

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Silicon Motion Inc.,

Petitioners

v.

Unification Technologies LLC,

Patent Owner

U.S. Patent No. 11,640,359

**DECLARATION OF R. JACOB BAKER, PH.D., P.E.,
REGARDING U.S. PATENT NO. 11,640,359**

TABLE OF CONTENTS

I.	INTRODUCTION	3
II.	EDUCATIONAL AND WORK BACKGROUND	6
III.	UNDERSTANDING OF THE LAW	14
A.	Scope of Declaration and Legal Standards	14
B.	Legal Standard for Prior Art.....	14
C.	Legal Standard for Obviousness	16
IV.	LEVEL OF ORDINARY SKILL IN THE ART.....	21
V.	PUBLIC AVAILABILITY OF SWSTE’05	24
VI.	BACKGROUND OF THE TECHNOLOGY.....	25
A.	Flash Memory Technology, Generally	25
B.	Overview of Bennett	45
C.	Overview of Suda.....	54
D.	Overview of Zipprich	64
E.	Overview of SwSTE’05	64
VII.	OBVIOUSNESS OF THE CLAIMS BASED ON SUDA.....	65
VIII.	OBVIOUSNESS OF THE CLAIMS BASED ON BENNETT	79
IX.	OBVIOUSNESS OF CLAIM 11 OVER SUDA AND/OR BENNETT IN FURTHER VIEW OF ZIPPRICH AND KNOWLEDGE OF A POSITA	98
X.	OBVIOUSNESS OF CLAIMS 7, 10-13 OVER BENNETT IN VIEW OF SUDA	99
XI.	OBVIOUSNESS OF CLAIMS 13-15 OVER BENNETT AND/OR SUDA IN VIEW OF SWSTE’05	99
X.	RESERVATION OF RIGHTS	101
XI.	CONCLUSION.....	101

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

I, R. Jacob Baker Ph.D., P.E., declare as follows:

I. INTRODUCTION

1. My name is R. Jacob Baker Ph.D., P.E., and I am a Professor of Electrical and Computer Engineering at the University of Nevada, Las Vegas. I have prepared this declaration as an expert witness on behalf of Petitioners. In this declaration, I give my opinions as to whether claims 7 and 10-17 of U.S. Patent No. 11,640,359 (“the ‘359 patent”) (Ex. 1001) are valid. I also provide the technical bases for these opinions. I also give my opinions on the public availability of certain documents.

2. This declaration contains statements of my opinions formed to date, and the bases and rationale for these opinions. I reserve the right to offer supplemental or amended opinions based on further review of materials in this case, including additional evidence, opinions, discovery argument, testimony and/or other information that may be provided to me after the date of this declaration.

3. For my efforts in connection with the preparation of this declaration, I have been compensated at my usual and customary rate for this type of consulting activity of \$765 per hour, plus reimbursement for any additional reasonable expenses. My compensation is in no way tied to the content of this declaration, the

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

substance of my opinions, or the outcome of this litigation. I have no other interests in the proceeding or with any of the parties.

4. I considered and relied upon the ‘359 patent, its prosecution history and at least the following materials in preparing this declaration:

Ex. No.	Brief Description
1001	U.S. Pat. No. 11,640,359 B2, titled “SYSTEMS AND METHODS FOR IDENTIFYING STORAGE RESOURCES THAT ARE NOT IN USE” to Flynn et al.
1002	U.S. Pat. No. 7,624,239 B2, titled “METHODS FOR THE MANAGEMENT OF ERASE OPERATIONS IN NON-VOLATILE MEMORIES” to Bennett et al.
1003	U.S. Pat. No. 7,057,942 B2, titled “MEMORY MANAGEMENT DEVICE AND MEMORY DEVICE” to Suda et al.
1005	American National Standard for Information Technology—AT Attachment with Packet Interface – 6 (ATA/ATAPI-6), ANSI INCITS 361-2002 (Sept. 2002) (excerpts filed with permission).
1006	U.S. Pat. Pub. 2003/0079078, titled “CONFIRMATION OF SECURE DATA FILE ERASURE” to Zipprich et al.
1007	Eran Gal et al., <i>Mapping Structures for Flash Memories: Techniques and Open Problems</i> , PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON SOFTWARE—SCIENCE, TECHNOLOGY & ENGINEERING (“SwSTE’05”) (digital version), Herzlia, Israel, 2005, pp. 83-92, doi: 10.1109/SWSTE.2005.14.
1008	Original Complaint for Patent Infringement, <i>Unification Techs. LLC v. Silicon Motion Inc.</i> , No. 2:23-cv-266 (E.D. Tex. 2023).
1009	U.S. Provisional Pat. No. 60/873,111 titled “Elemental Blade

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

Ex. No.	Brief Description
	System,” to Flynn et al.
1011	Frank Shu, <i>Notification of Deleted Data Proposal for ATA8-ACS2</i> , T13 (rev. 0 Apr. 21, 2007).
1012	Frank Shu & Nathan Obr, <i>Notification of Deleted Data Proposal for ATA8-ACS2 Revision 1</i> , T13 (July 26, 2007).
1013	Frank Shu & Nathan Obr, <i>Notification of Deleted Data Proposal for ATA8-ACS2 Revision 2</i> , T13 (September 5, 2007).
1014	Public file history of U.S. Pat. No. 11,640,359 B2, titled “SYSTEMS AND METHODS FOR IDENTIFYING STORAGE RESOURCES THAT ARE NOT IN USE” to Flynn et al.
1017	Expert Report of Sylvia Hall-Ellis, Ph.D. in Support of Public Availability of the Gal Publication.
1019	American National Standard for Information Technology—AT Attachment with Packet Interface – 7 Volume 1 – Register Delivered Command Set, Logical Register Set (ATA/ATAPI-7 V1), ANSI INCITS 397-2005 (Feb. 7, 2005) (excerpts filed with permission).
1020	Serial ATA (SATA) Revision 2.5, Serial ATA International Organization (Oct. 27, 2005).
1021	WILLIAM D. BROWN & JOE E. BREWER, NONVOLATILE SEMICONDUCTOR MEMORY TECHNOLOGY (IEEE 1998).
1022	BRIAN DIPERT & MARKUS LEVY, DESIGNING WITH FLASH MEMORY (Annabooks 1994).
1023	H. Nijjima, <i>Design of a Solid-State File Using Flash EEPROM</i> , IBM JOURNAL OF RESEARCH AND DEVELOPMENT, vol. 39, no. 5, pp. 531-545, Sep. 1995.
1024	U.S. Pat. No. 6,766,432 B2, titled “MEMORY MANAGEMENT SYSTEM SUPPORTING OBJECT DELETION IN NON-VOLATILE MEMORY” to Saltz et al.

Ex. No.	Brief Description
1025	<i>Proceedings. IEEE International Conference on Software – Science, Technology and Engineering</i> , IEEE COMPUTER SOCIETY, www.computer.org/csdl/proceedings/swste/2005/12OmNC17hWm (last visited Oct. 30, 2020).
1026	<i>Mapping Structures for Flash Memories: Techniques and Open Problems</i> , IEEE XPLORE, https://ieeexplore.ieee.org/document/1421068 (last visited Dec. 18, 2020).

II. EDUCATIONAL AND WORK BACKGROUND

5. My qualifications generally are set forth in my *curriculum vitae*, which is submitted as Ex. 1015.

6. I have been working as an Engineer since 1985 and I have been teaching Electrical and Computer Engineering courses since 1991. I am currently a Professor of Electrical and Computer Engineering at the University of Nevada, Las Vegas (“UNLV”). I am also currently an industry consultant for Freedom Photonics. I am the named inventor on over 150 U.S. patents resulting from my industry work.

7. I received B.S. and M.S. degrees in Electrical Engineering from UNLV in 1986 and 1988, respectively. I received my Ph.D. in Electrical Engineering from the University of Nevada, Reno, in 1993.

8. My doctoral research, culminating in the award of a Ph.D., investigated the use of power MOSFETs in the design of very high peak power, and high-speed, instrumentation. I developed techniques to reliably stack power MOSFETs to switch higher voltages, that is, greater than 1,000 V and 100 Amps of current with nanosecond switching times. This work was reported in the paper entitled “Transformerless Capacitive Coupling of Gate Signals for Series Operation of Power MOSFET Devices,” published in the IEEE Transactions on Power Electronics. The paper received the Best Paper Award in 2000.

Industry Experience

9. I have done technical and expert witness consulting for over 200 companies since I started working as an engineer in 1985. From 1985 to 1993 I worked for EG&G Energy Measurements and the Lawrence Livermore National Laboratory designing nuclear diagnostic instrumentation for underground nuclear weapon tests at the Nevada test site. During this time, I designed, and oversaw the fabrication of, over 30 electronic and electro-optic instruments including high-speed cable and fiber-optic receiver/transmitters, PLLs, frame and bit-syncs, data converters, streak-camera sweep circuits, Pockel’s cell drivers, micro-channel plate gating circuits, charging circuits for battery backup of equipment for recording test data, and analog oscilloscope electronics.

10. My work during this time, as one example, had a direct impact on my doctoral research work using power MOSFETs, subsequent publishing efforts, and industry designs. In addition to the 2000 Best Paper Award from the IEEE Power Electronics Society, I published several other papers in related areas while working in industry. I hold a patent, Patent No. 5,874,830, in the area of power supply design, titled, “Adaptively biased voltage regulator and operating method,” which was issued on February 23, 1999. I have designed dozens of linear and switching power supplies for commercial products and scientific instrumentation.

11. I am a licensed Professional Engineer and have extensive industry experience in circuit design, fabrication, and manufacture of Dynamic Random Access Memory (DRAM) semiconductor integrated circuit chips, Phase-Change Random Access Memory (PCRAM) chips, and CMOS Image Sensors (CISs) at Micron Technology, Inc. (“MTI”) in Boise, Idaho. I spent considerable time working on the development of flash memory chips while at MTI. My efforts resulted in more than a dozen patents relating to flash memory. One of my projects at MTI included the development, design, and testing of circuit design techniques for a multi-level cell (MLC) flash memory using signal processing. This effort resulted in higher-density memories for use in solid-state drives and flash memory cards having an ATA interface that are ubiquitous in consumer

electronics including cameras and data storage systems. Further, the use of higher-density memory can result in fewer changes in the flash translation layer for logical to physical addressing, less need for garbage collection, and larger data segments that can improve a computing system's performance. Another project I worked on at MTI focused on the design of buffers for high-speed double-data rate DRAM which resulted in around 10 U.S. patents in buffer design. Among many other experiences, I led the development of the delay locked loop (DLL) in the late 1990s so that MTI DRAM products could transition to the DDR memory protocol, used in mobile and non-mobile (server, desktop, cell phones, tablets, etc.) computing systems as main computer memory, for addressing and controlling accesses to memory via interprocess communications (IPC) with the memory controller (MC). I provided technical assistance with MTI acquisition of Photobit during 2001 and 2002, including transitioning the manufacture of CIS products into MTI's process technology. Further, I did consulting work at Sun Microsystems and then Oracle on the design of memory modules during 2009 and 2010. This work entailed the design of low-power, high-speed, and wide interconnection methods with the goal of transmitting data to/from the memory module and the MC at higher speeds.

12. I have extensive experience in the development of instrumentation and commercial products in a multitude of areas including: integrated electrical/biological circuits and systems, array (memory, imagers, and displays) circuit design, CMOS analog and digital circuit design, diagnostic electrical and electro-optic instrumentation for scientific research, CAD tool development and online tutorials, low-power interconnect and packaging techniques, design of communication/interface circuits (to meet commercial standards such as USB, firewire, DDR, PCIe, SPI, etc.), circuit design for the use and storage of renewable energy, and power electronics. For example, a part of my research at Boise State, for many years, focused on the use of Thru-Silicon-Vias (TSVs), aka Thru-Wafer Vias (TWVs), for high-density packaging. These packaging techniques were utilized in the memory module development work I did with Sun Microsystems and Oracle. As another example, I've designed circuitry for use in implementing Universal Serial Bus (USB) interface circuits while I did consulting at Tower Semiconductor. I designed PCI communication circuits for IPC between a Graphics Processor Unit (GPU) and memory while consulting for Rendition, Inc.

13. My current research work is focused in part on the design of integrated circuits for wireless sensing using LIDAR (LIght Detection And Ranging). I have worked with several companies in the development of these

circuits and systems including Freedom Photonics, Aerius Photonics and FLIR. In the early 1990s, I worked on wireless systems for wideband impulse radar while at Lawrence Livermore Laboratory. Further, part of my research for several years focused on the digitization of IQ channels using delta-sigma modulation. The knowledge and experience gained from this effort are reflected in my textbook *CMOS Mixed-Signal Circuit Design* and a presentation, which I have presented at several universities and companies,

http://cmosedu.com/jbaker/papers/talks/BP_DSM_talk.pdf.

Academic Experience

14. I was an adjunct faculty member in the Electrical Engineering department of the University of Nevada, Las Vegas in 1991 and 1992. From 1993 to 2000, I served on the faculty at the University of Idaho as an Assistant Professor and then as a tenured Associate Professor of Electrical Engineering. In 2000, I joined a new Electrical and Computer Engineering program at Boise State University (“BSU”) where I served as department chair from 2004 to 2007. At BSU, I helped establish graduate programs in Electrical and Computer Engineering including, in 2006, the university’s second Ph.D. degree. In 2012, I re-joined the faculty at UNLV. Over the course of my career as a professor, I have advised over 100 masters and doctoral students.

15. I have been recognized for my contributions as an educator in the field. While at Boise State University, I received the President's Research and Scholarship Award (2005), Honored Faculty Member recognition (2003), and Outstanding Department of Electrical Engineering Faculty recognition (2001). In 2007, I received the Frederick Emmons Terman Award (the "Father of Silicon Valley"). The Terman Award is bestowed annually upon an outstanding young electrical/computer engineering educator in recognition of the educator's contributions to the profession. In 2011 I received the IEEE Circuits and Systems Education Award. I received the Tau Beta Pi Outstanding Electrical and Computer Engineering Professor Award every year it was awarded while I have been back at UNLV.

16. I have authored several books and papers in the electrical and computer engineering area. My published books include *CMOS Circuit Design, Layout, and Simulation* (Baker, R.J., *Wiley-IEEE*, ISBN: 9781119481515 (4th ed., 2019)) and *CMOS Mixed-Signal Circuit Design* (Baker, R.J., *Wiley-IEEE*, ISBN: 9780470290262 (2nd ed., 2009) and ISBN: 9780471227540 (1st ed., 2002)). I co-authored *DRAM Circuit Design: Fundamental and High-Speed Topics* (Keeth, B., Baker, R.J., Johnson, B., and Lin, F., *Wiley-IEEE*, ISBN: 9780470184752 (2008)), *DRAM Circuit Design: A Tutorial* (Keeth, B. and Baker, R.J., *Wiley-IEEE*, ISBN:

0-7803-6014-1 (2001)), and *CMOS Circuit Design, Layout and Simulation* (Baker, R.J., Li, H.W., and Boyce, D.E., *Wiley-IEEE*, ISBN: 9780780334168 (1998)). I contributed as an editor and co-author on several other electrical and computer engineering books.

Other Relevant Experience

17. I have given more than 50 invited talks at conferences, companies, and universities. Further, I am the author or co-author of more than 100 papers and presentations in the areas of electrical and computer engineering design, fabrication and packaging.

18. I currently serve, or have served, as a volunteer on: the IEEE Press Editorial Board (1999-2004); as editor for the Wiley-IEEE Press Book Series on Microelectronic Systems (2010-2018); as the Technical Program Chair of the 2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS 2015); on the IEEE Solid-State Circuits Society (SSCS) Administrative Committee (2011-2016); as a Distinguished Lecturer for the SSCS (2012-2015); and as the Technology Editor (2012-2014) and Editor-in-Chief (2015-2020) for *IEEE Solid-State Circuits Magazine*. These meetings, groups, and publications are intended to allow researchers to share and coordinate research. My active participation in

these meetings, groups, and publications allowed me to see what other researchers in the field have been doing.

19. In addition to the above, I am an IEEE Fellow for contributions to semiconductor memory design and a member of the honor societies Eta Kappa Nu and Tau Beta Pi.

III. UNDERSTANDING OF THE LAW

A. Scope of Declaration and Legal Standards

20. I am not an attorney and will not offer opinions on the law, but I have been instructed in and applied the law as described in this section.

B. Legal Standard for Prior Art

21. I understand that a patent or other publication must first qualify as prior art before it can be used to invalidate a patent claim.

22. I am informed by counsel that the applicable law in this case for prior art is 35 U.S.C. § 102 as written before the America Invents Act.

23. I am informed that the patent owner Unification Technologies LLC (“UTL”) contends that the ‘359 patent claims invention dates or “priority dates” of December 6, 2006 and September 22, 2007.

24. I further understand that the “priority date” of a patent is the date on which it is filed, or the date on which an earlier-filed patent application is filed if

the patentee claims the benefit of priority to that earlier-filed patent application. I further understand that the priority date is significant because patents, systems, or documents that are public prior to the priority date may invalidate the claims.

