower bit cost than SRAMs and DRAMs.

Among several products utilizing flash EEPROMs on the market is the Intel® FlashFile™ subsystem [11]. This emulates HDDs by means of a device driver on the host computer and a flash card that consists of a plane array of flash EEPROMs. The data on the flash card can be accessed through calls to DOS and BIOS. The file system deals with the flash card as a large flash device; i.e., the written data are accumulated on the card. Once the card is full, the entire contents must be erased explicitly at one time. Since the behavior is different from that of HDDs, the user must remain aware of it.

SunDisk® is a product conforming to JEIDA Ver.4.1 and PCMCIA™ Ver.2.1 with interfaces compatible with the ATA (AT-attachment) [12]. This type of card behaves exactly as do HDDs with ATA interfaces. The SunDisk Corporation manufactures a unique type of flash device [13, 14] that erases and writes cells by 576-byte units. Since the devices can erase and write sector by sector (512 bytes of user data and 64 bytes of system data), SunDisk can directly replace old data with new data. Although the direct replacement is somewhat slow because of the requirement to erase the sector at the time of the sector write operation, it makes the control algorithm much simpler.

Flash EEPROM characteristics

Virtually all current flash EEPROMs can be classified into two main types of devices, NOR and NAND [1–3]. (As mentioned above, the SunDisk Corporation manufactures a unique type of device for its own products.) All types of flash EEPROM devices, however, have the following common characteristics:

- Write ("program") operations can change the value of a bit from 1 to 0.[1]
- Erase operations are needed to change the value from 0 to 1.
- Write operations take much longer than read operations.
- Erase operations take much longer than write operations.
- The number of write/erase operations allowed over the lifetime of each memory cell is strictly limited.
- The memory is divided into *blocks*, and all the cells in a block are erased at the same time.

The limited endurance (number of write/erase cycles permitted) is related to the reliability of the memory cells [15], which are gradually degraded (e.g., the threshold voltage, V_T, shifts) by write and erase operations. It requires more time to write and erase the cells as the number of write/erase operations that have been performed increases. With an excessive number of write/erase operations, the cells cannot be written or erased because of a large V_T shift; in the worst case, the cells are physically broken down. As a result of the V_T shift, data-retention time is also degraded, and errors occur. Since the write and erase times of the cells inevitably change, a verification operation during write and erase operations is required. If the verification indicates a failure, the write or erase operation in progress is repeated a number of times. If the operation cannot be completed successfully within a specified number of retries, the cells are considered to be "broken." Dealing with broken cells is a part of the system implementation. (Sections of storage containing broken cells are marked as bad sectors, as described below.) The system is also responsible for the verification process, but some flash EEPROMs provide automatic verification mechanisms. One should never write or erase cells more times than the specified endurance, even if the verification process is successful, because this may result in read errors after a long time lapse.

In current NOR and NAND devices, the block is much larger than a *sector* (the unit of data access handled by the host system), which is 512 bytes for most current HDDs for PCs. It is also larger than the unit of data for write operations. NOR devices, for example, can write the cells byte by byte and can erase them in 64KB (kilobyte) blocks.

Methods for writing and reading NOR devices are similar to those for accessing SRAMs, since both types of device have random-access capability. Along with this capability, a NOR device can overwrite a byte in any location "bit by bit" without disturbing other cells on the chip. By virtue of the overwrite capability, it can program a byte from the bit pattern 11111110 to 11111100 and then to 00111100 and so on. This capability, of course, does not allow the bit pattern to be changed from 00000000 to 00000001, since the erase operation is needed to change any bit from 0 to 1.

Methods for accessing NAND devices, on the other hand, are very different from those for accessing SRAM and NOR devices. In the case of the Toshiba® 16Mb[2] NAND EEPROM that we use in our SSF product, a block is organized into 16 *pages*, each of which consists of 264 bytes. The page is the unit for reading and writing, while the block is the one for erasure. The device is provided with a 264-byte data register through which the system can access the data for read and write operations. For a read operation, a whole page in the memory block is transferred to the data register at one time; the system then reads the register sequentially. It takes about 25 μs to transfer a page to the register and 80 ns/byte to read the register sequentially. For a write operation, the system first fills the data register sequentially and then transfers the data to a

[1] The standard notation for flash memory cells refers to the programmed state as 0 and the erased state as 1.

[2] The reader should be careful to observe the difference between the notations Mb (megabit) and MB (megabyte).

Table 1 Characteristics of flash EEPROMs.

	Toshiba NAND (16 Mb)	*Intel NOR (8 Mb)*
Write (program)	$1 \rightarrow 0$	$1 \rightarrow 0$
Access unit	Page (264 bytes)	1 byte
Access method	Random by page*	Random by byte
Read access time	25 μs (array to register) 80 ns (serial register access)	85 ns (random)
Write time of sector† (512 bytes)	0.3 ms	3 ms
Erase block size	16 pages (264 bytes each)	64 KB
Erase time of block†	6 ms	300 ms
Write/erase endurance	250,000 cycles	100,000 cycles

*Writing must be sequential by page.
†Excluding possible retry operations.

page, taking 300 μs on average for the latter operation. The architecture allows pages to be read at random, but the location of pages to be written is restricted. The pages must be written sequentially within a block, since writing a page destroys the data in all succeeding pages in the block. This limitation raises the need for a special treatment to invalidate the data sectors in NAND devices; such a treatment is discussed later in this paper.

Table 1 summarizes the characteristics of currently available flash EEPROM devices.

Considerations regarding SSFs that use flash EEPROMs

Many factors must be taken into account before implementing an SSF with the use of flash EEPROM devices. The most important are

- How to extend the lifetime of the SSF, in view of the strict limitation on the number of erase-and-rewrite operations possible.
- How to develop an effective algorithm for the erase operation, so that the erase overhead will be almost hidden.

In order to extend the lifetime of an SSF, emphasis must be placed on avoiding excessive write operations on any memory cells of the SSF. To prevent write/erase operations to a specific cell from exceeding the limit, an *erase count*, which is the count of write/erase operations to a cell, must be maintained for all cells in the SSF. (Since all cells in a block are erased at the same time, the erase count may be maintained only for each block.)

In the PC DOS file system, the file allocation table and directory areas are overwritten whenever a file is written onto a disk. This means that these system areas might reach the maximum allowable erase count and become "worn out" within a few hours. Furthermore, the erase counts for blocks in which work files, such as swap data files, reside increase rapidly and soon reach the maximum. On the other hand, the erase counts for the blocks in which mostly read-only files, such as PC DOS system files, reside are small and rarely increased. When the erase count of a block reaches its maximum, without mechanisms for re-mapping the "worn-out" block to a new one, the SSF is unable to perform further write operations, despite its potential for writing to other blocks. To avoid this situation and to extend the practical lifetime of an SSF, it is necessary to balance the erase count of all physical blocks.

