IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent of Gonzalez *et al.*:	Petition for *Inter Partes* Review
U.S. Patent No. 7,120,729	Attorney Docket No.: 337722-000080.729
Issued: October 10, 2006	
Title: Automated Wear Leveling in Non-Volatile Storage Systems	Customer No.: 26379
	Petitioner: Apple Inc.
	Real Party-in-Interest: Apple Inc.

PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 7,120,729

Mail Stop Patent Board

Patent Trial and Appeal Board

P.O. Box 1450

Alexandria, VA 22313-1450

Dear Sir:

Pursuant to the provisions of 35 U.S.C. §§ 311-319, Apple Inc. (hereinafter "Petitioner") hereby petitions the Patent Trial and Appeal Board to institute an *inter partes* review of claims 15-17 of United States Patent No. 7,120,729 (the "'729 patent") (Ex. 1001).

TABLE OF CONTENTS

Page

I. MANDATORY NOTICES 1
A. Real Party-in-Interest 1
B. Related Matters 1
C. Lead and Back-up Counsel 2
D. Service Information 2
II. GROUNDS FOR STANDING 2
III. RELIEF REQUESTED 3
IV. THE REASONS FOR THE REQUESTED RELIEF 3
A. Summary of Reasons 3
B. Relevant Background Technology 3
1. Overview of Flash Memory 4
2. Wearing Down of Flash Memory Cells 8
3. Non-In-Place Update 11
C. Overview of the '729 patent 14
D. Prosecution History of the '729 Patent 15
E. Level of Ordinary Skill in the Art 17
F. Claim Construction under 37 C.F.R. § 42.104(b)(3) 17
1. "the plurality of blocks maintained as an erased block pool" (claim 15) 18
2. "identifying at least one of the plurality of physical blocks at a time" (claim 15) 19
G. Challenge #1: Wells Anticipates Claims 15-17 19
1. Overview of Wells 19

TABLE OF CONTENTS

(continued)

Page

2. Wells anticipates independent claim 15.....23

3. Wells anticipates dependent claim 16.....30

4. Wells anticipates dependent claim 17.....32

H. Challenge #2: Claims 15-17 are obvious over the Linux pcmcia-cs package version 3.1.21 by David Hinds ("the Linux publication") in view of the knowledge of a person of ordinary skill in the art ("POSITA") as evidenced by the Media Storage Formats Specification (Volume 7) from the PC Card Standard Release 7.0 (1999), ("PC Card Standard").....33

1. Overview of the Linux publication.....33

2. The Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders independent claim 15 obvious 37

3. The Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders dependent claim 16 obvious.....45

4. The Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders dependent claim 17 obvious.....48

V. CONCLUSION.....50

Exhibit Number	**Description**
Ex. 1001	U.S. Patent No. 7,120,729 to Gonzalez et al. ("the '729 Patent")
Ex. 1002	Prosecution File History For U.S. Patent No. 7,120,729 to Gonzalez et al.
Ex. 1003	Declaration of Dr. R. Jacob Baker in Support of Petition
Ex. 1004	U.S. Patent No. 5,341,339 to Wells
Ex. 1005	U.S. Patent No. 6,151,246 to So
Ex. 1006	U.S. Patent No. 6,396,744 to Wong
Ex. 1007	U.S. Patent No. 5,838,614 to Estakhri
Ex. 1008	Excerpts from *Designing with FLASH MEMORY: The Definitive Guide to Designing Flash Memory Hardware and Software for Components and PCMCIA Cards* by Brian Dipert & Markus Levy Annabooks (1993, 1994)
Ex. 1009	U.S. Patent No. 5,485,595 to Assar
Ex. 1010	PC Card Standard, Volume 7, Media Storage Formats Specification
Ex. 1011	*A Floating Gate and Its Application to Memory Devices*, D. Kahng and S. M. Sze
Ex. 1012	*New Ultra High Density EPROM and Flash EEPROM with NAND Structure Cell*, Fujio Masuoka

Exhibit Number	Description
	et. al.
Ex. 1013	Toshiba Web at flash25.toshiba.com
Ex. 1014	U.S. Patent No. 5,418,752 to Harari
Ex. 1015	U.S. Patent No. 6,362,049 to Cagnina
Ex. 1016	Intel FDI User Manual
Ex. 1017	*Cleaning Policies in Mobile Computers Using Flash Memory*, M.-L. Chiang & R.-C. Chang
Ex. 1018	*Managing Flash Memory in Personal Communication Devices*, Mei-Ling Chiang *et. al.*
Ex. 1019	U.S. Patent No. 5,740,395 to Wells
Ex. 1020	U.S. Patent No. 5,940,861 Brown
Ex. 1021	U.S. Patent No. 7,012,835 to Gonzalez
Ex. 1022	U.S. Publication No. 2003/0046487 to Swaminathan
Ex. 1023	Intel Series 2 Flash Memory Cards Data Sheet
Ex. 1024	*Curriculum Vitae* of Dr. R. Jacob Baker
Ex. 1025	Declaration of Dr. David Hinds in Support of Petition
Ex. 1026	SUPPORTED.CARDS file within published pcmcia-cs-3.1.21.tar
Ex. 1027	CHANGES file within published pcmcia-cs-3.1.21.tar
Ex. 1028	PCMCIA Programmer's Guide within published pcmcia-cs-3.1.21.tar
Ex. 1029	PCMCIA How-To within published pcmcia-cs-3.1.21.tar

Exhibit Number	**Description**
Ex. 1030	ftl.h within published pcmcia-cs-3.1.21.tar
Ex. 1031	ftl_cs.c within published pcmcia-cs-3.1.21.tar
Ex. 1032	ftl_check.c within published pcmcia-cs-3.1.21.tar
Ex. 1033	ftl_format.c within published pcmcia-cs-3.1.21.tar
Ex. 1034	iflash.h within published pcmcia-cs-3.1.21.tar
Ex. 1035	iflash2_mtd.c within published pcmcia-cs-3.1.21.tar
Ex. 1036	iflash2+_mtd.c within published pcmcia-cs-3.1.21.tar
Ex. 1037	*Nonvolatile Semiconductor Memory Technology: A Comprehensive Guide to Understanding and Using NVSM Devices*, by William D. Brown and Joe E. Brewer, ed., IEEE Press (1998).

I. MANDATORY NOTICES

A. Real Party-in-Interest

Pursuant to 37 C.F.R. § 42.8(b)(1), the real party-in-interest is Apple Inc. ("Petitioner").

B. Related Matters

Pursuant to 37 C.F.R. § 42.8(b)(2), Petitioner states that Longitude Flash Memory Systems S.A.R.L. ("Patent Owner") is asserting U.S. Patent 7,120,729 (the "'729 patent") against the Real Party-In-Interest in a suit filed September 23, 2014, styled *Longitude Licensing Ltd., and Longitude Flash Memory Systems S.A.R.L. v. Apple Inc.*, Case No. 3:14-cv-4275, pending in the United States District Court for the Northern District of California (the "Related Litigation"). Petitioner has filed, or soon will file, petitions for *inter partes* review of U.S. Patent Nos. 6,510,488; 6,763,424; 6,831,865; 6,968,421; 7,012,835; 7,224,607; 7,181,611; 7,657,702; 7,818,490; 7,970,987; 8,050,095; and 8,316,177.

As of the filing of this petition, no other judicial or administrative matters are known to Petitioner that would affect, or be affected by, a decision in an *inter partes* review of the U.S. Patent No. 7,120,729 ("the '729 patent").

C. Lead and Back-up Counsel

Lead counsel for this matter is Brent Yamashita (USPTO Reg. No. 53,808), and back-up counsel for this matter is Edward Sikorski (USPTO Reg. No. 39478), both at the e-mail address: Apple-Longitude-IPR@dlapiper.com. The postal and hand delivery address for both is DLA Piper LLP (US), 2000 University Avenue, East Palo Alto, California, 94303, and the telephone and fax numbers are (650) 833-2348 (for phone) and (650) 687-1206 (for fax).

D. Service Information

Pursuant to 37 C.F.R. § 42.8(b)(4), papers concerning this matter should be served on the following email address: Apple-Longitude-IPR@dlapiper.com.

II. GROUNDS FOR STANDING

Pursuant to 37 CFR § 42.104(a), Petitioner certifies that the ’729 patent is available for *inter partes* review, and Petitioner is not estopped or barred from requesting *inter partes* review challenging the ’729 patent on the grounds identified in this petition.

III. RELIEF REQUESTED

Petitioner asks that the Board review the accompanying prior art and analysis, institute a trial for *inter partes* review of claims 15-17 of the ’729 patent, and cancel claims 15-17 as invalid for the reasons set forth below.

IV. THE REASONS FOR THE REQUESTED RELIEF

The full statement of the reasons for relief requested is as follows:

A. Summary of Reasons

- **Challenge #1:** Claims 15-17 are anticipated by U.S. Patent No. 5, 341, 339 ("Wells") (Ex. 1004)
- **Challenge #2:** Claims 15-17 are obvious over the Linux pcmcia-cs package version 3.1.21 publication by David Hinds ("the Linux publication") in view of the knowledge of a person of ordinary skill in the art ("POSITA") as evidenced by the PC Card Standard, Volume 7, Media Storage Formats Specification ("PC Card Standard") (Ex. 1008).

B. Relevant Background Technology

Broadly speaking, the '729 patent addresses wear leveling techniques in flash memory to mitigate the effects of uneven memory cell wear after repeated uses. *See* Ex. 1001 at ABSTRACT. ("Methods and apparatus for performing wear leveling in a non-volatile memory system are disclosed. . . ."). *See id.* at 1:46-50. ("Although non-volatile memory or, more specifically, non-volatile memory storage cells within flash memory systems may be repetitively programmed and erased, each cell or physical location may only be erased a certain number of times before the cell wears out."). However, flash memory technologies, including the

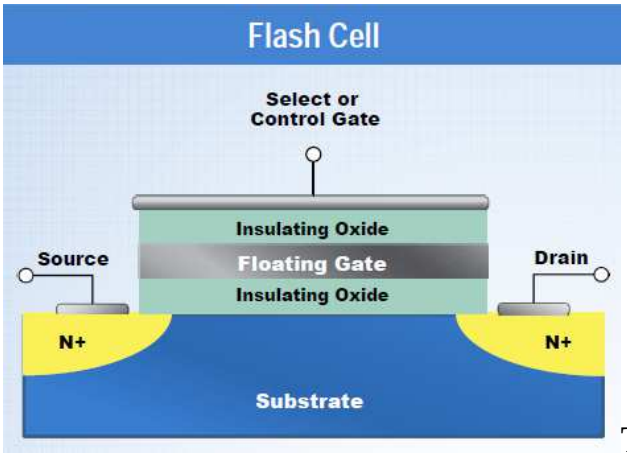
need to perform wear leveling, were already well understood for years prior to 2002, the date to which the '729 patent claims priority.