25. I understand that a U.S. patent may qualify as prior art to an asserted patent if the date of publication or issuance is prior to the invention in the asserted patent. I further understand that a printed publication, such as a book or an article published in a magazine or trade publication, may qualify as prior art to an asserted patent if the date of publication is prior to the invention of the asserted patent. A paper or presentation may also qualify as prior art if presented at a publicly available seminar, conference, or trade show prior to the invention in the asserted patent.

26. I understand that a U.S. patent or published patent application may qualify as prior art to the asserted patent if the application for that U.S. patent or published patent application was filed in the United States before the invention of the asserted patent.

27. I understand that to qualify as prior art, a reference must contain an enabling disclosure that allows one of ordinary skill to practice the claims without undue experimentation.

28. I understand that documents and materials that qualify as prior art can be used to invalidate a patent claim as anticipated or as obvious.

C. Legal Standard for Obviousness

29. I have been instructed by counsel on the law regarding obviousness, and understand that even if a patent is not anticipated, it is still invalid under 35 U.S.C. § 103, as written before the America Invents Act, if the differences between the claimed subject matter and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person of ordinary skill in the pertinent art.

30. I understand that a person of ordinary skill in the art provides a reference point from which the prior art and claimed invention should be viewed. This reference point prevents a person of ordinary skill from using one's insight or hindsight in deciding whether a claim is obvious.

31. I also understand that an obviousness determination includes the consideration of various factors such as (1) the scope and content of the prior art, (2) the differences between the prior art and the challenged claims, (3) the level of ordinary skill in the pertinent art, and (4) the existence of secondary considerations such as commercial success, long-felt but unresolved needs, failure of others, etc.

32. I am informed that secondary considerations of non-obviousness may include (1) a long felt but unmet need in the prior art that was satisfied by the invention of the patent; (2) commercial success or lack of commercial success of processes covered by the patent; (3) unexpected results achieved by the invention; (4) praise of the invention by others skilled in the art; (5) taking of licenses under the patent by others; and (6) deliberate copying of the invention. I also understand that there must be a relationship or “nexus” between any such secondary indicia and the invention. I further understand that contemporaneous and independent invention by others is a secondary consideration supporting an obviousness determination.

33. I understand that an obviousness evaluation can be based on a combination of multiple prior art references. I understand that the prior art references themselves may provide a suggestion, motivation, or reason to combine, but other times the link between two or more prior art references is simple common sense. I further understand that obviousness analysis recognizes that market demand, rather than scientific literature, can drive innovation, and that a motivation to combine references may be supplied by the direction of the marketplace.

34. I understand that if a technique has been used to improve one device, and a person of ordinary skill in the art would recognize that it would improve similar devices in the same way, using the technique is obvious unless its actual application is beyond his or her skill.

35. I also understand that practical and common-sense considerations should guide a proper obviousness analysis, because familiar items may have obvious uses beyond their primary purposes. I further understand that a person of ordinary skill in the art looking to overcome a problem will often be able to fit the teachings of multiple publications together like pieces of a puzzle, although the prior art need not be like two puzzle pieces that must fit perfectly together. I understand that obviousness analysis therefore considers the inferences and creative steps that a person of ordinary skill in the art would employ under the circumstances.

36. I understand that a particular combination may be proven obvious by showing that it was obvious to try the combination. For example, when there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within his or her technical grasp because the result is likely the product not of innovation but of ordinary skill and common sense.

37. I understand that the combination of familiar elements according to known methods may be proven obvious when it does no more than yield predictable results. When a work is available in one field of endeavor, design incentives and other market forces can prompt variations of it, either in the same field or a different one. If a person of ordinary skill can implement a predictable variation, obviousness likely bars its patentability.

38. It is further my understanding that a proper obviousness analysis focuses on what was known or obvious to a person of ordinary skill in the art, not just the patentee. Accordingly, I understand that any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed. Additionally, a reference outside of the field of endeavor may still be prior art if the reference is reasonably pertinent to the particular problem being solved by the patentee.

39. I understand that a claim can be obvious in light of a single reference, without the need to combine references, if the elements of the claim that are not found explicitly or inherently in the reference can be supplied by the common sense and knowledge of one of skill in the art.

40. I understand that a person of ordinary skill could have combined two pieces of prior art or substituted one prior art element for another if the substitution

can be made with predictable results, even if the swapped-in element is different from the swapped-out element. In other words, the prior art need not be like two puzzle pieces that must fit together perfectly. The relevant question is whether prior art techniques are interoperable with respect to one another, such that that a person of skill would view them as a design choice, or whether a person of skill could apply prior art techniques into a new combined system. The person of ordinary skill would use ordinary creativity and is not limited like an automaton.

41. In sum, my understanding is that prior art teachings are properly combined where a person of ordinary skill in the art having the understanding and knowledge reflected in the prior art and motivated by the general problem facing the inventor, would have been led to make the combination of elements recited in the claims. Under this analysis, the prior art references themselves, or any need or problem known in the field of endeavor at the time of the invention, can provide a reason for combining the elements of multiple prior art references in the claimed manner.

42. I have been informed and understand that the obviousness analysis requires a comparison of the properly construed claim language to the prior art on a limitation-by-limitation basis.

43. I have written this report with the understanding that in an *inter partes* review, obviousness must be shown by a preponderance of the evidence.

IV. LEVEL OF ORDINARY SKILL IN THE ART

44. I have been informed by counsel that a “person of ordinary skill in the art at the time of the inventions” is a hypothetical person who is presumed to be familiar with the relevant field and its literature at the time of the inventions. This hypothetical person is also a person of ordinary creativity, capable of understanding the scientific principles applicable to the pertinent field.

45. In my opinion, a person of ordinary skill in the art at the time of the claimed inventions would have a bachelor’s degree in computer engineering, electrical engineering, computer science, or a closely related field, along with at least two years of experience in the design, development, implementation, or management of memory devices and systems. An individual with an advanced degree in a relevant field, such as computer or electrical engineering, would require less experience in the development and use of memory devices and systems (e.g., 1-2 years).

46. A person of ordinary skill in the art would have been familiar at least basic memory management concepts discussed in relevant parts of:

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

- William D. Brown & Joe E. Brewer, *Nonvolatile Semiconductor Memory Technology* (IEEE 1998). Ex. 1021.
- Brian Dipert & Markus Levy, *Designing with Flash Memory* (Annabooks 1994). Ex. 1022.
- Eran Gal et al., *Mapping Structures for Flash Memories: Techniques and Open Problems*, Proceedings of the IEEE International Conference on Software – Science, Technology & Engineering (SwSTE’05), Herzlia, Israel, 2005, pp. 83-92. Ex. 1007.
- H. Nijjima, *Design of a Solid-state File Using Flash EEPROM*, IBM JOURNAL OF RESEARCH AND DEVELOPMENT, vol. 39, no. 5, pp. 531-545, Sep. 1995. Ex. 1023.
- American National Standard for Institute Technology – AT Attachment with Packet Interface – 6 (ATA/ATAPI-6), INCITS 361-2002 (Sept. 2002). Ex. 1005.
- American National Standard for Institute Technology – AT Attachment with Packet Interface – 7 Volume 1 – Register Delivered Command Set, Logical Register Set (ATA/ATAPI-7 V1), INCITS 397-2005 (Feb. 7, 2005). Ex. 1019.

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

- Serial ATA (SATA) Revision 2.5, Serial ATA International Organization (Oct. 27, 2005). Ex. 1020.
- U.S. Pat. No. 7,057,942 B2, titled “MEMORY MANAGEMENT DEVICE AND MEMORY DEVICE” to Suda et al. Ex. 1003.
- U.S. Pat. No. 7,624,239 B2, titled “METHODS FOR THE MANAGEMENT OF ERASE OPERATIONS IN NON-VOLATILE MEMORIES” to Bennett et al. Ex. 1002.
- U.S. Pat. No. 6,766,432 B2, titled “MEMORY MANAGEMENT SYSTEM SUPPORTING OBJECT DELETION IN NON-VOLATILE MEMORY” to Saltz et al. Ex. 1024.

47. The basis for my familiarity with the level of ordinary skill is my interaction with large numbers of students seeking related technical degrees over many years of teaching and with people working in the relevant disciplines who were at this level of skill. In reaching this opinion, I have considered the types of problems encountered in the art, the prior art solutions to those problems, the speed with which innovations are made, the matureness of the relevant technology, and the educational and professional levels of people working in the field.

V. PUBLIC AVAILABILITY OF SWSTE'05

48. I have reviewed the publication Mapping Structures for Flash Memories: Techniques and Open Problems by Eran Gal and Sivan Toledo (“SwSTE’05”) (Ex. 1007). I have reviewed this article’s publication data on IEEE Digital Library (Ex. 1025) and on IEEE *Xplore* (Ex. 1026). As an editor of IEEE journals, I have access to and have used both of these databases during the IEEE publication process and for research in my field. IEEE publishes accurate, reliable information in these two databases as part of its ordinary publication process.

49. Based on my experience having published numerous IEEE articles, attended IEEE conferences, and serving as an IEEE editor, and based on the information from IEEE *Xplore* (Ex. 1026) and IEEE Digital Library (Ex. 1025), it is my opinion that the SwSTE’05 article was publicly presented at a conference on either February 22 or 23, 2005, and then electronically published by May 23, 2005. These dates are consistent with the typical presentation/publication schedules for IEEE papers presented at conferences, and I have no reason to doubt the accuracy of the information as reported in the IEEE Digital Library or IEEE *Xplore*. The IEEE’s conferences and papers are accessible to the general public. The IEEE advertises its conferences to POSITAs and its papers are searchable by POSITAs.

I have further reviewed the declaration of Dr. Sylvia D. Hall-Ellis (IPR2021-00344) and it further support my opinions. Ex. 1017.

VI. BACKGROUND OF THE TECHNOLOGY

A. Flash Memory Technology, Generally

50. Before the priority date, flash memory devices generally were widely used. The following nine sections summarize well-understood flash memory technologies at that time. Each of these background-level technologies would have been implemented in flash memory systems, and POSITAs would have reliably and predictably known how to do so.

1. Flash Memory, Generally

51. Computer memories can be classified as volatile and non-volatile. Volatile memories, such as DRAM, SRAM, SDRAM, and DDR-DRAM, only store data while powered by the system they are in, and lose the data stored therein when the power is turned off. Non-volatile memories retain their stored data even without power. Examples of non-volatile memories and memory systems include magnetic tape, magnetic and optical drives, read only memories (ROMs), electrically programmable read only memories (EPROMs), and electrically erasable programmable read-only memories (EEPROMs).

52. Flash memories are EEPROM memories with cells made up of floating gate transistors. Flash memories are erased in blocks and programmed (written to) in portions of a block called a page. For example, block of flash memory may have 64 pages.

53. Four characteristics of flash memory have consistently affected the design of flash memory systems. Ex. 1023 at 533. First, write operations take much longer than read operations. *Id.* Second, erase operations take much longer than write operations. *Id.* Third, a memory cell can only endure a limited number of write/erase operations. *Id.* Fourth, the memory is divided into blocks, and **all** cells in a block are erased at the same time. *Id.* In other words, flash memory does not erase individual smaller units, such as cells or pages, that make up a block; all cells and pages in an entire block are erased together.

54. The modern flash EEPROM was proposed in 1984, and big companies such as Kioxia, AMD, Intel, TI, NEC, and SanDisk contributed to the technology in the 1980s and 1990s. *See* Ex. 1021 at 193-96. Flash memory has been widely used since the 1990s in a variety of electronic devices. Before December 6, 2006, flash memory was widely used in all sorts of electronic devices, including cell phones, computers, vehicles, microsystems, and all sorts of other products. A variety of flash memory products in the form of USB flash

drives, SecureDigital (SD) cards, PCMCIA cards, CompactFlash (CF) cards, Memory Stick, MultiMediaCard (MMC), xD, and variants thereof were widely available before December 6, 2006. These products typically contained a package that included an interface component, a controller, and the flash memory itself. An example from *Designing with FLASH MEMORY* (Ex. 1022 at 66) is reproduced below:

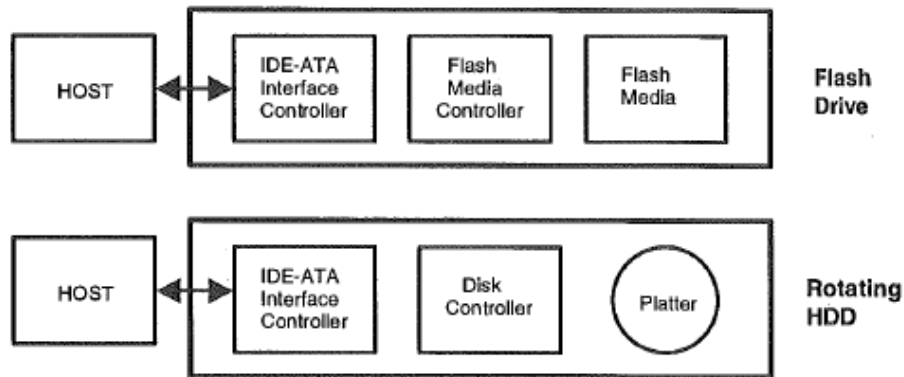


Figure 4.14: Mass Storage Architecture

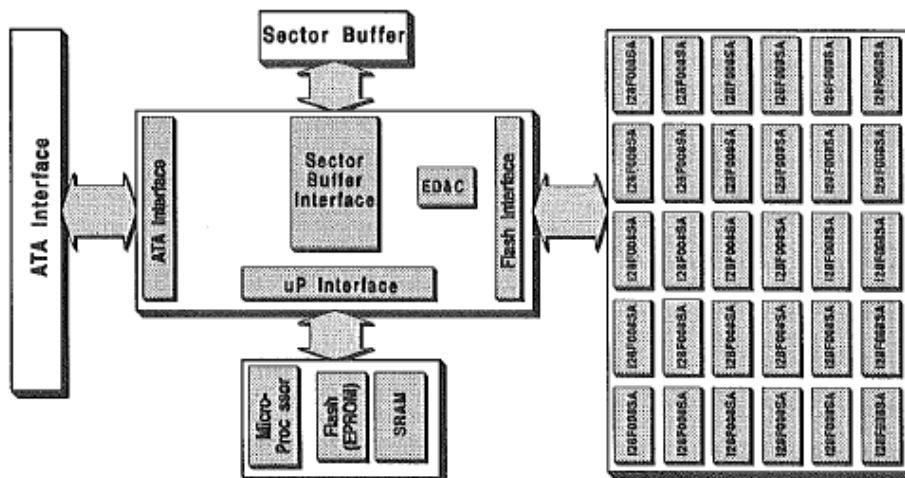


Figure 4.15: Flash Drive Architecture

Ex. 1022 at Fig. 4.15.

2. Flash Speeds Relative to Other Memory and Caches

55. Programming and erasing flash memories are relatively slow operations compared to equivalent operations in RAM. For example, depending on the particular technology, it can take a millisecond or more to program a page of flash memory, whereas writing a word to DDR DRAM can be accomplished in a matter of nanoseconds (a million times faster).

56. Flash memory allowed for relatively fast data access speeds compared to mechanical hard drives, which had spinning platters. The mechanical hard drives move a spindle and spin up a platter to read data. This “seek time” would usually take over ten milliseconds. In comparison, flash memory can be accessed on the order of tens of nanoseconds (a million times faster). Unlike mechanical hard drives, flash memory also has no moving parts, making flash more durable than mechanical hard drives. These speed and durability advantages made flash memory a generally desirable replacement for mechanical hard drives.

57. Memory systems often used layers of progressively faster memory, called “caches.” The “highest” layers of cache usually included cheaper, slower, but greater quantities of memory (despite being shown at the bottom of the pyramid in the figure below). “Lower” layers of cache would use faster, more

expensive, and smaller quantities of memory (despite being shown at the top of the pyramid in the figure below). A system would typically operate with data in the fastest memory available, and the system would use the slower memory only when needed. The following diagram from *Nonvolatile Semiconductor Memory Technology* (Ex. 1021 at 192) illustrates an example of cache hierarchy:

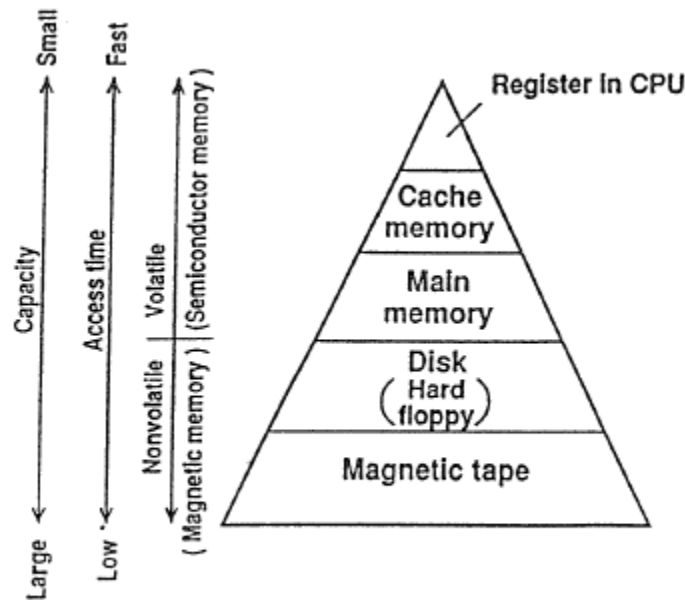


Figure 4.3 Hierarchy of memories in a computer system [4.1].