Another issue concerns performance of write operations. If the SSF writes individual sectors to the flash EEPROM as PC HDDs write to their storage media, performance may suffer. Consider, for example, an SSF with a 4KB block and a 512-byte sector. Writing one sector onto the SSF requires the mechanism to manage the sector in the SSF as follows: 1) Copy the block that holds the sector into a work area in volatile memory (e.g., an SRAM) located in the SSF unit. 2) Erase the SSF block. 3) Replace the sector in the SRAM with the new data. 4) Write the block data in the SRAM back to the same physical location as the original block. This procedure obliges the SSF to read and write a 4KB block for every sector write operation, which makes the write performance poor; moreover, it shortens the life of the SSF. The procedure also creates a problem of protecting the integrity of data from an unexpected power failure that might cause the loss of the original data of the block on the SRAM when the block is being erased. A practical write algorithm should avoid this kind of procedure. From a performance point of view, an erase operation should not be a part of the write operation, since it takes from several milliseconds to hundreds of milliseconds to erase one block.

SSF control mechanism

To solve the problems mentioned above, both dynamic sector allocation and background garbage collection mechanisms are used; thus, the limitations of flash EEPROMs are reduced, allowing practical implementation of an SSF.

• *Dynamic sector allocation*

Dynamic sector allocation writes all modifications to the SSF in sequential sectors. The method has two main aims. The first is to prolong the lifetime of the SSF by avoiding creation of write hot-spots anywhere in the flash EEPROM, so that write operations are performed with equal frequency throughout the SSF. The second aim is to reduce the sector write time by eliminating the erase operation as a necessary preliminary to the write operation. The key strategies of the method are as follows:

1. The old data in the physical sector are not erased, but another sector, already erased by the background garbage collection mechanism, is selected for the write operation.
2. The relation between the logical sector address and the physical sector address is stored in an *address translation table* (ATT) on volatile memory, such as SRAM or DRAM, in the SSF unit.

The logical sector address (LA) is managed by the host operating system, while the physical sector address (PA) indicates the location of the corresponding physical sector in the flash EEPROM in the SSF. Since only the ATT relates the LA to the PA, keeping the ATT consistent at all times is the most important issue related to the dynamic sector allocation mechanism. One approach is to use a battery backup for the SRAMs or DRAMs that store the ATT. This requires battery replacement during the SSF lifetime, and might result in loss of data if the battery were improperly replaced or used up. Another approach is to transfer a "snapshot" (an exact copy) of the ATT onto the flash EEPROM every few operations. In this case, the overhead of the storage time of the ATT is not negligible, and the possibility of losing the ATT on account of an unexpected power failure still exists. In addition, increasing the number of write operations in the SSF inevitably reduces the lifetime of the SSF.

Our approach differs from both of the above. We do not keep the contents of the ATT during power-off; instead, the ATT is reconstructed during every SSF power-up sequence. This approach requires information for reconstructing the ATT to be written on the EEPROM devices. Since our SSF is mainly intended for use in PC environments, we cannot expect the "shut-down" operation to take a snapshot or to save any information before system power-off. This means that the information

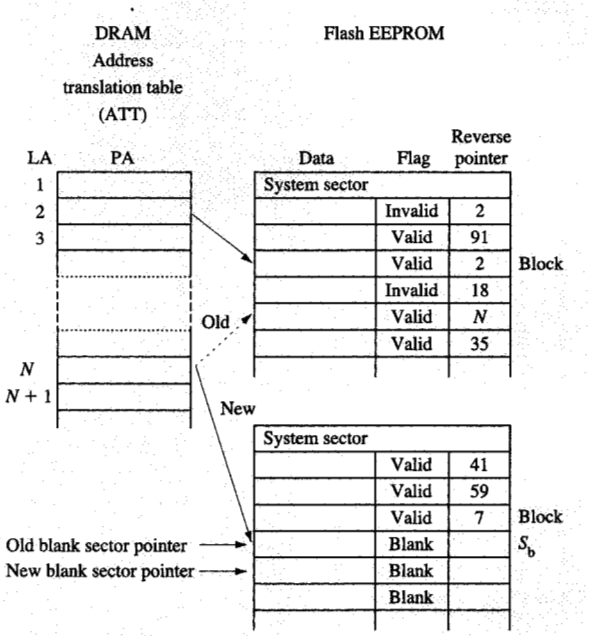

Figure 1

Dynamic sector allocation mechanism for NOR type of flash EEPROM.

for reconstructing the ATT should always be maintained on the SSFs in preparation for power-off. Since power-on and power-off are frequent operations in PC environments, the information must be very robust. The dynamic sector allocation and garbage collection mechanisms work together to maintain the information. The following sections give the details of our approach.

Implementation of dynamic sector allocation depends on the characteristics of the flash EEPROMs used. First we describe an implementation using a NOR flash EEPROM architecture, then one with a NAND flash memory.

Control algorithm using NOR flash EEPROMs

It is comparatively easy to implement dynamic sector allocation when the SSF uses a NOR type of flash EEPROM, since the NOR type has the following advantages over the NAND type:

1. Capability for byte-by-byte random read and write access.
2. No restriction on the order in which memory cells are written.

Figure 1 illustrates the principles of the dynamic sector allocation mechanism for a NOR architecture. It shows a

Table 2 Sector status flags.

Sector status	*Flag*
Blank	111
Being written	110
Valid	100
Invalid	000

DRAM used to store the ATT, and blocks of NOR devices in the SSF, two of which are shown to the right side. The logical sector address is the index into the table, while the table value is the corresponding PA. The first sector in every block is reserved as a *system sector*, and is followed by *data sectors*. The status of a data sector may be *blank*, *valid*, or *invalid*. Blank indicates that the sector has been erased and can be written to. Valid indicates that the sector holds data that can be used. Invalid indicates that the sector holds old data (garbage). The upper block in the figure is shown to be full of data, some valid and some invalid. In the lower block, the first three data sectors are used, and the remaining sectors, still unused, are blank. The physical address at which the contiguous group of blank sectors begins is pointed to by the controller in the SSF. This address is referred to as the *blank sector pointer*. In Figure 1, the blank sector pointer points to the fourth data sector in the lower block. The system sector holds block-management data such as the erase count, described above, and the *bad-sector location map* of the block, which indicates the locations of sectors in the block that are unusable owing to cell failure, if any. Every data sector consists of three components: *user data*, *flag*, and *reverse pointer*. User data are raw data written by the host system. The flag shows the sector status (blank, valid, or invalid). The reverse pointer gives the LA of the user data in the sector; for example, in Figure 1, logical sector 2 is stored in the third physical data sector of the upper block and contains reverse pointer = 2. (Note that the reverse pointer in the first data sector of that block also equals 2, but it contains invalid data.)