1. Overview of Flash Memory

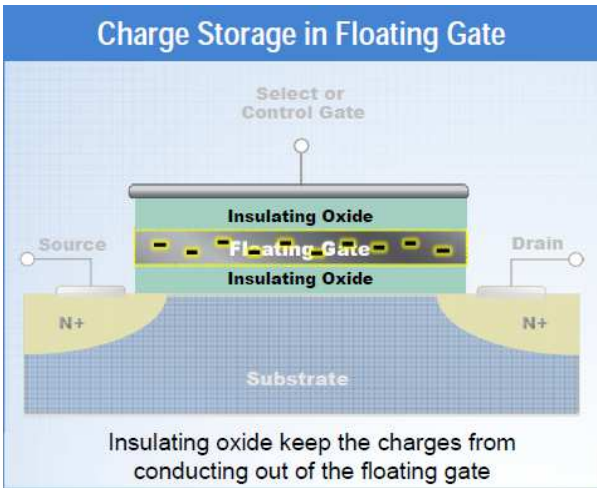
Flash memory is a type of solid state semiconductor non-volatile memory. These devices are now ubiquitous in consumer electronic devices as data storage devices, and are increasingly used as a replacement for magnetic disk drives even in desktop computers. Ex. 1003, Declaration of Dr. Jacob Baker at ¶ 15.

Flash memory typically comprises an array of flash memory cells organized in rows and columns, as in other conventional semiconductor memory systems (such as DRAM or SRAM). Ex. 1003 at ¶¶ 24, 32, 33. Each flash memory cell utilizes a floating gate within a field effect transistor ("FET") to store electrical charge. Ex. 1003 at ¶ 19.

Shown below is an illustration of a typical flash memory cell with a floating gate added to a standard FET structure.



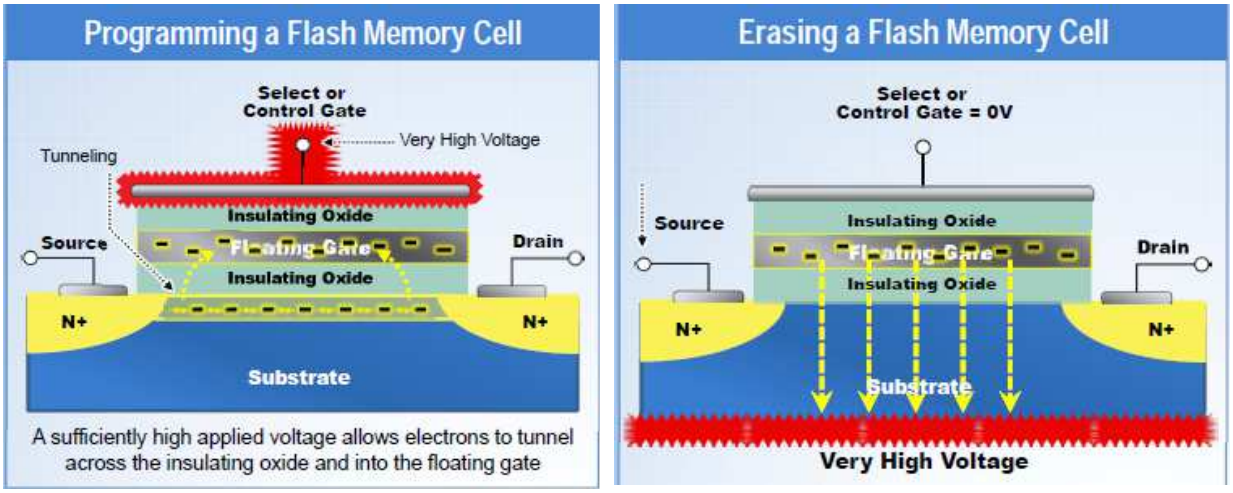
Taken from Ex. 1003 at ¶ 20.



Taken from Ex. 1003 at ¶ 21.

The amount of electrical charge stored in the floating gate can be used to represent data bits ("1" or "0"). Ex. 1003 at ¶¶ 21,22. Since the "floating gate" is electrically insulated from the terminals of the FET, charge cannot readily conduct into or out of the floating gate, which allows long-term storage of the charge even when power is removed from the device. Ex. 1003 at ¶ 19.

In order to utilize such floating gate FET's as memory cells, there must be a way to controllably add or remove electrical charge from the floating gate. This can be accomplished by applying high voltage differences across the terminals of the memory cell (including across the insulating oxide allowing the floating gate to float). *See, e.g.*, Ex. 1008 at 27, 28, 33, 34, 36. Adding charge to the floating gate is termed "programming" (changing the memory from "1" state to "0" state) and removing charge is termed "erasing" (changing from "0" to "1"). Ex. 1003 at ¶¶ 23-24. This is shown in the following illustrations:



This distinction in terminology is significant, because for some types of flash memories (*e.g.* "NOR flash"), the erase operations can take significantly longer than data programming or reading operations. *See, e.g.*, Ex. 1008 at 32, Table 3.1 (reproduced below, showing the state of the art NOR flash characteristics *ca*. 1993, with read operations taking 60 nanoseconds, program operations taking between 6 and 9 microseconds – 100x longer than a read operation, and erase operations taking between 0.3 and 1.6 seconds—5,000,000 times longer than a read operation); Ex. 1003 at ¶ 57.

Density	8 Mbit
Access Time	60 ns
Data Program Time	6 µs (min) 9 µs (typ)
Block Erase Time (64 kbyte block)	0.3 sec (min) 1.6 sec (typ)

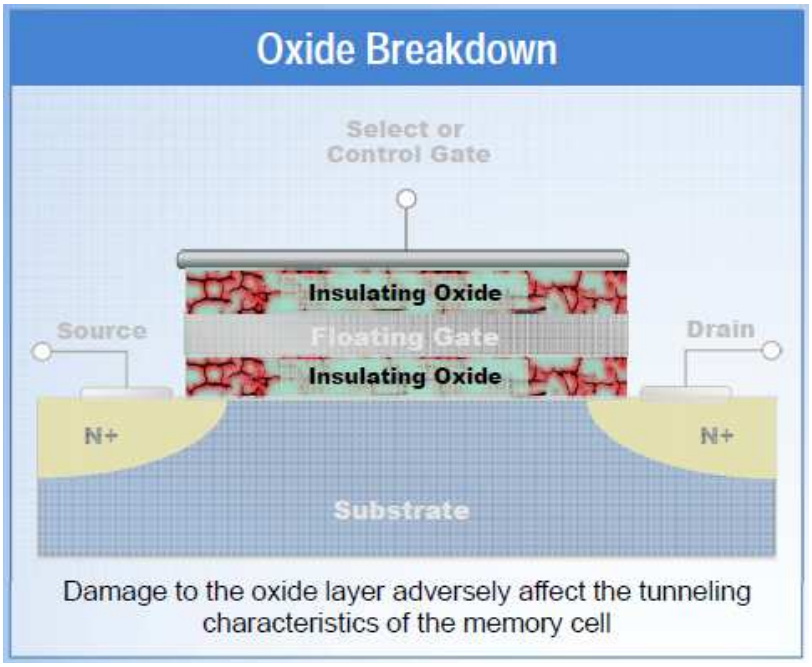
Table 3.1: NOR Flash Memory Characteristics

In real-world devices that require a compact integration of an array of interconnected floating gate FETs, this distinct mechanism for programming vs. erasing also means that the data in flash memories cannot be "overwritten" in the conventional sense—they must first be erased before updated data can be programmed back in the same memory cell. Ex. 1003 at ¶ 24. Specifically, because of how the cells are electrically connected, the erase operation must be simultaneously performed on a much larger group of memory cells (often referred to as a "block" or "erase block") than the number of cells that can be programmed or read simultaneously (often referred to as a "page"). Therefore, "overwriting" a single bit would require erasing an entire "block" of memory cells, including those memory cells where the data is not modified, and therefore can be left unchanged. This is in stark contrast to *e.g.* magnetic storage devices, where a single bit can be changed back and forth. Ex. 1003 at ¶¶ 24-25. This requirement to erase-before-write in flash memory have other practical considerations important in implementing a flash memory storage system, to be discussed below.

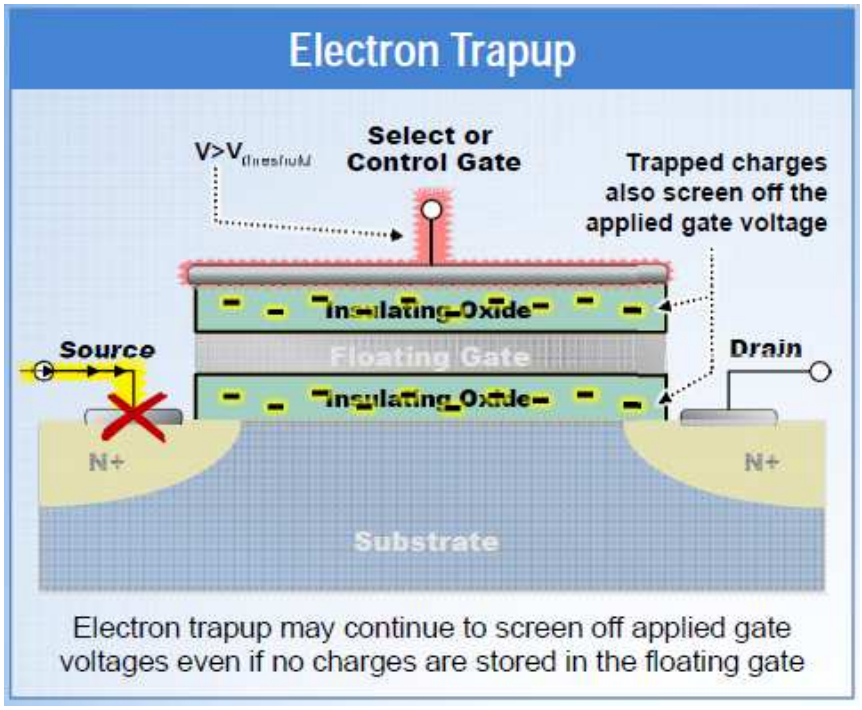
2. Wearing Down of Flash Memory Cells

In actual real-world usage of these flash memory devices, individual cells are repeatedly programmed and erased (referred to in the art as a program and erase cycle, or "P/E cycle"), causing electrons to be moved back and forth across the floating gate's oxide region. This movement of the electrons through the

material unavoidably creates stress on the oxide layer, and eventually causes the oxide to break down. When the oxide breaks down, the cell short circuits, and becomes unusable. Ex. 1008 at 40. This failure mechanism is known as "oxide breakdown" and is illustrated below. Ex. 1003 at ¶ 37.



