

Harari et al.

[45] **Date of Patent:** **Aug. 10, 1999**

- | | | | | |
|------|---|-----------|---------|------------------|
| [54] | MULTI-STATE FLASH EEPROM SYSTEM
WITH CACHE MEMORY | 58-215794 | 12/1983 | Japan . |
| | | 58-215795 | 12/1983 | Japan . |
| | | 59-45695 | 3/1984 | Japan . |
| [75] | Inventors: Eliyahou Harari , Los Gatos; Robert
D. Norman , San Jose; Sanjay
Mehrotra , Milpitas, all of Calif. | 59-162695 | 9/1984 | Japan . |
| | | 60-076097 | 4/1985 | Japan . |
| | | 60-212900 | 10/1985 | Japan . |
| | | A60178564 | 2/1986 | Japan . |
| [73] | Assignee: SanDisk Corporation , Sunnyvale,
Calif. | 61-96598 | 5/1986 | Japan . |
| | | 62-283496 | 12/1987 | Japan . |
| | | 62-283497 | 12/1987 | Japan . |
| | | 63-183700 | 7/1988 | Japan . |
| [21] | Appl. No.: 08/931,133 | A01054543 | 3/1989 | Japan . |
| | | 2136692 | 9/1984 | United Kingdom . |
| [22] | Filed: Sep. 16, 1997 | WO8400628 | 2/1984 | WIPO . |

Related U.S. Application Data

- [63] Continuation of application No. 08/249,049, May 25, 1994, Pat. No. 5,671,229, which is a continuation of application No. 07/963,837, Oct. 20, 1992, abandoned, which is a division of application No. 07/337,566, Apr. 13, 1989, abandoned.
- [51] **Int. Cl.**⁶ **G06F 11/00**
- [52] **U.S. Cl.** **371/10.2; 711/136; 365/200**
- [58] **Field of Search** **371/10.2; 365/200; 365/201; 395/182.04; 711/136**

References Cited

- [56]

U.S. PATENT DOCUMENTS

3,633,175	1/1972	Harper .
4,051,354	9/1977	Choate .
4,093,985	6/1978	Das .
4,210,959	7/1980	Wozniak .
4,250,570	2/1981	Tsang et al. .
4,272,830	6/1981	Moench .
4,279,024	7/1981	Schrenk .
4,281,398	7/1981	McKenney et al. .
4,287,570	9/1981	Stark .
4,295,205	10/1981	Kunstadt .
4,354,253	10/1982	Naden .

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

0557723	1/1987	Australia .
0086886	8/1983	European Pat. Off. .
0243503B1	7/1987	European Pat. Off. .
0300264B1	10/1989	European Pat. Off. .

OTHER PUBLICATIONS

- Lucero et al., "A 16 kbit Smart 5 V-only EEPROM with Redundancy," *IEEE Journal of Solid-State Circuits*, vol. SC-18, No. 5, pp. 539-543 (Oct. 1983).
- Torelli et al., "An improved method for programming a word-erasable EEPROM," *Alta Frequenza*, vol. 52, No. 6, pp. 487-494 (Nov.-Dec. 1983).

(List continued on next page.)

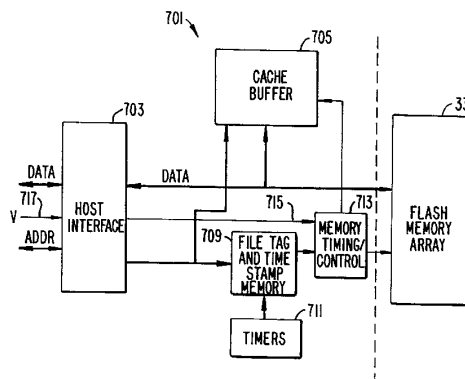
Primary Examiner—Phung M. Chung
Attorney, Agent, or Firm—Majestic, Parsons, Siebert & Hsue

ABSTRACT

- [57]

A system of Flash EEPROM memory chips with controlling circuits serves as non-volatile memory such as that provided by magnetic disk drives. Improvements include selective multiple sector erase, in which any combinations of Flash sectors may be erased together. Selective sectors among the selected combination may also be de-selected during the erase operation. Another improvement is the ability to remap and replace defective cells with substitute cells. The remapping is performed automatically as soon as a defective cell is detected. When the number of defects in a Flash sector becomes large, the whole sector is remapped. Yet another improvement is the use of a write cache to reduce the number of writes to the Flash EEPROM memory, thereby minimizing the stress to the device from undergoing too many write/erase cycling.

26 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

4,355,376 10/1982 Gould .
 4,380,066 4/1983 Spencer et al. .
 4,405,952 9/1983 Slakmon .
 4,422,161 12/1983 Kressel et al. .
 4,449,205 5/1984 Hoffman .
 4,450,559 5/1984 Bond et al. .
 4,456,971 6/1984 Fukuda et al. .
 4,463,450 7/1984 Haeusele .
 4,466,059 8/1984 Bastian et al. .
 4,479,214 10/1984 Ryan .
 4,493,075 1/1985 Anderson et al. .
 4,498,146 2/1985 Martinez .
 4,514,830 4/1985 Hagiwara et al. .
 4,527,251 7/1985 Nibby, Jr. et al. .
 4,530,055 7/1985 Hamstra et al. 711/136
 4,586,163 4/1986 Koike .
 4,601,031 7/1986 Walker et al. .
 4,612,640 9/1986 Mehrotra et al. .
 4,616,311 10/1986 Sato .
 4,617,624 10/1986 Goodman .
 4,617,651 10/1986 Ip et al. .
 4,642,759 2/1987 Foster .
 4,653,023 3/1987 Suzuki et al. .
 4,654,847 3/1987 Dutton .
 4,672,240 6/1987 Smith et al. .
 4,718,041 1/1988 Baglee et al. .
 4,733,394 3/1988 Burkhard .
 4,746,998 5/1988 Robinson et al. .
 4,785,425 11/1988 Lavelle .
 4,794,568 12/1988 Lim et al. .
 4,796,233 1/1989 Awaya et al. .
 4,800,520 1/1989 Iijima .
 4,805,109 2/1989 Kroll et al. .
 4,821,240 4/1989 Nakamura et al. .
 4,847,808 7/1989 Kobatake .
 4,882,642 11/1989 Tayler et al. .
 4,887,234 12/1989 Iijima .
 4,896,262 1/1990 Wayama et al. .
 4,914,529 4/1990 Bonke .
 4,916,605 4/1990 Beardsley et al. .
 4,920,478 4/1990 Furuya et al. .
 4,920,518 4/1990 Nakamura et al. .
 4,924,331 5/1990 Robinson et al. .
 4,933,906 6/1990 Terada et al. .
 4,942,556 7/1990 Sasaki et al. .
 4,945,535 7/1990 Hosotani et al. .
 4,949,240 8/1990 Iijima .
 4,953,122 8/1990 Williams .
 5,043,940 8/1991 Harari .
 5,051,887 9/1991 Berger et al. .
 5,053,990 10/1991 Kreifels et al. .
 5,070,474 12/1991 Tuma et al. .
 5,095,344 3/1992 Harari .
 5,136,546 8/1992 Fukuda et al. .
 5,163,021 11/1992 Mehrotra et al. .
 5,172,338 12/1992 Mehrotra et al. .

5,226,168 7/1993 Kobayashi et al. .
 5,297,148 3/1994 Harari et al. .
 5,359,569 10/1994 Fujita et al. .
 5,430,859 7/1995 Norman et al. .
 5,488,711 1/1996 Hewitt et al. .
 5,546,351 8/1996 Tanaka et al. .
 5,606,532 2/1997 Lambrache et al. .
 5,671,229 9/1997 Harari et al. .

OTHER PUBLICATIONS

Data Sheet: "27F256 256K(32K×8) CMOS Flash Memory," *Intel Corporation*, pp. 1–24 (May 1988).
 Preliminary Data Sheet, "48F512 512K Flash EEPROM," *SEEQ Technology, Incorporated*, pp. 2–1 through 2–12 (Oct. 1988).
 Advanced Data Sheet, "48F010 1024K Flash EEPROM," *SEEQ Technology, Incorporated*, pp. 2–13 thru 2–24 (Oct. 1988).
 Lai, Robert S., *Writing MS-DOS Device Drivers, The Waite Group, Inc.*, (Sep. 1987), pp. i–xi and 235–319.
 Miller, "Semidisk Disk Emulator," *Interface Age*, p. 102, Nov. 1982.
 Clewitt, "Bubble Memories as a Floppy Disk Replacement," *1978 MIDCON Technical Papers*, vol. 2, pp. 1–7, Dec. 1978.
 Hancock, "Architecting a CCD Replacement for the IBM 2305 Fixed Head Disk Drive," Digest of Papers, *Eighteenth IEEE Computer Society International Conference*, pp. 182–184, 1979.
 Wilson, "1-Mbit flash memories seek their role in system design," *Computer Design*, vol. 28, No. 5, pp. 30–32 (Mar. 1989).
 Bleiker et al., "A Four-State EEPROM Using Floating-Gate Memory Cells," *IEEE Journal of Solid-State Circuits*, SC-22 (1987) Jun., No. 3, New York, NY, USA.
 Intel Corporation, 27F256, 256K (32K×8) CMOS Flash Memory, May 1988, pp. 1–21.
 Krick, "Three-State MNOS FET Memory Array," *IBM Technical Disclosure Bulletin*, vol. 18, No. 12, May 1976, pp. 4192–4193.
 Alberts C.S. et al., "Multi-Bit Storage FET EAROM Cell," *IBM Technical Disclosure Bulletin*, vol. 24, No. 74, Dec. 1981, pp. 3311–3314.
 Horiguchi et al., "An Experimental Large-Capacity Semiconductor File Memory Using 16-Levels Cell Storage," *IEEE Journal of Solid-State Circuits*, vol. 23, No. 1, Feb. 1988, pp. 27–33.
 Furuyama, et al., "An Experimental 2-Bit/Cell Storage Dram for Macro Cell or Memory-on-logic Application," *Jan. 1988 IEEE*, pp. 4.4.1 to 4.4.4.
 Stark, "Two Bits Per Cell ROM," *Jan. 1981 IEEE Catalog* No. 81-CH1626-1, pp. 201–212.