Some of the advantages of using the NOR type of flash EEPROM rather than the NAND type lie in flag management. Overwrite operations onto any type of flash EEPROM are ordinarily prohibited; however, an overwrite operation to a byte of a NOR device is possible if no bit in the byte is changed from 0 to 1. Thanks to characteristics 1 and 2 given above and the overwrite capability, the status of a sector is easily modified by consecutively altering its flag from 111 to 110 to 100 to 000, as shown in **Table 2**. The status *being written* shown in the table means that the sector is neither blank, valid, nor invalid. This status is provided to allow resumption of a suspended write operation in the event of unexpected power failure. When a sector with this status is found during the power-up sequence of the SSF, the controller should notify the host system of the existence of a write failure and invalidate the sector by changing the flag to invalid.

The read and write operations are as follows:

Example of a read operation: To read sector 2 (LA = 2),

1. Get the PA from entry 2 of the ATT (PA = the third data sector in the upper block of Figure 1).
2. Getthedatafieldoftheindicatedphysicalsector.

Example of a write operation: To write sector N (LA = N),

1. Find the physical sector pointed to by the blank sector pointer (the sector designated S_b), and change the flag value in S_b from 111 to 110, to indicate that S_b is being written.
2. Write the new data from the host in the data field of S_b.
3. Write the number N in the reverse-pointer area of S_b, to indicate that S_b holds the data for LA = N.
4. Access the ATT to find the PA of the sector where the old data for LA = N are located (the fifth data sector in the upper block in Figure 1), and change the flag of that sector from 100 to 000 to indicate that it is invalid.
5. Change the flag of S_b to 100 to indicate that S_b holds valid data.
6. Write the address of S_b in the Nth entry of the ATT, to indicate that S_b is the physical sector containing the new data for logical sector N.
7. Change the blank sector pointer to the next physical sector. If that sector is marked as a bad sector in the system sector of the block, move the pointer once more. If there are no more blank sectors in the current block, the blank sector pointer should be set to point to the first data sector of the blank block that is always maintained in the SSF by the background garbage collection mechanism.

Since the ATT on the DRAM is lost when the power is turned off, the table must be reconstructed during every power-up sequence of the SSF by scanning both the reverse pointers and the flag fields of all the data sectors. The reconstruction process finds all the sectors with valid flags and places their PAs into the ATT at the entries indicated by the reverse pointers. This process may take about half a second for small SSFs, but it varies significantly, depending upon the flash devices used, the SSF capacity, and the implementation (architecture) of the SSF. The size of the ATT also depends on the capacity of the SSF. In the case of a 32MB SSF, there are 64K sectors. To translate 64K sector addresses, an ATT of 64K entries with 16 bits per entry is required, which equals 128 KB.

Control algorithm using NAND flash EEPROMs

In the Toshiba 16Mb NAND flash EEPROM, 264 bytes (a page) of data are transferred between the data register and the memory array at a time, and access is allowed only on a page basis, not byte by byte. Furthermore, the order of writing the pages in a block is strictly sequential, from the lowest page address to the highest. These restrictions imply that sector invalidation must be achieved in some way other than the simple flag-overwriting operation employed for dynamic sector allocation on NOR devices. Here, the clustered sector allocation mechanism, which is very similar to the dynamic sector allocation mechanism, is employed for recognizing valid sectors. (Note that the objective of sector invalidation is to provide a means for distinguishing a valid sector from others during the construction of the ATT if more than one physical sector contains the same reverse pointer value.) With the clustered sector allocation mechanism, overwriting a sector is not necessary for invalidating old data.

Clustered sector allocation

We define a new structure called a *cluster*. Each cluster consists of a small number of blocks (e.g., 2–16), which is the same for all clusters. The SSF controller erases all the blocks contained in a cluster at one time. In that sense, the cluster is a logical erasure unit, behaving in the same way as the block of Figure 1. The number of blocks in a cluster is one of the key factors affecting the efficiency of garbage collection and the lifetime of the SSF, as we discuss later.

We can use Figure 1 for an explanation of the clustered sector allocation mechanism, noting that this mechanism does not require the flag area shown in the figure and that a block in Figure 1 represents a cluster, since a block and a cluster behave identically. A physical sector on the SSF consists of two pages of the NAND flash EEPROM—528 bytes. Of these, 512 bytes contain user data, and the rest are used for sector management, such as reverse pointer and error correction code (not shown in Figure 1). As in the NOR type of flash EEPROM, the erase counts and bad-sector location maps reside in the system sectors. A unique *cluster sequence number* is also held in the system sector. This number is used by the garbage collection mechanism as follows. The clusters are initially numbered from 1 to B_n (the total number of clusters in the SSF) and are used in sequential order. When a cluster is made blank by the garbage collection mechanism, its former cluster sequence number is no longer used, and it is assigned a number one larger than S_{max}, the largest sequence number currently used in the system. In each cluster, the data sectors are written in order, from the lowest address number to the highest (from the top data sector to the bottom in Figure 1).

Now, suppose that the host requests the SSF to write a sector with LA = N. The SSF controller writes both the sector data and N, the value of the reverse pointer, onto the physical sector to which the blank sector pointer points, and changes the PA value of the Nth LA entry in the ATT to point to the new physical sector. In the procedure, sector invalidation is performed only implicitly, since the cluster sequence number and the fact that physical sectors are written in order within the cluster imply what data sectors are valid. When more than one sector is found to contain the same LA value, the sector in the cluster with the largest sequence number is the valid sector; if more than one sector with the LA value is in that cluster, the sector with the largest PA is the valid one. During the power-on sequence, the ATT reconstruction process is carried out by scanning through all the sectors in succession, from top to bottom, in all the clusters, in order of the cluster sequence numbers. To speed up the reconstruction process, the clusters are sorted, in advance, in order of the cluster sequence numbers.

In the clustered sector allocation mechanism, managing the cluster sequence number is important. **Figure 2** shows how the number is handled when the cluster is made blank

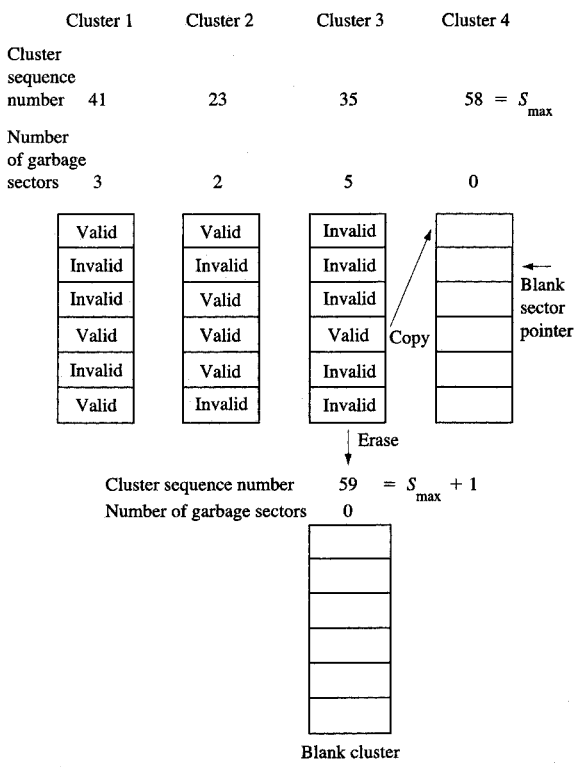

Figure 2

Garbage collection mechanism.

by the background garbage collection mechanism. Let the maximum sequence number of all the clusters be S_{max} (58 in Figure 2), whose initial value is B_n. If cluster 3, whose cluster sequence number is 35, should be erased, the cluster is renumbered as $S_{max} + 1$ (59) at the end of the erase operation. As a result, cluster 3 will have the new maximum sequence number and will be the last of the blank clusters to be used for writing. This numbering algorithm helps prevent any cluster from being written excessively. (The discussion of the background garbage collection mechanism in the following subsection indicates how clusters that otherwise would rarely be written are written more frequently.)