Another failure mechanism that affects floating gate type flash memories is known as "electron trapup," illustrated below. Ex. 1003 at ¶ 38.



Over repeated P/E cycles, electrons migrating through the oxide layer can also become trapped. These trapped charges affect the mobility of electrons into and out of the floating gate, leading to longer and longer program/erase times. Eventually, the cell becomes unusable for any practical purpose. Ex. 1008 at 41. By 2002, both oxide breakdown and electron trapup were well known to affect all floating gate type memory cells and were thus extensively studied. Ex. 1037, *Nonvolatile Semiconductor Memory Technology*, Brown et al., at 69. *See also id*. at 130-144.

Since flash memory cells degrade as a result of normal use, it was customary practice and well known to equalize as much as possible the programming and erasing activity across all memory cells in the entire flash array so that certain

memory cells do not fail prematurely from excessive P/E cycles. Ex. 1003 at ¶ 44. For example, memory cells that store program applications may rarely be erased, while memory cells that store user data files may be erased frequently. Therefore, without some form of equalization scheme, those cells with user data will already fail, even while the cells with the application program have only experienced minimal P/E cycles. *Id.* As can be readily appreciated, once failure occurs, the problem will accelerate, because the remaining memory cells that are used for storing user data will now experience even higher rate of P/E cycles. This concept of usage equalization is referred to in the art as "wear leveling," and was already well known and commonplace by 2002. Ex. 1008 at 263-265. Ex. 1003 at ¶¶ 44-46.

Common and well-known ways to implement wear leveling in flash memory generally involved copying data from one block (the "source" block) to another block (the "destination" block) to balance the amount of P/E cycles incurred by the source block and the destination block. Ex. 1003 at ¶¶ 45-46. In the example given above, a good source block candidate would be the block of cells with the application data (having undergone minimal erases), and the corresponding destination block may be one of the user data blocks that is closest to failure. This "exchange" allows the low usage rate block to be placed back into "circulation"

while allowing the high usage rate block to store data that will not require erasing by the host, and thus experience less P/E cycles going forward.

Another strategy is to more intelligently assign appropriate blocks for the incoming host data to be stored. For example, data that is likely to be erased again should be programmed into blocks that have less P/E cycles. Therefore, it was widely known to preserve and keep track of *e.g.* erase count information and other relevant statistics for memory blocks in erasable flash memory to facilitate the use of such algorithms. Ex. 1003 at ¶ 46. By 2002, many wear leveling algorithms for optimizing the selection of appropriate source and destination blocks were already known or in use. Ex. 1003 at ¶¶ 45-46; *see also* Ex. 1017 and 1018, and references cited therein.

3. Non-In-Place Update

Given the unavoidable memory cell breakdown mechanism in flash memory, it is known that excessive erase operations would wear out the device prematurely and should therefore be avoided. However, recall that real-world flash memory devices require an erase-before-write procedure, as discussed above. *See also* Ex. 1003 at ¶ 24. As one can imagine, a requirement to erase an entire block simply to overwrite a few bits of data is not only highly inefficient, but would also lead to much higher numbers of erase operations. Ex. 1003 at ¶ 25. Such constraints in real-world applications of flash memory led to the development and

implementation of so-called "non-in-place update" schemes for flash memory devices. *Id.* at ¶ 26. Under such a scheme, when the host system needs to overwrite existing data, the new data (which is tracked by the host using the same logical address) is actually stored in a new physical memory location, and the data in the old location need not be (and generally is not) immediately erased. Ex. 1008 at 255-265. Erases can occur at a later time when a greater percentage of the data within a block has been superseded and out-of-date (termed "dirty" or "invalid" in the art). Such erase blocks can be "cleaned" or "recovered" at that time by erasing the entire block. In the clean-up operation, valid data that still remains in a block must be moved to a different block (to preserve the stored data), and thereafter the entire block can be erased. By waiting until most of the block only holds invalid data, the erase operation can be delayed and minimized, compared to what would be required in an update-in-place scheme.

The advantages of such non-in-place update schemes for flash memory were well known and understood. As discussed, because it can postpone an erase cycle, excessive erase cycles which were known to wear down the memory device can be avoided. Additionally, such a scheme allowed greater flexibility as to the timing of the erase cycles, because overwritten data need not be immediately erased. As discussed above, the erase operation is comparatively the most time-consuming operation by far. Ex. 1008 at 32, 170. Such a scheme would allow the time-

consuming erase operation to occur in the background at a suitable time, *i.e.* when it would not interfere with the host system requiring access to the memory system. Ex. 1003 at ¶ 58.

The tradeoff in implementing this non-in-place update scheme is that there must be additional "bookkeeping" involved to keep track of where the "correct version" of the data is stored, because specifying a logical address alone is no longer sufficient for locating the corresponding data. This adds additional overhead and complexity to the memory system. However, in the simplest form, this can be a lookup table that translates the mapping between the host supplied logical address to the corresponding physical address where the data is actually stored. Ex. 1003 at ¶ 29. When a host data overwrite occurs, the flash interface updates the logical-to-physical address mapping in the lookup table. As another example, when valid data is moved during a cleanup operation, the mapping information must likewise be updated. When the host subsequently requests to read back the modified data, the flash interface consults the lookup table and is thereafter able to retrieve the up-to-date version of the data in the correct physical address. *Id.* This interface function was important and prevalent enough in flash memory devices that around 1994, an industry group known as the Personal Computer Memory Card International Association or PCMCIA, consisting of

several hundred companies, standardized one such interface that became known as the "Flash Translation Layer" or "FTL." Ex. 1003 at ¶ 69.

Exhibit 1010 is the Media Storage Formats Specification (Volume 7) from the PC Card Standard Release 7.0 (1999). It was a widely used and adopted standard, and ordinary artisans at the time would have been familiar with the concepts described in the FTL portion of the standard. Ex. 1003 at ¶ 73.

C. Overview of the '729 patent

The '729 patent, titled "Automated Wear Leveling in Non-Volatile Storage Systems," was filed on October 14, 2003, claims priority to a provisional application filed on October 28, 2002, and issued October 10, 2006 to Carlos J. Gonzalez and Kevin M. Conley. Ex. 1001.

The '729 patent is directed to methods and processes for performing automated wear leveling operations on a flash memory storage system. *See* Ex. 1001 at 2:66-3:1. The patent describes several aspects, each with various embodiments. *Id.* at 3:1-4:22. The embodiments relating to claims that are not being challenged (*e.g.* those requiring "zones") will not be discussed here. The relevant embodiment of the challenged claims 15-17 involves a wear leveling exchange utilizing a collection of temporarily unused but physically erased blocks that the '729 patent describes as an "erased block pool" *Id.* at 9:14-15.

The basic operation of a wear leveling scheme based on an erased block pool is shown in Figure 7, reproduced below:

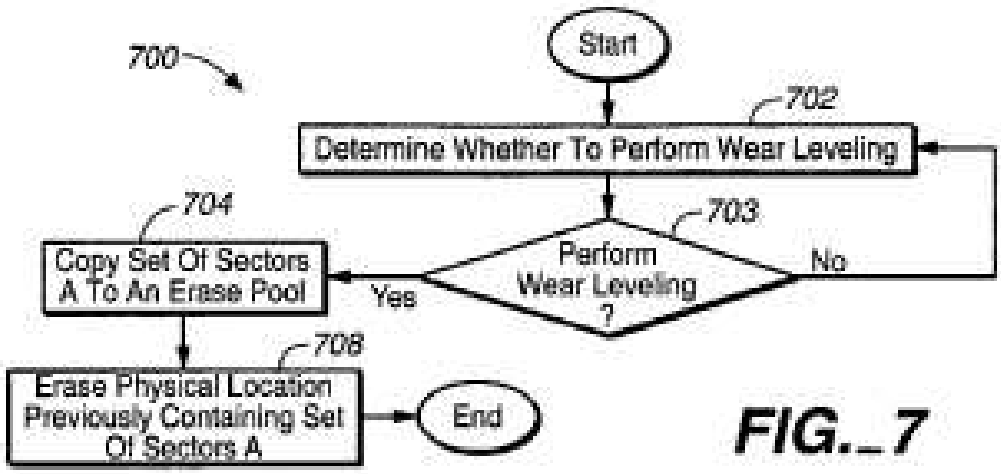


FIG._7

See also id. at 14:30-15:2.

D. Prosecution History of the ’729 Patent

During prosecution, claims 15-17 (and others) were rejected as anticipated by U.S. Patent No. 5,963,474 to Uno (“the Uno reference”). Ex. 1002-Pages 1136-1137 (Nov. 19, 2005 Office Action at 6-7). Specifically, the Examiner found that the Uno reference disclosed the limitations of claim 15 directed to identifying at least one physical block for a wear leveling exchange by cycling through block addresses in a predefined order and exchanging the identified block(s) with a corresponding number of blocks from the erased block pool. *Id.* The Examiner also found the additional limitation recited in claim 16 of copying data from the identified blocks to the erased block pool and the additional limitation recited in

claim 17 of erasing the identified block and placing it in the erase pool in Uno's disclosure. *Id.*

In its response, Applicants did not amend the claims, but instead challenged the sufficiency of the Uno reference's disclosure, arguing that it is not clear that it discloses an erased block pool. Ex. 1002-Pages 1166-1167 (Mar. 20, 2006 Remarks at 11-12). Rather, Applicants argued, the Uno reference only teaches an "alternate block area," which could not be considered an erased block pool, noting that it "appears to hold data prior to the operation described." Ex. 1002-Page 1166 (Mar. 20, 2006 Remarks at 11). Applicants also argued that "[Uno's] block area with physical number 2 would not appear to be from an erased block pool because it is described as a block 'in which the information with the logical number 2 has been stored until then,' column 1, lines 53-54 [of Uno]. Thus, it is not clear how the cited portion of Uno could support an anticipation rejection." Ex. 1002-Page 1168 (Mar. 20, 2006 Remarks at 13).