Id. at Fig. 4.3.

3. Flash Memory Read, Write, and Erase Speeds

58. Flash memory takes significantly longer to write (or “program”) data than to read (or “access”) data. Table 3.2 below, from *Designing with FLASH MEMORY* (Ex. 1022 at 35), provides example read and write times for an Atmel flash memory device in the early 1990s:

Density	1 Mbit
Access Time	90 ns
Data Program Time	150 μ s

Table 3.2: EEPROM Flash Memory Characteristics

Ex. 1022 at Fig. 3.2.

59. Although the read and write speeds have changed over time with designs and fabrication processes, the time needed to program data in flash memory has remained orders of magnitude slower than the time needed to read data. Programming data has usually been at least 3-4 orders of magnitude times slower than reading data in flash memory, if not slower.

60. Flash memory takes even longer to erase a block of data. Table 3.3 below, from *Designing with Flash Memory* (Ex. 1022 at 38), provides example access, program, and block erase times for a Toshiba NAND flash memory device in the early 1990's:

Density	4 Mbit
Access Time	15 μ s (initial read) 80 ns (subsequent serial access)
Data Program Time	4 ms (min)
Block Erase Time (4 kbyte block)	6 ms

Table 3.3: NAND Flash Memory Characteristics

Ex. 1022 at Fig. 3.3.

61. The table shows that a block erase time requires an order of milliseconds, which is several orders of magnitude slower than read speeds that required tens of microseconds or tens of nanoseconds for reads. Although the exact read, write, and erase speeds have changed over time with designs and fabrication processes, the time needed to erase a block in flash memory has remained many orders of magnitude slower than the time needed read data.

4. Flash Memory Erase and Overwrite Processes

62. At the semiconductor operation level, flash memory uses different techniques for writing and for erasing data. *E.g.*, Ex. 1022 at 32-36. In flash memory, data must be written to an *erased* memory location. Standard flash memory “erases” data in a block by changing any bits storing a “0” in that block from a 0 to 1. Data can then be “written” or “programmed” by changing bits that need to store a 0 from a 1 to 0. Ex. 1023 at 533. Data cannot typically be “overwritten” or “rewritten” in an area of flash memory containing existing data because changing any bits in a block from 0 to 1 would require the “erasing” technique. Sometimes, literature may loosely refer to “writing” all ones or all zeros to erase flash memory. The flash memory may use the write technique to set all data to zeros, thereby making the data appear erased, but when responding to a

command to “write” all ones, the flash memory will actually use the “erase” technique.

63. If data in a specific location of flash memory is to be changed then: 1) the new data must be stored someplace else temporarily, 2) the existing data in that location of flash memory must be erased, and finally 3) the new data is programmed into the erased memory location of flash memory. Of these 3 steps, the erase process requires the longest time, which is undesirable in high-speed digital systems.

5. Garbage Collection

64. “Garbage collection” has been a “key algorithm” for managing data on flash memory since at least 1995. Ex. 1023 at 531. This is because obsolete/invalid data (“garbage”) must be erased before new data can be written in its place, as discussed in preceding section 4. *Id.* at 531, 537-40. The garbage collection process usually performs this erasure in the background but may also actively run to free up space. *Id.* A block may contain garbage mixed with valid data, so the valid data must be copied and preserved elsewhere while the entire block is erased. *Id.* The following figure illustrates an example garbage collection on cluster 3. As shown below, cluster 3 has a single valid sector. In the garbage collection process, this valid cluster is copied to cluster 4 followed by erasing

cluster 3 so that it contains only erased sectors and is available for future use. The valid data is preserved in cluster 4. The invalid data (“garbage”) in cluster 3 was “collected” for erasure.

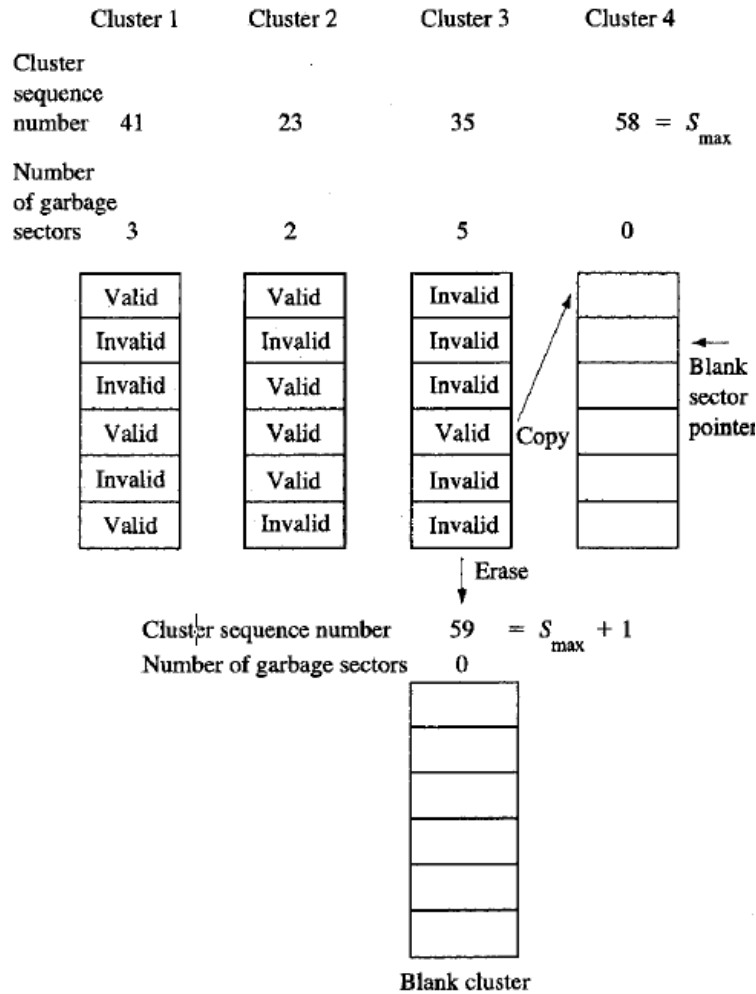


Figure 2

Garbage collection mechanism.

Ex. 1023 at 537. As part of this process, the valid data was copied to cluster 4 (“clusters” refer to blocks, *id.* at 538). So that a computer system may continue to find the valid data originally stored in cluster/block 3 after the data has been moved during garbage collection, a “logical” address used by a computer to find this valid data will be updated to indicate that the valid data has been moved to cluster/block 4. *Id.* at 537-39; Ex. 1007 at § 2.3. I explain this address mapping process more in section 7, “Flash Translation Layer,” below.

65. Garbage collection greatly improves the on-demand operational speed of flash memory devices. This is because garbage collection can conduct the relatively slower erasures during a background process. That erased memory can then be written to on-demand at relatively fast speeds. Without garbage collection, an area of memory might first need to be erased before writing, taking orders of magnitude longer.

66. By the priority date, garbage collection had become a routine and conventional part of flash memory systems. Ex. 1007 at § 2.3, reference [18]; *see also* Ex. 1022 at 259-62 (explaining “background clean-up” process as of 1993); Ex. 1024 at 1:39-42 (“Garbage collection can be performed in conventional computing systems using a number of well-known techniques” as of 2002).

6. Endurance and Wear Leveling

67. Flash memory can only be erased a limited number of times. This is referred to as the “wear” or “endurance” of the memory. Because flash memory must be erased before writing, the endurance effectively limits the number of write cycles as well. The endurance of flash memory is a serious issue that affects the design and implementation of flash memory systems. Ex. 1023 at 531, 534-37.

68. To prevent one area of memory from hitting the endurance limit faster than other areas, various wear-leveling techniques can be used. Ex. 1007. These techniques make sure that the same physical sectors are not erased and written each time there is new data to store. This is achieved by changing the physical location of where data is erased/written for any given logical address. *Id.* That data can remain addressed by the same logical address (sometimes referred to as a virtual address) while being assigned to different physical locations in the flash memory. *Id.* The physical locations are identified by physical addresses. In order to implement wear-leveling techniques, a flash translation layer must be used.

7. Flash Translation Layer: Logical to Physical Addressing

69. The Flash Translation Layer (FTL) allows for the mapping (also referred to as “translating,” “indexing,” or “converting”) of logical addresses to physical addresses and is implemented on the flash device itself. Ex. 1007 at §§ 2-

2.2. Logical addresses are the addresses used by computer operating systems to identify certain data. Physical addresses indicate a physical storage location of the data. In platter hard drives, physical addresses would indicate a cylinder, sector, and/or head of a platter where data is stored. In flash drives, physical addresses indicate the block (and also usually pages) of flash memory where data is stored.

70. The FTL was patented by Ban in 1995 in U.S. Pat. No. 5,404,485 titled “Flash file system.” The FTL was adopted as part of the PCMCIA standard. Ex. 1007 at § 2.2. It is used in conjunction with both wear leveling and garbage collection. Ex. 1023 at 535-39.

71. An FTL can map the logical to physical addresses using various data structures, such as an “index,” “table,” or other similar structure. *E.g.*, Ex. 1002 at Figs. 2, 3B, 8B; Ex. 1003 at Figs. 1, 7; Ex. 1007 at § 2.2; Ex. 1023 at Fig. 1. The data structure must be preserved on nonvolatile memory in case of power loss. Ex. 1023 at 535; Ex. 1007 at § 2.2; Ex. 1003 at 8:3-17. The copy of the FTL table preserved on nonvolatile memory may be a direct copy (Ex. 1022 at 246), an inverse/reverse table (Ex. 1023 at 535, Ex. 1007 at § 2.2), or some other structure that allows the FTL table to be reconstructed. During powered operation, the FTL data structure maintaining the relationship between logical and physical addresses

may be reconstructed in volatile memory, such as SRAM or DRAM, for faster access and updating. Ex. 1023 at 535; Ex. 1007 at §§ 2-2.2.

72. The FTL mapping between logical and physical identifiers is updated in response to changes in the data. Ex. 1007 at § 2.3 (“data structures that map logical blocks to sectors are updated if necessary, to reflect the relocation”). For example, if the physical storage location of data is changed (such as due to wear leveling or garbage collection as explained in § VI(A)(5)-(6), *supra*), then the logical-to-physical address mapping for that data will be updated to reflect the new physical storage location. Similarly, if data is erased altogether, then the corresponding logical-to-physical entry may either be deleted (typically for any mapping is stored in RAM, where erasures are easily performed) or marked as invalid/erased (more common if the mapping is stored in flash, where erasures are relatively slow). By using an FTL, an operating system can track the location of data using only the logical address. This simplifies the management by the operating system, which can remain oblivious to where data is physically stored. By using FTLs and operating on logical addresses only, operating systems do not need to know how each and every unique storage device uniquely stores and uniquely updates data in different physical locations—such tracking is offloaded to the FTL.

8. Interaction Between Flash Memory and File Systems

73. Users interact with a computer system through a user interface of an application or a user interface of an operating system. The users may issue commands to read, write, or delete data such as documents, pictures, and the like. In relevant systems, the data resides on flash memory accessed by the computer system.

74. The picture below shows a simplified view of layers of components in an interaction between an operating system and the underlying flash memory. Ex. 1022 at 230. The operating system (e.g., Windows, MS-DOS) uses a flash file system to communicate with the underlying flash memory device. Ex. 1022 at Ch. 9-10; Ex. 1007 at § 2.

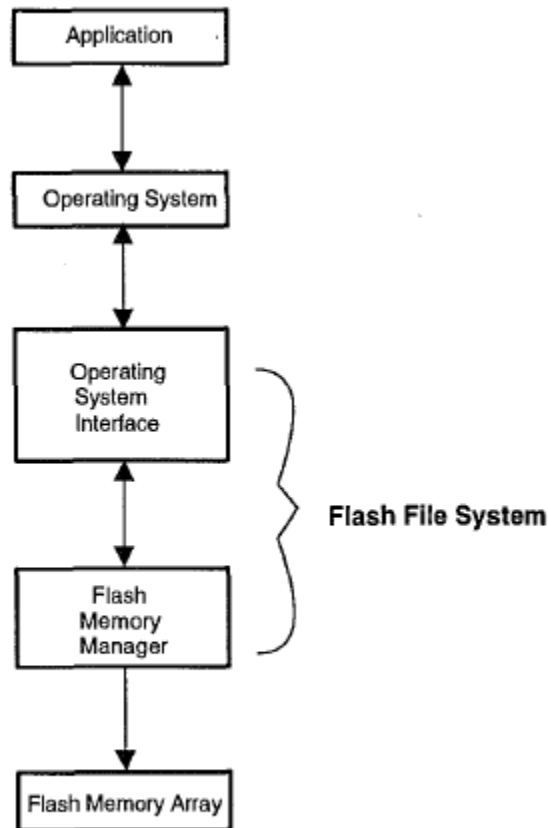


Figure 9.1: Flash Memory Manager and Operating System Interface

Ex. 1022 at 230.

75. When converting operating system commands into the specific signals suitable for a particular type of flash memory device, drivers work with the file system to translate logical addresses to generate the specific electrical signals for communicating with the flash drive. *E.g.*, Ex. 1022 at 228, 237, 247-49; Ex. 1007 at § 2. Earlier computer systems treated flash memory as block devices to allow compatibility between flash devices and standard file systems, such as a File

Allocation Table (FAT). Ex. 1007 at § 2. Computer systems also used other file systems for managing flash devices. *Id.*; Ex. 1022 at Ch. 9-10.

76. Figure 9.1, reproduced again below on the top left, imprecisely shows the “Flash File System” as separate from the “Flash Memory Array.” Ex. 1022 at 230. In actuality, file system data is stored on the flash disk and also stored in a computer system’s RAM, such as more accurately shown in Figs. 9.10 and 9.11, reproduced below. *Id.* at 245-46. For example, Fig. 9.10 below shows a FAT stored in the flash memory array. *Id.* at 245.

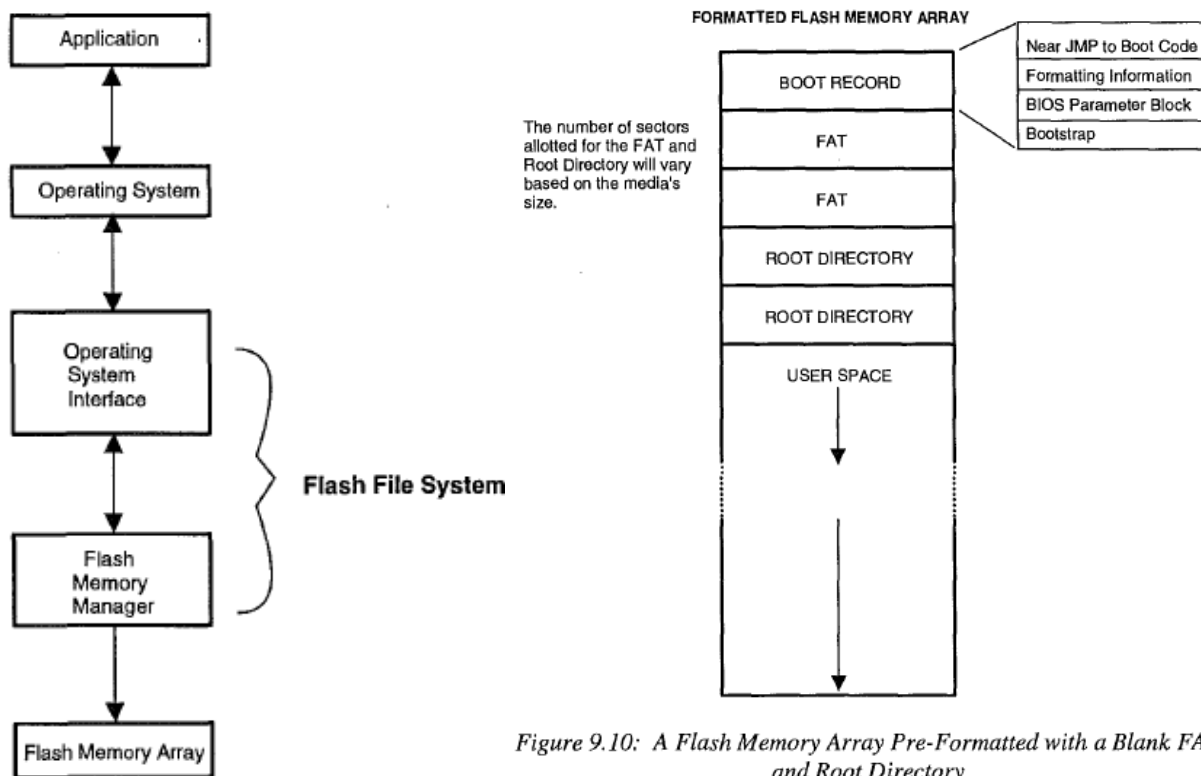


Figure 9.10: A Flash Memory Array Pre-Formatted with a Blank FAT and Root Directory

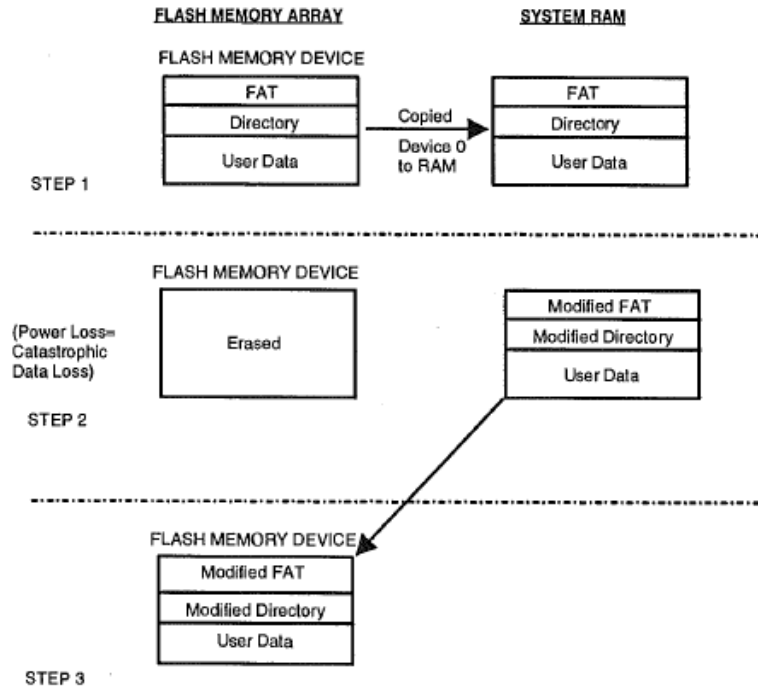


Figure 9.11: Sector-Level Modification Requires Considerable Overhead

Id. at 230, 245, 246.