Space must be provided in the system sectors to store the cluster sequence numbers. The maximum value ever attained by the numbers is proportional to the ratio of the total capacity of the SSF to the cluster size. For example, if we choose a cluster of 32 KB for a 320MB SSF, the SSF will contain 10^4 clusters. If the endurance of the flash EEPROMs is 10^6 cycles (the largest value currently given by manufacturers) and all clusters are written an equal number of times, the cluster sequence number may have a maximum value of

$$10^6 \times 10^4 = 10^{10} < 2^{34}.$$

Thus, 34 bits is sufficient to hold the cluster sequence number. The size of a cluster may be set to a few kilobytes, depending upon the block size of the NAND-type flash EEPROMs. It is not practical, however, to choose a small size for a cluster, because the space occupied by the system sector in every cluster is hardly negligible.

In summary: 1) The clustered sector allocation mechanism does not need explicit sector invalidation; therefore, the flag area is unnecessary. 2) The clustered sector allocation mechanism provides a way to prevent any cluster from being written excessively. 3) The clustered sector allocation mechanism requires a sorting operation at the power-up sequence, which may take hundreds of milliseconds.

- *Background garbage collection*

Dynamic sector allocation inevitably accumulates unnecessary data on the SSF, which must be erased in order to provide free space for the SSF. Garbage collection is a free-space-management mechanism performed as a background process in the SSF. The overall performance of the SSF is very sensitive to when and how the free space (blank sectors) is maintained. Free-space management is one of the most difficult issues in log-structured-type systems. Careless garbage collection (the term *cleaning* is used in most LFS implementations) makes data very fragmented. On HDDs, data fragmentation causes significant performance degradation, since HDDs lack random access capabilities. With dynamic sector allocation and garbage collection mechanisms on the SSF, data fragmentation is not a significant concern, because of the random sector-access capability of flash EEPROMs. On the other hand, the mechanism for the SSF must take account of the limited write/erase endurance of flash EEPROMs. Without this limitation, garbage collection could be performed in the background as often as desired with no need to worry about the erase-count management. Three principles of garbage collection are emphasized:

1. Start the garbage collection as late as possible in order to improve the efficiency of garbage collection.
2. Prepare enough blank sectors to accommodate data immediately when requested by the host.
3. Keep the number of erase operations as even as possible over all the clusters.

Here, we use the term *cluster* to denote the unit that is erased at one time, which may consist of only one block. When a cluster is erased by the garbage collection process, in preparation for being used again, the *efficiency of the garbage collection* (E_{gc}) for that cluster is defined as the fraction of garbage sectors in the cluster. (When all the data sectors in the cluster are garbage, the efficiency is 1.) The lower the efficiency, the higher the number of valid sectors that must be copied onto other clusters before the erasure, as we describe below. As the number of copied sectors increases, the lifetime of the SSF decreases. For the sole purpose of improving the E_{gc}, garbage collection should begin as late as possible while satisfying the other principles.

The second principle is important from the point of view of SSF performance. Usually, garbage collection is conducted in the background, to conceal its operation time. Only when the number of data sectors to be written from the host exceeds the number of blank sectors left in the SSF must the host wait for garbage collection. The collection takes more than a hundred milliseconds, since the erasure operation for a block in flash EEPROMs requires several tens of milliseconds. Consequently, should the situation occur frequently, the operation of writing to the SSF will be much slower than in the case of HDDs. To avoid this situation, garbage collection should be repeated as frequently and promptly as possible. (We discuss this contradiction below.)

The third principle is also an important issue for garbage collection mechanisms for SSFs, inasmuch as this mechanism solely determines which cluster should be selected for erasure. In order to write and erase clusters with equal frequency throughout the SSF, there must be a means to erase a cluster that is full of valid data (e.g., a cluster containing a system program).

Clearly, the requirements of the first two principles are contradictory. We have observed that for an SSF using 16Mb NAND flash EEPROMs, keeping between 16 KB and 64 KB of blank sectors in the SSF is sufficient to satisfy the second principle. This value, however, will vary significantly, depending on the balance between the write and erase performances of the flash EEPROMs. When the erase operation is much slower than the write operation, a larger number of blank sectors must be provided in order to keep up with the data writing. In accordance with the above considerations, garbage collection is practiced as follows:

1. Begin garbage collection when
   a cluster is full of garbage, or
   the number of blank sectors falls below a specified threshold value, t_b. (As discussed in the following section, the value of t_b is selected experimentally to keep write-performance degradation negligible.)
2. Select a cluster for erasure, as follows:
   A. If the difference between the maximum and minimum erase counts in the SSF surpasses a specified threshold value, t_e (chosen experimentally), select the cluster with the minimum erase count.
   B. Otherwise, if one cluster has the maximum number of garbage sectors, select it.
   C. Otherwise, more than one cluster has the maximum number of garbage sectors; from them, select the one with the minimum erase count.
3. Copy the first valid sector, if any, in the selected cluster, into the sector to which the blank sector pointer points. Update the blank sector pointer and ATT appropriately. Repeat until all the valid sectors have been copied.
4. Erase the selected cluster.
5. Initialize the cluster. This operation includes writing the updated erase count and rewriting the bad-sector location map in the system sector. When the clustered sector allocation mechanism is utilized, the appropriate cluster sequence number should also be written during this operation.

(Other mechanisms, not discussed here because of our wish to avoid presenting minute details, prevent failures due to power loss during the garbage collection process.)

Figure 2 shows the progress of the process. The assumptions are that $S_{max} = 58$, that only cluster 4 is blank, and that the threshold number of blank sectors to be reserved in the SSF, t_b, is the same as the number of sectors in a cluster (6 in Figure 2). Once a sector has been written into cluster 4, the number of blank sectors is unable to satisfy t_b, so garbage collection starts. If criterion A of step 2 above is not satisfied, cluster 3 is selected for erasure by criterion B. Cluster 3 is erased after the valid sectors (only one in Figure 2) in the cluster are moved into the sector pointed to by the blank sector pointer (the first data sector in cluster 4). When the erasure operation of cluster 3 is finished, S_{max} is incremented to 59, which is the new cluster sequence number assigned to cluster 3.