Following the Applicants' remarks on these and other claims, a Notice of Allowance issued on June 2, 2006. Ex. 1002-Pages 1189-1195 (June 2, 2006 Notice of Allowance). Petitioner notes that in the Examiner's Statement of Reasons for Allowance, while it expressly identifies limitations recited in the unchallenged claims, it is silent as to the limitations recited in challenged claim 15. Ex. 1002-Pages 1193-1194 (June 2, 2006 Notice of Allowance at 2-3).

E. Level of Ordinary Skill in the Art

A person of ordinary skill in the art ("POSITA") is a hypothetical person who is presumed to have known the relevant art at the time of the alleged invention. *Custom Accessories, Inc. v. Jeffrey-Allan Indus., Inc.*, 807 F.2d 955, 962 (Fed. Cir. 1986). Petitioner submits that a person of ordinary skill in the art at the time of the '729 patent would have a Bachelor of Science degree in electrical engineering, computer science, computer engineering, or related field, and at least two years of experience working in the field of semiconductor memory design, or equivalent. Ex. 1003 at ¶ 81. Such a person would have been capable of understanding the '729 patent and applying the prior art references as explained in this Petition.

F. Claim Construction under 37 C.F.R. § 42.104(b)(3)

Pursuant to 37 C.F.R. § 42.100(b), for the purposes of this review, the claim language is construed such that it is "given its broadest reasonable construction in light of the specification of the patent in which it appears." For terms not specifically listed and construed below, Petitioner interprets them for purposes of this review in accordance with their plain and ordinary meaning under the required broadest reasonable interpretation consistent with the specification and the prosecution history of the '729 patent in view of a POSITA's knowledge. Because the standard for claim construction at the Patent Office is different than that used in

litigation, *see In re Am. Acad. of Sci. Tech Ctr.*, 367 F.3d 1359, 1364, 1369 (Fed. Cir. 2004); MPEP § 2111, Petitioner expressly reserves the right in litigation to argue and offer different constructions for any term in the '729 patent, as appropriate to that proceeding. Petitioner further notes that in the Related Litigation, Patent Owner takes the position that none of the claim terms in the challenged claims of the '729 patent require construction and each should be given its plain and ordinary meaning. Moreover, Patent Owner takes the position that the preamble of claim 15 is not limiting. Nevertheless, Petitioner proposes the following claim constructions relevant to this proceeding. Petitioner concedes, however, that Patent Owner's positions on claim construction may prove to be broader than Petitioner's construction. Therefore, to the extent the Commission deems any of Patent Owner's positions to be broader, yet still reasonable, then Patent Owner's views should be adopted in this proceeding under the "broadest reasonable interpretation" standard.

1. "the plurality of blocks maintained as an erased block pool" (claim 15)

As illustrated in the background section, the term "erased block" as used in the '729 specification is well understood in the art. *See also* Ex. 1003 at ¶ 86. Within the context of memory devices, a "pool" of memory is understood and recognized to mean a collection of memory that is currently unused or unallocated,

but otherwise available for use. *See* Ex.1003 at ¶ 87. Therefore, Petitioner proposes a construction of "the plurality of blocks maintained as an erased block pool" to be " a plurality of blocks kept in existence continually as a collection of temporarily unused but physically erased blocks." This interpretation is consistent with the usage of this term in the specification. Ex. 1001 at 9:14-15.

2. "identifying at least one of the plurality of physical blocks at a time" (claim 15)

This limitation recites identifying at least one of the plurality of physical blocks at a time. Petitioner submits that the words "at a time" is not superfluous, and further restricts the scope of the claim. The '729 patent discloses a wear leveling operation that is repeated for a plurality of physical blocks, but in every instance, each physical block is identified individually, one at a time. *See* Ex. 1001 at 21:37-55. In other words, the evaluation and ultimate selection of a given block for the wear level exchange is performed one block at a time. *See* Ex.1003 at ¶ 88. Accordingly, under the broadest reasonable construction, this term means "identifying, one at a time, at least one of a plurality of physical blocks."

G. Challenge #1: Wells Anticipates Claims 15-17.

1. Overview of Wells

U.S. Patent No. 5,341,339, titled "Method for Wear Leveling in a Flash EEPROM Memory," was filed on November 1, 1993 and issued to Steven E.

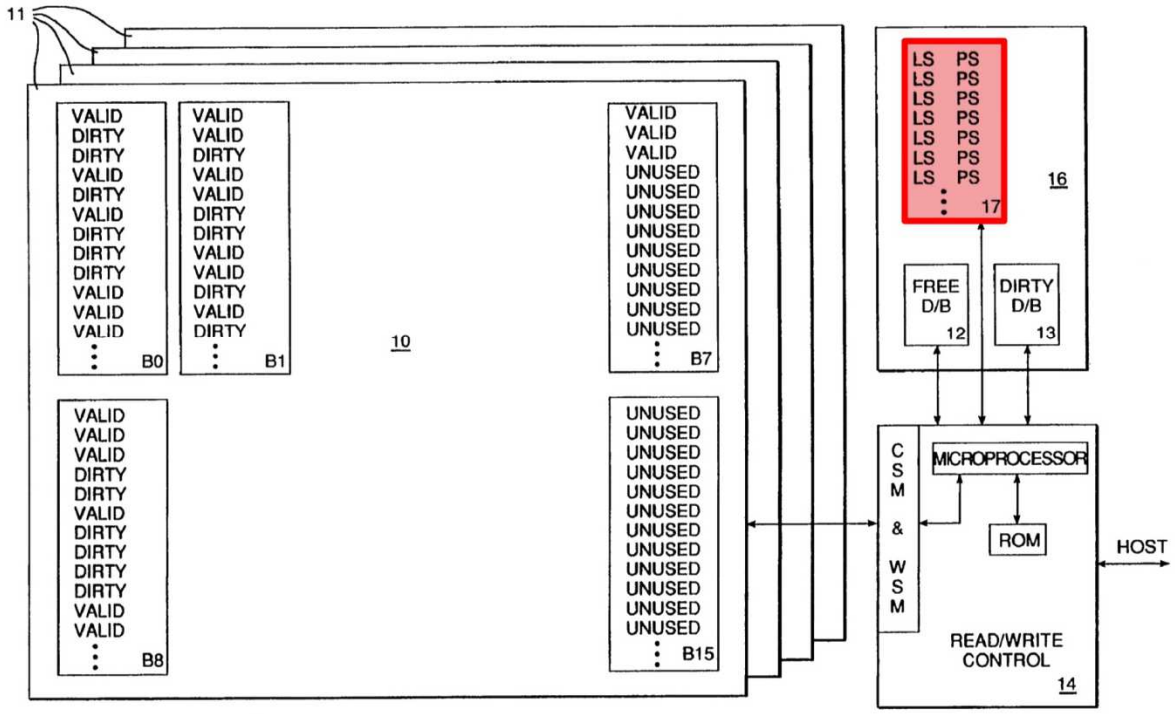
Wells, and assignee Intel Corporation on August 23, 1994. Wells is prior art to the '729 patent under at least pre-AIA 35 U.S.C. § 102(a), (b), and (e). *See* Ex. 1004.

Wells, which issued over nine years before the '729 patent was filed, describes a wear leveling technique that is substantially identical to the one described and claimed in the '729 patent. Specifically, Wells teaches a wear leveling method for a flash EEPROM memory array composed of a plurality of blocks. *Id.* at 4:35-45.

Because the minimum unit of flash EEPROM memory cells that must be erased together is a relatively large block, when pre-existing data in the memory is modified, Wells acknowledges that it is not feasible to attempt to erase all of the data in a block, and then replace the modified information along with the unmodified information that had been erased. Ex. 1004 at 2:65-3:2.

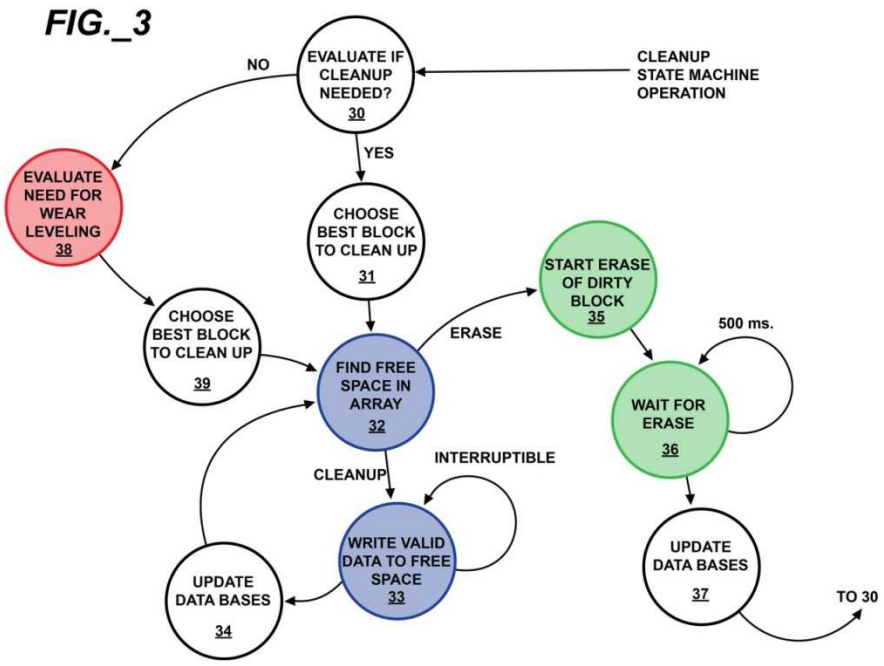
Wells discloses that to overcome this problem, it was known to use a scheme where data that needs to be replaced is not immediately "erased" from the physical flash memory. By contrast, the new data is written to an "empty" sector located on an erased block (*i.e.* a different physical location), and the old sector (with the original data still stored therein) is marked invalid, or "dirty." *Id.* at 3:3-22; 6:28-54. To accommodate this arrangement, "the sector number which is used to indicate where data is stored is a logical sector number rather than a physical sector number." *Id.* at 6:58-61; *see also* lookup table 17 of Figure 1:

FIG_1



Wells further discloses that under such an overwrite scheme, memory blocks would eventually still need to be erased in order to free up storage space to accept new data. *Id.* at 7:1-18. Wells refers to this process as "cleaning up a block," whereby any remaining valid data is moved to a fresh block, and thereafter the old block is erased. Notably, Wells teaches that this cleanup operation is recognized as an operation that achieves a substantial amount of wear leveling. *Id.* at 12:15-18. Wells further discloses a second "cleanup operation" with a different selection criterion that places an even greater emphasis on leveling wear in the erase blocks. *Id.* at 12:18-59.