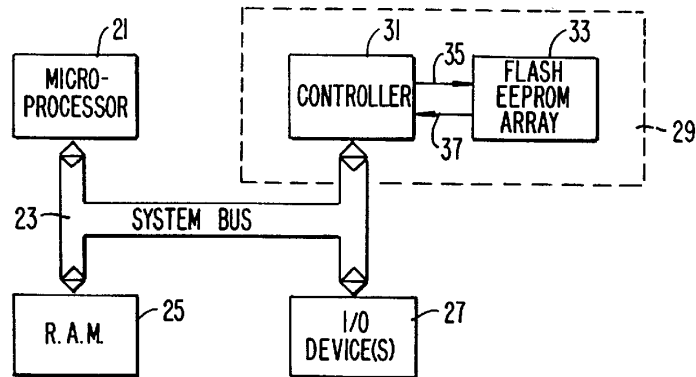


FIG. 1A.

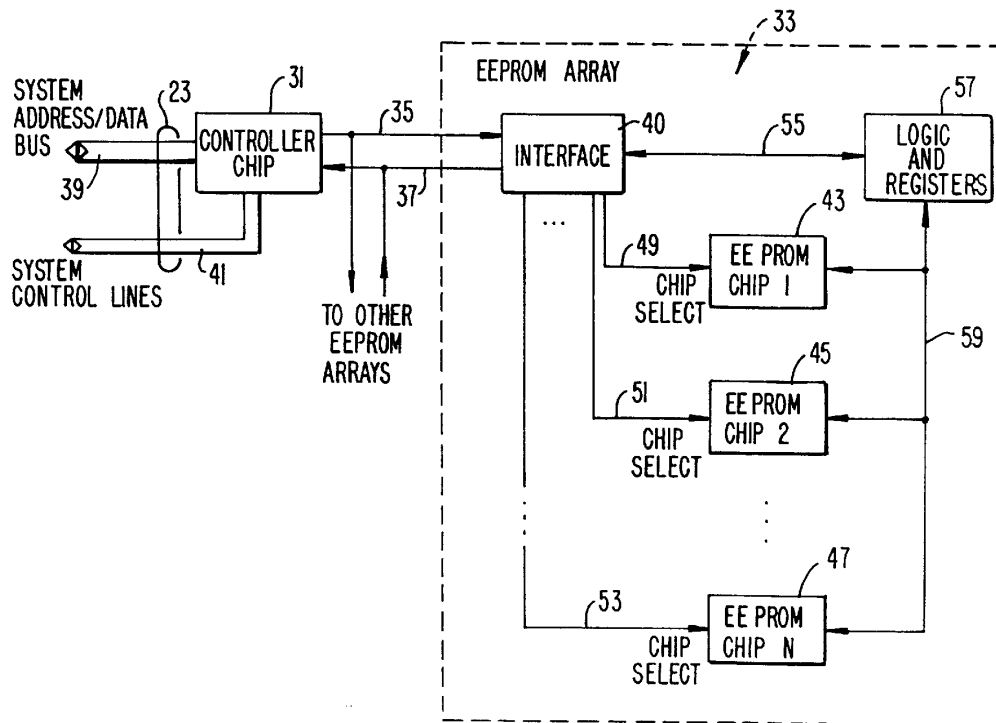


FIG. 1B.

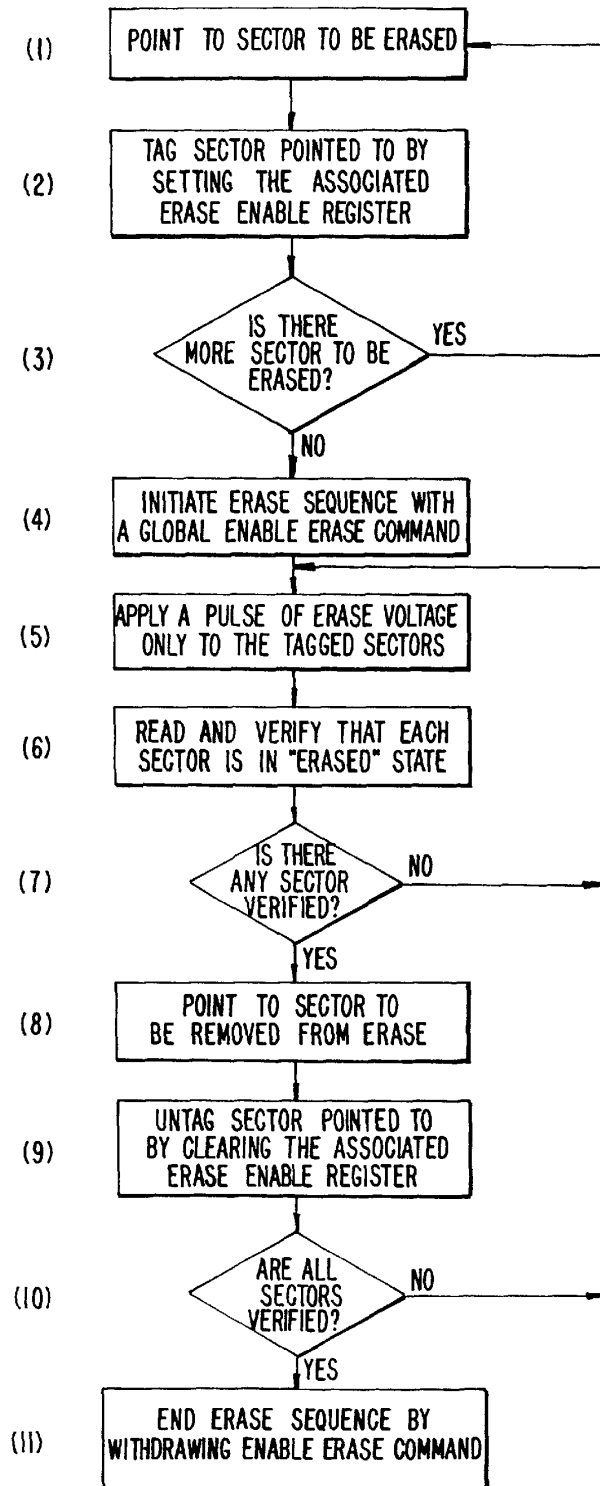
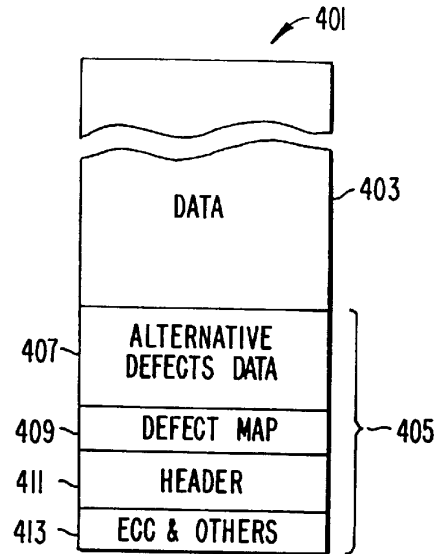


FIG. 4.



SECTOR PARTITION

FIG. 5.