Because the number of blank sectors is changed only by a write operation, the decision concerning whether or not to start garbage collection is made only after write operations. Once the garbage collection operation has been started, it is carried out with the lowest priority in the SSF; that is, it may be suspended so that write-sector requests by the host can be fulfilled first. Only when all blank sectors have been used up as a result of write-sector requests from the host does the garbage collection operation move into the foreground and become a top-priority process, at which time the host must wait until the garbage collection in the SSF is finished before completing the write-sector requests.

- *SSF simulation*

We wrote an SSF simulator to enable us to analyze the SSF performance. Because our primary target is to develop an SSF that is compatible with HDDs in the PC DOS environment, we also wrote a PC DOS simulator of the behavior of the PC DOS file system. The PC DOS file system maintains a *file allocation table* (FAT) and a *directory*. Each time a file is written, both the FAT and the directory are examined and overwritten. Actually, there are two copies of the FAT in the file system, both of which are updated at almost the same time. The PC DOS simulator simulates this mechanism; that is, three sectors (one for the directory and two for the FATs) are always updated when a file is written or erased. One major difference between this simulator and PC DOS is the method of allocating sectors on the SSF in logical address space. When writing a large file, PC DOS initially uses large contiguous areas on the disk; consequently, areas with small numbers of free sectors tend to be left unused. The PC DOS simulator, on the other hand, does not try to find contiguous areas, even for large files. When allocating space for a file, it always scans the FAT from the beginning and allocates the first free sector it finds (if any) to the file. This scan continues until all the sectors needed for the file have been successfully allocated. The difference between the two allocation algorithms in logical address space will never cause any difference as to where the data are written physically on the SSF, since the SSF writes all data onto flash EEPROM sequentially, in exactly the same way as a log-structured file system does.

The SSF simulator uses every mechanism described in this paper, namely, dynamic sector allocation, background garbage collection, and clustered sector allocation. The

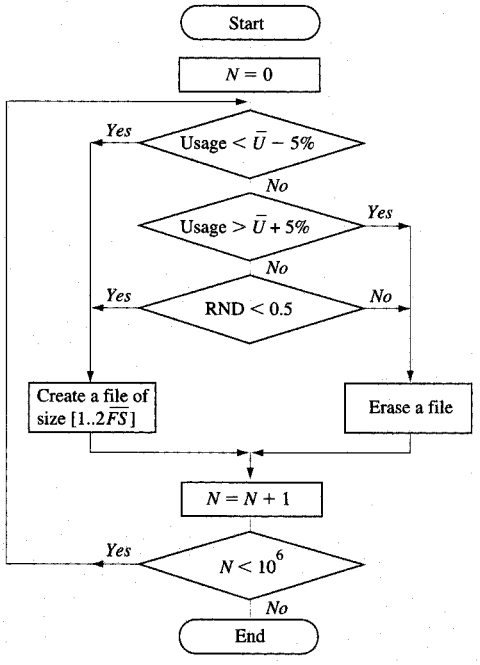

Figure 3

Operation of the SSF simulator. RND = a random number between 0 and 1; [X..Y] = a random integer from X to Y.

threshold value for the necessary number of blank sectors, t_b, is fixed at 64 (32 KB), a value chosen experimentally. If the number of blank sectors goes below t_b, garbage collection is automatically triggered. As explained in the previous section, the smaller the threshold, the greater the write-performance degradation caused by foreground garbage collection. Since time was not simulated, we did not measure write-performance degradation; nevertheless, write-performance degradation due to foreground garbage collection was observed to be almost negligible for the value of 32 KB chosen for the threshold. Throughout our studies, the efficiency of garbage collection, the main subject of the simulation, was not very sensitive to the threshold, unless we selected too large a value (e.g., the same number of sectors as data sectors). During the simulation, we calculated the value of E_{gc},

$$E_{gc} \equiv \frac{\text{number of garbage sectors in a cluster}}{\text{number of data sectors in a cluster}}$$

$$= 1 - \frac{\text{number of sectors copied}}{\text{number of data sectors in a cluster}},$$

and the *average efficiency of garbage collection* ($\overline{E_{gc}}$),

$$\overline{E_{gc}} = \frac{1}{G}\sum_{k=1}^{G} E_{gc}(k),$$

where $E_{gc}(k)$ = efficiency of the kth garbage collection and G = number of garbage collections performed in the simulation. The value of $\overline{E_{gc}}$ is one of the key parameters of both the performance and the lifetime of the SSF, since a low value of $\overline{E_{gc}}$ indicates that a large number of sectors have to be copied. The copy operation increases the write/erase count of the flash EEPROMs and requires a great deal of time, which increases the probability that foreground garbage collection will be needed.

- *Simulation parameters*

The parameters used in the simulations are as follows: redundant area, average file size, average usage, and cluster size. In order for a dynamic sector allocation mechanism to be implemented, some space must be reserved for blank and garbage sectors. For this purpose, the SSF keeps a "redundant" area hidden from the host system; for example, a 10MB SSF might actually have an 11MB capacity, with 1 MB used as the redundant area. One concern is how much redundancy should be provided. We denote the *average file size* $\overline{FS}$ (in number of sectors) and select file sizes at random from one sector to $2 \times \overline{FS}$ sectors to be written onto the SSF. *Average usage* (denoted $\bar{U}$) is the fraction of the *user space* (this excludes the redundant area and system sectors) occupied by valid data. For a specified value of $\bar{U}$, the PC DOS simulator causes the valid sectors to occupy user space in the range from $\bar{U} - 5\%$ to $\bar{U} + 5\%$. (For example, when $\bar{U} = 70\%$, the fraction of valid sectors in user space is restricted to the range 65%–75% by the simulator.) As noted previously, the clustered sector allocation mechanism is implemented in the simulator. Cluster sizes of 16 KB, 32 KB, and 64 KB were chosen. For all the simulations, combinations of file-write and file-erase operations were simulated until a total of 10^6 file creations and erasures had been performed. The flowchart of the simulator is shown in **Figure 3**.

- *FAT monitor*

The SSF fully emulates an HDD but neither manages nor is aware of the logical structure of file-system components such as the FAT and the directory. PC DOS erases files simply by updating the FAT and the directory to indicate that the area previously allocated is now free. If the SSF is ignorant of the fact that a file has been erased, the physical sectors that were occupied by the file will still have the *valid* attribute and be copied into a new cluster during garbage collection. Obviously, this copy operation wastes both time and SSF lifetime. To avoid this situation, we

developed a mechanism called the *FAT monitor* and employed it for this simulation. This mechanism, used in a device-driver layer of PC DOS, monitors modifications of the FAT. When the FAT is modified by an erase command, the FAT monitor instructs the SSF to mark the appropriate data sectors invalid, since they are logically erased. (For NAND architectures, the FAT monitor marks the deleted sectors *invalid* in the ATT entry. Since the garbage collection mechanism refers to the table entry for every copy operation, the deleted sectors are not copied.) The drawback of the mechanism is that it is unable to handle the PC DOS undelete command, which recovers "deleted" files under the assumption that the file itself is left on the recording medium. (Being ignorant of file erasure is an appropriate strategy for SSF hardware. Since it may be used for many types of file systems, SSF hardware should not be aware of the specific file system.)