Figure 3 in Wells is a flow chart illustrating the disclosed cleanup operations, reproduced below with color-coded annotations that match the annotation for the corresponding disclosure of the flowchart in Figure 7 of the ’729 patent.



Compare with Fig. 7 of the ’729 patent, annotated:

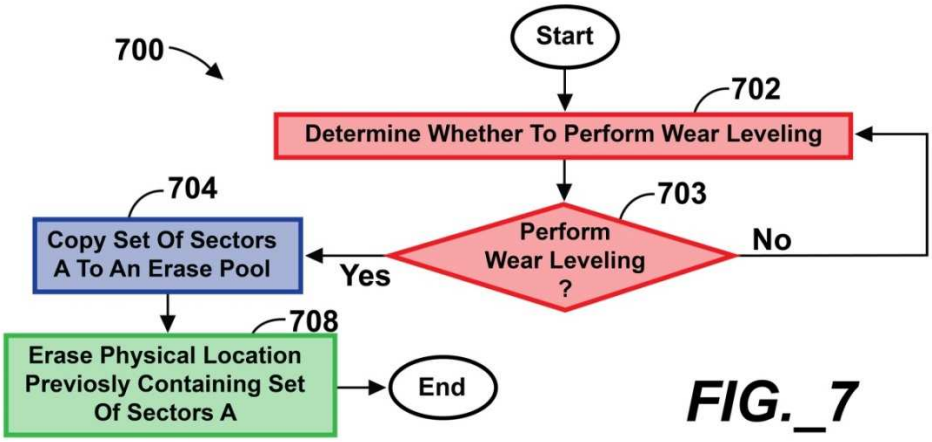


FIG._7

The wear leveling operations of both the ’729 patent and Wells are identical in relevant respects, and both involve

(i) a determination of whether to wear level, in red

(ii) finding an erased block with free space that can accommodate a wear level exchange (*i.e.* copying data from the source block to the erased block), in blue

(iii) erasing the source block once the exchange is complete, and returning the newly erased block back into circulation for subsequent re-use by the memory system, in green. *See* Ex. 1003 at ¶¶ 104-107.

2. Wells anticipates independent claim 15

a. Preamble: A method of operating a system of erasable and re-programmable non-volatile memory cells organized into a plurality of physical blocks of a minimum number of memory cells that are simultaneously erasable and wherein incoming data are programmed into those of the plurality of physical blocks

maintained as an erased block pool, comprising:

In the Related Litigation, Patent Owner takes the position that the preamble is not limiting. However, to the extent the Commission determines that the preamble is limiting, Wells discloses it.

For example, one of Wells' objectives was to "provide a method for extending significantly the life of a flash memory array" by providing "a method for equalizing the switching of different portions of a flash memory array." Ex. 1004 at 4:37-43. "Each logical block of flash memory is separately erasable from all other such blocks." *Id.* at 2:56-58.

In addition, Wells expressly teaches a method of programming incoming data into a pool of erased blocks:

> When a **host begins writing data to be stored** in the array (such as an application program) to some block of the array **which has been completely erased**, the data to be stored is written sequentially, sector by sector, to that block until that block has been filled with data. Then **writing proceeds to the next sequential block having free space**.

Id. 6:18-24, emphases added;

> Once the valid information is written to another block and the new addresses are recorded in the lookup table 17, the **block from which the information was read is erased**. **It is then placed back in operation as an entirely clean block**. In order to allow this cleanup operation to occur, **some number of blocks must be kept in reserve**

> **to be used when cleanup is necessary**. In an arrangement in which the present invention is utilized, **fourteen blocks of the 240 blocks available are used to provide sufficient space for both continuing write operations and for cleanup operations**.

Id. at 7:11-25, emphases added. *See also id.* at 7:54-61 ("The unusual arrangement for rewriting data used by a flash memory array requires that the memory allocation system continually make new or newly-erased memory available for data to be written and rewritten").

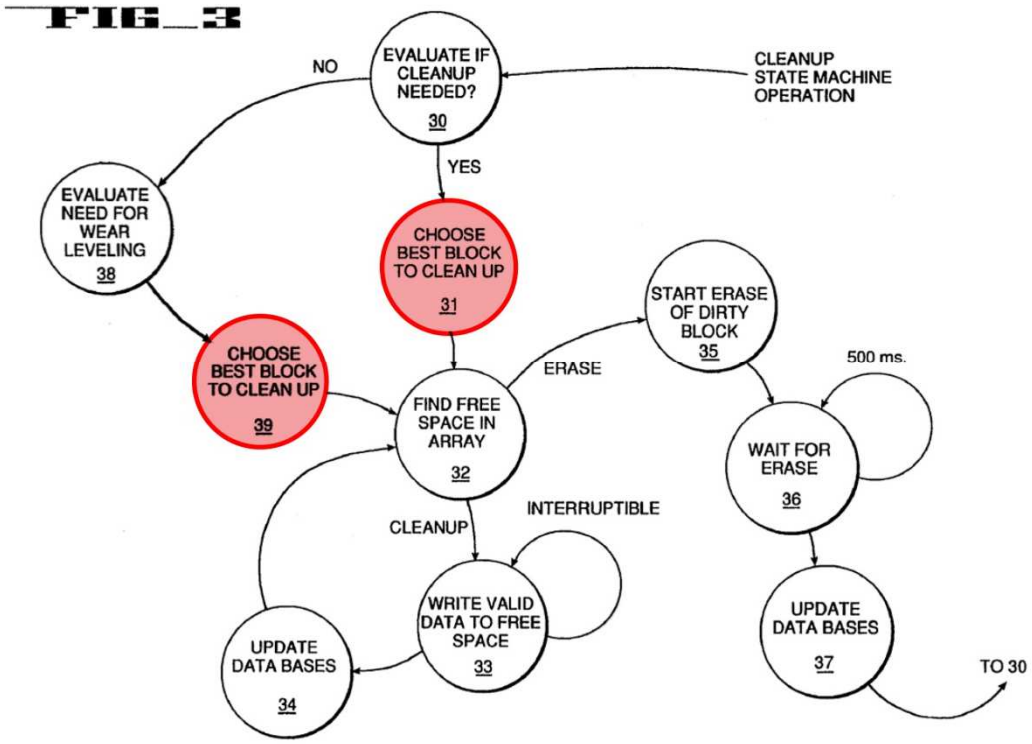
It is unsurprising that Wells teaches to maintain a plurality of blocks as an erased block pool. As discussed above, flash memory cells can only be programmed into an already erased block, and once programmed, the entire block must be erased before it can be re-programmed. Moreover, erase operations in flash memories were known to take many orders of magnitude more time than reading or programming operations. Ex. 1008 at 32, 170. Ex. 1003 at ¶ 57. Thus, it was already common practice to perform erase operations in the background whenever the host is not accessing the memory system. This would provide a group of erased blocks to be held "in reserve" and readily available whenever a programming need were to arise. *Id.* at ¶¶ 57, 87. Consider the alternative to keeping such a pool of erased blocks – in one exemplary commercial product, an erase operation can take at least 0.3s, meaning absent a pool of already erased

blocks, the system would need to wait until this (time-consuming) erase operation to complete before it could program in data provided by the host for storage. This would result in a severe bottleneck in the data throughput between the host system and the flash memory device and result in an impractical system that may incur data loss. *Id.* at ¶ 57. *See also id.* at Table 1, Claim 15.[pre].

b. **Claim 15a:** identifying at least one of the plurality of physical blocks at a time other than those in the erased block pool for a wear leveling exchange by cycling through addresses of the plurality of physical blocks in a predefined order, and

Wells discloses identifying at least one physical block at a time other than those in the erased block pool for a wear leveling exchange. Wells illustrates this, for example, in step 31(or alternatively, in step 39) of Figure 3, annotated below:

FIG_3



In one scenario, the flash memory cleanup program "moves to a step 31 in which the block **best suited to cleanup is selected**" based in large part on the block with the most dirty (*i.e.* invalid) sectors in the array, but also factors in the number of erase cycles. *Id.* at 10:50-63. The weighting between the two criteria in this process is 80% for "dirty" and 20% for "cycling." *Id.* at 11:1-9. Wells describes this process as performing "some substantial portion of . . . wear leveling" in this step. *Id*. at 12:15-18. Wells discloses that this evaluation is performed one block at a time for each block in the database. *See* Ex. 1003 at Table 1, claim 15.[a].

Wells also describes a second wear leveling process in which the flash memory cleanup program "moves to a step 39 at which an evaluation is made of

the best block to clean up based on wear leveling criteria." *Id.* at 12:33-35. In this process, each block is evaluated using the same factors (*i.e.* number of dirty sectors and the number cycles), but more heavily weighted based on the smallest number of cycles, and the block with the highest value is chosen to be cleaned up. *Id.* at 12:35-49. This evaluation is also performed one block at a time for each block in the database. *See* Ex. 1003 at Table 1, claim 15.[a]. Accordingly, Wells clearly discloses identifying at least one of physical block at a time.

Moreover, in either case, the identifying step involves stepping through and assessing **each and every** block in the database for the given criteria in order to choose the "best" block. With the best block chosen in this manner, a wear level exchange (to be discussed below) is performed, and the cycle repeats itself thereafter as required. *See id.* at 12:6-10, 12:55-59. Accordingly, Wells discloses the claimed step of cycling through addresses of the plurality of physical blocks in a predefined order as the claim requires. *See also* Ex.1003 at Table 1, Claim 15.[a].