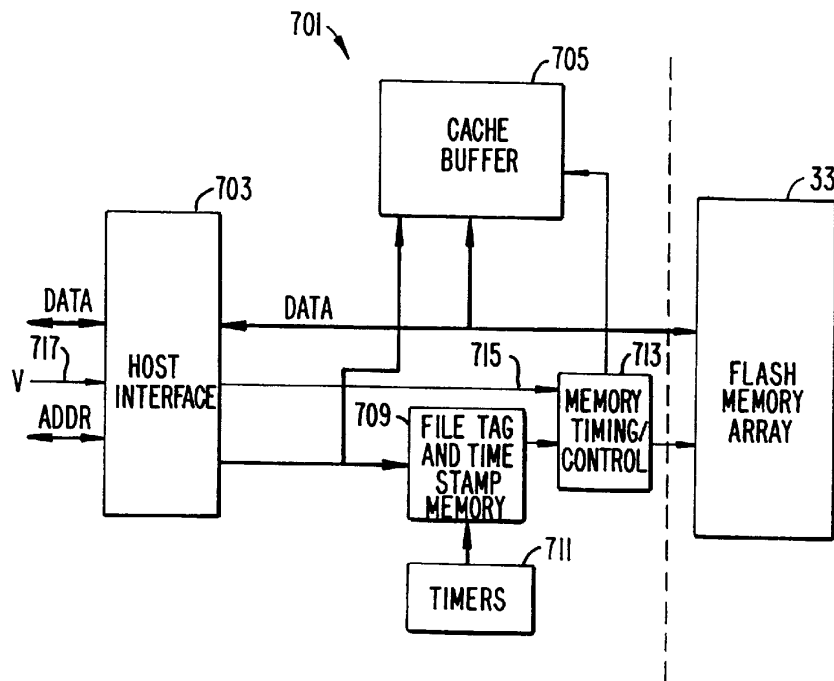
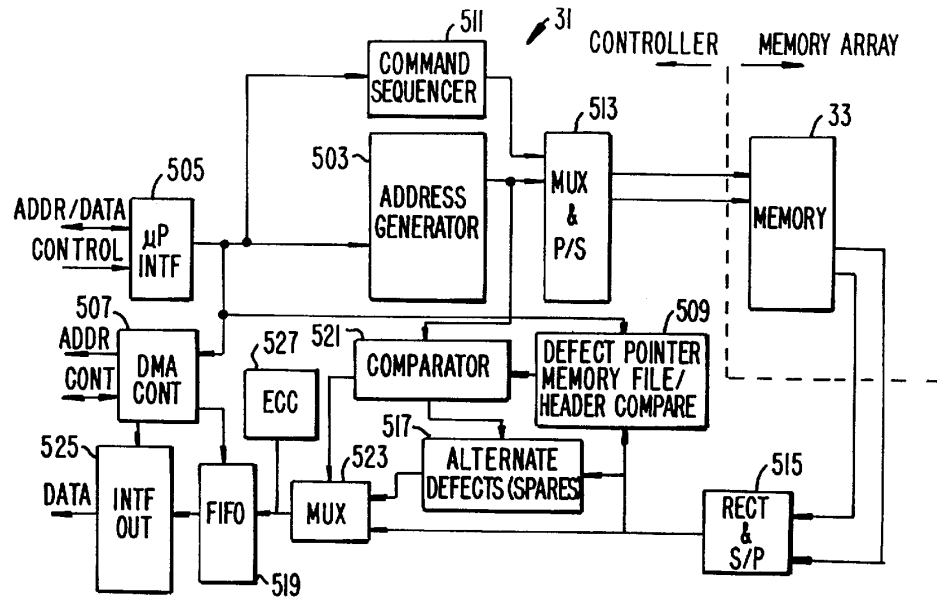
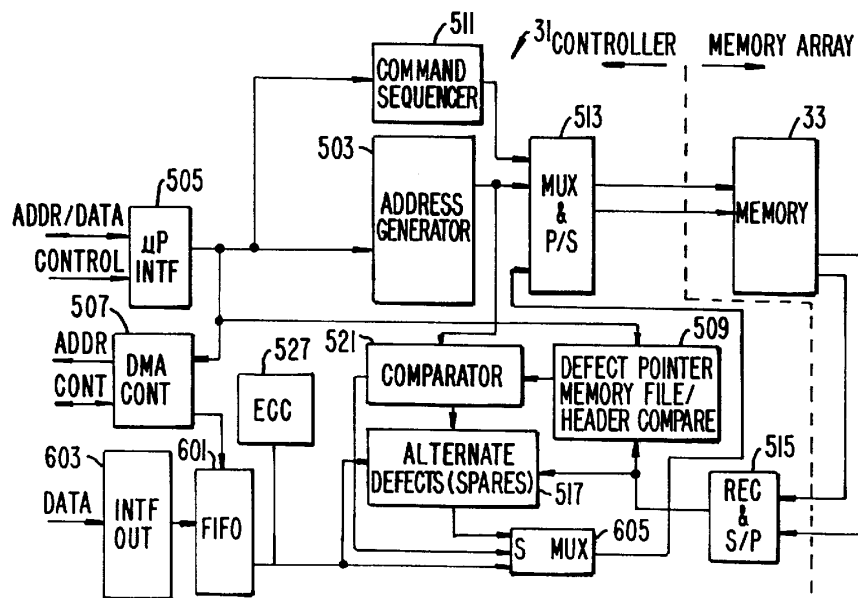


FIG. 8.



READ DATA PATH CONTROL

FIG. 6.



WRITE DATA PATH CONTROL

FIG. 7.

MULTI-STATE FLASH EEPROM SYSTEM WITH CACHE MEMORY

CROSS-REFERENCE TO RELATED APPLICATIONS

This is a continuation of application Ser. No. 08/249,049, filed May 25, 1994, now U.S. Pat. No. 5,671,229, which is a continuation of application Ser. No. 07/963,837, filed Oct. 20, 1992, now abandoned, which in turn is a division of application Ser. No. 07/337,566, filed Apr. 13, 1989, now abandoned.

BACKGROUND OF THE INVENTION

This invention relates generally to semiconductor electrically erasable programmable read only memories (EEPROM), and specifically to a system of integrated circuit Flash EEPROM chips.

Computer systems typically use magnetic disk drives for mass storage of data. However, disk drives are disadvantageous in that they are bulky and in their requirement for high precision moving mechanical parts. Consequently they are not rugged and are prone to reliability problems, as well as consuming significant amounts of power. Solid state memory devices such as DRAM's and SRAM's do not suffer from these disadvantages. However, they are much more expensive, and require constant power to maintain their memory (volatile). Consequently, they are typically used as temporary storage.

EEPROM's and Flash EEPROM's are also solid state memory devices. Moreover, they are nonvolatile, and retain their memory even after power is shut down. However, conventional Flash EEPROM's have a limited lifetime in terms of the number of write (or program)/erase cycles they can endure. Typically the devices are rendered unreliable after 10^2 to 10^3 write/erase cycles. Traditionally, they are typically used in applications where semi-permanent storage of data or program is required but with a limited need for reprogramming.

Accordingly, it is an object of the present invention to provide a Flash EEPROM memory system with enhanced performance and which remains reliable after enduring a large number of write/erase cycles.

It is another object of the present invention to provide an improved Flash EEPROM system which can serve as non-volatile memory in a computer system.

It is another object of the present invention to provide an improved Flash EEPROM system that can replace magnetic disk storage devices in computer systems.

It is another object of the present invention to provide a Flash EEPROM system with improved erase operation.

It is another object of the present invention to provide a Flash EEPROM system with improved error correction.

It is yet another object of the present invention to provide a Flash EEPROM with improved write operation that minimizes stress to the Flash EEPROM device.

It is still another object of the present invention to provide a Flash EEPROM system with enhanced write operation.

SUMMARY OF THE INVENTION

These and additional objects are accomplished by improvements in the architecture of a system of EEPROM chips, and the circuits and techniques therein.

According to one aspect of the present invention, an array of Flash EEPROM cells on a chip is organized into sectors

such that all cells within each sector are erasable at once. A Flash EEPROM memory system comprises one or more Flash EEPROM chips under the control of a controller. The invention allows any combination of sectors among the chips to be selected and then erased simultaneously. This is faster and more efficient than prior art schemes where all the sectors must be erased every time or only one sector at a time can be erased. The invention further allows any combination of sectors selected for erase to be deselected and prevented from further erasing during the erase operation. This feature is important for stopping those sectors that are first to be erased correctly to the "erased" state from over erasing, thereby preventing unnecessary stress to the Flash EEPROM device. The invention also allows a global de-select of all sectors in the system so that no sectors are selected for erase. This global reset can quickly put the system back to its initial state ready for selecting the next combination of sectors for erase. Another feature of the invention is that the selection is independent of the chip select signal which enables a particular chip for read or write operation. Therefore it is possible to perform an erase operation on some of the Flash EEPROM chips while read and write operations may be performed on other chips not involved in the erase operation.

According to another aspect of the invention, improved error correction circuits and techniques are used to correct for errors arising from defective Flash EEPROM memory cells. One feature of the invention allows defect mapping at cell level in which a defective cell is replaced by a substitute cell from the same sector. The defect pointer which connects the address of the defective cell to that of the substitute cell is stored in a defect map. Every time the defective cell is accessed, its bad data is replaced by the good data from the substitute cell.

Another feature of the invention allows defect mapping at the sector level. When the number of defective cells in a sector exceeds a predetermined number, the sector containing the defective cells is replaced by a substitute sector.

An important feature of the invention allows defective cells or defective sectors to be remapped as soon as they are detected thereby enabling error correction codes to adequately rectify the relatively few errors that may crop up in the system.

According to yet another aspect of the present invention, a write cache is used to minimize the number of writes to the Flash EEPROM memory. In this way the Flash EEPROM memory will be subject to fewer stress inducing write/erase cycles, thereby retarding its aging. The most active data files are written to the cache memory instead of the Flash EEPROM memory. Only when the activity levels have reduced to a predetermined level are the data files written from the cache memory to the Flash EEPROM memory. Another advantage of the invention is the increase in write throughput by virtue of the faster cache memory.

According to yet another aspect of the present invention, one or more printed circuit cards are provided which contain controller and EEPROM circuit chips for use in a computer system memory for long term, non-volatile storage, in place of a hard disk system, and which incorporate various of the other aspects of this invention alone and in combination.