- *Simulation results*

Figure 4 shows the effect of the FAT monitor mechanism. In all the experiments run, the values of $\overline{E_{gc}}$ when the FAT monitor is used exceed the values when it is not used. When the two lines for $\bar{U}$ = 90% are compared, it is observed that the larger the $\overline{FS}$, the greater the improvement brought about by the FAT monitor. This conclusion is reasonable, since when larger files are erased, the FAT monitor marks more sectors in one cluster as garbage, which improves the efficiency of garbage collection. The remaining simulation results were calculated with the FAT monitor mechanism present.

Figure 4 also shows the effect of $\bar{U}$ on the value of $\overline{E_{gc}}$. Under low usage conditions, there is much free space, which allows a large number of garbage sectors to stay in the SSF, so the SSF controller can delay starting garbage collection. The longer garbage collection is delayed, the greater the accumulation of garbage sectors in the clusters. Thus, when the usage is low enough, high values of $\overline{E_{gc}}$ are expected, even without the FAT monitor mechanism. The result observed concerning varying $\overline{FS}$ is that larger files give higher values of $\overline{E_{gc}}$. If a file is smaller than a cluster, it generally occupies only a part of a cluster (but can overlap two clusters). When the file is erased, even with the FAT monitor mechanism, only part of the cluster is invalidated as garbage. Since files are written and erased randomly in this simulation, the invalidated area is so fragmented that it is difficult for the garbage collection process to find a cluster full of garbage. Thus, the value of $\overline{E_{gc}}$ decreases.

Figure 5 shows the impact of the redundant area on the value of $\overline{E_{gc}}$. In these simulation runs, the size of the SSF (10 MB) excludes the redundant area. The results show that $\overline{E_{gc}}$ increases as the usage decreases and the redundant area increases. As expected, when $\bar{U}$ is 50% or less, even a small redundant area results in an ideal value

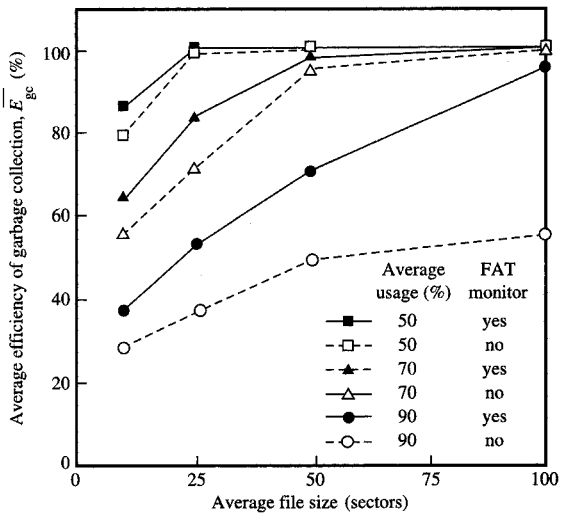

Figure 4

Effect of FAT monitor. This demonstrates the improvement in the average efficiency of garbage collection brought about by use of the FAT monitor. (10MB SSF with an additional 1 MB of redundancy; cluster size = 64 sectors.)

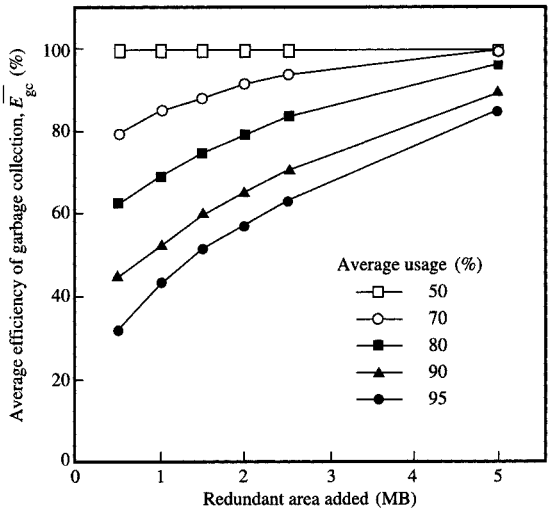

Figure 5

Effect of size of redundant area. This demonstrates the improvement in the average efficiency of garbage collection as the size of the redundant area increases and the average usage decreases. (10MB SSF, excluding redundant area; cluster size = 64 sectors; average file size = 25 sectors.)

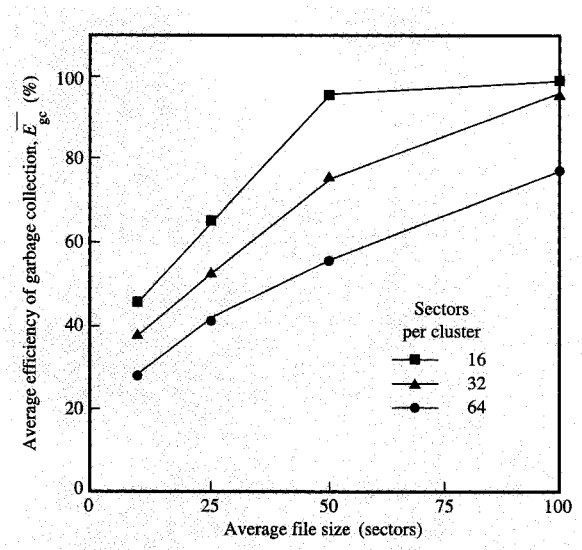

Figure 6

Effect of cluster size. This demonstrates the improvement in the average efficiency of garbage collection as the cluster size is decreased. (10MB SSF with an additional 1 MB of redundancy; average usage = 95%.)

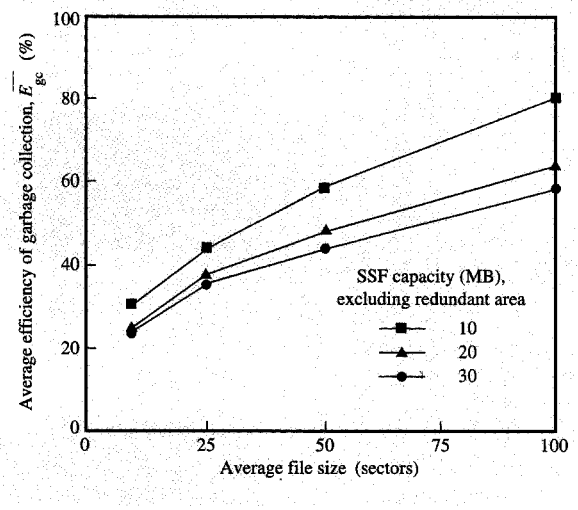

Figure 7

Effect of SSF capacity with fixed redundancy. This demonstrates the improvement in the average efficiency of garbage collection as the SSF file size decreases, when the redundant area is fixed. (1 MB of redundancy; cluster size = 64 sectors; average usage = 95%.)

of $\overline{E_{gc}}$. This is reasonable, since at least half of the user space can be used to hold garbage, just like a redundant area. Consider the results for a 1MB redundant area. When $\bar{U}$ is over 90%, the garbage collection mechanism hardly ever finds a cluster whose fraction of garbage sectors is over 50%. When $\bar{U}$ = 80%, $\overline{E_{gc}}$ is around 70%. This implies that if $\overline{E_{gc}}$ = 70% is acceptable, around 8 MB out of the 10MB SSF (excluding the 1MB redundant area) can be used on average, even when writing and erasing small files ($\overline{FS}$ = 25 sectors).