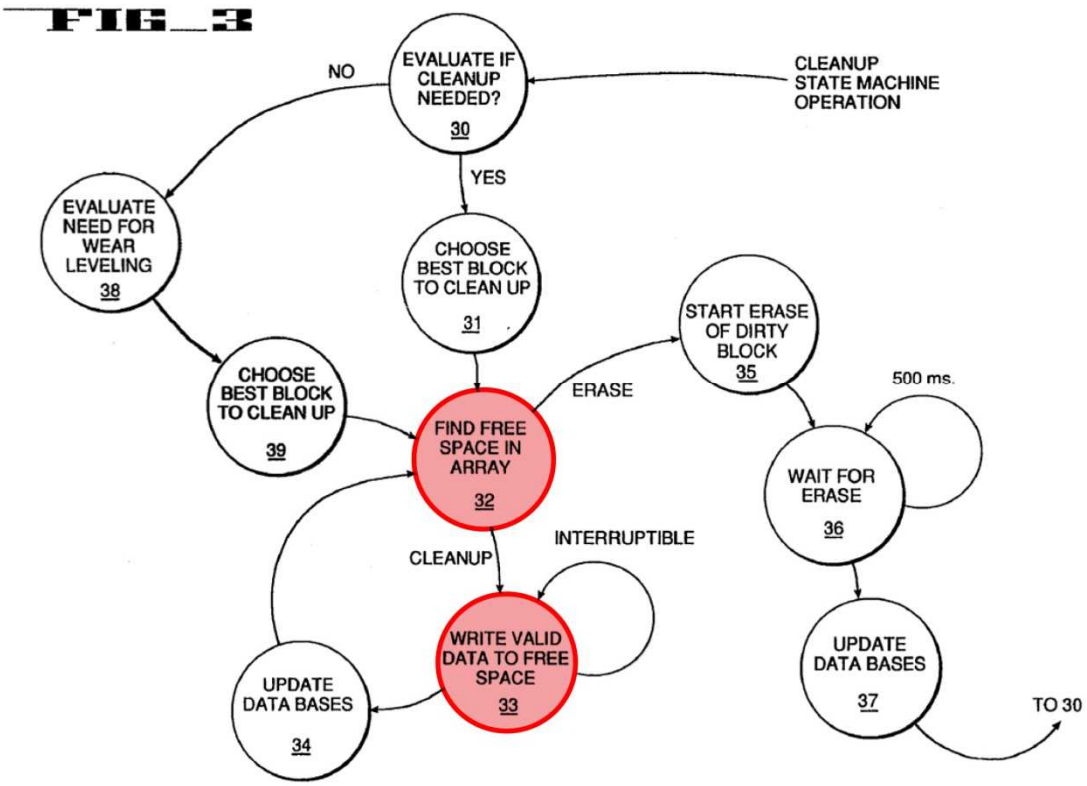
c. **Claim 15b:** exchanging the identified at least one of the plurality of physical blocks with a corresponding number of at least one of the plurality of physical blocks within the erased block pool.

Wells discloses exchanging at least one of the physical blocks with a corresponding number of physical blocks within the erased block pool. Continuing

with the flow chart of Figure 3, Wells teaches that at steps 32 and 33, the program will locate free sector space in other blocks, and thereafter write the valid data from the block being cleaned up (as identified in wear leveling steps 31 and 39) to the available space:

> Once the appropriate block to clean up has been chosen, the process moves to a step 32 at which available free sector space in other blocks is located. Once the space to store a valid sector has been located, the process moves to step 33 to **write the valid data from the sector of the block being cleaned up to the available space**.

Ex. 1004 at 11:20-26, emphasis added; *see also id.* at 7:18-25 regarding the erase block pool: ("In order to allow this cleanup operation to occur, some number of blocks must be kept in reserve to be used when cleanup is necessary. In an arrangement in which the present invention is utilized, fourteen blocks of the 240 blocks available are used to provide sufficient space for both continuing write operations and for cleanup operations"); *see also id.* at 12:55-58 ("Once the appropriate block has been chosen to implement the wear leveling cleanup operation, the program moves to the step 32 at which the cleanup begins.")



See also Ex. 1003 at Table 1, Claim 15.[b].

3. Wells anticipates dependent claim 16.

a. Claim 16a: The method of claim 15, wherein exchanging the identified blocks includes copying data from the identified at least one of the plurality of physical blocks into said corresponding number of at least one of the physical blocks within the erased block pool, and

Wells anticipates claim 15 from which this claim depends. *See* discussion in Section IV.G.2

As shown above with respect to claim 15b in Section IV.G.2.c, Wells discloses exchanging the identified blocks by copying data from the block to be

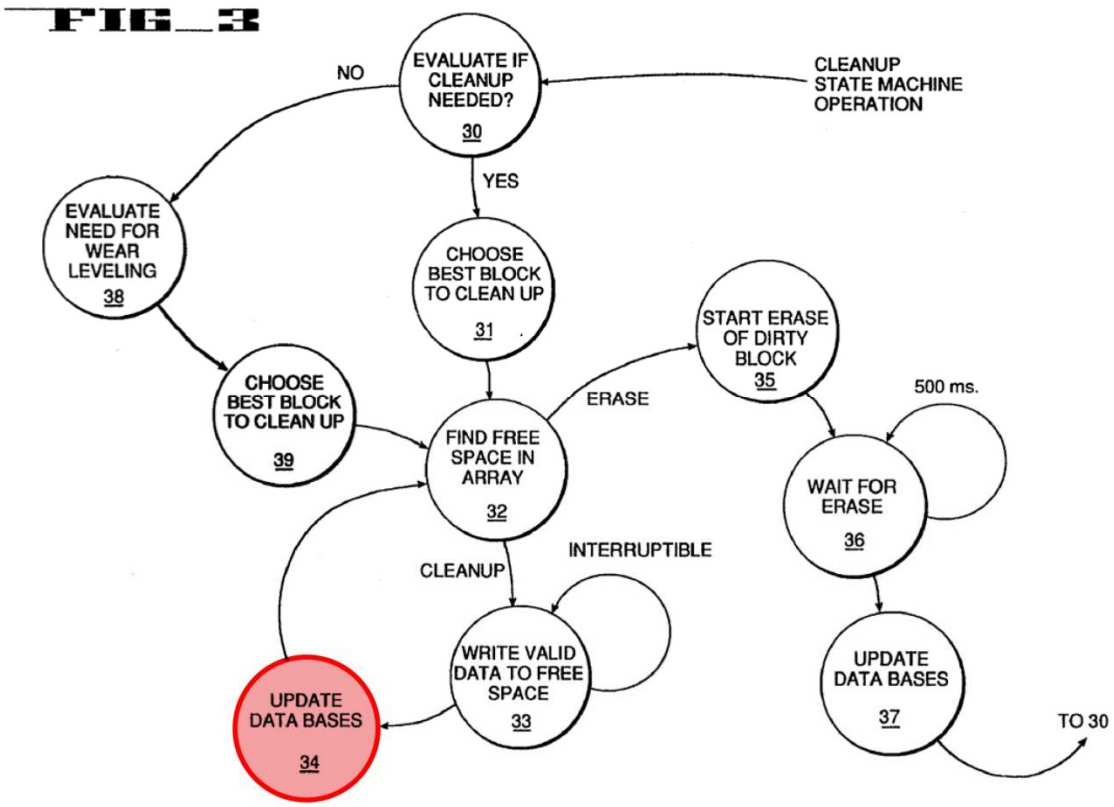
cleaned into a corresponding block within the erased block pool. Ex. 1004 at 7:11-25; 11:20-26; 11:55-62; FIG. 3. *See also* Ex. 1003 at Table 1, Claim 16.[a].

b. **Claim 16b:** changing mapping of at least one logical block address from said at least one of the plurality of physical blocks to said corresponding number of at least one of the physical blocks within the erased block pool.

Wells discloses changing the mapping of a logical block address from a physical block to a corresponding physical block within the erased block pool.

> "Once the writing of valid data in a sector on the block being cleaned up to a sector on another block has been accomplished, the program moves to a step 34 in which the various data bases kept for the array are updated. This includes **updating the sector translation tables** (the headers) [and] **updating with the new physical address the lookup table 17** in the SRAM which is associated with the flash memory and allows the logical sectors to be placed anywhere in memory."

Ex. 1004 at 11:42-52; *see also* Fig. 3.



See also Ex. 1003 at Table 1, claim 16.[b].

4. Wells anticipates dependent claim 17

a. Claim 17: The method of claim 16, additionally comprising, after copying the data, of erasing the identified at least one of the plurality of physical blocks and placing the erased at least one block into the erase pool.

Wells discloses erasing the identified physical block and placing the erased block into the erased block pool after the copying step of claim 16. As part of the cleanup operation, after the valid data from the cleanup block has been copied, "the block from which the information was read is erased . . . [and] is then **placed back in operation as an entirely clean block**." Ex. 1004 at 7:11-25, emphasis added.

Wells further discloses that "[i]n order to allow this cleanup operation to occur, some number of blocks **must be kept in reserve** to be used when cleanup is necessary" to ensure enough space is available for both write operations and cleanup operations. *Id.* at 7:11-25 (emphasis added). *See also id.* at 11:67-12:5, disclosing the newly erased block is now tracked as part of the free database. *See also* Ex. 1003 at Table 1, claim 17. A POSITA would understand and appreciate that if the erased block is not placed into the erased block pool, then the pool would soon run out of empty blocks, and there would be no way to accommodate new data. Accordingly, a POSITA would understand these disclosures to mean that the freshly erased block would become part of the blocks kept in reserve and available for future programming, meaning it is placed into the pool of erased blocks. *See also* Ex. 1003 at Table 1, claim 17.

H. Challenge #2: Claims 15-17 are obvious over the Linux pcmcia-cs package version 3.1.21 by David Hinds ("the Linux publication") in view of the knowledge of a person of ordinary skill in the art ("POSITA") as evidenced by the Media Storage Formats Specification (Volume 7) from the PC Card Standard Release 7.0 (1999), ("PC Card Standard").

1. Overview of the Linux publication

The Linux publication was authored by Dr. David Hinds, packaged into a single .tar file ("pcmcia-cs-3.1.21.tar") and made available to the public in its entirety on October 3, 2000. Ex. 1025 at ¶ 10. Therefore, the Linux Publication

qualifies as prior art to the '729 patent under at least pre-AIA 35 U.S.C. §§ 102 (a) and (b). For the purpose of this *inter partes* review, Petitioner has submitted only selected excerpts from the .tar container file as separate Exhibits (Ex. 1026-1036), although a true and correct copy of the full .tar file can still be accessed publicly at the following URL: http://sourceforge.net/projects/pcmcia-cs/files/pcmcia-cs/3.1.21/. Ex. 1025 at ¶ 10. However, Petitioner submits that collectively, Exhibits 1026-1036 constitute excerpts from a single piece of prior art publication.

A few of these exhibits are discussed specifically below. In particular, Exhibits 1026 and 1031 are true and correct copies of the files SUPPORTED.CARDS and ftl_cs.c, respectively, within the pcmcia-cs-3.1.21.tar file. Ex. 1025 at ¶ 13.

Dr. Hinds began working on and publishing earlier versions of this publication in as early as 1995. His goal was, in part, to create card services driver software for the Linux operating system to enable products based on what is known as the "PCMCIA Standard." *See* Ex. 1025 at ¶ 4.