Additional objects, features, and advantages of the present invention will be understood from the following description of its preferred embodiments, which description should be taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a general microprocessor system including the Flash EEPROM memory system of the present invention;

3

FIG. 1B is schematic block diagram illustrating a system including a number of Flash EEPROM memory chips and a controller chip;

FIG. 2 is a schematic illustration of a system of Flash EEPROM chips, among which memory sectors are selected to be erased;

FIG. 3A is a block circuit diagram in the controller for implementing selective multiple sector erase according to the preferred embodiment;

FIG. 3B shows details of a typical register used to select a sector for erase as shown in FIG. 2A;

FIG. 4 is a flow diagram illustrating the erase sequence of selective multiple sector erase;

FIG. 5 is a schematic illustration showing the partitioning of a Flash EEPROM sector into a data area and a spare redundant area;

FIG. 6 is a circuit block diagram illustrating the data path control during read operation using the defect mapping scheme of the preferred embodiment;

FIG. 7 is a circuit block diagram illustrating the data path control during the write operation using the defect mapping scheme of the preferred embodiment;

FIG. 8 is a block diagram illustrating the write cache circuit inside the controller.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

EEProm System

A computer system in which the various aspects of the present invention are incorporated is illustrated generally in FIG. 1A. A typical computer system architecture includes a microprocessor 21 connected to a system bus 23, along with random access, main system memory 25, and at least one or more input-output devices 27, such as a keyboard, monitor, modem, and the like. Another main computer system component that is connected to a typical computer system bus 23 is a large amount of long-term, non-volatile memory 29. Typically, such a memory is a disk drive with a capacity of tens of megabytes of data storage. This data is retrieved into the system volatile memory 25 for use in current processing, and can be easily supplemented, changed or altered.

One aspect of the present invention is the substitution of a specific type of semiconductor memory system for the disk drive but without having to sacrifice non-volatility, ease of erasing and rewriting data into the memory, speed of access, low cost and reliability. This is accomplished by employing an array of electrically erasable programmable read only memories (EEPROM's) integrated circuit chips. This type of memory has additional advantages of requiring less power to operate, and of being lighter in weight than a hard disk drive magnetic media memory, thereby being especially suited for battery operated portable computers.

The bulk storage memory 29 is constructed of a memory controller 31, connected to the computer system bus 23, and an array 33 of EEPROM integrated circuit chips. Data and instructions are communicated from the controller 31 to the EEPROM array 33 primarily over a serial data line 35. Similarly, data and status signals are communicated from the EEPROM 33 to the controller 31 over serial data lines 37. Other control and status circuits between the controller 31 and the EEPROM array 33 are not shown in FIG. 1A.

Referring to FIG. 1B, the controller 31 is preferably formed primarily on a single integrated circuit chip. It is connected to the system address and data bus 39, part of the system bus 33, as well as being connected to system control lines 41, which include interrupt, read, write and other usual computer system control lines.

4

The EEPROM array 33 includes a number of EEPROM integrated circuit chips 43, 45, 47, etc. Each includes a respective chip select and enable line 49, 51 and 53 from interface circuits 40. The interface circuits 40 also act to interface between the serial data lines 35, 37 and a circuit 55. Memory location addresses and data being written into or read from the EEPROM chips 43, 45, 47, etc. are communicated from a bus 55, through logic and register circuits 57 and thence by another bus 59 to each of the memory chips 43, 45, 47 etc.

The bulk storage memory 29 of FIGS. 1A and 1B can be implemented on a single printed circuit card for moderate memory sizes. The various lines of the system buses 39 and 41 of FIG. 1B are terminated in connecting pins of such a card for connection with the rest of the computer system through a connector. Also connected to the card and its components are various standard power supply voltages (not shown).

For large amounts of memory, that which is conveniently provided by a single array 33 may not be enough. In such a case, additional EEPROM arrays can be connected to the serial data lines 35 and 37 of the controller chip 31, as indicated in FIG. 1B. This is preferably all done on a single printed circuit card but if space is not sufficient to do this, then one or more EEPROM arrays may be implemented on a second printed circuit card that is physically mounted onto the first and connected to a common controller chip 31.

Erase of Memory Structures

In system designs that store data in files or blocks the data will need to be periodically updated with revised or new information. It may also be desirable to overwrite some no longer needed information, in order to accommodate additional information. In a Flash EEPROM memory, the memory cells must first be erased before information is placed in them. That is, a write (or program) operation is always preceded by an erase operation.

In conventional Flash erase memory devices, the erase operation is done in one of several ways. For example, in some devices such as the Intel corporation's model 27F-256 CMOS Flash EEPROM, the entire chip is erased at one time. If not all the information in the chip is to be erased, the information must first be temporarily saved, and is usually written into another memory (typically RAM). The information is then restored into the nonvolatile Flash erase memory by programming back into the device. This is very slow and requires extra memory as holding space.

In other devices such as Seeq Technology Incorporated's model 48512 Flash EEPROM chip, the memory is divided into blocks (or sectors) that are each separately erasable, but only one at a time. By selecting the desired sector and going through the erase sequence the designated area is erased. While, the need for temporary memory is reduced, erase in various areas of the memory still requires a time consuming sequential approach.

In the present invention, the Flash EEPROM memory is divided into sectors where all cells within each sector are erasable together. Each sector can be addressed separately and selected for erase. One important feature is the ability to select any combination of sectors for erase together. This will allow for a much faster system erase than by doing each one independently as in prior art.

FIG. 2 illustrates schematically selected multiple sectors for erase. A Flash EEPROM system includes one or more Flash EEPROM chips such as 201, 203, 205. They are in communication with a controller 31 through lines 209. Typically, the controller 31 is itself in communication with a microprocessor system (not shown). The memory in each

Flash EEPROM chip is partitioned into sectors where all memory cells within a sector are erasable together. For example, each sector may have 512 byte (i.e. 512x8 cells) available to the user, and a chip may have 1024 sectors. Each sector is individually addressable, and may be selected, such as sectors **211**, **213**, **215**, **217** in a multiple sector erase. As illustrated in FIG. 2, the selected sectors may be confined to one EEPROM chip or be distributed among several chips in a system. The sectors that were selected will all be erased together. This capability will allow the memory and system of the present invention to operate much faster than the prior art architectures.

FIG. 3A illustrates a block diagram circuit **220** on a Flash EEPROM chip (such as the chip **201** of FIG. 2) with which one or more sectors such as **211**, **213** are selected (or deselected) for erase. Essentially, each sector such as **211**, **213** is selected or tagged by setting the state of an erase enable register such as **221**, **223** associated with the respective sectors. The selection and subsequent erase operations are performed under the control of the controller **31** (see FIG. 2). The circuit **220** is in communication with the controller **31** through lines **209**. Command information from the controller is captured in the circuit **220** by a command register **225** through a serial interface **227**. It is then decoded by a command decoder **229** which outputs various control signals. Similarly, address information is captured by an address register **231** and is decoded by an address decoder **233**.

For example, in order to select the sector **211** for erase, the controller sends the address of the sector **211** to the circuit **220**. The address is decoded in line **235** and is used in combination with a set erase enable signal in bus **237** to set an output **239** of the register **221** to HIGH. This enables the sector **211** in a subsequent erase operation. Similarly, if the sector **213** is also desired to be erased, its associated register **223** may be set HIGH.

FIG. 3B shows the structure of the register such as **221**, **223** in more detail. The erase enable register **221** is a SET/RESET latch. Its set input **241** is obtained from the set erase enable signal in bus **237** gated by the address decode in line **235**. Similarly, the reset input **243** is obtained from the clear erase enable signal in bus **237** gated by the address decode in line **235**. In this way, when the set erase enable signal or the clear erase enable signal is issued to all the sectors, the signal is effective only on the sector that is being addressed.

After all sectors intended for erase have been selected, the controller then issues to the circuit **220**, as well as all other chips in the system a global erase command in line **251** along with the high voltage for erasing in line **209**. The device will then erase all the sectors that have been selected (i.e. the sectors **211** and **213**) at one time. In addition to erasing the desired sectors within a chip, the architecture of the present system permits selection of sectors across various chips for simultaneous erase.

FIGS. 4(1)–4(11) illustrate the algorithm used in conjunction with the circuit **220** of FIG. 3A. In FIG. 4(1), the controller will shift the address into the circuit **220** which is decoded in the line to the erase enable register associated with the sector that is to be erased. In FIG. 4(2), the controller shifts in a command that is decoded to a set erase enable command which is used to latch the address decode signal onto the erase enable register for the addressed sector. This tags the sector for subsequent erase. In FIG. 4(3), if more sectors are to be tagged, the operations described relative to FIGS. 4(1)–4(2) are repeated until all sectors intended for erase have been tagged. After all sectors

intended for erase have been tagged, the controller initiates an erase cycle as illustrated in FIG. 4(4).