Figure 6 shows that when the cluster size decreases, the value of $\overline{E_{gc}}$ increases. This is obvious, since the same number of garbage sectors is a larger fraction of a smaller cluster. The selection of the cluster size is a trade-off between the efficiency of garbage collection and the system area overhead. As we mentioned in a previous section, there is a system sector in every cluster. The smaller the cluster, the larger the fraction of space occupied by the system sectors. In addition, smaller cluster size results in a larger number of clusters on the SSF, which requires more time for sorting their sequence numbers during power-up. We observed that 64 sectors per cluster seems to be a good value for an SSF with a few tens of megabytes.

The next question concerns the significance of a fixed amount of redundancy for various SSF capacities. **Figure 7** shows the contributions of 1 MB of redundancy to SSFs with 10 MB, 20 MB, and 30 MB of total storage (excluding the redundant area). The smaller the SSF capacity, the higher the value of $\overline{E_{gc}}$. This implies that the *work area ratio*, defined as the number of free sectors (blank and invalid) divided by the total number of data sectors, affects the value of $\overline{E_{gc}}$. We note that

$$\text{Work area ratio} \equiv \frac{\text{number of free sectors}}{\text{total number of data sectors in the SSF}} = \frac{(1-\bar{U})N_{user} + N_{redun}}{N_{user} + N_{redun}},$$

where N_{user} = number of data sectors in user space and N_{redun} = number of redundant data sectors.

Figure 8 shows $\overline{E_{gc}}$ as a function of the work area ratio. The points in the figure were acquired by varying both the redundant area and the average usage for fixed values of the average file size. For example, the points plotted by solid triangles represent *all* the points plotted in Figure 5 (average file size = 25 sectors). Although the points were obtained by varying both the redundant area and the average usage, the group of points corresponding to the same average file size forms a smooth curve. Thus, the results show that the parameters of redundant area and average usage have similar effects on the value of $\overline{E_{gc}}$. Figure 8 illustrates the case of a 10MB SSF; the data obtained for 20MB and 30MB SSFs are almost exactly the

same. These results indicate that the value of the work area ratio is the only significant parameter for the SSF with regard to area usage (the others are cluster size and SSF capacity). From Figure 8, therefore, we can estimate the work area ratio necessary to provide a specified $\overline{E_{gc}}$ value. For example, if the average file size = 25 sectors, in order to keep the value of $\overline{E_{gc}}$ over 70%, about 30% of all data sectors in the SSF must be free sectors. As mentioned above, the parameters of redundant area and average usage are considered equivalent; how to maintain a work area ratio of 30% is an SSF design option. One way is to provide redundancy; for example, an SSF with a total capacity of 10 MB might be designated as a 7MB SSF, so that 3 MB can be reserved and concealed from the user. With this approach, the value of $\overline{E_{gc}}$ can be kept over 70%, even when the average usage is nearly 100%; however, this approach has the drawback that a user can never store more than 7 MB of data, even temporarily. Another approach is to force the user to restrict the usage. In this case, a 10MB SSF might be designated as a 9MB SSF, and the user advised to keep data under 7 MB, on the average. This approach is risky, since a user can store data up to the point of causing a degradation of $\overline{E_{gc}}$ to less than 30%; however, by keeping the average usage around 7 MB, the user is awarded an extra 2 MB for temporary storage.

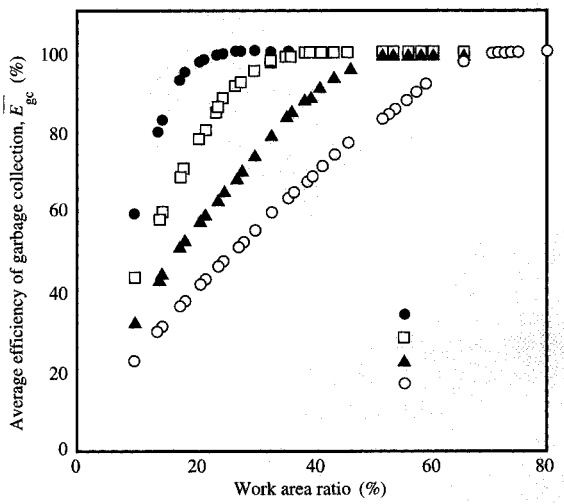

Figure 8

Effect of work area ratio. This demonstrates the improvement in the average efficiency of garbage collection as the work area ratio increases and average file size increases. (10MB SSF, with no redundant area; cluster size = 64 sectors.) Results obtained for 20MB and 30MB SSFs are almost exactly the same.

SSF lifetime estimation

The dynamic sector allocation and garbage collection mechanisms even out the erase counts of the sectors; thus, the SSF fails when almost all of its sectors have been written a number of times equal to their write/erase limits. In theory, we can write a number of sectors equal to

Endurance of a sector

× number of data sectors in the SSF

before the SSF is exhausted. In practice, we cannot write that many sectors, since the write count also increases as a result of the copy operation during garbage collection. If a cluster consists of N_c data sectors, an average of N_c $(1 - \overline{E_{gc}})$ sectors are copied by the garbage collection mechanism to a new cluster, so only $N_c\ \overline{E_{gc}}$ sectors are available as user data sectors. Taking this into consideration, we see that the maximum number of sectors that the user can write on the SSF during its lifetime is

$$S_{lifetime} = \text{endurance of a sector} \times (N_{user} + N_{redun}) \times \overline{E_{gc}}\ .$$

If garbage collection is hidden in the background, the minimum lifetime of the SSF can be calculated by determining how long it would take to write $S_{lifetime}$ sectors at a speed corresponding to the minimum write time of flash EEPROM (in other words, to write data at maximum speed). In practice, however, this notion of minimum lifetime does not make much sense, since it assumes that a user writes data onto the SSF all the time. To calculate a practical lifetime, we assume that a user writes 100 MB per day, which is quite a large number for personal use. For our estimation, we make the following additional assumptions: The total capacity of the SSF, which includes redundant area, is 10 MB; the endurance of the flash EEPROM is 2.5×10^5 write/erase cycles per block (the value given by Toshiba for the device used in our product); 100 MB (2×10^5 sectors) a day are written; and the average efficiency of garbage collection ($\overline{E_{gc}}$) is 70%.