77. The system may cache a copy of the file system data in system RAM (a faster, volatile memory that loses data when powered off) for faster access. See § VI(A)(3), *supra* (explaining caching). As pictured above in Fig. 9.11, modifications to the file system data will be pushed through to the underlying flash memory (a nonvolatile memory that does not lose data when powered off) in order to avoid catastrophic data loss during unexpected power-off events. Ex. 1022 at 246. The principle of protecting against power loss applies generally to different file systems (and generally to any data). Ex. 1022 at 246; Ex. 1007 at § 2.2. For

example, SwSTE'05 discusses a file system that uses inverse maps on the flash memory and direct maps stored at least partially in RAM. Ex. 1007 at § 2.2.

78. Given this background, the following summarizes a chain of events when a user reads, writes, or deletes data in a computer system with flash memory. A user interacts, through a user interface, with an application or an operating system to modify certain data. For example, the user may specify a file name (e.g., "C:\document.doc") to be read/written/deleted. Operating systems then access a "file" that translates the file name into another logical address (e.g., a file indicating that "C:\document.doc" is stored at logical address 0x4000). The operating system then generates a command to read, write, or delete data stored at a logical identifier (e.g., at the logical address, or at the file name). The driver and the file system then work together to convert the operating system's command into the specific electrical signals understood by the flash memory device in order to read, write, or delete data on the flash drive. As part of this process, some file system data in RAM may be modified, and a modification to corresponding file system data on the flash drive will be performed. For example, when sending erase commands, the computer updates its own system "file" in RAM to indicate that C:\document.doc is erased and no longer stored at logical address 0x4000,

such as by erasing this file entry. The flash device that receives the command may then operate as described by Bennett or Suda in Sections VI(B) and VI(C) below.

9. Industry Standards

79. The industry adopted a number of standards affecting flash memory devices. Standards allow different industry participants to make devices operate with devices of other participants. As a result, a person of ordinary skill in the art would have been aware of standards affecting flash memory.

80. Many flash devices used the Advance Technology Attachment (ATA) interface. Ex. 1007 at § 2. The ATA standard is set by the Technical Committee 13 (T13) of the InterNational Committee on Information Technology Standards (INCITS), which operates under the American National Standards Institute (ANSI). T13 is open to the public through its website T13.org, and it is free to access its proposals, minutes, and meetings. Final standards are published and purchasable through ANSI. The minutes show that the Trim command was first proposed by Frank Shu on April 6, 2007. Ex. 1011. Frank followed up with several proposal revisions, including revision 1 on June 26, 2007 and revision 2 on September 5, 2007. Exs. 1012, 1013. These proposals are accessible through T13.org's document archive. The Trim command, after further proposal revisions, would ultimately be adopted into the ATA standard.

81. Before Frank Shu’s proposal, the ANSI published the ATA/ATAPI-6 version of this standard in 2002 and the ATA/ATAPI-7 version of this standard in 2005. This interface was used as the command set in the Serial ATA (SATA) Revision 2.5 standard. Ex. 1020 at 19. The SATA 2.5 standard enjoyed widespread adoption throughout the computer industry. Ex. 1020 at 1 (listing major companies as board members, including Dell, Hewlett-Packard, Hitachi, Intel, Maxtor, Seagate, and Vitesse). The inventors even admit that SATA was “ubiquitous.” Ex. 1009 at claim 29. The SATA 2.5 standard defines a high-speed interface to ease integration and enable scalable performance with data rates of 1.5 Gbps and 3.0 Gbps. Ex. 1020 at 19.

82. These standards contain a set of commands known as the CompactFlash Association (CFA) Feature set. This feature set provides support for solid state memory devices. Ex. 1005 at § 6.16, Ex. 1019 at § 4.10.

83. The CFA Feature set includes the command “CFA Erase Sectors.” The CFA Erase Sectors command preconditions the sector for a subsequent CFA Write Sectors Without Erase or CFA Write Multiple Without Erase command to achieve higher performance. Ex. 1005 at §§ 6.16, 8.1; Ex. 1019 at §§ 4.10, 6.2. How a device implements preconditioning in response to this command is left to the designers of flash devices. The ATA/ATAPI standard does not specify what

the device must do in response to a CFA Erase Sectors command. In fact, supporting the CFA Feature Set is optional, and a device may ignore these commands. Ex. 1005 at § 6.16, 8.1; Ex. 1019 at §§ 4.10, 6.2. As described in Sections VI(B) and VI(C) below, Bennett and Suda provide examples of how a flash drive may respond to the CFA Erase Sectors command.

84. The CFA Erase Sectors command specifies a logical block address identifying the starting sector to be erased as well as a sector count to be erased.

The format of the CFA Erase Sectors Command is as follows:

8.1.4 Inputs

The LBA High, LBA Mid, LBA Low, and Device registers specify the starting sector address to be erased. The Sector Count register specifies the number of sectors to be erased.

Register	7	6	5	4	3	2	1	0
Features	na							
Sector Count	Sector count							
LBA Low	LBA (7:0)							
LBA Mid	LBA (15:8)							
LBA High	LBA (23:16)							
Device	obs	LBA	obs	DEV	LBA (27:24)			
Command	C0h							

Sector Count –
number of sectors to be erased. A value of 00h specifies that 256 sectors are to be erased.

Ex. 1005 at § 8.1.4.

B. Overview of Bennett

85. The USPTO issued U.S. Patent No. 7,624,239 entitled “Methods for the Management of Erase Operations in Non-Volatile Memories” on November 24,

2009 to inventors Alan David Bennett, Alan Douglas Bryce, and Sergey Anatolievich Gorobets (referred to as “Bennett” in this overview). Ex. 1002.

86. Bennett teaches improvements for managing the time required for the erase process, discussed above, when data in a specific address location is changed. Bennett teaches that the logical address from a host is mapped to a physical address in a flash memory system. *Id.* at Fig. 3B. When data stored in a specific address used by the host (logical address) is changed, the new data is written to a different physical location and then the logical address is pointed to this new physical location. This eliminates the need for an erase operation when data in a specific address location is changed but leaves behind a physical block that is no longer needed and that can be erased and reused at some point.

87. Specifically, Bennett teaches using a logical group (LG) of sectors of flash memory that are mapped using a directory to a physical group, also called a “MetaBlock” or “MB” of sectors of flash memory. Any logical group can be mapped to any metablock and over time, logical groups may be mapped to different metablocks as data is written and erased by the system. This way if data in a specific logical address is changed then instead of waiting on the corresponding physical block to be erased, so the new data can be written to the same physical location, the logical address is remapped to a different physical

block address. This remapping of the logical address to a new physical address avoids the slow erase process impeding the saving of the new data in the same physical location. The new data is saved in a different, erased, physical block and the logical to physical address directory is updated to associate the same logical address with the new physical address and location. The older, unneeded, physical block can then be erased during the garbage collection process when the flash memory is more inactive or when an erase command is received as discussed below.

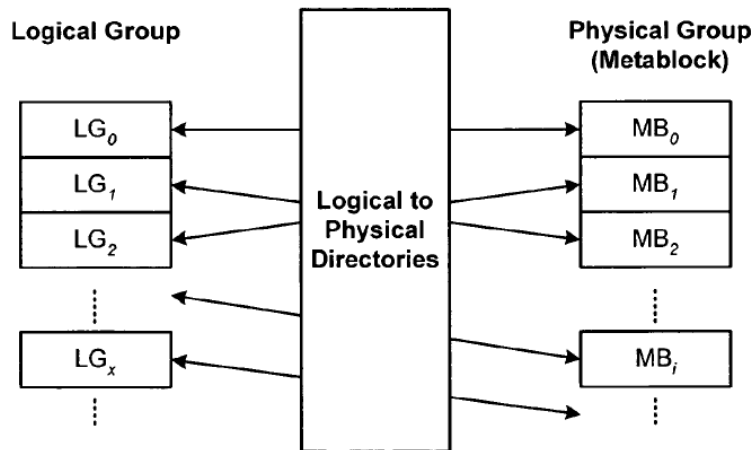


FIG. 3B

Ex. 1002 at Fig. 3B.

88. Fig. 2 of Bennett shows flash memory, 200, being organized into physical groups of sectors (again, MetaBlocks, MB) $S_0 \dots S_{N-1}$ that are erasable together. The host, 10, addresses data in memory, 200, with a logical address. The

memory-side memory manager, 100, keeps track of the logical to physical address translation as well as which blocks should be erased, updated, and linked. When an erase command is received from the host the specified memory locations are either physically erased (the actual transistors implementing flash memory on the chip are erased) or logically erased (the memory is marked for physical erase at a later time). Bennett teaches logically erasing partial groups of sectors while physically erasing full groups of sectors (MetaBlock) in order to improve the performance of the memory system.

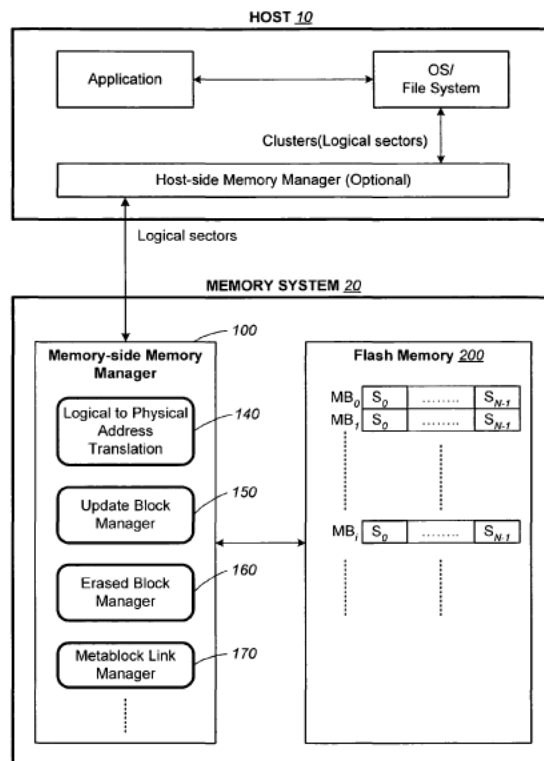


FIG. 2

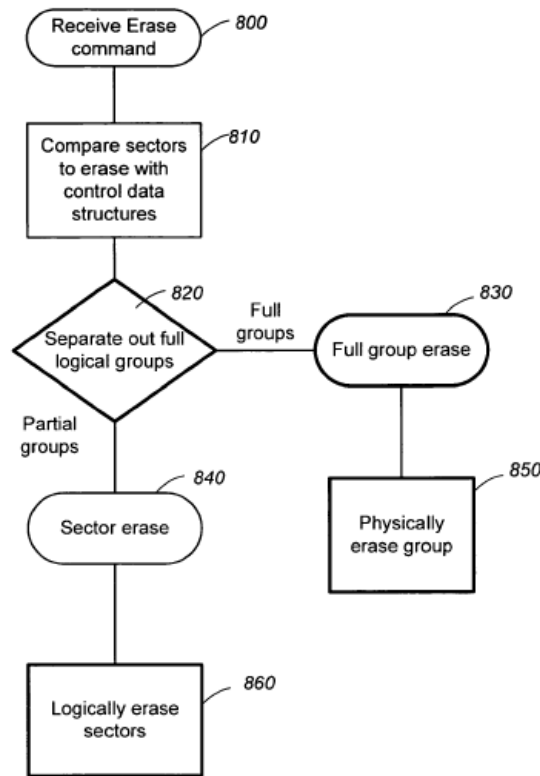
Ex. 1002 at Fig. 2.

89. Fig. 10 shows a flowchart detailing an erase management method taught in Bennett. When an erase command is received, 800, the specified sectors to be erased are compared to the memory system's control data, 810. The control data provides information on how the flash memory is used such as which sectors have been logically erased, physically erased, or contain data. If the sectors that the erase command specifies, either alone or with the already logically marked for erase sectors specified by the control data, indicate a full group of memory (MB₀, MB₁, etc. in Fig. 2) to be erased, then a physical erase is performed on the associated full group. If a partial group of memory is specified in the erase command and not made full by the logically erased sectors specified by the control data, then the associated memory is logically erased. Logically erase memory will be physically erased when either another erase command is received resulting in a full group of memory, as just discussed, or during garbage collection where the physical erase takes place when the flash memory is inactive.

90. As an example, assume that starting from sector number 0, every 100 sectors are designated as a logical group (which are mapped to a 100 sector metablock of physical sectors). So, sectors from 0-99 are the first logical group, sectors from 100-199 are the second logical group and so on. If an erase command specifies logical sectors 10-290 should be erased, and the control data indicates no

other sectors have been logically erased, then sectors 100-199 would be *physically* erased. But sectors 10-99 and 200-290, as each is less than one complete logical group, would only be *logically* erased.

91. For a second example, again assume that every 100 sectors are designated as a logical group (which are mapped to a 100 sector metablock of physical sectors). If an erase command specifies logical sectors 110-220 should be erased, and the control data indicates that sectors 77-109 have been logically erased then sectors 100-199 would be *physically* erased (because sectors 100-109 were already logically erased and now sectors 110-199 are joining them as logically erased, which makes the entire logical group of 100-199 logically erased). Sectors 77-99 would remain *logically* erased (because they were already logically erased, yet form only a portion of the 0-99 logical group) and sectors 200-220 would become *logically* erased (because they were not previously and they are only a portion of the 200-299 logical group). Note that if the control data, in this example, indicated no sectors had previously been logically erased, then sectors 110-220 would all be only logically erased (and not physically erased) because sectors 110-220 don't include any complete logical group (i.e., 110-220 are only subparts of the 100-199 and the 200-299 groups).



Multi-Mode Erase Process

FIG. 10

Ex. 1002 at Fig. 10.

92. Fig. 6 is a block diagram of the MetaBlock management system implemented in the controller and flash memory. This system is made up of the controller, 100, in Fig. 2 containing RAM, 130, and flash memory, 200. The RAM stores control data that is frequently changing or accessed including the allocation block list (ABL) which keeps track of the MBs that have been used and the cleared block list (CBL) which indicates which MBs are currently erased. Control data is

also stored in the nonvolatile flash memory including the group address tables (GAT, a list relating logical groups of sectors to the corresponding MB), chaotic block indices (CBI, a list of the mapping of logically non-sequential sectors during an update), erased block lists (EBL, a list of the MBs that have been erased), and MAP (a dedicated MB containing a bitmap indicating the erase status of all MetaBlocks in the flash memory). This approach, in the event of a power shutdown or failure, allows the control data in the volatile RAM to be rebuilt quickly by scanning a small set of control data in the nonvolatile flash memory.

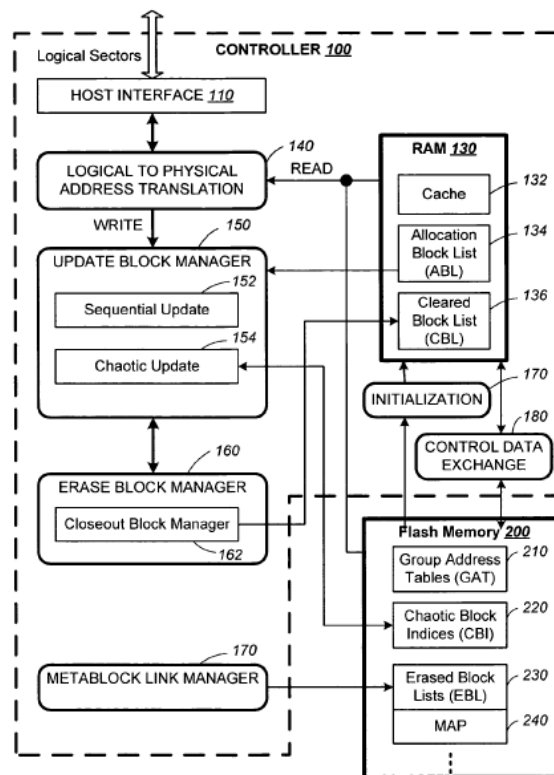


FIG. 6

Ex. 1002 at Fig. 6.