Optimized erase implementations have been disclosed in two copending U.S. patent applications. They are copending U.S. patent applications, Ser. No. 204,175, filed Jun. 8, 1988, by Dr. Eliyahou Harari, now U.S. Pat. No. 5,095,344, and one entitled "Multi-State EEPROM Read and Write Circuits and Techniques," Ser. No. 07/337,579, filed Apr. 13, 1989, now abandoned, by Sanjay Mehrotra and Dr. Eliyahou Harari. The disclosures of the two applications are hereby incorporated by reference. The Flash EEPROM cells are erased by applying a pulse of erasing voltage followed by a read to verify if the cells are erased to the "erased" state. If not, further pulsing and verifying are repeated until the cells are verified to be erased. By erasing in this controlled manner, the cells are not subject to over-erasure which tends to age the EEPROM device prematurely as well as make the cells harder to program.

As the group of selected sectors is going through the erase cycle, some sectors will reach the "erase" state earlier than others. Another important feature of the present invention is the ability to remove those sectors that have been verified to be erased from the group of selected sectors, thereby preventing them from over-erasing.

Returning to FIG. 4(4), after all sectors intended for erase have been tagged, the controller initiates an erase cycle to erase the group of tagged sectors. In FIG. 4(5), the controller shifts in a global command called Enable Erase into each Flash EEPROM chip that is to perform an erase. This is followed in FIG. 4(5) by the controller raising of the erase voltage line (V_e) to a specified value for a specified duration. The controller will lower this voltage at the end of the erase duration time. In FIG. 4(6), the controller will then do a read verify sequence on the sectors selected for erase. In FIG. 4(7), if none of the sectors are verified, the sequences illustrated in FIGS. 4(5)–4(7) are repeated. In FIGS. 4(8) and 3(9), if one or more sectors are verified to be erased, they are taken out of the sequence. Referring also to FIG. 3A, this is achieved by having the controller address each of the verified sectors and clear the associated erase enable registers back to a LOW with a clear enable command in bus **237**. The sequences illustrated in FIGS. 4(5)–4(10) are repeated until all the sectors in the group are verified to be erased in FIG. 4(11). At the completion of the erase cycle, the controller will shift in a No Operation (NOP) command and the global Enable Erase command will be withdrawn as a protection against a false erasure.

The ability to select which sectors to erase and which ones not to, as well as which ones to stop erasing is advantageous. It will allow sectors that have erased before the slower erased sectors to be removed from the erase sequence so no further stress on the device will occur. This will increase the reliability of the system. Additional advantage is that if a sector is bad or is not used for some reason, that sector can be skipped over with no erase occurring within that sector. For example, if a sector is defective and have shorts in it, it may consume much power. A significant system advantage is gained by the present invention which allows it to be skipped on erase cycles so that it may greatly reduce the power required to erase the chip.

Another consideration in having the ability to pick the sectors to be erased within a device is the power savings to the system. The flexibility in erase configuration of the present invention enables the adaptation of the erase needs to the power capability of the system. This can be done by configuring the systems to be erased differently by software on a fixed basis between different systems. It also will allow

the controller to adaptively change the amount of erasing being done by monitoring the voltage level in a system, such as a laptop computer.

An additional performance capability of the system in the present invention is the ability to issue a reset command to a Flash EEprom chip which will clear all erase enable latches and will prevent any further erase cycles from occurring. This is illustrated in FIGS. 2A and 2B by the reset signal in the line 261. By doing this in a global way to all the chips, less time will be taken to reset all the erase enable registers.

An additional performance capability is to have the ability to do erase operations without regard to chip select. Once an erase is started in some of the memory chips, the controller in the system can access other memory chips and do read and write operations on them. In addition, the device(s) doing the erase can be selected and have an address loaded for the next command following the erase.

Defect Mapping

Physical defects in memory devices give rise to hard errors. Data becomes corrupted whenever it is stored in the defective cells. In conventional memory devices such as RAM's and Disks, any physical defects arising from the manufacturing process are corrected at the factory. In RAM's, spare redundant memory cells on chip may be patched on, in place of the defective cells. In the traditional disk drive, the medium is imperfect and susceptible to defects. To overcome this problem manufacturers have devised various methods of operating with these defects present, the most usual being defect mapping of sectors. In a normal disk system the media is divided into cylinders and sectors. The sector being the basic unit in which data is stored. When a system is partitioned into the various sectors the sectors containing the defects are identified and are marked as bad and not to be used by the system. This is done in several ways. A defect map table is stored on a particular portion of the disk to be used by the interfacing controller. In addition, the bad sectors are marked as bad by special ID and flag markers. When the defect is addressed, the data that would normally be stored there is placed in an alternative location. The requirement for alternative sectors makes the system assign spare sectors at some specific interval or location. This reduces the amount of memory capacity and is a performance issue in how the alternative sectors are located.

One important application of the present invention is to replace a conventional disk storage device with a system incorporating an array of Flash EEprom memory chips. The EEprom system is preferably set up to emulate a conventional disk, and may be regarded as a "solid-state disk".

In a "disk" system made from such solid-state memory devices, low cost considerations necessitate efficient handling of defects. Another important feature of the invention enables the error correction scheme to conserve as much memory as possible. Essentially, it calls for the defective cells to be remapped cell by cell rather than by throwing away the whole sector (512 bytes typically) whenever a defect occurs in it. This scheme is especially suited to the Flash EEprom medium since the majority of errors will be bit errors rather than a long stream of adjacent defects as is typical in traditional disk medium.

In both cases of the prior art RAM and magnetic disk, once the device is shipped from the factory, there is little or no provision for replacing hard errors resulting from physical defects that appear later during normal operation. Error corrections then mainly rely on schemes using error correction codes (ECC).

The nature of the Flash EEprom device predicates a higher rate of cell failure especially with increasing program/erase cycling. The hard errors that accumulate with use would eventually overwhelm the ECC and render the device unusable. One important feature of the present invention is the ability for the system to correct for hard errors whenever they occur. Defective cells are detected by their failure to program or erase correctly. Also during read operation, defective cells are detected and located by the ECC. As soon as a defective cell is identified, the controller will apply defect mapping to replace the defective cell with a spare cell located usually within the same sector. This dynamic correction of hard errors, in addition to conventional error correction schemes, significantly prolongs the life of the device.

Another feature of the present invention is an adaptive approach to error correction. Error correction code (ECC) is employed at all times to correct for soft errors as well as any hard errors that may arise. As soon as a hard error is detected, defect mapping is used to replace the defective cell with a spare cell in the same sector block. Only when the number of defective cells in a sector exceeds the defect mapping's capacity for that specific sector will the whole sector be replaced as in a conventional disk system. This scheme minimized wastage without compromising reliability.

FIG. 5 illustrates the memory architecture for the cell remapping scheme. As described before, the Flash EEprom memory is organized into sectors where the cells in each sector are erasable together. The memory architecture has a typical sector 401 organized into a data portion 403 and a spare (or shadow) portion 405. The data portion 403 is memory space available to the user. The spare portion 405 is further organized into an alternative defects data area 407, a defect map area 409, a header area 411 and an ECC and others area 413. These areas contain information that could be used by the controller to handle the defects and other overhead information such as headers and ECC.

Whenever a defective cell is detected in the sector, a good cell in the alternative defects data area 407 is assigned to backup the data designated for the defective cell. Thus even if the defective cell stores the data incorrectly, an error-free copy is stored in the backup cell. The addresses of the defective cell and the backup cell are stored as defect pointers in the defect map 409.

It is to be understood that the partitioning between the user data portion 403 and the spare portion 405 need not be rigid. The relative size of the various partitioned areas may be logically reassigned. Also the grouping of the various areas is largely for the purpose of discussion and not necessarily physically so. For example, the alternative defects data area 407 has been schematically grouped under the spare portion 405 to express the point that the space it occupies is no longer available to the user.

In a read operation, the controller first reads the header, the defect map and the alternative defects data. It then reads the actual data. It keeps track of defective cells and the location of the substitute data by means of the defect map. Whenever a defective cell is encountered, the controller substitutes its bad data with the good data from the alternative defects.

FIG. 6 illustrates the read data path control in the preferred embodiment. A memory device 33 which may include a plurality of Flash EEprom chips is under the control of the controller 31. The controller 31 is itself part of a microcomputer system under the control of a microprocessor (not shown). To initiate the reading of a sector, the microproces-

sor loads a memory address generator **503** in the controller with a memory address for starting the read operation. This information is loaded through a microprocessor interface port **505**. Then the microprocessor loads a DMA controller **507** with the starting location in buffer memory or bus address that the data read should be sent. Then the microprocessor loads the header information (Head, Cylinder and sector) into a holding register file **509**. Finally, the microprocessor loads a command sequencer **511** with a read command before passing control to the controller **31**.

After assuming control, the controller **31** first addresses the header of the sector and verifies that the memory is accessed at the address that the user had specified. This is achieved by the following sequence. The controller selects a memory chip (chip select) among the memory device **33** and shifts the address for the header area from the address generator **503** out to the selected memory chip in the memory device **33**. The controller then switches the multiplexer **513** and shifts also the read command out to the memory device **33**. Then the memory device reads the address sent it and begins sending serial data from the addressed sector back to the controller. A receiver **515** in the controller receives this data and puts it in parallel format. In one embodiment, once a byte (8 bits) is compiled, the controller compares the received data against the header data previously stored by the microprocessor in the holding register file **509**. If the compare is correct, the proper location is verified and the sequence continues.