The PCMCIA Standard was developed by the Personal Computer Memory Card International Association (PCMCIA) group. Ex. 1003 at ¶ 28. The group was formed to standardize specifications for peripheral cards which became known as PCMCIA Cards (also commonly referred to as "PC Cards") and allowed users to connect removable flash memory cards, as well as other peripheral cards such as

network cards and modems, to their personal computers. *Id.* at ¶ 68. *See, e.g.*, Ex. 1008 at 187-189. It is a comprehensive and widely adopted standard, and specifies many aspects of these peripheral cards such as physical dimensions, electrical connections and other such characteristics to ensure compatibility. *Id.* at ¶ 69.

The PCMCIA group also approved the Flash Translation Layer (FTL) specification around 1994 as part of the standard pertinent for compliant storage media such as flash memory cards, and became a widely-used interface between the host file system and the storage media. Exhibit 1010 is one volume of the PCMCIA standard that contains the FTL specification, the Media Storage Formats Specification (Volume 7) from the PC Card Standard Release 7.0 (1999), ("PC Card Standard"). It is published in 1999, and therefore qualifies as prior art to the '729 patent under at least pre-AIA 35 U.S.C. §§ 102 (a) and (b).

The PC Card Standard FTL "masks the characteristics of flash devices from higher level software layers such as file systems by emulating a traditional block device. From the perspective of such higher level layers, a block storage device is a contiguous array of blocks…These higher level software layers expect to be able to write these blocks at will, without any regard for the need to first erase the media and certainly without any need to erase an area that exceeds the size of the block being written." Ex. 1010 at 24.

The Linux publication discloses numerous PCMCIA standard compliant cards, including Ethernet cards, modems, token-ring adapters, wireless network adapters, memory cards. *See* Ex.1026. Among these supported PCMCIA standard compliant device are flash memory storage cards, such as Smartmedia flash, Compact flash cards, Intel's Series 2 Flash Memory Cards. *See id.* at lines 337-346, 465-468. *See also* Ex. 1023, a data sheet for the Intel Series 2 product. Dr. Hinds' publication is compatible with the Media Storage Formats Specification of the PC Card Standard (Ex. 1010), including the use of the data structures standardized in the flash translation layer ("FTL") format that allows a flash card to be used "as if it were an ordinary disk device." Ex. 1029 at 37. Ex. 1025 at ¶ 4.

The Linux publication describes many facets of flash memory management, including a description of an implementation of a wear leveling algorithm for these flash memory PC cards. Ex. 1025 at ¶ 7. The disclosed wear leveling algorithm is implemented as part of a "reclaim" operation and is described within ftl_cs.c of the Linux Publication. (Ex. 1031).

Ex. 1031 (ftl_cs.c) is a source code file written in the C programming language, and contains plain English comments, in addition to human-readable code for implementing the Linux card services driver. Ex. 1003 at ¶ 115. A person of ordinary skill in the art, reading the Linux publication, would be able to appreciate the teachings of not only the comments written in plain English, but also

the lines of C code themselves, and glean relevant disclosures and teachings regarding implementation of a wear leveling algorithm for flash memory devices. *Id.* Further, a person of ordinary skill in the art would also readily recognize the relationship between the Linux Publication and the PC Card Standard, and would possess at least some working familiarity with the PC Card Standard. *Id.* at ¶ 73, 115.

2. The Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders independent claim 15 obvious.

a. Claim 15 preamble: A method of operating a system of erasable and re-programmable non-volatile memory cells organized into a plurality of physical blocks of a minimum number of memory cells that are simultaneously erasable and wherein incoming data are programmed into those of the plurality of physical blocks maintained as an erased block pool, comprising:

Patent Owner contends in the Related Litigation that the preamble is not limiting. To the extent the preamble is limiting, the Linux publication discloses it.

The Linux publication includes source code for PC-Card compliant driver and card services interface. Ex. 1025 at ¶¶ 3-4. The Linux publication discloses support for using flash memory cards with cells organized into erase blocks. ("The current release includes … memory card drivers that should support most SRAM cards and some flash cards. The SUPPORTED.CARDS file included with each release of Card Services lists all cards that are known to work in at least one actual

system.") Ex. 1029 at 37. *See also* Ex. 1026 (SUPPORTED.CARDS file) at line 343, disclosing the Intel Series 2 flash memory cards. A POSITA would readily recognize that these flash memory cards include a plurality of physical blocks of a minimum number of memory cells that are simultaneously erasable. Specifically, "[t]he Series 2 Flash Memory Card contains a 2 to 20 Megabyte Flash Memory array consisting of 2 to 20 28F008SA FlashFile Memory devices. Each 28F008SA **contains sixteen individually-erasable, 64 Kbyte blocks**." Ex. 1023 at 5, emphasis added.

The Linux publication also discloses a plurality of physical blocks (also referred to therein as "erase units") maintained as an erased block pool. Blocks within the pool are identified as "transfer units" in the Linux publication. For example, Ex. 1031 at lines 1279-1358, describes the ftl_write() function for writing host data to available erased free space (transfer units). A person of ordinary skill in the art would understand that incoming data are programmed into transfer units that are erased. Ex. 1003 at Table 2, claim 15.[Pre]. Further, as discussed in the background section, flash memory have an erase-before-write requirement. As will be discussed below, the transfer units are blocks that have been erased and are ready to accept programming of data. The programmed data can be sourced from a wear level exchange (within the reclaim_block() function, defined in Ex. 1031 at lines 880-966), or it can be sourced from data in response to

a ftl_write() (host write) request. By default, the find_free() function within ftl_write() that searches for free space will search within the more recently identified transfer unit. Ex. 1031 at 1310-1311; *see also id.* at 968-1041 and at 998-1000. *See also* discussion in Ex. 1003 at Table 2, claim 15.[Pre].

To the extent the Linux publication does not expressly disclose a plurality of such blocks ("transfer units") maintained as an erased block pool, it would have been obvious to POSITA reading the disclosure of the Linux publication, because it expressly discloses algorithms for selecting a "best" transfer unit. Ex. 1031 at lines 886-892, 904-919. A POSITA would understand that absent a plurality of such blocks, there would be no option or choice of selection for the transfer unit, and nullify the potential benefits of that aspect of the wear leveling algorithm. *See* Ex. 1003 at Table 2, claim 15.[Pre]. Accordingly, the disclosure of such an algorithm should clearly indicate to, and at a minimum clearly motivate a POSITA, to configure the flash memory partition with a plurality of transfer units to take advantage of the wear level selection algorithm.

Moreover, it was known that certain commercial PC Card standard compliant flash memory cards, such as the Intel Series 2, required long erase times.

COMMON AND ATTRIBUTE MEMORY, AC CHARACTERISTICS: Write Operations(1)

Symbol		Parameter	Notes	Min	Max	Unit
JEDEC	PCMCIA					
t_{AVAV}	t_{WC}	Write Cycle Time		150		ns
t_{WLWH}	t_w (WE)	Write Pulse Width		80		ns
t_{AVWL}	t_{su} (A)	Address Setup Time		20		ns
t_{AVWH}	t_{su} (A-WEH)	Address Setup Time for WE#		100		ns
t_{VPWH}	t_{vps}	V_{PP} Setup to WE# Going High		100		ns
t_{ELWH}	t_{su} (CE-WEH)	Card Enable Setup Time for WE#		100		ns
t_{DVWH}	t_{su} (D-WEH)	Data Setup Time for WE#		50		ns
t_{WHDX}	t_h (D)	Data Hold Time		20		ns
t_{WHAX}	t_{rec} (WE)	Write Recover Time		20		ns
t_{WHRL}		WE# High to RDY/BSY#			120	ns
t_{WHQV1}		Duration of Data Write Operation		6		µs
t_{WHQV2}		Duration of Block Erase Operation		0.3		sec
t_{QVVL}		V_{PP} Hold from Operation Complete	2	0		ns
t_{WHGL}	t_h (OE-WE)	Write Recovery before Read		10		ns
t_{RHWL}		Reset-PwrDwn Recovery to WE# Going Low		1		µs

Ex. 1023 at 32, showing a minimum erase time of 0.3 seconds. Because of the long erase times, it was already common practice to perform erase operations in the background whenever the host is not accessing the memory system. This would provide a group of erased blocks to be held "in reserve" and readily available whenever a programming need were to arise. *Id.* at ¶¶ 57, 87. Therefore, a POSITA would additionally have recognized the advantages of keeping a collection of such temporarily unused but physically erased blocks prior to when the host requires free space from the storage media. A POSITA would have understood that absent such a pool of blocks, the host would need to wait a minimum of 0.3 seconds just for the erase operation to complete in order to have

free space to accommodate new programming, and may potentially lead to a loss of data. *See* Ex. 1003 at ¶ 57; *see also id.* at Table 2, claim 15.[Pre].

b. **Claim 15a:** identifying at least one of the plurality of physical blocks at a time other than those in the erased block pool for a wear leveling exchange by cycling through addresses of the plurality of physical blocks in a predefined order, and

The Linux publication discloses identifying at least one of physical block at a time other than those in the erased block pool for a wear leveling exchange. The Linux publication describes two types of "erase units" (corresponding to the claimed physical blocks) that are separately tracked at a logical level within a data structure: "data units" and "transfer units." Ex. 1031 at lines 160-170, 880-894; *See also* Ex.1003 at ¶ 119. *See also* discussion in the preamble of claim 15 in Section IV.H.2.a above. The Linux publication further describes within the reclaim_block() function a wear level exchange operation that identifies a data unit to be copied into a selected transfer unit.

> "reclaim_block() picks a full erase unit and a transfer unit and then calls copy_erase_unit() to copy one to the other. Then, it schedules an erase on the expired block.
>
> What's a good way to decide which transfer unit and which erase unit to use? Beats me. My way is to always pick the transfer unit with the fewest erases, and **usually pick the data unit with the most deleted blocks. But with a small probability, pick the**

> **oldest data unit instead**. This means that we generally postpone the next reclaimation as long as possible, but shuffle static stuff around a bit for wear leveling."

Ex. 1031 at lines 882-892, emphasis added.

As discussed in Section IV.H.2.a, the erased block pool comprises the transfer units. Therefore, identifying a data unit meets the limitation of identifying one of the plurality of physical blocks at a time other than those in the erased block pool (*i.e*. group of transfer units) as required.