93. As an example of how information is stored in one of these lists/tables Fig. 8A shows the data fields in the sector of nonvolatile flash memory used to store the GAT. Each GAT sector includes two components: 1) entries for MB addresses corresponding to each logical group and 2) the GAT sector index. The former contains information for locating a MB corresponding to a specific logical address. The later indicates the location of the sectors within a specific MB and contains three fields: the MB number, the page tag (e.g., indicating if the sectors are offset, e.g. a MB may be addressed, for example, from 66-99 and 0-65 instead of from 0-99 because of an offset of 66), and a flag indicating if the MB has been relinked (redefined using groups of sectors that form a minimum erasable unit from various locations to form a new MB, Fig. 4).

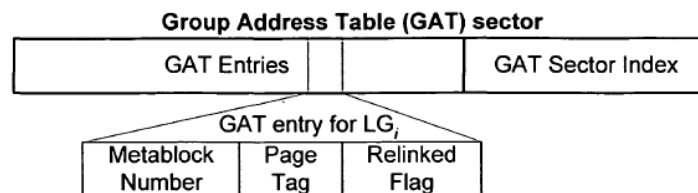


FIG. 8A

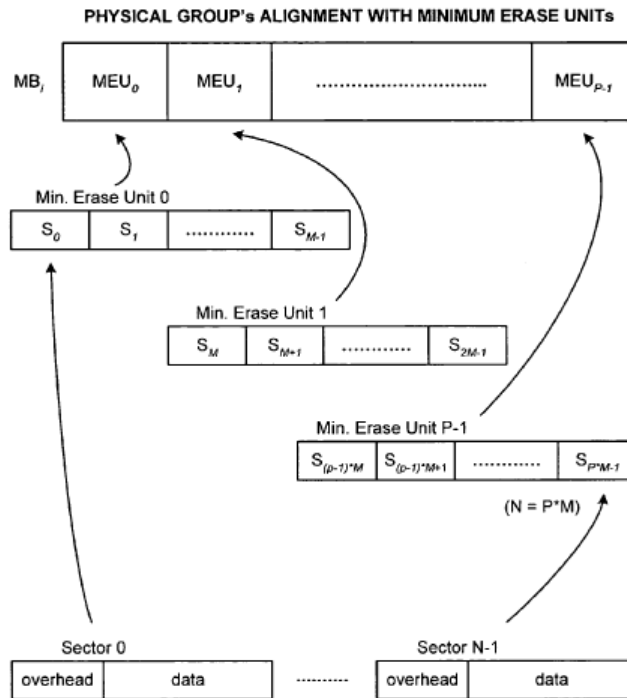


FIG. 4

Ex. 1002 at Figs. 8A, 4.

C. Overview of Suda

94. The USPTO issued U.S. Patent 7,057,942 entitled “Memory Management Device and Memory Device” on June 6, 2006, to inventors Takaya Suda and Hiroaki Muraoka (referred to as “Suda” in this overview). Ex. 1003.

95. Suda describes a memory management device for managing a nonvolatile semiconductor, such as flash memory, on a memory device such as the memory card, 1, seen in Fig. 1 of Suda. The memory card in Suda is connected to a host device, 2, such as a digital camera. The memory card has a host interface,

12, flash memory controlling section, 11, flash memory, 14, and a random access memory (RAM), 13. The flash memory is made up of physical blocks of flash memory so that the data stored in a block may be erased without changing the data in other blocks.

96. Commands and data are sent to the memory card through command and data lines via the host interface of the memory card. The flash memory controlling section manages the erasing of data and a table, stored in RAM 13a, indicating the relationship between logical and physical blocks in the flash memory. This is how Suda implements the flash translation layer. *See* § VI(A)(7), *supra* (providing background on the FTL). The logical address is the address used by the host device to read or write information stored in flash memory. An unused physical block, one that is erased and thus has initial-value data written to it, can be used when it is associated with a logical block address.

97. The logical to physical association can change over time (for example, as result of wear-leveling or garbage collection described above in §§ VI(A)(5)-(6), *supra*). For example, Ex. 1003 Figs. 2 and 10 show changing physical address assignments. As another example, the host device may read data located in logical address 483. This logical address may correspond to a physical location (address)

on the flash memory, for example, physical address 29. Over time, the data stored in physical location 29 may be moved to a new location, say physical location 428.

98. The information stored in the logical and physical address table portion of RAM, that is 13a, correspondingly changes so that the logical address used by the host, in this example logical address 483, now points to the physical address 428 instead of physical address 29. This logical-to-physical mapping is unknown to the host and allows the memory to be useful even when parts of the memory are failing. Also stored in the RAM, 13, of the memory card is an erasure area pointer storage area 13b, pointing to the blocks of flash memory to be erased as indicated by an erase command from the host device, 2.

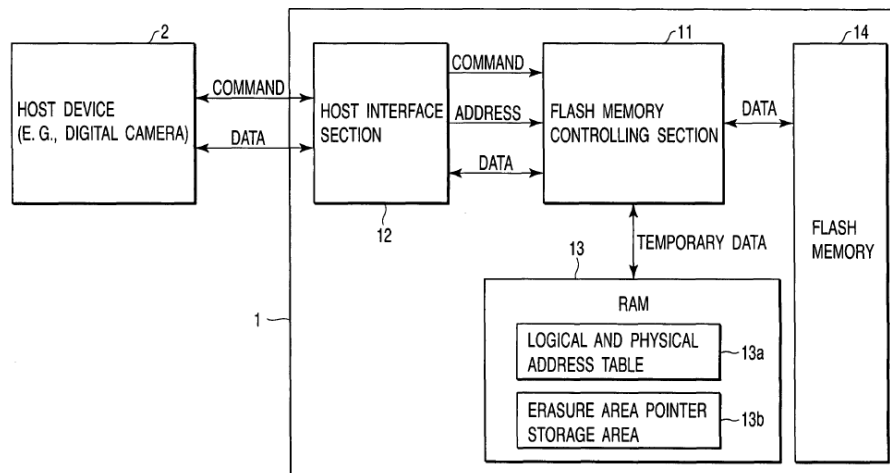


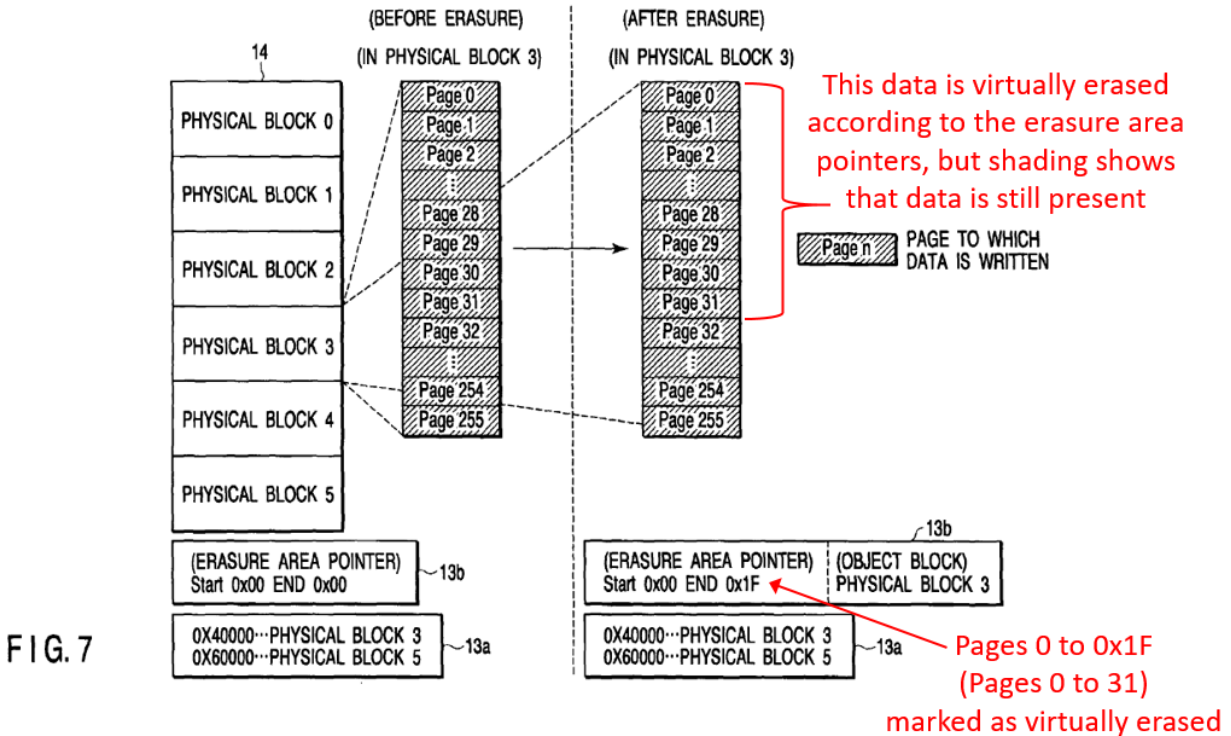
FIG. 1

Ex. 1003 at Fig. 1.

99. The memory management device taught by Suda includes the blocks of flash memory in 14, and the flash memory controlling section 11. The flash memory controlling section 11 includes a setting unit that is used to set an address range of data to be erased when an erase command specifying a logical address is received from the host device. The flash memory controlling section 11 also processes read commands by either returning the requested data or instead outputting “initial-value” data if the requested data is marked as virtually erased. “Initial-value” refers to the original data values initially set in the memory device before ever using the device. These initial values are the erased state (usually filled with all 1’s).

100. Suda addresses the slow erase speeds, which is explained with the help of Fig. 2, copied below. In this figure, the host device issues an erase command to erase the 32 pages, page 0 to page 31 on the left side of Fig. 2, in physical block 3. Some valid data in pages 32 to 255, that is, 224 pages within the physical block 3, is to be retained, that is, not erased. In flash memory, a single page of data cannot be erased by itself. Rather, erases occur on physical blocks of flash memory, which include many pages. In Fig. 2, this means that all 256 pages within physical block 3 are erased (set to their initial, erased value, typically “1” for NAND flash) at the same time. This whole block erasure makes it appear to

the host that the data in pages 0-31 is erased. But to preserve the valid data in pages 32-255, pages 32-255 are copied/moved to an erased physical block 4 before the erasure of physical block 3. In physical block 4, pages 0-31 are not written to and thus contain the initial, erased value and can be programmed to store new data. Then, the logical address used by the host corresponding to physical block 3 is changed in the logical and physical address table to point to physical block 4. From the host's point of view, the same logical address is used for the retained data in pages 32-255 and the host can remain oblivious to the underlying physical change. All of block 3 is erased (block 3 is not illustrated in the "after erasure" side) can then be programmed to store new data. Pages 0-31 in block 4 remain erased and can also be programmed to store new data. The problem with this approach, however, is that it takes a considerable amount of time to write the data stored in the 224 pages in physical block 3 to physical block 4 and to erase block 3. Suda looks to shorten the time to make this transfer by using erasure area pointer or pointers.



Ex. 1003 at Fig. 7 (annotated).

101. The erasure area pointer or pointers are information stored in the erasure area pointer storage area 13b shown in Fig. 1. These pointers can also indicate the range of page addresses assigned to successive pages in a physical block of flash memory. The flash memory controlling section 11 determines the pages in the address range indicated by the erasure area pointer or pointers are in a virtual erased state. The virtual erased state is a state in which the flash memory controlling section 11 does not actually erase data items to be erased. Virtually erased data is subjected to virtual erasure in response to an access (e.g., read) command from the host device 2, meaning that if a host attempts to read virtually

erased data, the memory device will respond as though the virtually erased data is empty. The erasure area pointer or pointers identify the virtually erased part of each physical block in the flash memory 14. Fig. 3 shows an example of the erasure area pointers. A start pointer and an end pointer are set as the erasure area pointers. Suda also teaches pointing to entire blocks of memory and to points within a block that indicate only a start or a stopping point for multiple block erases.

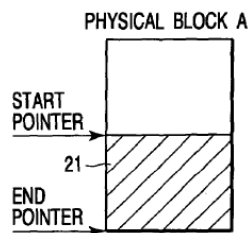
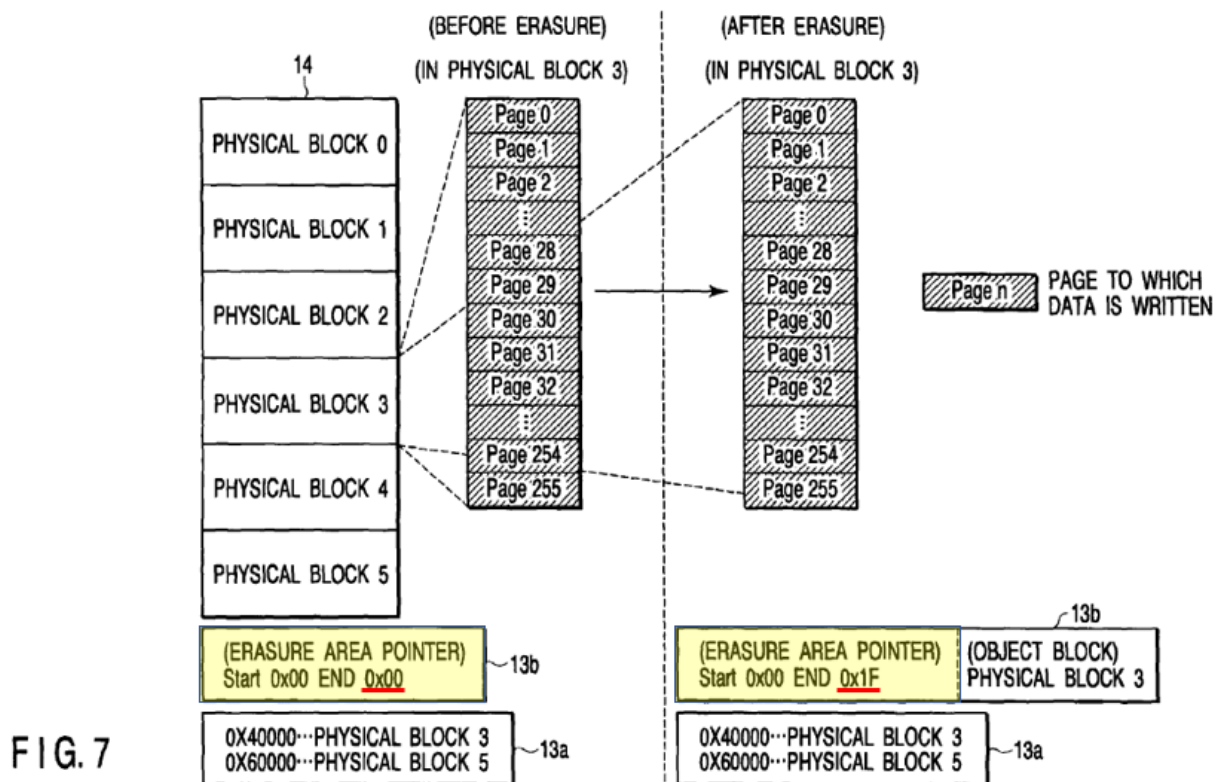


FIG. 3

Ex. 1003 at Fig. 3.

102. Using the erase pointer or pointers then, Fig. 7 of Suda teaches a solution to the long transfer time discussed above with relation in Fig. 2. When an erase command for pages 0-31 is received from the host, instead of transferring the remaining pages 32-255 to erased physical block 4, the erase pointers are updated in 13b of the RAM to indicate that that pages 0-31 of block 3 are virtually erased. Pages 32-255 remain in physical block 3 and thus no time is needed to transfer this information or to erase physical block 3.

103. Update of the pointers is shown below highlighted in Fig. 7. On the left side of the figure, the start and end erase pointers both point to the same address 0x00 (hexadecimal 0 and decimal 0), which means no erased memory (start and stop are pointing to the same location) is in the block. On the right side of Fig. 7 the erase pointer start is pointing to 0x00 (hexadecimal 0 or decimal 0) and the erase pointer end is point to 0x1F (hexadecimal 1F or decimal 31). Again, pages 32-255 remain in physical block 3 after receiving the erase command (for pages 0-31 in physical block 3) from the host and thus no time is needed to transfer information.



Ex. 1003 at Fig. 7 (annotated).

104. Fig. 8 of Suda is a flowchart that details how data is erased on the memory card, 1. An erase command is issued from the host in step S1. This causes the flash memory controlling section, 11, to use the logical and physical address table 13a in RAM to determine the physical address corresponding to the host device's logical address (step S2). Next, the flash memory controlling section determines if the addressed range corresponds to an area where the data to be erased falls within an erasure area pointer range of addresses (step S3). If yes, then flash memory controlling section outputs a control signal to the host device indicating that data erasure processing is completed (step S8). If no, then the erasure area pointer is updated in pointer storage area RAM 13b (step S4). Then the flash memory controlling section determines whether the address range indicated by the erasure area pointer in step S4 is the size of a physical block. If so, then the block is erased (step S6) and the flash memory controlling section outputs a control signal to the host device indicating that data erasure processing is completed (step S8). If not, then the flash memory controlling section writes the erasure area pointer storage area information in 13b (updated in step S4) to the flash memory 14 (step S7) and then the flash memory controlling section outputs a

control signal to the host device indicating that data erasure processing is completed (step S8).