Next the controller **31** reads the defect pointers and loads these bad address locations into the holding register file **509**. This is followed by the controller reading the alternative defects data that were written to replace the bad bits as they were written. The alternative bits are stored in an alternative defects data file **517** that will be accessed as the data bits are read.

Once the Header has been determined to be a match and the defect pointers and alternative bits have been loaded, the controller begins to shift out the address of the lowest address of the desired sector to be read. The data from the sector in the memory device **33** is then shifted into the controller chip **31**. The receiver **515** converts the data to a parallel format and transfers each byte into a temporary holding FIFO **519** to be shipped out of the controller.

A pipeline architecture is employed to provide efficient throughput as the data is gated through the controller from the receiver **515** to the FIFO **519**. As each data bit is received from memory the controller is comparing the address of the data being sent (stored in the address generator **507**) against the defect pointer map (stored in the register file **509**). If the address is determined to be a bad location, by a match at the output of the comparator **521**, the bad bit from the memory received by the receiver **515** is replaced by the good bit for that location. The good bit is obtained from the alternative defects data file **517**. This is done by switching the multiplexer **523** to receive the good bit from the alternative defects data file instead of the bad bit from the receiver **515**, as the data is sent to the FIFO **519**. Once the corrected data is in the FIFO it is ready to be sent to buffer memory or system memory (not shown). The data is sent from the controller's FIFO **519** to the system memory by the controller's DMA controller **507**. This controller **507** then requests and gets access to the system bus and puts out an address and gates the data via an output interface **525** out to the system bus. This is done as each byte gets loaded into the FIFO **519**. As the corrected data is loaded into the FIFO it will also be gated into an ECC hardware **527** where the data file will be acted on by the ECC.

Thus in the manner described, the data read from the memory device **33** is gated through the controller **31** to be sent to the system. This process continues until the last bit of addressed data has been transferred.

In spite of defect mapping of previously detected defective cells, new hard errors might occur since the last mapping. As the dynamic defect mapping constantly "puts away" new defective cells, the latest hard error that may arise between defect mapping would be adequately handled by the ECC. As the data is gated through the controller **31**, the controller is gating the ECC bits into the ECC hardware **527** to determine if the stored value matched the just calculated remainder value. If it matches then the data transferred out to the system memory was good and the read operation was completed. However, if the ECC registers an error then a correction calculation on the data sent to system memory is performed and the corrected data retransmitted. The method for calculating the error can be done in hardware or software by conventional methods. The ECC is also able to calculate and locate the defective cell causing the error. This may be used by the controller **31** to update the defect map associated with the sector in which the defective cell is detected. In this manner, hard errors are constantly removed from the Flash EEPROM system.

FIG. 7 illustrates the write data path control in the preferred embodiment. The first portion of a write sequence is similar to a read sequence described previously. The microprocessor first loads the Address pointers for the memory device **33** and the DMA as in the read sequence. It also loads the header desired into the address generator **503** and the command queue into the command sequencer **511**. The command queue is loaded with a read header command first. Thereafter, control is passed over to the controller **31**. The controller then gates the address and command to the memory device **33**, as in the read sequence. The memory device returns header data through controller's receiver **515**. The controller compares the received header data to the expected value (stored in the holding register file **509**). If the compare is correct, the proper location is verified and the sequence continues. Then the controller loads the defective address pointers from the memory device **33** into the holding register file **509** and the alternative data into the alternative defects data file **517**.

Next, the controller begins to fetch the write data from system memory (not shown). It does this by getting access to the system bus, outputs the memory or bus address and does the read cycle. It pulls the data into a FIFO **601** through an input interface **603**. The controller then shifts the starting sector address (lowest byte address) from the address generator **503** to the selected memory device **33**. This is followed by data from the FIFO **601**. These data are routed through multiplexers **605** and **513** and converted to serial format before being sent to the memory device **33**. This sequence continues until all bytes for a write cycle have been loaded into the selected memory.

A pipeline architecture is employed to provide efficient throughput as the data is gated from the FIFO **601** to the selected memory **33**. The data gated out of the FIFO **601** is sent to the ECC hardware **527** where a remainder value will be calculated within the ECC. In the next stage, as the data is being sent to the memory device through multiplexers **605** and **513**, the comparator **521** is comparing its address from the address generator **503** to the defect pointer address values in the holding register file **509**. When a match occurs, indicating that a defective location is about to be written, the controller saves this bit into the alternative defect data file **517**. At the same time, all bad bits sent to memory will be sent as zeroes.

After the bytes for a write cycle have been loaded into the selected memory device, the controller issues a program command to the memory device and initiate a write cycle. Optimized implementations of write operation for Flash EEPROM device have been disclosed in two previously cited co-pending U.S. patent applications, Ser. No. 204,175 now U.S. Pat. No. 5,095,344, and one entitled "Multi-State EEPROM Read and Write Circuits and Techniques," Ser. No. 07/337,579, filed Apr. 13, 1989, now abandoned. Relevant portions of the disclosures are hereby incorporated by reference. Briefly, during the write cycle, the controller applies a pulse of programming (or writing) voltages. This is followed by a verify read to determine if all the bits have been programmed properly. If the bits did not verify, the controller repeats the program/verify cycle until all bits are correctly programmed.

If a bit fails to verify after prolonged program/verify cycling, the controller will designate that bit as defective and update the defect map accordingly. The updating is done dynamically, as soon as the defective cell is detected. Similar actions are taken in the case of failure in erase verify.

After all the bits have been programmed and verified, the controller loads the next data bits from the FIFO 601 and addresses the next location in the addressed sector. It then performs another program/verify sequence on the next set of bytes. The sequence continues until the end of the data for that sector. Once this has occurred, the controller addresses the shadow memory (header area) associated with the sector (see FIG. 5) and writes the contents of the ECC registers into this area.

In addition, the collection of bits that was flagged as defective and were saved in the alternative defects data file 516 is then written in memory at the alternative defects data locations (see FIG. 5), thereby saving the good bit values to be used on a subsequent read. Once these data groups are written and verified, the sector write is considered completed.

The present invention also has provision for defect mapping of the whole sector, but only after the number of defective cells in the sector has exceeded the cell defect mapping's capacity for that specific sector. A count is kept of the number of defective cells in each sector. When the number in a sector exceeds a predetermined value, the controller marks that sector as defective and maps it to another sector. The defect pointer for the linked sectors may be stored in a sector defect map. The sector defect map may be located in the original defective sector if its spare area is sufficiently defect-free. However, when the data area of the sector has accumulated a large number of defects, it is quite likely that the spare area will also be full of defects.

Thus, it is preferable in another embodiment to locate the sector map in another memory maintained by the controller. The memory may be located in the controller hardware or be part of the Flash EEPROM memory. When the controller is given an address to access data, the controller compares this address against the sector defect map. If a match occurs then access to the defective sector is denied and the substitute address present in the defect map is entered, and the corresponding substitute sector is accessed instead.

In yet another embodiment, the sector remapping is performed by the microprocessor. The microprocessor looks at the incoming address and compares it against the sector defect map. If a match occurs, it does not issue the command to the controller but instead substitute the alternative location as the new command.

Apart from the much higher speed of the solid-state disk, another advantage is the lack of mechanical parts. The long

seek times, rotational latency inherent in disk drives are not present. In addition, the long synchronization times, sync mark detects and write gaps are not required. Thus the overhead needed for accessing the location where data is to be read or written is much less. All of these simplifications and lack of constraints result in a much faster system with much reduced overheads. In addition, the files can be arranged in memory in any address order desired, only requiring the controller to know how to get at the data as needed.

Another feature of the invention is that defect mapping is implemented without the need to interrupt the data stream transferred to or from the sector. The data in a block which may contain errors are transferred regardless, and is corrected afterwards. Preserving the sequential addressing will result in higher speed by itself. Further, it allows the implementation of an efficient pipeline architecture in the read and write data paths.

Write Cache System

Cache memory is generally used to speed up the performance of systems having slower access devices. For example in a computer system, access of data from disk storage is slow and the speed would be greatly improved if the data could be obtained from the much faster RAM. Typically a part of system RAM is used as a cache for temporarily holding the most recently accessed data from disk. The next time the data is needed, it may be obtained from the fast cache instead of the slow disk. The scheme works well in situations where the same data is repeatedly operated on. This is the case in most structures and programs since the computer tends to work within a small area of memory at a time in running a program. Another example of caching is the using of faster SRAM cache to speed up access of data normally stored in cheaper but slower DRAM.

Most of the conventional cache designs are read caches for speeding up reads from memory. In some cases, write caches are used for speeding up writes to memory. However in the case of writes to system memory (e.g. disks), data is still being written to system memory directly every time they occur, while being written into cache at the same time. This is done because of concern for loss of updated data files in case of power loss. If the write data is only stored in the cache memory (volatile) a loss of power will result in the new updated files being lost from cache before having the old data updated in system memory (non-volatile). The system will then be operating on the old data when these files are used in further processing. The need to write to main memory every time defeats the caching mechanism for writes. Read caching does not have this concern since the data that could be lost from cache has a backup on disk.