Specifically, the Linux publication discloses that the identification of the data unit for wear leveling would select, with a low probability, the "oldest" data unit (*i.e.* one with the lowest erase count) in order to "shuffle static stuff around a bit for wear leveling." *See* Ex. 1031 at lines 886-892; *see also* Ex. 1031 at lines 936-959. The Linux publication expressly discloses a "for loop" that would be understood by POSITA to scan through (*i.e.* cycle through) the offset address for each of the data blocks in order to find the "oldest" data unit.

```
" for (i = 0; i < part->DataUnits; i++)
        if (part->EUNInfo[i].EraseCount <= best) {
            best = part->EUNInfo[i].EraseCount;
            eun = i;
        }"
```

Ex. 1031 at lines 940-944. Additionally, the Linux publication clearly discloses in this for loop identification of only a single "best" data unit at a time,

since it loops through all the data units within the partition (*see* line 940) and only keeps one "best" candidate (with the best index stored as "eun"). Therefore, this data unit "block" is identified one at a time as required by the claim. *See also* Ex.1003 at Table 2, claim 15.[a].

c. **Claim 15b:** exchanging the identified at least one of the plurality of physical blocks with a corresponding number of at least one of the plurality of physical blocks within the erased block pool.

The Linux publication discloses exchanging at least one of the physical blocks with a corresponding number of physical blocks within the erased block pool. Within the reclaim_block() function, a data unit ("eun") and a transfer unit ("xfer") are identified, as discussed in Section IV.G.2.b. As discussed, a single data unit is identified by the algorithm. Correspondingly, a single transfer unit is likewise selected as disclosed in the Linux publication.

```
"/* Pick the least erased transfer unit */
 best = 0xffffffff; xfer = 0xffff;
 do {
      queued = 0;
      for (i = 0; i < part->header.NumTransferUnits; i++) {
         if (part->XferInfo[i].state == XFER_UNKNOWN)
              erase_xfer(dev, part, i);
         if (part->XferInfo[i].state == XFER_ERASING)
              queued = 1;
         else if (part->XferInfo[i].state == XFER_ERASED)
```

```
        prepare_xfer(part, i);
    if ((part->XferInfo[i].state == XFER_PREPARED) &&
        (part->XferInfo[i].EraseCount <= best)) {
           best = part->XferInfo[i].EraseCount;
           xfer = i;
        }"
```

Ex. 1031 at lines 904-919, emphasis added.

Once the best data unit and the best transfer unit have been identified, the reclaim_block() function then calls the copy_erase_unit() function.

"ret = copy_erase_unit(part, eun, xfer);"

Ex. 1031 at line 960. The copy_erase_unit() function exchanges the single identified data unit with exactly one transfer unit selected out of the erased block pool.

> "Copy_erase_unit() takes a full erase block and a transfer unit, copies everything to the transfer unit, then swaps the block pointers.
>
> All data blocks are copied to the corresponding blocks in the target unit, so the virtual block map does not need to be updated."

Ex. 1031 at lines 767-773. *See also* corresponding description of the function as defined in source code at Ex. 1031 at lines 765-878; *see also id.* at line 813. *See also* Ex. 1003 at Table 2, claim 15.[b].

3. The Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders dependent claim 16 obvious.

a. Claim 16a: The method of claim 15, wherein exchanging the identified blocks includes copying data from the identified at least one of the plurality of physical blocks into said corresponding number of at least one of the physical blocks within the erased block pool, and

As shown in Section IV.H.2, the Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders obvious claim 15. In addition, as shown above for claim 15b in Section IV.H.2.c, the Linux publication discloses exchanging the identified blocks by copying data from the block to be reclaimed (selected data unit) into a corresponding block within the erased block pool (selected transfer unit). *See e.g.* Ex. 1031 at lines 765-775, 776-878; 960. *See also* Ex. 1003 at Table 2, claim 16.[a].

b. Claim 16b: changing mapping of at least one logical block address from said at least one of the plurality of physical blocks to said corresponding number of at least one of the physical blocks within the erased block pool.

The Linux publication discloses changing the mapping of a logical block address from a physical block to a corresponding physical block within the erased block pool. A POSITA would readily recognize that the Linux publication is compatible with the PC Card Standard (Ex. 1010) and its implementation of the standard's FTL non-in-place update scheme allows a flash card to be used "as if it were an ordinary disk device… This [FTL] layer hides the device-specific details

of flash memory programming and make the card look like a simple block device." Ex. 1029 at 37. The PC Card Standard explains:

> "With the translation layer in place, the host system believes it is using traditionally formatted media. However, **the translation layer requires sector mapping information to be stored on the media** and **what the host feels are contiguous sectors are actually placed on the media out of sequence. To allow another system to recover the data stored on the media** requires an understanding of **how the translation layer remaps sectors** and the native storage format intended by the original file system." Ex. 1010 at 23, emphases added.

Accordingly, a POSITA reading the disclosure of the reclaim operation in the Linux publication, where user data is moved to a different physical block, would understand the disclosure to inherently require a remapping of the sector (logical address) information. *See* Ex. 1003 at Table 2, claim 16.[b]

Moreover, this remapping is also expressly disclosed in the Linux publication. Referring again to the copy_erase_unit() function called within the reclaim_block() function, the Linux publication discloses to "[u]pdate the maps and usage stats" as a result of the exchange between the data unit and the transfer unit. *See* Ex. 1031 at lines 854. Within the source code, this is disclosed as follows:

```
"i = xfer->Offset;
xfer->Offset = eun->Offset;
```

eun->Offset = i;"

Ex. 1031 at lines 858-860. This code is readily understood by a POSITA to exchange the "Offset" address parameter between the "eun" data unit and the "xfer" transfer unit, where the "i" variable is a temporary placeholder to allow the swapping of the Offset information, which in turn changes the logical to physical address mapping of the two respective blocks. Accordingly, the Linux publication discloses changing the mapping of at least one logical block address from said at least one of the plurality of physical blocks to said corresponding number of at least one of the physical blocks within the erased block pool. Additional source code citations that would be recognized by a POSITA as providing additional disclosure related to this "Offset" parameter and the mapping can be found at Ex. 1031 at lines 781-789; *id.* at lines 157 and 639-640, and the disclosure of VirtualBlockMap; *id.* at line 1351 within the build_maps() function; *id.* at lines 161, 167, 579 and 598; *see also* Ex. 1003 at Table 2, claim 16.[b].

4. The Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders dependent claim 17 obvious.

a. Claim 17: The method of claim 16, additionally comprising, after copying the data, of erasing the identified at least one of the plurality of physical blocks and placing the erased at least one block into the erase pool.

A s shown in Section IV.H.3, the Linux publication in view of the knowledge of POSITA as evidenced by the PC Card Standard renders obvious claim 16. The Linux publication further discloses erasing the identified physical block and placing the erased block into the erased block pool after the copying step of claim 16.

Within the disclosed reclaim_block() function, the copy_erase_unit() function is called in Ex. 1031 at line 960, as discussed regarding limitation 15b in Section IV.H.2.c. As part of the final steps of the reclaim_block() function, the Linux publication discloses the following immediately following the copy_erase_unit() function call:

```
"ret = copy_erase_unit(part, eun, xfer);
if (ret == CS_SUCCESS)
    erase_xfer(dev, part, xfer);
else
    printk(KERN_NOTICE "ftl_cs: copy_erase_unit failed!\n");
return ret;
```

} /* reclaim_block */”

Ex. 1031 at lines 960-966, emphasis added.

Line 962 (“erase_xfer(dev, part, xfer)”) is a function call that initiates the erase operation of the “xfer” block. Recall that “xfer” now refers to the erase unit that was (prior to the wear level exchange and the remapping) the data unit. *See* discussion in Section IV.H.3. The erase_xfer() function is further disclosed in Ex 1031 at lines 650-688. (“Erase_xfer() schedules an asynchronous erase operation for a transfer unit.” Ex. 1031 at lines 652-653.)

Further, once the transfer unit has been erased, the Linux publication discloses that it is returned to the erased block pool. For example, a POSITA would understand the Linux publication at Ex. 1031 at lines 463-499, and in particular lines 481-484 to disclose an event handler that calls a save_status() function once the erase operation is complete.

> “case CS_EVENT_ERASE_COMPLETE:
> save_status((eraseq_entry_t *)(args->info));
> wake_up(&dev->erase_pending);
> break;” Ex. 1031 at lines 481-484.

A POSITA would understand in reviewing the Linux publication at Ex. 1031 at lines 697-720 that this save_status() function will detect that the newly erased unit has a different physical offset from that of the transfer units already in the

transfer unit pool, and therefore places the newly erased unit into the transfer unit pool (*i.e*. the claimed erased block pool). *See also* Ex. 1003 at Table 2, claim 17.

Moreover, as discussed in the preamble of claim 15 in Section IV.H.2.a, the Linux publication discloses only two types of erase units, *i.e.* data units that store data, and transfer units that form part of the erased block pool. Ex. 1031 at lines 160-170, 880-894; *See also* Ex.1003 at ¶ 119. A POSITA would understand that the act of erasing the former data unit would erase all the data stored therein, and so it would naturally be updated in the accounting by the FTL with modified status as a transfer unit, joining the rest of the pool of transfer units.

V. CONCLUSION

Claims 15-17 of the ’729 patent are anticipated or rendered obvious by prior art not yet considered by the Patent Office. There is a reasonable likelihood that Petitioner will prevail as to each of the claims. Petitioner respectfully requests that the Patent Office initiate an *inter partes* review of claims 15-17, that it find those claims invalid in light of the prior art, and that it cancel those claims.

Respectfully submitted,

Dated: September 14, 2015

/s/ Brent Yamashita

Brent Yamashita
Registration No. 53,808

DLA PIPER LLP (US)
2000 University Avenue
East Palo Alto, CA 94303-2214
Telephone: 650.833.2348
Facsimile: 650.833.2001

CERTIFICATE OF SERVICE

The undersigned certifies service pursuant to 37 C.F.R. 37 C.F.R. §§ 42.6(e) and 42.105(b) on September 14, 2015 by UPS Overnight Delivery of a copy of this Petition for *Inter Partes* Review and supporting material at the following correspondence address of record for the '729 Patent:

DAVIS WRIGHT TREMAINE LLP - SANDISK CORPORATION
IP Docketing Dept.
Davis Wright Tremaine LLP
1201 Third Avenue, Suite 2200
Seattle WA 98101

Dated: September 14, 2015

/s/ Brent K. Yamashita
Brent K. Yamashita
Registration No. 53,808

DLA PIPER LLP (US)
2000 University Avenue
East Palo Alto, CA 94303-2215
Telephone: 650.833.2348
Facsimile: 650.687.1206