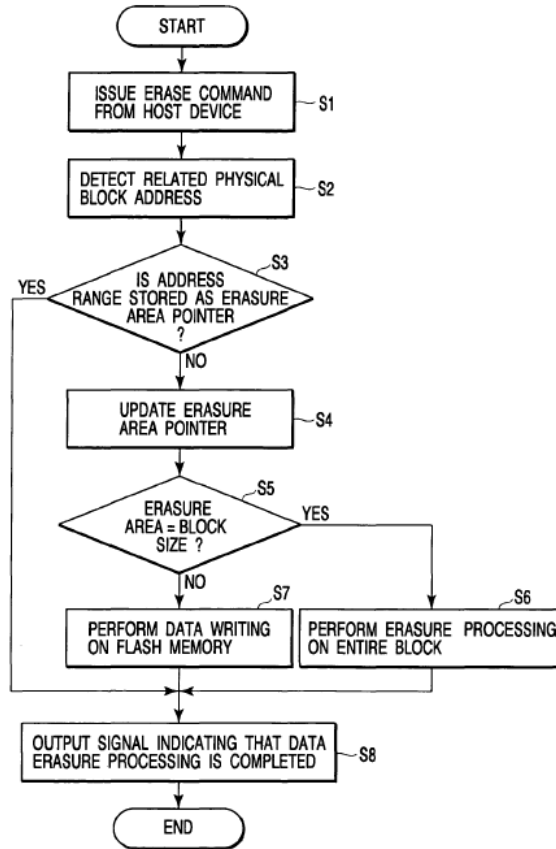


FIG. 8

Ex. 1003 at Fig. 8.

105. After step S7 is completed even if power to the memory card is accidentally lost or purposely turned off, the erasure area pointer information is maintained and thus a virtual erased state is also maintained. When the memory card is powered back up the flash memory controlling section can read the

information stored in the flash memory in step S7 and then write this information back to the erasure area pointer storage area in table 13b in RAM.

D. Overview of Zipprich

106. The USPTO published U.S. Patent Publication 2003/0079078 entitled “Confirmation of Secure Data File Erasure” on April 24, 2003, to inventors Andrew Zipprich, Bruce Talbert, and Keith Bunker (referred to as “Zipprich” in this overview). Ex. 1006.

107. Zipprich describes a process for providing a status report after data is erased. Zipprich teaches that the status report can be generated in the form of a log entry, email, or hard copy. *Id.* At Fig. 7. The status report tracks events within an overwrite process and provides feedback indicating success or failure.

E. Overview of SwSTE’05

108. SwSTE’05 was published in 2005 by Eran Gal and Sivan Toledo. Ex. 1007.

109. SwSTE’05 provides two relevant disclosures. First, SwSTE’05 discloses that memory devices used an FTL to remap the same virtual block number (a logical address) to different physical sectors (physical addresses) to implement wear-leveling by distributing the writes/erases in different physical

locations. *Id.*, §§ 2-2.1. Second, SwSTE'05 discloses that a sector is reclaimed using a “garbage collection” process to make more space available. *Id.*, § 2.3.

VII. OBVIOUSNESS OF THE CLAIMS BASED ON SUDA

1. Claim 7

- a. Preamble: “A method for managing data in a system having a host processor and a block-based NAND flash storage system, the NAND flash storage system having physical NAND flash storage locations controlled by a NAND controller, the method comprising:”*

110. I have been asked by counsel to assume the preamble is a limitation of the claim. Suda teaches this preamble.

111. Specifically, Suda discloses “managing a nonvolatile semiconductor memory which comprises a plurality of blocks” (the claimed “method for managing data in a system having... a block-based flash storage system”). Ex. 1003, Abstract. The flash memory 14 is provided as, e.g., a NAND type nonvolatile memory.” Ex. 1003 at 2:63-66. Figure 1 displays it below:

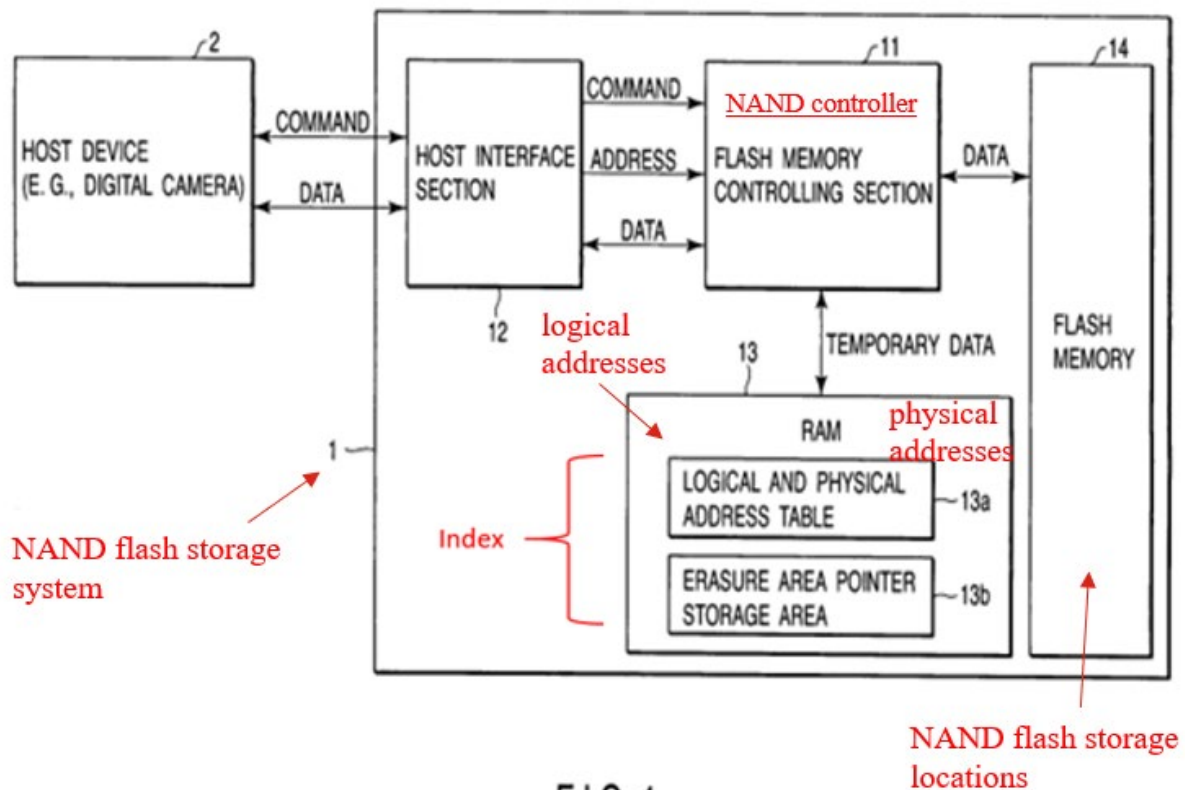


FIG. 1

Id. at Fig. 1 (annotated). Figure 1 shows a NAND flash storage system (1) which has physical NAND flash storage locations (14) controlled by a NAND controller (11). A POSITA would have understood that a host device (2) has a host processor. Thus, a POSITA would have understood that Suda teaches this claim element.

b. “receiving, via the host processor, an indication of deletion of a file at the host processor;”

112. A POSITA would have understood that Suda teaches this claim element. For example, Suda provides the example with a digital camera that

allows the user to delete data, such as unwanted photos. Ex. 1003 at Fig. 1, 2:61-62. In my background section, I showed Figure 9.11 as an example of a system caching file data. See § VI(A)(8), *supra*. A POSITA would have known that the digital camera keeps, in cache memory, data about which logical identifiers are associated with which photos. When the user deletes a photo, the digital camera will reference the cache for the corresponding logical identifier and then designates this logical identifier to be erased in the erase command. Ex. 1003 at 8:66-9:3. The system would delete the cached entry and its corresponding logical identifier, which frees up cache memory and makes the logical identifier available to assigned to a different physical address. A POSITA would have understood this process to have been performed by the digital camera's host processor. Thus, a POSITA would have understood that Suda discloses "receiving, via the host processor, an indication of deletion of a file at the host processor."

c. "creating, via the host processor, an empty-block identifier in response to the indication of deletion;"

113. A POSITA would have understood that Suda teaches this claim element. A POSITA would have understood that when the camera's host processor issues an erase command, the identified logical address would have been deleted from the cache memory of the digital camera. Further, Suda teaches a logical and physical address table that maps logical addresses to corresponding

physical block addresses. Ex. 1003, 3:41-55. Thus, a POSITA would have understood that Suda teaches “creating, via the host processor, an empty-block identifier in response to the indication of deletion.”

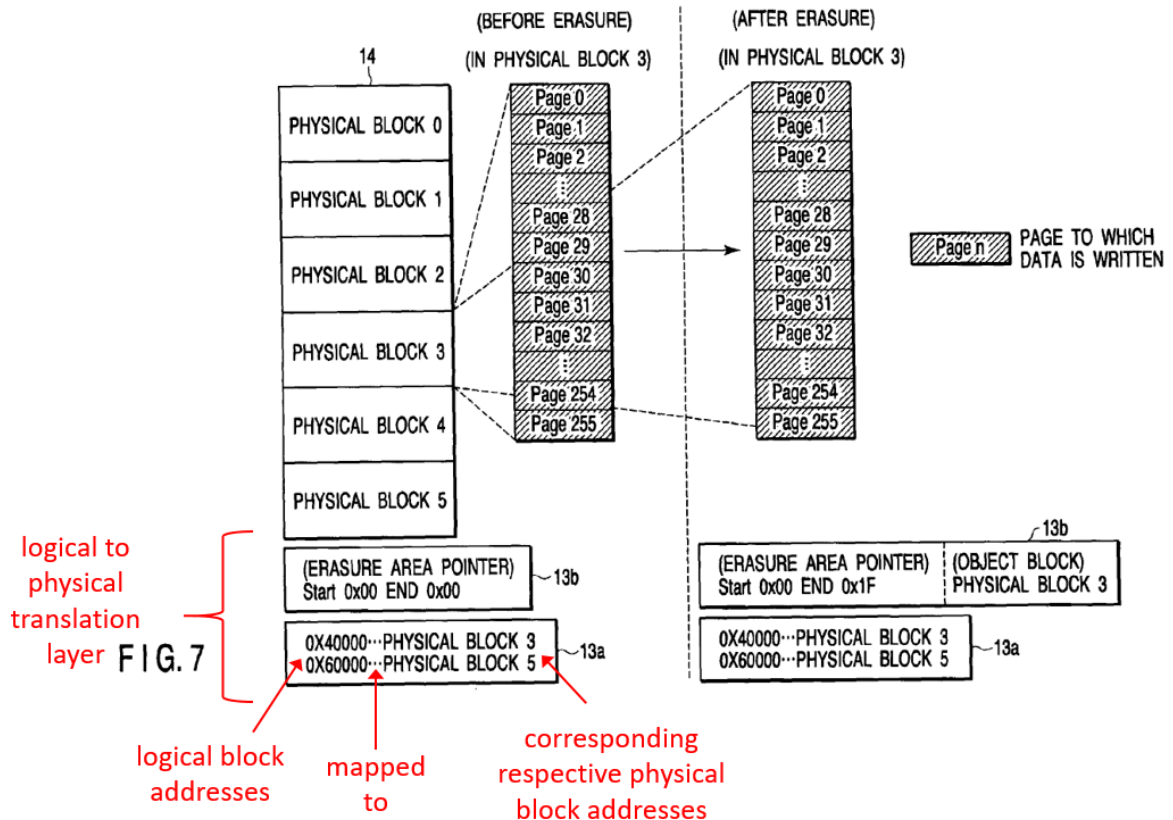
d. “receiving, via the NAND controller, the empty-block identifier;”

114. A POSITA would have understood that Suda teaches this claim element. Suda teaches that when the host device issues an erase command, the flash memory controlling section (the claimed “NAND controller”) references the logical-to-physical conversion table and detects the physical block address mapped to the logical block address designated in the erase command. Ex. 1003 at 8:66-9:3. Therefore, Suda discloses that the flash memory controlling section responds to the host processor. Thus, a POSITA would have understood that Suda discloses “receiving, via the NAND controller, the empty-block identifier.”

e. “maintaining, via the NAND controller, an index of mappings between logical identifiers and physical NAND flash storage locations; and”

115. Suda discloses this claim element. Specifically, Suda discloses a logical-to-physical translation layer that includes the logical and physical address table and the erasure area pointer storage area. Ex. 1003 at Fig. 1. These tables are maintained by the flash memory controlling section (the claimed “NAND

controller”). Suda shows how the logical and physical address table 13a maps logical block addresses to corresponding respective physical addresses. *Id.* at Fig. 7 (annotated below); 3:41-55 (describing how the logical and physical address table maps logical block addresses to physical block addresses).



Id. at Fig. 7 (annotated).

116. Thus, a POSITA would have understood that Suda teaches “maintaining an index of mappings between logical identifiers and physical NAND flash storage locations.”

f. “updating the index of mappings, in response to the receiving the empty-block identifier, to indicate that

data associated with the file on one or more of the physical NAND flash storage locations identified by the empty-block identifier does not need to be preserved.”.

117. A POSITA would understand that Suda teaches this claim element.

Suda discloses two examples for removing assignments of logical addresses to physical addresses. In the first example, an erase command for an entire block is received, causing the entire block to be marked by erasure area pointers as shown in Fig. 4. Ex. 1003, Fig. 4. In response, Suda teaches “canceling the relation between the logical block addresses and the physical addresses.” *Id.*, 5:65-6:3. Thus, the relation (the claimed “assignment”) between the logical block address and the physical address are canceled (“removed,” as claimed). In a second example of Fig. 6, an erase command is received to erase memory including block B. *Id.*, 6:15-21. In response, Suda again reiterates to “erase[] address information of the physical block B and a logical block address,” from the logical and physical address table. *Id.*, 6:33-41. Again, the information (the claimed “assignment”) of the logical block address and the physical address are erased (“removed,” as claimed). A POSITA would have understood both of these examples as Suda’s flash memory controlling system cancelling/erasing an assignment between an identified logical address and a physical address of the solid-state storage medium in response to an erase message.

118. Additionally, Suda discloses using erasure area pointers to mark data at the designated addresses as in a “virtual erased state.” *E.g.*, Ex. 1003, 5:19-27. A POSITA would have understood Suda’s teaching of updating erasure area pointers to indicate that data is to be erased and not to be preserved.

119. Based on the reasons above, a POSITA would have understood Suda to teach “updating the index of mappings, in response to the receiving the empty-block identifier, to indicate that data associated with the file on one or more of the physical NAND flash storage locations identified by the empty-block identifier does not need to be preserved.”

2. Claim 10

- a. “The method of claim 7, wherein the empty-block identifier indicates that the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier should be overwritten.”***

120. A POSITA would have understood that Suda teaches this element. Suda discloses start and end erasure area pointers that designate a range of addresses as virtually erased data, meaning that they are to be erased from flash memory. Ex. 1003, 5:19-27, 5:36-53, 8:66-9:3. Suda further teaches that once these area pointers indicate that a block is fully filled with virtually erased data, the block will be erased and contain “initial-value” data and can be written to again.

Id., 3:56-67, 5:65-6:3. Suda further discloses that when erasure area pointers indicate an “address range [that] is coincident with the size of the physical block subjected to data erasure, it erases a physical block address”. Ex. 1003, 5:56-6:2. Thus, the erasure area pointers indicate that data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier should be overwritten as claimed. *Id.*

3. Claim 11

- a. “The method of claim 7, further comprising transmitting a message indicating that the data associated with the file on one or more physical NAND flash storage locations identified by the empty-block identifier has been overwritten.”***

121. A POSITA would have understood that Suda teaches this element. As stated previously with respect to claim 10, the block will contain “initial-value” data after it is erased and set in an unused state ready to be written to again. Suda explains that outputted “initial-value data” is a string of empty values, typically all 1’s, that are initially written to an unused memory device—as opposed to user data. Ex. 1003, 3:58-59. Suda discloses that the flash memory controlling section “outputs a control signal indicating that data erasure processing is completed, to the host device 2 through the host interface section 12 (step S8).” Ex. 1003, 8:17-20.

122. Thus, a POSITA would have understood Suda to disclose “transmitting a message indicating that the data associated with the file on one or more physical NAND flash storage locations identified by the empty-block identifier has been overwritten.”

4. Claim 12

a. “The method of claim 7, further comprising overwriting the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier with a series of characters such that the overwritten data is non-recoverable.”

123. A POSITA would have understood Suda to teach this element. A POSITA would have understood that during the typical process of garbage collection, data is overwritten with a series of characters such that the overwritten data is non-recoverable. For example, as I note above, Bennett discloses writing FFs or 00s to the sector and setting an erased flag. Suda discloses that “[w]hen the flash memory controlling section **11** determines in step **S5** that the above address range is coincident with the size of the physical block subjected to data erasure, it erases a physical block address given to the above physical block and data indicating a logical block address related to the physical block address (step **S6**).” Ex. 1002, 7:64-8:2. A POSITA would have known that the erase process involves writing “initial-value data.”—i.e., a string of empty values, typically all 1’s, that

are initially written to an unused memory device—as opposed to user data. *Id.* at 8:35-41; Fig. 9 (steps A2, A3).