In the present invention, a system of Flash EEPROM is used to provide non-volatile memory in place of traditional system memories such as disk storage. However, Flash EEPROM memory is subject to wearing out by excessive program/erase cycles. Even with the improved Flash EEPROM memory device as disclosed in co-pending U.S. patent applications, Ser. No. 204,175, now U.S. Pat. No. 5,095,344, and Harari, Ser. No. 07/337,579, filed Apr. 13, 1989, now abandoned, filed on the same day as the present application, the endurance limit is approximately 10^6 program/erase cycles. In a ten-year projected life time of the device, this translates to a limit of one program/erase cycle per 5 minutes. This may be marginal in normal computer usage.

To overcome this problem, a cache memory is used in a novel way to insulate the Flash EEPROM memory device from enduring too many program/erase cycles. The primary

13

function of the cache is to act on writes to the Flash EEPROM memory and not on reads of the Flash EEPROM memory, unlike the case with traditional caches. Instead of writing to the Flash EEPROM memory every time the data is updated, the data may be operated on several times in the cache before being committed to the Flash EEPROM memory. This reduces the number of writes to the Flash EEPROM memory. Also, by writing mostly into the faster cache memory and reducing the number of writes to the slower Flash EEPROM, an additional benefit is the increase in system write throughput.

A relatively small size cache memory is quite effective to implement the present invention. This helps to overcome the problem of data loss in the volatile cache memory during a power loss. In that event, it is relatively easy to have sufficient power reserve to maintain the cache memory long enough and have the data dumped into a non-volatile memory such as a specially reserved space in the Flash EEPROM memory. In the event of a power down or and power loss to the system, the write cache system may be isolated from the system and a dedicated rechargeable power supply may be switch in only to power the cache system and the reserved space in the Flash EEPROM memory.

FIG. 8 illustrates schematically a cache system 701 as part of the controller, according to the present invention. On one hand the cache system 701 is connected to the Flash EEPROM memory array 33. On the other hand it is connected to the microprocessor system (not shown) through a host interface 703. The cache system 701 has two memories. One is a cache memory 705 for temporarily holding write data files. The other is a tag memory 709 for storing relevant information about the data files held in the cache memory 705. A memory timing/control circuit 713 controls the writing of data files from the cache memory 705 to the Flash EEPROM memory 33. The memory control circuit 713 is responsive to the information stored in the tag memory as well as a power sensing input 715 with is connected through the host interface 703 via a line 717 to the power supply of the microprocessor system. A power loss in the microprocessor system will be sensed by the memory control circuit 713 which will then down load all the data files in the volatile cache memory 705 to the non-volatile Flash EEPROM memory 33.

In the present invention, the Flash EEPROM memory array 33 is organized into sectors (typically 512 byte size) such that all memory cells within each sector are erasable together. Thus each sector may be considered to store a data file and a write operation on the memory array acts on one or more such files.

During read of a new sector in the Flash EEPROM memory 33, the data file is read out and sent directly to the host through the controller. This file is not used to fill the cache memory 705 as is done in the traditional cache systems.

After the host system has processed the data within a file and wishes to write it back to the Flash EEPROM memory 33, it accesses the cache system 701 with a write cycle request. The controller then intercepts this request and acts on the cycle.

In one embodiment of the invention, the data file is written to the cache memory 705. At the same time, two other pieces of information about the data file are written to a tag memory 709. The first is a file pointer which identifies the file present in the cache memory 705. The second is a time stamp that tells what time the file was last written into the cache memory. In this way, each time the host wishes to write to the Flash EEPROM memory 33, the data file is actually first stored in the cache memory 705 along with pointers and time stamps in the tag memory 709.

14

In another embodiment of the invention, when a write from the host occurs, the controller first checks to see if that file already existed in the cache memory 705 or has been tagged in the tag memory 709. If it has not been tagged, the file is written to the Flash memory 33, while its identifier and time stamp are written to the tag memory 709. If the file already is present in the cache memory or has been tagged, it is updated in the cache memory and not written to the Flash memory. In this way only infrequently used data files are written into the Flash memory while frequently used data files are trapped in the cache memory.

In yet another embodiment of the invention, when a write from the host occurs, the controller first checks to see if that data file has been last written anywhere within a predetermined period of time (for example, 5 minutes). If it has not, the data file is written to the Flash memory 33, while its identifier and time stamp are written to the tag memory 709. If the data file has been last written within the predetermined period of time, it is written into the cache memory 705 and not written to the Flash memory. At the same time, its identifier and time stamp are written to the tag memory 709 as in the other embodiments. In this way also, only infrequently used data files are written into the Flash memory while frequently used data files are trapped in the cache memory.

In all embodiments, over time the cache memory 705 will start to fill up. When the controller has detected that some predetermined state of fullness has been reached, it begins to archive preferentially some files over others in the cache memory 705 by writing them to the Flash memory 33.

In either embodiments, over time the cache memory 705 will start to fill up. When the controller has detected that some predetermined state of fullness has been reached, it begins to archive preferentially some files over others in the cache memory 705 by writing them to the Flash memory 33. The file identifier tag bits for these files are then reset, indicating that these files may be written over. This makes room for new data files entering the cache memory.

The controller is responsible for first moving the least active files back into the Flash memory 33 to make room for new active files. To keep track of each file's activity level, the time stamp for each file is incremented by the controller at every time step unless reset by a new activity of the file. The timing is provided by timers 711. At every time step (count), the controller systematically accesses each data file in the cache memory and reads the last time stamp written for this data file. The controller then increments the time stamp by another time step (i.e. increments the count by one).

Two things can happen to a file's time stamp, depending on the activity of the file. One possibility is for the time stamp to be reset in the event of a new activity occurring. The other possibility is that no new activity occurs for the file and the time stamp continues to increment until the file is removed from the cache. In practice a maximum limit may be reached if the time stamp is allowed to increase indefinitely. For example, the system may allow the time stamp to increment to a maximum period of inactivity of 5 minutes. Thus, when a data file is written in the cache memory, the time stamp for the file is set at its initial value. Then the time stamp will start to age, incrementing at every time step unless reset to its initial value again by another write update. After say, 5 minutes of inactivity, the time stamp has incremented to a maximum terminal count.

In one embodiment of keeping count, a bit can be shifted one place in a shift register each time a count increment for a file occurs. If the file is updated (a new activity has

occurred) the bit's location will be reset to the initial location of the shift register. On the other hand, if the file remains inactive the bit will eventually be shifted to the terminal shift position. In another embodiment, a count value for each file is stored and incremented at each time step. After each increment, the count value is compared to a master counter, the difference being the time delay in question.

Thus, if a file is active its incremented time stamp is reset back to the initial value each time the data file is rewritten. In this manner, files that are constantly updated will have low time stamp identifiers and will be kept in cache until their activity decreases. After a period of inactivity has expired, they acquire the maximum time stamp identifiers. The inactive files are eventually archived to the Flash memory freeing space in the cache memory for new, more active files. Space is also freed up in the tag memory when these inactive files are moved to the Flash memory.

At any time when room must be made available for new data files coming into the cache memory, the controller removes some of the older files and archives them to the Flash memory 33. Scheduling is done by a memory timing/control circuit 713 in the controller. The decision to archive the file is based on several criteria. The controller looks at the frequency of writes occurring in the system and looks at how full the cache is. If there is still room in the cache, no archiving need to be done. If more room is needed, the files with the earliest time stamps are first removed and archived to the Flash memory.

Although the invention has been described with implementation in hardware in the controller, it is to be understood that other implementations are possible. For example, the cache system may be located elsewhere in the system, or be implemented by software using the existing microprocessor system. Such variations are within the scope of protection for the present invention.

The Profile of how often data is written back to the Flash memory is determined by several factors. It depends on the size of the cache memory and the frequency of writes occurring in the system. With a small cache memory system, only the highest frequency files will be cached. Less frequently accessed files will also be cached with increasing cache memory size. In the present invention, a relatively cheap and small amount of cache memory, preferably about 1 Mbyte, may be used to good advantage. By not constantly writing the most active files (the top 5%), the write frequency of the Flash EEPROM may be reduced from the usual one every millisecond to one every 5 minutes. In this way the wear-out time for the memory can be extended almost indefinitely. This improvement is also accompanied by increased system performance during write.

Incorporating time tag into the write cache concept has the advantage that the size of the write cache buffer memory can be relatively small, since it is used only to store frequently written data files, with all other files written directly into the Flash EEPROM memory. A second advantage is that the management of moving data files in and out of the write cache buffer can be automated since it does not require advanced knowledge of which data files are to be called next.

The various aspects of the present invention that have been described co-operate in a system of Flash EEPROM memory array to make the Flash EEPROM memory a viable alternative to conventional non-volatile mass storage devices.

While the embodiments of the various aspects of the present invention that have been described are the preferred implementation, those skilled in the art will understand that

variations thereof may also be possible. Therefore, the invention is entitled to protection within the full scope of the appended claims.