According to these assumptions, $N_{user} + N_{redun}$ = 10 MB/512 bytes = 2×10^4 sectors, so

$$S_{lifetime} = 2.5 \times 10^5 \times 2 \times 10^4 \times 0.7 = 3.5 \times 10^9 \text{ sectors.}$$

Writing on $S_{lifetime}$ sectors will require

3.5×10^9 sectors/2×10^5 sectors per day = 17500 days

> 47 years.

As mentioned in a previous section, the efficiency of garbage collection is highly dependent on the usage of the SSF. Even if the efficiency were downgraded to 50%, the lifetime would still be about 34 years—quite sufficient for most users. (One possible mode of operation when the limit is reached is for the SSF to become write-protected

and return to a status of "write fail" when an attempt is made to write data.)

Summary

We have discussed the mechanisms of dynamic sector allocation and background garbage collection. In addition, we have demonstrated, by simulation, that solid-state files using flash EEPROMs have acceptable lifetimes for personal applications. The garbage collection mechanism takes account of many restrictions of flash EEPROMs in managing the free space on the SSF. These mechanisms are fundamental and applicable to SSF implementations using either NOR or NAND flash EEPROMs; however, a special mechanism, called clustered sector allocation, is required when dynamic sector allocation is applied to an SSF using the NAND type of flash EEPROM. SSFs using these mechanisms possess the following characteristics:

- The write operation is fast because data are written onto blank sectors that are erased beforehand.
- The SSF does not need battery protection against power loss. This improves system reliability and data integrity.
- The erase operation is hidden from the host system. Consequently, the host need not be concerned with the device characteristics of the flash EEPROMs.
- About 30% of the total sectors of the SSF are needed for a work area, to keep the efficiency of garbage collection over 70%.
- The SSF lifetime is acceptable because a leveling algorithm in the garbage collection mechanism ensures that all the sectors in the SSF are written to the limit of their endurance.

We have developed SSFs jointly with Toshiba Corporation. The SSF products (PCMCIA Intelligent Flash Memory) use 16Mb NAND flash EEPROMs and conform to JEIDA Ver.4.1 and PCMCIA Ver.2.1 guidelines. The SSFs utilize all the mechanisms presented in the paper.

Acknowledgment

Many people helped to make this paper possible. I am sincerely grateful to Yoshinori Sakaue and Takashi Toyooka for discussions of SSF architecture and control algorithms. Shigenori Shimizu, Nobuaki Takahashi, and Akashi Satoh provided helpful comments on a draft of this paper. Hiroyuki Akatsu helped me understand the device physics of flash EEPROMs.

References

1. B. Dipert and L. Hebert, "Flash Memory Goes Mainstream," *IEEE Spectrum* **30,** No. 10, 48–52 (October 1993).
2. F. Masuoka, "Technology Trend of Flash-EEPROM—Can Flash-EEPROM Overcome DRAM?," *Digest of Technical Papers*, 1992 Symposium on VLSI Technology, Seattle, June 2–4, 1992, pp. 6–9.
3. J. Eldridge, "Filing in a Flash," *IEEE Spectrum* **30,** No. 10, 53–54 (October 1993).
4. *Samsung NAND Flash, Revision 2.0*, Samsung Electronics Japan, Tokyo, 1994/1995.
5. M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Trans. Computer Syst.* **10,** No. 1, 26–52 (February 1992).
6. T. J. Robinson and P. A. Franaszek, "Analysis of Reorganization Overhead in Log-Structured File Systems," *Research Report RC-19056(83173)*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1993.
7. M. Burrows, C. Jerian, B. Lampson, and T. Mann, "On-Line Data Compression in a Log-Structured File System," *Proceedings of ASPLOS V* (Architectural Support for Programming Language and Operating System), Boston, October 12–15, 1992, pp. 2–9.
8. M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, "Non-Volatile Memory for Fast, Reliable File Systems," *Proceedings of ASPLOS V*, Boston, October 12–15, 1992, pp. 10–22.
9. C. Staelin, "The Coconut File System: Utilizing Tape Based Robotic Storage," *Proceedings of the USENIX File System Workshop*, Ann Arbor, MI, May 21–22, 1992, pp. 143–144.
10. C. U. Orji and J. A. Solworth, "The Delta File System (Delta FS)," *Proceedings of the USENIX File System Workshop*, Ann Arbor, MI, May 21–22, 1992, pp. 143–144.
11. *Intel Flash Memory System Developer's Kit, Hardware Reference Manual*, Order Number 296871-001, Intel Corporation, Santa Clara, CA, December 1990.
12. *PC Card ATA Specification, Release 1.01*, Personal Computer Memory Card International Association (PCMCIA), Sunnyvale, CA, November 1992.
13. S. Mehrotra, J. H. Yuan, R. A. Cernea, W. Y. Chien, D. C. Guterman, G. Samachisa, R. D. Norman, M. Mofidi, W. Lee, Y. Fong, A. Mihnea, E. Harari, R. W. Gregor, E. P. Eberhardt, J. R. Radosevich, K. R. Stiles, R. A. Kohler, C. W. Leung, and T. J. Mulrooney, "Serial 9 Mb Flash EEPROM for Solid State Disk Applications," *Digest of Technical Papers*, 1992 Symposium on VLSI Circuits, Seattle, June 4–6, 1992, pp. 24–25.
14. D. J. Lee, R. A. Cernea, M. Mofidi, S. Mehrotra, E. Y. Chang, W. Y. Chien, L. Goh, J. H. Yuan, A. Mihnea, G. Samachisa, Y. Fong, D. C. Guterman, R. D. Norman, K. Sato, H. Onishi, K. Ueda, F. Noro, K. Miyamoto, M. Morita, K. Umeda, and K. Kubo, "An 18 Mb Serial Flash EEPROM for Solid State Disk Applications," *Digest of Technical Papers*, 1994 Symposium on VLSI Circuits. Honolulu, June 9–11, 1994, pp. 59–60.
15. S. Aritome, R. Shirota, G. Hemink, T. Endoh, and F. Masuoka, "Reliability Issues of Memory Cells," *Proc. IEEE* **81,** No. 5, 776–788 (May 1993).

Received August 9, 1994; accepted for publication May 19, 1995

Hideto Niijima *IBM Tokyo Research Laboratory, 1623-14, Shimo-tsuruma, Yamato-shi, Kanagawa 242, Japan (NIJIMA at TRLVM, nijima@trl.vnet.ibm.com).* Mr. Niijima received the B.S. and M.S. degrees in electrical engineering from Waseda University, Tokyo, Japan. He joined the IBM Tokyo Research Laboratory in 1986 and has been involved in the research and development of MOS VLSI technology and I/O subsystems. His research interests include high-function memory devices, logical ASIC design, and storage subsystem design.