124. Thus, a POSITA would understand that Suda teaches “overwriting the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier with a series of characters such that the overwritten data is non-recoverable.”

5. Claim 13

a. “The method of claim 7, further comprising performing garbage collection, via the NAND controller, in the NAND flash storage system.”

125. Suda teaches this claim. Specifically, Suda’s flash memory will not allow rewriting or overwriting data, so it cannot perform “update-in-place of data in the storage medium.” Ex. 1003 at 1:54-55. I described this property of flash memory in background section V(A)(4) above. Thus, a POSITA would have understood that Suda’s system taught garbage collection.

126. “Garbage collection” was a well-known method for erasing data in a background process at a more convenient time. Suda teaches to erase the contents of blocks in a storage recovery process whenever the erasure area pointers indicate a block contains entirely deleted or virtually erased data. For example, Suda teaches to use start and end erasure area pointers to designate an identified set of

data that is in a “virtual erased state.” Ex. 1003 at 5:19-23, Figs. 4-6 (showing examples of erasure area pointers), Figs. 7, 10 (showing erasure area pointers in a table). The data that is in a “virtual erased state” is data that is “to be erased,” but has not been erased, meaning the data is suitable for removal. *E.g., id.* at 5:40-48, 5:57-61, 6:9-14, 6:29-45, 6:60-64, 7:38-41. For example, Fig. 7 (annotated at section VI(C), *supra*) uses shading to show that data remains in the block while being marked by erasure pointers as suitable for removal. The erasure area pointers point the physical addresses identified by logical addresses in an erase command. *E.g.,* Ex. 1003 at 6:15-21, 7:29-34, 8:66-9:3. These pointers are set so that at a later time, the physical address in memory will be erased and will be recovered, as recited in the claim. Once an entire block is marked for erasure, the data will be recovered. *Id.* at Figs. 4, 6, Fig. 8 (steps S5 and S6), 5:54-6:3, 6:15-41. As stated previously with respect to claim element 7[e], the erasure area pointer storage area is maintained by the flash memory controlling section (the claimed “NAND controller”).

127. For these reasons, a POSITA would have understood that Suda teaches “performing garbage collection, via the NAND controller, in the NAND flash storage system.”

6. Claim 14

- a. *“The method of claim 13, wherein performing garbage collection comprises: copying a block containing data that should be preserved from an erase block comprising blocks containing data that does not need to be preserved; and”***

128. Suda teaches this element. For example, Fig. 2 of Suda discloses an example where data written to pages 0 to 31 are to be erased while following pages are to be preserved (“[w]hen the data items of the pages ‘Page0’ to ‘Page31’ are erased, data items written to the pages following ‘Page32’ in the physical block having physical block address ‘3’ are also erased along with the data items of the pages ‘Page0’ to ‘Page31’” as “data erasure is carried out in units of one physical block.”). Ex. 1003, 4:33-38. Suda further discloses that data written to pages following “Page 31” are copied to another block. *Id.* 4:39-43.

129. Thus, Suda discloses “copying a block containing data that should be preserved from an erase block comprising blocks containing data that does not need to be preserved.”

- b. *“erasing the erase block.”***

130. Suda teaches this element. For example, Suda instructs to perform “[e]rasure processing on entire block” in step S6 of Fig. 8 after the erasure area pointer is updated (S4) such that it equals a block size (S5). In a second example, with respect to the block illustrated in Fig. 4, Suda instructs that virtually erased,

physical block (area 22) is “to be erased” and to “be set in an unused state.” Ex. 1003, 5:65-6:3, 3:56-67 (explaining that blocks in an unused state have “initial-value” data and “can be used” again). In a third example, with respect to physical block B in Fig. 6, Suda discloses, “the data items written to the entire area of the physical block B are to be erased,” and further instructs “setting the entire area (area 25) of the physical block B in an unused state.” Ex. 1003, 6:33-41.

131. Thus, Suda discloses “erasing the erase block.”

7. Claim 15

a. “The method of claim 14, further comprising writing the copied data, via the NAND controller, to the erased erase block.”

132. A POSITA would have understood that Suda teaches this element. Suda discloses that “an unused physical block can be used when its physical block address is related to a logical block address in accordance with the control of the flash memory controlling section 11.” Ex. 1003, 3:64-66. Further, Fig. 10 in Suda demonstrates the assignment of an unused physical block. A POSITA would have known that Suda’s flash memory controller section would assign an address to a physical address that is erased and set to an unused state ready to be written to again. Thus, a POSITA would have understood that Suda teaches “writing the copied data, via the NAND controller, to the erased erase block.”

8. Claim 16

- a. “The method of claim 7, wherein the NAND flash storage system is an append-only storage system.”***

133. A POSITA would have understood that Suda teaches this claim element. As I stated previously with respect to claim 13, Suda’s flash memory does not support “update-in-place of data.” As stated previously with respect to claim 7[e], Suda discloses a process of maintaining an index of mappings between logical identifiers and physical NAND flash storage locations. A POSITA would have known that this process does not change the stored data. Thus, a POSITA would have understood that Suda teaches that “the NAND flash storage system is an append-only storage system.”

9. Claim 17

- a. “The method of claim 14, wherein each of the one or more physical NAND flash storage locations are smaller than the erase block.”***

134. A POSITA would have understood that Suda teaches this claim element. A POSITA would have known that flash memories are erased in blocks. *See* § VI.A.1. Suda discloses that blocks are made up of multiple physical NAND flash storage locations. Ex. 1003, 8:47-48. Thus, a POSITA would have understood that Suda teaches that “each of the one or more physical NAND flash storage locations are smaller than the erase block.”

VIII. OBVIOUSNESS OF THE CLAIMS BASED ON BENNETT

1. Claim 7

- a. Preamble: “A method for managing data in a system having a host processor and a block-based NAND flash storage system, the NAND flash storage system having physical NAND flash storage locations controlled by a NAND controller, the method comprising:”*

135. I have been asked to assume that the preamble of Claim 7 is a limitation of the claims. Bennett is titled “Methods For The Management Of Erase Operations In Non-Volatile Memory.” Ex. 1002 at Title. Further, Bennett discloses a host (10) and memory system (20). Within a host 10, Bennett discloses a Host-side Memory Manager (the claimed “host processor”), as shown below.

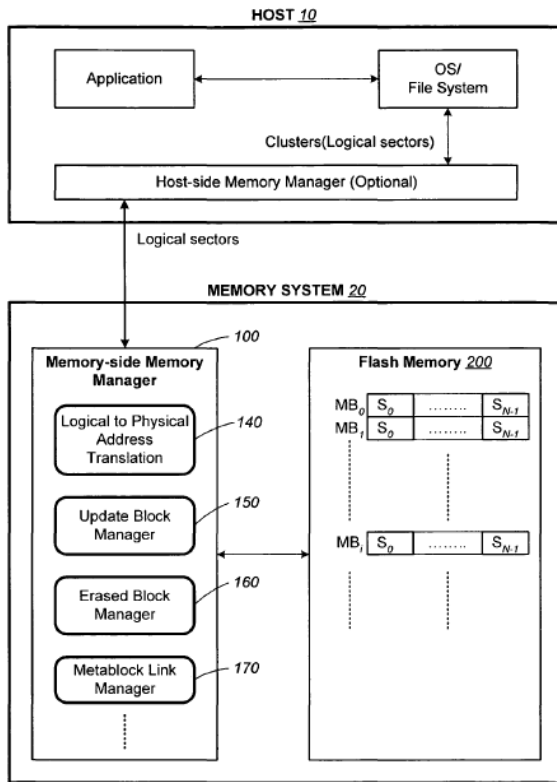


FIG. 2

See Ex. 1002 at Fig. 2.

136. Bennett also discloses a “Flash memory 200.” *Id.* A POSITA would have known that one type of flash memory is “NAND” flash. *Id.*, 2:44-46. A POSITA would also have known that NAND flash has physical NAND storage locations controlled by a NAND processor.

137. Bennett also teaches a memory system that has memory “whose operations are controlled by a controller.” Ex. 1002 at 6:62-63. The controller has an interface, which “has one component interfacing the controller to a host and another component interfacing to the memory 200.” *Id.* at 6:66-7:4, Fig. 6

(depicting “memory 200” as flash memory). As a result, a POSITA would have understood that this controller is configured to perform storage operations on the flash memory.

138. Bennett’s storage operations involve the controller’s “memory side manager” which “contains a number of software modules for managing erase, read and write operations of the metablocks, . . . [and] maintains system control and directory data associated with its operations among the flash memory 200 and the controller RAM 130. *Id.* at 7:36-41. Figure 2 of Bennett demonstrates the memory organized into physical groups of sectors, or metablocks, and the memory is managed by a memory manager of the controller. *Id.* at 5:15-18. Bennett teaches that the memory is accessed by a host, e.g. for a read or write operation, when “running an application under a file system or operating system.” *Id.* at 7:22-24. A POSITA would have understood that the host addresses data in terms of file names and logical identifiers (e.g., as mapped in a file system) while the flash memory addresses data in terms of physical addresses. *See, e.g., id.* at 7:32-35, 7:42-46, Fig. 3a (depicting mapping logical identifiers to physical addresses).

139. As such, a POSITA would have understood that Bennett teaches “a method for managing data in a system having a host processor and a block-based

NAND flash storage system, the NAND flash storage system having physical NAND flash storage locations controlled by a NAND controller.”

b. “receiving, via the host processor, an indication of deletion of a file at the host processor;”

140. A POSITA would have understood that Bennett teaches this claim element. Bennett discloses that “[t]he host 10 accesses the memory 200 when running an application under a file system or operating system.” Ex. 1002, 7:21-22. A POSITA would have understood that file systems store logical addresses and that memory systems store data at physical addresses. Bennett further discloses “a host issu[ing] a command to erase a portion of the memory, such as an Erase Sectors command that specifies the logical sectors to erase.” Ex. 1002, 12:15-17, 4:22 (“a host issues an Erase Sectors command”), Fig. 1 (showing host computer 10).

141. Alternatively, as I explain above, a POSITA would have been well aware of certain common steps that regularly occur when a user deletes data on a computer system with flash memory. *See, e.g.*, § VI.A.8. A POSITA would have known these steps to show that an indication of deletion is received at the OS/File system. Bennett’s Fig. 2 shows that the file system is connected to the host-side memory manager (the claimed “host processor”). Accordingly, a POSITA would have understood that an application notifies the OS/File system of a file being

deleted and that this notification is transferred to the host-side memory manager to then generate an erase command. Thus, a POSITA would have known that Bennett teaches “receiving, via the host processor, an indication of deletion of a file at the host processor.”

c. “creating, via the host processor, an empty-block identifier in response to the indication of deletion;”

142. A POSITA would have understood that Bennett teaches this claim element. Specifically, as illustrated with respect to Figure 2, Bennett’s erase command can be sent by a “host-side memory manager” in a host. A POSITA would have understood that such an erase command often originates from a user interacting with an application to delete data, e.g., a document or other file. And, as I explain above, a POSITA would have been well aware of certain common steps that regularly occur when a user deletes data on a computer system with flash memory. *See, e.g.*, § VI.A.8.

143. Bennett does not expressly disclose that this erase command also indicates that the logical address has been deleted by the host. But a POSITA would have understood that hosts can delete logical addresses from a file system in response to user action that deletes a file. At the very least, notifying storage of deleted logical addresses, in response to user actions that delete files, would have been an obvious thing for a POSITA to try and a simple matter to implement. All

that would be necessary to implement this would be for the erase command to include an indication that the logical address (corresponding to the data) has been deleted by the host. A POSITA would have recognized that including this command can be part of the process where a host releases obsolete file data—including logical addresses corresponding to deleted data—to free up storage. Bennett provides teachings in furtherance of this goal, explaining that when “the data in a portion of the memory becomes obsolete, or the memory receives a command to erase a particular portion, in more advanced memory systems it is common for the designated portions not to be erased immediately at that time, but to be “logically erased” by being marked for erase, with the actual, physical erase taking place at a later time.” *Id.* at 3:25-32. A POSITA would know that Bennett’s erase commands indicate that the data (stored at the physical block addresses) corresponding to the identified logical block addresses is deleted at the host.

144. Thus, a POSITA would have understood that Bennett teaches “creating, via the host processor, an empty-block identifier in response to the indication of deletion.”

d. “receiving, via the NAND controller, the empty-block identifier;”

145. Bennett teaches this claim element. Bennett discloses an erase command that originates “either from the host or with the memory system itself.” Ex. 1002 at 5:56-58.

146. Figure 2 of Bennett shows how this erase command is received—it is received from the host 10 through the memory system 20’s “Memory-side Memory Manager” 100 (the claimed “NAND controller”).

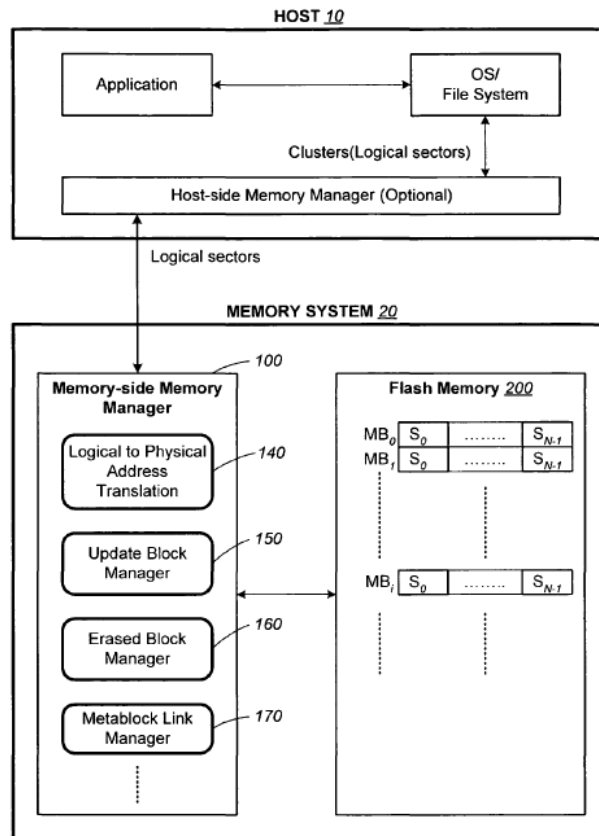


FIG. 2

147. As stated with respect to claim 7[c], Bennett’s erase command is an empty-block identifier. See § VI(1)(c). As discussed for claim 7[a], the controller

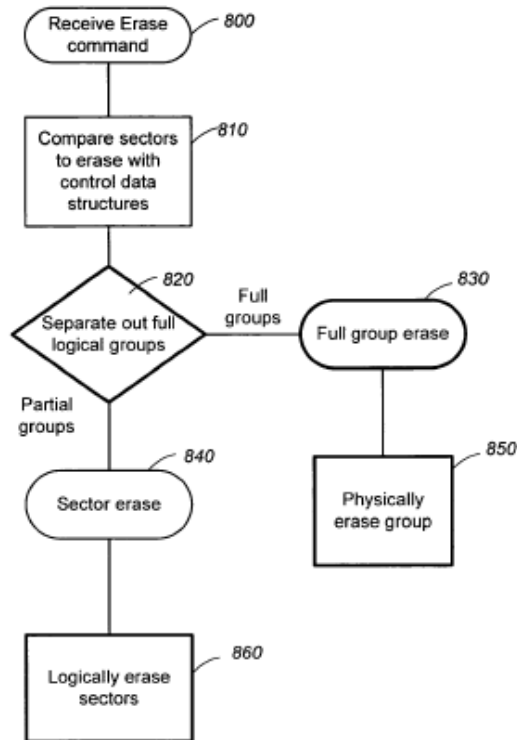
is a NAND controller. *See* § VI(1)(a). Thus, a POSITA would have understood that Bennett teaches “receiving, via the NAND controller, the empty-block identifier.”

e. “maintaining, via the NAND controller, an index of mappings between logical identifiers and physical NAND flash storage locations; and”

148. Bennett teaches this claim element. Bennett teaches that its memory side memory manager includes a “logical to physical address table translation module” that will assign logical addresses of the logical address space to physical addresses in flash memory. Ex. 1002 at 10:37-39. Bennett has a detailed description of mapping the logical addresses and physical addresses in a table/index, e.g., in a flash memory GAT. *Id.* at Figs 6, 8B, 10:19-21 (“The GAT 210 keeps track of the mapping between logical groups of sectors and their corresponding metablocks.”). Thus, Bennett teaches “maintaining, via the NAND controller, an index of mappings between logical identifiers and physical NAND flash storage locations.”

f. “updating the index of mappings, in response to the receiving the empty-block identifier, to indicate that data associated with the file on one or more of the physical NAND flash storage locations identified by the empty-block identifier does not need to be preserved.”