We claim:

1. A method for extending the life of Flash EEPROM memory in a Flash EEPROM system, comprising the steps of: temporarily storing data files from a host system intended for the Flash EEPROM memory in a cache memory, said cache memory able to undergo significantly more write/erase cycles than the Flash EEPROM memory; writing new data files into the cache memory instead of the Flash EEPROM memory in response to a write request from the host system; determining the time since each data file was last written into said cache memory; and moving first a data file having the longest time since last written first from the cache memory to the Flash EEPROM memory when additional space for new data files is required in the cache memory, thereby substantially reducing the number of actual writes and associated stress to the Flash EEPROM memory.
2. A method for extending the life of Flash EEPROM memory in a Flash EEPROM system, comprising the steps of: temporarily storing data files from a host system intended for the Flash EEPROM memory in a cache memory, said cache memory able to undergo significantly more write/erase cycles than the Flash EEPROM memory; writing new data files into the cache memory instead of the Flash EEPROM memory in response to a write request from the host system; storing the identity of data files and the time each data file was last written into said cache memory in a tag memory; and by reference to the tag memory, moving data file having the longest time since last written first from the cache memory to the Flash EEPROM memory when additional space for new data files is required in the cache memory, thereby substantially reducing the number of actual writes and associated stress to the Flash EEPROM memory.
3. A method for extending the life of Flash EEPROM memory in a Flash EEPROM system, comprising the steps of: temporarily storing data files from a host system intended for the Flash EEPROM memory in a cache memory, said cache memory able to undergo significantly more write/erase cycles than the Flash EEPROM memory; in response to a write request from the host system, writing a data file either into the Flash EEPROM memory when a previous copy of said data file is not present in the cache memory, or into the cache memory, without writing at that time into the Flash EEPROM memory, when a previous copy of said data file is present in the cache memory; moving data file having the longest time since last written first from the cache memory to the Flash EEPROM memory when additional space for new data files is required in the cache memory, thereby substantially reducing the number of actual writes and associated stress to the Flash EEPROM memory; and in response to read requests from the host system, reading data files from the flash EEPROM memory rather than from the cache memory.
4. A method for extending the life of Flash EEPROM memory in a Flash EEPROM system, comprising the steps of: temporarily storing data files from a host system intended for the Flash EEPROM memory in a cache memory, said

17

cache memory able to undergo significantly more write/erase cycles than the Flash EEprom memory;

in response to a write request from the host system, writing a data file either into the Flash EEprom memory when said data file is last written after a predetermined period of time, or into the cache memory, without writing at that time into the Flash EEprom memory, when said data file is last written within the predetermined period of time;

moving data file having the longest time since last written first from the cache memory to the Flash EEprom memory when additional space for new data files is required in the cache memory, thereby substantially reducing the number of actual writes and associated stress to the Flash EEprom memory; and

in response to read requests from the host system, reading data files from the flash EEprom memory rather than from the cache memory.

5. A method for extending the life of Flash EEprom memory in a Flash EEprom system, comprising the steps of:

temporarily storing data files from a host system intended for the Flash EEprom memory in a cache memory, said cache memory able to undergo significantly more write/erase cycles than the Flash EEprom memory;

storing the identity of data files and the time each data file was last written into said cache memory in a tag memory;

in response to a write request from the host system, writing a data file into the Flash EEprom memory when the data file is not identified in the tag memory, or into the cache memory when the data file is identified in the tag memory;

by reference to the tag memory, moving data file having the longest time since last written first from the cache memory to the Flash EEprom memory when additional space for new data files is required in the cache memory, thereby substantially reducing the number of actual writes and associated stress to the Flash EEprom memory; and

in response to read requests from the host system, reading data files from the flash EEprom memory rather than from the cache memory.

6. A method of writing data files into a system of flash EEprom cells that are programmable into more than two states in order to store more than one bit of data per cell, comprising:

temporarily storing, in a cache memory, data files from a host system intended for the flash EEprom memory;

writing new data files into the cache memory instead of the flash EEprom memory in response to a write request from the host system;

determining the time since each data file was last written into said cache memory; and

moving from the cache memory a data file having the longest time since last written, when additional space for new data files is required in the cache memory, into the flash EEprom memory by programming individual flash EEPROM cells into one of said more than two programmable states.

7. A method of writing data files into a system of flash EEprom cells that are programmable into more than two states in order to store more than one bit of data per cell, comprising:

temporarily storing, in a cache memory, data files from a host system intended for the flash EEprom memory;

18

writing new data files into the cache memory instead of the flash EEprom memory in response to a write request from the host system;

storing, in a tag memory, the identity of data files and the time each data file was last written into said cache memory; and

by reference to the tag memory, moving a data file having the longest time since last written first from the cache memory to the flash EEprom when additional space for new data files is required in the cache memory, individual flash EEPROM cells being written into one of said more than two programmable states.

8. A method of writing data files into a system of flash EEprom cells that are programmable into more than two states in order to store more than one bit of data per cell, comprising:

temporarily storing, in a cache memory, data files from a host system intended for the flash EEprom memory;

in response to a write request from the host system, writing a data file either into the flash EEprom memory when a previous copy of said data file is not present in the cache memory, or into the cache memory when a previous copy of said data file is present in the cache memory;

moving a data file having the longest time since last written first from the cache memory to the flash EEprom memory when additional space for new data files is required in the cache memory, thereby substantially reducing the number of actual writes to the flash EEprom memory;

wherein a data file is written into the flash EEprom by programming individual cells thereof into one of said more than two programmable states; and

in response to read requests from the host system, reading data files from the flash EEprom memory rather than from the cache memory.

9. A method of writing data files into a system of flash EEprom cells that are programmable into more than two states in order to store more than one bit of data per cell, comprising:

temporarily storing, in a cache memory, data files from a host system intended for the flash EEprom memory;

in response to a write request from the host system, writing a data file either into the flash EEprom memory when said data file is last written after a predetermined period of time, or into the cache memory when said data file is last written within the predetermined period of time;

moving from the cache memory a data file having the longest time since last written, when additional space for new data files is required in the cache memory, into the flash EEprom memory by programming individual flash EEPROM cells into one of said more than two programmable states; and

in response to read requests from the host system, reading data files from the flash EEprom memory rather than from the cache memory.

10. A method of writing data files into a system of flash EEprom cells that are programmable into more than two states in order to store more than one bit of data per cell, comprising:

temporarily storing, in a cache memory, data files from a host system intended for the flash EEprom memory,

storing, in a tag memory, the identity of data files and the time each data file was last written into said cache memory;

19

in response to a write request from the host system, writing a data file into the flash EEPROM memory when the data file is not identified in the tag memory, or into the cache memory when the data file is identified in the tag memory;

by reference to the tag memory, moving data file having the longest time since last written first from the cache memory to the flash EEPROM memory when additional space for new data files is required in the cache memory;

wherein a data file is written into the flash EEPROM by programming individual cells thereof into one of said more than two programmable states; and

in response to read requests from the host system, reading data files from the flash EEPROM memory rather than from the cache memory.

11. A method of a host computing system utilizing a mass data storage system that includes a non-volatile flash EEPROM system to store data files, comprising:

writing individual new data files from the host to a cache memory provided as part of the mass data storage system without writing the new data files to the flash EEPROM system,

selecting a data file stored in the cache memory based upon a length of time since the data file was last written into the cache memory,

writing the selected data file from the cache memory to the flash EEPROM system, and

when written into the flash EEPROM memory, reading a data file requested by the host system from the flash EEPROM system rather than from the cache memory.

12. The method of claim 11, wherein the individual new data files are written into a volatile random-access-memory as the cache memory.

13. The method of claim 11, wherein the individual new data files are written into a memory separate from the flash EEPROM system as the cache memory.

14. The method of claim 11, wherein each of the steps thereof is accomplished within the mass data storage system.

15. The method of claim 14, wherein the mass data storage system provides an ATA interface with the host computing system.

16. The method of claim 14, wherein the mass data storage system is provided on a card that is electrically and mechanically removably connectable with the host computing system.

20

17. The method of any one of claims 11–16, wherein selecting a data file and writing the selected data file to the flash EEPROM memory is caused to occur when additional space for new data files is required in the cache memory.

18. The method of claim 17, wherein writing the selected data file to the flash EEPROM system includes programming individual memory cells of the flash EEPROM system into exactly two states in order to store exactly one bit of data per cell.

19. The method of claim 17, wherein writing the selected data file to the flash EEPROM system includes programming individual memory cells of the flash EEPROM system into more than two states in order to store more than one bit of data per cell.

20. The method of any one of claims 11–16, wherein writing the selected data file to the flash EEPROM system includes programming individual memory cells of the flash EEPROM system into exactly two states in order to store exactly one bit of data per cell.

21. The method of any one of claims 11–16, wherein writing the selected data file to the flash EEPROM system includes programming individual memory cells of the flash EEPROM system into more than two states in order to store more than one bit of data per cell.

22. The method of any one of claims 11–16, wherein data files are written to the flash EEPROM system by writing to one or more blocks of EEPROM cells, wherein the cells of each block are erasable together.

23. The method of claim 22, wherein the flash EEPROM system operates to map an address of a defective block of cells into another block of cells.

24. The method of claim 22, wherein the flash EEPROM system operates to erase the EEPROM cells in multiple designated blocks at one time.

25. The method of any one of claims 11–16, wherein writing a new data file to the cache memory from the host occurs in less time than if written directly into the flash EEPROM memory from the host.

26. The method of claim 25, wherein writing the selected data file to the flash EEPROM system includes programming individual memory cells of the flash EEPROM system into more than two states in order to store more than one bit of data per cell.

* * * * *