149. Bennett teaches this claim element. Bennett teaches updating the index of mappings by its disclosure of the memory-side memory manager storing logical-to-physical address tables in a GAT (group address table). *Id.* at 10:19-21, 10:65-11:60, Figs. 6, 8A-8B. Bennett also teaches “marking sectors as erased” in response to an erase command. *Id.*, 20:20-61. For logical erasures, Bennett setting an erased flag. *Id.* at 20:47-49. In doing so, the previously assigned physical address is no longer used, and therefore the assignment between a logical and a physical address is removed. *Id.* at 20:47-50 (“In this case, all the data of the Logical Group will not be changed, but will not be read, as the host will be sent FFs or 00s as if the sectors were erased.”). Bennett further teaches indicating that data associated with a physical NAND flash storage location does not need to be preserved. For example, Fig. 10 below shows a flowchart to illustrate erase methods:



Ex. 1002, Fig. 10. As shown, the memory controller (at step 820) separates erase commands for “partial groups” and “full groups.” *Id.* Bennett teaches that “logically erased” data can be “physically erased when convenient, for example in a background process.” *Id.*, 3:39-41.” A POSITA would have understood that this data need not be preserved as claimed.

150. To the extent that Bennett’s disclosure of setting an erase flag does not satisfy this limitation, it is also my opinion that a POSITA would have found this limitation to be obvious in view of Bennett’s teachings. For example, Bennett teaches that “sectors are ‘logically’ erased at the sector level by standard

techniques.” *Id.* at Abstract, 6:18-20 (“For the specified sectors not forming complete block, the system uses the system’s standard, logical erase method.”).

151. For these reasons, a POSITA would have understood that Bennett teaches “wherein the indexer is further configured to remove an assignment between an identified logical address and a physical address of the solid-state storage medium in response to a message received from a host operating system, the message indicating that the identified logical address is erased.”

2. Claim 10

g. “The method of claim 7, wherein the empty-block identifier indicates that the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier should be overwritten.”

152. A POSITA would have understood that Bennett teaches this element. Bennett teaches that “[w]hen an erase command is received, the specified sectors are checked against the memory system's control data. If the specified sectors span any full logical grouping, the full logical groupings can each be treated as a whole and erased according to one process (such as performing a true, physical erase), while other sectors are “logically” erased at the sector level by standard techniques.” Ex. 1002, Abstract. A POSTIA would have understood that an erase command specifying a “full logical group” indicates that the data should be

overwritten. Bennett teaches overwriting in the form of writing FFs or 00s to the sector and setting an erased flag. *Id.*, 20:20-28. Further, Bennett teaches a cleared block list that keeps a list of physical block addresses that have been erased and are thus available for storing new data. *Id.*, 10:27-33. *See* § VI.A.1, *supra*.

153. Based on this, a POSITA would have understood that Bennett teaches “the empty-block identifier indicates that that the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier should be overwritten.”

3. Claim 11

h. “The method of claim 7, further comprising transmitting a message indicating that the data associated with the file on one or more physical NAND flash storage locations identified by the empty-block identifier has been overwritten.”

154. A POSITA would have understood that Bennett teaches this element. Bennett discloses “marking full logical groups as erased, where the logical group has all sectors written with FF or 00 data and an ‘erased’ flag set in corresponding headers.” Ex. 1002, 20:20-28. Bennett also discloses returning status information regarding whether sectors have been erased. Ex. 1002, 4:36-40 (“the invention is able is to provide compatibility with the protocols where the system needs to return the information about whether the sector being read was erased by an Erase Sectors

command or not”), 4:39-42 (“an erased block (or erased sector) status can be returned if a sector belonging to a block (or the sector itself) is erased”).

155. Thus, a POSITA would have understood that Bennett teaches “transmitting a message indicating that the data associated with the file on one or more physical NAND flash storage locations identified by the empty-block identifier has been overwritten.”

4. Claim 12

- a. “The method of claim 7, further comprising overwriting the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier with a series of characters such that the overwritten data is non-recoverable.”*

156. A POSITA would have understood that Bennett teaches this claim. Specifically, Bennett discloses writing FFs or 00s” to the non-volatile memory. *Id.*, 20:25-27. A POSITA would have understood that writing FFs or 00s causes the erasure of flash memory. *See* § V.A.1, *supra*. Data associated with the file on a deleted physical address location cannot be recovered.

157. Based on this, a POSITA would have understood that Bennett teaches “overwriting the data associated with the file on the one or more physical NAND flash storage locations identified by the empty-block identifier with a series of characters such that the overwritten data is non-recoverable.”

5. Claim 13

- a. “The method of claim 7, further comprising performing garbage collection, via the NAND controller, in the NAND flash storage system.”***

158. “Garbage collection” was a well-known method for erasing data in a background process at a more convenient time. *See* § V.A.5, *supra*. And Bennett taught this. For example, Bennett discloses “in more advanced memory systems it is common for the designated portions not to be erased immediately at that time, but to be ‘logically erased’ by being marked for erase, with the actual erasure in physical memory taking place at a later time.” Ex. 1002 at 3:26-32. Bennett explicitly points out that one way to rewrite the update data is to do so in the same physical memory location, which is inefficient. Ex. 1002 at 3:12-17. The garbage collection method is more efficient.

159. Bennett also discloses that “it is desirable to have the erase block of substantial size ... [so the] erase time is amortized over a large aggregate of memory cells.” Ex. 1002 at 3:7-9. A POSITA would have found it obvious to perform garbage collection in response to an erase command to determine if there are other areas that need to be erased at the same time in order to be more efficient. Furthermore, setting garbage collection to potentially initiate in response to an erase command is one of a limited number of available options (e.g., determining

whether to initiate garbage collection in response to receiving an erase command, in response to a certain percentage of invalid data, or after a period of time has lapsed) each of which would have been obvious to try with Bennett's disclosure.

6. Claim 14

- a. *“The method of claim 13, wherein performing garbage collection comprises: copying a block containing data that should be preserved from an erase block comprising blocks containing data that does not need to be preserved; and”***

160. Bennett teaches this claim element. Bennett discloses that GAT blocks are allocated and valid GAT sectors are copied in sequential order from the full Gat block. Ex. 1002, 12:6-11. Bennett further describes the sequence of erasing logical groups including the following steps:

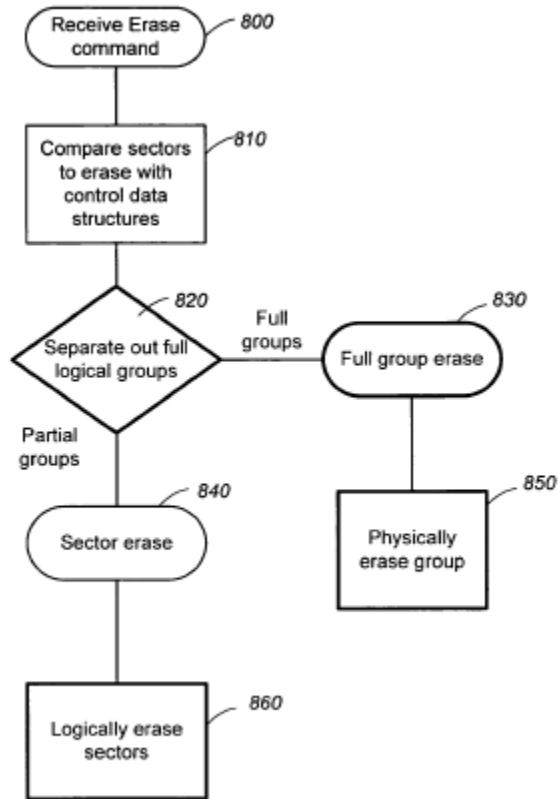
1. Open a new Update Block for the Logical Group. If the current number of Update Blocks is up to the limit, then close (garbage collect) another Update block. If the Logical Group has an open Update block, then force closure of the Update block which belongs to the Logical group.
2. Write at least one sector to the new Update block with any data and “erased” flag set in the header.
3. Erase the Original block of the Logical Group.
4. Update GAT marking the Logical Group as physically erased. Update erase block management (EBM) 760 (FIG. 18 a) by adding the erased block to the erase pool. Perform EBM-MAP exchange if EBL get full as the erase pool increases.
5. Erase the new Update Block of the Logical Group. There is no need to maintain it in control update, as this block is not referenced by any data structure and is meant to be erased.

6. As a result, all the block associated with the Logical Group are erased. If power loss happens in the middle of this sequence, the “erased” sector can be found and the operation can be completed.

Id. 19:5-28. Thus, a POSITA would have understood that Bennett teaches “copying a block containing data that should be preserved from an erase block comprising blocks containing data that does not need to be preserved.”

b. “erasing the erase block.”

161. Bennett teaches this element. For example, with respect to Figure 10, Bennett discloses that a memory controller (step 820) separates erase commands for partial and full logical groups. The full groups can then be physically erased (step 850) and the partially groups are logically erased (step 860). *Id.* at Fig. 10, 4:12-21. The logically erased data can then be “physically erased when convenient, for example in a background process.” *Id.* at 3:39-41.



Ex. 1002 at Fig. 10. A POSITA would have understood that Bennett’s disclosure of responding to erase commands, specifying a full logical group, by actually erasing the physical memory, returns the memory to a point where it can be written again. Thus, a POSITA would have understood that Bennett teaches “erasing the erase block.”

7. Claim 15

- a. “The method of claim 14, further comprising writing the copied data, via the NAND controller, to the erased erase block.”**

162. Bennett teaches this claim element. Bennett discloses writing to erased storage. *See, e.g.*, Ex. 1002 at 3:15-17 (“the entire erase block contain that physical location will have to be first erased and then rewritten with the updated data”). Thus, a POSITA would have known that Bennett teaches “writing the copied data, via the NAND controller, to the erased erase block.”

8. Claim 16

a. “The method of claim 7, wherein the NAND flash storage system is an append-only storage system.”

163. A POSITA would have understood that Bennett teaches this claim element. A POSITA would have known that for NAND flash memory, blocks must be erased before written to again. *See* § V.A.1, *supra*. As stated previously with respect to claim 7[e], Bennett discloses a process of maintaining an index of mappings between logical identifiers and physical NAND flash storage locations. A POSITA would have known that this process does not change the stored data.

164. Thus, a POSITA would have understood that Bennett teaches that the NAND flash storage system is append-only.

9. Claim 17

a. “The method of claim 14, wherein each of the one or more physical NAND flash storage locations are smaller than the erase block.”

165. Bennett teaches this claim. Bennett teaches that entire erase blocks contain physical locations. Ex. 1002 at 3:14-17. Further, Fig. 4 of Bennett shows a diagram demonstrating the composition of metablocks in physical memory:

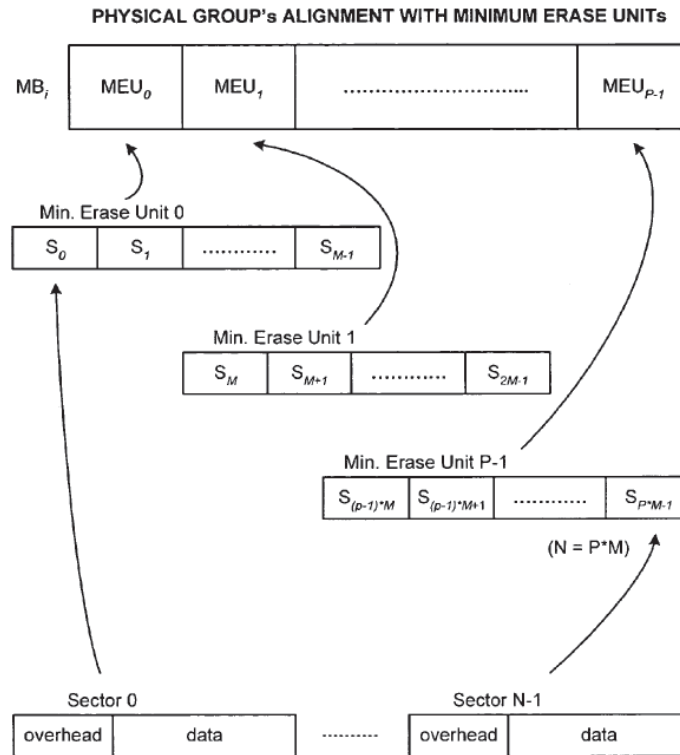


FIG. 4

Ex. 1002, Fig. 4.

166. Thus, a POSITA would have understood that Bennett teaches that “each of the one or more physical NAND flash storage locations are smaller than the erase block.”

IX. OBVIOUSNESS OF CLAIM 11 OVER SUDA AND/OR BENNETT IN FURTHER VIEW OF ZIPPRICH AND KNOWLEDGE OF A POSITA

167. A POSITA would have been motivated to combine teachings from Suda, Bennett, and Zipprich. All references disclose techniques for data erasure and improving indications of data erasure. Suda discloses outputting a control signal indicating that data erasure processing is completed. Ex. 1003, 8:17-20. Bennett discloses writing FFs or 00s to the sector and setting an erased flag to mark full logical groups as erased. *See* § VIII.A.f. Zipprich discloses a “status report” that indicates the status of an overwrite. Ex. 1006, [0018]. Zipprich also discloses that the report provides immediate feedback and logging/tracking of the overwrite events. *Id.* Zipprich further teaches that the report may come in the form of email message or log file. *Id.*

168. A POSITA would have found it obvious for Suda and/or Bennett’s method to include transmitting a message indicating that data has been overwritten. This message would have informed the flash memory controlling section in Suda that a physical block address is available for storing new data, increasing its ability to control physical flash storage locations. Thus, it would have been obvious to a POSITA that Suda and/or Bennett, alone, or in combination with Zipprich, renders obvious the limitation “transmitting a message indicating that the data associated

with the file on one or more physical NAND flash storage locations identified by the empty-block identifier has been overwritten.”

X. OBVIOUSNESS OF CLAIMS 7, 10-13 OVER BENNETT IN VIEW OF SUDA

169. A POSITA would have understood that Bennett renders these claims obvious, for example, in view of Suda. Although Bennett does not explicitly disclose receiving an indication of deletion of a file, as I explain above, a POSITA would have been well aware of certain common steps that regularly occur when a user deletes data on a computer system with flash memory. *See, e.g.*, § VI.A.8. Suda discloses steps specific to a digital camera that occur when a user deletes photos. *See* § VII.A.b.

170. A POSITA would have found it obvious that the host processor in Bennett’s system operates similarly to the host processor in the Suda’s disclosed digital camera. Operating in such a manner would have increased the ability of Bennett’s host processor to identify a logical address when issuing an erase command. Thus, it would have been obvious to a POSITA that Bennett alone, or in combination with Suda, renders obvious claims 7 and 10-13.

XI. OBVIOUSNESS OF CLAIMS 13-15 OVER BENNETT AND/OR SUDA IN VIEW OF SWSTE’05

171. A POSITA would have been motivated to combine teachings from Bennett, Suda, and SwSTE'05. All references discuss the management of flash storage devices. Ex. 1002, *passim*; Ex. 1003, *passim*; Ex. 1007, *passim*. All references propose techniques for improving performance given the same limitations of flash memory. Namely, that flash memory only supports erasing blocks, not erasure of individual pages within a block (Ex. 1002, 3:43-50; Ex. 1003, 4:33-38; Ex. 1007, Abstract, § 1), and blocks cannot be directly overwritten without erasure (Ex. 1002, 3:10-17; Ex. 1003, 1:19-22, 1:54-55; Ex. 1007 § 1). Bennett discloses writing to erased storage. Ex. 1002, 3:15-17 (“the entire erase block contain that physical location will have to be first erased and then rewritten with the updated data”). Suda in Fig. 10 demonstrates the assignment of an unused physical block. Ex. 1003, Fig. 10. SwSTE'05 teaches a garbage collection process including a step for copying valid sectors to newly allocated free space. Ex. 1007 § 2.3.

172. A POSITA would have found it obvious to apply the copying step to Bennett's system in order to free up blocks. A POSITA would have recognized that the “valid sectors” are Bennett's “data,” and that the “newly allocated free space” are Bennett's “erased blocks” that have been restored to a point where they can be written to again. A POSITA would have found it obvious for Bennett's

system to include “writing the copied data, via the NAND controller, to the erase block increase its ability to allow data to be written again, without the delay of erasing invalid data, which greatly improves the on-demand write speed of flash memory. A POSITA would have had a reasonable expectation of success because garbage collection was well-known and had been a standard part of flash memory management since the mid-1990s.

X. RESERVATION OF RIGHTS

173. My opinions are based upon the information that I have considered to date. I am unaware of any evidence of secondary considerations with respect to the ‘359 patent that would render any of the asserted claims non-obvious. I reserve the right, however, to supplement my opinions in the future to respond to any arguments raised by the owner of the ‘359 patent and to consider new information that becomes available to me.

XI. CONCLUSION

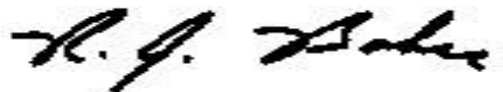
174. For the reasons given above, claims 7 and 10-17 of the ‘359 patent are invalid.

175. In signing this declaration, I recognize that the declaration will be filed as evidence in a case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to

Declaration in Support of Petition for IPR of U.S. Patent No. 11,640,359

cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

176. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

By: 
R. Jacob Baker, Ph.D., P.E.

Date: 12/18/2020